Міністерство освіти і науки України
Львівський національний університет ім. Івана Франка
Факультет електроніки та комп’ютерних технологій
Кафедра системного проектування

КУРСОВА РОБОТА

на тему: “Дослідження методів автентифікації користувачів у вебзастосунках”

Виконав:
студент групи ФеІм-12с
Дребот А. С.
Науковий керівник:
доцент Ненчук Т. М.

Львів - 2023р.

АНОТАЦІЯ

Дана курсова робота присвячена дослідженню різних методів автентифікації користувачів у вебзастосунках з фокусом на їх популярність, переваги, недоліки і складність реалізації та використання.

Результати дослідження дозволять краще зрозуміти сучасні методи автентифікації користувачів у вебзастосунках, а також розробити власний вебзастосунок “Media Storage”, в якому буде впроваджено один з найпопулярніших методів автентифікації користувачів, а саме “Багатофакторна автентифікація” який вимагатиме підтвердження ідентичності користувача через два різні фактори: звичайний пароль та тимчасовий код створений TOTP-алгоритмом.

ANNOTATION

This coursework is devoted to research of various methods of user authentication in web applications in view of their popularity, advantages, disadvantages and complexity of implementation and use.

The results of the research will allow a better understanding of modern user authentication methods in web applications, as well as the development of our own “Media Storage” web application, which will implement one of the most popular user authentication methods, namely “Multi-factor authentication”, which will require confirmation of the user's identity through two different factors: the usual password and the temporary code is created by the TOTP algorithm.

ЗМІСТ

ВСТУП 4
1. МЕТОДИ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ У ВЕБЗАСТОСУНКАХ 5
1.1. Класифікація 5
1.1.1. Парольна автентифікація 5
1.1.2. Біометрична автентифікація 7
1.1.3. Автентифікація на основі місцезнаходження 9
1.1.4. Автентифікація за допомогою сертифікатів 10
1.1.5. Багатофакторна автентифікація 11
1.2. Переваги та недоліки 13
1.2.1. Переваги та недоліки парольної автентифікації 13
1.2.2. Переваги та недоліки біометричної автентифікації 14
1.2.3. Переваги та недоліки автентифікації на основі місцезнаходження 15
1.2.4. Переваги та недоліки автентифікації за допомогою сертифікатів 16
1.2.5. Переваги та недоліки багатофакторної автентифікації 18
1.3. Популярність 19
1.4. Складність реалізації та використання 21
1.5. Підсумок 24
2. ОБРАНІ ТЕХНОЛОГІЇ 26
2.1. Фреймворк Apache Maven 26
2.1.1. Об'єктна модель опису проекту 26
2.2. Мова програмування Java 27
2.2.1. Особливості 28
2.3. Фреймворк Spring 29
2.3.1. Spring архітектура 30
2.4. Фреймворк Spring Boot 33
2.5. JSON Web Token 34
2.5.1. Структура 35

2.6. Мова програмування TypeScript 36
2.6.1. Переваги 36
2.7. Фреймворк Angular 37
2.7.1. Особливості 37
2.8. TOTP-алгоритм 39
3. РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ “MEDIA STORAGE” 41
3.1. Реалізація серверної частини 41
3.1.1. Структура проекту 41
3.1.2. Автентифікація 44
3.1.3. RESTful вебсервіс 50
3.1.4. Бізнес логіка 52
3.1.5. Взаємодія з базою даних 54
3.2. Реалізація клієнтської частини 59
3.2.1. Структура проекту 59
3.2.2. Компоненти 60
3.2.3. Маршрутизація 61
3.2.4. HTTP-перехоплювач 62
3.3. Опис реалізованого функціоналу 65
3.3.1. Вхід в обліковий запис 65
3.3.2. Реєстрація нового облікового запису 66
3.3.3. Зміна паролю 71
3.3.4. Галерея 75
3.3.5. Сторінка "Налаштування" 76
ВИСНОВОК 82
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 83
ДОДАТКИ 86

ВСТУП

У сучасному цифровому світі, де майже кожен аспект нашого життя стає все більш залежним від технологій, забезпечення безпеки та захисту конфіденційності стають ключовими пріоритетами. Одним з фундаментальних аспектів цифрової безпеки є методи автентифікації користувачів.

Автентифікація - це процес перевірки ідентичності користувача, що дає можливість контролювати доступ до різних ресурсів та послуг в Інтернеті. Вебзастосунки, такі як онлайн-банкінг, соціальні мережі, електронна пошта, електронна комерція, та інші, вимагають надійних методів автентифікації для захисту від несанкціонованого доступу та злочинних дій.

Мета даної курсової роботи полягає в дослідженні різних методів автентифікації, які застосовуються в сучасних комп'ютерних системах, а також зокрема у вебзастосунках, з фокусом на аналізі їх популярності, переваг, недоліків і складності реалізації та використання.

Результати дослідження дозволять нам краще зрозуміти сучасні методи автентифікації користувачів у вебзастосунках, а також розробити власний вебзастосунок “Media Storage”, в якому буде впроваджено один з найпопулярніших методів автентифікації користувачів, а саме “Багатофакторна автентифікація” який вимагатиме підтвердження ідентичності користувача через два різні фактори: звичайний пароль та тимчасовий код створений TOTP-алгоритмом.

1. МЕТОДИ АВТЕНТИФІКАЦІЇ КОРИСТУВАЧІВ У ВЕБЗАСТОСУНКАХ

1.1. Класифікація

1.1.1. Парольна автентифікація

Парольна автентифікація є одним з найпоширеніших методів перевірки ідентичності користувача в комп'ютерних системах. Цей метод полягає в тому, що користувач повинен ввести правильний пароль для підтвердження своєї ідентичності перед отриманням доступу до облікового запису, системи або ресурсів. На рис. 1.1.1.1. продемонстрований вигляд форми входу з використанням парольної автентифікації.

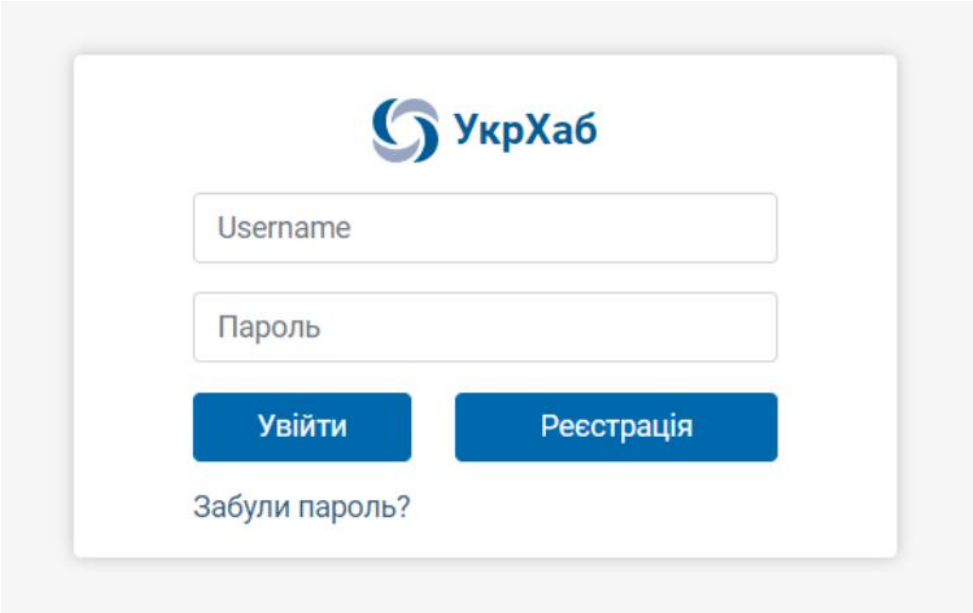

Рис. 1.1.1.1. – Вигляд форми входу з використанням парольної автентифікації.

Процес парольної автентифікації зазвичай включає наступні кроки:

1. Користувач вводить свій ідентифікатор, такий як ім'я користувача або електронну адресу.
2. Користувач вводить свій пароль, який він обирає при створенні облікового запису або який надається системою.

3. Система перевіряє, чи співпадає введений пароль зі збереженим паролем для відповідного облікового запису.
4. Якщо введений пароль збігається зі збереженим паролем, система вважає, що користувач пройшов автентифікацію і надає йому доступ до відповідних ресурсів або функцій. У протилежному випадку доступ буде відхилений.

Для вебзастосунків існує кілька стандартних протоколів для автентифікації паролем, а саме:

- Протокол **HTTP Basic**: Цей протокол використовує заголовок авторизації у форматі "Authorization: Basic <credentials>". При цьому ім'я користувача та пароль кодуються у Base64 і передаються у заголовку запиту HTTP. Варто зауважити, що протокол HTTP Basic не є найбезпечнішим, оскільки дані передаються у відкритому вигляді без шифрування.
- Протокол **HTTP Digest**: Цей протокол також використовує заголовок авторизації, але паролі передаються у вигляді хеш-суми. Запит включає хеш-суму пароля, що заснована на відомих параметрах, і сервер перевіряє, чи збігається хеш-сума. Це покращує безпеку порівняно з HTTP Basic, оскільки паролі не передаються у відкритому вигляді, але протокол все ще має певні обмеження.
- Протокол **OAuth**: OAuth - це протокол авторизації та делегування доступу, який використовується для автентифікації вебзастосунків. Хоча він більше спрямований на делегування доступу до ресурсів, OAuth також може бути використаний для автентифікації за допомогою паролів. У цьому випадку пароль використовується для отримання маркера доступу, який потім використовується для здійснення запитів до застосунка.
- Протокол **Lightweight Directory Access Protocol (LDAP)**: LDAP використовується для доступу до каталогів, таких як Active Directory та

включає можливість автентифікації за допомогою паролів. Він широко використовується в корпоративних середовищах.

1.1.2. Біометрична автентифікація

Біометрична автентифікація у вебзастосунках використовується для ідентифікації користувачів на основі їх біометричних даних, коли вони взаємодіють з веб-платформою або сервісом. Це може бути використано як альтернатива або додатковий метод автентифікації поряд з традиційними методами, такими як паролі чи пін-коди.

На рис. 1.1.2.1. продемонстрований вигляд форми входу з використанням біометричної автентифікації (сканування відбитку пальця).

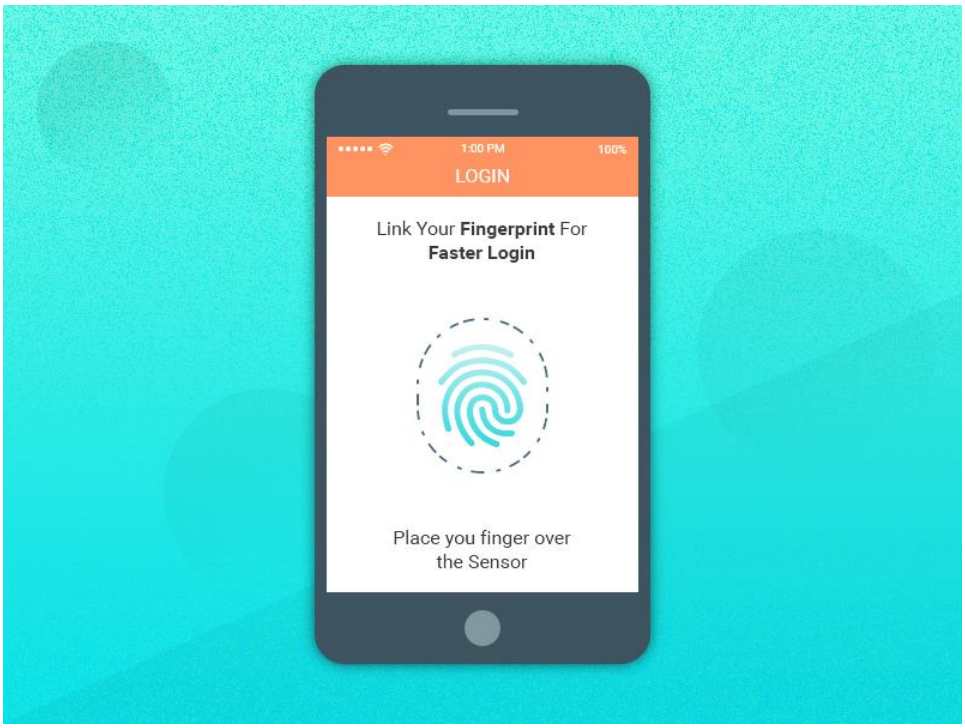

Рис. 1.1.2.1. – Вигляд форми входу з використанням біометричної автентифікації (сканування відбитку пальця).

До основних біометричних характеристик належать:

- **Відбиток пальця**: Користувачі можуть сканувати свій відбиток пальця за допомогою вбудованих датчиків на пристроях, таких як смартфони або сканери відбитків пальців, щоб ідентифікувати себе.
- **Розпізнавання обличчя**: Камери вбудовані в пристрої можуть збирати дані про обличчя користувача, які потім аналізуються для перевірки та ідентифікації особи.
- **Голосовий відбиток**: Алгоритми обробки звуку можуть аналізувати особливості голосу користувача, такі як тон, ритм і інтонація для ідентифікації.
- **Характеристики DNA**: Ідентифікація на основі генетичних даних може бути використана, хоча це зазвичай вимагає спеціалізованого обладнання і процедури для збору та аналізу зразків DNA.

Для впровадження біометричної автентифікації у веб-застосунках необхідно реалізувати наступні кроки:

1. **Збір біометричних даних**: Вебзастосунок повинен мати можливість збирати біометричні дані від користувачів. Це може включати використання камери або сканеру відбитків пальців, або можливість завантажити фото чи відео для аналізу.
2. **Обробка даних**: Зібрані біометричні дані перетворюються на цифровий формат та підлягають обробці за допомогою спеціальних алгоритмів. Це включає виявлення та виділення особливостей біометричної характеристики.
3. **Збереження та забезпечення безпеки даних**: Біометричні дані повинні бути збережені в безпечному середовищі, оскільки вони представляють особисту інформацію. Для захисту цих даних можуть використовуватися різні методи шифрування та захисту доступу.

4. **Порівняння та верифікація**: Зібрані біометричні дані порівнюються з збереженими зразками, щоб перевірити, чи відповідають вони наявним даним. Це може включати використання алгоритмів визначення подібності, шаблонів або штучного інтелекту для забезпечення точності та достовірності процесу верифікації.

Важливо враховувати приватність та захист даних користувачів при використанні біометричної автентифікації. Потрібно забезпечити відповідні правила збору, збереження та використання біометричних даних, а також регулярно оновлювати систему для запобігання можливим атакам або злому.

1.1.3. Автентифікація на основі місцезнаходження

Автентифікація на основі місцезнаходження - це метод перевірки ідентичності користувача, використовуючи його фізичне місцезнаходження. Цей метод використовує дані про геолокацію, IP-адресу та інші параметри, щоб встановити, звідки здійснюється доступ до вебзастосунку.

Процес автентифікації на основі місцезнаходження може включати наступні кроки:

- **Збір даних**: Система збирає дані про місцезнаходження користувача, такі як геолокація через GPS, IP-адресу, або інші методи, які надають загальну інформацію про місцезнаходження.
- **Порівняння з довідником**: Зібрані дані про місцезнаходження порівнюються зі заздалегідь встановленим довідником, який містить дозволені або обмежені місця доступу.
- **Прийняття рішення**: На основі порівняння система приймає рішення про допуск або відхилення доступу користувача. Якщо місцезнаходження відповідає вимогам, користувачу надається доступ до вебзастосунку.

Автентифікація на основі місцезнаходження може бути використана для різних цілей, таких як контроль доступу до важливих ресурсів або послуг, забезпечення безпеки та виявлення потенційно шкідливої або шахрайської діяльності.

1.1.4. Автентифікація за допомогою сертифікатів

Автентифікація за допомогою сертифікатів - це процес перевірки і підтвердження ідентичності користувача за допомогою цифрового сертифіката. У цьому методі користувач надає свій цифровий сертифікат, який видається надійною стороною (центром сертифікації). Сертифікат містить публічний ключ користувача, а також інші відомості, такі як ім'я, організація, термін дії тощо. Сертифікат підписується центром сертифікації за допомогою його приватного ключа, що підтверджує його валідність.

Під час процесу автентифікації система перевіряє підпис цифрового сертифіката, використовуючи публічний ключ центра сертифікації, а також перевіряє, чи дійсний сертифікат на момент перевірки. Якщо перевірка успішна, користувач або сутність вважаються автентичними і їм надається доступ до відповідних ресурсів або послуг.

Автентифікація за допомогою сертифікатів є широко використовуваним методом у різних галузях, таких як комп'ютерна безпека, електронна комерція, бездротові мережі тощо. Вона забезпечує високий рівень безпеки, оскільки використовує криптографічні методи для перевірки та захисту ідентичності.

На рис. 1.1.4.1. продемонстрований приклад створеного сертифікату.

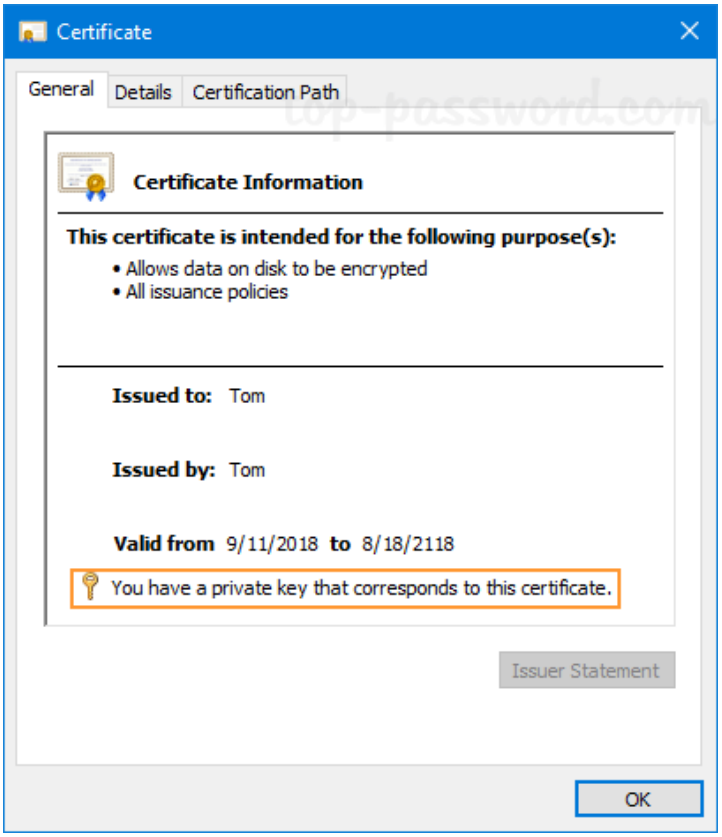

Рис. 1.1.4.1. – Приклад створеного сертифікату.

1.1.5. Багатофакторна автентифікація

Багатофакторна автентифікація - це метод захисту облікових записів, який вимагає від користувачів надати кілька різних факторів для підтвердження їх ідентичності перед отриманням доступу до системи або ресурсів. Замість одного пароля, який може бути вкрадений або вгаданий, багатофакторна автентифікація вимагає введення додаткових факторів, таких як:

- Щось, що ви знаєте: тимчасовий пароль, пін-код або секретна відповідь.
- Щось, що ви маєте: фізичний об'єкт, наприклад, смарт-карту, USB-ключ або мобільний пристрій.
- Щось, що ви є: біологічні характеристики, такі як відбиток пальця або розпізнавання обличчя.

Багатофакторна автентифікація додає ще один шар безпеки, оскільки зловмиснику потрібно мати не тільки правильний пароль, але й доступ до додаткових факторів. Це робить несанкціонований доступ до облікових записів значно складнішим для зловмисників.

Багатофакторна автентифікація використовується в багатьох сферах, включаючи комп'ютерні системи, мобільні додатки, банківські операції, електронну пошту, соціальні мережі та інші онлайн-сервіси. Вона є важливим інструментом для захисту конфіденційної інформації та запобігання несанкціонованому доступу до особистих даних.

На рис. 1.1.4.2. продемонстрований вигляд форми входу з використанням багатофакторної автентифікації.

Рис. 1.1.4.2. – Вигляд форми входу з використанням багатофакторної автентифікації.

1.2. Переваги та недоліки

1.2.1. Переваги та недоліки парольної автентифікації

Парольна автентифікація є одним з найпоширеніших методів автентифікації, проте як і любий метод автентифікації він є не ідеальний і має свої переваги та недоліки.

Переваги парольної автентифікації:

- **Простота використання**: Паролі є простими у використанні для більшості користувачів. Їх можна легко створити, запам'ятати і використовувати.
- **Низькі витрати на реалізацію**: Інфраструктура парольної автентифікації не потребує складних технологій або дорогих ресурсів. Більшість систем вже підтримують автентифікацію за допомогою паролів.
- **Гнучкість і змінність**: Користувачі можуть легко змінювати свої паролі при необхідності, що дозволяє їм зберігати безпеку своїх облікових записів.

Недоліки парольної автентифікації:

- **Низький рівень безпеки**: Паролі можуть бути легко вгадані або вкрадені. Багато користувачів використовують слабкі паролі, такі як прості слова або комбінації цифр які легко вгадати. Це робить їх вразливими перед атаками на злам паролів.
- **Вимога до складності**: Для збільшення безпеки, паролі потребують складної структури, такої як поєднання великих і малих літер, цифр і спеціальних символів. Однак, це може ускладнити запам'ятовування паролів.
- **Використання одного пароля для кількох сервісів**: Багато користувачів мають звичку використовувати один і той самий пароль для різних сервісів. Це створює ризик, що якщо один пароль стає компрометованим, всі облікові записи користувача стають вразливими.

1.2.2. Переваги та недоліки біометричної автентифікації

Біометрична автентифікація стає все більш популярною у різних сферах життя, включаючи смартфони, комп'ютери, фінансові установи, охоронні системи та громадський транспорт. Вона пропонує доволі зручний та безпечний спосіб ідентифікації, оскільки вимагає лише фізичного представлення особи, що унеможливлює втрату або забування паролю. Однак, варто зазначити, що біометрична автентифікація має як переваги так і недоліки.

Переваги біометричної автентифікації:

- **Висока безпека**: Біометричні характеристики унікальні для кожної особи, що робить їх важкими для підробки або викрадення.
- **Зручність**: Використання біометричних даних для автентифікації зручне для користувачів, оскільки не потрібно запам'ятовувати паролі або пін-коди.
- **Швидкість**: Біометрична автентифікація може бути швидшою в порівнянні з традиційними методами, оскільки не потрібно вводити пароль або пін-код.
- **Відсутність забуття або втрати**: Оскільки біометричні характеристики прив'язані до самої особи, неможливо їх забути або втратити, як це можливо з паролями чи картками доступу.

Недоліки біометричної автентифікації:

- **Приватність і конфіденційність**: Збір і збереження біометричних даних може породжувати питання щодо приватності та можливості використання цих даних для недоречних цілей.
- **Необоротність**: У разі викрадення або компрометації біометричних даних, їх неможливо змінити або скасувати, на відміну від паролів чи пін-кодів, які можна змінити у разі потреби.

- **Вартість та складність**: Впровадження біометричної автентифікації вимагає спеціалізованого обладнання та програмного забезпечення, що може бути витратним і складним процесом для організації.
- **Чутливість до зовнішніх факторів**: Деякі біометричні характеристики можуть бути чутливими до змін у зовнішніх умовах, що може впливати на точність ідентифікації, наприклад, освітлення або пошкодження шкіри.

1.2.3. Переваги та недоліки автентифікації на основі місцезнаходження

Автентифікація на основі місцезнаходження є доволі специфічною а через це має свої переваги і недоліки.

Переваги автентифікації на основі місцезнаходження:

- **Додатковий рівень безпеки**: Використання місцезнаходження для автентифікації може допомогти виявити незвичну або підозрілу активність. Якщо користувач намагається отримати доступ з незвичайного або недозволеного місця, система може відхилити такий доступ або вимагати додаткову перевірку.
- **Географічне обмеження доступу**: Використання автентифікації на основі місцезнаходження дозволить обмежити доступ до веб-застосунків залежно від географічного регіону. Це корисно для захисту конфіденційної інформації або дотримання правових вимог у певних юрисдикціях.
- **Викриття шахрайства та несанкціонованої діяльності**: Автентифікація на основі місцезнаходження може допомогти виявити випадки шахрайства або несанкціонованої діяльності, такі як спроби несанкціонованого доступу або фінансових маніпуляцій з незвичайних місць.

Недоліки автентифікації на основі місцезнаходження:

- **Обхід та обман**: Деякі користувачі можуть намагатися обійти автентифікацію на основі місцезнаходження шляхом використання технологій, які змінюють їх місцезнаходження, таких як віртуальні приватні мережі (VPN).
- **Недостовірні дані місцезнаходження**: Деякі фактори, такі як технічні помилки, неправильні налаштування або використання анонімайзерів, можуть призвести до неправильних або недостовірних даних місцезнаходження. Це може призвести до відхилення доступу правильним користувачам або неправильного допуску недозволеним.
- **Приватність та недовіра**: Збір і використання інформації про місцезнаходження може порушувати приватність користувачів і спричиняти недовіру. Користувачі можуть бути обережні щодо передачі своїх геолокаційних даних через можливість зловживання або недостатню захищеність такої інформації.

Враховуючи ці переваги і недоліки, вирішення про використання автентифікації на основі місцезнаходження у веб-застосунках повинно здійснюватися з урахуванням конкретних потреб, ризиків та вимог безпеки кожного конкретного випадку.

1.2.4. Переваги та недоліки автентифікації за допомогою сертифікатів

Переваги автентифікації за допомогою сертифікатів:

- **Високий рівень безпеки**: Сертифікати забезпечують безпечний механізм перевірки ідентичності, запобігаючи підробці та підставному представленню даних.
- **Сумісність**: Сертифікати базуються на стандартах, що дозволяє їх використання в різних системах і сприяє взаємодії між ними.

- **Централізоване управління**: Центри сертифікації забезпечують контроль над виданням, відкликанням та оновленням сертифікатів, спрощуючи управління ідентифікацією користувачів.

Недоліки автентифікації за допомогою сертифікатів:

- **Складність налаштування**: Інтеграція та встановлення сертифікатів може бути складним процесом, особливо для звичайний користувачів або систем.
- **Залежність від центру сертифікації**: Для перевірки сертифіката необхідна довірена інфраструктура центру сертифікації. Якщо центр сертифікації компрометований або недоступний, можуть виникнути проблеми з автентифікацією.
- **Складність використання**: Звичайним користувачам може бути складно встановити та налаштовувати сертифікати на своїх пристроях і виконувати необхідні кроки для їх використання під час автентифікації.
- **Вартість**: Отримання та керування сертифікатами можуть бути пов'язані з витратами, особливо для організацій, що використовують багато сертифікатів або прагнуть до високого рівня безпеки.
- **Обмеження приватності**: Використання сертифікатів може вимагати розкриття деяких персональних даних центру сертифікації або іншим сторонам, що може порушувати приватність.

Загалом, автентифікація за допомогою сертифікатів є ефективним та безпечним способом перевірки ідентичності користувача, особливо в сферах, де важлива конфіденційність та безпека даних. Втім, необхідно враховувати складність налаштування та використання, залежність від центру сертифікації, вартість та можливі обмеження приватності.

1.2.5. Переваги та недоліки багатофакторної автентифікації

Багатофакторна автентифікація має кілька переваг і недоліків:

Переваги багатофакторної автентифікації:

- **Підвищена безпека**: Використання декількох факторів автентифікації зменшує ризик несанкціонованого доступу. Навіть якщо один фактор компрометований, зловмиснику буде неможливо отримати доступ без інших факторів.
- **Захист від парольного відтворення**: Оскільки MFA не залежить виключно від пароля, він зменшує ризик використання слабких або повторюваних паролів.
- **Гнучкість вибору факторів**: MFA дозволяє вибирати різні фактори залежно від потреб і можливостей користувачів, що забезпечує більшій комфорт і зручність використання.

Недоліки багатофакторної автентифікації (MFA):

- **Складність впровадження**: Імплементація багатофакторної автентифікації може бути складною і вимагати додаткових ресурсів та інтеграції з існуючими системами.
- **Додаткові затрати та зусилля**: Використання MFA може вимагати витрат на придбання та управління фізичними об'єктами або біометричними технологіями. Крім того, користувачам потрібно здійснювати додаткові кроки для автентифікації, що може бути не зручним.
- **Можливі проблеми з сумісністю**: Деякі системи або платформи можуть не підтримувати всі доступні фактори MFA, що може ускладнити їх використання.

Не зважаючи на недоліки, використання багатофакторної автентифікації є важливим кроком у покращенні безпеки інформаційних систем, особливо в умовах зростаючої

кількості кібератак. Оптимальний підхід полягає в збалансованому поєднанні рівнів безпеки та зручності для користувачів.

1.3. Популярність

Останнім часом зростає свідомість про важливість безпеки та конфіденційності в онлайн-середовищі. Користувачі все більше довіряють веб-застосункам свої особисті дані, тому методи автентифікації стають ключовим елементом у забезпеченні доступу лише правомірним користувачам. Популярність різних методів автентифікації у веб-застосунках постійно змінюється, враховуючи безпекові стандарти та зручність використання для користувачів. Для аналізу популярності методів автентифікації був використаний список найпопулярніших вебсайтів у світі за версією сайту SimilarWeb. Зі списку були виключені веб-сайти з Росії, а також ті, які містять непристойний контент або не вимагають автентифікації користувачів.

В таблиці 1.3.1. Продемонстровано доступність різних методів автентифікації на веб-сайтах.

Вебсайт	Парольна автентифікація	Біометрична автентифікація	Автентифікація на основі місцезнаходження	Автентифікація за допомогою сертифікатів	Багатофакторна автентифікація	Інші методи автентифікації
google.com	✓	✓	✓	✗	✓	✓
youtube.com	✓	✓	✓	✗	✓	✓
facebook.com	✓	✓	✓	✗	✓	✓
baidu.com	✓	✗	✗	✗	✗	✗
instagram.com	✓	✓	✗	✗	✓	✓
twitter.com	✓	✓	✓	✗	✓	✓
yahoo.com	✓	✗	✓	✗	✓	✓
wikipedia.org	✓	✗	✗	✗	✗	✗
amazon.com	✓	✓	✗	✗	✓	✓
live.com	✓	✗	✗	✗	✓	✗
netflix.com	✓	✓	✓	✗	✓	✗
reddit.com	✓	✗	✗	✗	✓	✓
qq.com	✓	✗	✗	✗	✗	✓
ebay.com	✓	✗	✓	✗	✓	✓
bing.com	✓	✗	✗	✗	✓	✗
linkedin.com	✓	✓	✓	✗	✓	✓
whatsapp.com	✓	✓	✗	✗	✓	✓
aliexpress.com	✓	✗	✗	✗	✓	✓
sogou.com	✓	✗	✗	✗	✗	✓
pinterest.com	✓	✗	✗	✗	✓	✓
office.com	✓	✗	✗	✗	✓	✗
msn.com	✓	✗	✗	✗	✓	✗
twitch.tv	✓	✗	✗	✗	✓	✓
paypal.com	✓	✓	✗	✗	✓	✓
imdb.com	✓	✗	✗	✗	✓	✓
taobao.com	✓	✗	✗	✗	✗	✓
github.com	✓	✓	✗	✓	✓	✓
microsoft.com	✓	✓	✓	✓	✓	✓

Таблиця 1.3.1. – Доступність різних методів автентифікації на веб-сайтах.

З результатів аналізу можна зробити висновок, що на даний момент парольна автентифікація та доповнення до неї у вигляді багатофакторної автентифікації є найбільш поширеними способами підтвердження ідентичності користувача на веб-сайтах. Більшість веб-сайтів також надають інші способи автентифікації, наприклад, через SMS-повідомлення або за допомогою облікових записів Google або Apple.

Хоча біометрична автентифікація стає все більш популярною, вона все ще не так широко використовується на веб-сайтах. Автентифікація на основі сертифікатів, зі свого боку, є найменш поширеною формою автентифікації на даний момент.

1.4. Складність реалізації та використання

Окрім надійності, простота реалізації та використання методів автентифікації є також важливою як для користувачів, так і для розробників.

Для користувачів, простота використання методів автентифікації є ключовим фактором. Складні процедури автентифікації можуть бути заплутаними та вимагати додаткових зусиль від користувачів. Простота використання допомагає забезпечити зручність та швидкість проходження процесу автентифікації, що робить його приємним і безпроблемним для користувачів.

Для розробників, простота реалізації та використання методів автентифікації відіграє важливу роль у швидкості і надійності розробки систем. Легкість інтеграції методів автентифікації у вже існуючі системи та програми дозволяє розробникам ефективно використовувати їх без зайвих зусиль. Простота реалізації допомагає зменшити ризик помилок і забезпечує гнучкість у конфігурації та управлінні методами автентифікації.

Отже, простота використання та реалізації методів автентифікації є важливими факторами, що сприяють покращенню досвіду користувачів та ефективності розробки для розробників.

В таблиці 1.4.1. рродемонстровано порівняння складності реалізації та використання різних методів автентифікації у вебзастосунках.

Метод автентифікації	Складність реалізації	Складність використання
Парольна автентифікація	Низька	Середня
Біометрична автентифікація	Висока	Середня
Автентифікація на основі місцезнаходження	Середня	Середня
Автентифікація за допомогою сертифікатів	Висока	Висока
Багатофакторна автентифікація	Середня	Середня

Таблиця 1.4.1. – Порівняння складності реалізації та використання різних методів автентифікації у вебзастосунках.

Парольна автентифікація, хоч і є широко використовуваним методом, проте має свої складності в реалізації та використанні. З точки зору реалізації, система повинна забезпечити безпечне збереження паролів, а також потрібно розробити механізми для керування паролями для попередження зламу паролів.

З точки зору використання, **парольна автентифікація** може бути викликом для користувачів. Часто вони мають згадувати та пам'ятати багато різних паролів для різних сервісів або облікових записів. Це може стати незручним та призводити до використання слабких паролів або повторного використання одного пароля для кількох сервісів. Також можуть виникати проблеми зі зміною паролів та відновленням доступу в разі його втрати.

В свою чергу для **біометричної автентифікації** необхідно мати належне обладнання для збору та обробки біометричних даних. Це можуть бути сканери відбитків пальців, камери для розпізнавання обличчя, сенсори розпізнавання голосу або інші пристрої. Такі обладнання повинні бути надійними та точними для ефективної роботи з біометричними даними.

Окрім цього, розробники повинні враховувати питання безпеки при роботі з біометричними даними. Ці дані є особистими та унікальними для кожного користувача, тому важливо забезпечити їх захист від несанкціонованого доступу або злому. Це включає в себе шифрування біометричних даних, зберігання їх в захищених базах даних та використання механізмів ідентифікації та автентифікації. З точки зору використання, біометрична автентифікація може стикатися з деякими проблемами. Наприклад, точність розпізнавання може залежати від якості збору біометричних даних, таких як освітлення, якість зображення тощо. Також можуть виникати проблеми зі збором біометричних даних у випадках, коли користувач має пошкодження або зміни в фізичних характеристиках.

Щодо **автентифікації на основі місцезнаходження**, впровадження цього методу потребує складних технічних рішень. Для його реалізації необхідно мати доступ до даних про місцезнаходження користувача, таких як геолокація через GPS, IP-адресу, або інші методи. Однак, точність та надійність цих даних можуть бути обмеженими, особливо в умовах з обмеженим доступом до мережі або у внутрішніх приміщеннях, що може призводити до помилок під час автентифікації та незручностей для користувачів.

Автентифікація за допомогою сертифікатів є складною з точки зору реалізації та використання. Вона вимагає розгортання інфраструктури публічного ключа (PKI), що включає в себе випуск, розповсюдження та управління сертифікатами, а також інтеграції з додатками та системами. Це може вимагати змін у програмному забезпеченні, розширеннях протоколів або налаштуванні схеми автентифікації. Крім того, користувачам може бути складно встановити та налаштовувати сертифікати на своїх пристроях і виконувати необхідні кроки для їх використання під час автентифікації.

Багатофакторна автентифікація в свою чергу, може бути дещо складнішою для реалізації та використання порівняно зі звичайними методами автентифікації, такими як пароль.

Для реалізації багатофакторної автентифікації необхідно налаштувати та інтегрувати додаткові фактори автентифікації, такі як фізичний пристрій (наприклад, токен або смарт-карта), біометричні дані (наприклад, сканування відбитку пальця або розпізнавання обличчя) або тимчасові паролі. Це вимагає наявності певних апаратних або програмних засобів для підтримки цих факторів.

Для користувачів, використання MFA може вимагати додаткових кроків під час процесу автентифікації. Наприклад, користувач може заздалегідь налаштувати свій смартфон як додатковий фактор автентифікації і отримувати повідомлення з тимчасовими кодами підтвердження. Цей код необхідно буде ввести разом зі своїм паролем, щоб отримати доступ до облікового запису.

1.5. Підсумок

Протягом останніх десятиліть було розроблено та впроваджено багато методів автентифікації з метою забезпечення надійного захисту доступу до інформації та ресурсів. Проте на сьогоднішній день, немає єдиного та ідеального методу, що задовольняв би всі потреби.

Одним з найпоширеніших методів автентифікації є парольна автентифікація. Паролі дозволяють доволі зручно підтвердити особистість користувача в комп'ютерних системах, проте їх використання може бути небезпечним через можливість зламу та недостатню складність. Через це, для посилення безпеки майже всі вебзастосунки комбінують цей метод з багатофакторною автентифікацією. Замість одного пароля, який може бути вкрадений або вгаданий, багатофакторна автентифікація вимагає введення додаткових факторів, наприклад SMS-кодів підтвердження або тимчасових кодів.

В останні роки біометрична автентифікація набуває популярності, проте саме у вебзастосунках її використання є все ще сильно обмеженим по причині того, що такий метод автентифікації потребує спеціалізованого обладнання та програмного забезпечення, яке не завжди є доступним для звичайних користувачів.

Щодо автентифікації на основі місцезнаходження то на даний момент її рідко використовують як самостійний спосіб ідентифікації користувачів через його обмежену точність, проте він добре зарекомендував себе як додатковий шар захисту який може допомогти виявити незвичну або підозрілу активність.

В свою чергу автентифікація за допомогою сертифікатів є ефективна та безпечна, але не зручна для повсякденного використання звичайними користувачами через складність створення та налаштування сертифікатів на пристроях.

2. ОБРАНІ ТЕХНОЛОГІЇ

2.1. Фреймворк Apache Maven

Apache Maven – це фреймворк для автоматизації роботи з програмними проектами. Основна конфігурація збірки відбувається за допомогою xml файла, який містить в собі опис проекту, його властивості, зв’язки зі зовнішніми модулями і компонентами, конфігурацію порядку побудови та налаштування плагінів які використовуються на різних етапах збірки проекту. Зовнішні модулі можуть знаходитись як в локальному репозиторії так і на віддалених з можливістю динамічного підвантаження. Для опису конфігурації, використовується POM конструкція.

2.1.1. Об'єктна модель опису проекту

Основна конфігурація побудови проекту, підтримуваного Apache Maven, описується в xml файлі, який за замовчуванням називається pom.xml. На початку побудови проекту відбувається перевірка конфігураційного файлу на його синтаксичну правильність та наявність всіх необхідних даних.

Приклад файлу pom.xml:

```
<project>
  <!-- версія моделі для POM-файлу Maven 2.x завжди 4.0.0 -->
  <modelVersion> 4.0.0 </ modelVersion>
  <!-- координати проекту, тобто набір значень, який
       дозволяє однозначно ідентифікувати цей проект -->
  <groupId> com.mycompany.app </ groupId>
  <artifactId> my-app </ artifactId>
  <version> 1.0 </ version>
  <!-- залежності -->
  <dependencies>
    <dependency>
      <!-- координати необхідної бібліотеки ->
```

```
      <groupId> junit </ groupId>
      <artifactId> junit </ artifactId>
      <version> 3.8.1 </ version>
      <! - ця бібліотека використовується тільки для запуску і компілювання тестів ->
      <scope> test </ scope>
    </ dependency>
  </ dependencies>
</ project>
```

2.2. Мова програмування Java

Java - це високорівнева, об'єктно-орієнтована мова програмування загального призначення, основна ідея якої полягає в зменшенні залежностей реалізації від операційної системи. Випущена 1995 року компанією “Sun Microsystems”. Ця мова програмування надає програмістам можливість писати код один раз і запускати в будь-якому місці, тобто скомпільований код може працювати на любій платформі яка підтримує Java, без необхідності перекомпіляції. Програми написані мовою програмування Java компілюються у байт-код, який запускається на JVM. Використання такого прошарку робить код незалежним від архітектури комп'ютера на якому він запускається. На відміну від мов програмування C та C++, Java має менше низькорівневих засобів. Проте, через прошарок у вигляді JVM, середовище виконання Java надає програмісту динамічні можливості відображення та модифікування написаного коду, під час виконання програми. Для традиційних мов програмування, які перед виконанням, повністю перетворюються в машинний код, такі можливості зазвичай недоступні.

2.2.1. Особливості

Особливості мови програмування Java:

- **Незалежність від платформи**: Компілятор перетворює вихідний код у байт-код, який потім виконується JVM. Байт-код може працювати на будь які платформі: Windows, Linux, macOS. Адже для кожної операційної системи є окрема JVM, яка дозволяє запустити байт-код.
- **Об'єктно-орієнтована мова програмування**: Використаний підхід до програмування, в основі якого лежить поняття об’єкта як певної сутності.
- **Простота**: Java це доволі проста мова, оскільки вона не має таких складних функцій як: вказівники, перевантаження операторів, множинне спадкування, явне виділення пам'яті.
- **Надійність**: Java розроблена так, що докладає багато зусиль для перевірки помилок якомога раніше, тому компілятор java здатний виявити навіть ті помилки, які нелегко виявити іншою мовою програмування.
- **Безпечність**: У Java неможливі такі недоліки безпеки як пошкодження стека або переповнення буфера.
- **Багатопотоковість**: Java підтримує багатопотоковість.
- **Висока продуктивність**: Архітектура Java визначена таким чином, що вона зменшує накладні витрати під час виконання. Для цього використовується компілятор Just In Time (JIT), який компілює лише ті методи які потрібні лише в певний момент часу.
- **Динамічна гнучкість**: Оскільки Java повністю об'єктно-орієнтована мова програмування, програміст може легко розширювати написані програми, завдяки додаванню нових класів, методів до існуючих класів і навіть створюючи нові класи через підкласи.

2.3. Фреймворк Spring

Сьогодні Spring фреймворк це стандарт за замовчуванням при створенні будь якої програми Java на професійному рівні.

Spring — це фреймворк з відкритим вихідним кодом та контейнери з підтримкою інверсії управління для платформи Java, розроблений Родом Джонсоном у 2003 році. Цей фреймворк надає багато функцій з точки зору простоти, можливості тестування та слабкості зв'язування. Spring пропонує на заміну старим Java об'єктам нові, які називають beans, для досягнення речей, які раніше були можливими лише з технологією Enterprise Java Beans (EJB).

Однією з головних переваг Spring є те, що він легкий та не викликає засмічення програми власним API. Spring є модульним, що дозволяє використовувати лише ті модулі, які вам потрібні, не вводячи нічого зайвого. Ці модулі надають усе необхідне для розробки програми корпоративного рівня. Використання модульної архітектури дозволяє уникнути повного базування програми на основі Spring фреймворк. Програміст може вибрати лише такий модуль, який відповідає його потребам. Також Spring пропонує слабке зв'язування компонентів у програмі за допомогою ін'єкції залежностей.

Ін'єкція залежностей (Dependency Injection) - це фундаментальний аспект фреймворку Spring , за допомогою якого контейнер Spring впроваджує об'єкти в інші об'єкти або залежності. Простіше кажучи, це дозволяє реалізувати слабке зв'язування компонентів і перекладає відповідальність за керування компонентами на сторону контейнера.

2.3.1. Spring архітектура

На рис. 2.3.1.1. зображена діаграма яка демонструє основні компоненти архітектури Spring.

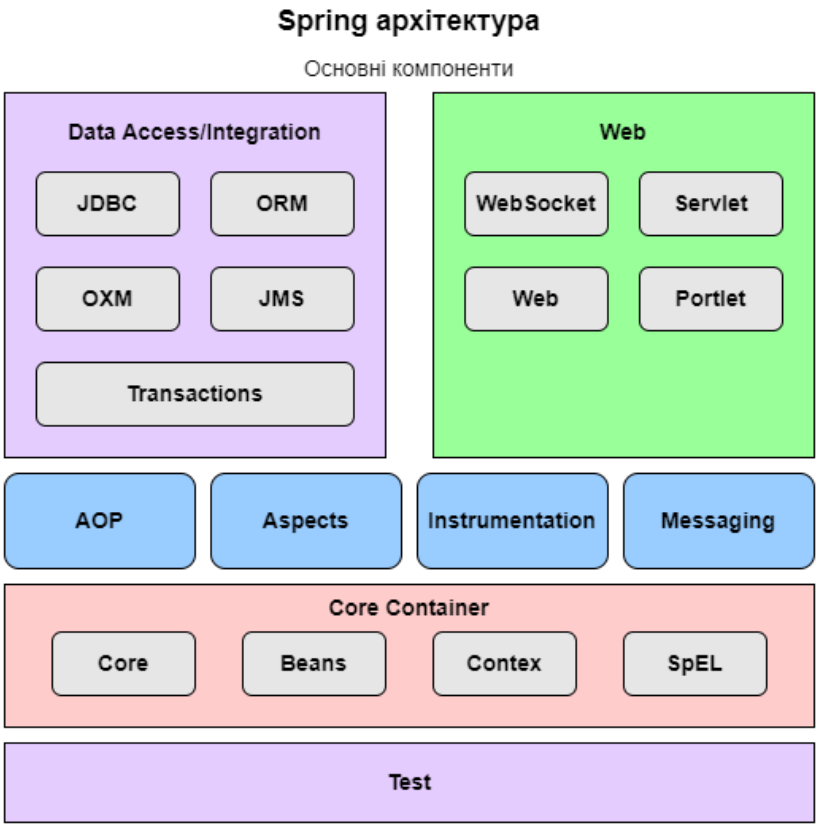

Рис. 2.3.1.1. – Основні компоненти архітектури Spring.

Контейнер **Core**, містить основні модулі, які є фундаментом фреймворку Spring:

- **Core (spring-core)** – модуль, який є ядром фреймворка, забезпечує роботу таких функцій, як інверсія керування та ін'єкція залежностей.
- **Beans (spring-beans)** – модуль, який забезпечує Beanfactory, яка є складною реалізацією патерну Фабрика.
- **Context (spring-context)** – модуль, який ґрунтується на Core та Beans і надає середовище для доступу до визначених об’єктів. Інтерфейс ApplicationContext є основною частиною модуля Context, а spring-context-support забезпечує

підтримку сторонніх взаємодій, таких як кешування, розсилка та механізми шаблонів.

- **SpEL (spring-expression)** – модуль, який дозволяє користувачам використовувати мову Spring Expression для запитів та керування об'єктами під час виконання.

Контейнер **Data access / integration** (Доступ до даних / інтеграція), містить модулі, які використовуються для надання доступу до даних та обробки транзакцій у програмі. Серед основних, варто відзначити:

- **JDBC (spring-jdbc)** – модуль, який забезпечує рівень абстракції JDBC, усуває необхідність окремого налаштування JDBC під час роботи з базами даних.
- **ORM (spring-orm)** – модуль, який забезпечує рівень інтеграції для популярних API об’єктно-реляційного відображення, таких як JPA, JDO Hibernate.
- **OXM (spring-oxm)** – модуль, який надає підтримку для перетворення XML-документа з та в об'єкти.
- **JMS (spring-jms)** – модуль служби обміну повідомленнями Java, який створює та використовує повідомлення, які безпосередньо інтегруються з модулем обміну повідомленнями Spring.
- **Transaction (spring-tx)** - модуль, який пропонує програмне та декларативне керування транзакціями для класів, які включають спеціальні інтерфейси та POJO.

Контейнер **Web**, містить модулів, які забезпечують роботу веб-функцій у Spring додатку. Серед основних, варто відзначити:

- **WebSocket (spring-websocket)** – модуль, який забезпечує зв'язок клієнтів і сервера на основі протоколу WebSocket.

- **Servlet (spring-webmvc)** – модуль, який містить реалізації MVC і REST.
- **Web (spring-web)** – модуль, який надає всі основні веб-орієнтовані функції та містить HTTP-клієнт.
- **Portlet (spring-webmvc-portlet)** – модуль, який забезпечує реалізацію MVC для використання в середовищі портлетів.

Інші модулі:

- **AOP (spring-aop)** – модуль, який забезпечує реалізацію аспектно-орієнтовану програмування, яку можна використовувати під час створення додатків.
- **Aspects (spring-aspects)** – модуль, який забезпечує пряму інтеграцію з програмним розширенням AspectJ для середовища розробки Eclipse.
- **Instrumentation (spring-instrument)** – модуль, який забезпечує підтримку виявлення класів і реалізацію завантажувача класів, для серверних додатків.
- **Messaging (spring-messaging)** – модуль, який забезпечує надійну платформу для керування повідомленнями в програмах.
- **Test (spring-test)** – модуль, який забезпечує модульне та інтеграційне тестування з JUnit і TestNG.

2.4. Фреймворк Spring Boot

Spring Boot — це фреймворк з відкритим кодом, який підтримується компанією Pivotal Team. З його допомогою розробники можуть швидко розпочати роботу, не витрачаючи часу на підготовку та початкове налаштування програми на основі Spring фреймворку.

Як підсумовано на рис. 2.4.1., Spring Boot, це комбінація фреймворку Spring і вбудованих серверів без необхідності повної конфігурації через .xml файли.

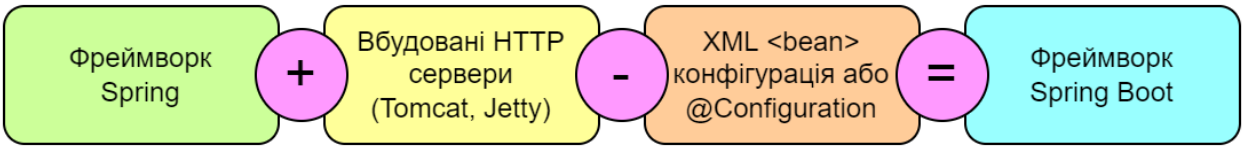

Рис. 2.4.1. – Сутність модифікації Spring Boot

У Spring Boot немає вимог до XML конфігурації. Він використовує парадигму конфігураційного проектування програмного забезпечення, що означає, що це зменшує зусилля розробника. Його основна мета, це скоротити час розробки та тестування.

Spring Boot дотримується багаторівневої архітектури, в якій кожен рівень взаємодіє з рівнем безпосередньо під ним або над ним (ієрархічна структура). На рис. 2.4.2. зображені ключові рівні архітектури Spring Boot фреймворку та їхня взаємодія.

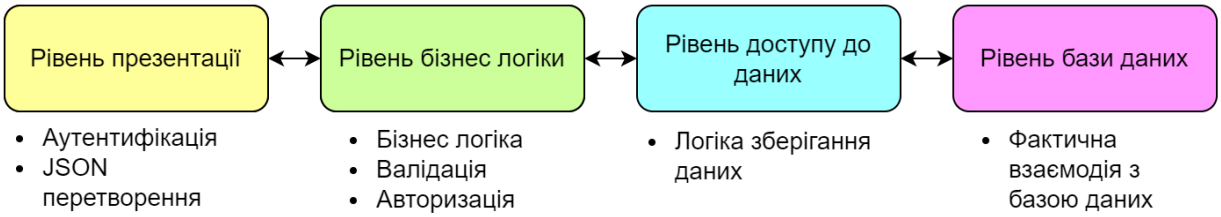

Рис. 2.4.2. – Ключові рівні архітектури Spring Boot фреймворку та їхня взаємодія

Опис основних рівнів архітектури Spring Boot фреймворку:

- **Рівень презентації**: обробляє HTTP запити, перетворює параметр JSON в об'єкт, автентифікує запити і передає їх на рівень бізнес логіки.
- **Рівень бізнес логіки**: обробляє всю бізнес-логіку, та реалізовує взаємодію з різноманітними класами сервісами, а також виконує авторизацію та валідацію даних.
- **Рівень доступу до даних**: містить усю логіку зберігання даних та перетворення бізнес-об'єктів з та в рядки бази даних.
- **Рівень бази даних**: на рівні бази даних виконуються операції CRUD (створення, читання, оновлення, видалення).

2.5. JSON Web Token

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами через об'єкт JSON. Цю інформацію можна перевірити на оригінальність, оскільки вона має цифровий підпис. JWT можна підписати за допомогою секретного ключа (за допомогою алгоритму HMAC) або пари відкритих/приватних ключів за допомогою RSA або ECDSA.

Через його відносно невеликий розмір, JWT можна надіслати у тілі POST методу або всередині заголовка HTTP і на передачу такої кількості даних не потрібно буде витрачати велику кількість мережевих ресурсів. JWT містить всю необхідну інформацію про сутність, щоб уникнути запитів до бази даних більше одного разу. Одержувачу JWT також не потрібно викликати сервер для перевірки маркера.

Проте варто зауважити, що успішно перевірений маркер, означає лише те, що інформація, що міститься в токені є оригінальна і не була підмінена. Проте ця інформація не є захищена від читання. Через це, в середині JWT ніколи не треба зберігати конфіденційну інформацію і слід вживати інших додаткових заходів, щоб

гарантувати, що JWT не буде перехоплено. Наприклад, надсилати JWT лише через HTTPS з’єднання, дотримуючись найкращих практик.

2.5.1. Структура

Структура JWT токену складається з трьох частин які кодуються за допомогою шифрування Base64url і об’єднуються за допомогою крапок.

Приклад структури продемонстрований на рис. 2.5.1.1.

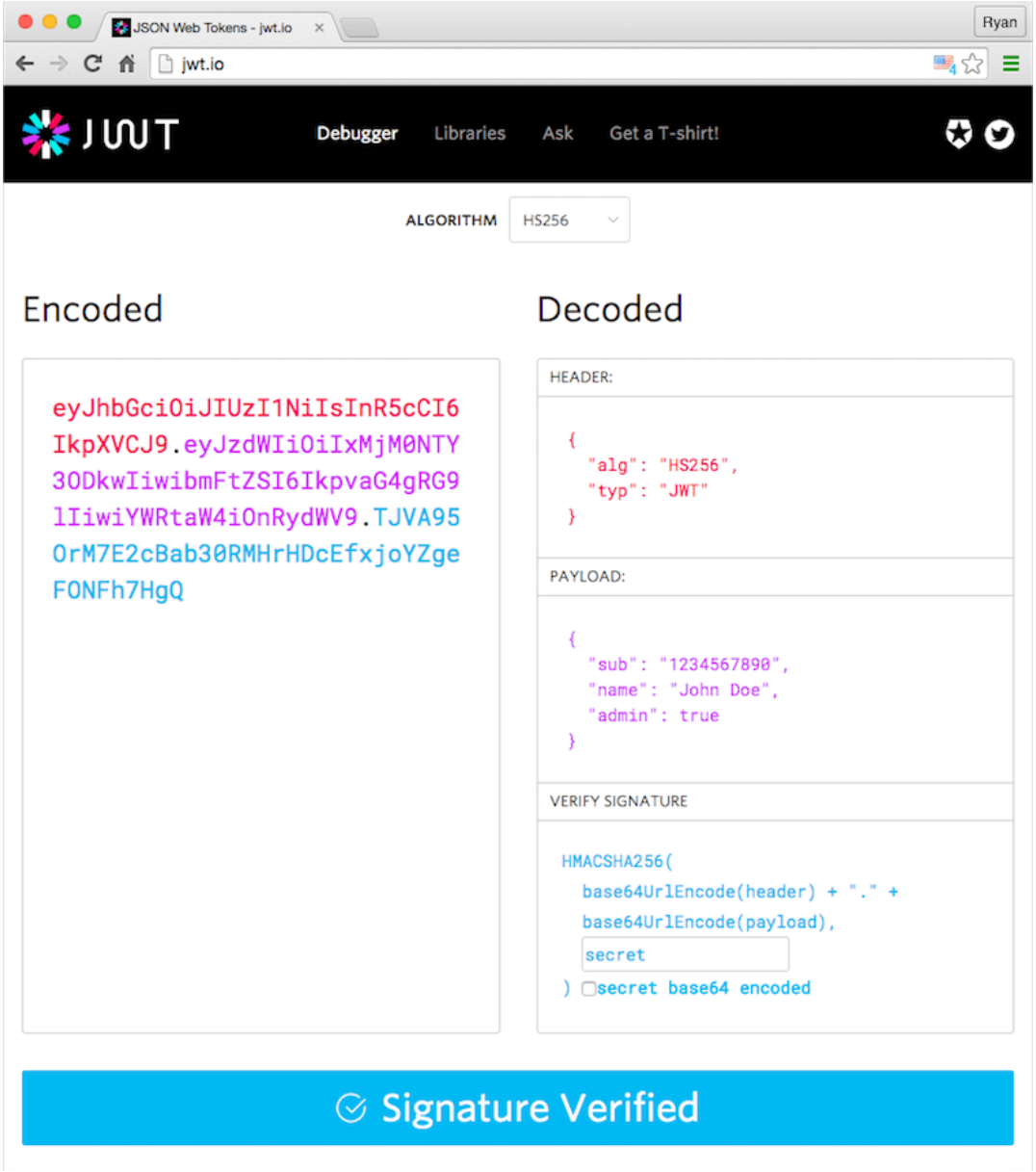

Рис. 2.5.1.1. – Структура JWT токену.

Основні частини структури JWT:

- **Заголовок**: Визначає, який алгоритм використовується для створення підпису.
- **Корисне навантаження**: Містить корисну інформацію яка повинна бути захищена від підміни.
- **Підпис**: Ключ який гарантує оригінальність заголовку та корисного навантаження. Для кодування використовується криптографічний алгоритм вказаний у заголовку.

2.6. Мова програмування TypeScript

TypeScript — це об'єктно-орієнтована мова програмування з відкритим вихідним кодом, розроблена та підтримувана Microsoft, ліцензована за ліцензією Apache 2. TypeScript розширює мову програмування JavaScript, додаючи типи даних, класи та інші об'єктно-орієнтовані функції з перевіркою типів. Це типізований JavaScript, який компілюється у звичайний JavaScript. Причина використання TypeScript полягає в тому, що він виявляє помилки під час компіляції, щоб розробник міг виправити їх перед запуском коду.

2.6.1. Переваги

Серед переваг мови програмування TypeScript, варто відзначити:

1. TypeScript — мова з відкритим вихідним кодом, яка постійно розвивається та підтримується Microsoft.
2. TypeScript подібний до JavaScript і використовує такий самий синтаксис і семантику. Це дозволяє розробникам, які програмують на JavaScript, швидше освоїти TypeScript.
3. Код написаний на JavaScript можна використовувати в поєднанні з кодом написаним мовою програмування TypeScript. Через це TypeScript також без проблем працює з існуючими фреймворками та бібліотеками JavaScript.

4. Використання визначення типів даних дозволяє скористатись перевагами перевірки типів та автоматичне заповнення коду.
5. TypeScript підтримує новітні функції JavaScript з ECMAScript 2015. Він включає функції з ES6 і ES7, які можуть працювати в JavaScript двигунах рівня ES5, як-от Node.js. Це дає величезну перевагу використання функцій новіших версій JavaScript у двигунах розрахованих на старіші версії.

2.7. Фреймворк Angular

Angular — це фреймворк з відкритим кодом на основі TypeScript для створення односторінкових клієнтських програм. Розробляється компанією Google, а також приватними розробниками. Цей фреймворк реалізує основні та додаткові функції у вигляді набору бібліотек TypeScript, які розробник імпортуєте у свої програми.

2.7.1. Особливості

Серед особливостей фреймворку Angular, варто відзначити:

- **Об'єктна модель документа**: Angular використовує звичайний DOM, який розглядає XML або HTML-документ як деревовидну структуру, в якій кожен вузол представляє частину документа. Також варто зауважити, що всі оновлення відбуваються на одній HTML сторінці. Оновленню підлягають лише ті теги інформація яких була змінена.
- **Прив'язка даних**: Прив'язка даних – це процес, який дозволяє користувачам маніпулювати елементами веб-сторінки. Він використовує динамічний HTML і не вимагає складних сценаріїв програмування . Прив'язка даних використовується на веб-сторінках, які містять інтерактивні компоненти, такі як калькулятори, навчальні посібники, форуми та ігри. Це також дозволяє краще відображати веб-сторінку, коли сторінки містять велику кількість даних. Angular використовує двостороннє прив'язування. Стан моделі

відображає будь-які зміни, внесені у відповідні елементи інтерфейсу користувача. І навпаки, стан UI відображає будь-які зміни в стані моделі.

- **Тестування**: Angular використовує Jasmine фреймворк для тестування. Фреймворк Jasmine надає безліч функцій для написання різних типів тестових сценаріїв. Karma – це засіб для запуску тестів, який використовує файл конфігурації для налаштування запуску, звітів і меж тестування.
- **Архітектура**: Angular – це повноцінний MVC (модель-представлення-контролер) фреймворк, який надає чіткі вказівки щодо структурування програми та забезпечує двонаправлений потік даних.

Складові частини Angular програми:

- **Модулі**: Програма Angular має кореневий модуль під назвою AppModule, який забезпечує механізм завантаження для запуску програми.
- **Компоненти**: Кожен компонент програми визначає клас, який містить логіку програми та дані. Компонент зазвичай визначає частину інтерфейсу користувача (UI).
- **Шаблони**: Angular шаблони поєднують розмітку Angular з HTML, щоб змінити елементи HTML перед їх відображенням.
- **Метадані**: Метадані повідомляють Angular, як обробляти клас. Він використовується для декорування класу, щоб він міг налаштувати очікувану поведінку класу.
- **Сервіси**: Якщо у вас є дані або логіка, які не пов’язані з представленням, але мають бути спільними між компонентами, створюється клас сервісу. Клас завжди асоціюється з декоратором @Injectible.
- **Ін'єкція залежностей**: Angular використовує шаблон проектування DI (Ін'єкція залежностей), щоб підвищити гнучкість та модульність у своїх програмах.

2.8. TOTP-алгоритм

TOTP (Time-Based One-Time Password) - це алгоритм для генерації тимчасових паролів на основі часу. Він широко використовується для забезпечення двофакторної автентифікації (2FA) та безпеки в різних онлайн-сервісах, таких як інтернет-банкінг, соціальні мережі та інші системи, які потребують додаткового рівня захисту.

Основою TOTP є спільний секрет між сервером і користувачем. Цей секрет, який зазвичай представлений у вигляді текстового ключа, використовується для генерації тимчасових паролів. Сервер і користувач синхронізуються в часі, щоб мати однаковий вихідний пункт для генерації кодів.

Процес генерації тимчасового коду заснований на поточному часі та хеш-функції. Зазвичай використовується хеш-функція SHA-1, але можуть бути використані й інші алгоритми, такі як SHA-256. Ключовий фактор - це часовий фактор, який змінюється з певною фіксованою швидкістю, зазвичай кожні 30 секунд.

Хеш-функції, які використовуються в TOTP, є криптографічно стійкими, що означає, що неможливо зворотньо відновити вихідний текст з хеш-значення. Вони також відомі своєю стійкістю до колізій, тобто надзвичайно мало ймовірно, що два різні вхідні повідомлення мають одне й те ж хеш-значення.

Однак, важливо підкреслити, що криптостійкість TOTP-алгоритму залежить від безпеки секретного ключа та правильного використання алгоритму. Секретний ключ має бути випадковим, достатньо довгим і надійно зберігатися.

На рис. 2.8.1. продемонстрований наглядний приклад обчислення тимчасового паролю двома сторонами (клієнтська і серверна), без підключення до інтернету.

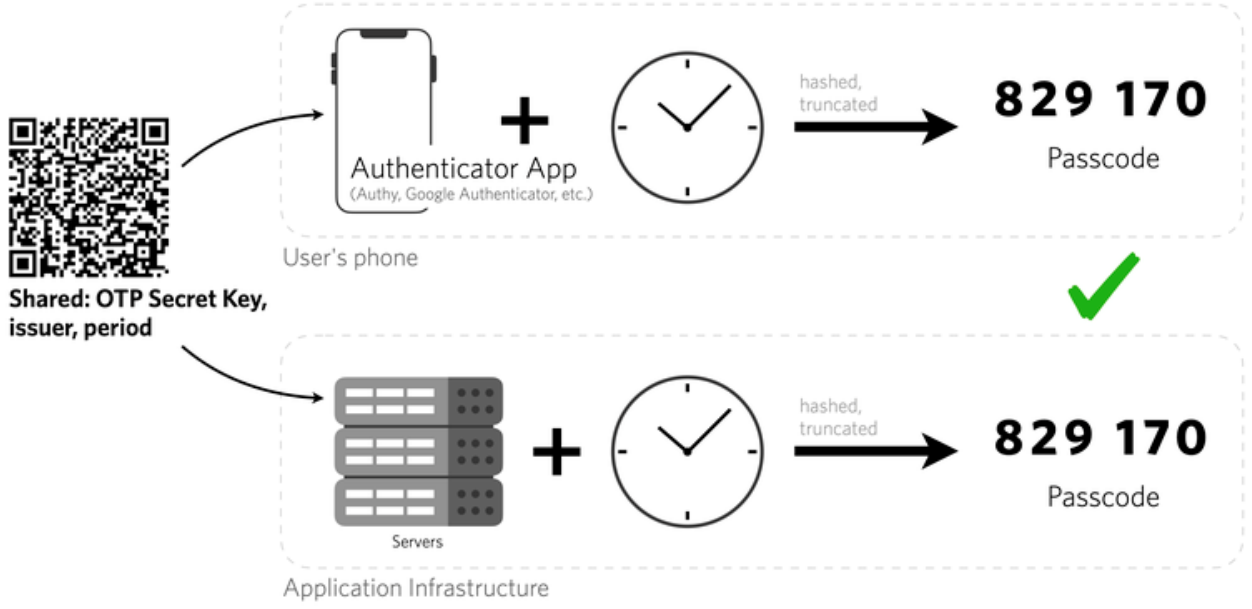

Рис. 2.8.1. – Наглядний приклад обчислення тимчасового коду двома сторонами (клієнтська і серверна) без підключення до інтернету.

Користувач запускає алгоритм TOTP на своєму пристрої, використовуючи спільний секрет і поточний час. Алгоритм генерує хеш-код із поєднанням часового фактора та секрету, який потім перетворюється в тимчасовий код, зазвичай шляхом обрізання частини хеш-коду та перетворення його в шестизначне число.

На сервері, який перевіряє тимчасовий код, також виконується аналогічна процедура генерації тимчасового коду з використанням спільного секрету і поточного часу. Потім сервер порівнює згенеровані дані з тими, що були введені користувачем і приймає рішення щодо автентифікації.

Використання TOTP значно підвищує безпеку автентифікації, оскільки тимчасові коди залежать від поточного часу, і їх можна використовувати лише протягом короткого періоду. Це ускладнює завдання зловмисникам, які намагаються підбирати паролі або використовувати викрадені паролі для несанкціонованого доступу.

3. РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ “MEDIA STORAGE”

3.1. Реалізація серверної частини

3.1.1. Структура проекту

Для створення структури серверної частини проекту було вирішено використати вебсервіс “Spring Initializr”, який дозволяє створити і налаштувати початкову структуру Spring boot проекту. Spring Boot, це корисний фреймворк, метою якого є спрощення розробки Java програм на основі Spring фреймворку. Завдяки такому підходу, початкова конфігурація програми була зведена до мінімуму, що дало можливість пришвидшити початок розробки. На Рис. 3.1.1.1. продемонстровано вигляд головної сторінки вебсервісу “Spring Initializr” з налаштованими параметрами та вибраними залежностями для генерації проекту.

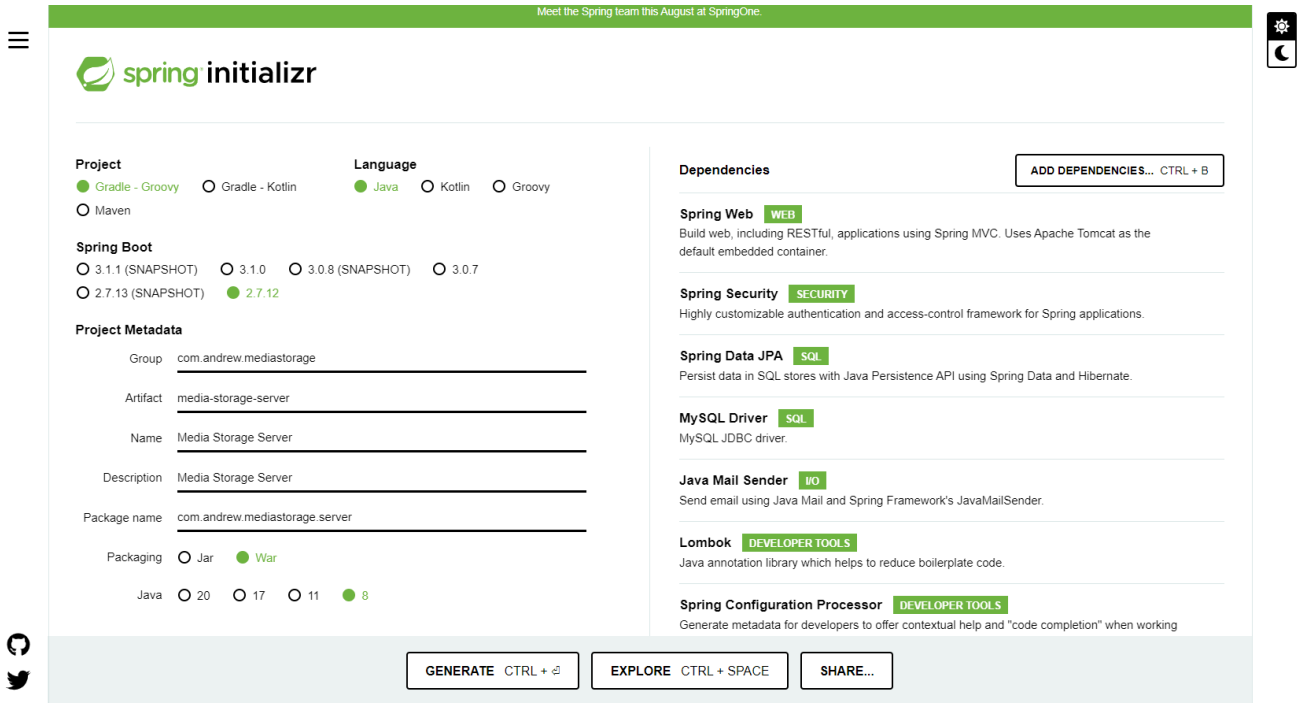

Рис. 3.1.1.1. Вигляд головної сторінки вебсервісу “Spring Initializr” з налаштованими параметрами та вибраними залежностями для генерації проекту.

Залежності, які були вибрані під час генерації проекту:

- **Spring Web** – дозволяє створювати веб-додатки, включаючи RESTful, за допомогою Spring MVC фреймворку.
- **Spring Security** – дозволяє широко налаштовувати систему автентифікації та контролю доступу для додатків Spring.
- **Spring Data JPA** – дозволяє легко реалізувати репозиторії на основі JPA
- **MySQL Driver** – драйвер JDBC і R2DBC, який дозволяє програмам Java підключатися до бази даних MySQL за допомогою стандартного коду Java.
- **Spring Configuration Processor** – дозволяє генеруйте метадані для розробників, щоб запропонувати контекстну допомогу та «завершення коду» під час роботи з властивостями конфігурації.
- **Java Mail Sender** – дозволяє відправляти електронні листи з сервера.
- **Lombok** – Бібліотека анотацій Java, яка допомагає зменшити шаблонний код.

На рис. 3.1.1.2. зображена кінцева структура серверної частини проекту, після створення всіх необхідних пакетів, класів та файлів конфігурації.

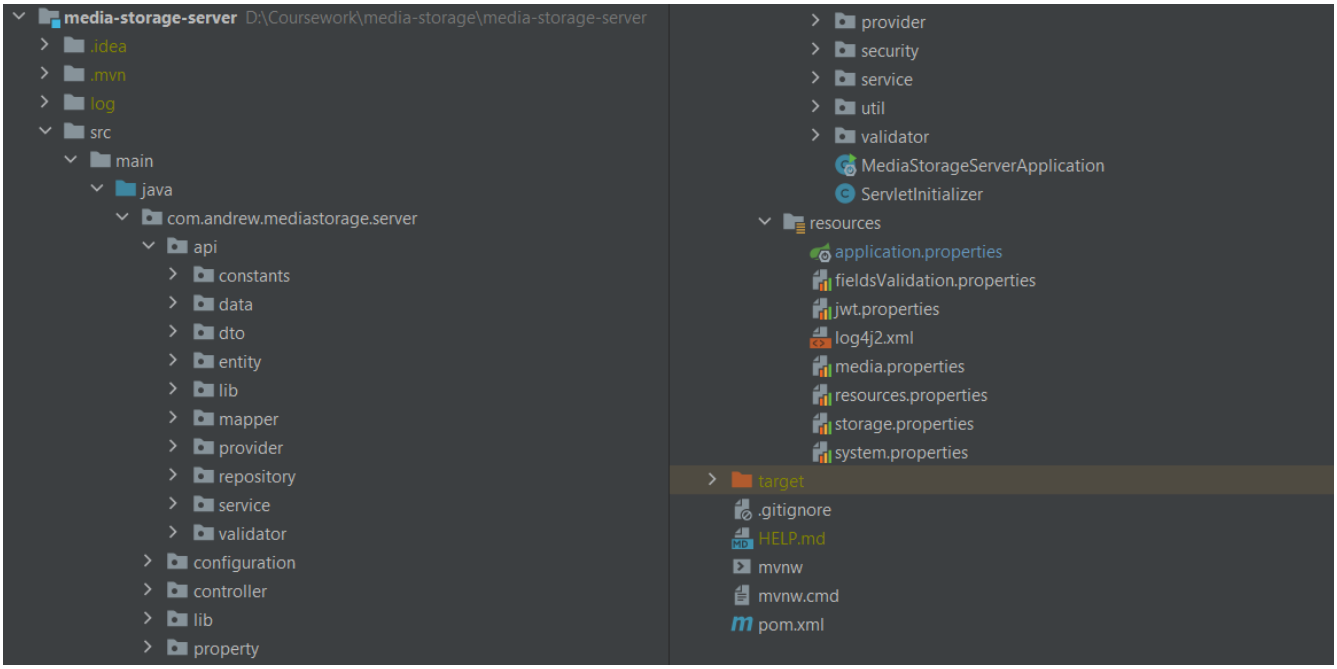

Рис. 3.1.1.2. Кінцева структура серверної частини проекту, після створення всіх необхідних пакетів, класів та файлів конфігурації.

Основним файлом для збірки серверної частини проекту, є файл pom.xml (Project Object Model), який завжди зберігається в базовій директорії проекту. Даний файл містить інформацію про проект та різні деталі конфігурації, які використовуються Maven для побудови проекту.

3.1.2. Автентифікація

Для забезпечення безпеки серверної частини проекту, був використаний фреймворк Spring Security, TOTP-алгоритмом для генерації та перевірки тимчасових кодів та стандарт токена доступу на основі JSON, а саме JWT (JSON Web Token).

Spring Security, це фреймворк, що надає механізми побудови систем автентифікації та авторизації, а також інші можливості забезпечення безпеки для корпоративних додатків, створених за допомогою фреймворку Spring.

Основне налаштування безпеки відбувається в класі SecurityConfig, який розширює абстрактний клас WebSecurityConfigurerAdapter та перевизначає головний метод цього класу: configure().

Метод configure():

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
            .csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .authorizeRequests().antMatchers(
                    LOGIN_ENDPOINT,
                    LOGOUT_ENDPOINT,
                    REGISTRATION_ENDPOINT,
                    REFRESH_TOKEN_ENDPOINTS,
                    CONFIRM_ENDPOINTS,
                    RESTORE_ENDPOINTS,
                    RESET_PASSWORD_ENDPOINTS
            ).permitAll()
            .and()
            .authorizeRequests().anyRequest().authenticated()
            .and()
            .apply(securityConfigurer());
}
```

В цьому методі, відбувається налаштування об’єкту класу HttpSecurity, а саме:

- Визначення кінцевих точок, для яких не потрібно перевіряти автентифікацію.
- Добавлення об’єкту класу SecurityConfigurer, який містить в собі налаштування фільтрів, що будуть застосовуватись до HTTP запитів та відповідей.

Головний метод класу SecurityConfigurer, а саме configure():

```
@Override
public void configure(HttpSecurity httpSecurity) {
    httpSecurity.addFilterBefore(corsFilter,
UsernamePasswordAuthenticationFilter.class);
    httpSecurity.addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);
    httpSecurity.addFilter(authenticationFilter);
}
```

В цьому методі вказуються 3 фільтри, які будуть застосовуватись до HTTP запитів та відповідей, а саме: **corsFilter**, **jwtFilter** та **authenticationFilter**.

Створення та налаштування об’єкту класу **CorsFilter**, відбувається в методі corsFilter():

```
@Bean
public CorsFilter corsFilter() {
    UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
    CorsConfiguration config = new CorsConfiguration();
    config.addAllowedOrigin("*");
    config.addAllowedHeader("*");
    config.addAllowedMethod("OPTIONS");
    config.addAllowedMethod("GET");
    config.addAllowedMethod("POST");
    config.addAllowedMethod("PUT");
    config.addAllowedMethod("PATCH");
    config.addAllowedMethod("DELETE");
    source.registerCorsConfiguration("/**", config);
    return new CorsFilter(source);
}
```

CorsFilter використовується для налаштування політики CORS (Cross-Origin Resource Sharing).

Cross-Origin Resource Sharing (CORS, спільне використання ресурсів з різних джерел) – механізм безпеки сучасних браузерів, який дозволяє вебсторінкам використовувати дані, що знаходяться на інших доменах. Тобто реалізує захист для вебсторінок, які не відповідають політиці одного походження.

JwtFilter це клас, який розширює абстрактний клас OncePerRequestFilter та перевизначає метод doFilterInternal():

```
@Override
protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws
ServletException, IOException {
    if (SecurityConfig.LOGIN_ENDPOINT.equals(request.getServletPath()) ||
            SecurityConfig.LOGOUT_ENDPOINT.equals(request.getServletPath())
||

SecurityConfig.REGISTRATION_ENDPOINT.equals(request.getServletPath()) ||

SecurityConfig.REFRESH_TOKEN_ENDPOINTS.equals(request.getServletPath()) ||

SecurityConfig.CONFIRM_ENDPOINTS.equals(request.getServletPath()) ||

SecurityConfig.RESTORE_ENDPOINTS.equals(request.getServletPath()) ||

SecurityConfig.RESET_PASSWORD_ENDPOINTS.equals(request.getServletPath())
    ) {
        filterChain.doFilter(request, response);
    } else {
        try {
            String accessToken = jwtProvider.resolveToken(request);
            if (accessToken == null) {
                throw new Exception("Access token is missing!");
            }
            DecodedJWT decodedJWT = jwtProvider.verifyToken(accessToken,
TokenType.ACCESS_TOKEN);
            Authentication authentication =
jwtProvider.getAuthentication(decodedJWT);

SecurityContextHolder.getContext().setAuthentication(authentication);
            filterChain.doFilter(request, response);
        } catch (Exception e) {
            LOGGER.error(e.getMessage());
```

```
            response.setStatus(HttpStatus.FORBIDDEN.value());
            Map<String, String> error = new HashMap<>();
            error.put("error_message", e.getMessage());
            response.setContentType(MediaType.APPLICATION_JSON_VALUE);
            new ObjectMapper().writeValue(response.getOutputStream(),
error);
        }
    }
```

В цьому методі відбувається JWT авторизація вхідного HTTP запиту.

AuthenticationFilter, це клас, який розширює абстрактний клас UsernamePasswordAuthenticationFilter та перевизначає два методи attemptAuthentication(), successfulAuthentication(). Цей фільтр застосовується до вхідних HTTP запитів, для яких кінцевою точкою є **/api/login**, тобто для обробки запитів на логінацію користувача.

Створення та налаштування об’єкту класу AuthenticationFilter, відбувається в методі authenticationFilter():

```
AuthenticationFilter authenticationFilter() throws Exception {
    AuthenticationFilter authenticationFilter = new
AuthenticationFilter(authenticationManagerBean(), jwtProvider);
    authenticationFilter.setFilterProcessesUrl(LOGIN_ENDPOINT);
    return authenticationFilter;
}
```

Метод attemptAuthentication(), який викликає фактичну автентифікацію користувача:

```
@Override
public Authentication attemptAuthentication(HttpServletRequest request,
HttpServletResponse response) throws AuthenticationException {
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    String mfaToken = request.getParameter("mfaToken");
    MFAUsernamePasswordAuthenticationToken authenticationToken = new
MFAUsernamePasswordAuthenticationToken(username, password, mfaToken);
    Authentication authentication =
```

```
authenticationManager.authenticate(authenticationToken);
    return authentication;
}
```

Метод successfulAuthentication(), який викликається при успішній автентифікації:

```
@Override
protected void successfulAuthentication(HttpServletRequest request,
HttpServletResponse response, FilterChain chain, Authentication
authentication) throws IOException, ServletException {
    User user = (User) authentication.getPrincipal();
    List<String> roles = new ArrayList<>();
    user.getAuthorities().forEach(grantedAuthority ->
roles.add(grantedAuthority.getAuthority()));
    String accessToken = jwtProvider.createAccessToken(user.getUsername(),
roles);
    String refreshToken =
jwtProvider.createRefreshToken(user.getUsername());

    Map<String, String> tokens = new HashMap<>();
    tokens.put("access_token", accessToken);
    tokens.put("refresh_token", refreshToken);

    response.setContentType(MediaType.APPLICATION_JSON_VALUE);
    new ObjectMapper().writeValue(response.getOutputStream(), tokens);
}
```

При успішній автентифікації, в цьому методі створюються два JWT токени, а саме access_token та refresh_token, які записуються в тіло відповіді на запит логінації та відправляються користувачу.

Кожен токен має свій час дії:

- **access_token**: 10 хв
- **refresh_token**: 12 год

Для оновлення токену access_token, необхідно відправити запит з кінцевою точкою **/api/refreshToken** та заголовком Authorization зі значенням Bearer {refresh_token}.

Запит буде прийнятий REST контролером MainController, а саме методом refreshToken() та переданий в клас AuthLibImpl, а саме в метод refreshToken() для подальшої валідації і створення нового токена accessToken.

Результатом виконання методу attemptAuthentication() буде виклик автентифікації користувача, ключовим моментом якого буде перевірка пароля користувача та тимчасового коду, якщо такий необхідний.

Перевірка тимчасового коду відбувається в класі MFAAuthenticationProvider, а саме в його методі additionalAuthenticationChecks():

```
@Override
protected void additionalAuthenticationChecks(UserDetails userDetails,
UsernamePasswordAuthenticationToken authentication) throws
AuthenticationException {
    MFAUser mfaUser = (MFAUser) userDetails;
    MFAUsernamePasswordAuthenticationToken authenticationToken =
(MFAUsernamePasswordAuthenticationToken) authentication;
    String mfaToken = authenticationToken.getMfaToken();
    if (mfaUser.isMfaActive()) {
        if (!mfaTokenProvider.verifyTotp(mfaToken, mfaUser.getSecretKey()))
{
            ReserveToken selectedReserveToken = mfaUser
                    .getReserveTokenList().stream()
                    .filter(reserveToken ->
reserveToken.getValue().equals(mfaToken))
                    .findFirst().orElse(null);
            if (selectedReserveToken != null) {

reserveTokenService.deleteById(selectedReserveToken.getId());
            } else {
                throw new BadCredentialsException(
                        messages.getMessage(

"AbstractUserDetailsAuthenticationProvider.badCredentials",
                                "Bad credentials"
                        )
                );
            }
        }
    } else {
        if (StringUtils.hasText(mfaToken)) {
            throw new BadCredentialsException(
                    messages.getMessage(
```

```
"AbstractUserDetailsAuthenticationProvider.badCredentials",
                            "Bad credentials"
                    )
            );
        }
    }
    super.additionalAuthenticationChecks(userDetails, authentication);
}
```

Основне завдання цього методу, це за потреби перевірити наявність і правильність тимчасового коду надісланого користувачем. У випадку якщо тимчасовий код є хибний, то автентифікація користувача не відбувається і повертається HTTP відповідь 403 (Forbidden).

Для перевірки правильності тимчасового коду використовується клас DefaultMFATokenProvider який використовує класи бібліотеки Totp. Дана бібліотека надає весь необхідний функціонал для генерування та перевірки тимчасових паролів створених за допомогою TOTP-алгоритму.

Користувач має можливість увімкнути двофакторну автентифікацію під час реєстрації облікового запису, або в любий інший момент після реєстрації через сторінку “Налаштування”, також при потребі на тій самі сторінці її можна і вимкнути.

3.1.3. RESTful вебсервіс

Під час розробки веб застосунку, була використана REST архітектура, тобто сервер та клієнт розроблялися окремо, а взаємодія між ними відбувається завдяки використанню протоколу HTTP та REST API.

Для обробки вхідних REST запитів, були реалізовані 3 контроллери: AuthController, MediaController, UserController. Кожен з них використовує анотацію @RestController. В цих класах створюються кінцеві точки, які можуть були використані клієнтом для відправки запитів на сервер. Анотація @RequestMapping("/api/auth") наприклад вказує, що до кінцевих точок на початок

добавляється частина **/api/auth**. Як результат всі кінцеві точки створені в цьому контролері будуть мати вигляд: **/api/auth/registration**, **/api/auth/confirm** і тд.

Реалізовані кінцеві точки в класі AuthController та опис їхнього призначення:

- **/api/auth/registration** – приймання REST запитів на реєстрацію нового користувача.
- **/api/auth/confirm** – приймання REST запитів на підтвердження email пошти для завершення реєстрації нового облікового запису.
- **/api/auth/restore** – приймання REST запитів на створення запиту для відновлення паролю до облікового запису користувача.
- **/api/auth/resetPassword** – приймання REST запитів на зміну паролю до облікового запису користувача.
- **/api/auth/activateMfa** – приймання REST запитів на активацію двофакторної автентифікації.
- **/api/auth/deactivateMfa** – приймання REST запитів на вимкнення двофакторної автентифікації.
- **/api/auth/deleteAccount** – приймання REST запитів на видалення облікового запису користувача.
- **/api/auth/refreshToken** – приймання REST запитів на оновлення JWT токену accessToken.

3.1.4. Бізнес логіка

Основна бізнес логіка обробки вхідних REST запитів реалізована у трьох класах: AuthLibImpl, MediaLibImpl та UserLibImpl.

Клас AuthLibImpl містить всю необхідну логіку для реєстрації нового користувача, обробки запитів на відновлення паролю, видалення облікового запису, активації/вимкнення двофакторної автентифікації та відновлення “refresh” JWT токену.

Клас UserLibImpl відповідає за обробку запитів на зміну налаштувань облікового запису.

Клас MediaLibImpl в свою чергу відповідає за обробку запитів на збереження медіа файлів, їхньої обробки, отримання, та видалення.

Демонстрація реалізованих методів activateMfa() та deactivateMfa() для активації та вимкнення двофакторної автентифікації:

Метод activateMfa():

```
@Override
public ActivateMfaResultDto activateMfa(String password) {
    org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    User userDbi = userService.findByUsername(user.getUsername());

    FormStatusDto formStatusDto = activateMfaValidator.validate(password,
userDbi);
    ActivateMfaResultDto activateMfaResultDto = new ActivateMfaResultDto();
    activateMfaResultDto.setFormStatusDto(formStatusDto);

    if (formStatusDto.getStatus().equals(StatusCode.FAILURE)) {
        return activateMfaResultDto;
    }

    try {
        String secretKey = mfaTokenProvider.generateSecretKey();
        List<String> reserveTokenList =
mfaTokenProvider.generateReserveTokenList();
        String qrCode = mfaTokenProvider.getQRCode(secretKey,
userDbi.getUsername());
```

```
        userDbi.setMfaActive(true);
        userDbi.setSecretKey(secretKey);
        activateMfaResultDto.setSecretKey(secretKey);
        activateMfaResultDto.setReserveTokenList(reserveTokenList);
        activateMfaResultDto.setQrCode(qrCode);

        userService.save(userDbi);

reserveTokenService.saveAll(DtoEntityMapper.mapToReserveTokenList(reserveTo
kenList, userDbi));
    } catch (Exception e) {
        LOGGER.error(e);
    }

    return activateMfaResultDto;
}
```

Для активації двофакторної автентифікації користувач повинен надіслати свій актуальний пароль. Після успішної перевірки паролю, відбувається створення секретного ключа, QR-коду, та 12 резервних токенів на випадок втрати секретного ключа. Згенероввані дані зберігаються в базу даних та відправляються у вигляді відповіді на HTTP запит.

Метод deactivateMfa():

```
@Override
public FormStatusDto deactivateMfa(DeactivateMfaRequestDto
deactivateMfaRequestDto) {
    org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    User userDbi = userService.findByUsername(user.getUsername());
    FormStatusDto formStatusDto =
deactivateMfaValidator.validate(deactivateMfaRequestDto, userDbi);
    if (formStatusDto.getStatus().equals(StatusCode.SUCCESS)) {
        userDbi.setMfaActive(false);
        userDbi.setSecretKey(null);
        userService.save(userDbi);
        reserveTokenService.deleteAllByUser(userDbi);
    }
    return formStatusDto;
}
```

Для вимкнення двофакторної автентифікації користувач повинен надіслати звичайний пароль та тимчасовий. Після успішної перевірки цих двох факторів, для користувача вимикається двофакторна автентифікації.

3.1.5. Взаємодія з базою даних

Найкращим рішенням для зберігання великих об'ємів даних є використання бази даних. Для цього проекту, було вирішено використати СКБД MySQL та фреймворк Spring Data JPA, який додає додатковий рівень абстракції поверх ORM реалізації JPA.

Для збереження всієї інформації, яка є необхідна для роботи вебзастосунку, було реалізовано 5 сутностей.

На рис. 3.1.5.1. зображений пакет "entity" у якому розташовані всі класи сутностей, які були реалізовані.

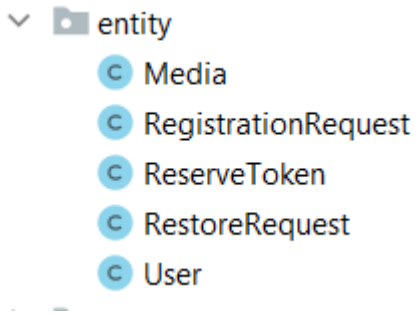

Рис. 3.1.5.1 – Пакет "entity" у якому розташовані всі класи сутностей, які були реалізовані.

Реалізовані сутності:

- Media – містить інформацію про медіа файли які були завантажені користувачами.
- RegistrationRequest – містить інформацію про запити на реєстрацію нового облікового запису.
- ReserveToken – містить інформацію про резервні токени користувачі.
- RestoreRequest – містить інформацію про запити на зміну паролю.
- User – містить інформацію про обліковий запис користувача.

Структура головної сутності User:

```
@Getter
@Setter
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(nullable = false)
    private boolean mfaActive;
    private String secretKey;
    @Column(nullable = false)
    private String firstName;
    @Column(nullable = false)
    private String lastName;
    @Column(nullable = false)
    private String email;
    @Column(nullable = false)
    private Timestamp registrationTime;

    @OneToMany(
            mappedBy = "user",
            fetch = FetchType.EAGER,
            orphanRemoval = true
```

```
    )
    private List<ReserveToken> reserveTokenList = new ArrayList<>();
```

Анотації @Getter, @Setter – це анотації сторонньої бібліотеки “Lombok”, які використовуються для зменшення коду в проекті. Вони заміняють явне прописування методів get та set для параметрів класу.

Анотація @Table – дозволяє вказати деталі таблиці, які будуть використані для зберігання сутності в базі даних.

Анотація @id – над полем id вказує на ідентифікатор цієї сутності.

Анотація @GenetatedValue – використовується разом з анотацією @Id та застосовується для вказівки стратегії генерації первинних ключів.

Анотація @Column – використовується для відображення полів сутності.

Анотація @OneToMany – вказує на відношення “один-до-багатьох” з сутністю ReserveToken.

Після першого запуску сервера, Spring Data JPA відповідно до реалізованих сутностей створює відповідні таблиці.

На рис. 3.1.5.2. продемонстровані поля створеної таблиці “media”.

Column Name	#	Data Type	Not Null	Auto Increment	Key
id	1	bigint	[v]	[v]	PRI
extension	2	varchar(255)	[v]	[ ]	
height	3	int	[v]	[ ]	
last_modified	4	datetime(6)	[v]	[ ]	
name	5	varchar(255)	[v]	[ ]	
user_id	6	bigint	[v]	[ ]	
width	7	int	[v]	[ ]	

Рис. 3.1.5.2. – Поля таблиці “media”.

На рис. 3.1.5.3. продемонстровані поля створеної таблиці “registration_requests”.

Column Name	#	Data Type	Not Null	Auto Increment	Key
id	1	bigint	[v]	[v]	PRI
confirm_key	2	varchar(255)	[v]	[ ]	
creation_time	3	datetime(6)	[v]	[ ]	
email	4	varchar(255)	[v]	[ ]	
first_name	5	varchar(255)	[v]	[ ]	
last_name	6	varchar(255)	[v]	[ ]	
mfa_active	7	bit(1)	[v]	[ ]	
password	8	varchar(255)	[v]	[ ]	
reserve_tokens	9	varchar(255)	[ ]	[ ]	
secret_key	10	varchar(255)	[ ]	[ ]	
username	11	varchar(255)	[v]	[ ]	

Рис. 3.1.5.3. – Поля таблиці “registration_requests”.

На рис. 3.1.5.4. продемонстровані поля створеної таблиці “reserve_tokens”.

Column Name	#	Data Type	Not Null	Auto Increment	Key
id	1	bigint	[v]	[v]	PRI
value	2	varchar(255)	[v]	[ ]	
user_id	3	bigint	[v]	[ ]	MUL

Рис. 3.1.5.4. Поля таблиці “reserve_tokens”.

На рис. 3.1.5.5. продемонстровані поля створеної таблиці “restore_requests”.

Column Name	#	Data Type	Not Null	Auto Increment	Key
id	1	bigint	[v]	[v]	PRI
creation_time	2	datetime(6)	[v]	[ ]	
restore_key	3	varchar(255)	[v]	[ ]	
user_id	4	bigint	[v]	[ ]	

Рис. 3.1.5.5. – Поля таблиці “restore_requests”.

На рис. 3.1.5.6. продемонстровані поля створеної таблиці “users”.

Column Name	#	Data Type	Not Null	Auto Increment	Key
id	1	bigint	[v]	[v]	PRI
email	2	varchar(255)	[v]	[ ]	
first_name	3	varchar(255)	[v]	[ ]	
last_name	4	varchar(255)	[v]	[ ]	
mfa_active	5	bit(1)	[v]	[ ]	
password	6	varchar(255)	[v]	[ ]	
registration_time	7	datetime(6)	[v]	[ ]	
secret_key	8	varchar(255)	[ ]	[ ]	
username	9	varchar(255)	[v]	[ ]	

Рис. 3.1.5.6. Поля таблиці “users”.

Для роботи з БД, а саме для побудови запитів до кожної з сутностей, були реалізовані репозиторії, які розширюють інтерфейс JpaRepository.

JpaRepository – це інтерфейс фреймворку Spring Data, що надає набір стандартних методів JPA для роботи з БД.

На рис. 3.1.5.7. зображений пакет “repositories” у якому розташовані всі інтерфейси репозиторії, які були реалізовані.

Рис. 3.1.5.7. – Пакет “repositories” у якому розташовані всі інтерфейси репозиторії, які були реалізовані.

3.2. Реалізація клієнтської частини

3.2.1. Структура проекту

Для написання клієнтської частини проекту був використаний фреймворк Angular. Початкова структура клієнтської частини проекту була створена за допомогою консольної команди наступної структури:

```
npx --package @angular/cli ng new media-storage-client --style=scss --skip-tests --routing
```

Ключові опції:

- **--package** – визначає пакет, який буде встановлено.
- **--style=scss** – вказує на використання .scss файлів стилів.
- **--routing** – вказує на необхідність створення модуля маршрутизації при генерації структури проекту.

На рис. 3.2.1.1. зображена кінцева структура клієнтської частини проекту після створення всіх необхідних компонентів, сервісів та перехоплювачів.

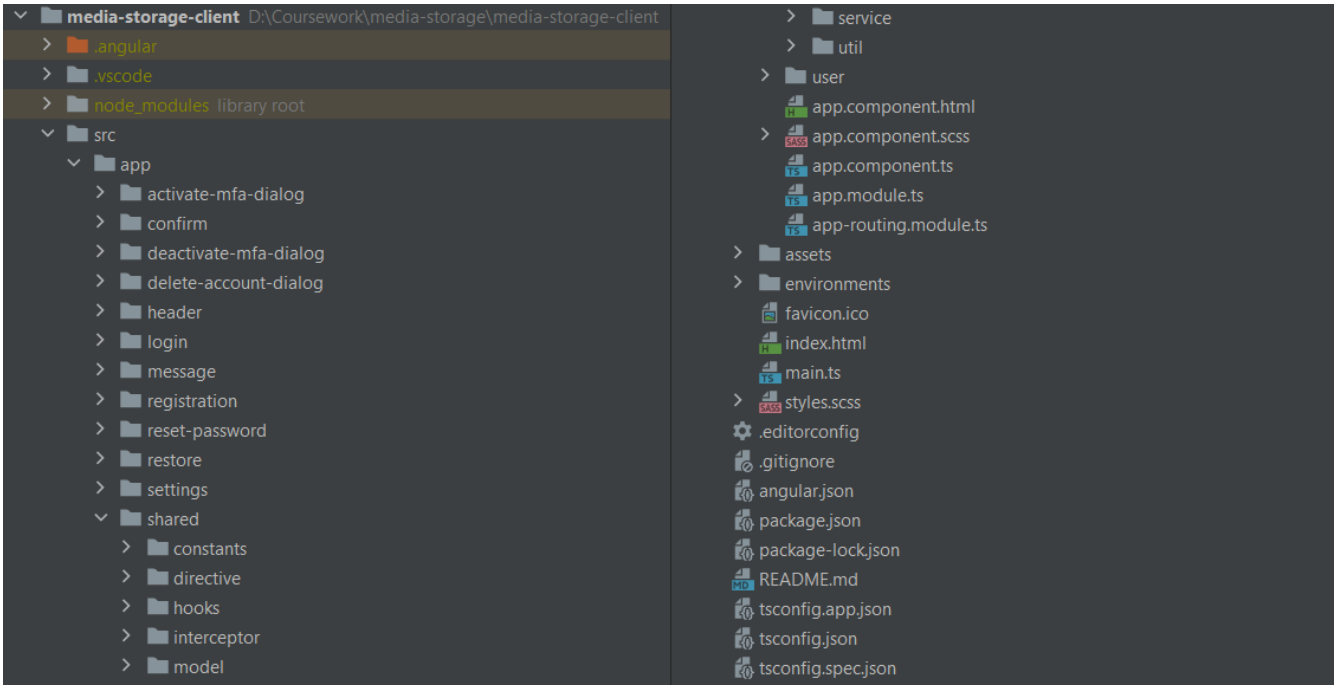

Рис. 3.2.1.1. – Кінцева структура клієнтської частини проекту, після створення всіх необхідних компонентів, сервісів та перехоплювачів

Пакет **src** – містить основний код клієнтської частини проекту.

Пакет **node_modules** – містить всі пакети, які використовуються для створення програми.

Файл **angular.json** – файл конфігурації, який містить всі налаштування нашого проекту, а також стилі та скрипти.

Файл **package.json** – файл конфігурації, який містить список пакетів, необхідних для проекту з потрібними версіями.

Файл **tsconfig.json** – файл конфігурації, в якому вказується, де будуть знаходитися файли програми та в яку директорію буде збиратися готова програма.

В пакеті **src.app.shared** розташовані спільні компоненти, серед яких класи констант, моделів, сервісів, перехоплювачів та утиліт.

3.2.2. Компоненти

Компоненти є основним будівельним блоком інтерфейсу програми Angular. Кожна програма Angular містить дерево компонентів.

Для реалізації клієнтської сторони соціальної мережі, було створено 12 компонентів:

- activate-mfa-dialog – відповідає за вигляд та функціонал діалогового вікна активації двофакторної автентифікації.
- confirm – відповідає за вигляд та функціонал сторінки підтвердження реєстрації облікового запису.
- deactivate-mfa-dialog – відповідає за вигляд та функціонал діалогового вікна вимкнення двофакторної автентифікації.
- delete-account-dialog – відповідає за вигляд та функціонал діалогового вікна підтвердження видалення облікового запису.
- header – відповідає за вигляд та функціонал шапки сайту.
- login – відповідає за вигляд та функціонал сторінки входу.

- message – відповідає за вигляд та функціонал вікна оголошення.
- registration – відповідає за вигляд та функціонал сторінки “Реєстрація”
- reset-password – відповідає за вигляд та функціонал сторінки зміни паролю.
- restore – відповідає за вигляд та функціонал сторінки строення запиту на зміну паролю.
- settings – відповідає за вигляд та функціонал сторінки “Налаштування”.
- user – відповідає за вигляд та функціонал головної сторінки користувача а саме галереї.

Кожен компонент складається з 3 файлів:

- {name}.component.html – містить структуру сторінки створену з використанням мови гіпертекстової розмітки HTML.
- {name}.component.scss – містить опис стилів.
- {name}.component.ts – містить опис функціоналу.

3.2.3. Маршрутизація

У Angular додатку, маршрутизація, це перехід від одного уявлення (шаблону) до іншого, в залежності від заданого URL. Навігація в додатках Angular відбувається без перезавантаження сторінки. За організацію маршрутизації в Angular відповідає модуль RouterModule бібліотеки @angular/router.

Для налаштування маршрутизації, при генерації структури проекту, була використана опція **--routing**, завдяки якій був створений модуль AppRoutingModule. Цей модуль призначений для налаштування маршрутизації та імпортується кореневим модулем AppModule.

Модуль AppRoutingModule:

```
const routes: Routes = [
  {path: '', component: LoginComponent},
  {path: 'registration', component: RegistrationComponent},
  {path: 'confirm/:key', component: ConfirmComponent},
```

```
  {path: 'restore', component: RestoreComponent},
  {path: 'resetPassword/:key', component: ResetPasswordComponent},
  {path: 'users/:username', component: UserComponent},
  {path: 'settings', component: SettingsComponent},
  {path: 'message', component: MessageComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Поле routes містить в собі список маршрутів, який вказує маршрутизатору, який вид відображати при натисканні на посилання або копіювання URL-адреси в адресному рядку браузера.

Для навігації при натисканні на посилання використовується директива routerLink:

```
<a [routerLink]="['/users', username]"> </a>
```

Іншим способом навігації, є використання сервісу Router, який забезпечує навігацію між представленнями та можливість маніпулювання URL-адресами.

Приклад використання сервісу Router в компоненті LoginComponent:

```
async ngOnInit() {
  if (this.authService.authCredentials) {
    await this.router.navigate([`users/${this.authService.getUsername()}`])
    return
  }
}
```

Якщо в сервісі AuthService є присутні автентифікаційні дані (користувач зайшов у свій обліковий запис), то відбувається перенаправлення на головну сторінку облікового запису користувача.

3.2.4. HTTP-перехоплювач

Більшість запитів, які відправляються на серверну частину додатку, потребують вказаного заголовку авторизації. Також, необхідна обробка невдалих запитів, через

завершення терміну дії токену авторизації access_token. Для того, щоб не проводити ці операції окремо для кожного запиту, був реалізований HTTP-перехоплювач AuthInterceptor.

HTTP-перехоплювач AuthInterceptor:

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(
    private authService: AuthService
  ) {
  }

  intercept(request: HttpRequest<unknown>, next: HttpHandler):
Observable<HttpEvent<unknown>> {
    let pathname = new URL(request.url).pathname
    if (!pathname.endsWith(EndpointConstants.LOGIN_ENDPOINT) &&
      !pathname.endsWith(EndpointConstants.REGISTRATION_ENDPOINT) &&
      !pathname.endsWith(EndpointConstants.REFRESH_TOKEN_ENDPOINT) &&
      !pathname.endsWith(EndpointConstants.RESTORE_ENDPOINT)
    ) {
      request = this.authService.addAuthHeader(request)
      return next.handle(request).pipe(
        catchError(error => {
          if (error.status == HttpStatusCode.Forbidden) {
            return this.authService.refreshToken().pipe(
              switchMap(() => {
                request = this.authService.addAuthHeader(request)
                return next.handle(request)
              }),
              catchError(() => {
                this.authService.logout()
                throw error
              })
            )
          } else {
            throw error
          }
        })
      )
    } else {
      return next.handle(request)
    }
  }}
```

Його головний метод **intercept()** обробляє всі перехоплені HTTP запити та у випадку, якщо кінцева точна запиту не дорівнює **/api/login**, **/api/registration**, **/api/refreshToken** або **/api/restore**, то виконується ряд операцій. Добавляється заголовок авторизації. Виконується перевірка статусу відповіді. Якщо статус відповіді 403 (час дії токену access_token минув), то відправляється запит refreshToken, якщо він успішний, то до попереднього запиту додається оновлений заголовок авторизації та виконується повторне відправлення. У випадку якщо запит refreshToken також не був успішний (наприклад час дії токену refresh_token також минув), то викликається метод logout() сервісу AuthService який виконує наступну послідовність дій:

- Видалення токенів access_token та refresh_token з локального сховища.
- Переадресація на сторінку входу.

3.3. Опис реалізованого функціоналу

3.3.1. Вхід в обліковий запис

Якщо користувач не зайшов у свій обліковий запис, то при вході на сайт він побачить сторінку входу. Ця сторінка надає можливість увійти в обліковий запис ввівши свій username, пароль а також опціонально тимчасовий код. За потреби є можливість перейти на сторінку “Реєстрація” або відновити пароль у випадку втрати.

Вигляд сторінки “Вхід” продемонстровано на рис. 3.3.1.1.

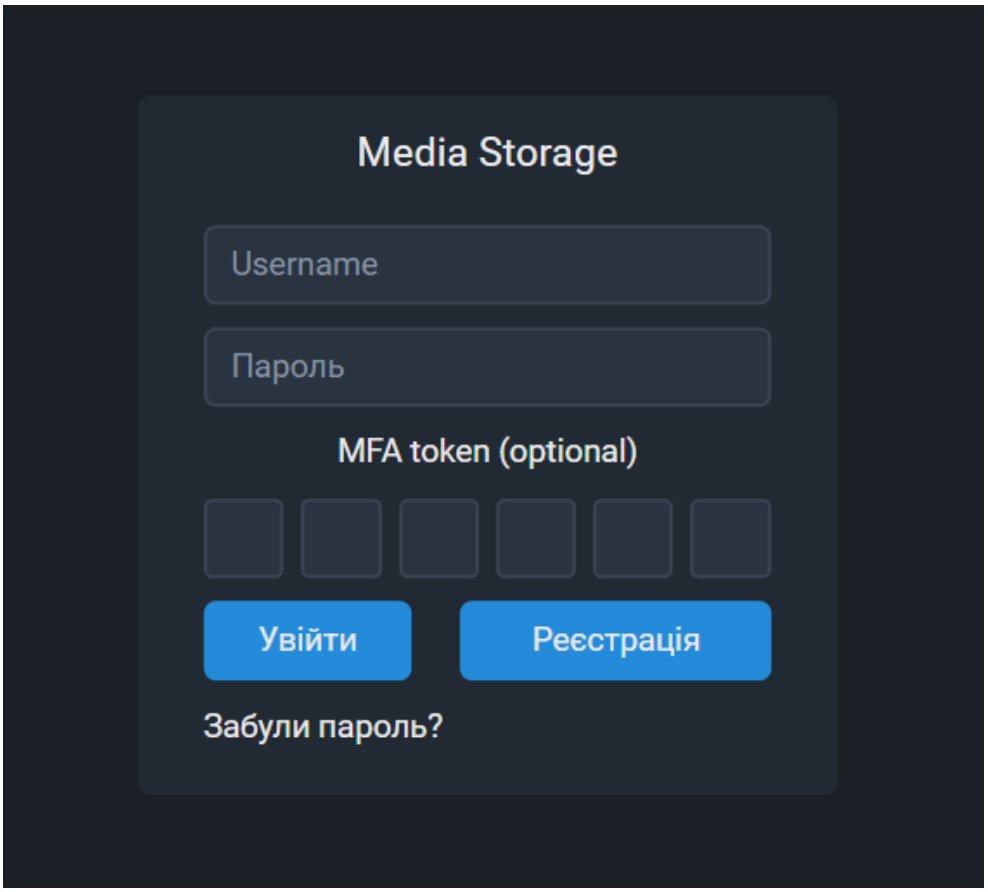

Рис. 3.1.1.1. – Вигляд сторінки “Вхід”.

3.3.2. Реєстрація нового облікового запису

Щоб користувач мав можливість створити обліковий запис, була створена сторінка “Реєстрація”.

Ця сторінка передбачає заповнення форми, що містить наступні поля:

- Ім’я
- Прізвище
- Email
- Username
- Пароль

Перевірка введених даних відбувається в два етапи:

- Груба перевірка при введенні даних: перевірка довжини введених даних.
- Перевірка після відправлення форми на сервер: валідація даних за допомогою регулярних виразів, перевірка вільності email адреси та username, перевірка username на невідповідність зарезервованим значенням.

Вигляд сторінки “Реєстрація” продемонстровано на рис. 3.3.2.1.

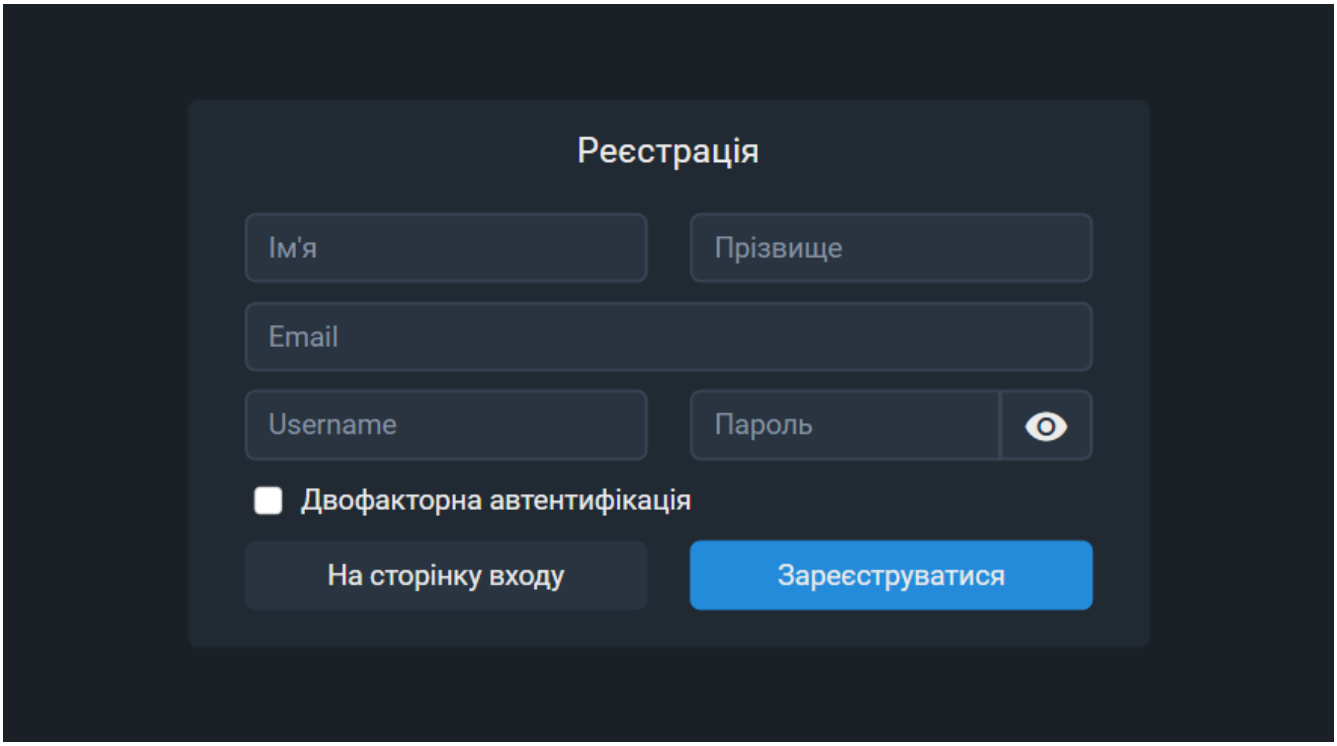

Рис. 3.3.2.1. Вигляд сторінки “Реєстрація”.

Вигляд сторінки “Реєстрація” з хибними значеннями полів “Email” та “Username” продемонстровано на рис. 3.3.2.2.

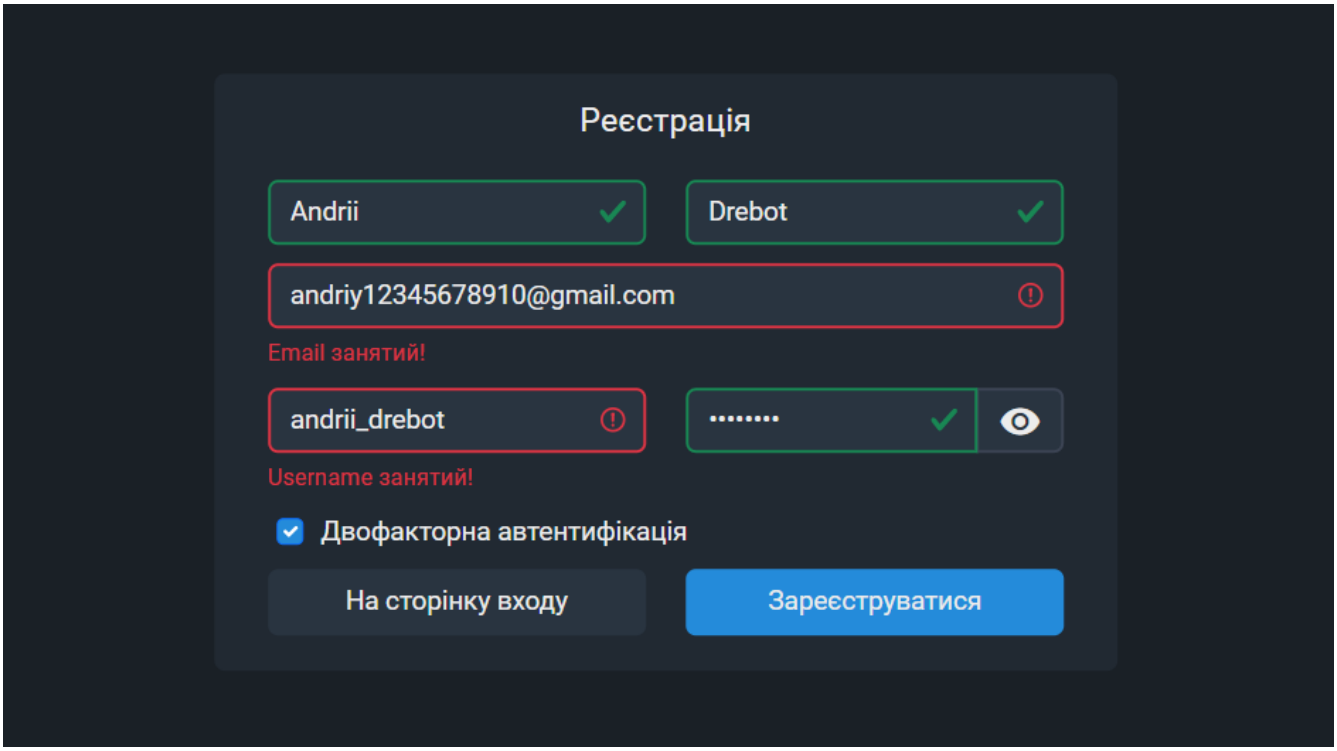

Рис. 3.3.2.2. – Вигляд сторінки “Реєстрація” з хибними значеннями полів “Email” та “Username”.

У випадку, якщо перевірка введених даних виконалась успішно, користувач побачить повідомлення з інформацією про наступні кроки для завершення реєстрації облікового запису. Якщо під час реєстрації була вибрана опція двофакторної автентифікації то повідомлення буде містити згенерований QR-код з секретним ключом, текстове представлення секретного ключа (для випадку коли зісканувати QR-код немає можливості), а також резервні одноразові токени на випадок втрати секретного ключа.

Дане повідомлення продемонстровано на рис. 3.3.2.3.

Рис. 3.3.2.3. – Вигляд повідомлення з інформацією про наступні кроки для завершення реєстрації облікового запису.

Після сканування QR-коду, додатком автентифікатором, додаток запамятає секретний ключ та розпочне генерацію тимчасових кодів двофакторної автентифікації на пристрої користувача.

На рис. 3.3.2.4. продемонстрований приклад генерації тимчасових кодів за допомогою розширення “Authenticator” для веббраузера “Google Chrome”

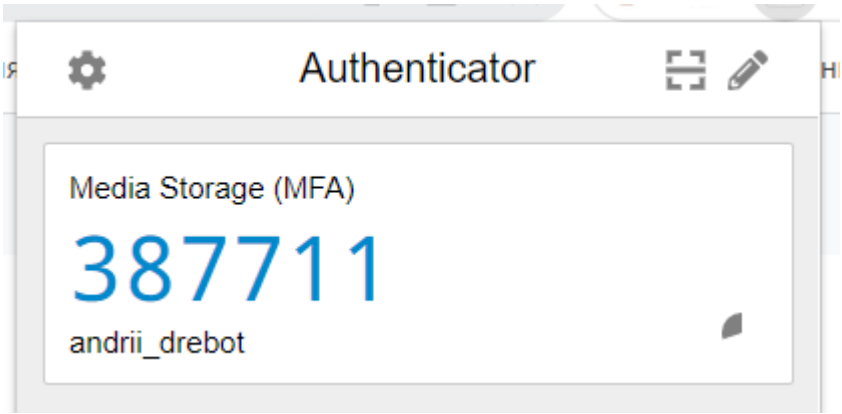

Рис. 3.3.2.4. – Приклад генерації тимчасових кодів за допомогою розширення “Authenticator” для веббраузера “Google Chrome”.

На рис. 3.3.2.5. продемонстрований електронний лист з посиланням для підтвердження реєстрації облікового запису.

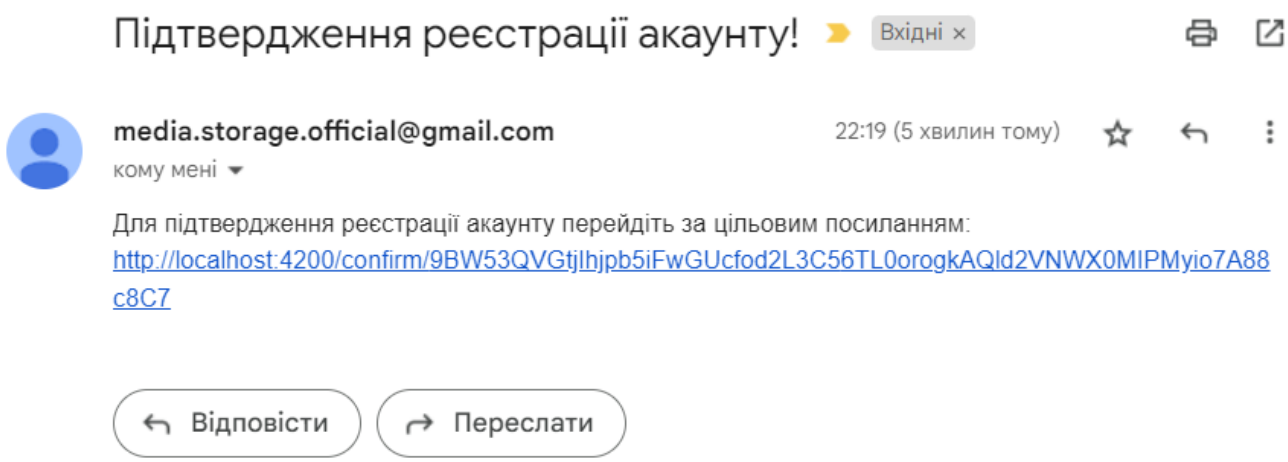

Рис. 3.3.2.5. – Вигляд електронного листа з посиланням для підтвердження реєстрації облікового запису.

Після переходу за посилання для підтвердження реєстрації облікового запису, користувач побачить повідомлення, яке продемонстровано на рис. 3.3.2.6.

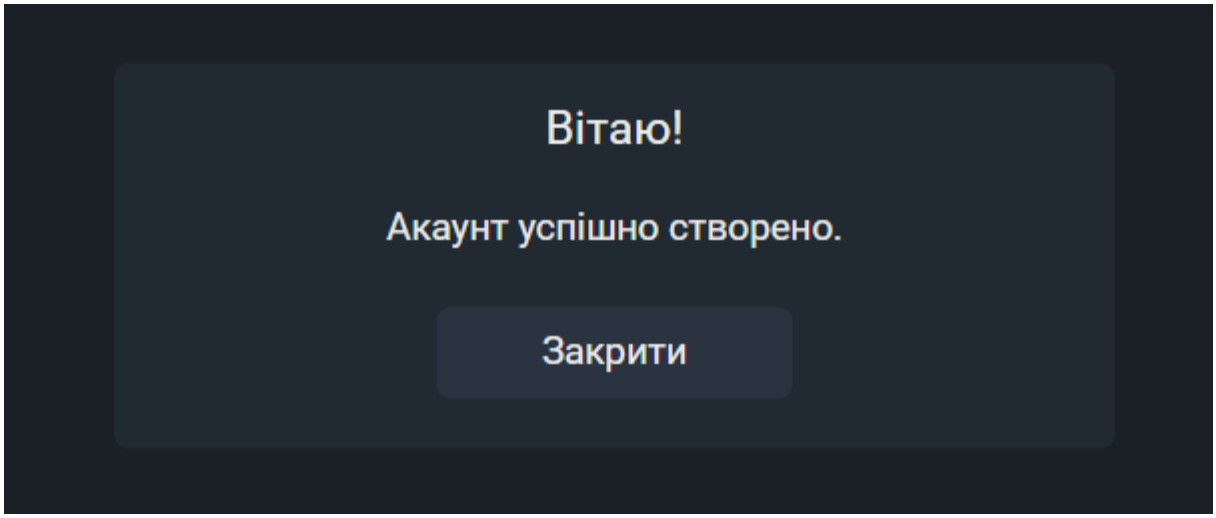

Рис. 3.3.2.6. – Вигляд повідомлення з інформацією про успішне створення облікового запису.

3.3.3. Зміна паролю

У випадку якщо пароль до облікового запису був втрачений, або користувач має бажання його змінити, йому необхідно перейти за посиланням “Забули пароль?” на сторінці входу.

Для зміни пароля необхідно вказати email адресу, яка була використана при реєстрації облікового запису.

Вигляд форми для введення email адреси для зміни пароля продемонстровано на рис. 3.3.3.1.

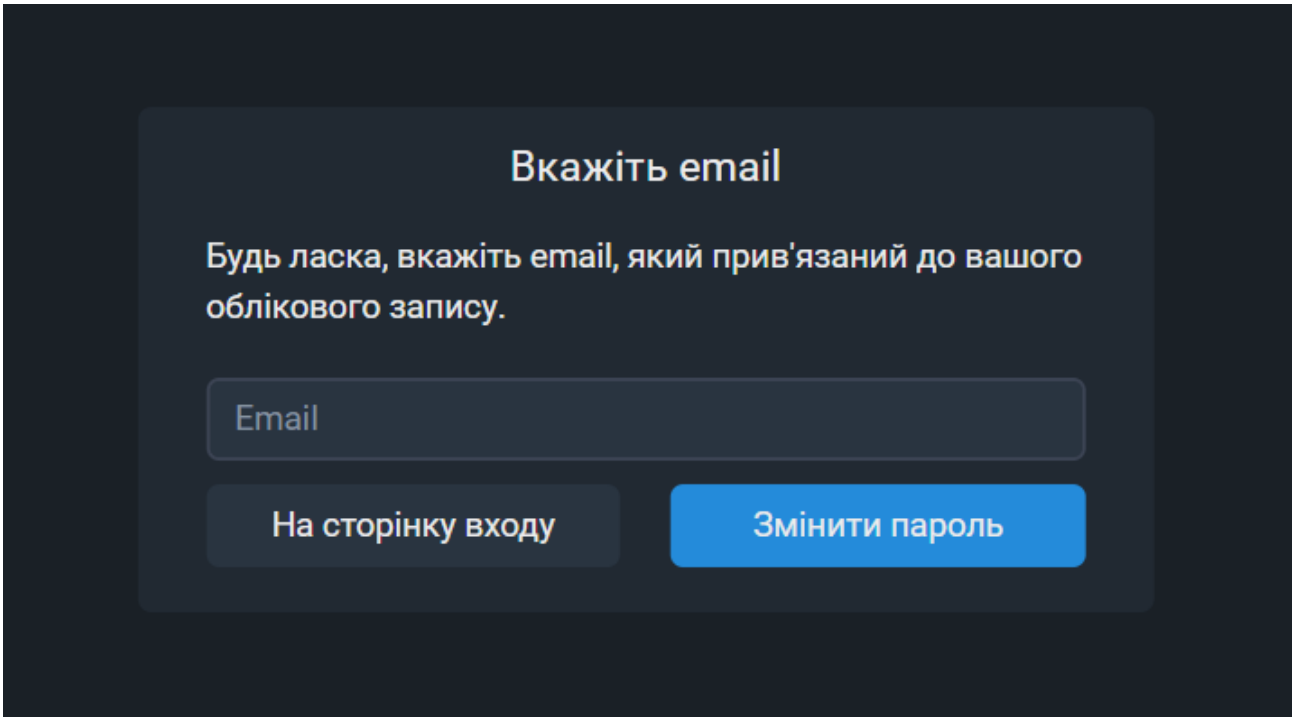

Рис. 3.3.3.1. – Вигляд форми для введення email адреси для зміни паролю.

Після відправлення форми зі вказаною email адресою, користувач побачить повідомлення, яке продемонстровано на рис. 3.3.3.2.

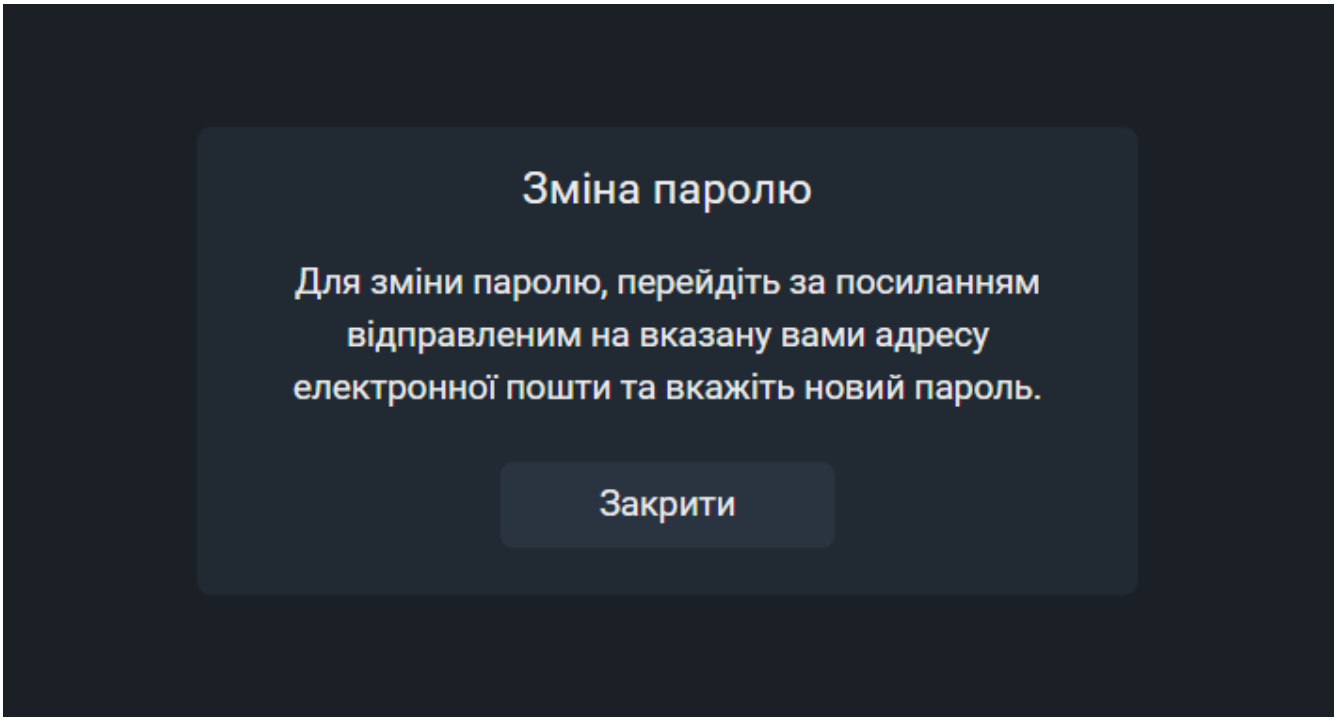

Рис. 3.3.3.2. – Вигляд повідомлення з інформацією про наступний крок для зміни паролю.

На рис. 3.3.3.3. продемонстрований електронний лист з посиланням на форму для зміни паролю.

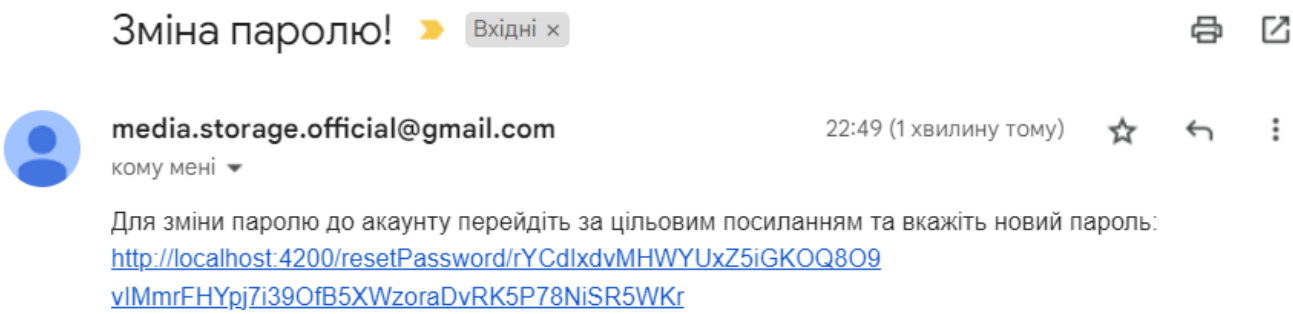

Рис. 3.3.3.3. – Вигляд електронного листа з посиланням на форму для зміни паролю.

Після переходу за цільовим посиланням, користувачу буде доступна форма для зміни паролю, яка продемонстрована на рис. 3.3.3.4.

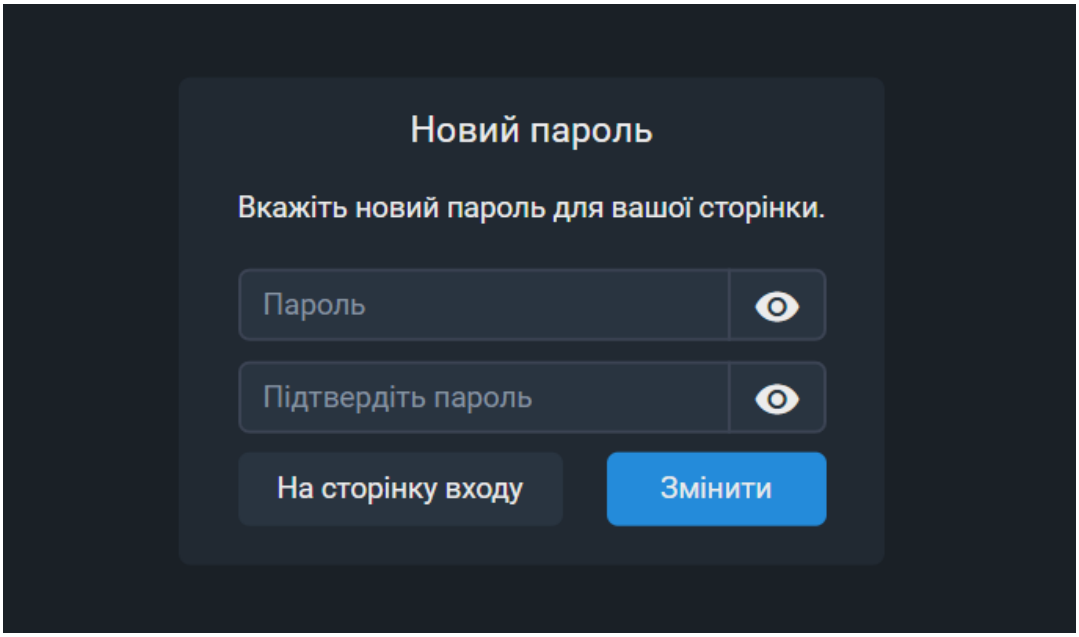

Рис. 3.3.3.4. – Вигляд форми для зміни паролю.

3.3.4. Галерея

Галерея, це основна сторінка вебзастосунку. На ній користувач має можливість переглянути збережені медіа файли, добавити нові, видалити, або завантажити їх на власний пристрій.

Вигляд сторінки “Галерея”, продемонстровано на рис. 3.3.4.1.

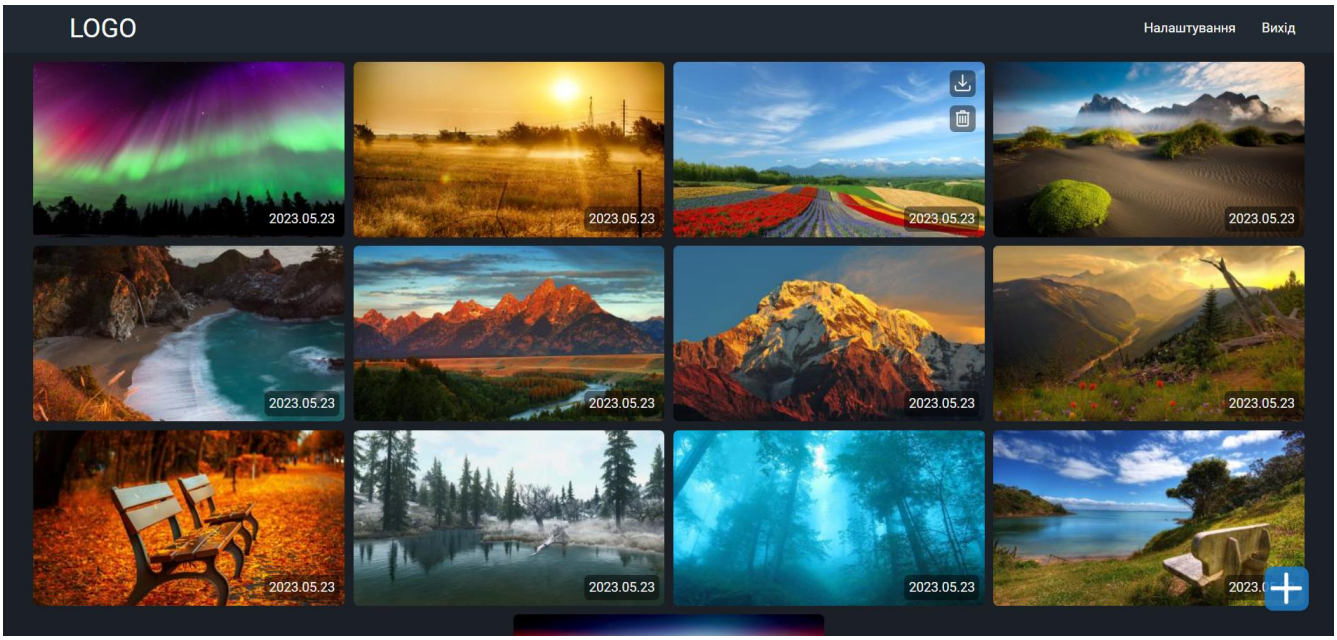

Рис. 3.3.4.1. – Вигляд сторінки “Галерея”

Вигляд вікна для перегляду зображення у збільшеному форматі, продемонстровано на рис. 3.3.4.2.

Рис. 3.3.4.2. – Вигляд вікна для перегляду зображення у збільшеному форматі.

3.3.5. Сторінка "Налаштування"

Сторінка "Налаштування", призначена для налаштування облікового запису.

Ця сторінка містить 2 вкладки:

- Вкладка “Загальне”: дозволяє змінити основну інформацію облікового запису, таку як: ім’я, прізвище та username.
- Вкладка “Безпека”: містить дві кнопки, а саме кнопку активації/вимкнення двофакторної автентифікації, та кнопку “Видалити обліковий запис”, яка дозволяє видалити обліковий запис.

Перевірка введених даних для кожного поля відбувається в два етапи:

- Груба перевірка при введенні даних: перевірка довжини введених даних та перевірка дати народження.

- Перевірка після відправлення форми на сервер: валідація даних за допомогою регулярних виразів, перевірка username на вільність та доступність.

На рис. 3.3.5.1. продемонстрований вигляд вкладки “Загальне” на сторінці “Налаштування”.

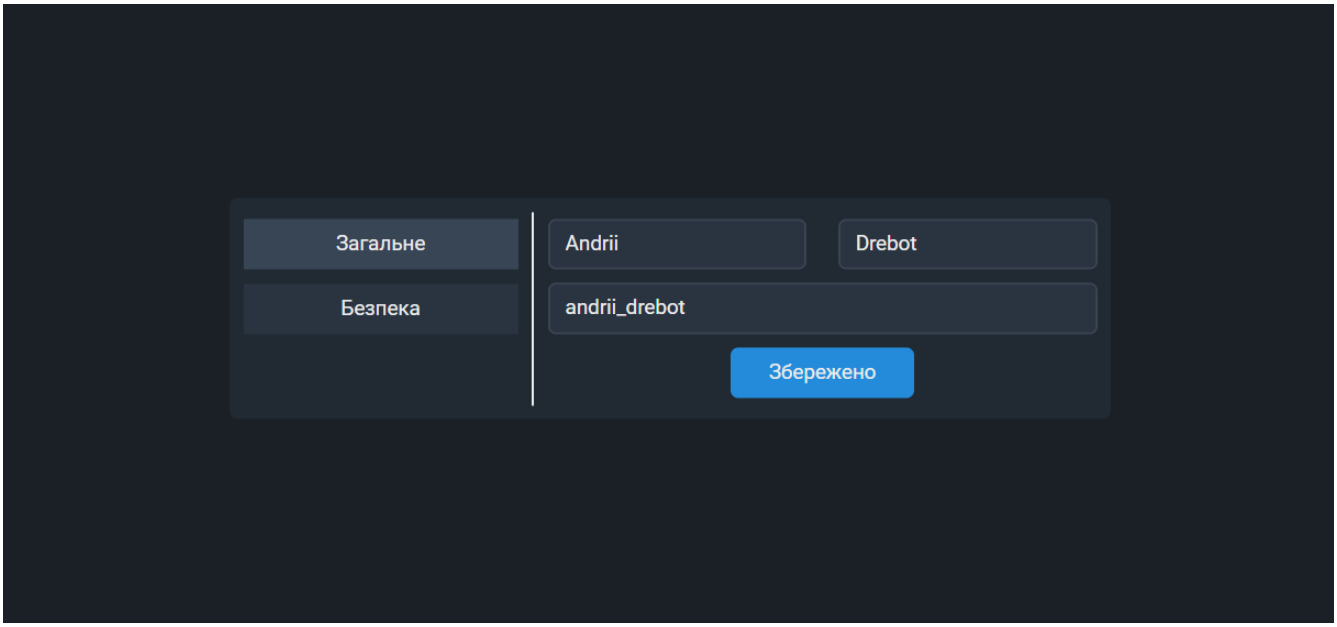

Рис. 3.3.5.1. – Вигляд вкладки “Загальне” на сторінці “Налаштування”.

На рис. 3.3.5.2. продемонстрований вигляд вкладки “Безпека” на сторінці “Налаштування”.

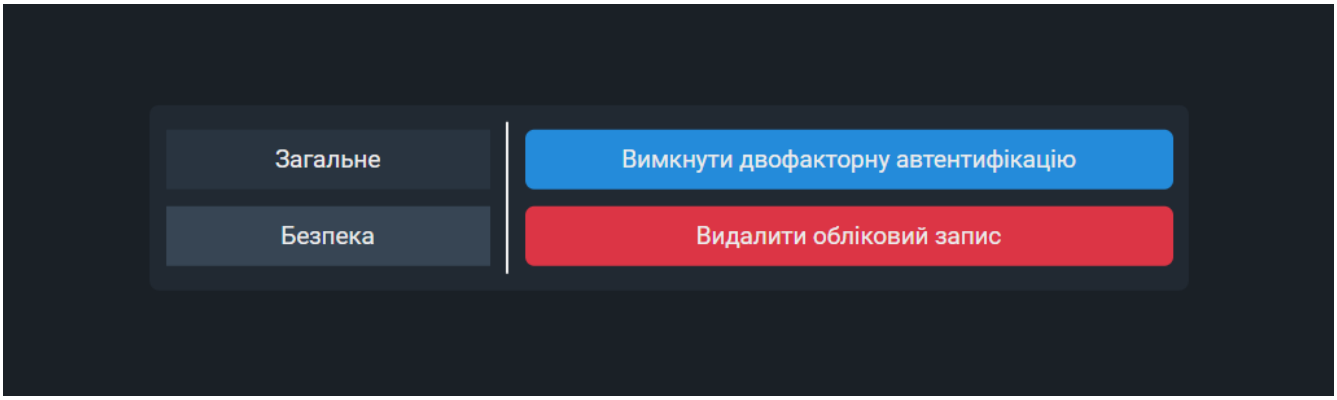

Рис. 3.3.5.2. – Вигляд вкладки “Безпека” на сторінці “Налаштування”.

На рис. 3.3.5.3. продемонстровано вигляд вікна вимкнення двофакторної автентифікації.

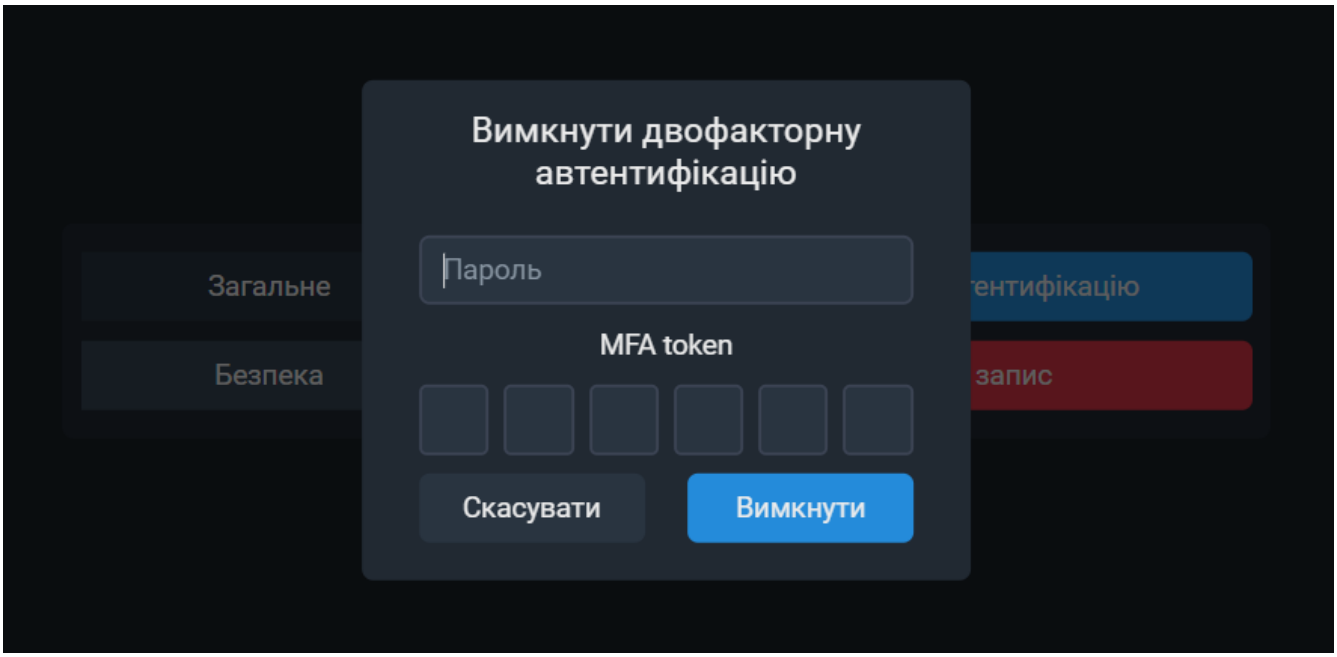

Рис. 3.3.5.3. – Вигляд вікна вимкнення двофакторної автентифікації.

На рис. 3.3.5.4. продемонстровано вигляд вікна активації двофакторної автентифікації.

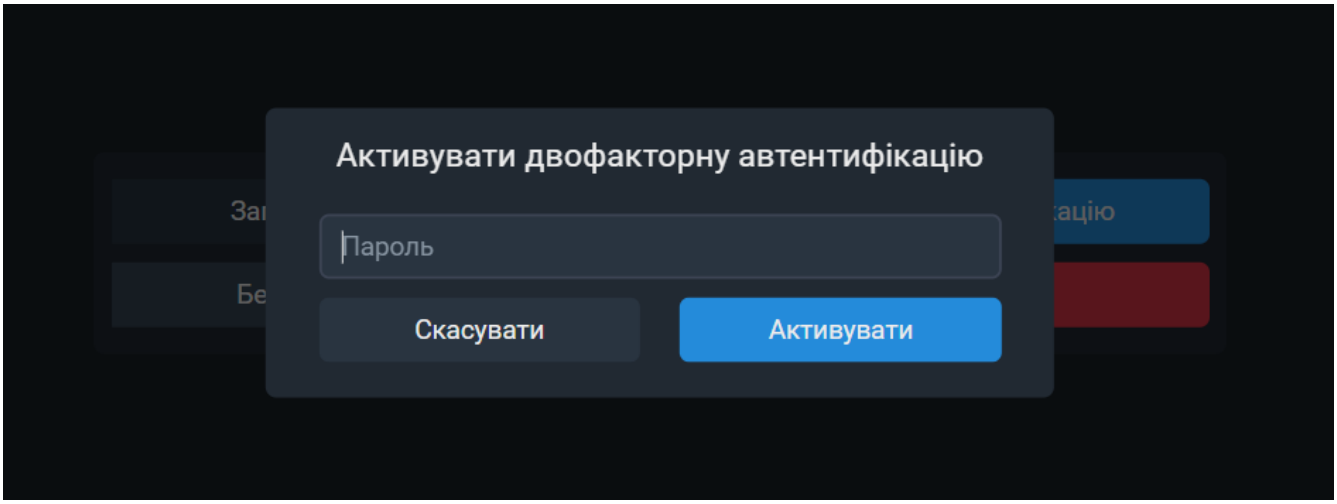

Рис. 3.3.5.4. – Вигляд вікна активації двофакторної автентифікації.

На рис. 3.3.5.5. продемонстровано вигляд вікна активації двофакторної автентифікації після успішного введення паролю облікового запису.

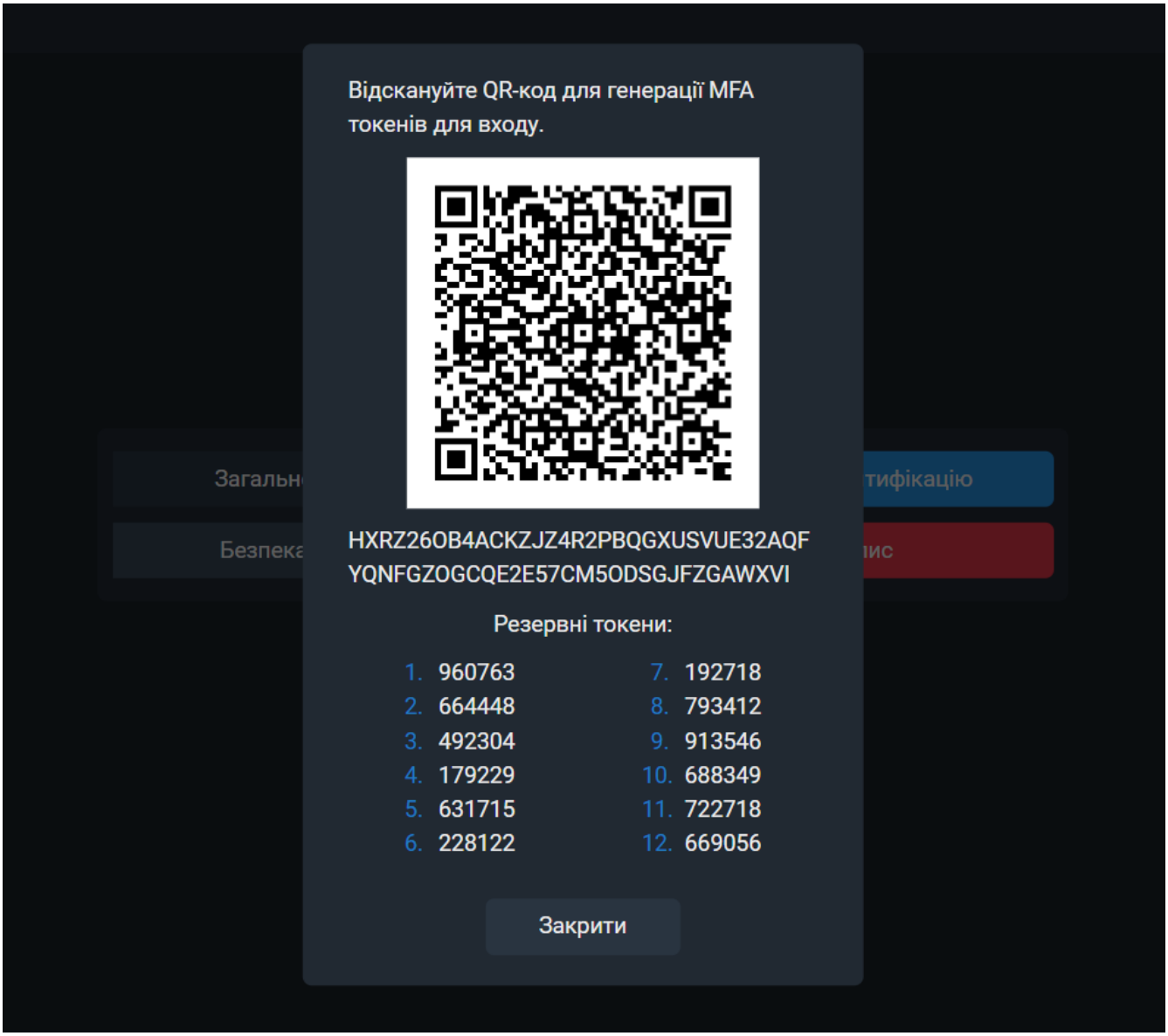

Рис. 3.3.5.5. – Вигляд вікна активації двофакторної автентифікації після успішного введення паролю облікового запису.

На рис. 3.3.5.6. продемонстроване вікно підтвердження видалення облікового запису користувача. У випадку, якщо активована двофакторна автентифікація, то для видалення облікового запису, окрім паролю, також необхідно буде ввести тимчасовий код.

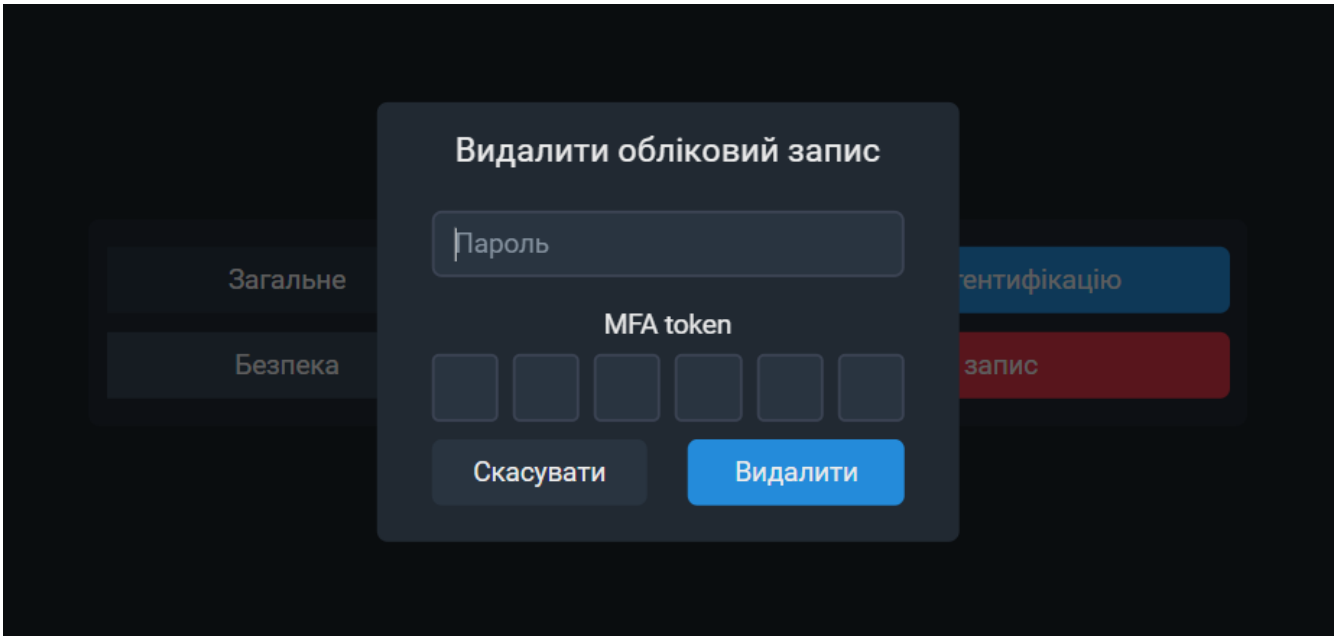

Рис. 3.3.5.6. – Вікно підтвердження видалення облікового запису користувача.

ВИСНОВОК

У процесі виконання курсової роботи було досліджено різні методи автентифікації користувачів у вебзастосунках з фокусом на їх популярність, переваги, недоліки і складність реалізації та використання.

Результати дослідження дозволили краще зрозуміти сучасні методи автентифікації користувачів у вебзастосунках, а також розробити власний вебзастосунок “Media Storage”, в якому було впроваджено один з найпопулярніших методів автентифікації користувачів, а саме “Багатофакторна автентифікація” який вимагає підтвердження ідентичності користувача через два різні фактори: звичайний пароль та тимчасовий код створений TOTP-алгоритмом.

Під час розробки проекту, була використана REST архітектура, тобто сервер та клієнт розроблялися окремо, а взаємодія між ними відбувається завдяки використанню протоколу HTTP та REST API.

Серверна частина проекту була написана на мові програмування Java з використанням фреймворку Spring. Для збереження інформації використовувалась СКБД MySQL. Для автоматизації роботи з програмним проектом використовувався фреймворк “Apache Maven”.

Клієнтська частина проекту була написана на мові програмування TypeScript з використанням фреймворку Angular. Для збірки проекту використовувався менеджер пакунків npm.

Серед основного функціоналу розробленого вебзастосунку варто відмітити такі функції як: двофакторна автентифікація, реєстрація користувача з підтвердженням через email пошту, відновлення паролю, налаштування особистої інформації користувача, видалення облікового запису та можливість зберігати, переглядати, завантажувати та видаляти медіа файли.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. "What is Password-Based Authentication?" [Електронний ресурс] – Режим доступу до ресурсу: https://www.descope.com/learn/post/password-authentication
2. "What is Username and Password Authentication?" [Електронний ресурс] – Режим доступу до ресурсу: https://www.logintc.com/types-of-authentication/password-authentication/
3. "Biometric Authentication" [Електронний ресурс] – Режим доступу до ресурсу: https://www.logintc.com/types-of-authentication/biometric-authentication/
4. "What Is Biometric Authentication? A Complete Overview" [Електронний ресурс] – Режим доступу до ресурсу: https://heimdalsecurity.com/blog/biometric-authentication/
5. "Is location an Authentication Factor?" [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/os-techblog/is-location-an-authentication-factor-9ed33d633993
6. "Certificate-based Authentication" [Електронний ресурс] – Режим доступу до ресурсу: https://www.pingidentity.com/en/resources/identity-fundamentals/authentication/certificate-authentication.html
7. "What is Certificate-Based Authentication?" [Електронний ресурс] – Режим доступу до ресурсу: https://www.yubico.com/resources/glossary/what-is-certificate-based-authentication/
8. "What is Multi-Factor Authentication (MFA)?" [Електронний ресурс] – Режим доступу до ресурсу: https://www.loginradius.com/blog/identity/what-is-multi-factor-authentication/
9. "Two-factor authentication: What you need to know (FAQ)" [Електронний ресурс] – Режим доступу до ресурсу: https://www.cnet.com/news/privacy/two-factor-authentication-what-you-need-to-know-faq/

10."Costs and benefits of authentication" [Електронний ресурс] – Режим доступу до ресурсу: https://www.miteksystems.com/blog/costs-and-benefits-of-authentication
11."5 User Authentication Methods that Can Prevent the Next Breach" [Електронний ресурс] – Режим доступу до ресурсу: https://www.idrnd.ai/5-authentication-methods-that-can-prevent-the-next-breach/
12."Top Websites Ranking" [Електронний ресурс] – Режим доступу до ресурсу: https://www.similarweb.com/top-websites/
13."TOTP (Time-based one-time Password algorithm)" [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/articles/534064/
14."Introduction to Apache Maven" [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-apache-maven-buildautomation-tool-java-projects/
15."Java (programming language)" [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Java_(programming_language)
16."Introduction to Java" [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/introduction-to-java/
17."Spring Framework Overview" [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/lifeinhurry/spring-framework-overview927f39acc1b1
18."Spring Framework - Overview" [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/spring/spring_overview.htm
19."Overview of Spring Framework" [Електронний ресурс] – Режим доступу до ресурсу: https://docs.spring.io/spring-framework/docs/4.1.x/spring-frameworkreference/html/overview.html
20."Spring Boot - Introduction" [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

21."Introduction to Spring Boot" [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/adessoturkey/introduction-to-spring-boot458cb814ec14
22."About PostgreSQL" [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/about/
23."What is PostgreSQL" [Електронний ресурс] – Режим доступу до ресурсу: https://aiven.io/blog/an-introduction-to-postgresql
24."Introduction to JSON Web Tokens" [Електронний ресурс] – Режим доступу до ресурсу: https://jwt.io/introduction
25."JWT: The Complete Guide to JSON Web Tokens" [Електронний ресурс] – Режим доступу до ресурсу: https://blog.angular-university.io/angular-jwt/
26."TypeScript Overview" [Електронний ресурс] – Режим доступу до ресурсу: https://www.tutorialsteacher.com/typescript/typescript-overview
27."Introduction to Angular concepts" [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/guide/architecture
28."What is Angular?: Architecture, Features, and Advantages" [Електронний ресурс] – Режим доступу до ресурсу: https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular
29."An overview of Angular" [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@mlbors/an-overview-of-angular-3ccd2950648e

ДОДАТКИ

Додаток 1

Файли серверної частини проекту:

Файл .gitignore:

```
HELP.md
target/
!.mvn/wrapper/maven-wrapper.jar
!**/src/main/**/target/
!**/src/test/**/target/

### STS ###
.apt_generated
.classpath
.factorypath
.project
.settings
.springBeans
.sts4-cache

### IntelliJ IDEA ###
.idea
*.iws
*.iml
*.ipr

### NetBeans ###
/nbproject/private/
/nbbuild/
/dist/
/nbdist/
/.nb-gradle/
build/
!**/src/main/**/build/
!**/src/test/**/build/

### VS Code ###
.vscode/
```

Файл pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.10</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
```

```
<groupId>com.andrew.mediastorage</groupId>
<artifactId>media-storage-server</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>Media Storage Server</name>
<description>Media Storage Server</description>
<properties>
    <java.version>1.8</java.version>
    <version.commons-io.commons-io>2.11.0</version.commons-io.commons-io>
    <version.com.auth0.java-jwt>4.4.0</version.com.auth0.java-jwt>
    <version.dev.samstevens.totp.totp>1.7.1</version.dev.samstevens.totp.totp>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
        <exclusions>
            <exclusion>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-logging</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-mail</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>${version.commons-io.commons-io}</version>
    </dependency>
    <dependency>
        <groupId>com.auth0</groupId>
        <artifactId>java-jwt</artifactId>
        <version>${version.com.auth0.java-jwt}</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/dev.samstevens.totp/totp -->
    <dependency>
        <groupId>dev.samstevens.totp</groupId>
        <artifactId>totp</artifactId>
        <version>${version.dev.samstevens.totp.totp}</version>
    </dependency>
```

```
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
            <scope>provided</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>

</project>
```

Файл FormField.java:

```
package com.andrew.mediastorage.server.api.constants;

public enum FormField {
    FIRST_NAME,
    LAST_NAME,
    EMAIL,
    USERNAME,
    PASSWORD,
    MFA_TOKEN,
    RESTORE_KEY,
    ALL_FIELDS
```

```
}
```

Файл JwtConstants.java:

```
package com.andrew.mediastorage.server.api.constants;

public interface JwtConstants {
    String TOKEN_HEADER = "Bearer ";
    String TOKEN_TYPE = "token_type";

}
```

Файл StatusCode.java:

```
package com.andrew.mediastorage.server.api.constants;

public enum StatusCode {
    SUCCESS,
    FAILURE

}
```

Файл TokenType.java:

```
package com.andrew.mediastorage.server.api.constants;

public enum TokenType {
    ACCESS_TOKEN,
    REFRESH_TOKEN

}
```

Файл MFAUser.java:

```
package com.andrew.mediastorage.server.api.data;

import com.andrew.mediastorage.server.api.entity.ReserveToken;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.User;

import java.util.Collection;
import java.util.List;
import java.util.Objects;

public class MFAUser extends User {

    private boolean mfaActive;
    private String secretKey;
    private List<ReserveToken> reserveTokenList;
```

```
    public MFAUser(String username, String password, boolean mfaActive, String secretKey,
List<ReserveToken> reserveTokenList, Collection<? extends GrantedAuthority> authorities) {
        super(username, password, authorities);
        this.mfaActive = mfaActive;
        this.secretKey = secretKey;
        this.reserveTokenList = reserveTokenList;
    }

    public MFAUser(String username, String password, boolean enabled, boolean
accountNonExpired, boolean mfaActive, String secretKey, List<ReserveToken>
reserveTokenList, boolean credentialsNonExpired, boolean accountNonLocked, Collection<?
extends GrantedAuthority> authorities) {
        super(username, password, enabled, accountNonExpired, credentialsNonExpired,
accountNonLocked, authorities);
        this.mfaActive = mfaActive;
        this.secretKey = secretKey;
        this.reserveTokenList = reserveTokenList;
    }

    public boolean isMfaActive() {
        return mfaActive;
    }

    public String getSecretKey() {
        return secretKey;
    }

    public List<ReserveToken> getReserveTokenList() {
        return reserveTokenList;
    }

    @Override
    public int hashCode() {
        return Objects.hash(super.hashCode(), mfaActive, secretKey, reserveTokenList);
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(getClass().getSimpleName()).append(" [");
        sb.append(super.toString()).append(", ");
        sb.append("mfaActive=").append(mfaActive).append(", ");
        sb.append("secretKey=").append(secretKey).append(", ");
        sb.append("reserveTokenList=").append(reserveTokenList).append("]");
        return sb.toString();
    }
}
```

Файл MFAUsernamePasswordAuthenticationToken.java:

```
package com.andrew.mediastorage.server.api.data;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.GrantedAuthority;

import java.util.Collection;
```

```
import java.util.Objects;

public class MFAUsernamePasswordAuthenticationToken extends
UsernamePasswordAuthenticationToken {

    private String mfaToken;

    public MFAUsernamePasswordAuthenticationToken(Object principal, Object credentials,
String mfaToken) {
        super(principal, credentials);
        this.mfaToken = mfaToken;
    }

    public MFAUsernamePasswordAuthenticationToken(Object principal, Object credentials,
String mfaToken, Collection<? extends GrantedAuthority> authorities) {
        super(principal, credentials, authorities);
        this.mfaToken = mfaToken;
    }

    public String getMfaToken() {
        return mfaToken;
    }

    @Override
    public int hashCode() {
        return Objects.hash(super.hashCode(), mfaToken);
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(getClass().getSimpleName()).append(" [");
        sb.append(super.toString()).append(", ");
        sb.append("mfaToken=").append(mfaToken).append("]");
        return sb.toString();
    }
}
```

Файл ActivateMfaResultDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Getter
@Setter
public class ActivateMfaResultDto {

    private FormStatusDto formStatusDto;
    private String qrCode;
    private String secretKey;
    private List<String> reserveTokenList;
```

```
}
```

Файл BasicSettingsDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class BasicSettingsDto {

    private String firstName;
    private String lastName;
    private String username;

}
```

Файл DeactivateMfaRequestDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class DeactivateMfaRequestDto {
    private String password;
    private String mfaToken;

}
```

Файл DeleteAccountRequestDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class DeleteAccountRequestDto {

    private String password;
    private String mfaToken;

}
```

Файл FormStatusDto.java:

```
package com.andrew.mediastorage.server.api.dto;
```

```
import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import lombok.Getter;
import lombok.Setter;

import java.util.HashMap;
import java.util.Map;

@Getter
@Setter
public class FormStatusDto {

    private StatusCode status = StatusCode.FAILURE;
    private Map<FormField, String> invalidFieldsMap = new HashMap<>();

}
```

Файл MediaDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class MediaDto {

    private Long id;
    private String name;
    private String extension;
    private Integer width;
    private Integer height;
    private Integer previewWidth;
    private Integer previewHeight;
    private Long lastModified;

}
```

Файл RegistrationFormDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class RegistrationFormDto {

    private String firstName;
    private String lastName;
    private String email;
    private String username;
```

```
    private String password;
    private boolean useMfa;

}
```

Файл RegistrationResultDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

import java.util.List;

@Getter
@Setter
public class RegistrationResultDto {

    private FormStatusDto formStatusDto;
    private String qrCode;
    private String secretKey;
    private List<String> reserveTokenList;

}
```

Файл ResetPasswordRequestDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class ResetPasswordRequestDto {
    private String restoreKey;
    private String newPassword;

}
```

Файл SettingsDto.java:

```
package com.andrew.mediastorage.server.api.dto;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.Column;

@Getter
@Setter
public class SettingsDto {

    private String firstName;
```

```
    private String lastName;
    private String username;
    private boolean mfaActive;

}
```

Файл Media.java:

```
package com.andrew.mediastorage.server.api.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.sql.Timestamp;

@Getter
@Setter
@Entity
@Table(name = "media")
public class Media {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private Long userId;
    @Column(nullable = false)
    private String name;
    @Column(nullable = false)
    private String extension;
    @Column(nullable = false)
    private Integer width;
    @Column(nullable = false)
    private Integer height;
    @Column(nullable = false)
    private Timestamp lastModified;

}
```

Файл RegistrationRequest.java:

```
package com.andrew.mediastorage.server.api.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.sql.Timestamp;

@Getter
@Setter
@Entity
@Table(name = "registration_requests")
public class RegistrationRequest {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(nullable = false)
    private boolean mfaActive;
    private String secretKey;
    private String reserveTokens;
    @Column(nullable = false)
    private String firstName;
    @Column(nullable = false)
    private String lastName;
    @Column(nullable = false)
    private String email;
    @Column(nullable = false)
    private Timestamp creationTime;
    @Column(nullable = false)
    private String confirmKey;

}
```

Файл ReserveToken.java:

```
package com.andrew.mediastorage.server.api.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "reserve_tokens")
public class ReserveToken {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String value;

    @ManyToOne
    @JoinColumn(name="user_id", nullable=false)
    private User user;

}
```

Файл RestoreRequest.java:

```
package com.andrew.mediastorage.server.api.entity;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import javax.persistence.*;
import java.sql.Timestamp;

@Getter
@Setter
@ToString
@NoArgsConstructor
@Entity
@Table(name = "restore_requests")
public class RestoreRequest {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private Long userId;
    @Column(nullable = false)
    private Timestamp creationTime;
    @Column(nullable = false)
    private String restoreKey;

}
```

Файл User.java:

```
package com.andrew.mediastorage.server.api.entity;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Entity
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column(nullable = false)
    private String username;
    @Column(nullable = false)
    private String password;
    @Column(nullable = false)
    private boolean mfaActive;
```

```
    private String secretKey;
    @Column(nullable = false)
    private String firstName;
    @Column(nullable = false)
    private String lastName;
    @Column(nullable = false)
    private String email;
    @Column(nullable = false)
    private Timestamp registrationTime;

    @OneToMany(
            mappedBy = "user",
            fetch = FetchType.EAGER,
            orphanRemoval = true
    )
    private List<ReserveToken> reserveTokenList = new ArrayList<>();

}
```

Файл AuthLib.java:

```
package com.andrew.mediastorage.server.api.lib;

import com.andrew.mediastorage.server.api.dto.*;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public interface AuthLib {

    RegistrationResultDto registration(RegistrationFormDto registrationFormDto);

    boolean confirm(String key);

    FormStatusDto restore(String email);

    FormStatusDto resetPassword(ResetPasswordRequestDto resetPasswordRequestDto);

    ActivateMfaResultDto activateMfa(String password);

    FormStatusDto deactivateMfa(DeactivateMfaRequestDto deactivateMfaRequestDto);

    FormStatusDto deleteAccount(DeleteAccountRequestDto deleteAccountRequestDto);

    void refreshToken(HttpServletRequest request, HttpServletResponse response);

}
```

Файл MediaLib.java:

```
package com.andrew.mediastorage.server.api.lib;

import com.andrew.mediastorage.server.api.dto.MediaDto;
import org.springframework.web.multipart.MultipartFile;
```

```
import java.util.List;

public interface MediaLib {

    List<MediaDto> getMediaListBlock(Long beforeDate);

    byte[] getMediaPreview(Long id);

    byte[] getMedia(Long id);

    List<MediaDto> addMedia(List<MultipartFile> mediaList);

    boolean deleteAllMedia();

    boolean deleteMedia(Long id);

}
```

Файл UserLib.java:

```
package com.andrew.mediastorage.server.api.lib;

import com.andrew.mediastorage.server.api.dto.BasicSettingsDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.SettingsDto;

public interface UserLib {

    SettingsDto getSettings();

    FormStatusDto changeBasicSettings(BasicSettingsDto basicSettingsDto);
}
```

Файл DtoEntityMapper.java:

```
package com.andrew.mediastorage.server.api.mapper;

import com.andrew.mediastorage.server.api.dto.MediaDto;
import com.andrew.mediastorage.server.api.dto.RegistrationFormDto;
import com.andrew.mediastorage.server.api.dto.SettingsDto;
import com.andrew.mediastorage.server.api.entity.Media;
import com.andrew.mediastorage.server.api.entity.RegistrationRequest;
import com.andrew.mediastorage.server.api.entity.ReserveToken;
import com.andrew.mediastorage.server.api.entity.User;

import java.text.ParseException;
import java.util.ArrayList;
import java.util.List;

public class DtoEntityMapper {

    public static MediaDto mapToMediaDto(Media media) {
        MediaDto mediaDto = new MediaDto();
        mediaDto.setId(media.getId());
        mediaDto.setName(media.getName());
```

```
        mediaDto.setExtension(media.getExtension());
        mediaDto.setWidth(media.getWidth());
        mediaDto.setHeight(media.getHeight());
        mediaDto.setLastModified(media.getLastModified().getTime());
        return mediaDto;
    }

    public static SettingsDto mapToSettingsDto(User userDbi) {
        SettingsDto settingsDto = new SettingsDto();
        settingsDto.setFirstName(userDbi.getFirstName());
        settingsDto.setLastName(userDbi.getLastName());
        settingsDto.setUsername(userDbi.getUsername());
        settingsDto.setMfaActive(userDbi.isMfaActive());
        return settingsDto;
    }

    public static RegistrationRequest mapToRegistrationRequest(RegistrationFormDto
registrationFormDto) throws ParseException {
        RegistrationRequest registrationRequest = new RegistrationRequest();
        registrationRequest.setFirstName(registrationFormDto.getFirstName());
        registrationRequest.setLastName(registrationFormDto.getLastName());
        registrationRequest.setEmail(registrationFormDto.getEmail());
        registrationRequest.setUsername(registrationFormDto.getUsername());
        registrationRequest.setPassword(registrationFormDto.getPassword());
        return registrationRequest;
    }

    public static User mapToUser(RegistrationRequest registrationRequest) {
        User user = new User();
        user.setUsername(registrationRequest.getUsername());
        user.setPassword(registrationRequest.getPassword());
        user.setMfaActive(registrationRequest.isMfaActive());
        user.setSecretKey(registrationRequest.getSecretKey());
        user.setFirstName(registrationRequest.getFirstName());
        user.setLastName(registrationRequest.getLastName());
        user.setEmail(registrationRequest.getEmail());
        return user;
    }

    public static List<ReserveToken> mapToReserveTokenList(String reserveTokens, User
user) {
        List<ReserveToken> reserveTokenList = new ArrayList<>();
        if (reserveTokens != null) {
            for (String token : reserveTokens.split(",")) {
                reserveTokenList.add(mapToReserveToken(token, user));
            }
        }
        return reserveTokenList;
    }

    public static List<ReserveToken> mapToReserveTokenList(List<String> reserveTokenList,
User user) {
        List<ReserveToken> reserveTokenDbiList = new ArrayList<>();
        for (String token : reserveTokenList) {
            reserveTokenDbiList.add(mapToReserveToken(token, user));
        }
        return reserveTokenDbiList;
```

```
    }

    public static ReserveToken mapToReserveToken(String token, User user) {
        ReserveToken reserveToken = new ReserveToken();
        reserveToken.setValue(token);
        reserveToken.setUser(user);
        return reserveToken;
    }

}
```

Файл JwtProvider.java:

```
package com.andrew.mediastorage.server.api.provider;

import com.andrew.mediastorage.server.api.constants.TokenType;
import com.auth0.jwt.exceptions.JWTCreationException;
import com.auth0.jwt.exceptions.JWTVerificationException;
import com.auth0.jwt.interfaces.DecodedJWT;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

import javax.servlet.http.HttpServletRequest;
import java.util.List;

public interface JwtProvider {
    String createAccessToken(String username, List<String> roles) throws
IllegalArgumentException, JWTCreationException;

    String createRefreshToken(String username) throws IllegalArgumentException,
JWTCreationException;

    String resolveToken(HttpServletRequest request);

    DecodedJWT verifyToken(String token, TokenType tokenType) throws
JWTVerificationException;

    Authentication getAuthentication(DecodedJWT decodedJWT) throws
UsernameNotFoundException;
}
```

Файл MFATokenProvider.java:

```
package com.andrew.mediastorage.server.api.provider;

import dev.samstevens.totp.exceptions.QrGenerationException;

import java.util.List;

public interface MFATokenProvider {

    String generateSecretKey();

    List<String> generateReserveTokenList();
```

```
    String getQRCode(String secretKey, String username) throws QrGenerationException;

    boolean verifyTotp(String code, String secret);

}
```

Файл MediaRepository.java:

```
package com.andrew.mediastorage.server.api.repository;

import com.andrew.mediastorage.server.api.entity.Media;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.transaction.annotation.Transactional;

import java.sql.Timestamp;
import java.util.List;

@Transactional
public interface MediaRepository extends JpaRepository<Media, Long> {

    Media findByIdAndUserId(Long id, Long userId);

    List<Media> findByUserId(Long userId);

    List<Media> findByUserIdAndLastModifiedBeforeOrderByLastModifiedDesc(Long userId,
Timestamp lastModifiedBefore, Pageable pageable);

    List<Media> findByUserIdOrderByLastModifiedDesc(Long userId, Pageable pageable);

    void deleteByUserId(Long userId);

    void deleteByIdAndUserId(Long id, Long userId);

}
```

Файл RegistrationRequestRepository.java:

```
package com.andrew.mediastorage.server.api.repository;

import com.andrew.mediastorage.server.api.entity.RegistrationRequest;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.transaction.annotation.Transactional;

@Transactional
public interface RegistrationRequestRepository extends JpaRepository<RegistrationRequest,
Long> {

    RegistrationRequest findByUsername(String username);

    void deleteByEmail(String email);

    RegistrationRequest findByConfirmKey(String confirmKey);

}
```

Файл ReserveTokenRepository.java:

```
package com.andrew.mediastorage.server.api.repository;

import com.andrew.mediastorage.server.api.entity.ReserveToken;
import com.andrew.mediastorage.server.api.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.transaction.annotation.Transactional;

@Transactional
public interface ReserveTokenRepository extends JpaRepository<ReserveToken, Long> {

    void deleteAllByUser(User user);

}
```

Файл RestoreRequestRepository.java:

```
package com.andrew.mediastorage.server.api.repository;

import com.andrew.mediastorage.server.api.entity.RestoreRequest;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.transaction.annotation.Transactional;

@Transactional
public interface RestoreRequestRepository extends JpaRepository<RestoreRequest, Long> {

    RestoreRequest findByRestoreKey(String restoreKey);

    void deleteByUserId(Long userId);

}
```

Файл UserRepository.java:

```
package com.andrew.mediastorage.server.api.repository;

import com.andrew.mediastorage.server.api.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.transaction.annotation.Transactional;


@Transactional
public interface UserRepository extends JpaRepository<User, Long> {

    User findByUsername(String username);

    User findByEmail(String email);

}
```

Файл EmailSenderService.java:

```
package com.andrew.mediastorage.server.api.service;

import org.springframework.mail.MailException;
import org.springframework.scheduling.annotation.Async;

public interface EmailSenderService {
    void sendEmail(String toEmail, String subject, String body) throws MailException;

}
```

Файл MediaService.java:

```
package com.andrew.mediastorage.server.api.service;

import com.andrew.mediastorage.server.api.entity.Media;

import java.util.List;

public interface MediaService {

    Media findByIdAndUserId(Long id, Long userId);

    List<Media> findByUserId(Long userId);

    List<Media> findMediaListBlock(Long userId, Long beforeDate, int size);

    Media save(Media media);

    void deleteByUserId(Long userId);

    void deleteByIdAndUserId(Long id, Long userId);

}
```

Файл RegistrationRequestService.java:

```
package com.andrew.mediastorage.server.api.service;

import com.andrew.mediastorage.server.api.entity.RegistrationRequest;

public interface RegistrationRequestService {

    RegistrationRequest findByUsername(String username);

    RegistrationRequest findByConfirmKey(String confirmKey);

    RegistrationRequest save(RegistrationRequest registrationRequest);

    void delete(RegistrationRequest registrationRequest);

    void deleteByEmail(String email);

}
```

Файл RegistrationService.java:

```
package com.andrew.mediastorage.server.api.service;

import com.andrew.mediastorage.server.api.dto.RegistrationFormDto;
import com.andrew.mediastorage.server.api.dto.RegistrationResultDto;
import org.springframework.scheduling.annotation.Async;

public interface RegistrationService {

    @Async
    RegistrationResultDto register(RegistrationFormDto registrationFormDto);

}
```

Файл ReserveTokenService.java:

```
package com.andrew.mediastorage.server.api.service;

import com.andrew.mediastorage.server.api.entity.ReserveToken;
import com.andrew.mediastorage.server.api.entity.User;

import java.util.List;

public interface ReserveTokenService {
    List<ReserveToken> saveAll(List<ReserveToken> reserveKeyList);

    void deleteById(Long id);

    void deleteAllByUser(User user);

}
```

Файл RestoreRequestService.java:

```
package com.andrew.mediastorage.server.api.service;

import com.andrew.mediastorage.server.api.entity.RestoreRequest;

public interface RestoreRequestService {

    RestoreRequest save(RestoreRequest restoreRequest);

    RestoreRequest findByRestoreKey(String confirmKey);

    void deleteByUserId(Long userId);

}
```

Файл RestoreService.java:

```
package com.andrew.mediastorage.server.api.service;
```

```
import com.andrew.mediastorage.server.api.entity.User;
import org.springframework.scheduling.annotation.Async;

public interface RestoreService {

    @Async
    void restore(User user);

}
```

Файл RestoreServiceImpl.java:

```
package com.andrew.mediastorage.server.api.service;

import com.andrew.mediastorage.server.api.entity.RestoreRequest;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.property.ResourcesProperties;
import com.andrew.mediastorage.server.property.SystemProperties;
import com.andrew.mediastorage.server.util.GeneratorUtil;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;
import org.springframework.web.util.UriComponentsBuilder;

@Service
public class RestoreServiceImpl implements RestoreService {

    private static final Logger LOGGER = LogManager.getLogger(RestoreServiceImpl.class);

    private static final String RESET_PASSWORD_ENDPOINT = "resetPassword";

    private RestoreRequestService restoreRequestService;
    private EmailSenderService emailSenderService;
    private SystemProperties systemProperties;
    private ResourcesProperties resourcesProperties;

    @Async
    @Override
    public void restore(User user) {
        try {
            String restoreKey =
GeneratorUtil.genRandStr(GeneratorUtil.DEFAULT_KEY_LENGTH);
            String restoreUri = UriComponentsBuilder
                    .fromUriString(systemProperties.getClientServerUrl())
                    .pathSegment(RESET_PASSWORD_ENDPOINT)
                    .pathSegment(restoreKey)
                    .build()
                    .toString();
            emailSenderService.sendEmail(user.getEmail(),
resourcesProperties.getRestoreMailSubjectText(),
String.format(resourcesProperties.getRestoreMailBodyText(), restoreUri));
            restoreRequestService.deleteByUserId(user.getId());
            RestoreRequest restoreRequest = new RestoreRequest();
```

```
            restoreRequest.setUserId(user.getId());
            restoreRequest.setRestoreKey(restoreKey);
            restoreRequestService.save(restoreRequest);
        } catch (Exception e) {
            LOGGER.error(e);
        }
    }

    @Autowired
    public void setRestoreRequestService(RestoreRequestService restoreRequestService) {
        this.restoreRequestService = restoreRequestService;
    }

    @Autowired
    public void setEmailSenderService(EmailSenderService emailSenderService) {
        this.emailSenderService = emailSenderService;
    }

    @Autowired
    public void setSystemProperties(SystemProperties systemProperties) {
        this.systemProperties = systemProperties;
    }

    @Autowired
    public void setResourcesProperties(ResourcesProperties resourcesProperties) {
        this.resourcesProperties = resourcesProperties;
    }

}
```

Файл UserService.java:

```
package com.andrew.mediastorage.server.api.service;

import com.andrew.mediastorage.server.api.entity.User;

public interface UserService {

    User findById(Long id);

    User findByUsername(String username);

    User findByEmail(String email);

    User save(User user);

    void delete(User userDbi);

}
```

Файл ActivateMfaValidator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.dto.FormStatusDto;
```

```
import com.andrew.mediastorage.server.api.entity.User;

public interface ActivateMfaValidator {

    FormStatusDto validate(String password, User user);

}
```

Файл DeactivateMfaValidator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.dto.DeactivateMfaRequestDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.entity.User;

public interface DeactivateMfaValidator {

    FormStatusDto validate(DeactivateMfaRequestDto deactivateMfaRequestDto, User user);

}
```

Файл DeleteAccountValidator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.dto.DeleteAccountRequestDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.entity.User;

public interface DeleteAccountValidator {
    FormStatusDto validate(DeleteAccountRequestDto deleteAccountRequestDto, User user);

}
```

Файл RegistrationValidator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.RegistrationFormDto;

public interface RegistrationValidator {

    FormStatusDto validate(RegistrationFormDto registrationFormDto);

}
```

Файл ResetPasswordValidator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.ResetPasswordRequestDto;
```

```
public interface ResetPasswordValidator {
    FormStatusDto validate(ResetPasswordRequestDto resetPasswordRequestDto);

}
```

Файл RestoreValidator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.dto.FormStatusDto;

public interface RestoreValidator {
    FormStatusDto validate(String email);

}
```

Файл SettingsValidator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.dto.BasicSettingsDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;

public interface SettingsValidator {

    FormStatusDto validateBasicSettings(BasicSettingsDto basicSettingsDto, String
previousUsername, String email);

}
```

Файл Validator.java:

```
package com.andrew.mediastorage.server.api.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.entity.ReserveToken;

import java.util.List;
import java.util.Map;

public interface Validator {

    Map<FormField, String> validateFirstName(String firstName);

    Map<FormField, String> validateLastName(String lastName);

    Map<FormField, String> validateEmail(String email, boolean mustExist);

    Map<FormField, String> validateUsername(String username, String previousUsername,
String email);

    Map<FormField, String> validatePassword(String password);
```

```
    Map<FormField, String> checkPassword(String password, String passwordHash);

    Map<FormField, String> checkTryActivateMfa(boolean mfaActive);

    Map<FormField, String> checkTryDeactivateMfa(boolean mfaActive);

    Map<FormField, String> checkMfaToken(String mfaToken, String secretKey,
List<ReserveToken> reserveTokenList);

    Map<FormField, String> validateRestoreKey(String restoreKey);

}
```

Файл AppConfig.java:

```
package com.andrew.mediastorage.server.configuration;

import dev.samstevens.totp.code.CodeVerifier;
import dev.samstevens.totp.code.DefaultCodeGenerator;
import dev.samstevens.totp.code.DefaultCodeVerifier;
import dev.samstevens.totp.qr.QrGenerator;
import dev.samstevens.totp.qr.ZxingPngQrGenerator;
import dev.samstevens.totp.secret.DefaultSecretGenerator;
import dev.samstevens.totp.secret.SecretGenerator;
import dev.samstevens.totp.time.SystemTimeProvider;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class AppConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public SecretGenerator secretGenerator() {
        return new DefaultSecretGenerator(64);
    }

    @Bean
    public QrGenerator qrGenerator() {
        return new ZxingPngQrGenerator();
    }

    @Bean
    public CodeVerifier codeVerifier() {
        return new DefaultCodeVerifier(new DefaultCodeGenerator(), new
SystemTimeProvider());
    }

}
```

Файл SecurityConfig.java:

```
package com.andrew.mediastorage.server.configuration;

import com.andrew.mediastorage.server.api.provider.JwtProvider;
import com.andrew.mediastorage.server.security.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManag
erBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdap
ter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import org.springframework.web.filter.CorsFilter;

@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private final JwtProvider jwtProvider;

    public static final String LOGIN_ENDPOINT = "/api/auth/login";
    public static final String LOGOUT_ENDPOINT = "/api/auth/logout";
    public static final String REGISTRATION_ENDPOINT = "/api/auth/registration";
    public static final String REFRESH_TOKEN_ENDPOINTS = "/api/auth/refreshToken";
    public static final String CONFIRM_ENDPOINTS = "/api/auth/confirm";
    public static final String RESTORE_ENDPOINTS = "/api/auth/restore";
    public static final String RESET_PASSWORD_ENDPOINTS = "/api/auth/resetPassword";

    public SecurityConfig(JwtProvider jwtProvider) {
        this.jwtProvider = jwtProvider;
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .csrf().disable()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests().antMatchers(
                        LOGIN_ENDPOINT,
                        LOGOUT_ENDPOINT,
                        REGISTRATION_ENDPOINT,
                        REFRESH_TOKEN_ENDPOINTS,
                        CONFIRM_ENDPOINTS,
                        RESTORE_ENDPOINTS,
                        RESET_PASSWORD_ENDPOINTS
                ).permitAll()
```

```
                .and()
                .authorizeRequests().anyRequest().authenticated()
                .and()
                .apply(securityConfigurer());
    }

    @Bean
    MFAAuthenticationProvider mfaAuthenticationProvider() {
        return new MFAAuthenticationProvider();
    }

    @Bean
    public SecurityConfigurer securityConfigurer() throws Exception {
        return new SecurityConfigurer(corsFilter(), jwtFilter(), authenticationFilter());
    }

    @Bean
    JwtFilter jwtFilter() {
        return new JwtFilter(jwtProvider);
    }

    AuthenticationFilter authenticationFilter() throws Exception {
        AuthenticationFilter authenticationFilter = new
AuthenticationFilter(authenticationManagerBean(), jwtProvider);
        authenticationFilter.setFilterProcessesUrl(LOGIN_ENDPOINT);
        return authenticationFilter;
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Bean
    public CorsFilter corsFilter() {
        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        CorsConfiguration config = new CorsConfiguration();
        config.addAllowedOrigin("*");
        config.addAllowedHeader("*");
        config.addAllowedMethod("OPTIONS");
        config.addAllowedMethod("GET");
        config.addAllowedMethod("POST");
        config.addAllowedMethod("PUT");
        config.addAllowedMethod("PATCH");
        config.addAllowedMethod("DELETE");
        source.registerCorsConfiguration("/**", config);
        return new CorsFilter(source);
    }

}
```

Файл AuthController.java:

```
package com.andrew.mediastorage.server.controller;
```

```
import com.andrew.mediastorage.server.api.dto.*;
import com.andrew.mediastorage.server.api.lib.AuthLib;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@RestController
@RequestMapping("/api/auth")
public class AuthController {

    private AuthLib authLib;

    @PostMapping("/registration")
    public ResponseEntity<RegistrationResultDto> registration(@RequestBody
RegistrationFormDto registrationFormDto) {
        RegistrationResultDto registrationResultDto =
authLib.registration(registrationFormDto);
        return ResponseEntity.ok().body(registrationResultDto);
    }

    @GetMapping("/confirm")
    public ResponseEntity<Void> confirm(@RequestParam String key) {
        if (authLib.confirm(key)) {
            return ResponseEntity.ok().build();
        }
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
    }

    @GetMapping("/restore")
    public ResponseEntity<FormStatusDto> restore(@RequestParam String email) {
        FormStatusDto formStatusDto = authLib.restore(email);
        return ResponseEntity.ok().body(formStatusDto);
    }

    @PostMapping("/resetPassword")
    public ResponseEntity<FormStatusDto> resetPassword(@RequestBody
ResetPasswordRequestDto resetPasswordRequestDto) {
        FormStatusDto formStatusDto = authLib.resetPassword(resetPasswordRequestDto);
        return ResponseEntity.ok().body(formStatusDto);
    }

    @PostMapping("/activateMfa")
    public ResponseEntity<ActivateMfaResultDto> activateMfa(@RequestBody String password)
{
        ActivateMfaResultDto activateMfaResultDto = authLib.activateMfa(password);
        return ResponseEntity.ok().body(activateMfaResultDto);
    }

    @PostMapping("/deactivateMfa")
    public ResponseEntity<FormStatusDto> deactivateMfa(@RequestBody
DeactivateMfaRequestDto deactivateMfaRequestDto) {
        FormStatusDto formStatusDto = authLib.deactivateMfa(deactivateMfaRequestDto);
        return ResponseEntity.ok().body(formStatusDto);
```

```
    }

    @PostMapping("/deleteAccount")
    public ResponseEntity<FormStatusDto> deleteAccount(@RequestBody
DeleteAccountRequestDto deleteAccountRequestDto) {
        FormStatusDto formStatusDto = authLib.deleteAccount(deleteAccountRequestDto);
        return ResponseEntity.ok().body(formStatusDto);
    }

    @GetMapping("/refreshToken")
    public void refreshToken(HttpServletRequest request, HttpServletResponse response) {
        authLib.refreshToken(request, response);
    }

    @Autowired
    public void setAuthLib(AuthLib authLib) {
        this.authLib = authLib;
    }

}
```

Файл MediaController.java:

```
package com.andrew.mediastorage.server.controller;

import com.andrew.mediastorage.server.api.dto.MediaDto;
import com.andrew.mediastorage.server.api.lib.MediaLib;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

@RestController
@RequestMapping("/api/media")
public class MediaController {

    private MediaLib mediaLib;

    @GetMapping("/getMediaListBlock")
    public ResponseEntity<List<MediaDto>> getMediaListBlock(@RequestParam(required =
false) Long beforeDate) {
        List<MediaDto> mediaDtoList = mediaLib.getMediaListBlock(beforeDate);
        return ResponseEntity.ok().body(mediaDtoList);
    }

    @GetMapping(value = "/getMediaPreview", produces = {MediaType.IMAGE_JPEG_VALUE})
    public ResponseEntity<byte[]> getMediaPreview(@RequestParam Long id) {
        byte[] mediaPreviewBytes = mediaLib.getMediaPreview(id);
        if (mediaPreviewBytes == null) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        } else {
            return ResponseEntity.ok(mediaPreviewBytes);
```

```
        }
    }


    @GetMapping(value = "/getMedia", produces = {MediaType.IMAGE_PNG_VALUE,
MediaType.IMAGE_JPEG_VALUE, MediaType.IMAGE_GIF_VALUE})
    public ResponseEntity<byte[]> getMedia(@RequestParam Long id) {
        byte[] mediaBytes = mediaLib.getMedia(id);
        if (mediaBytes == null) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        } else {
            return ResponseEntity.ok(mediaBytes);
        }
    }

    @PostMapping("/addMedia")
    public ResponseEntity<List<MediaDto>> addMedia(@RequestParam List<MultipartFile>
multipartFileList) {
        List<MediaDto> mediaDtoList = mediaLib.addMedia(multipartFileList);
        return ResponseEntity.ok().body(mediaDtoList);
    }

    @GetMapping("/deleteMedia")
    public ResponseEntity<Void> deleteMedia(@RequestParam Long id) {
        boolean deleted = mediaLib.deleteMedia(id);
        if (deleted) {
            return ResponseEntity.ok().build();
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build();
        }
    }

    @Autowired
    public void setMediaLib(MediaLib mediaLib) {
        this.mediaLib = mediaLib;
    }

}
```

Файл UserController.java:

```
package com.andrew.mediastorage.server.controller;

import com.andrew.mediastorage.server.api.dto.BasicSettingsDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.SettingsDto;
import com.andrew.mediastorage.server.api.lib.UserLib;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/user")
public class UserController {

    private UserLib userLib;
```

```
    @GetMapping("/getSettings")
    public ResponseEntity<SettingsDto> getSettings() {
        SettingsDto settingsDto = userLib.getSettings();
        return ResponseEntity.ok().body(settingsDto);
    }

    @PostMapping("/changeBasicSettings")
    public ResponseEntity<FormStatusDto> changeBasicSettings(@RequestBody BasicSettingsDto
basicSettingsDto) {
        FormStatusDto formStatusDto = userLib.changeBasicSettings(basicSettingsDto);
        return ResponseEntity.ok().body(formStatusDto);
    }

    @Autowired
    public void setUserLib(UserLib userLib) {
        this.userLib = userLib;
    }

}
```

Файл AuthLibImpl.java:

```
package com.andrew.mediastorage.server.lib;

import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.constants.TokenType;
import com.andrew.mediastorage.server.api.dto.*;
import com.andrew.mediastorage.server.api.entity.RegistrationRequest;
import com.andrew.mediastorage.server.api.entity.RestoreRequest;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.lib.AuthLib;
import com.andrew.mediastorage.server.api.lib.MediaLib;
import com.andrew.mediastorage.server.api.mapper.DtoEntityMapper;
import com.andrew.mediastorage.server.api.provider.JwtProvider;
import com.andrew.mediastorage.server.api.provider.MFATokenProvider;
import com.andrew.mediastorage.server.api.service.*;
import com.andrew.mediastorage.server.api.validator.*;
import com.andrew.mediastorage.server.util.CommonUtil;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
@Component
public class AuthLibImpl implements AuthLib {

    private static final Logger LOGGER = LogManager.getLogger(AuthLibImpl.class);

    private MediaLib mediaLib;

    private UserService userService;
    private ReserveTokenService reserveTokenService;
    private RestoreService restoreService;
    private RegistrationService registrationService;
    private RegistrationRequestService registrationRequestService;
    private RestoreRequestService restoreRequestService;

    private RestoreValidator restoreValidator;
    private ResetPasswordValidator resetPasswordValidator;

    private ActivateMfaValidator activateMfaValidator;
    private DeactivateMfaValidator deactivateMfaValidator;
    private DeleteAccountValidator deleteAccountValidator;

    private JwtProvider jwtProvider;
    private MFATokenProvider mfaTokenProvider;

    private PasswordEncoder passwordEncoder;

    @Override
    public RegistrationResultDto registration(RegistrationFormDto registrationFormDto) {
        return registrationService.register(registrationFormDto);
    }

    @Override
    public boolean confirm(String key) {
        RegistrationRequest registrationRequest =
registrationRequestService.findByConfirmKey(key);
        if (registrationRequest == null) {
            return false;
        }
        registrationRequestService.delete(registrationRequest);
        User user = DtoEntityMapper.mapToUser(registrationRequest);
        user = userService.save(user);

reserveTokenService.saveAll(DtoEntityMapper.mapToReserveTokenList(registrationRequest.getR
eserveTokens(), user));
        return true;
    }

    @Override
    public FormStatusDto restore(String email) {
        FormStatusDto formStatusDto = restoreValidator.validate(email);
        if (formStatusDto.getStatus().equals(StatusCode.SUCCESS)) {
            User user = userService.findByEmail(email);
            restoreService.restore(user);
        }
        return formStatusDto;
    }
```

```
    @Override
    public FormStatusDto resetPassword(ResetPasswordRequestDto resetPasswordRequestDto) {
        FormStatusDto formStatusDto =
resetPasswordValidator.validate(resetPasswordRequestDto);
        if (formStatusDto.getStatus().equals(StatusCode.SUCCESS)) {
            RestoreRequest restoreRequest =
restoreRequestService.findByRestoreKey(resetPasswordRequestDto.getRestoreKey());
            restoreRequestService.deleteByUserId(restoreRequest.getUserId());
            User user = userService.findById(restoreRequest.getUserId());
            if (user != null) {

user.setPassword(passwordEncoder.encode(resetPasswordRequestDto.getNewPassword()));
                userService.save(user);
            }
        }
        return formStatusDto;
    }

    @Override
    public ActivateMfaResultDto activateMfa(String password) {
        org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        User userDbi = userService.findByUsername(user.getUsername());

        FormStatusDto formStatusDto = activateMfaValidator.validate(password, userDbi);
        ActivateMfaResultDto activateMfaResultDto = new ActivateMfaResultDto();
        activateMfaResultDto.setFormStatusDto(formStatusDto);

        if (formStatusDto.getStatus().equals(StatusCode.FAILURE)) {
            return activateMfaResultDto;
        }

        try {
            String secretKey = mfaTokenProvider.generateSecretKey();
            List<String> reserveTokenList = mfaTokenProvider.generateReserveTokenList();
            String qrCode = mfaTokenProvider.getQRCode(secretKey, userDbi.getUsername());

            userDbi.setMfaActive(true);
            userDbi.setSecretKey(secretKey);
            activateMfaResultDto.setSecretKey(secretKey);
            activateMfaResultDto.setReserveTokenList(reserveTokenList);
            activateMfaResultDto.setQrCode(qrCode);

            userService.save(userDbi);

reserveTokenService.saveAll(DtoEntityMapper.mapToReserveTokenList(reserveTokenList,
userDbi));
        } catch (Exception e) {
            LOGGER.error(e);
        }

        return activateMfaResultDto;
    }
```

```
    @Override
    public FormStatusDto deactivateMfa(DeactivateMfaRequestDto deactivateMfaRequestDto) {
        org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        User userDbi = userService.findByUsername(user.getUsername());
        FormStatusDto formStatusDto =
deactivateMfaValidator.validate(deactivateMfaRequestDto, userDbi);
        if (formStatusDto.getStatus().equals(StatusCode.SUCCESS)) {
            userDbi.setMfaActive(false);
            userDbi.setSecretKey(null);
            userService.save(userDbi);
            reserveTokenService.deleteAllByUser(userDbi);
        }
        return formStatusDto;
    }

    @Override
    public FormStatusDto deleteAccount(DeleteAccountRequestDto deleteAccountRequestDto) {
        org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        User userDbi = userService.findByUsername(user.getUsername());
        FormStatusDto formStatusDto =
deleteAccountValidator.validate(deleteAccountRequestDto, userDbi);
        if (formStatusDto.getStatus().equals(StatusCode.SUCCESS)) {
            mediaLib.deleteAllMedia();
            userService.delete(userDbi);
        }
        return formStatusDto;
    }

    @Override
    public void refreshToken(HttpServletRequest request, HttpServletResponse response) {
        try {
            String refreshToken = jwtProvider.resolveToken(request);
            if (refreshToken == null) {
                throw new Exception("Refresh token is missing!");
            }
            DecodedJWT decodedJWT = jwtProvider.verifyToken(refreshToken,
TokenType.REFRESH_TOKEN);
            String username = decodedJWT.getSubject();
            User user = userService.findByUsername(username);
            List<String> roles = new ArrayList<>();
            String accessToken = jwtProvider.createAccessToken(user.getUsername(), roles);

            Map<String, String> tokens = new HashMap<>();
            tokens.put("access_token", accessToken);
            tokens.put("refresh_token", refreshToken);

            response.setContentType(MediaType.APPLICATION_JSON_VALUE);
            new ObjectMapper().writeValue(response.getOutputStream(), tokens);
        } catch (Exception e) {
            LOGGER.error(e.getMessage());
            CommonUtil.addErrorToResponse(response, HttpStatus.FORBIDDEN.value(),
e.getMessage());
        }
```

```
    }

    @Autowired
    public void setMediaLib(MediaLib mediaLib) {
        this.mediaLib = mediaLib;
    }

    @Autowired
    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    @Autowired
    public void setReserveTokenService(ReserveTokenService reserveTokenService) {
        this.reserveTokenService = reserveTokenService;
    }

    @Autowired
    public void setRestoreService(RestoreService restoreService) {
        this.restoreService = restoreService;
    }

    @Autowired
    public void setRegistrationService(RegistrationService registrationService) {
        this.registrationService = registrationService;
    }

    @Autowired
    public void setRegistrationRequestService(RegistrationRequestService
registrationRequestService) {
        this.registrationRequestService = registrationRequestService;
    }

    @Autowired
    public void setRestoreRequestService(RestoreRequestService restoreRequestService) {
        this.restoreRequestService = restoreRequestService;
    }

    @Autowired
    public void setRestoreValidator(RestoreValidator restoreValidator) {
        this.restoreValidator = restoreValidator;
    }

    @Autowired
    public void setResetPasswordValidator(ResetPasswordValidator resetPasswordValidator) {
        this.resetPasswordValidator = resetPasswordValidator;
    }

    @Autowired
    public void setActivateMfaValidator(ActivateMfaValidator activateMfaValidator) {
        this.activateMfaValidator = activateMfaValidator;
    }

    @Autowired
    public void setDeactivateMfaValidator(DeactivateMfaValidator deactivateMfaValidator) {
        this.deactivateMfaValidator = deactivateMfaValidator;
    }
```

```
    @Autowired
    public void setDeleteAccountValidator(DeleteAccountValidator deleteAccountValidator) {
        this.deleteAccountValidator = deleteAccountValidator;
    }

    @Autowired
    public void setJwtProvider(JwtProvider jwtProvider) {
        this.jwtProvider = jwtProvider;
    }

    @Autowired
    public void setMfaTokenProvider(MFATokenProvider mfaTokenProvider) {
        this.mfaTokenProvider = mfaTokenProvider;
    }

    @Autowired
    public void setPasswordEncoder(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

}
```

Файл MediaLibImpl.java:

```
package com.andrew.mediastorage.server.lib;

import com.andrew.mediastorage.server.api.dto.MediaDto;
import com.andrew.mediastorage.server.api.entity.Media;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.lib.MediaLib;
import com.andrew.mediastorage.server.api.mapper.DtoEntityMapper;
import com.andrew.mediastorage.server.api.service.MediaService;
import com.andrew.mediastorage.server.api.service.UserService;
import com.andrew.mediastorage.server.property.MediaProperties;
import com.andrew.mediastorage.server.property.StorageProperties;
import com.andrew.mediastorage.server.util.ImageUtil;
import org.apache.commons.io.FilenameUtils;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.util.CollectionUtils;
import org.springframework.web.multipart.MultipartFile;

import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
```

```
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.List;

@Component
public class MediaLibImpl implements MediaLib {

    private static final Logger LOGGER = LogManager.getLogger(MediaLibImpl.class);
    private static final String JPG_EXTENSION = "jpg";

    private UserService userService;
    private MediaService mediaService;
    private StorageProperties storageProperties;
    private MediaProperties mediaProperties;


    @Override
    public List<MediaDto> getMediaListBlock(Long beforeDate) {
        org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        User userDbi = userService.findByUsername(user.getUsername());
        Long userId = userDbi.getId();
        List<Media> mediaList = mediaService.findMediaListBlock(userId, beforeDate,
mediaProperties.getMediaListBlockSize());
        int previewImageHeight = mediaProperties.getPreviewImageHeight();
        List<MediaDto> mediaDtoList = new ArrayList<>();
        for (Media media : mediaList) {
            MediaDto mediaDto = DtoEntityMapper.mapToMediaDto(media);
            mediaDto.setPreviewHeight(previewImageHeight);
            mediaDto.setPreviewWidth(ImageUtil.calcWidth(previewImageHeight,
media.getWidth(), media.getHeight()));
            mediaDtoList.add(mediaDto);
        }
        return mediaDtoList;
    }

    @Override
    public byte[] getMediaPreview(Long id) {
        try {
            org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
            User userDbi = userService.findByUsername(user.getUsername());
            Long userId = userDbi.getId();
            Media media = mediaService.findByIdAndUserId(id, userId);
            if (media != null) {
                String fileName = id + "." + JPG_EXTENSION;
                Path filePath = Paths.get(storageProperties.getPreviewMediaFolderPath(),
fileName);
                if (!filePath.toFile().exists()) {
                    throw new FileNotFoundException("Media preview not found: " + id);
                }
                return Files.readAllBytes(filePath);
            }
        } catch (FileNotFoundException e) {
            LOGGER.warn(e);
```

```
        } catch (Exception e) {
            LOGGER.error(e);
        }
        return null;
    }

    @Override
    public byte[] getMedia(Long id) {
        try {
            org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
            User userDbi = userService.findByUsername(user.getUsername());
            Long userId = userDbi.getId();
            Media media = mediaService.findByIdAndUserId(id, userId);
            if (media != null) {
                Path filePath = Paths.get(storageProperties.getMediaFolderPath(), id + "."
+ media.getExtension());
                if (!filePath.toFile().exists()) {
                    throw new FileNotFoundException("Media not found: " + id);
                }
                return Files.readAllBytes(filePath);
            }
        } catch (FileNotFoundException e) {
            LOGGER.warn(e);
        } catch (IOException e) {
            LOGGER.error(e);
        }
        return null;
    }

    @Override
    public List<MediaDto> addMedia(List<MultipartFile> multipartFileList) {
        org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        User userDbi = userService.findByUsername(user.getUsername());
        Long userId = userDbi.getId();
        int previewImageHeight = mediaProperties.getPreviewImageHeight();
        List<MediaDto> mediaDtoList = new ArrayList<>();
        for (MultipartFile multipartFile : multipartFileList) {
            try {
                String fileName = multipartFile.getOriginalFilename();
                String extension = FilenameUtils.getExtension(fileName);
                InputStream is = new ByteArrayInputStream(multipartFile.getBytes());
                BufferedImage bufferedImage = ImageIO.read(is);

                Media media = new Media();
                media.setUserId(userId);
                media.setName(fileName);
                media.setExtension(extension);
                media.setWidth(bufferedImage.getWidth());
                media.setHeight(bufferedImage.getHeight());
                media.setLastModified(new Timestamp(System.currentTimeMillis()));
                Media savedMedia = mediaService.save(media);
```

```
                BufferedImage mediaPreview =
ImageUtil.toBufferedImage(ImageUtil.resizeByHeight(bufferedImage,
mediaProperties.getPreviewImageHeight(), Image.SCALE_SMOOTH), false);
                Path mediaPreviewPath =
Paths.get(storageProperties.getPreviewMediaFolderPath(), savedMedia.getId() + "." +
JPG_EXTENSION);
                ImageIO.write(mediaPreview, JPG_EXTENSION, mediaPreviewPath.toFile());

                Path mediaPath = Paths.get(storageProperties.getMediaFolderPath(),
savedMedia.getId() + "." + extension);
                Files.write(mediaPath, multipartFile.getBytes());
                MediaDto mediaDto = DtoEntityMapper.mapToMediaDto(media);
                mediaDto.setPreviewHeight(previewImageHeight);
                mediaDto.setPreviewWidth(ImageUtil.calcWidth(previewImageHeight,
media.getWidth(), media.getHeight()));
                mediaDtoList.add(mediaDto);
            } catch (IOException e) {
                LOGGER.error(e);
            }
        }
        return mediaDtoList;
    }

    @Override
    public boolean deleteAllMedia() {
        try {
            org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
            User userDbi = userService.findByUsername(user.getUsername());
            Long userId = userDbi.getId();
            List<Media> mediaList = mediaService.findByUserId(userId);
            if (!CollectionUtils.isEmpty(mediaList)) {
                for (Media media : mediaList) {
                    Long mediaId = media.getId();
                    Path mediaPath = Paths.get(storageProperties.getMediaFolderPath(),
mediaId + "." + media.getExtension());
                    Path previewMediaPath =
Paths.get(storageProperties.getPreviewMediaFolderPath(), mediaId + "." + JPG_EXTENSION);
                    if (!previewMediaPath.toFile().delete() ||
!mediaPath.toFile().delete()) {
                        throw new FileNotFoundException("Failed to delete media: " +
mediaId);
                    }
                }
                mediaService.deleteByUserId(userId);
                return true;
            }
        } catch (Exception e) {
            LOGGER.error(e);
        }
        return false;
    }

    @Override
    public boolean deleteMedia(Long id) {
        try {
```

```
            org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
            User userDbi = userService.findByUsername(user.getUsername());
            Long userId = userDbi.getId();
            Media media = mediaService.findByIdAndUserId(id, userId);
            if (media != null) {
                Path mediaPath = Paths.get(storageProperties.getMediaFolderPath(), id +
"." + media.getExtension());
                Path previewMediaPath =
Paths.get(storageProperties.getPreviewMediaFolderPath(), id + "." + JPG_EXTENSION);
                if (!previewMediaPath.toFile().delete() || !mediaPath.toFile().delete()) {
                    throw new FileNotFoundException("Failed to delete media: " + id);
                }
                mediaService.deleteByIdAndUserId(media.getId(), userId);
                return true;
            }
        } catch (Exception e) {
            LOGGER.error(e);
        }
        return false;
    }

    @Autowired
    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    @Autowired
    public void setMediaService(MediaService mediaService) {
        this.mediaService = mediaService;
    }

    @Autowired
    public void setStorageProperties(StorageProperties storageProperties) {
        this.storageProperties = storageProperties;
    }

    @Autowired
    public void setMediaProperties(MediaProperties mediaProperties) {
        this.mediaProperties = mediaProperties;
    }

}
```

Файл UserLibImpl.java:

```
package com.andrew.mediastorage.server.lib;

import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.BasicSettingsDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.SettingsDto;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.lib.UserLib;
import com.andrew.mediastorage.server.api.mapper.DtoEntityMapper;
```

```
import com.andrew.mediastorage.server.api.service.UserService;
import com.andrew.mediastorage.server.api.validator.SettingsValidator;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;

@Component
public class UserLibImpl implements UserLib {

    private static final Logger LOGGER = LogManager.getLogger(UserLibImpl.class);

    private UserService userService;

    private SettingsValidator settingsValidator;

    @Override
    public SettingsDto getSettings() {
        org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        User userDbi = userService.findByUsername(user.getUsername());
        return DtoEntityMapper.mapToSettingsDto(userDbi);
    }

    @Override
    public FormStatusDto changeBasicSettings(BasicSettingsDto basicSettingsDto) {
        org.springframework.security.core.userdetails.User user =
(org.springframework.security.core.userdetails.User)
SecurityContextHolder.getContext().getAuthentication().getPrincipal();
        User userDbi = userService.findByUsername(user.getUsername());
        FormStatusDto formStatusDto =
settingsValidator.validateBasicSettings(basicSettingsDto, userDbi.getUsername(),
userDbi.getEmail());
        if (formStatusDto.getStatus().equals(StatusCode.SUCCESS)) {
            userDbi.setFirstName(basicSettingsDto.getFirstName());
            userDbi.setLastName(basicSettingsDto.getLastName());
            userDbi.setUsername(basicSettingsDto.getUsername());
            userService.save(userDbi);
        }
        return formStatusDto;
    }

    @Autowired
    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    @Autowired
    public void setSettingsValidator(SettingsValidator settingsValidator) {
        this.settingsValidator = settingsValidator;
    }
}
```

Файл MediaStorageServerApplication.java:

```
package com.andrew.mediastorage.server;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;

@SpringBootApplication
public class MediaStorageServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(MediaStorageServerApplication.class, args);
    }

}
```

Файл FieldsValidationProperties.java:

```
package com.andrew.mediastorage.server.property;

import lombok.Getter;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Setter
@Getter
@Configuration
@PropertySource(value = "classpath:fieldsValidation.properties", encoding = "UTF-8")
@ConfigurationProperties(prefix = "fields-validation")
public class FieldsValidationProperties {

    private String firstNameRegEx;
    private String lastNameRegEx;
    private String usernameRegEx;
    private String passwordRegEx;
    private String reservedUsernames;
    private String separator;

}
```

Файл JwtProperties.java:

```
package com.andrew.mediastorage.server.property;

import lombok.Getter;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Setter
@Getter
@Configuration
@PropertySource(value = "classpath:jwt.properties", encoding = "UTF-8")
```

```
@ConfigurationProperties(prefix = "jwt")
public class JwtProperties {
    private String secretKey;
    private long accessTokenExpirationTime;
    private long refreshTokenExpirationTime;

}
```

Файл MediaProperties.java:

```
package com.andrew.mediastorage.server.property;

import lombok.Getter;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Setter
@Getter
@Configuration
@PropertySource(value = "classpath:media.properties", encoding = "UTF-8")
@ConfigurationProperties(prefix = "media")
public class MediaProperties {

    private int previewImageHeight;
    private int mediaListBlockSize;

}
```

Файл ResourcesProperties.java:

```
package com.andrew.mediastorage.server.property;

import lombok.Getter;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Setter
@Getter
@Configuration
@PropertySource(value = "classpath:resources.properties", encoding = "UTF-8")
@ConfigurationProperties(prefix = "resources")
public class ResourcesProperties {

    private String forbiddenFirstNameErrorMsg;
    private String forbiddenLastNameErrorMsg;
    private String invalidEmailErrorMsg;
    private String busyEmailErrorMsg;
    private String missingAccountByEmailErrorMsg;
    private String forbiddenUsernameErrorMsg;
    private String reservedUsernameErrorMsg;
    private String busyUsernameErrorMsg;
```

```
    private String forbiddenPasswordErrorMsg;
    private String wrongPasswordErrorMsg;
    private String mfaIsAlreadyActivatedErrorMsg;
    private String mfaIsAlreadyDeactivatedErrorMsg;
    private String invalidMfaTokenErrorMsg;
    private String invalidRestoreKeyErrorMsg;
    private String nullFieldErrorMsg;
    private String confirmMailSubjectText;
    private String confirmMailBodyText;
    private String restoreMailSubjectText;
    private String restoreMailBodyText;

}
```

Файл StorageProperties.java:

```
package com.andrew.mediastorage.server.property;

import lombok.Getter;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Setter
@Getter
@Configuration
@PropertySource(value = "classpath:storage.properties", encoding = "UTF-8")
@ConfigurationProperties(prefix = "storage")
public class StorageProperties {

    private String previewMediaFolderPath;
    private String mediaFolderPath;
    private String avatarFolderPath;

}
```

Файл SystemProperties.java:

```
package com.andrew.mediastorage.server.property;

import lombok.Getter;
import lombok.Setter;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;

@Setter
@Getter
@Configuration
@PropertySource(value = "classpath:system.properties", encoding = "UTF-8")
@ConfigurationProperties(prefix = "system")
public class SystemProperties {
    private String clientServerUrl;
```

```
}
```

Файл DefaultJwtProvider.java:

```
package com.andrew.mediastorage.server.provider;

import com.andrew.mediastorage.server.api.constants.JwtConstants;
import com.andrew.mediastorage.server.api.constants.TokenType;
import com.andrew.mediastorage.server.api.provider.JwtProvider;
import com.andrew.mediastorage.server.property.JwtProperties;
import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTCreationException;
import com.auth0.jwt.exceptions.JWTVerificationException;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.auth0.jwt.interfaces.JWTVerifier;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import java.util.Date;
import java.util.List;

@Component
public class DefaultJwtProvider implements JwtProvider, InitializingBean {

    JwtProperties jwtProperties;

    private UserDetailsService userDetailsService;

    private Algorithm algorithm;
    private JWTVerifier verifier;

    @Override
    public void afterPropertiesSet() {
        algorithm = Algorithm.HMAC256(jwtProperties.getSecretKey().getBytes());
        verifier = JWT.require(algorithm).build();
    }

    @Override
    public String createAccessToken(String username, List<String> roles) throws
IllegalArgumentException, JWTCreationException {
        Date expiresAt = new Date(System.currentTimeMillis() +
jwtProperties.getAccessTokenExpirationTime());
        return JWT.create()
                .withSubject(username)
                .withExpiresAt(expiresAt)
                .withClaim(JwtConstants.TOKEN_TYPE, TokenType.ACCESS_TOKEN.name())
                .sign(algorithm);
```

```
    }

    @Override
    public String createRefreshToken(String username) throws IllegalArgumentException,
JWTCreationException {
        Date expiresAt = new Date(System.currentTimeMillis() +
jwtProperties.getRefreshTokenExpirationTime());
        return JWT.create()
                .withSubject(username)
                .withExpiresAt(expiresAt)
                .withClaim(JwtConstants.TOKEN_TYPE, TokenType.REFRESH_TOKEN.name())
                .sign(algorithm);
    }

    @Override
    public String resolveToken(HttpServletRequest request) {
        String bearerToken = request.getHeader(HttpHeaders.AUTHORIZATION);
        if (bearerToken != null && bearerToken.startsWith(JwtConstants.TOKEN_HEADER)) {
            return bearerToken.substring(JwtConstants.TOKEN_HEADER.length());
        }
        return null;
    }

    @Override
    public DecodedJWT verifyToken(String token, TokenType tokenType) throws
JWTVerificationException {
        DecodedJWT decodedJWT = verifier.verify(token);
        if
(!decodedJWT.getClaim(JwtConstants.TOKEN_TYPE).as(String.class).equals(tokenType.name()))
{
            throw new JWTVerificationException("Wrong token type!");
        }
        return decodedJWT;
    }

    @Override
    public Authentication getAuthentication(DecodedJWT decodedJWT) throws
UsernameNotFoundException {
        UserDetails userDetails =
userDetailsService.loadUserByUsername(decodedJWT.getSubject());
        return new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
    }

    @Autowired
    public void setUserDetailsService(UserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Autowired
    public void setJwtProperties(JwtProperties jwtProperties) {
        this.jwtProperties = jwtProperties;
    }

}
```

Файл DefaultMFATokenProvider.java:

```
package com.andrew.mediastorage.server.provider;

import com.andrew.mediastorage.server.api.provider.MFATokenProvider;
import dev.samstevens.totp.code.CodeVerifier;
import dev.samstevens.totp.exceptions.QrGenerationException;
import dev.samstevens.totp.qr.QrData;
import dev.samstevens.totp.qr.QrGenerator;
import dev.samstevens.totp.secret.SecretGenerator;
import dev.samstevens.totp.util.Utils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

@Component
public class DefaultMFATokenProvider implements MFATokenProvider {

    private int RESERVE_KEYS_COUNT = 12;

    private SecretGenerator secretGenerator;

    private QrGenerator qrGenerator;

    private CodeVerifier codeVerifier;

    @Override
    public String generateSecretKey() {
        return secretGenerator.generate();
    }

    @Override
    public List<String> generateReserveTokenList() {
        Random rnd = new Random();
        List<String> reserveKeyList = new ArrayList<>();
        for (int i = 0; i < RESERVE_KEYS_COUNT; i++) {
            int reserveKey = rnd.nextInt(999999);
            reserveKeyList.add(String.format("%06d", reserveKey));
        }
        return reserveKeyList;
    }

    @Override
    public String getQRCode(String secretKey, String username) throws
QrGenerationException {
        QrData data = new QrData.Builder()
                .secret(secretKey)
                .issuer("Media Storage (MFA)")
                .label(username)
                .digits(6)
                .period(30)
                .build();
        return Utils.getDataUriForImage(
                qrGenerator.generate(data),
```

```
                qrGenerator.getImageMimeType()
        );
    }

    @Override
    public boolean verifyTotp(String code, String secretKey) {
        return codeVerifier.isValidCode(secretKey, code);
    }

    @Autowired
    public void setSecretGenerator(SecretGenerator secretGenerator) {
        this.secretGenerator = secretGenerator;
    }

    @Autowired
    public void setQrGenerator(QrGenerator qrGenerator) {
        this.qrGenerator = qrGenerator;
    }

    @Autowired
    public void setCodeVerifier(CodeVerifier codeVerifier) {
        this.codeVerifier = codeVerifier;
    }

}
```

Файл AuthenticationFilter.java:

```
package com.andrew.mediastorage.server.security;

import com.andrew.mediastorage.server.api.data.MFAUsernamePasswordAuthenticationToken;
import com.andrew.mediastorage.server.api.provider.JwtProvider;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.http.MediaType;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.User;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class AuthenticationFilter extends UsernamePasswordAuthenticationFilter {

    private static final Logger LOGGER = LogManager.getLogger(AuthenticationFilter.class);
```

```
    private final AuthenticationManager authenticationManager;
    private final JwtProvider jwtProvider;

    public AuthenticationFilter(AuthenticationManager authenticationManager, JwtProvider
jwtProvider) {
        this.authenticationManager = authenticationManager;
        this.jwtProvider = jwtProvider;
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
HttpServletResponse response) throws AuthenticationException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");
        String mfaToken = request.getParameter("mfaToken");
        MFAUsernamePasswordAuthenticationToken authenticationToken = new
MFAUsernamePasswordAuthenticationToken(username, password, mfaToken);
        Authentication authentication =
authenticationManager.authenticate(authenticationToken);
        return authentication;
    }

    @Override
    protected void successfulAuthentication(HttpServletRequest request,
HttpServletResponse response, FilterChain chain, Authentication authentication) throws
IOException, ServletException {
        User user = (User) authentication.getPrincipal();
        List<String> roles = new ArrayList<>();
        user.getAuthorities().forEach(grantedAuthority ->
roles.add(grantedAuthority.getAuthority()));
        String accessToken = jwtProvider.createAccessToken(user.getUsername(), roles);
        String refreshToken = jwtProvider.createRefreshToken(user.getUsername());

        Map<String, String> tokens = new HashMap<>();
        tokens.put("access_token", accessToken);
        tokens.put("refresh_token", refreshToken);

        response.setContentType(MediaType.APPLICATION_JSON_VALUE);
        new ObjectMapper().writeValue(response.getOutputStream(), tokens);
    }

}
```

Файл JwtFilter.java:

```
package com.andrew.mediastorage.server.security;

import com.andrew.mediastorage.server.api.constants.TokenType;
import com.andrew.mediastorage.server.api.provider.JwtProvider;
import com.andrew.mediastorage.server.configuration.SecurityConfig;
import com.auth0.jwt.interfaces.DecodedJWT;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.MediaType;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class JwtFilter extends OncePerRequestFilter {

    private static final Logger LOGGER = LogManager.getLogger(JwtFilter.class);

    private final JwtProvider jwtProvider;

    public JwtFilter(JwtProvider jwtProvider) {
        this.jwtProvider = jwtProvider;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain) throws ServletException, IOException {
        if (SecurityConfig.LOGIN_ENDPOINT.equals(request.getServletPath()) ||
                SecurityConfig.LOGOUT_ENDPOINT.equals(request.getServletPath()) ||
                SecurityConfig.REGISTRATION_ENDPOINT.equals(request.getServletPath()) ||
                SecurityConfig.REFRESH_TOKEN_ENDPOINTS.equals(request.getServletPath()) ||
                SecurityConfig.CONFIRM_ENDPOINTS.equals(request.getServletPath()) ||
                SecurityConfig.RESTORE_ENDPOINTS.equals(request.getServletPath()) ||
                SecurityConfig.RESET_PASSWORD_ENDPOINTS.equals(request.getServletPath())
        ) {
            filterChain.doFilter(request, response);
        } else {
            try {
                String accessToken = jwtProvider.resolveToken(request);
                if (accessToken == null) {
                    throw new Exception("Access token is missing!");
                }
                DecodedJWT decodedJWT = jwtProvider.verifyToken(accessToken,
TokenType.ACCESS_TOKEN);
                Authentication authentication = jwtProvider.getAuthentication(decodedJWT);
                SecurityContextHolder.getContext().setAuthentication(authentication);
                filterChain.doFilter(request, response);
            } catch (Exception e) {
                LOGGER.error(e.getMessage());
                response.setStatus(HttpStatus.FORBIDDEN.value());
                Map<String, String> error = new HashMap<>();
                error.put("error_message", e.getMessage());
                response.setContentType(MediaType.APPLICATION_JSON_VALUE);
                new ObjectMapper().writeValue(response.getOutputStream(), error);
            }
        }
    }
```

```
}
```

Файл MFAAuthenticationProvider.java:

```
package com.andrew.mediastorage.server.security;

import com.andrew.mediastorage.server.api.data.MFAUser;
import com.andrew.mediastorage.server.api.data.MFAUsernamePasswordAuthenticationToken;
import com.andrew.mediastorage.server.api.entity.ReserveToken;
import com.andrew.mediastorage.server.api.provider.MFATokenProvider;
import com.andrew.mediastorage.server.api.service.ReserveTokenService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.util.StringUtils;

public class MFAAuthenticationProvider extends DaoAuthenticationProvider {

    private ReserveTokenService reserveTokenService;

    private MFATokenProvider mfaTokenProvider;

    @Override
    protected void additionalAuthenticationChecks(UserDetails userDetails,
UsernamePasswordAuthenticationToken authentication) throws AuthenticationException {
        MFAUser mfaUser = (MFAUser) userDetails;
        MFAUsernamePasswordAuthenticationToken authenticationToken =
(MFAUsernamePasswordAuthenticationToken) authentication;
        String mfaToken = authenticationToken.getMfaToken();
        if (mfaUser.isMfaActive()) {
            if (!mfaTokenProvider.verifyTotp(mfaToken, mfaUser.getSecretKey())) {
                ReserveToken selectedReserveToken = mfaUser
                        .getReserveTokenList().stream()
                        .filter(reserveToken -> reserveToken.getValue().equals(mfaToken))
                        .findFirst().orElse(null);
                if (selectedReserveToken != null) {
                    reserveTokenService.deleteById(selectedReserveToken.getId());
                } else {
                    throw new BadCredentialsException(
                            messages.getMessage(

"AbstractUserDetailsAuthenticationProvider.badCredentials",
                                    "Bad credentials"
                            )
                    );
                }
            }
        } else {
            if (StringUtils.hasText(mfaToken)) {
                throw new BadCredentialsException(
                        messages.getMessage(
```

```
"AbstractUserDetailsAuthenticationProvider.badCredentials",
                                "Bad credentials"
                        )
                );
            }
        }
        super.additionalAuthenticationChecks(userDetails, authentication);
    }

    @Autowired
    public void setReserveTokenService(ReserveTokenService reserveTokenService) {
        this.reserveTokenService = reserveTokenService;
    }

    @Autowired
    public void setMfaTokenProvider(MFATokenProvider mfaTokenProvider) {
        this.mfaTokenProvider = mfaTokenProvider;
    }

    @Override
    @Autowired
    public void setUserDetailsService(UserDetailsService userDetailsService) {
        super.setUserDetailsService(userDetailsService);
    }

    @Override
    @Autowired
    public void setPasswordEncoder(PasswordEncoder passwordEncoder) {
        super.setPasswordEncoder(passwordEncoder);
    }

}
```

Файл SecurityConfigurer.java:

```
package com.andrew.mediastorage.server.security;

import org.springframework.security.config.annotation.SecurityConfigurerAdapter;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.DefaultSecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.filter.CorsFilter;

public class SecurityConfigurer extends
SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {

    private final CorsFilter corsFilter;
    private final JwtFilter jwtFilter;
    private final AuthenticationFilter authenticationFilter;

    public SecurityConfigurer(CorsFilter corsFilter, JwtFilter jwtFilter,
AuthenticationFilter authenticationFilter) {
        this.corsFilter = corsFilter;
        this.jwtFilter = jwtFilter;
```

```
        this.authenticationFilter = authenticationFilter;
    }

    @Override
    public void configure(HttpSecurity httpSecurity) {
        httpSecurity.addFilterBefore(corsFilter,
UsernamePasswordAuthenticationFilter.class);
        httpSecurity.addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);
        httpSecurity.addFilter(authenticationFilter);
    }

}
```

Файл EmailSenderServiceImpl.java:

```
package com.andrew.mediastorage.server.service;

import com.andrew.mediastorage.server.api.service.EmailSenderService;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.mail.MailException;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

@Service
public class EmailSenderServiceImpl implements EmailSenderService {

    private static final Logger LOGGER =
LogManager.getLogger(EmailSenderServiceImpl.class);

    @Value("${spring.mail.username}")
    private String from;
    private JavaMailSender javaMailSender;

    @Override
    public void sendEmail(String toEmail, String subject, String body) throws
MailException {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom(from);
        message.setTo(toEmail);
        message.setSubject(subject);
        message.setText(body);
        javaMailSender.send(message);
        LOGGER.debug("Mail sent to: " + toEmail);
    }

    @Autowired
    public void setJavaMailSender(JavaMailSender javaMailSender) {
        this.javaMailSender = javaMailSender;
    }
```

```
}
```

Файл MediaServiceImpl.java:

```
package com.andrew.mediastorage.server.service;

import com.andrew.mediastorage.server.api.entity.Media;
import com.andrew.mediastorage.server.api.repository.MediaRepository;
import com.andrew.mediastorage.server.api.service.MediaService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Service;

import java.sql.Timestamp;
import java.util.List;

@Service
public class MediaServiceImpl implements MediaService {

    private MediaRepository mediaRepository;

    @Override
    public Media findByIdAndUserId(Long id, Long userId) {
        return mediaRepository.findByIdAndUserId(id, userId);
    }

    @Override
    public List<Media> findByUserId(Long userId) {
        return mediaRepository.findByUserId(userId);
    }

    @Override
    public List<Media> findMediaListBlock(Long userId, Long beforeDate, int size) {
        if (beforeDate != null) {
            return
mediaRepository.findByUserIdAndLastModifiedBeforeOrderByLastModifiedDesc(userId, new
Timestamp(beforeDate), PageRequest.of(0, size));
        }
        return mediaRepository.findByUserIdOrderByLastModifiedDesc(userId,
PageRequest.of(0, size));
    }

    @Override
    public Media save(Media media) {
        return mediaRepository.save(media);
    }

    @Override
    public void deleteByUserId(Long userId) {
        mediaRepository.deleteByUserId(userId);
    }

    @Override
    public void deleteByIdAndUserId(Long id, Long userId) {
        mediaRepository.deleteByIdAndUserId(id, userId);
    }
```

```
    @Autowired
    public void setMediaRepository(MediaRepository mediaRepository) {
        this.mediaRepository = mediaRepository;
    }

}
```

Файл RegistrationRequestServiceImpl.java:

```
package com.andrew.mediastorage.server.service;

import com.andrew.mediastorage.server.api.entity.RegistrationRequest;
import com.andrew.mediastorage.server.api.repository.RegistrationRequestRepository;
import com.andrew.mediastorage.server.api.service.RegistrationRequestService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.sql.Timestamp;

@Service
public class RegistrationRequestServiceImpl implements RegistrationRequestService {

    private RegistrationRequestRepository registrationRequestRepository;

    private PasswordEncoder passwordEncoder;

    @Override
    public RegistrationRequest findByUsername(String username) {
        return registrationRequestRepository.findByUsername(username);
    }

    @Override
    public RegistrationRequest findByConfirmKey(String confirmKey) {
        return registrationRequestRepository.findByConfirmKey(confirmKey);
    }

    @Override
    public RegistrationRequest save(RegistrationRequest registrationRequest) {

registrationRequest.setPassword(passwordEncoder.encode(registrationRequest.getPassword()))
;
        registrationRequest.setCreationTime(new Timestamp(System.currentTimeMillis()));
        return registrationRequestRepository.save(registrationRequest);
    }

    @Override
    public void delete(RegistrationRequest registrationRequest) {
        registrationRequestRepository.delete(registrationRequest);
    }


    @Override
    public void deleteByEmail(String email) {
        registrationRequestRepository.deleteByEmail(email);
```

```
    }

    @Autowired
    public void setRegistrationRequestRepository(RegistrationRequestRepository
registrationRequestRepository) {
        this.registrationRequestRepository = registrationRequestRepository;
    }

    @Autowired
    public void setPasswordEncoder(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

}
```

Файл RegistrationServiceImpl.java:

```
package com.andrew.mediastorage.server.service;

import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.RegistrationFormDto;
import com.andrew.mediastorage.server.api.dto.RegistrationResultDto;
import com.andrew.mediastorage.server.api.entity.RegistrationRequest;
import com.andrew.mediastorage.server.api.mapper.DtoEntityMapper;
import com.andrew.mediastorage.server.api.provider.MFATokenProvider;
import com.andrew.mediastorage.server.api.service.EmailSenderService;
import com.andrew.mediastorage.server.api.service.RegistrationRequestService;
import com.andrew.mediastorage.server.api.service.RegistrationService;
import com.andrew.mediastorage.server.api.validator.RegistrationValidator;
import com.andrew.mediastorage.server.property.ResourcesProperties;
import com.andrew.mediastorage.server.property.SystemProperties;
import com.andrew.mediastorage.server.util.GeneratorUtil;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.web.util.UriComponentsBuilder;

import java.sql.Timestamp;
import java.util.List;

@Service
public class RegistrationServiceImpl implements RegistrationService {

    private static final Logger LOGGER =
LogManager.getLogger(RegistrationServiceImpl.class);

    private static final String CONFIRM_ENDPOINT = "confirm";

    private RegistrationRequestService registrationRequestService;
    private EmailSenderService emailSenderService;

    private RegistrationValidator registrationValidator;

    private MFATokenProvider mfaTokenProvider;
```

```
    private SystemProperties systemProperties;
    private ResourcesProperties resourcesProperties;

    @Override
    public RegistrationResultDto register(RegistrationFormDto registrationFormDto) {
        FormStatusDto formStatusDto = registrationValidator.validate(registrationFormDto);
        RegistrationResultDto registrationResultDto = new RegistrationResultDto();
        registrationResultDto.setFormStatusDto(formStatusDto);

        if (formStatusDto.getStatus().equals(StatusCode.FAILURE)) {
            return registrationResultDto;
        }

        try {
            String confirmKey =
GeneratorUtil.genRandStr(GeneratorUtil.DEFAULT_KEY_LENGTH);
            String confirmUri = UriComponentsBuilder
                    .fromUriString(systemProperties.getClientServerUrl())
                    .pathSegment(CONFIRM_ENDPOINT)
                    .pathSegment(confirmKey)
                    .build()
                    .toString();

            RegistrationRequest registrationRequest =
DtoEntityMapper.mapToRegistrationRequest(registrationFormDto);
            registrationRequest.setCreationTime(new
Timestamp(System.currentTimeMillis()));
            registrationRequest.setConfirmKey(confirmKey);
            if (registrationFormDto.isUseMfa()) {
                String secretKey = mfaTokenProvider.generateSecretKey();
                List<String> reserveTokenList =
mfaTokenProvider.generateReserveTokenList();
                String qrCode = mfaTokenProvider.getQRCode(secretKey,
registrationFormDto.getUsername());
                registrationRequest.setMfaActive(true);
                registrationRequest.setSecretKey(secretKey);
                registrationRequest.setReserveTokens(String.join(",", reserveTokenList));
                registrationResultDto.setSecretKey(secretKey);
                registrationResultDto.setReserveTokenList(reserveTokenList);
                registrationResultDto.setQrCode(qrCode);
            }
            emailSenderService.sendEmail(registrationFormDto.getEmail(),
resourcesProperties.getConfirmMailSubjectText(),
String.format(resourcesProperties.getConfirmMailBodyText(), confirmUri));
            registrationRequestService.deleteByEmail(registrationFormDto.getEmail());
            registrationRequestService.save(registrationRequest);
        } catch (Exception e) {
            LOGGER.error(e);
        }
        return registrationResultDto;
    }

    @Autowired
    public void setRegistrationRequestService(RegistrationRequestService
registrationRequestService) {
        this.registrationRequestService = registrationRequestService;
```

```
    }

    @Autowired
    public void setEmailSenderService(EmailSenderService emailSenderService) {
        this.emailSenderService = emailSenderService;
    }

    @Autowired
    public void setRegistrationValidator(RegistrationValidator registrationValidator) {
        this.registrationValidator = registrationValidator;
    }

    @Autowired
    public void setMfaTokenProvider(MFATokenProvider mfaTokenProvider) {
        this.mfaTokenProvider = mfaTokenProvider;
    }

    @Autowired
    public void setSystemProperties(SystemProperties systemProperties) {
        this.systemProperties = systemProperties;
    }

    @Autowired
    public void setResourcesProperties(ResourcesProperties resourcesProperties) {
        this.resourcesProperties = resourcesProperties;
    }

}
```

Файл ReserveTokenServiceImpl.java:

```
package com.andrew.mediastorage.server.service;

import com.andrew.mediastorage.server.api.entity.ReserveToken;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.repository.ReserveTokenRepository;
import com.andrew.mediastorage.server.api.service.ReserveTokenService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ReserveTokenServiceImpl implements ReserveTokenService {

    private ReserveTokenRepository reserveTokenRepository;

    @Override
    public List<ReserveToken> saveAll(List<ReserveToken> reserveKeyList) {
        return reserveTokenRepository.saveAll(reserveKeyList);
    }

    @Override
    public void deleteById(Long id) {
        reserveTokenRepository.deleteById(id);
    }
```

```
    @Override
    public void deleteAllByUser(User user) {
        reserveTokenRepository.deleteAllByUser(user);
    }

    @Autowired
    public void setReserveTokenRepository(ReserveTokenRepository reserveTokenRepository) {
        this.reserveTokenRepository = reserveTokenRepository;
    }
}
```

Файл RestoreRequestServiceImpl.java:

```
package com.andrew.mediastorage.server.service;

import com.andrew.mediastorage.server.api.entity.RestoreRequest;
import com.andrew.mediastorage.server.api.repository.RestoreRequestRepository;
import com.andrew.mediastorage.server.api.service.RestoreRequestService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.sql.Timestamp;

@Service
public class RestoreRequestServiceImpl implements RestoreRequestService {

    private RestoreRequestRepository restoreRequestRepository;

    @Override
    public RestoreRequest save(RestoreRequest restoreRequest) {
        restoreRequest.setCreationTime(new Timestamp(System.currentTimeMillis()));
        return restoreRequestRepository.save(restoreRequest);
    }

    @Override
    public RestoreRequest findByRestoreKey(String confirmKey) {
        return restoreRequestRepository.findByRestoreKey(confirmKey);
    }

    @Override
    public void deleteByUserId(Long userId) {
        restoreRequestRepository.deleteByUserId(userId);
    }

    @Autowired
    public void setRestoreRequestRepository(RestoreRequestRepository
restoreRequestRepository) {
        this.restoreRequestRepository = restoreRequestRepository;
    }

}
```

Файл UserServiceImpl.java:

```
package com.andrew.mediastorage.server.service;

import com.andrew.mediastorage.server.api.data.MFAUser;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.repository.UserRepository;
import com.andrew.mediastorage.server.api.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.sql.Timestamp;
import java.util.Collection;
import java.util.Collections;

@Service
public class UserServiceImpl implements UserService, UserDetailsService {

    private static final String USER_ROLE = "USER";

    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByUsername(username);
        if (user == null) {
            throw new UsernameNotFoundException("User with username (" + username + ") not
found!");
        }
        Collection<SimpleGrantedAuthority> authorities = Collections.singletonList(new
SimpleGrantedAuthority(USER_ROLE));
        return new MFAUser(user.getUsername(), user.getPassword(), user.isMfaActive(),
user.getSecretKey(), user.getReserveTokenList(), authorities);
    }

    @Override
    public User findById(Long id) {
        return userRepository.findById(id).orElse(null);
    }

    @Override
    public User findByUsername(String username) {
        return userRepository.findByUsername(username);
    }

    @Override
    public User findByEmail(String email) {
        return userRepository.findByEmail(email);
    }

    @Override
    public User save(User user) {
        user.setRegistrationTime(new Timestamp(System.currentTimeMillis()));
        return userRepository.save(user);
```

```
    }

    @Override
    public void delete(User userDbi) {
        userRepository.delete(userDbi);
    }

    @Autowired
    public void setUserRepository(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

}
```

Файл ServletInitializer.java:

```
package com.andrew.mediastorage.server;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(MediaStorageServerApplication.class);
    }

}
```

Файл CommonUtil.java:

```
package com.andrew.mediastorage.server.util;

import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.http.MediaType;

import javax.servlet.http.HttpServletResponse;
import java.util.HashMap;
import java.util.Map;

public class CommonUtil {

    private static final Logger LOGGER = LogManager.getLogger(CommonUtil.class);

    public static void addErrorToResponse(HttpServletResponse response, int status, String
errorMessage) {
        response.setStatus(status);
        Map<String, String> error = new HashMap<>();
        error.put("error_message", errorMessage);
        response.setContentType(MediaType.APPLICATION_JSON_VALUE);
        try {
            new ObjectMapper().writeValue(response.getOutputStream(), error);
```

```
        } catch (Exception e) {
            LOGGER.error(e);
        }
    }

}
```

Файл GeneratorUtil.java:

```
package com.andrew.mediastorage.server.util;

import java.security.SecureRandom;

public class GeneratorUtil {

    private static final String CHARS =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    public static final int DEFAULT_KEY_LENGTH = 64;

    public static String genRandStr(int length) {
        SecureRandom random = new SecureRandom();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < length; i++) {
            int index = random.nextInt(CHARS.length());
            sb.append(CHARS.charAt(index));
        }
        return sb.toString();
    }

}
```

Файл ImageUtil.java:

```
package com.andrew.mediastorage.server.util;

import java.awt.*;
import java.awt.image.BufferedImage;

public class ImageUtil {

    public static Image resize(Image image, int width, int height, int hints) {
        return image.getScaledInstance(width, height, hints);
    }

    public static Image resizeByWidth(Image image, int width, int hints) {
        return image.getScaledInstance(width, calcHeight(width, image.getWidth(null),
image.getHeight(null)), hints);
    }

    public static Image resizeByHeight(Image image, int height, int hints) {
        return image.getScaledInstance(calcWidth(height, image.getWidth(null),
image.getHeight(null)), height, hints);
    }
```

```
    public static BufferedImage toBufferedImage(Image image, boolean saveAlpha) {
        if (image instanceof BufferedImage) {
            return (BufferedImage) image;
        }

        int imageType = BufferedImage.TYPE_INT_RGB;
        if (saveAlpha) {
            imageType = BufferedImage.TYPE_INT_ARGB;
        }

        BufferedImage bufferedImage = new BufferedImage(image.getWidth(null),
image.getHeight(null), imageType);
        Graphics2D graphics2D = bufferedImage.createGraphics();
        graphics2D.drawImage(image, 0, 0, null);
        graphics2D.dispose();
        return bufferedImage;
    }

    public static int calcHeight(int width, int originWidth, int originHeight) {
        return (originHeight * width) / originWidth;
    }

    public static int calcWidth(int height, int originWidth, int originHeight) {
        return (originWidth * height) / originHeight;
    }

}
```

Файл ActivateMfaValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.validator.ActivateMfaValidator;
import com.andrew.mediastorage.server.api.validator.Validator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class ActivateMfaValidatorImpl implements ActivateMfaValidator {

    private Validator validator;

    @Override
    public FormStatusDto validate(String password, User user) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();

        invalidFieldsMap.putAll(validator.checkPassword(password, user.getPassword()));
        invalidFieldsMap.putAll(validator.checkTryActivateMfa(user.isMfaActive()));
```

```
        FormStatusDto formStatusDto = new FormStatusDto();
        if (invalidFieldsMap.size() > 0) {
            formStatusDto.setStatus(StatusCode.FAILURE);
            formStatusDto.setInvalidFieldsMap(invalidFieldsMap);
        } else {
            formStatusDto.setStatus(StatusCode.SUCCESS);
        }
        return formStatusDto;
    }

    @Autowired
    public void setValidator(Validator validator) {
        this.validator = validator;
    }

}
```

Файл DeactivateMfaValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.DeactivateMfaRequestDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.validator.DeactivateMfaValidator;
import com.andrew.mediastorage.server.api.validator.Validator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class DeactivateMfaValidatorImpl implements DeactivateMfaValidator {

    private Validator validator;

    @Override
    public FormStatusDto validate(DeactivateMfaRequestDto deactivateMfaRequestDto, User
user) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();


invalidFieldsMap.putAll(validator.checkPassword(deactivateMfaRequestDto.getPassword(),
user.getPassword()));

invalidFieldsMap.putAll(validator.checkMfaToken(deactivateMfaRequestDto.getMfaToken(),
user.getSecretKey(), user.getReserveTokenList()));
        invalidFieldsMap.putAll(validator.checkTryDeactivateMfa(user.isMfaActive()));

        FormStatusDto formStatusDto = new FormStatusDto();
        if (invalidFieldsMap.size() > 0) {
            formStatusDto.setStatus(StatusCode.FAILURE);
            formStatusDto.setInvalidFieldsMap(invalidFieldsMap);
```

```
        } else {
            formStatusDto.setStatus(StatusCode.SUCCESS);
        }
        return formStatusDto;
    }

    @Autowired
    public void setValidator(Validator validator) {
        this.validator = validator;
    }
}
```

Файл DeleteAccountValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.DeleteAccountRequestDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.entity.User;
import com.andrew.mediastorage.server.api.validator.DeleteAccountValidator;
import com.andrew.mediastorage.server.api.validator.Validator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class DeleteAccountValidatorImpl implements DeleteAccountValidator {

    private Validator validator;

    @Override
    public FormStatusDto validate(DeleteAccountRequestDto deleteAccountRequestDto, User
user) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();


invalidFieldsMap.putAll(validator.checkPassword(deleteAccountRequestDto.getPassword(),
user.getPassword()));
        if (user.isMfaActive()) {

invalidFieldsMap.putAll(validator.checkMfaToken(deleteAccountRequestDto.getMfaToken(),
user.getSecretKey(), user.getReserveTokenList()));
        }

        FormStatusDto formStatusDto = new FormStatusDto();
        if (invalidFieldsMap.size() > 0) {
            formStatusDto.setStatus(StatusCode.FAILURE);
            formStatusDto.setInvalidFieldsMap(invalidFieldsMap);
        } else {
            formStatusDto.setStatus(StatusCode.SUCCESS);
        }
        return formStatusDto;
```

```
    }

    @Autowired
    public void setValidator(Validator validator) {
        this.validator = validator;
    }

}
```

Файл RegistrationValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.RegistrationFormDto;
import com.andrew.mediastorage.server.api.validator.RegistrationValidator;
import com.andrew.mediastorage.server.api.validator.Validator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class RegistrationValidatorImpl implements RegistrationValidator {

    Validator validator;

    @Override
    public FormStatusDto validate(RegistrationFormDto registrationFormDto) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();


invalidFieldsMap.putAll(validator.validateFirstName(registrationFormDto.getFirstName()));

invalidFieldsMap.putAll(validator.validateLastName(registrationFormDto.getLastName()));
        invalidFieldsMap.putAll(validator.validateEmail(registrationFormDto.getEmail(),
false));

invalidFieldsMap.putAll(validator.validateUsername(registrationFormDto.getUsername(),
null, registrationFormDto.getEmail()));

invalidFieldsMap.putAll(validator.validatePassword(registrationFormDto.getPassword()));

        FormStatusDto formStatusDto = new FormStatusDto();
        if (invalidFieldsMap.size() > 0) {
            formStatusDto.setStatus(StatusCode.FAILURE);
            formStatusDto.setInvalidFieldsMap(invalidFieldsMap);
        } else {
            formStatusDto.setStatus(StatusCode.SUCCESS);
        }
        return formStatusDto;
    }
```

```
    @Autowired
    public void setValidator(Validator validator) {
        this.validator = validator;
    }

}
```

Файл ResetPasswordValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.dto.ResetPasswordRequestDto;
import com.andrew.mediastorage.server.api.validator.ResetPasswordValidator;
import com.andrew.mediastorage.server.api.validator.Validator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class ResetPasswordValidatorImpl implements ResetPasswordValidator {

    private Validator validator;

    @Override
    public FormStatusDto validate(ResetPasswordRequestDto resetPasswordRequestDto) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();


invalidFieldsMap.putAll(validator.validateRestoreKey(resetPasswordRequestDto.getRestoreKey
()));

invalidFieldsMap.putAll(validator.validatePassword(resetPasswordRequestDto.getNewPassword(
)));

        FormStatusDto formStatusDto = new FormStatusDto();
        if (invalidFieldsMap.size() > 0) {
            formStatusDto.setStatus(StatusCode.FAILURE);
            formStatusDto.setInvalidFieldsMap(invalidFieldsMap);
        } else {
            formStatusDto.setStatus(StatusCode.SUCCESS);
        }
        return formStatusDto;
    }

    @Autowired
    public void setValidator(Validator validator) {
        this.validator = validator;
    }

}
```

Файл RestoreValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.validator.RestoreValidator;
import com.andrew.mediastorage.server.api.validator.Validator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.HashMap;
import java.util.Map;

@Component
public class RestoreValidatorImpl implements RestoreValidator {

    private Validator validator;

    @Override
    public FormStatusDto validate(String email) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();

        invalidFieldsMap.putAll(validator.validateEmail(email, true));

        FormStatusDto formStatusDto = new FormStatusDto();
        if (invalidFieldsMap.size() > 0) {
            formStatusDto.setStatus(StatusCode.FAILURE);
            formStatusDto.setInvalidFieldsMap(invalidFieldsMap);
        } else {
            formStatusDto.setStatus(StatusCode.SUCCESS);
        }
        return formStatusDto;
    }

    @Autowired
    public void setValidator(Validator validator) {
        this.validator = validator;
    }

}
```

Файл SettingsValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.constants.StatusCode;
import com.andrew.mediastorage.server.api.dto.BasicSettingsDto;
import com.andrew.mediastorage.server.api.dto.FormStatusDto;
import com.andrew.mediastorage.server.api.validator.SettingsValidator;
import com.andrew.mediastorage.server.api.validator.Validator;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
```

```
import java.util.HashMap;
import java.util.Map;

@Component
public class SettingsValidatorImpl implements SettingsValidator {

    private Validator validator;

    @Override
    public FormStatusDto validateBasicSettings(BasicSettingsDto basicSettingsDto, String
previousUsername, String email) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();


invalidFieldsMap.putAll(validator.validateFirstName(basicSettingsDto.getFirstName()));

invalidFieldsMap.putAll(validator.validateLastName(basicSettingsDto.getLastName()));
        invalidFieldsMap.putAll(validator.validateUsername(basicSettingsDto.getUsername(),
previousUsername, email));

        FormStatusDto formStatusDto = new FormStatusDto();
        if (invalidFieldsMap.size() > 0) {
            formStatusDto.setStatus(StatusCode.FAILURE);
            formStatusDto.setInvalidFieldsMap(invalidFieldsMap);
        } else {
            formStatusDto.setStatus(StatusCode.SUCCESS);
        }
        return formStatusDto;
    }

    @Autowired
    public void setValidator(Validator validator) {
        this.validator = validator;
    }
}
```

Файл ValidatorImpl.java:

```
package com.andrew.mediastorage.server.validator;

import com.andrew.mediastorage.server.api.constants.FormField;
import com.andrew.mediastorage.server.api.entity.RegistrationRequest;
import com.andrew.mediastorage.server.api.entity.ReserveToken;
import com.andrew.mediastorage.server.api.entity.RestoreRequest;
import com.andrew.mediastorage.server.api.provider.MFATokenProvider;
import com.andrew.mediastorage.server.api.service.RegistrationRequestService;
import com.andrew.mediastorage.server.api.service.RestoreRequestService;
import com.andrew.mediastorage.server.api.service.UserService;
import com.andrew.mediastorage.server.api.validator.Validator;
import com.andrew.mediastorage.server.property.FieldsValidationProperties;
import com.andrew.mediastorage.server.property.ResourcesProperties;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Component;
```

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Component
public class ValidatorImpl implements Validator {

    private UserService userService;
    private RegistrationRequestService registrationRequestService;
    private RestoreRequestService restoreRequestService;

    private MFATokenProvider mfaTokenProvider;

    private PasswordEncoder passwordEncoder;

    private ResourcesProperties resourcesProperties;
    private FieldsValidationProperties fieldsValidationProperties;


    @Override
    public Map<FormField, String> validateFirstName(String firstName) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (firstName == null) {
            invalidFieldsMap.put(FormField.FIRST_NAME,
resourcesProperties.getNullFieldErrorMsg());
        } else if (!firstName.matches(fieldsValidationProperties.getFirstNameRegEx())) {
            invalidFieldsMap.put(FormField.FIRST_NAME,
resourcesProperties.getForbiddenFirstNameErrorMsg());
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> validateLastName(String lastName) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (lastName == null) {
            invalidFieldsMap.put(FormField.LAST_NAME,
resourcesProperties.getNullFieldErrorMsg());
        } else if (!lastName.matches(fieldsValidationProperties.getLastNameRegEx())) {
            invalidFieldsMap.put(FormField.LAST_NAME,
resourcesProperties.getForbiddenLastNameErrorMsg());
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> validateEmail(String email, boolean mustExist) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (email == null) {
            invalidFieldsMap.put(FormField.EMAIL,
resourcesProperties.getNullFieldErrorMsg());
        } else if (!email.contains("@")) {
            invalidFieldsMap.put(FormField.EMAIL,
resourcesProperties.getInvalidEmailErrorMsg());
        } else {
            if (userService.findByEmail(email) != null) {
```

```
                if (!mustExist) {
                    invalidFieldsMap.put(FormField.EMAIL,
resourcesProperties.getBusyEmailErrorMsg());
                }
            } else {
                if (mustExist) {
                    invalidFieldsMap.put(FormField.EMAIL,
resourcesProperties.getMissingAccountByEmailErrorMsg());
                }
            }
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> validateUsername(String username, String
previousUsername, String email) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (username == null) {
            invalidFieldsMap.put(FormField.USERNAME,
resourcesProperties.getNullFieldErrorMsg());
        } else if (email == null) {
            invalidFieldsMap.put(FormField.EMAIL,
resourcesProperties.getNullFieldErrorMsg());
        } else if (!username.equals(previousUsername)) {
            if (!username.matches(fieldsValidationProperties.getUsernameRegEx())) {
                invalidFieldsMap.put(FormField.USERNAME,
resourcesProperties.getForbiddenUsernameErrorMsg());
            } else if
(getSplitValues(fieldsValidationProperties.getReservedUsernames()).contains(username)) {
                invalidFieldsMap.put(FormField.USERNAME,
resourcesProperties.getReservedUsernameErrorMsg());
            } else if (userService.findByUsername(username) != null) {
                invalidFieldsMap.put(FormField.USERNAME,
resourcesProperties.getBusyUsernameErrorMsg());
            } else {
                RegistrationRequest registrationRequest =
registrationRequestService.findByUsername(username);
                if (registrationRequest != null &&
!registrationRequest.getEmail().equals(email)) {
                    invalidFieldsMap.put(FormField.USERNAME,
resourcesProperties.getBusyUsernameErrorMsg());
                }
            }
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> validatePassword(String password) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (password == null) {
            invalidFieldsMap.put(FormField.PASSWORD,
resourcesProperties.getNullFieldErrorMsg());
        } else if (!password.matches(fieldsValidationProperties.getPasswordRegEx())) {
            invalidFieldsMap.put(FormField.PASSWORD,
resourcesProperties.getForbiddenPasswordErrorMsg());
```

```
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> checkPassword(String password, String passwordHash) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (password == null) {
            invalidFieldsMap.put(FormField.PASSWORD,
resourcesProperties.getNullFieldErrorMsg());
        } else if (!passwordEncoder.matches(password, passwordHash)) {
            invalidFieldsMap.put(FormField.PASSWORD,
resourcesProperties.getWrongPasswordErrorMsg());
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> checkTryActivateMfa(boolean mfaActive) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (mfaActive) {
            invalidFieldsMap.put(FormField.ALL_FIELDS,
resourcesProperties.getMfaIsAlreadyActivatedErrorMsg());
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> checkTryDeactivateMfa(boolean mfaActive) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (!mfaActive) {
            invalidFieldsMap.put(FormField.ALL_FIELDS,
resourcesProperties.getMfaIsAlreadyDeactivatedErrorMsg());
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> checkMfaToken(String mfaToken, String secretKey,
List<ReserveToken> reserveTokenList) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
        if (!mfaTokenProvider.verifyTotp(mfaToken, secretKey)) {
            ReserveToken selectedReserveToken = reserveTokenList.stream()
                    .filter(reserveToken -> reserveToken.getValue().equals(mfaToken))
                    .findFirst().orElse(null);
            if (selectedReserveToken == null) {
                invalidFieldsMap.put(FormField.MFA_TOKEN,
resourcesProperties.getInvalidMfaTokenErrorMsg());
            }
        }
        return invalidFieldsMap;
    }

    @Override
    public Map<FormField, String> validateRestoreKey(String restoreKey) {
        Map<FormField, String> invalidFieldsMap = new HashMap<>();
```

```
        RestoreRequest restoreRequest =
restoreRequestService.findByRestoreKey(restoreKey);
        if (restoreRequest == null) {
            invalidFieldsMap.put(FormField.RESTORE_KEY,
resourcesProperties.getInvalidRestoreKeyErrorMsg());
        }
        return invalidFieldsMap;
    }

    private List<String> getSplitValues(String property) {
        return Arrays.asList(property.split(fieldsValidationProperties.getSeparator()));
    }

    @Autowired
    public void setUserService(UserService userService) {
        this.userService = userService;
    }

    @Autowired
    public void setRegistrationRequestService(RegistrationRequestService
registrationRequestService) {
        this.registrationRequestService = registrationRequestService;
    }

    @Autowired
    public void setRestoreRequestService(RestoreRequestService restoreRequestService) {
        this.restoreRequestService = restoreRequestService;
    }

    @Autowired
    public void setMfaTokenProvider(MFATokenProvider mfaTokenProvider) {
        this.mfaTokenProvider = mfaTokenProvider;
    }

    @Autowired
    public void setPasswordEncoder(PasswordEncoder passwordEncoder) {
        this.passwordEncoder = passwordEncoder;
    }

    @Autowired
    public void setResourcesProperties(ResourcesProperties resourcesProperties) {
        this.resourcesProperties = resourcesProperties;
    }

    @Autowired
    public void setFieldsValidationProperties(FieldsValidationProperties
fieldsValidationProperties) {
        this.fieldsValidationProperties = fieldsValidationProperties;
    }

}
```

Файл application.properties:

```
spring.datasource.url=jdbc:mysql://localhost:3306/media_storage?createDatabaseIfNotExist=t
rue
```

```
spring.datasource.username=root
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.open-in-view=false

spring.mail.host=smtp.gmail.com
spring.mail.port=587
spring.mail.username=media.storage.official@gmail.com
spring.mail.password=rmaxdafbshyxrvmt
spring.mail.properties.mail.smtp.auth=true
spring.mail.properties.mail.smtp.starttls.enable=true

spring.servlet.multipart.max-file-size=40MB
spring.servlet.multipart.max-request-size=800MB

# logging.level.org.springframework=DEBUG
logging.level.com.andrew.mediastorage.server=DEBUG
```

Файл fieldsValidation.properties:

```
fields-validation.firstNameRegEx=[a-zA-Za-яA-ЯiIїЇєЄґҐ-]{2,15}
fields-validation.lastNameRegEx=[a-zA-Za-яA-ЯiIїЇєЄґҐ-]{2,15}
fields-validation.usernameRegEx=[a-zA-Z0-9_]{2,20}
fields-validation.passwordRegEx=.{8,}
fields-
validation.reservedUsernames=resetPassword,confirm,restore,0,about,access,account,accounts
,activate,activities,activity,ad,add,address,adm,admin,administration,administrator,ads,ad
ult,advertising,affiliate,affiliates,ajax,all,alpha,analysis,analytics,android,anon,anonym
ous,api,app,apps,archive,archives,article,asct,asset,atom,auth,authentication,avatar,backu
p,balancer-
manager,banner,banners,beta,billing,bin,blog,blogs,board,book,bookmark,bot,bots,bug,busine
ss,cache,cadastro,calendar,call,campaign,cancel,captcha,career,careers,cart,categories,cat
egory,cgi,cgi-
bin,changelog,chat,check,checking,checkout,client,cliente,clients,code,codereview,comercia
l,comment,comments,communities,community,company,compare,compras,config,configuration,conn
ect,contact,contact-
us,contact_us,contactus,contest,contribute,corp,create,css,dashboard,data,db,default,delet
e,demo,design,designer,destroy,dev,devel,developer,developers,diagram,diary,dict,dictionar
y,die,dir,direct_messages,directory,dist,doc,docs,documentation,domain,download,downloads,
ecommerce,edit,editor,edu,education,email,employment,empty,end,enterprise,entries,entry,er
ror,errors,eval,event,everyone,exit,explore,facebook,faq,favorite,favorites,feature,featur
es,feed,feedback,feeds,file,files,first,flash,fleet,fleets,flog,follow,followers,following
,forgot,form,forum,forums,founder,free,friend,friends,ftp,gadget,gadgets,game,games,get,gh
ost,gift,gifts,gist,github,graph,group,groups,guest,guests,help,home,homepage,host,hosting
,hostmaster,hostname,howto,hpg,html,http,httpd,https,i,iamges,icon,icons,id,idea,ideas,ima
ge,images,imap,img,index,indice,info,information,inquiry,instagram,intranet,invitations,in
vite,ipad,iphone,irc,is,issue,issues,it,item,items,java,javascript,job,jobs,join,js,json,j
ump,knowledgebase,language,languages,last,ldap-
status,legal,license,link,links,linux,list,lists,log,log-in,log-
out,log_in,log_out,login,logout,logs,m,mac,mail,mail1,mail2,mail3,mail4,mail5,mailer,maili
ng,maintenance,manager,manual,map,maps,marketing,master,me,media,member,members,message,me
ssages,messenger,microblog,microblogs,mine,mis,mob,mobile,movie,movies,mp3,msg,msn,music,m
usicas,mx,my,mysql,name,named,nan,navi,navigation,net,network,new,news,newsletter,nick,nic
```

```
kname,notes,noticias,notification,notifications,notify,ns,ns1,ns10,ns2,ns3,ns4,ns5,ns6,ns7
,ns8,ns9,null,oauth,oauth_clients,offer,offers,official,old,online,openid,operator,order,o
rders,organization,organizations,overview,owner,owners,page,pager,pages,panel,password,pay
ment,perl,phone,photo,photoalbum,photos,php,phpmyadmin,phppgadmin,phpredisadmin,pic,pics,p
ing,plan,plans,plugin,plugins,policy,pop,pop3,popular,portal,post,postfix,postmaster,posts
,pr,premium,press,price,pricing,privacy,privacy-
policy,privacy_policy,privacypolicy,private,product,products,profile,project,projects,prom
o,pub,public,purpose,put,python,query,random,ranking,read,readme,recent,recruit,recruitmen
t,register,registration,release,remove,replies,report,reports,repositories,repository,req,
request,requests,reset,roc,root,rss,ruby,rule,sag,sale,sales,sample,samples,save,school,sc
ript,scripts,search,secure,security,self,send,server,server-info,server-
status,service,services,session,sessions,setting,settings,setup,share,shop,show,sign-
in,sign-
up,sign_in,sign_up,signin,signout,signup,site,sitemap,sites,smartphone,smtp,soporte,source
,spec,special,sql,src,ssh,ssl,ssladmin,ssladministrator,sslwebmaster,staff,stage,staging,s
tart,stat,state,static,stats,status,store,stores,stories,style,styleguide,stylesheet,style
sheets,subdomain,subscribe,subscriptions,suporte,support,svn,swf,sys,sysadmin,sysadministr
ator,system,tablet,tablets,tag,talk,task,tasks,team,teams,tech,telnet,term,terms,terms-of-
service,terms_of_service,termsofservice,test,test1,test2,test3,teste,testing,tests,theme,t
hemes,thread,threads,tmp,todo,tool,tools,top,topic,topics,tos,tour,translations,trends,tut
orial,tux,tv,twitter,undef,unfollow,unsubscribe,update,upload,uploads,url,usage,user,usern
ame,users,usuario,vendas,ver,version,video,videos,visitor,watch,weather,web,webhook,webhoo
ks,webmail,webmaster,website,websites,welcome,widget,widgets,wiki,win,windows,word,work,wo
rks,workshop,ww,wws,www,www1,www2,www3,www4,www5,www6,www7,wwws,wwww,xfn,xml,xmpp,xpg,xxx,
yaml,year,yml,you,yourdomain,yourname,yoursite,yourusername
fields-validation.separator=,
```

Файл jwt.properties:

```
jwt.secretKey=secret key :)
# 10 min (time in ms)
jwt.accessTokenExpirationTime=600000
# 12 hour (time in ms)
jwt.refreshTokenExpirationTime=43200000
```

Файл log4j2.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="WARN">
    <Properties>
        <Property name="pattern">%d{yyyy-MM-dd HH:mm:ss.SSS} %-5p {%t} [%c{1}] -
%m%n</Property>
        <Property name="log.dir">./log</Property>
    </Properties>
    <Appenders>
        <Console name="system-out" target="SYSTEM_OUT">
            <ThresholdFilter level="warn" onMatch="DENY" onMismatch="NEUTRAL"/>
            <PatternLayout pattern="${pattern}"/>
        </Console>
        <Console name="system-err" target="SYSTEM_ERR">
            <ThresholdFilter level="warn" onMatch="NEUTRAL" onMismatch="DENY"/>
            <PatternLayout pattern="${pattern}"/>
        </Console>
        <RollingFile name="file-out-debug"
                     fileName="${log.dir}/log-debug.log"
```

```
                 filePattern="${log.dir}/log-debug.log-%d{yyyyMMddHHmmss}.zip"
                 append="true">
            <Policies>
                <SizeBasedTriggeringPolicy size="50MB"/>
            </Policies>
            <DefaultRolloverStrategy max="100">
                <Delete basePath="${log.dir}">
                    <IfFileName glob="log-debug.log*.zip">
                        <IfAny>
                            <IfLastModified age="10d"/>
                            <IfAccumulatedFileCount exceeds="100"/>
                        </IfAny>
                    </IfFileName>
                </Delete>
            </DefaultRolloverStrategy>
            <ThresholdFilter level="DEBUG" onMatch="NEUTRAL" onMismatch="DENY"/>
            <PatternLayout pattern="${pattern}"/>
        </RollingFile>
        <RollingFile name="file-out-info"
                     fileName="${log.dir}/log-info.log"
                     filePattern="${log.dir}/log-info.log-%d{yyyyMMddHHmmss}.zip"
                     append="true">
            <Policies>
                <SizeBasedTriggeringPolicy size="50MB"/>
            </Policies>
            <DefaultRolloverStrategy max="100">
                <Delete basePath="${log.dir}">
                    <IfFileName glob="log-info.log*.zip">
                        <IfAny>
                            <IfLastModified age="10d"/>
                            <IfAccumulatedFileCount exceeds="100"/>
                        </IfAny>
                    </IfFileName>
                </Delete>
            </DefaultRolloverStrategy>
            <ThresholdFilter level="INFO" onMatch="NEUTRAL" onMismatch="DENY"/>
            <PatternLayout pattern="${pattern}"/>
        </RollingFile>
    </Appenders>
    <Loggers>
        <Root level="info">
            <AppenderRef ref="system-out"/>
            <AppenderRef ref="system-err"/>
            <AppenderRef ref="file-out-debug"/>
            <AppenderRef ref="file-out-info"/>
        </Root>
    </Loggers>
</Configuration>
```

Файл media.properties:

```
media.previewImageHeight=200
media.mediaListBlockSize=100
```

Файл resources.properties:

```
resources.forbiddenFirstNameErrorMsg=Заборонене ім'я!
resources.forbiddenLastNameErrorMsg=Заборонене прізвище!
resources.invalidEmailErrorMsg=Недійсна електронна адреса!
resources.busyEmailErrorMsg=Email занятий!
resources.missingAccountByEmailErrorMsg=Обліковий запис з цією електронною поштою
відсутній!
resources.forbiddenUsernameErrorMsg=Заборонений username!
resources.reservedUsernameErrorMsg=Зарезервоване значення!
resources.busyUsernameErrorMsg=Username занятий!
resources.forbiddenPasswordErrorMsg=Мінімальна довжина паролю 8 символів!
resources.wrongPasswordErrorMsg=Хибний пароль!
resources.mfaIsAlreadyActivatedErrorMsg=Двофакторна аутентифікація вже активована!
resources.mfaIsAlreadyDeactivatedErrorMsg=Двофакторна аутентифікація викнена!
resources.invalidMfaTokenErrorMsg=Хибний MFA токен!
resources.invalidRestoreKeyErrorMsg=Хибне посилання для зміни паролю!
resources.nullFieldErrorMsg=Це поле не може бути null!
resources.confirmMailSubjectText=Підтвердження реєстрації акаунту!
resources.confirmMailBodyText=Для підтвердження реєстрації акаунту перейдіть за цільовим
посиланням: %s
resources.restoreMailSubjectText=Зміна паролю!
resources.restoreMailBodyText=Для зміни паролю до акаунту перейдіть за цільовим посиланням
та вкажіть новий пароль: %s
```

Файл storage.properties:

```
storage.previewMediaFolderPath=D:/Coursework/storage/preview
storage.mediaFolderPath=D:/Coursework/storage/media
storage.avatarFolderPath=D:/Coursework/storage/avatar
```

Файл system.properties:

```
system.clientServerUrl=http://localhost:4200
```

Додаток 2

Файли клієнтської частини проекту:

Файл activate-mfa-dialog.component.html:

```
<div class="dialog-background">
  <div class="dialog-container">
    <div *ngIf="!showActivateMfaSuccessfulMessage; else activateMfaSuccessfulMessage">
      <form #form="ngForm" (ngSubmit)="onSubmit(form)">
        <div class="def-title text-center">
          <h5>Активувати двофакторну автентифікацію</h5>
        </div>
        <div class="input-field">
          <input type="password" name="password" class="form-control shadow-none def-
input" placeholder="Пароль"
                 ngModel #password="ngModel"
                 [class.is-invalid]="password.touched && password.invalid"
                 [class.is-valid]="password.touched && password.valid"
```

```
                 autocomplete="off" required>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(password.name) &&
password.invalid">{{errorMessages.get(password.name)}}</div>
        </div>
        <div class="input-field">
          <div class="row">
            <div class="col-6">
              <button type="button" class="btn back-btn" (click)="cancel()">
                Скасувати
              </button>
            </div>
            <div class="col-6">
              <button type="submit" class="btn def-btn">
                Активувати
              </button>
            </div>
          </div>
        </div>
      </form>
    </div>
    <ng-template #activateMfaSuccessfulMessage>
      <div style="width: 310px">
        <div class="input-field">
          Відскануйте QR-код для генерації MFA токенів для входу.
        </div>
        <div class="input-field d-flex justify-content-center">
          <div class="col-9">
            <img class="img-fluid" src="{{qrCode}}">
          </div>
        </div>
        <div class="input-field">
          <span class="text-break">{{secretKey}}</span>
        </div>
        <div class="input-field text-center">
          Резервні токени:
        </div>
        <div class="input-field">
          <ol class="two-columns text-center">
            <li *ngFor="let reserveToken of reserveTokenList">{{reserveToken}}</li>
          </ol>
        </div>
        <div class="input-field d-flex justify-content-center">
          <div class="col-5">
            <input type="button" value="Закрити" class="btn back-btn" (click)="close()">
          </div>
        </div>
      </div>
    </ng-template>
  </div>
</div>
```

Файл activate-mfa-dialog.component.ts:

```
import {Component} from '@angular/core';
import {MatDialogRef} from "@angular/material/dialog";
```

```
import {AuthService} from "../shared/service/auth.service";
import {StatusCodes} from "../shared/constants/status-codes";
import CommonUtil from "../shared/util/common-util";
import {NgForm} from "@angular/forms";

@Component({
  selector: 'app-activate-mfa-dialog',
  templateUrl: './activate-mfa-dialog.component.html',
  styleUrls: ['./activate-mfa-dialog.component.scss']
})
export class ActivateMfaDialogComponent {

  qrCode!: string
  secretKey!: string
  reserveTokenList!: string[]

  showActivateMfaSuccessfulMessage = false

  errorMessages = new Map<string, string>();

  constructor(
    private dialogRef: MatDialogRef<ActivateMfaDialogComponent>,
    private authService: AuthService,
  ) {
  }

  onSubmit(form: NgForm) {
    if (form.valid) {
      this.authService.activateMfa(form.value.password).subscribe({
        next: data => {
          let formStatus = data.formStatusDto
          if (formStatus.status == StatusCodes.FAILURE) {
            this.errorMessages = CommonUtil.updateForm(form, formStatus)
          } else if (formStatus.status == StatusCodes.SUCCESS) {
            this.qrCode = data.qrCode
            this.secretKey = data.secretKey
            this.reserveTokenList = data.reserveTokenList
            this.showActivateMfaSuccessfulMessage = true
          } else {
            this.dialogRef.close()
          }
        },
        error: () => {
          this.dialogRef.close()
        }
      })
    } else {
      this.setFormFieldsTouched(form, true)
    }
  }

  setFormFieldsTouched(form: any, type: boolean) {
    form.controls.password.touched = type
  }

  close() {
    this.dialogRef.close(true)
```

```
  }

  cancel() {
    this.dialogRef.close()
  }

}
```

Файл app-routing.module.ts:

```
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from '@angular/router';
import {UserComponent} from "./user/user.component";
import {LoginComponent} from "./login/login.component";
import {SettingsComponent} from "./settings/settings.component";
import {RegistrationComponent} from "./registration/registration.component";
import {ConfirmComponent} from "./confirm/confirm.component";
import {RestoreComponent} from "./restore/restore.component";
import {ResetPasswordComponent} from "./reset-password/reset-password.component";
import {MessageComponent} from "./message/message.component";

const routes: Routes = [
  {path: '', component: LoginComponent},
  {path: 'registration', component: RegistrationComponent},
  {path: 'confirm/:key', component: ConfirmComponent},
  {path: 'restore', component: RestoreComponent},
  {path: 'resetPassword/:key', component: ResetPasswordComponent},
  {path: 'users/:username', component: UserComponent},
  {path: 'settings', component: SettingsComponent},
  {path: 'message', component: MessageComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {
}
```

Файл app.component.html:

```
<router-outlet></router-outlet>
```

Файл app.component.ts:

```
import {Component} from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
```

```
}
```

Файл app.module.ts:

```
import {NgModule} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';

import {AppRoutingModule} from './app-routing.module';
import {AppComponent} from './app.component';
import {LAZYLOAD_IMAGE_HOOKS, LazyLoadImageModule} from "ng-lazyload-image";
import {HTTP_INTERCEPTORS, HttpClientModule} from "@angular/common/http";
import {UserComponent} from './user/user.component';
import {IsVisibleDirective} from './shared/directive/is-visible.directive';
import {HeaderComponent} from './header/header.component';
import {LoginComponent} from './login/login.component';
import {SettingsComponent} from './settings/settings.component';
import {JWT_OPTIONS, JwtHelperService, JwtModule} from "@auth0/angular-jwt";
import {AuthInterceptor} from "./shared/interceptor/auth.interceptor";
import {LazyLoadImageHooks} from "./shared/hooks/lazy-load-image-hooks";
import {FormsModule} from "@angular/forms";
import {RegistrationComponent} from './registration/registration.component';
import {ConfirmComponent} from './confirm/confirm.component';
import {RestoreComponent} from './restore/restore.component';
import {ResetPasswordComponent} from './reset-password/reset-password.component';
import {MessageComponent} from './message/message.component';
import {MatDialogModule} from "@angular/material/dialog";
import { DeleteAccountDialogComponent } from './delete-account-dialog/delete-account-
dialog.component';
import { ActivateMfaDialogComponent } from './activate-mfa-dialog/activate-mfa-
dialog.component';
import { DeactivateMfaDialogComponent } from './deactivate-mfa-dialog/deactivate-mfa-
dialog.component';
import {NgOtpInputModule} from "ng-otp-input";

@NgModule({
  declarations: [
    AppComponent,
    UserComponent,
    IsVisibleDirective,
    HeaderComponent,
    LoginComponent,
    SettingsComponent,
    RegistrationComponent,
    ConfirmComponent,
    RestoreComponent,
    ResetPasswordComponent,
    MessageComponent,
    DeleteAccountDialogComponent,
    ActivateMfaDialogComponent,
    DeactivateMfaDialogComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
```

```
    JwtModule,
    LazyLoadImageModule,
    FormsModule,
    MatDialogModule,
    NgOtpInputModule
  ],
  providers: [
    {provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true},
    {provide: JWT_OPTIONS, useValue: JWT_OPTIONS},
    {provide: LAZYLOAD_IMAGE_HOOKS, useClass: LazyLoadImageHooks},
    JwtHelperService
  ],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Файл confirm.component.ts:

```
import {Component} from '@angular/core';
import {ActivatedRoute, Params, Router} from "@angular/router";
import {AuthService} from "../shared/service/auth.service";
import {MessageDataService} from "../shared/service/message-data.service";

@Component({
  selector: 'app-confirm',
  templateUrl: './confirm.component.html',
  styleUrls: ['./confirm.component.scss']
})
export class ConfirmComponent {

  constructor(
    private router: Router,
    private activatedRoute: ActivatedRoute,
    private authService: AuthService,
    private messageDataService: MessageDataService
  ) {
  }

  async ngOnInit() {
    if (this.authService.getAuthCredentials()) {
      await this.router.navigate([`users/${this.authService.getUsername()}`])
      return
    }
    this.activatedRoute.params.subscribe((params: Params) => {
      this.authService.confirm(params['key']).subscribe({
        next: data => {
          this.messageDataService.title = "Вітаю!"
          this.messageDataService.text = "Акаунт успішно створено."
          this.router.navigate(['/message'])
        },
        error: error => {
          this.router.navigate([''])
        }
      })
    })
```

```
  }

}
```

Файл deactivate-mfa-dialog.component.html:

```
<div class="dialog-background">
  <div class="dialog-container">
    <div *ngIf="!showDeactivateMfaSuccessfulMessage; else deactivateMfaSuccessfulMessage"
style="width: 263px">
      <form #form="ngForm" (ngSubmit)="onSubmit(form)">
        <div class="def-title text-center">
          <h5>Вимкнути двофакторну автентифікацію</h5>
        </div>
        <div>
          <div class="invalid-feedback"
               [class.d-
block]="errorMessages.has('allFields')">{{errorMessages.get('allFields')}}</div>
        </div>
        <div class="input-field">
          <input type="password" name="password" placeholder="Пароль" class="form-control
shadow-none def-input"
                 [(ngModel)]="deactivateMfaRequest.password" #password="ngModel"
                 autocomplete="off" required>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(password.name) &&
password.invalid">{{errorMessages.get(password.name)}}</div>
        </div>
        <div class="input-field text-center">
          MFA token
        </div>
        <div class="input-field">
          <div>
            <input type="text" name="mfaToken"
                   [(ngModel)]="deactivateMfaRequest.mfaToken" #mfaToken="ngModel"
                   autocomplete="off" maxlength="6" minlength="6" hidden required>
          </div>
          <div class="d-flex justify-content-center">
            <ng-otp-input (onInputChange)="onOtpChange($event)"
[config]="otpInputConfig"></ng-otp-input>
          </div>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(mfaToken.name) &&
mfaToken.invalid">{{errorMessages.get(mfaToken.name)}}</div>
        </div>
        <div class="input-field">
          <div class="row">
            <div class="col-6">
              <button type="button" class="btn back-btn" (click)="cancel()">
                Скасувати
              </button>
            </div>
            <div class="col-6">
              <button type="submit" class="btn def-btn">
                Вимкнути
              </button>
```

```
            </div>
          </div>
        </div>
      </form>
    </div>
  </div>
  <ng-template #deactivateMfaSuccessfulMessage>
    <div>
      <div class="def-title text-center">
        <h5>Двофакторну автентифікацію вимкнено!</h5>
      </div>
      <div class="input-field d-flex justify-content-center">
        <div class="col-5">
          <input type="button" value="Закрити" class="btn back-btn" (click)="close()">
        </div>
      </div>
    </div>
  </ng-template>
</div>
```

Файл deactivate-mfa-dialog.component.ts:

```
import {Component, ViewChild} from '@angular/core';
import {MatDialogRef} from "@angular/material/dialog";
import {AuthService} from "../shared/service/auth.service";
import {NgForm} from "@angular/forms";
import {StatusCodes} from "../shared/constants/status-codes";
import CommonUtil from "../shared/util/common-util";
import {DeactivateMfaRequest} from "../shared/model/deactivate-mfa-request";
import {NgOtpInputComponent, NgOtpInputConfig} from "ng-otp-input";

@Component({
  selector: 'app-deactivate-mfa-dialog',
  templateUrl: './deactivate-mfa-dialog.component.html',
  styleUrls: ['./deactivate-mfa-dialog.component.scss']
})
export class DeactivateMfaDialogComponent {

  deactivateMfaRequest = new DeactivateMfaRequest()
  showDeactivateMfaSuccessfulMessage = false

  errorMessages = new Map<string, string>();

  otpInputConfig: NgOtpInputConfig = {
    length: 6,
    isPasswordInput: true,
    allowNumbersOnly: true,
    containerClass: "d-flex",
    inputClass: "form-control shadow-none otp-input test",
  }

  constructor(
    private dialogRef: MatDialogRef<DeactivateMfaDialogComponent>,
    private authService: AuthService,
  ) {
  }
```

```
  onSubmit(form: NgForm) {
    if (form.valid) {
      this.authService.deactivateMfa(this.deactivateMfaRequest).subscribe({
        next: data => {
          if (data.status == StatusCodes.FAILURE) {
            this.errorMessages = CommonUtil.updateForm(form, data)
          } else if (data.status == StatusCodes.SUCCESS) {
            this.showDeactivateMfaSuccessfulMessage = true
          } else {
            this.dialogRef.close()
          }
        },
        error: () => {
          this.dialogRef.close()
        }
      })
    } else {
      this.setFormFieldsTouched(form, true)
    }
  }

  setFormFieldsTouched(form: any, type: boolean) {
    form.controls.password.touched = type
    form.controls.mfaToken.touched = type
  }

  onOtpChange(value: string) {
    this.deactivateMfaRequest.mfaToken = value
  }

  close() {
    this.dialogRef.close(true)
  }

  cancel() {
    this.dialogRef.close()
  }

}
```

Файл delete-account-dialog.component.html:

```
<div class="dialog-background">
  <div class="dialog-container">
    <form #form="ngForm" (ngSubmit)="onSubmit(form)">
      <div class="def-title text-center">
        <h5>Видалити обліковий запис</h5>
      </div>
      <div class="input-field">
        <input type="password" name="password" placeholder="Пароль" class="form-control
shadow-none def-input"
               [(ngModel)]="deleteAccountRequest.password" #password="ngModel"
               autocomplete="off" required>
        <div class="invalid-feedback"
```

```
            [class.d-block]="errorMessages.has(password.name) &&
password.invalid">{{errorMessages.get(password.name)}}</div>
      </div>
      <div *ngIf="mfaActive" class="input-field">
        <div class="input-field text-center">
          MFA token
        </div>
        <div>
          <input type="text" name="mfaToken"
                 [(ngModel)]="deleteAccountRequest.mfaToken" #mfaToken="ngModel"
                 autocomplete="off" maxlength="6" minlength="6" hidden required>
        </div>
        <div class="d-flex justify-content-center">
          <ng-otp-input (onInputChange)="onOtpChange($event)"
[config]="otpInputConfig"></ng-otp-input>
        </div>
        <div class="invalid-feedback"
            [class.d-block]="errorMessages.has(mfaToken.name) &&
mfaToken.invalid">{{errorMessages.get(mfaToken.name)}}</div>
      </div>
      <div class="input-field">
        <div class="row">
          <div class="col-6">
            <button type="button" class="btn back-btn" (click)="cancel()">
              Скасувати
            </button>
          </div>
          <div class="col-6">
            <button type="submit" class="btn def-btn">
              Видалити
            </button>
          </div>
        </div>
      </div>
    </form>
  </div>
</div>
```

Файл delete-account-dialog.component.ts:

```
import {Component, Inject} from '@angular/core';
import {MAT_DIALOG_DATA, MatDialogRef} from "@angular/material/dialog";
import {AuthService} from "../shared/service/auth.service";
import {DeleteAccountRequest} from "../shared/model/delete-account-request";
import {StatusCodes} from "../shared/constants/status-codes";
import CommonUtil from "../shared/util/common-util";
import {NgForm} from "@angular/forms";
import {NgOtpInputConfig} from "ng-otp-input";

@Component({
  selector: 'app-delete-account-dialog',
  templateUrl: './delete-account-dialog.component.html',
  styleUrls: ['./delete-account-dialog.component.scss']
})
export class DeleteAccountDialogComponent {
```

```
  deleteAccountRequest = new DeleteAccountRequest()
  mfaActive: boolean

  errorMessages = new Map<string, string>();

  otpInputConfig: NgOtpInputConfig = {
    length: 6,
    isPasswordInput: true,
    allowNumbersOnly: true,
    containerClass: "d-flex",
    inputClass: "form-control shadow-none otp-input test",
  }

  constructor(
    private dialogRef: MatDialogRef<DeleteAccountDialogComponent>,
    @Inject(MAT_DIALOG_DATA) mfaActive: boolean,
    private authService: AuthService,
  ) {
    this.mfaActive = mfaActive
  }

  onSubmit(form: NgForm) {
    if (form.valid) {
      this.authService.deleteAccount(this.deleteAccountRequest).subscribe({
        next: data => {
          if (data.status == StatusCodes.FAILURE) {
            this.errorMessages = CommonUtil.updateForm(form, data)
          } else if (data.status == StatusCodes.SUCCESS) {
            this.dialogRef.close(true)
          } else {
            this.dialogRef.close()
          }
        },
        error: () => {
        }
      })
    } else {
      this.setFormFieldsTouched(form, true)
    }
  }

  setFormFieldsTouched(form: any, type: boolean) {
    form.controls.password.touched = type
    if (this.mfaActive) {
      form.controls.mfaToken.touched = type
    }
  }

  onOtpChange(value: any) {
    this.deleteAccountRequest.mfaToken = value
  }

  cancel() {
    this.dialogRef.close()
  }

}
```

Файл header.component.html:

```
<div class="header">
  <div class="logo-panel">
    <span style="color: #FAFAFA; font-size: 30px">LOGO</span>
  </div>
  <div class="tabs-panel">
    <a *ngIf="showSettingsPage()" [routerLink]="['/settings']" class="tab">
      <div class="tab-box">Налаштування</div>
    </a>
    <a *ngIf="showUserPage()" [routerLink]="['/users', username]" class="tab">
      <div class="tab-box">Галерея</div>
    </a>
    <button class="tab" (click)="logout()">Вихід</button>
  </div>
</div>
```

Файл header.component.scss:

```
.header {
  display: flex;
  position: fixed;
  left: 0;
  right: 0;
  top: 0;
  z-index: 100;
  background-color: #212932;
  height: 56px;
}

.logo-panel {
  width: 15%;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
}

.tabs-panel {
  width: 85%;
  height: 100%;
  display: flex;
  justify-content: flex-end;
  padding-right: 30px;
}

.tabs-panel a {
  text-decoration: none;
}

.tab {
  height: 100%;
  padding-inline: 15px;
  display: flex;
```

```
  align-items: center;
  justify-content: center;
  color: #FAFAFA;
  background-color: transparent;
  font-size: 15px;
  border: 0 solid transparent;
  cursor: pointer;
}

.tab:hover {
  background-color: #2d3744;
}

.tab:active {
  background-color: #3c4454;
}
```

Файл header.component.ts:

```
import {Component, Input, OnInit} from '@angular/core';
import {AuthService} from "../shared/service/auth.service";
import {Router} from "@angular/router";
import {TabConstants} from "../shared/constants/tab-constants";

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.scss']
})
export class HeaderComponent implements OnInit {

  @Input() visibleTabs!: string[]

  username!: string

  constructor(
    private router: Router,
    private authService: AuthService
  ) {
  }

  ngOnInit() {
    this.username = this.authService.getUsername()
  }

  logout() {
    this.authService.logout()
  }

  showSettingsPage() {
    return this.visibleTabs?.includes(TabConstants.SETTINGS)
  }

  showUserPage() {
    return this.visibleTabs?.includes(TabConstants.USER)
  }
```

```
}
```

Файл login.component.html:

```
<div class="dialog-container" style="width: 323px">
  <div class="def-title text-center">
    <h5>Media Storage</h5>
  </div>
  <form #form="ngForm" (ngSubmit)="onSubmit(form)">
    <div class="input-field">
      <input type="text" name="username" placeholder="Username" class="form-control
shadow-none def-input" ngModel
             required>
    </div>
    <div class="input-field">
      <input type="password" name="password" placeholder="Пароль" class="form-control
shadow-none def-input" ngModel
             required>
    </div>
    <div class="input-field text-center">
      MFA token (optional)
    </div>
    <div class="input-field d-flex justify-content-center">
      <ng-otp-input [config]="otpInputConfig"></ng-otp-input>
    </div>
    <div *ngIf="errorMessage" class="input-field">{{errorMessage}}</div>
    <div class="input-field">
      <div class="row">
        <div class="col-5">
          <input type="submit" value="Увійти" class="btn def-btn">
        </div>
        <div class="col-7">
          <input type="button" value="Реєстрація" class="btn def-btn"
(click)="onRegistrationClick()">
        </div>
      </div>
    </div>
    <div class="input-field">
      <a class="def-link" [routerLink]="['/restore']">Забули пароль?</a>
    </div>
  </form>
</div>
```

Файл login.component.ts:

```
import {Component, OnInit, ViewChild} from '@angular/core';
import {Router} from "@angular/router";
import {AuthService} from "../shared/service/auth.service";
import {NgForm} from "@angular/forms";
import {NgOtpInputComponent, NgOtpInputConfig} from "ng-otp-input";

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
```

```
})
export class LoginComponent implements OnInit {

  @ViewChild(NgOtpInputComponent) otpInput!: NgOtpInputComponent;

  errorMessage!: string

  otpInputConfig: NgOtpInputConfig = {
    length: 6,
    isPasswordInput: true,
    allowNumbersOnly: true,
    containerClass: "d-flex",
    inputClass: "form-control shadow-none otp-input"
  }

  constructor(
    private router: Router,
    private authService: AuthService
  ) {
  }

  async ngOnInit() {
    if (this.authService.getAuthCredentials()) {
      await this.router.navigate([`users/${this.authService.getUsername()}`])
      return
    }
  }

  onSubmit(form: NgForm) {
    if (form.valid) {
      let mfaToken = this.otpInput.currentVal ? this.otpInput.currentVal : ""
      this.authService.login(form.value.username, form.value.password,
mfaToken).subscribe({
        next: () => {
          this.router.navigate([`users/${this.authService.getUsername()}`])
        },
        error: error => {
          switch (error.status) {
            case 401:
            case 403: {
              this.errorMessage = "Хибні облікові дані!"
              break
            }
            default: {
              this.errorMessage = "Непередбачувана помилка!"
            }
          }
        }
      })
    }
  }

  onRegistrationClick() {
    this.router.navigate(['/registration'])
  }

}
```

Файл message.component.html:

```
<div class="dialog-container" style="width: 400px">
  <div class="def-title text-center">
    <h5>{{getTitle()}}</h5>
  </div>
  <div class="text-center">
    <p>{{getText()}}</p>
  </div>
  <div class="input-field d-flex justify-content-center">
    <div class="col-5">
      <input type="button" value="Закрити" class="btn back-btn"
(click)="router.navigate([''])">
    </div>
  </div>
</div>
```

Файл message.component.ts:

```
import { Component } from '@angular/core';
import {AuthService} from "../shared/service/auth.service";
import {MessageDataService} from "../shared/service/message-data.service";
import {Router} from "@angular/router";

@Component({
  selector: 'app-message',
  templateUrl: './message.component.html',
  styleUrls: ['./message.component.scss']
})
export class MessageComponent {

  constructor(
    public router: Router,
    private messageDataService: MessageDataService,
    private authService: AuthService
  ) {
  }

  async ngOnInit() {
    if (this.authService.getAuthCredentials()) {
      await this.router.navigate([`users/${this.authService.getUsername()}`])
      return
    }
    if (this.getTitle() == undefined || this.getText() == undefined) {
      await this.router.navigate([''])
      return
    }
  }

  getTitle(): string {
    return this.messageDataService.title;
  }

  getText(): string {
```

```
    return this.messageDataService.text;
  }

}
```

Файл registration.component.html:

```
<div class="dialog-container">
  <div *ngIf="!showRegistrationSuccessfulMessage; else registrationSuccessfulMessage"
style="width: 450px">
    <div class="def-title text-center">
      <h5>Реєстрація</h5>
    </div>
    <form #form="ngForm" (ngSubmit)="onSubmit(form)">
      <div class="row input-field">
        <div class="col">
          <input type="text" name="firstName" placeholder="Ім'я" class="form-control
shadow-none def-input"
                 [(ngModel)]="registrationForm.firstName" #firstName="ngModel"
                 [class.is-invalid]="firstName.touched && firstName.invalid"
                 [class.is-valid]="firstName.touched && firstName.valid"
                 [minlength]="getMinFirstNameLength()"
[maxlength]="getMaxFirstNameLength()"
                 required>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(firstName.name) &&
firstName.invalid">{{errorMessages.get(firstName.name)}}</div>
        </div>
        <div class="col">
          <input type="text" name="lastName" placeholder="Прізвище" class="form-control
shadow-none def-input"
                 [(ngModel)]="registrationForm.lastName" #lastName="ngModel"
                 [class.is-invalid]="lastName.touched && lastName.invalid"
                 [class.is-valid]="lastName.touched && lastName.valid"
                 [minlength]="getMinLastNameLength()" [maxlength]="getMaxLastNameLength()"
                 required>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(lastName.name) &&
lastName.invalid">{{errorMessages.get(lastName.name)}}</div>
        </div>
      </div>
      <div class="input-field">
        <input type="text" name="email" placeholder="Email" class="form-control shadow-
none def-input"
               [(ngModel)]="registrationForm.email" #email="ngModel"
               [class.is-invalid]="email.touched && email.invalid"
               [class.is-valid]="email.touched && email.valid"
               required>
        <div class="invalid-feedback"
             [class.d-block]="errorMessages.has(email.name) &&
email.invalid">{{errorMessages.get(email.name)}}</div>
      </div>
      <div class="row input-field">
        <div class="col">
          <input type="text" name="username" placeholder="Username" class="form-control
shadow-none def-input"
```

```
                [(ngModel)]="registrationForm.username" #username="ngModel"
                [class.is-invalid]="username.touched && username.invalid"
                [class.is-valid]="username.touched && username.valid"
                [minlength]="getMinUsernameLength()" [maxlength]="getMaxUsernameLength()"
autocomplete="off" required>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(username.name) &&
username.invalid">{{errorMessages.get(username.name)}}</div>
        </div>
        <div class="col">
          <div class="input-group">
            <input [type]="passwordVisibility ? 'text' : 'password'" name="password"
placeholder="Пароль"
                   class="form-control shadow-none def-input"
                   [(ngModel)]="registrationForm.password" #password="ngModel"
                   [class.is-invalid]="password.touched && password.invalid"
                   [class.is-valid]="password.touched && password.valid"
                   [minlength]="getMinPasswordLength()" autocomplete="off" required>
            <button type="button" class="btn input-group-btn"
(click)="togglePasswordVisibility()">
                <span class="material-icons align-middle">
                  {{passwordVisibility ? 'visibility_off' : 'visibility' }}
                </span>
            </button>
          </div>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(password.name) &&
password.invalid">{{errorMessages.get(password.name)}}</div>
        </div>
      </div>
      <div class="input-field">
        <input type="checkbox" name="useMfa" class="form-check-input shadow-none"
               [(ngModel)]="registrationForm.useMfa">
        <label class="def-label">Двофакторна автентифікація</label>
      </div>
      <div class="input-field">
        <div class="row">
          <div class="col-6">
            <input type="button" value="На сторінку входу" class="btn back-btn"
(click)="router.navigate([''])">
          </div>
          <div class="col-6">
            <input type="submit" value="Зареєструватися" class="btn def-btn">
          </div>
        </div>
      </div>
    </form>
  </div>
  <ng-template #registrationSuccessfulMessage>
    <div style="width: 310px">
      <div class="input-field">
        Для завершення реєстрації, перейдіть за посиланням відправленим на вказану вами
адресу електронної пошти.
        Відскануйте QR-код для генерації MFA токенів для входу.
      </div>
      <div class="input-field d-flex justify-content-center">
        <div class="col-9">
```

```
          <img class="img-fluid" src="{{qrCode}}">
        </div>
      </div>
      <div class="input-field">
        <span class="text-break">{{secretKey}}</span>
      </div>
      <div class="input-field text-center">
        Резервні токени:
      </div>
      <div class="input-field">
        <ol class="two-columns text-center">
          <li *ngFor="let reserveToken of reserveTokenList">{{reserveToken}}</li>
        </ol>
      </div>
      <div class="input-field d-flex justify-content-center">
        <div class="col-5">
          <input type="button" value="Закрити" class="btn back-btn"
(click)="router.navigate([''])">
        </div>
      </div>
    </div>
  </ng-template>
</div>
```

Файл registration.component.ts:

```
import {Component} from '@angular/core';
import {NgForm} from "@angular/forms";
import {Router} from "@angular/router";
import {AuthService} from "../shared/service/auth.service";
import {RegistrationForm} from "../shared/model/registration-form";
import {StatusCodes} from "../shared/constants/status-codes";
import CommonUtil from "../shared/util/common-util";
import {environment} from "../../environments/environment";

@Component({
  selector: 'app-registration',
  templateUrl: './registration.component.html',
  styleUrls: ['./registration.component.scss']
})
export class RegistrationComponent {

  registrationForm = new RegistrationForm()
  errorMessages = new Map<string, string>();
  passwordVisibility = false
  showRegistrationSuccessfulMessage = false

  qrCode!: string
  secretKey!: string
  reserveTokenList!: string[]

  constructor(
    public router: Router,
    private authService: AuthService
  ) {
  }
```

```
onSubmit(form: NgForm) {
  if (form.valid) {
    this.authService.registration(this.registrationForm).subscribe({
      next: data => {
        let formStatus = data.formStatusDto
        if (formStatus.status == StatusCodes.FAILURE) {
          this.errorMessages = CommonUtil.updateForm(form, formStatus)
        } else if (formStatus.status == StatusCodes.SUCCESS) {
          this.qrCode = data.qrCode
          this.secretKey = data.secretKey
          this.reserveTokenList = data.reserveTokenList
          this.showRegistrationSuccessfulMessage = true
        } else {
          console.error("Unexpected form status: " + formStatus.status)
          this.router.navigate([''])
        }
      },
      error: (err) => {
        console.error(err)
        this.router.navigate([''])
      }
    })
  } else {
    this.setFormFieldsTouched(form, true)
  }
}

setFormFieldsTouched(form: any, type: boolean) {
  form.controls.firstName.touched = type
  form.controls.lastName.touched = type
  form.controls.email.touched = type
  form.controls.username.touched = type
  form.controls.password.touched = type
  form.controls.useMfa.touched = type
}

togglePasswordVisibility() {
  this.passwordVisibility = !this.passwordVisibility;
}

getMinFirstNameLength() {
  return environment.minFirstNameLength
}

getMaxFirstNameLength() {
  return environment.maxFirstNameLength
}

getMinLastNameLength() {
  return environment.minLastNameLength
}

getMaxLastNameLength() {
  return environment.maxLastNameLength
}
```

```
  getMinUsernameLength() {
    return environment.minUsernameLength
  }

  getMaxUsernameLength() {
    return environment.maxUsernameLength
  }

  getMinPasswordLength() {
    return environment.minPasswordLength
  }

}
```

Файл reset-password.component.html:

```
<div class="dialog-container">
  <form #form="ngForm" (ngSubmit)="onSubmit(form)">
    <div class="def-title text-center">
      <h5>Новий пароль</h5>
    </div>
    <div>
      <p>Вкажіть новий пароль для вашої сторінки.</p>
    </div>
    <div>
      <input type="text" name="restoreKey" [(ngModel)]="resetPasswordRequest.restoreKey"
             #restoreKey="ngModel" hidden required>
      <div class="invalid-feedback input-field text-center"
           [class.d-block]="errorMessages.has(restoreKey.name) &&
restoreKey.invalid">{{errorMessages.get(restoreKey.name)}}
      </div>
    </div>
    <div class="input-field">
      <div class="input-group">
        <input [type]="passwordVisibility ? 'text' : 'password'" name="password"
placeholder="Пароль"
               class="form-control shadow-none def-input"
               (input)="validateRepeatPassword(form)"
               [(ngModel)]="resetPasswordRequest.newPassword" #password="ngModel"
               [class.is-invalid]="password.touched && password.invalid"
               [class.is-valid]="password.touched && password.valid"
               [minlength]="getMinPasswordLength()" autocomplete="off" required>
        <button type="button" class="btn input-group-btn"
(click)="togglePasswordVisibility()">
          <span class="material-icons align-middle">
            {{passwordVisibility ? 'visibility_off' : 'visibility' }}
          </span>
        </button>
      </div>
      <div class="invalid-feedback"
           [class.d-block]="errorMessages.has(password.name) &&
password.invalid">{{errorMessages.get(password.name)}}
      </div>
    </div>
    <div class="input-field">
      <div class="input-group">
```

```
        <input [type]="repeatPasswordVisibility ? 'text' : 'password'"
name="repeatPassword"
               placeholder="Підтвердіть пароль"
               class="form-control shadow-none def-input"
               (input)="validateRepeatPassword(form)"
               [(ngModel)]="form.value.repeatPassword" #repeatPassword="ngModel"
               [class.is-invalid]="repeatPassword.touched && repeatPassword.invalid"
               [class.is-valid]="repeatPassword.touched && repeatPassword.valid"
               [minlength]="getMinPasswordLength()" required>
        <button type="button" class="btn input-group-btn"
(click)="toggleRepeatPasswordVisibility()">
          <span class="material-icons align-middle">
            {{repeatPasswordVisibility ? 'visibility_off' : 'visibility' }}
          </span>
        </button>
      </div>
    </div>
    <div class="input-field">
      <div class="row">
        <div class="col-7">
          <input type="button" value="На сторінку входу" class="btn back-btn"
(click)="router.navigate([''])">
        </div>
        <div class="col-5">
          <input type="submit" value="Змінити" class="btn def-btn">
        </div>
      </div>
    </div>
  </form>
</div>
```

Файл reset-password.component.ts:

```
import { Component } from '@angular/core';
import {ActivatedRoute, Params, Router} from "@angular/router";
import {AuthService} from "../shared/service/auth.service";
import {MessageDataService} from "../shared/service/message-data.service";
import {ResetPasswordRequest} from "../shared/model/reset-password-request";
import {environment} from "../../environments/environment";
import {StatusCodes} from "../shared/constants/status-codes";
import CommonUtil from "../shared/util/common-util";

@Component({
  selector: 'app-reset-password',
  templateUrl: './reset-password.component.html',
  styleUrls: ['./reset-password.component.scss']
})
export class ResetPasswordComponent {

  passwordVisibility = false
  repeatPasswordVisibility = false
  resetPasswordRequest = new ResetPasswordRequest()
  errorMessages = new Map<string, string>();

  constructor(
    public router: Router,
```

```
    private activatedRoute: ActivatedRoute,
    private authService: AuthService,
    private messageDataService: MessageDataService
  ) {
  }

  async ngOnInit() {
    if (this.authService.getAuthCredentials()) {
      await this.router.navigate([`users/${this.authService.getUsername()}`])
      return
    }
    this.activatedRoute.params.subscribe((params: Params) => {
      this.resetPasswordRequest.restoreKey = params['key']
    })
  }

  onSubmit(form: any) {
    if (form.valid) {
      this.authService.resetPassword(this.resetPasswordRequest).subscribe({
        next: data => {
          if (data.status == StatusCodes.FAILURE) {
            this.errorMessages = CommonUtil.updateForm(form, data)
            if (this.errorMessages.has('password')) {
              this.resetPasswordRequest.newPassword = ''
              form.value.repeatPassword = ''
              form.controls.repeatPassword.touched = false
            }
          } else if (data.status == StatusCodes.SUCCESS) {
            this.messageDataService.title = "Вітаю!"
            this.messageDataService.text = "Пароль успішно змінено."
            this.router.navigate(['/message'])
          } else {
            this.router.navigate([''])
          }
        },
        error: () => {

        }
      })
    } else {
      this.setFormFieldsTouched(form, true)
    }
  }

  validateRepeatPassword(form: any) {
    if (!form.controls.password.valid || form.value.password != form.value.repeatPassword)
{
      form.controls.repeatPassword.setErrors({"failed_validation": true})
    } else {
      form.controls.repeatPassword.setErrors(null)
    }
  }

  setFormFieldsTouched(form: any, type: boolean) {
    form.controls.password.touched = type
    form.controls.repeatPassword.touched = type
  }
```

```
  togglePasswordVisibility() {
    this.passwordVisibility = !this.passwordVisibility;
  }

  toggleRepeatPasswordVisibility() {
    this.repeatPasswordVisibility = !this.repeatPasswordVisibility;
  }

  getMinPasswordLength() {
    return environment.minPasswordLength
  }

}
```

Файл restore.component.html:

```
<div class="dialog-container" style="width: 450px">
  <form #form="ngForm" (ngSubmit)="onSubmit(form)">
    <div class="def-title text-center">
      <h5>Вкажіть email</h5>
    </div>
    <div>
      <p>Будь ласка, вкажіть email, який прив'язаний до вашого облікового запису.</p>
    </div>
    <div class="input-field">
      <input type="email" name="email" placeholder="Email" class="form-control shadow-none
def-input"
             #email="ngModel"
             [class.is-invalid]="email.touched && email.invalid"
             [class.is-valid]="email.touched && email.valid"
             ngModel required>
    </div>
    <div class="invalid-feedback"
         [class.d-block]="errorMessages.has(email.name)  &&
email.invalid">{{errorMessages.get(email.name)}}</div>
    <div class="input-field">
      <div class="row">
        <div class="col-6">
          <input type="button" value="На сторінку входу" class="btn back-btn"
(click)="router.navigate([''])">
        </div>
        <div class="col-6">
          <input type="submit" value="Змінити пароль" class="btn def-btn">
        </div>
      </div>
    </div>
  </form>
</div>
```

Файл restore.component.ts:

```
import { Component } from '@angular/core';
import {NgForm} from "@angular/forms";
import {Router} from "@angular/router";
```

```
import {AuthService} from "../shared/service/auth.service";
import {MessageDataService} from "../shared/service/message-data.service";
import {StatusCodes} from "../shared/constants/status-codes";
import CommonUtil from "../shared/util/common-util";

@Component({
  selector: 'app-restore',
  templateUrl: './restore.component.html',
  styleUrls: ['./restore.component.scss']
})
export class RestoreComponent {

  errorMessages = new Map<string, string>();

  constructor(
    public router: Router,
    private authService: AuthService,
    private messageDataService: MessageDataService
  ) {
  }

  onSubmit(form: NgForm) {
    if (form.valid) {
      this.authService.restore(form.value.email).subscribe({
        next: data => {
          if (data.status == StatusCodes.FAILURE) {
            this.errorMessages = CommonUtil.updateForm(form, data)
          } else if (data.status == StatusCodes.SUCCESS) {
            this.messageDataService.title = "Зміна паролю"
            this.messageDataService.text = "Для зміни паролю, перейдіть за посиланням
відправленим на вказану вами адресу електронної пошти та вкажіть новий пароль."
            this.router.navigate(['/message'])
          } else {
            this.router.navigate([''])
          }
        },
        error: () => {
          this.router.navigate([''])
        }
      })
    } else {
      this.setFormFieldsTouched(form, true)
    }
  }

  setFormFieldsTouched(form: any, type: boolean) {
    form.controls.email.touched = type
  }
}
```

Файл settings.component.html:

```
<app-header [visibleTabs]="visibleTabs"></app-header>
<div class="settings-container">
  <div class="tabs-panel">
    <input type="button" class="tabs-button" value="Загальне"
```

```
          (click)="switchToBasicSettings()"
          [ngClass]="displaySwitch == 1 ? 'selected' : ''">
    <input type="button" class="tabs-button" value="Безпека"
          (click)="switchToSecuritySettings()"
          [ngClass]="displaySwitch == 2 ? 'selected' : ''">
  </div>
  <div class="vertical-separator"></div>
  <div class="settings-panel">
    <div *ngIf="displaySwitch == 1">
      <form #form="ngForm" (ngSubmit)="onSubmitBasicSettings(form)">
        <div class="row input-field">
          <div class="col">
            <input type="text" name="firstName" placeholder="Ім'я" class="form-control
shadow-none def-input"
                  (change)="setBasicSettingsFormChanged()"
                  [(ngModel)]="basicSettings.firstName" #firstName="ngModel"
                  [class.is-invalid]="firstName.touched && firstName.invalid"
                  [class.is-valid]="firstName.touched && firstName.valid"
                  [minlength]="getMinFirstNameLength()"
[maxlength]="getMaxFirstNameLength()" required>
            <div class="invalid-feedback"
                 [class.d-block]="errorMessages.has(firstName.name) &&
firstName.invalid">{{errorMessages.get(firstName.name)}}</div>
          </div>
          <div class="col">
            <input type="text" name="lastName" placeholder="Прізвище" class="form-control
shadow-none def-input"
                  (change)="setBasicSettingsFormChanged()"
                  [(ngModel)]="basicSettings.lastName" #lastName="ngModel"
                  [class.is-invalid]="lastName.touched && lastName.invalid"
                  [class.is-valid]="lastName.touched && lastName.valid"
                  [minlength]="getMinLastNameLength()"
[maxlength]="getMaxLastNameLength()" required>
            <div class="invalid-feedback"
                 [class.d-block]="errorMessages.has(lastName.name) &&
lastName.invalid">{{errorMessages.get(lastName.name)}}</div>
          </div>
        </div>
        <div class="input-field">
          <input type="text" name="username" placeholder="Username" class="form-control
shadow-none def-input"
                 (change)="setBasicSettingsFormChanged()"
                 [(ngModel)]="basicSettings.username" #username="ngModel"
                 [class.is-invalid]="username.touched && username.invalid"
                 [class.is-valid]="username.touched && username.valid"
                 [minlength]="getMinUsernameLength()" [maxlength]="getMaxUsernameLength()"
required>
          <div class="invalid-feedback"
               [class.d-block]="errorMessages.has(username.name) &&
username.invalid">{{errorMessages.get(username.name)}}</div>
        </div>
        <div class="input-field d-flex justify-content-center">
          <div class="col-4">
            <input type="submit" class="btn def-btn"
                  [class.btn-def]="!basicSettingsFormChanged"
                  [class.btn-not-active]="basicSettingsFormChanged"
                  [value]="basicSettingsFormChanged ? 'Зберегти':  'Збережено' ">
```

```
          </div>
        </div>
      </form>
    </div>
    <div *ngIf="displaySwitch == 2">
      <div class="input-field" *ngIf="securitySettings.mfaActive">
        <input type="button" value="Вимкнути двофакторну автентифікацію" class="btn def-
btn" (click)="deactivateMfa()">
      </div>
      <div class="input-field" *ngIf="!securitySettings.mfaActive">
        <input type="button" value="Активувати двофакторну автентифікацію" class="btn def-
btn" (click)="activateMfa()">
      </div>
      <div class="input-field">
        <input type="button" value="Видалити обліковий запис" class="btn red-btn"
(click)="deleteAccount()">
      </div>
    </div>
  </div>
</div>
```

Файл settings.component.scss:

```
.settings-container {
  display: flex;
  padding: 10px;
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  border-radius: 6px;
  background-color: #212932;
  color: #ececec;
}

.tabs-panel {
  width: 200px;
  margin-right: 10px;
}

.vertical-separator {
  border-left: 2px solid #ececec;
  height: auto;
}

.settings-panel {
  width: 400px;
  margin-left: 10px;
}

.tabs-button {
  width: 200px;
  height: 37px;
  font-size: 15px;
  line-height: 20px;
```

```
  border: none;
  outline: none;
  margin-block: 5px;
  color: #ecece c;
  background-color: #293440;
}

.tabs-button:hover {
  background-color: #323f4d;
}

.tabs-button:active {
  background-color: #374554;
}

.tabs-button.selected {
  background-color: #374554;
}
```

Файл settings.component.ts:

```
import {Component, OnInit} from '@angular/core';
import {Router} from "@angular/router";
import {AuthService} from "../shared/service/auth.service";
import {TabConstants} from "../shared/constants/tab-constants";
import {NgForm} from "@angular/forms";
import {BasicSettings} from "../shared/model/basic-settings";
import {environment} from "../../environments/environment";
import {SettingsService} from "../shared/service/settings.service";
import {StatusCodes} from "../shared/constants/status-codes";
import CommonUtil from "../shared/util/common-util";
import {SecuritySettings} from "../shared/model/security-settings";
import {MatDialog} from "@angular/material/dialog";
import {DeleteAccountDialogComponent} from "../delete-account-dialog/delete-account-
dialog.component";
import {ActivateMfaDialogComponent} from "../activate-mfa-dialog/activate-mfa-
dialog.component";
import {DeactivateMfaDialogComponent} from "../deactivate-mfa-dialog/deactivate-mfa-
dialog.component";

@Component({
  selector: 'app-settings',
  templateUrl: './settings.component.html',
  styleUrls: ['./settings.component.scss']
})
export class SettingsComponent implements OnInit {

  basicSettings = new BasicSettings()
  securitySettings = new SecuritySettings()

  basicSettingsFormChanged = false;
  errorMessages = new Map<string, string>();
  visibleTabs = [TabConstants.USER]
  displaySwitch = 1

  constructor(
```

```
  private router: Router,
  private authService: AuthService,
  private settingsService: SettingsService,
  private dialog: MatDialog
) {
}

async ngOnInit() {
  if (!this.authService.getAuthCredentials()) {
    await this.router.navigate(["/"])
    return
  }
  this.settingsService.loadSettings().subscribe({
    next: data => {
      this.basicSettings.firstName = data.firstName
      this.basicSettings.lastName = data.lastName
      this.basicSettings.username = data.username
      this.securitySettings.mfaActive = data.mfaActive
    },
    error: () => {
    }
  })
}

onSubmitBasicSettings(form: NgForm) {
  if (this.basicSettingsFormChanged) {
    if (form.valid) {
      this.settingsService.changeBasicSettings(this.basicSettings).subscribe({
        next: data => {
          if (data.status == StatusCodes.FAILURE) {
            this.errorMessages = CommonUtil.updateForm(form, data)
          } else if (data.status == StatusCodes.SUCCESS) {
            if (!form.controls['username'].pristine) {
              this.authService.logout()
            }
            this.basicSettingsFormChanged = false;
          } else {
            console.error("Unexpected form status: " + data.status)
            this.router.navigate(['''])
          }
        },
        error: () => {
        }
      })
    } else {
      this.setBasicSettingsFormFieldsTouched(form, true)
    }
  }
}

setBasicSettingsFormFieldsTouched(form: any, type: boolean) {
  form.controls.firstName.touched = type
  form.controls.lastName.touched = type
  form.controls.username.touched = type
}

activateMfa() {
```

```
  let dialogRef = this.dialog.open(ActivateMfaDialogComponent, {});
  dialogRef.afterClosed().subscribe((result) => {
    if (result) {
      this.securitySettings.mfaActive = true
    }
  })
}

deactivateMfa() {
  let dialogRef = this.dialog.open(DeactivateMfaDialogComponent, {});
  dialogRef.afterClosed().subscribe((result) => {
    if (result) {
      this.securitySettings.mfaActive = false
    }
  })
}

deleteAccount() {
  let dialogRef = this.dialog.open(DeleteAccountDialogComponent, {
    data: this.securitySettings.mfaActive
  });
  dialogRef.afterClosed().subscribe((result) => {
    if (result) {
      this.authService.logout()
    }
  })
}

setBasicSettingsFormChanged() {
  this.basicSettingsFormChanged = true;
}

getMinFirstNameLength() {
  return environment.minFirstNameLength
}

getMaxFirstNameLength() {
  return environment.maxFirstNameLength
}

getMinLastNameLength() {
  return environment.minLastNameLength
}

getMaxLastNameLength() {
  return environment.maxLastNameLength
}

getMinUsernameLength() {
  return environment.minUsernameLength
}

getMaxUsernameLength() {
  return environment.maxUsernameLength
}

switchToBasicSettings() {
```

```
    this.displaySwitch = 1
  }

  switchToSecuritySettings() {
    this.displaySwitch = 2
  }

}
```

Файл endpoint-constants.ts:

```
export class EndpointConstants {

  public static readonly LOGIN_ENDPOINT = '/api/auth/login'
  public static readonly REGISTRATION_ENDPOINT = '/api/auth/registration'
  public static readonly CONFIRM_ENDPOINT = '/api/auth/confirm'
  public static readonly RESTORE_ENDPOINT = '/api/auth/restore'
  public static readonly RESET_PASSWORD_ENDPOINTS = '/api/auth/resetPassword'
  public static readonly ACTIVATE_MFA = '/api/auth/activateMfa'
  public static readonly DEACTIVATE_MFA = '/api/auth/deactivateMfa'
  public static readonly DELETE_ACCOUNT = '/api/auth/deleteAccount'
  public static readonly REFRESH_TOKEN_ENDPOINT = '/api/auth/refreshToken'

  public static readonly GET_SETTINGS_ENDPOINT = '/api/user/getSettings'
  public static readonly CHANGE_BASIC_SETTINGS_ENDPOINT = '/api/user/changeBasicSettings'

  public static readonly GET_MEDIA_LIST_BLOCK_ENDPOINT = "/api/media/getMediaListBlock"
  public static readonly GET_MEDIA_PREVIEW_ENDPOINT = "/api/media/getMediaPreview"
  public static readonly GET_MEDIA_ENDPOINT = "/api/media/getMedia"
  public static readonly ADD_MEDIA_ENDPOINT = "/api/media/addMedia"
  public static readonly DELETE_MEDIA_ENDPOINT = "/api/media/deleteMedia"

}
```

Файл form-fields.ts:

```
export enum FormFields {

  FIRST_NAME = 'FIRST_NAME',
  LAST_NAME = 'LAST_NAME',
  EMAIL = 'EMAIL',
  USERNAME = 'USERNAME',
  PASSWORD = 'PASSWORD',
  MFA_TOKEN = 'MFA_TOKEN',
  RESTORE_KEY = 'RESTORE_KEY',
  ALL_FIELDS = 'ALL_FIELDS'

}
```

Файл jwt-constants.ts:

```
export class JwtConstants {

  public static readonly BEARER_PREFIX = "Bearer "
```

```
  public static readonly AUTH_CREDENTIALS_KEY = "auth_credentials"

}
```

Файл status-codes.ts:

```
export enum StatusCodes {

  SUCCESS = 'SUCCESS',
  FAILURE = 'FAILURE'

}
```

Файл tab-constants.ts:

```
export class TabConstants {

  public static readonly SETTINGS = 'SETTINGS'
  public static readonly USER = 'USER'

}
```

Файл is-visible.directive.ts:

```
import {AfterViewInit, Directive, ElementRef, EventEmitter, Input, Output} from
'@angular/core';

@Directive({
  selector: '[isVisible]'
})
export class IsVisibleDirective implements AfterViewInit {

  @Input() subscribe = false
  @Output() isIntersecting = new EventEmitter()

  constructor(private element: ElementRef) {
  }

  ngAfterViewInit() {
    if (this.subscribe) {
      let intersectionObserver = new IntersectionObserver(entries => {
        entries.forEach(entry => {
          if (entry.isIntersecting) {
            intersectionObserver.unobserve(entry.target)
            this.isIntersecting.emit()
          }
        })
      })
      intersectionObserver.observe(this.element.nativeElement)
    }
  }

}
```

Файл lazy-load-image-hooks.ts:

```
import {Attributes, IntersectionObserverHooks} from "ng-lazyload-image";
import {Injectable} from "@angular/core";
import {AuthService} from "../service/auth.service";

@Injectable()
export class LazyLoadImageHooks extends IntersectionObserverHooks {

  constructor(
    private authService: AuthService
  ) {
    super();
  }

  override loadImage({imagePath}: Attributes): Promise<string> {
    let bearer = this.authService.getBearer()
    return fetch(imagePath, bearer ? {
      headers: {
        Authorization: bearer,
      },
    } : undefined)
      .then((res) => res.blob())
      .then((blob) => URL.createObjectURL(blob))
  }

}
```

Файл auth.interceptor.ts:

```
import {Injectable} from '@angular/core';
import {HttpEvent, HttpHandler, HttpInterceptor, HttpRequest, HttpStatusCode} from
'@angular/common/http';
import {catchError, Observable, switchMap} from 'rxjs';
import {AuthService} from "../service/auth.service";
import {EndpointConstants} from "../constants/endpoint-constants";

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(
    private authService: AuthService
  ) {
  }

  intercept(request: HttpRequest<unknown>, next: HttpHandler):
Observable<HttpEvent<unknown>> {
    let pathname = new URL(request.url).pathname
    if (!pathname.endsWith(EndpointConstants.LOGIN_ENDPOINT) &&
      !pathname.endsWith(EndpointConstants.REGISTRATION_ENDPOINT) &&
      !pathname.endsWith(EndpointConstants.REFRESH_TOKEN_ENDPOINT)
    ) {
      request = this.authService.addAuthHeader(request)
      return next.handle(request).pipe(
        catchError(error => {
```

```
          if (error.status == HttpStatusCode.Forbidden) {
            return this.authService.refreshToken().pipe(
              switchMap(() => {
                request = this.authService.addAuthHeader(request)
                return next.handle(request)
              }),
              catchError(() => {
                this.authService.logout()
                throw error
              })
            )
          } else {
            throw error
          }
        })
      )
    } else {
      return next.handle(request)
    }
  }
}
```

Файл activate-mfa-result.ts:

```
import {FormStatus} from "./form-status";

export class ActivateMfaResult {
  formStatusDto!: FormStatus
  qrCode!: string
  secretKey!: string
  reserveTokenList!: string[]

}
```

Файл basic-settings.ts:

```
export class BasicSettings {
  firstName!: string
  lastName!: string
  username!: string

}
```

Файл deactivate-mfa-request.ts:

```
export class DeactivateMfaRequest {
  password!: string
  mfaToken!: string

}
```

Файл delete-account-request.ts:

```
export class DeleteAccountRequest {
  password!: string
  mfaToken!: string

}
```

Файл form-status.ts:

```
import {StatusCodes} from "../constants/status-codes";

export class FormStatus {
  status!: StatusCodes
  invalidFieldsMap: any

}
```

Файл media.ts:

```
export class Media {

  id!: number
  name!: string
  extension!: string
  width!: number
  height!: number
  previewWidth!: number
  previewHeight!: number
  lastModified!: number

}
```

Файл registration-form.ts:

```
export class RegistrationForm {

  firstName!: string
  lastName!: string
  email!: string
  username!: string
  password!: string
  useMfa!: boolean

}
```

Файл registration-result.ts:

```
import {FormStatus} from "./form-status";

export class RegistrationResult {
  formStatusDto!: FormStatus
  qrCode!: string
  secretKey!: string
  reserveTokenList!: string[]
```

```
}
```

Файл reset-password-request.ts:

```
export class ResetPasswordRequest {
  restoreKey!: string
  newPassword!: string

}
```

Файл security-settings.ts:

```
export class SecuritySettings {

  mfaActive!: boolean

}
```

Файл settings.ts:

```
export class Settings {

  firstName!: string
  lastName!: string
  username!: string
  mfaActive!: boolean

}
```

Файл auth.service.ts:

```
import {Injectable} from '@angular/core';
import {JwtConstants} from "../constants/jwt-constants";
import {environment} from "../../../environments/environment";
import {Router} from "@angular/router";
import {HttpClient, HttpParams, HttpRequest} from "@angular/common/http";
import {JwtHelperService} from "@auth0/angular-jwt";
import {EndpointConstants} from "../constants/endpoint-constants";
import {map} from "rxjs";
import {RegistrationForm} from "../model/registration-form";
import {RegistrationResult} from "../model/registration-result";
import {FormStatus} from "../model/form-status";
import {ResetPasswordRequest} from "../model/reset-password-request";
import {DeleteAccountRequest} from "../model/delete-account-request";
import {DeactivateMfaRequest} from "../model/deactivate-mfa-request";
import {ActivateMfaResult} from "../model/activate-mfa-result";

@Injectable({
  providedIn: 'root'
})
export class AuthService {
```

```
  constructor(
    private httpClient: HttpClient,
    private jwtHelperService: JwtHelperService,
    private router: Router
  ) {
  }

  login(username: string, password: string, mfaToken: string) {
    const body = new HttpParams()
      .set('username', username)
      .set('password', password)
      .set('mfaToken', mfaToken);
    return this.httpClient.post<any>(environment.serverUrl +
EndpointConstants.LOGIN_ENDPOINT, body)
      .pipe(map(data => {
        localStorage.setItem(JwtConstants.AUTH_CREDENTIALS_KEY, JSON.stringify(data))
      }))
  }

  logout() {
    localStorage.removeItem(JwtConstants.AUTH_CREDENTIALS_KEY)
    this.router.navigate(["/"])
  }

  registration(registrationForm: RegistrationForm) {
    return this.httpClient.post<RegistrationResult>(environment.serverUrl +
EndpointConstants.REGISTRATION_ENDPOINT, registrationForm);
  }

  confirm(key: string) {
    let params = new HttpParams()
      .set('key', key);
    return this.httpClient.get(environment.serverUrl + EndpointConstants.CONFIRM_ENDPOINT,
{
      params: params
    });
  }

  restore(email: string) {
    let params = new HttpParams()
      .set('email', email);
    return this.httpClient.get<FormStatus>(environment.serverUrl +
EndpointConstants.RESTORE_ENDPOINT, {
      params: params
    });
  }

  resetPassword(resetPasswordRequest: ResetPasswordRequest) {
    return this.httpClient.post<FormStatus>(environment.serverUrl +
EndpointConstants.RESET_PASSWORD_ENDPOINTS, resetPasswordRequest);
  }

  activateMfa(password: string) {
    return this.httpClient.post<ActivateMfaResult>(environment.serverUrl +
EndpointConstants.ACTIVATE_MFA, password);
  }
```

```
  deactivateMfa(deactivateMfaRequestDto: DeactivateMfaRequest) {
    return this.httpClient.post<FormStatus>(environment.serverUrl +
EndpointConstants.DEACTIVATE_MFA, deactivateMfaRequestDto);
  }

  deleteAccount(deleteAccountRequest: DeleteAccountRequest) {
    return this.httpClient.post<FormStatus>(environment.serverUrl +
EndpointConstants.DELETE_ACCOUNT, deleteAccountRequest);
  }

  getAuthCredentials() {
    let authCredentialsStr = localStorage.getItem(JwtConstants.AUTH_CREDENTIALS_KEY)
    return (authCredentialsStr) ? JSON.parse(authCredentialsStr) : null
  }

  getBearer() {
    let authCredentials = this.getAuthCredentials()
    if (authCredentials && authCredentials.access_token) {
      return JwtConstants.BEARER_PREFIX + authCredentials.access_token
    }
    return null
  }

  refreshToken() {
    let authCredentials = this.getAuthCredentials()
    let refresh_token = (authCredentials && authCredentials.refresh_token) ?
authCredentials.refresh_token : null;
    return this.httpClient.get<any>(environment.serverUrl +
EndpointConstants.REFRESH_TOKEN_ENDPOINT, {
      headers: {
        Authorization: JwtConstants.BEARER_PREFIX + refresh_token
      }
    }).pipe(map(data => {
      localStorage.setItem(JwtConstants.AUTH_CREDENTIALS_KEY, JSON.stringify(data))
    }))
  }

  addAuthHeader(request: HttpRequest<unknown>) {
    let authCredentials = this.getAuthCredentials()
    if (authCredentials && authCredentials.access_token) {
      request = request.clone({
        setHeaders: {
          Authorization: JwtConstants.BEARER_PREFIX + authCredentials.access_token
        }
      })
    }
    return request
  }

  getUsername() {
    return this.jwtHelperService.decodeToken(this.getAuthCredentials().access_token).sub
  }

}
```

Файл media.service.ts:

```
import {Injectable} from '@angular/core';
import {HttpClient, HttpParams} from "@angular/common/http";
import {Media} from "../model/media";
import {environment} from "../../../environments/environment";
import {EndpointConstants} from "../constants/endpoint-constants";

@Injectable({
  providedIn: 'root'
})
export class MediaService {

  constructor(private httpClient: HttpClient) {
  }

  loadMediaListBlock() {
    return this.httpClient.get<Media[]>(environment.serverUrl +
EndpointConstants.GET_MEDIA_LIST_BLOCK_ENDPOINT)
  }

  loadMediaListBlockBeforeDate(beforeDate: number) {
    let params = new HttpParams()
      .set('beforeDate', beforeDate)
    return this.httpClient.get<Media[]>(environment.serverUrl +
EndpointConstants.GET_MEDIA_LIST_BLOCK_ENDPOINT, {
      params: params
    })
  }

  addMedia(payload: FormData) {
    return this.httpClient.post<Media[]>(environment.serverUrl +
EndpointConstants.ADD_MEDIA_ENDPOINT, payload);
  }

  deleteMedia(id: number) {
    let params = new HttpParams()
      .set('id', id);
    return this.httpClient.get(environment.serverUrl +
EndpointConstants.DELETE_MEDIA_ENDPOINT, {
      params: params
    })
  }


}
```

Файл message-data.service.ts:

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class MessageDataService {

  title!: string
  text!: string
```

```
  constructor() {
  }

}
```

Файл settings.service.ts:

```
import {Injectable} from '@angular/core';
import {HttpClient} from "@angular/common/http";
import {Settings} from "../model/settings";
import {environment} from "../../../environments/environment";
import {EndpointConstants} from "../constants/endpoint-constants";
import {FormStatus} from "../model/form-status";
import {BasicSettings} from "../model/basic-settings";
import {DeleteAccountRequest} from "../model/delete-account-request";

@Injectable({
  providedIn: 'root'
})
export class SettingsService {

  constructor(private httpClient: HttpClient) {
  }

  loadSettings() {
    return this.httpClient.get<Settings>(environment.serverUrl +
EndpointConstants.GET_SETTINGS_ENDPOINT)
  }

  changeBasicSettings(basicSettings: BasicSettings) {
    return this.httpClient.post<FormStatus>(environment.serverUrl +
EndpointConstants.CHANGE_BASIC_SETTINGS_ENDPOINT, basicSettings);
  }

}
```

Файл common-util.ts:

```
import {FormStatus} from "../model/form-status";
import {FormFields} from "../constants/form-fields";

export default class CommonUtil {

  public static updateForm(form: any, formStatus: FormStatus) {
    let errorMessages = new Map<string, string>();
    if (formStatus.invalidFieldsMap) {
      if (formStatus.invalidFieldsMap[FormFields.FIRST_NAME]) {
        form.controls.firstName.setErrors({"failed_validation": true})
        errorMessages.set('firstName', formStatus.invalidFieldsMap[FormFields.FIRST_NAME])
      }
      if (formStatus.invalidFieldsMap[FormFields.LAST_NAME]) {
        form.controls.lastName.setErrors({"failed_validation": true})
        errorMessages.set('lastName', formStatus.invalidFieldsMap[FormFields.LAST_NAME])
      }
```

```
      if (formStatus.invalidFieldsMap[FormFields.EMAIL]) {
        form.controls.email.setErrors({"failed_validation": true})
        errorMessages.set('email', formStatus.invalidFieldsMap[FormFields.EMAIL])
      }
      if (formStatus.invalidFieldsMap[FormFields.USERNAME]) {
        form.controls.username.setErrors({"failed_validation": true})
        errorMessages.set('username', formStatus.invalidFieldsMap[FormFields.USERNAME])
      }
      if (formStatus.invalidFieldsMap[FormFields.PASSWORD]) {
        form.controls.password.setErrors({"failed_validation": true})
        errorMessages.set('password', formStatus.invalidFieldsMap[FormFields.PASSWORD])
      }
      if (formStatus.invalidFieldsMap[FormFields.RESTORE_KEY]) {
        form.controls.restoreKey.setErrors({"failed_validation": true})
        errorMessages.set('restoreKey',
formStatus.invalidFieldsMap[FormFields.RESTORE_KEY])
      }
      if (formStatus.invalidFieldsMap[FormFields.MFA_TOKEN]) {
        form.controls.mfaToken.setErrors({"failed_validation": true})
        errorMessages.set('mfaToken', formStatus.invalidFieldsMap[FormFields.MFA_TOKEN])
      }
      if (formStatus.invalidFieldsMap[FormFields.ALL_FIELDS]) {
        errorMessages.set('allFields', formStatus.invalidFieldsMap[FormFields.ALL_FIELDS])
      }
    }
    return errorMessages
  }

}
```

Файл user.component.html:

```
<app-header [visibleTabs]="visibleTabs"></app-header>
<div class="content">
  <div class="gallery">
    <div class="media-container" *ngFor="let media of mediaList; let last = last"
         [style.width.px]="limitRange(environment.minMediaPreviewWidth,
environment.maxMediaPreviewWidth, media.previewWidth)"
         [style.height.px]="media.previewHeight"
         (click)="selectedMedia = media;"
         isVisible [subscribe]="last" (isIntersecting)="isIntersecting()">
      <img [defaultImage]="'assets/loading-animation.svg'"
           [lazyLoad]="createMediaPreviewUrl(media.id)"
           [useSrcset]="true"/>
      <button class="pop-up download-button"
              (click)="downloadMedia(createMediaUrl(media.id), media.name);
$event.stopPropagation()">
        <img src="assets/download-icon.svg">
      </button>
      <button class="pop-up delete-button"
              (click)="deleteMedia(media.id); $event.stopPropagation()">
        <img src="assets/trash-can-icon.svg">
      </button>
      <div *ngIf="media.extension === 'gif'" class="info-label type-label">GIF</div>
      <div class="info-label date-label">
        <span class="date">{{media.lastModified | date:'yyyy.MM.dd'}}</span>
```

```
        <span class="date-time">{{media.lastModified | date:'yyyy.MM.dd H:mm'}}</span>
      </div>
    </div>
  </div>

  <input type="file" multiple hidden #photoFile accept="image/gif, image/jpeg, image/png"
         (change)="addMedia($event.target)">

  <button class="fixed-button add-button" [disabled]="loading"
(click)="photoFile.click()">
    <img *ngIf="!loading" src="assets/plus-icon.svg">
    <img *ngIf="loading" src="assets/loading-animation-40.svg">
  </button>

  <div *ngIf="selectedMedia" class="dialog-background">
    <div class="media-container" [ngClass]="checkFixSide(selectedMedia.width,
selectedMedia.height)">
      <img [lazyLoad]="createMediaUrl(selectedMedia.id)"
           [useSrcset]="true"/>
      <button class="translucent-container download-button"
              (click)="downloadMedia(createMediaUrl(selectedMedia.id),
selectedMedia.name)">
        <img src="assets/download-icon.svg">
      </button>
      <button class="translucent-container delete-button"
              (click)="deleteMedia(selectedMedia.id); selectedMedia = undefined">
        <img src="assets/trash-can-icon.svg">
      </button>
    </div>
    <button class="fixed-button close-button" (click)="selectedMedia = undefined">
      <img src="assets/close.svg">
    </button>
  </div>
</div>
```

Файл user.component.scss:

```
.content {
  padding-top: 56px;
}

.gallery {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 10px;
  padding: 10px;
}

.fix-width {
  width: 85%;
}

.fix-width img {
  width: 100%;
```

```
}

.fix-height {
  height: 90%;
}

.fix-height img {
  height: 100%;
}

.media-container {
  overflow: hidden;
  display: flex;
  align-items: center;
  justify-content: center;
  position: relative;
  border-radius: 6px;
  background-color: #242e38;
}

.media-container button {
  position: absolute;
  display: flex;
  align-items: center;
  justify-content: center;
  border: 0;
  border-radius: 6px;
  background-color: #00000072;
  cursor: pointer;
}

.media-container button:active {
  background-color: #00000099;
}

.media-container .download-button {
  height: 30px;
  width: 30px;
  top: 10px;
  right: 10px;
}

.media-container .delete-button {
  height: 30px;
  width: 30px;
  top: 50px;
  right: 10px;
}

.info-label {
  position: absolute;
  color: #ececec;
  font-size: 12px;
  padding: 3px 5px;
  border-radius: 6px;
  background-color: #00000072;
  cursor: default;
```

```
}

.type-label {
  left: 6px;
  bottom: 6px;
}

.date-label {
  right: 6px;
  bottom: 6px;
}

.date-label .date-time {
  display: none;
}

.date-label:hover > .date-time {
  display: block;
}

.date-label:hover > .date {
  display: none;
}

.pop-up {
  opacity: 0;
  transition: opacity 0.2s;
}

div:hover > .pop-up {
  opacity: 1;
}

.fixed-button {
  height: 50px;
  width: 50px;
  position: fixed;
  border: 0;
  border-radius: 10px;

  display: flex;
  align-items: center;
  justify-content: center;
}

.add-button {
  right: 30px;
  bottom: 30px;
  background-color: #1c7ccbd8;
  cursor: pointer;
}

.add-button:active {
  background-color: #1c7ccbb2;
}

.close-button {
```

```
  right: 30px;
  top: 30px;
  background-color: #0000;
  cursor: pointer;
}

.close-button:active {
  background-color: #5E5E5EB2;
}
```

Файл user.component.ts:

```
import {Component, OnInit} from '@angular/core';
import {Media} from "../shared/model/media";
import {MediaService} from "../shared/service/media.service";
import {EndpointConstants} from "../shared/constants/endpoint-constants";
import {environment} from "../../environments/environment";
import {AuthService} from "../shared/service/auth.service";
import {Router} from "@angular/router";
import {TabConstants} from "../shared/constants/tab-constants";

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html',
  styleUrls: ['./user.component.scss']
})
export class UserComponent implements OnInit {

  environment = environment
  mediaList!: Media[]
  loading = false
  checkNextMediaListBlock = true
  selectedMedia: Media | undefined
  visibleTabs = [TabConstants.SETTINGS]


  constructor(
    private router: Router,
    private authService: AuthService,
    private mediaService: MediaService
  ) {
  }

  async ngOnInit() {
    if (!this.authService.getAuthCredentials()) {
      await this.router.navigate(["/"])
      return
    }
    this.mediaService.loadMediaListBlock().subscribe({
      next: data => {
        this.mediaList = data
      },
      error: (err) => {
        console.error(err)
      }
    })
```

```
}

downloadMedia(url: string, name: string) {
  let bearer = this.authService.getBearer()
  fetch(url, bearer ? {
    headers: {
      Authorization: bearer,
    },
  } : undefined).then(response => response.blob())
    .then(blob => {
      let blobUrl = window.URL.createObjectURL(blob)
      let a = document.createElement('a')
      a.href = blobUrl
      a.download = name
      a.click()
      a.remove()
    })
}

addMedia(target: any) {
  if (target.files.length > 0) {
    let payload = new FormData()
    let files = target.files
    for (let i = 0; i < files.length; i++) {
      let file = files[i]
      payload.append("multipartFileList", file)
    }
    this.loading = true
    this.mediaService.addMedia(payload).subscribe({
      next: (data) => {
        this.mediaList = data.concat(this.mediaList)
        this.loading = false
      },
      error: (err) => {
        console.error(err)
        this.loading = false
      }
    })
  }
}

deleteMedia(id: number) {
  this.mediaService.deleteMedia(id).subscribe({
    next: () => {
      this.mediaList = this.mediaList.filter(media => media.id !== id)
    },
    error: (err) => {
      console.error(err)
    }
  })
}

createMediaPreviewUrl(id: number) {
  let url = new URL(environment.serverUrl)
  url.pathname = EndpointConstants.GET_MEDIA_PREVIEW_ENDPOINT
  url.searchParams.append("id", id.toString())
  return url.toString()
```

```
  }

  createMediaUrl(id: number) {
    let url = new URL(environment.serverUrl)
    url.pathname = EndpointConstants.GET_MEDIA_ENDPOINT
    url.searchParams.append("id", id.toString())
    return url.toString()
  }

  limitRange(min: number, max: number, value: number) {
    return Math.max(min, Math.min(max, value))
  }

  checkFixSide(width: number, height: number) {
    if (width > height) {
      let relativeHeight = window.innerWidth / width * height
      return relativeHeight > window.innerHeight ? "fix-height" : "fix-width"
    } else {
      let relativeWidth = window.innerHeight / height * width
      return relativeWidth > window.innerWidth ? "fix-width" : "fix-height"
    }
  }

  isIntersecting() {
    let lastMedia = this.mediaList.at(-1)
    if (lastMedia && this.checkNextMediaListBlock && !this.loading) {
      this.mediaService.loadMediaListBlockBeforeDate(lastMedia.lastModified).subscribe({
        next: (data) => {
          this.checkNextMediaListBlock = data.length > 0;
          this.mediaList = this.mediaList.concat(data)
        },
        error: (err) => {
          console.error(err)
        }
      })
    }
  }

}
```

Файл environment.development.ts:

```
export const environment = {
  serverUrl: 'http://localhost:8080',
  title: 'MediaStorage',
  minMediaPreviewWidth: 150,
  maxMediaPreviewWidth: 400,
  maxMediaSize: 10,
  minFirstNameLength: 2,
  maxFirstNameLength: 15,
  minLastNameLength: 2,
  maxLastNameLength: 15,
  minUsernameLength: 2,
  maxUsernameLength: 20,
  minPasswordLength: 8
```

```
};
```

Файл environment.ts:

```
export const environment = {
  serverUrl: 'http://localhost:8080',
  title: 'MediaStorage',
  minMediaPreviewWidth: 150,
  maxMediaPreviewWidth: 400,
  maxMediaSize: 10,
  minFirstNameLength: 2,
  maxFirstNameLength: 15,
  minLastNameLength: 2,
  maxLastNameLength: 15,
  minUsernameLength: 2,
  maxUsernameLength: 20,
  minPasswordLength: 8

};
```

Файл index.html:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Media Storage</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
  <link href="https://fonts.googleapis.com/icon?family=Material+Icons+Outlined"
rel="stylesheet">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Файл main.ts:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';


platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Файл styles.scss:

```
/* You can add global styles to this file, and also import other style files */
```

```
@import url('https://fonts.googleapis.com/css2?family=Roboto&display=swap');

* {
  margin: 0;
  padding: 0;
  font-family: Roboto, serif;
  font-size: 15px;
}

body {
  background-color: #1a2026;
}

.dialog-background {
  position: fixed;
  left: 0;
  right: 0;
  top: 0;
  bottom: 0;
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 1000;
  background-color: #00000099;
}

.dialog-container {
  padding: 15px 30px;
  border-radius: 6px;
  position: fixed;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  background-color: #212932;
  color: #ecece c;
}

.input-field {
  padding-block: 5px;
}

div.def-title {
  padding-block: 0 10px;
  padding-inline: 10px;
}

input.def-input,
input.def-input:focus {
  box-shadow: none;
  outline: none;
  color: #ecece c;
  border: 2px solid #3c4454;
  background-color: #293440;
}

input.def-input::placeholder {
  color: #8794a1;
```

```
}

input.otp-input.invalid,
input.otp-input.invalid:focus{
  border: 2px solid #dc3545 !important;
}

input.otp-input.valid,
input.otp-input.valid:focus{
  border: 2px solid #198754 !important;
}

input.otp-input,
input.otp-input:focus {
  height: 37px !important;
  width: 37px !important;
  font-size: 17px !important;
  box-shadow: none !important;
  outline: none !important;
  color: #ececec;
  border: 2px solid #3c4454 !important;
  background-color: #293440 !important;
}

input.btn,
button.btn {
  height: 37px !important;
  color: #ececec;
  border: none;
  background-color: #248bda;
}

input.input-group-btn,
button.input-group-btn {
  color: #ececec;
  border: 2px solid #3c4454;
  background-color: #293440;
}

input.input-group-btn:hover,
input.input-group-btn:active,
button.input-group-btn:hover,
button.input-group-btn:active {
  background-color: #3F4E60 !important;
}

input.back-btn,
button.back-btn {
  width: 100%;
  background-color: #293440;
}

input.back-btn:hover,
input.back-btn:active,
button.back-btn:hover,
button.back-btn:active {
  background-color: #3F4E60 !important;
```

```
}

input.def-btn,
button.def-btn {
  width: 100%;
  background-color: #248bda;
}

input.def-btn:hover,
input.def-btn:active,
button.def-btn:hover,
button.def-btn:active {
  background-color: #207cc2 !important;
}

input.red-btn,
button.red-btn {
  width: 100%;
  background-color: #dc3545;
}

input.red-btn:hover,
input.red-btn:active,
button.red-btn:hover,
button.red-btn:active {
  background-color: #56000a !important;
}

input.form-check-input {
  margin-inline: 5px 10px;
}

input.form-check-input:checked {
  background-color: #248bda;
}

.def-label {
  color: #ececec;
}

a.def-link {
  color: #ececec;
  text-decoration: none;
}

a.def-link:hover {
  color: #ececec;
  text-decoration: underline;
}

ol {
  list-style: none;
  counter-reset: li
}

li {
  counter-increment: li
```

```
}

ol li::before {
  content: "." counter(li);
  color: #207cc2;
  display: inline-block;
  width: 1em;
  text-align: left;
  margin-right: 0.5em;
  direction: rtl;
}


ol.two-columns {
  padding: 0;
  -moz-column-count: 2;
  -webkit-column-count: 2;
  column-count: 2;
}
```