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

КУРСОВА РОБОТА

на тему: «Розробка веб-сервісу для ведення блогу з підсистемою для автоматичного аналізу тексту»

Виконав:
студент групи ФеІм-12с
Вінницький Іван

Науковий керівник:
асистент Парубочий В. О.

Львів – 2023

Анотація

Основним завданням курсової роботи є створення веб-сервісу, який надає функції ведення блогу та має підсистему автоматичного аналізу тексту. Для цього під час виконання курсової роботі було досліджено такий напрям NLP як Text Summarization та способи його реалізації в різних мовах програмування, в тому числі C# та Python. Було досліджено бібліотеку для функцій NLP – bert-extractive-summarizer. Додаток створений з використанням наступного стеку технологій: .NET, Asp.NET Core, Entity Framework, React, TypeScript. Основною мовою написання серверної частина є мова програмування C#, клієнтської частини – TypeScript. Для реалізації підсистеми автоматичного аналізу тексту, було використано мову Python. Всі ці технології було поєднано в одному клієнт-серверному додатку, який реалізує функції ведення блогу з можливістю автоматичного аналізу тексту для генерування коротких анотацій.

Annotation

The main task of the course work is to create a web service that provides blogging functions and has a subsystem of automatic text analysis. To do this, during the course work, I investigated such a direction of NLP as Text Summarization and ways of its implementation in various programming languages, including C# and Python. Also I learned a library for NLP functions - bert-extractive-summarizer . The application was created using the following technology stack: .NET, Asp.NET Core, Entity Framework, React, TypeScript. The main language for writing the server part is the C# programming language, and the client part – TypeScript. The Python language was used to implement the automatic text analysis subsystem. All these technologies were combined in one client-server application, which implements blogging functions with the possibility of automatic text analysis to generate short annotations.

Зміст

Анотація 1
Вступ 3
1. Обробка природної мови 5
1.1 Як працює обробка природної мови? 5
1.2 Класифікація NLP за виконуваними задачами 6
1.3 Text Summarization як один з видів обробки природної мови 10
1.3.1 Класифікація Text Summarization 12
1.3.2 Популярні додатки для Text Summarization 14
2. Засоби для реалізації блогу з функціями інтелектуального аналізу тексту 15
2.1 Засоби мови C# 15
2.1.1 Технологія ASP.NET Core 15
2.1.2 Entity Framework 17
2.2 React – як засіб для реалізації клієнтської частини 19
2.3 Засоби мови Python 21
2.3.1 Flask – для створення http сервера 21
2.3.2 Бібліотека bert-extractive-summarizer для функцій NLP 21
3. Написання додатка 22
3.1 Реалізація серверної частини 23
3.1.1 Створення проекту та його структура 23
3.1.2 Реалізація CRUD операцій для основних функцій програми 24
3.2 Реалізація клієнтської частини 26
3.2.1 Створення проекту та його структура 26
3.2.2 Реалізація API 28
3.3 Реалізація Python сервера 31
3.3.1 Створення HTTP сервера 31
3.3.2 Реалізація методу із функцією підсумовування тексту 32
4. Опис додатка 32
4.1 Інструкція з використання 32
4.2 Напрямки подальшого вдосконалення додатка 39
Висновки 41
Використана література 42
Додатки 44

Вступ

Метою курсової роботи є написання простого та зручного додатку, який представляє собою блог та реалізовує підсистему автоматичного аналізу тексту.

Перед додатком ставились **наступні вимоги**:

1) **Реалізація додатку з функціями блогу** – додаток повинен реалізовувати застосунок, який виконує функції ведення блогу. Тобто, користувач повинен мати змогу додавати певні записи або якісь помітки, переглядати, змінювати та видаляти їх.
2) **Реалізація підсистеми автоматичного аналізу тексту** – додаток повинен реалізовувати не просто звичайний блог, а повинен мати підсистему автоматичного аналізу тексту. В моєму випадку, це підсистема підсумовування текстів (генерування коротких анотацій) у публікаціях користувачів.
3) **Додаток повинен бути побудований як розподілена система** – це означає, що додаток повинен мати клієнт-серверну архітектуру. Тобто, всі дані про користувачів і про створені ними публікації повинні зберігатися на сервері. А сам користувач буде мати тільки зручний інтерфейс доступу до цих даних. Це дає змогу користувачу додати свої публікації на одному пристрої, а переглянути їх в системі з іншого.

Проблематика автоматичного аналізу тексту є дуже важливою в умовах сучасного розвиненого світу. Системи, які дають змогу обробляти певним чином текст і аналізувати його за певними параметрами зараз є потрібними майже в усіх сферах життя. До таких систем можна віднести: виявлення спаму в електронних листах, автоматичний переклад текстів, генерування коротких анотацій до певних текстів, класифікація текстів за емоційним забарвленням та багато інших.

Тому, як ми можемо пересвідчитися, завдання інтеграції такої системи автоматичного аналізу тексту в блог було б дуже потрібним та багатообіцяючим

полем для можливих функцій цього блогу. За допомогою системи автоматичного аналізу тексту звичайний блог можна дуже розширити та зробити набагато цікавішим. Це можуть бути наступні підсистеми: генерування коротких анотацій до публікацій, визначення емоційного забарвлення текстів, кластеризація публікацій за тематикою та інше. Це значно розширить авдиторію цього блогу та привабить багато нових користувачів. Тож тематика та проблематика курсової роботи має як теоретичне, так і практичне застосування і відповідає потребам сучасного суспільства.

Структура курсової роботи складається з 4 розділів.

В розділі 1 описані основні підходи до галузі обробки природної мови (NLP) та принципи за якими відбуваються дослідження в цій галузі. Окрім цього в ньому розглянуто один із напрямків NLP, який безпосередньо використаний в додатку, а саме Text Summarization.

Розділ 2 містить опис засобів та технологій, які були використані для написання цього додатку. В цьому розділі окремо описані засоби, які були використані для написання серверної частини та засоби та технології, які використовувалися для клієнтської частини.

В розділі 3 міститься детальний опис самого додатку, його елементів та функцій, як серверної так і клієнтської частини. Також, в цьому розділі міститься покроковий опис ходу виконання всієї роботи з конкретними інструкціями щодо написання кожної частини проекту .

В розділі 4 подана детальна інструкція з використання створеного додатку. Окрім цього в ньому зазначені певні обмеження, яких треба дотримуватися при використанні додатка та показані можливі напрямки подальшого розвитку цього проекту.

1. Обробка природної мови

1.1 Як працює обробка природної мови?

Обробка природної мови (Natural Language Processing) – це дисципліна створення машин, які можуть маніпулювати людською мовою, або даними, що нагадують людську мову. Іншими словами, це здатність комп'ютерної програми розуміти людську мову під час процесу розмовного та письмового мовлення, який називають природною мовою. Це компонент штучного інтелекту (ШІ) [1].

NLP дає змогу комп'ютерам розуміти природну мову так само, як і люди. Незалежно від того, чи є мова розмовною чи письмовою, обробка природної мови використовує штучний інтелект, щоб сприймати вхідні дані з реального світу, обробляти їх і надавати сенсу таким чином, який може зрозуміти комп'ютер. Подібно до того, як люди мають різні датчики, наприклад вуха, щоб чути, і очі, щоб бачити, комп'ютери мають програми для читання та мікрофони, щоб збирати аудіо. І так само, як люди мають мозок для обробки цих вхідних даних, комп'ютери мають програму для обробки відповідних вхідних даних. У певний момент обробки вхідні дані перетворюються на код, зрозумілий комп'ютеру [2].

Ця галузь розвинулася з комп'ютерної лінгвістики, яка використовує інформатику для розуміння принципів мови, але замість того, щоб розробляти теоретичні основи, NLP є інженерною дисципліною, яка прагне створювати технології для виконання корисних завдань. NLP можна розділити на дві частини, що перетинаються: розуміння природної мови (NLU), яке зосереджується на семантичному аналізі або визначенні передбачуваного значення тексту і генерування природної мови (NLG), яке зосереджується на генеруванні тексту машиною. NLP відокремлюється від розпізнавання мовлення, але часто використовується в поєднанні з ним, яке прагне розібрати розмовну мову на слова, перетворюючи звук на текст і навпаки [1].

1.2 Класифікація NLP за виконуваними задачами

За виконуваними задачами галузь NLP класифікується на такі типи [1]:

- Аналіз настроїв – це процес класифікації емоційного змісту тексту. Як правило, вхідними даними для моделі класифікації настроїв є фрагмент тексту, а виходом є ймовірність того, що висловлені в тексті настрої є позитивними, негативними або нейтральними. Зазвичай, ця ймовірність базується або на створених вручну ознаках, n-грамах слів, функціях TF-IDF, або на використанні моделей глибокого навчання для фіксації послідовних довго- та короткострокових залежностей. Аналіз настроїв використовується для класифікації відгуків клієнтів на різних онлайн-платформах, а також для спеціальних програм, таких як визначення ознак психічного захворювання в онлайн-коментарях.

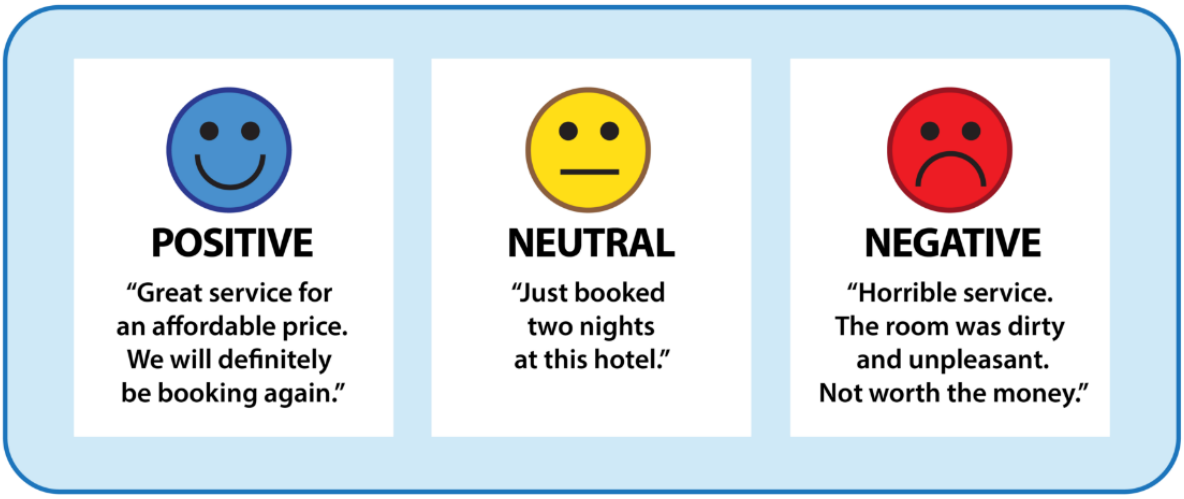


Рис. 1.1. Приклад видів настроїв на які NLP може класифікувати текст [1].

- Класифікація токсичності – це розділ аналізу настроїв, мета якого полягає не лише у класифікації негативних настроїв, а й у класифікації окремих категорій, таких як погрози, образи, непристойність і ненависть до певних осіб. Вхідними даними для такої моделі є текст, а виходом, як правило, є ймовірність кожного класу токсичності. Моделі класифікації токсичності можна використовувати для модерування та покращення онлайн-розмов

шляхом придушення образливих коментарів, виявлення ненависті або сканування документів на наявність наклепу.

- Машинний переклад – автоматизує переклад між різними мовами. Вхідними даними для такої моделі є текст на визначеній вихідній мові, а на виході – текст на вказаній цільовій мові. Перекладач Google Translate є, мабуть, найвідомішим масовим додатком. Такі моделі використовуються для покращення спілкування між людьми на платформах соціальних мереж, таких як Facebook або Skype. Ефективні підходи до машинного перекладу можуть розрізняти слова зі схожими значеннями. Деякі системи також виконують завдання ідентифікації мови, тобто класифікувати текст і віднести його до певної мовної групи.
- Розпізнавання іменованих сутностей – спрямоване на виділення сутностей у фрагменті тексту в попередньо визначені категорії, такі як особисті імена, організації, розташування та кількість. Вхідними даними для такої моделі, як правило, є текст, а виходом є різні іменовані сутності разом із їх початковою та кінцевою позиціями. Розпізнавання іменованих об’єктів корисно в таких програмах, як узагальнення новинних статей і боротьба з дезінформацією.

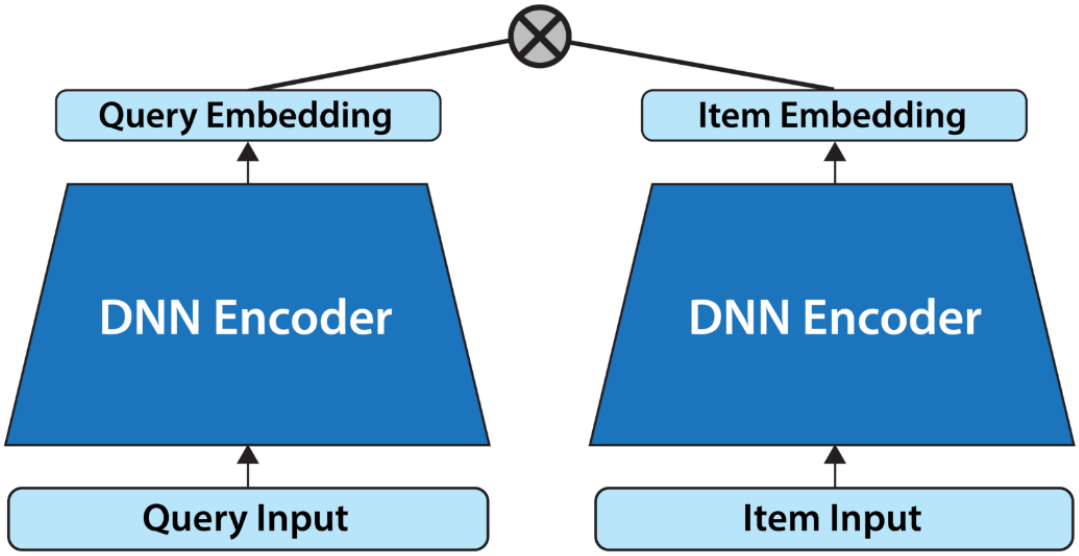
Рис. 1.2. Візуалізація розпізнавання іменованих сутностей у тексті [1].

- Виявлення спаму – є поширеною проблемою бінарної класифікації в NLP, метою якої є класифікація електронних листів як спаму чи ні. Детектори спаму беруть як вхідні дані текст електронної пошти разом із різними іншими підтекстами, як-от заголовок та ім’я відправника. Вони спрямовані на виведення ймовірності того, що лист є спамом. Провайдери електронної

пошти, як-от Gmail, використовують такі моделі, щоб забезпечити кращу взаємодію з користувачем, виявляючи непотрібні та небажані листи та переміщуючи їх у призначену папку для спаму.

- Моделі виправлення граматичних помилок – кодують граматичні правила для виправлення граматики в тексті. Це розглядається в основному як послідовне завдання, де модель навчається на неграматичному реченні як вхідних даних і правильному реченні як вихідних даних. Онлайн-засоби перевірки граматики, такі як Grammarly, і системи обробки текстів, такі як Microsoft Word, використовують такі системи, щоб надати своїм клієнтам кращий досвід письма. Школи також використовують їх для оцінювання учнівських есе.
- Моделювання тем – це завдання неконтрольованого аналізу тексту, яке бере корпус документів і виявляє абстрактні теми в цьому корпусі. Вхідними даними для моделі теми є набір документів, а виходом є список тем, який визначає слова для кожної теми, а також пропорції розподілу кожної теми в документі. Прихований розподіл Діріхле (LDA), один із найпопулярніших методів моделювання тем, намагається розглядати документ як набір тем, а тему як набір слів. Тематичне моделювання використовується в комерційних цілях, щоб допомогти юристам знайти докази в юридичних документах.
- Генерація тексту – офіційно відома як генерація природної мови (NLG), створює текст, схожий на текст, написаний людиною. Такі моделі можна налаштувати для створення тексту в різних жанрах і форматах, включаючи твіти, блоги і навіть комп’ютерний код. Генерація тексту була виконана за допомогою процесів Маркова, LSTM, BERT, GPT-2, LaMDA та інших підходів. Це особливо корисно для автозаповнення та чат-ботів.
- Інформаційний пошук – знаходить документи, які найбільше відповідають запиту. Це проблема, з якою стикається кожна система пошуку та рекомендацій. Мета полягає не в тому, щоб відповісти на конкретний запит, а в тому, щоб отримати з колекції документів, яка може бути

пронумерована мільйонами, набір, який найбільше відповідає запиту. Системи пошуку документів переважно виконують два процеси: індексування та зіставлення. У більшості сучасних систем індексування здійснюється за допомогою векторної просторової моделі за допомогою двохбаштових мереж, тоді як зіставлення здійснюється за допомогою показників подібності або відстані.



A two-tower network creates a representation of an input query and a group of documents (or items) through two separate networks. Then it compares the representation of the query with that of the documents to find documents that are most relevant to the query.

Рис. 1.3. Візуалізація інформаційного пошуку [1].

- Резюмування (підсумовування) – це завдання скоротити текст, щоб виділити найбільш актуальну інформацію. Дослідники Salesforce розробили підсумовувач, який також оцінює узгодженість фактів, щоб гарантувати точність результатів.
- Відповіді на запитання – це відповіді на запитання, поставлені людьми природною мовою. Одним із найпомітніших прикладів відповідей на запитання був Watson, який у 2011 році грав у телевізійну гру-шоу Jeopardy проти людських чемпіонів і виграв із значним відривом.

1.3 Text Summarization як один з видів обробки природної мови

Резюмування тексту (підсумовування) – це процес генерування короткого, плавного, а головне точного резюме (анотації) відповідно довшого текстового документа. Інакше, це процес створення коротшого тексту без усунення семантичної структури тексту [3].

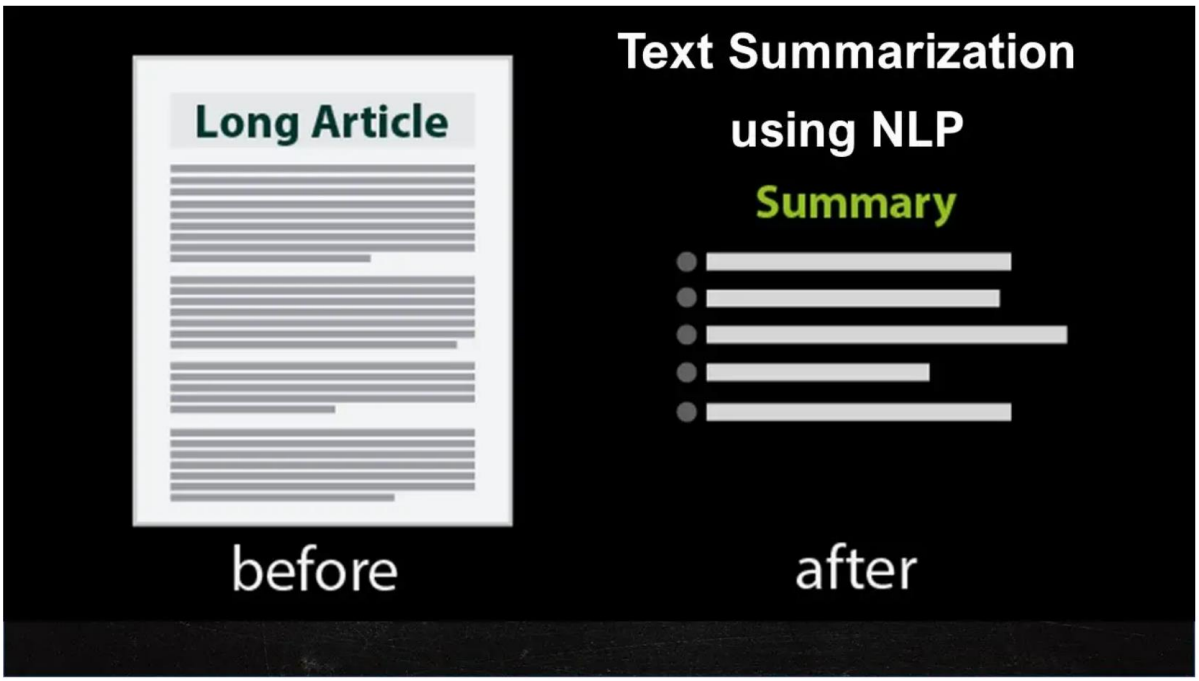


Рис. 1.4. Вхідний і вихідний тексти в процесі підсумовування [4].

Основна ідея автоматичного підсумовування тексту полягає в тому, щоб мати можливість знайти коротку частину найважливішої інформації з усього набору та представити її в зручному для читання форматі. Оскільки текстові дані онлайн зростають, автоматичні методи підсумовування тексту можуть бути дуже корисними, оскільки більше корисної інформації можна прочитати за короткий час.

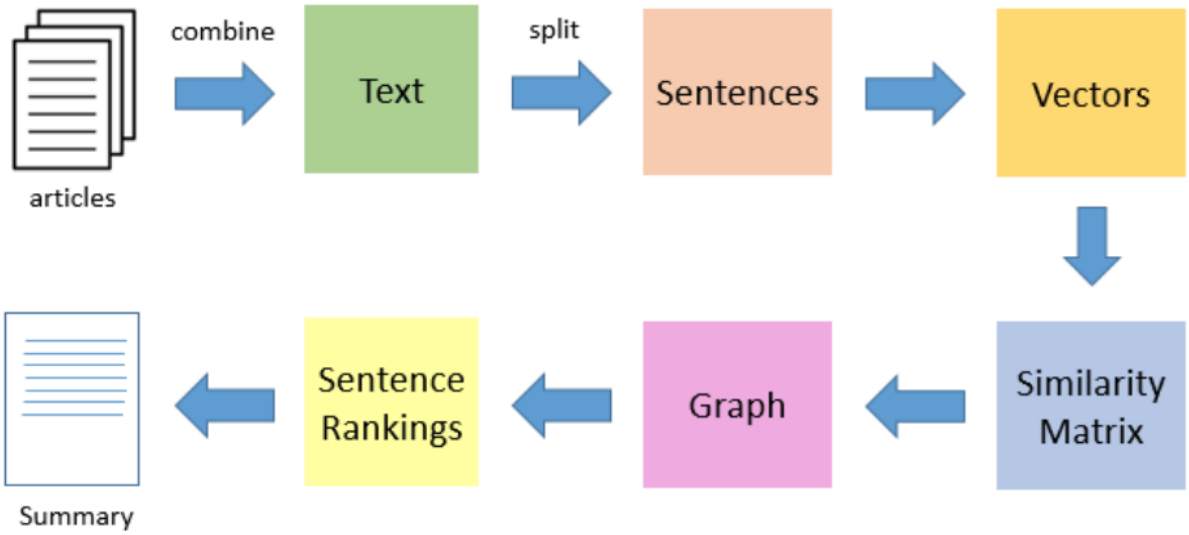


Рис. 1.5. Процес підсумовування тексту [5].

Резюмування тексту зробило революцію в тому, як компанії обробляють і розуміють письмову інформацію. Це дало змогу ефективно витягувати ключові моменти та ідеї з великої кількості тексту, заощаджуючи час і ресурси. Потенціал удосконалення та зростання резюмування тексту великий, оскільки це може значно покращити нашу здатність бути в курсі подій та приймати кращі рішення. Однак існують труднощі, які необхідно подолати, наприклад, потреба у високоякісних навчальних даних і можливість отримання упереджених або неповних підсумків [6].

У обробці природної мови, або NLP, підсумовування тексту відноситься до процесу використання моделей глибокого та машинного навчання для синтезу великих масивів текстів у їхні найважливіші частини. Резюмування тексту можна застосувати до статичних уже існуючих текстів, як-от наукових статей чи новин, або до аудіо- чи відеопотоків, як-от подкаст чи відео YouTube, за допомогою API перетворення мовлення в текст. Деякі API підсумовування тексту надають єдине резюме для тексту, незалежно від довжини, тоді як інші розбивають резюме на коротші позначки часу.

1.3.1 Класифікація Text Summarization

Резюмування тексту передбачає використання алгоритмів і методів для аналізу та вилучення найважливішої інформації з певного тексту. Ось три поширені типи підсумовування [3]:

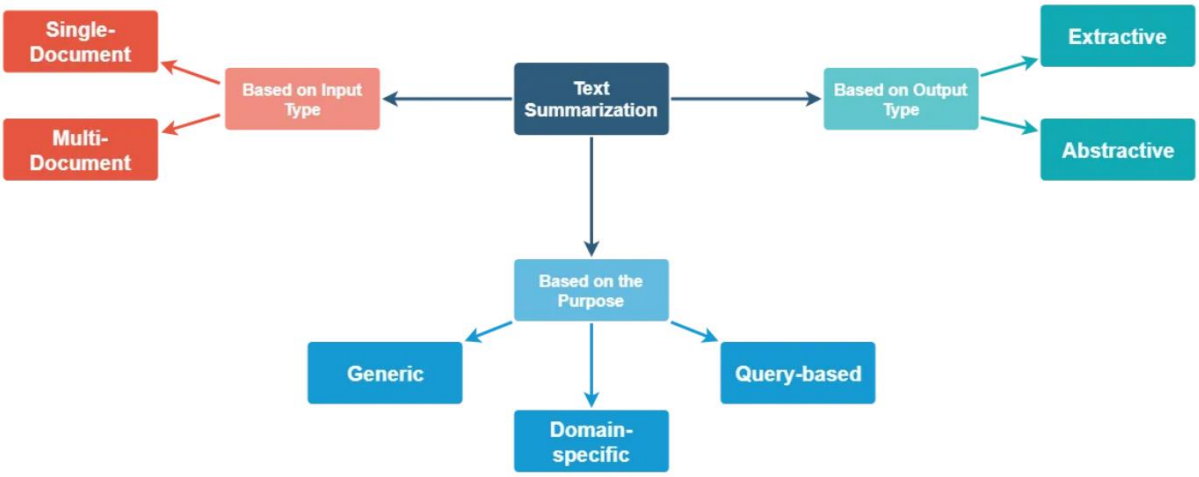


Рис. 1.6. Типи Text Summarization [3].

Залежно від типу введення:

1. Single-Document, де вхідна довжина тексту коротка. Багато ранніх систем підсумовування мали справу з одним документом.
2. Multi-Document, де вхідні дані можуть бути довільними. Часто це означає, що на вхід подається певний набір документів різної довжини.

Виходячи з призначення:

1. Generic, де модель не робить припущень щодо домену чи змісту тексту, який потрібно підсумувати, і розглядає всі вхідні дані як однорідні. Більша частина виконаної роботи зосереджена навколо узагальнення.
2. Domain-specific, де модель використовує предметно-специфічні знання для формування більш точного підсумку. Наприклад, узагальнення науково-дослідницьких статей певної галузі, біомедичних документів тощо.

3. Query-based, де зведення містить лише інформацію, яка відповідає на питання природної мови щодо введеного тексту.

Залежно від типу виводу:

1. Extractive, де важливі речення вибираються з вхідного тексту для формування резюме. Більшість сьогоднішніх підходів до резюмування мають екстрактний характер.

   Використовуючи екстрактний підхід, ми підсумовуємо наш текст на основі простих і традиційних алгоритмів. Наприклад, коли ми хочемо підсумувати наш текст на основі частотного методу, ми зберігаємо всі важливі слова та частоту всіх цих слів у словнику. На основі високочастотних слів ми зберігаємо речення, що містять це слово, у нашому підсумковому резюме. Це означає, що слова, які є в нашому резюме, підтверджують, що вони є частиною даного тексту.

   Це традиційний метод, який був розроблений першим. Основна мета – визначити значущі речення тексту та додати їх до анотації. Слід зазначити, що отримане резюме містить точні речення з оригінального тексту.
2. Abstractive, де модель формує власні фрази та речення, щоб запропонувати більш послідовне резюме, як те, що створила б людина. Такий підхід, безумовно, більш привабливий, але набагато складніший, ніж екстрактне резюмування.

   Абстрактний підхід є більш досконалим. На основі вимог часу ми замінюємо деякі речення на менші речення з тими ж семантичними підходами наших текстових даних. Підхід полягає у визначенні важливих розділів, інтерпретації контексту та відтворенні їх по-новому. Це гарантує, що основна інформація передається через найкоротший текст. Зауважте, що тут підсумкові речення генеруються, а не просто вилучаються з оригінального тексту.

1.3.2 Популярні додатки для Text Summarization

Існує багато доступних інструментів підсумовування тексту, які можуть допомогти вам швидко й точно витягнути ключові моменти та ідеї з великої кількості тексту. Серед популярних варіантів [6]:

- GPT-3 Summarizer – інструмент, розроблений OpenAI, використовує вдосконалену технологію обробки мови для створення людських резюме тексту.
- MonkeyLearn – це хмарне рішення для аналітики тексту, яке має безкоштовний підсумовувач тексту, а також попередньо навчені моделі аналізу тексту, такі як класифікатори тексту та екстрактори тексту.
- SummarizeBot – цей інструмент використовує обробку природної мови та алгоритми машинного навчання для створення резюме тексту кількома мовами.
- One AI – має інструмент підведення підсумків на основі AI, створений за допомогою API мовних навичок One AI. API резюме створює контекстні підсумки текстів, включаючи новини, статті та розмови. Результати є короткими, містять всю відповідну інформацію та можуть використовуватися разом з іншими мовними навичками для покращення результатів шляхом обробки лише ключової інформації.
- BERT – це трансформер, який використовується для подолання обмежень RNN та інших нейронних мереж як довгострокових залежностей. Це попередньо навчена модель, яка природно двонаправлена. Цю попередньо навчену модель можна налаштувати для легкого виконання завдань NLP.

2. Засоби для реалізації блогу з функціями інтелектуального аналізу тексту

2.1 Засоби мови C#

C# – це об'єктно-орієнтована мова програмування з безпечною системою типізації для платформи .NET [7]. Була розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під головуванням Microsoft Research (Microsoft).

Коли розробники говорять C#, нерідко говориться про всі взаємопов’язані технології платформи .NET (Windows Forms, WPF, ASP.NET, Xamarin). І, навпаки, коли говорять .NET, часто мають на увазі саме мову C#, хоча для цієї платформи існує безліч мов. Проте, хоча ці поняття все таки пов'язані, прирівнювати їх невірно. Мова C# була розроблена спеціально для роботи з фреймворком .NET, проте саме поняття фреймворку .NET набагато ширше.

2.1.1 Технологія ASP.NET Core

ASP.NET Core – це кросплатформний, високопродуктивний засіб із відкритим вихідним кодом для створення сучасних хмарних додатків, підключених до Інтернету [8]. ASP.NET Core дає змогу виконувати наступні завдання:

- Створювати веб-додатки та служби, додатки Інтернету речей (IoT) та серверні частини для мобільних додатків;
- Використовувати вибрані засоби розробки в Windows, macOS та Linux;
- Виконувати розгортання в хмарі або локальному середовищі;
- Запускати додатки в середовищі .NET Core.

Фреймворк являє собою збірку, яке об’єднує раніше створені окремі технології ASP.NET MVC і ASP.NET Web API в єдину модель програмування.

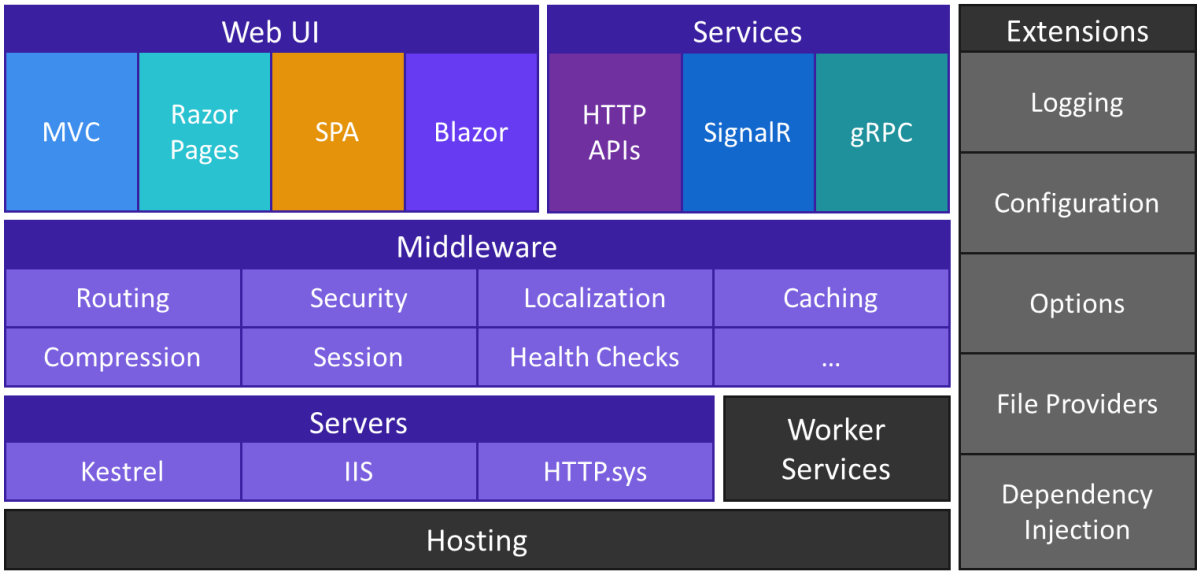


Рис. 2.1. Основні складові ASP.NET Core [10].

ASP.NET Core надає наступні переваги [9]:

- Єдине рішення для створення веб-інтерфейсу користувача та веб-API;
- Легкий процес тестування;
- Razor Pages сприяє написанню коду для сценаріїв сторінок і підвищує його ефективність;
- Blazor дає змогу використовувати в браузері мову C# разом із JavaScript.
- Спільне використання серверної та клієнтської логіки додатків, написаних за допомогою .NET;
- Можливість розробки та використання в ОС Windows, MacOS та Linux;
- Відкритий вихідний код і орієнтація на спільноту розробників;
- Інтеграція сучасних клієнтських платформ і робочих процесів розробки;
- Підтримка розміщення служби віддаленого виклику (RPC) за допомогою gRPC;
- Хмарна система конфігурації на основі середовища;
- Вбудоване введення залежностей;
- Спрощений високопродуктивний модульний конвеєр HTTP-запитів;
- Наступні можливості розміщення:
    - Kestrel;

    - Служби IIS;
    - HTTP.sys;
    - Nginx;
    - Apache;
    - Docker.
- Управління паралельними версіями;
- Надає інструментарій, який спрощує процес сучасної веб-розробки.

2.1.2 Entity Framework

Entity Framework (EF) – це платформа об’єктно-реляційного відображення (ORM) з відкритим вихідним кодом для ADO.NET [11]. Це набір технологій в ADO.NET, який використовується для розробки програмних додатків, орієнтованих на дані. Розробники додатків та програм, які орієнтовані на роботу з даними, зазвичай стикаються з необхідністю досягнення двох дуже різних цілей. Вони повинні моделювати сутності, відношення між ними та логіку бізнес-процесів, які вони вирішують і одночасно працювати з різними механізмами даних, які використовуються для зберігання, обробки та отримання вище зазначених сутностей. Дані можуть бути представлені декількома системами зберігання, які будуть мати різні протоколи. Навіть ті програмні додатки, які працюють з єдиною системою зберігання даних, повинні все таки збалансовувати вимоги системи зберігання та вимоги написання ефективного та підтримуваного коду програми. Для цього якраз і використовують ORM.

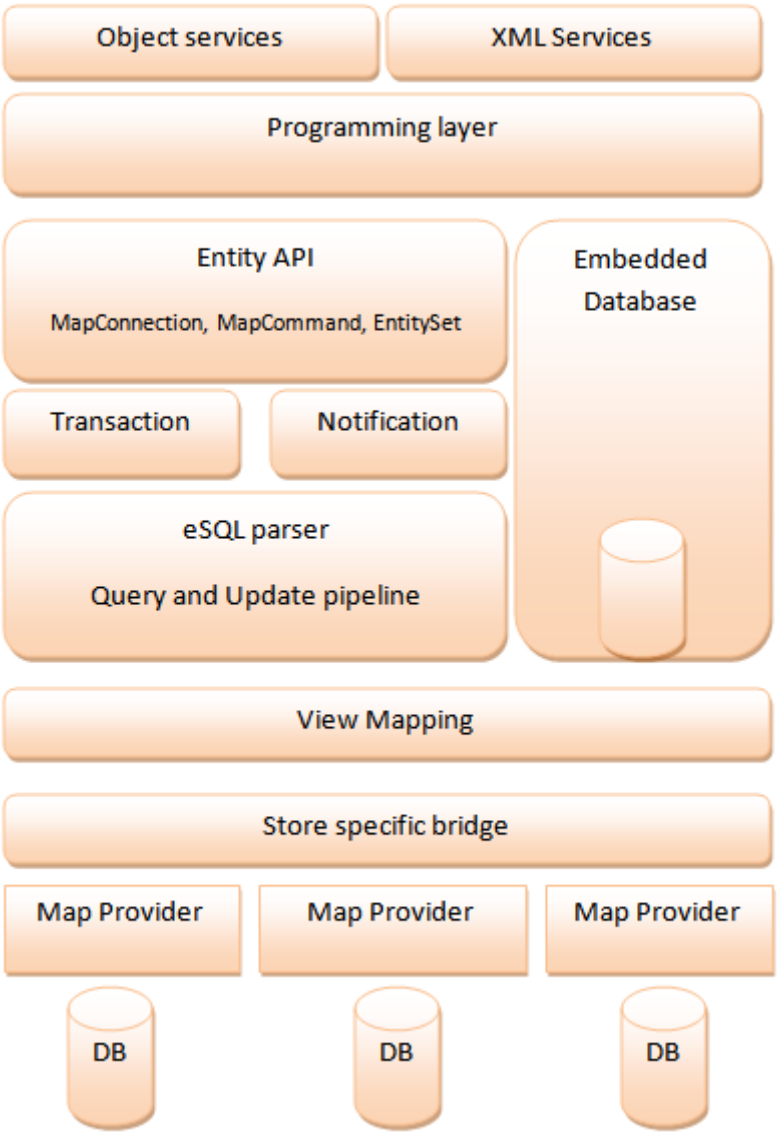


Рис. 2.2. Стек засобів, які використовуються в Entity Framework [11].

Багато інструментів об’єктно-реляційного відображення (ORM) (так звані “об’єктно-реляційні менеджери”) були розроблені, щоб дати змогу розробникам працювати з даними у вигляді предметних об’єктів і властивостей, таких як клієнти та адреси клієнтів, без необхідності глибоко розбиратися в логіці роботи базових таблиць та стовпців баз даних, де ці дані зберігаються. За допомогою ORM розробники можуть працювати на більш високому рівні абстракцій, коли вони мають справу з даними, а також можуть створювати та підтримувати орієнтовані на дані програми з меншою кількістю коду, ніж у традиційних програмах.

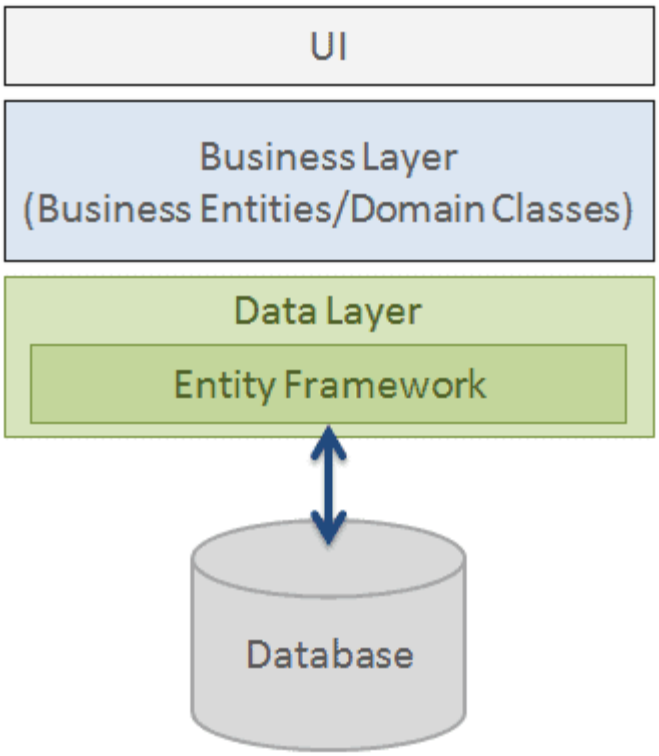


Рис. 2.3 Взаємодія Entity Framework з іншими рівнями додатка [12].

2.2 React – як засіб для реалізації клієнтської частини

React (React.js або ReactJS) – це безкоштовна бібліотека JavaScript із відкритим вихідним кодом для створення інтерфейсів користувача на основі UI компонентів [13]. Її підтримують Meta (раніше Facebook) і спільнота окремих розробників і компаній. React можна використовувати як основу при розробці односторінкових або мобільних додатків. Однак React займається лише керуванням станом і відтворенням цього стану в DOM, тому створення додатків React зазвичай вимагає використання додаткових бібліотек для маршрутизації, а також певної функціональності на стороні клієнта.

React не є фреймворком – він навіть не розрахований виключно на веб-розробку. Він використовується для візуалізації та у зв'язці з іншими бібліотеками.

Основна мета React – мінімізувати помилки, що виникають при розробці інтерфейсів користувача. Це досягається за рахунок використання компонентів

– автономних логічних фрагментів коду, які описують частину інтерфейсу користувача. А вже ці компоненти об'єднуються для створення повноцінного інтерфейсу користувача. React абстрагує більшість роботи з візуалізації, залишаючи вам можливість зосередитися на дизайні.

Переваги використання React [14]:

1) Декларативність

React полегшує створення інтерактивних інтерфейсів користувача. Він надає можливість створювати прості представлення для кожного стану вашої програми і буде ефективно оновлювати та відтворювати лише потрібні компоненти при зміні ваших даних.

Декларативні уявлення роблять ваш код більш передбачуваним та легшим для налагодження.

2) Заснований на компонентах

Надає можливість створювати інкапсульовані компоненти, які керують своїм станом, а потім компонувати їх для створення складних інтерфейсів користувача.

Оскільки логіка компонента написана на JavaScript замість використання шаблонів, ви можете легко передавати різного типу дані через свою програму та зберігати стан поза DOM.

3) Вивчіть лише раз – пишіть будь-де.

React надає вам всі можливості, щоб ви могли програмувати нові функції вашого додатку, не маючи потреби при цьому переписувати існуючий код.

Крім цього, React додатки можуть рендеритись на стороні сервера, використовуючи для цього Node.js, і підтримувати виконання на мобільних платформах, використовуючи при цьому React Native.

2.3 Засоби мови Python

2.3.1 Flask – для створення http сервера

Flask – мікрофреймворк для вебдодатків, створений з використанням Python. Його основу складає інструментарій Werkzeug та рушій шаблонів Jinja2. Поширюється відповідно до умов ліцензії BSD. Це дає розробникам гнучкість і є більш доступною структурою для нових розробників, оскільки ви можете швидко створити веб-програму, використовуючи лише один файл Python. Flask також є розширюваним і не нав’язує певну структуру каталогів та не вимагає складного шаблонного коду перед початком роботи [15].

Flask використовує механізм шаблонів Jinja для динамічного створення HTML-сторінок за допомогою знайомих концепцій Python, таких як змінні, цикли, списки тощо [16].

Flask називається мікрофреймворком, оскільки він не вимагає спеціальних засобів чи бібліотек. У ньому відсутній рівень абстракції для роботи з базою даних, перевірки форм або інші компоненти, які надають широковживані функції за допомогою сторонніх бібліотек. Однак, Flask має підтримку розширень, які надають додаткові властивості таким чином, наче вони були доступні у Flask із самого початку. Існують розширення для встановлення об'єктно-реляційних зв'язків, перевірки форм, контролю процесу завантаження, підтримки різноманітних відкритих технологій аутентифікації та декількох поширених засобів для фреймворку. Розширення оновлюються частіше аніж базовий код.

2.3.2 Бібліотека bert-extractive-summarizer для функцій NLP

Резюмування тексту використовуючи екстрактний тип за допомогою BERT Extractive Summarizer передбачає реферування статті з виділенням ключової інформації за допомогою моделі природної мови BERT. Екстрактне підсумовування має зменшити використання пам’яті, захищаючи цінність вмісту. Інформація про вміст захищена за допомогою екстрактивних підсумків.

BERT використовується для розуміння людських мов через RNN, механізми уваги та трансформатори. Витягнене резюме BERT забезпечує контроль над кількістю речень у підсумку та кількістю символів. Витягнене резюме вибирає найбільш помітні речення та значущі сутності для опису загального значення документа [17].

Вбудовані функції BERT забезпечують створення методу підсумовування тексту за допомогою моделі TextRank і Lead3. BERT використовується для розуміння внутрішньої структури статті. Модель TextRank без контролю використовується для вибору речень на основі нижчої природи корпусу. Lead3 використовується багатьма іншими видавцями контенту як метод узагальнення статей. Lead3 означає використання перших 3 речень для підсумовування всього документа. Таким чином, назва документа і останній заголовок, а також декілька речень документа залишаються в гармонії зі змістом всього документа. Підхід Lead3 бере послідовну інформацію зі статті, щоб створити резюме тексту [17].

Також BERT Extractive Summarizer використовує бібліотеку трансформаторів HuggingFace Pytorch для виконання екстрактивних узагальнень. Це працює за допомогою запуску алгоритму кластеризації, знаходячи речення, які є найближчими до центроїдів кластера. Ця бібліотека також використовує методи кореференції, використовуючи бібліотеку neuralcoref для визначення слів у резюме, які потребують додаткового контексту [18].

3. Написання додатка

Створений додаток складається з трьох частин: сервера .Net, http-сервера Python та Angular клієнта. Тому кожна з цих частин буде описана окремо.

3.1 Реалізація серверної частини

Для написання серверної частини, я використав наступні технології: .Net, Asp.NET Core, Entity Framework. Крім цього, такі технології як JWT та WebSocket використовуються як для серверної, так і для клієнтської частини.

3.1.1 Створення проекту та його структура

Для створення додатку було використано IDE Visual Studio. За основу проекту backend частини взято WebAPI проект. Тобто, це проект який дозволяє запрограмувати основні API інтерфейси для виконання CRUD операцій з даними. При створенні у Visual Studio було вибрано шаблон WebApi.

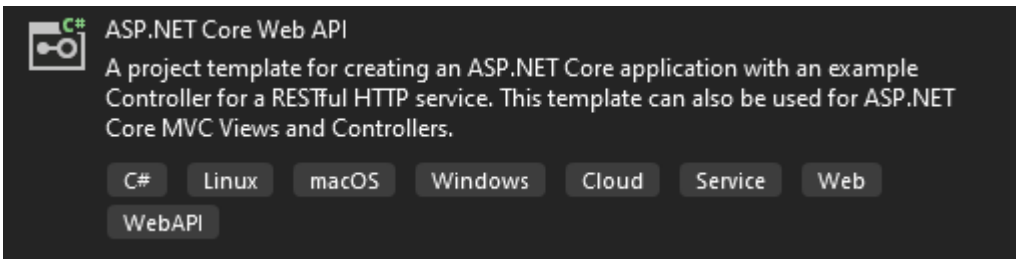


Рис. 3.1. Вікно вибору шаблону проекту.

Структурно весь проект поділений на декілька підпроектів (модулів), кожен з яких відповідає за певну область програми:

- BusinessLogic – модуль, в якому містяться всі сервіси та реалізована основна бізнес логіка додатку;
- DataAccess – модуль, який визначає доступ до бази даних та містить налаштування цього доступу;
- PersonalDiary.Server – основний модуль, в якому знаходять головні файли програми, такі як Startup.cs та Program.cs. Окрім цього, цей модуль містить контролери та інші сервіси необхідні для роботи всього додатка;
- SharedData – модуль, який містить дані, які є спільними для використання у всьому додатку, тобто це різні моделі, DTO та інше.

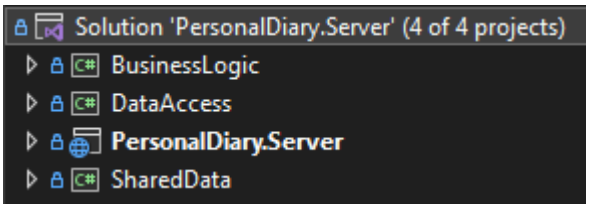


Рис.3.2. Структура проекту.

3.1.2 Реалізація CRUD операцій для основних функцій програми

Для реалізація основного призначення додатку – блогу, необхідно надати користувачу можливість створювати певні публікації в цьому блозі, переглядати їх, редагувати та видаляти. Для цього потрібно реалізувати основні CRUD операції для взаємодії з даними. Це відбувається за допомогою контролера, який містить кінцеві точки для HTTP запитів. Контролер в свою чергу приймає вхідні параметри та викликає певний сервіс. Сервіс вже має доступ до репозиторію, тому він може безпосередньо обробляти дані та виконувати необхідні дії з ними. Далі сервіс повертає отриманий результат назад в контролер, який в свою чергу повертає відповідь щодо HTTP запиту.

Для цього, контролер містить основні методи для виконання CRUD операцій:

GET

```
// GET api/notes/5
[HttpGet("{id}")]
public async Task<IActionResult> Get(Guid id)
{
	var note = await _noteService.GetNoteByIdAsync(id);
	if (note is null)
		return NotFound();
	return Ok(note);
}
```

POST

```
// POST api/notes
[HttpPost]
public async Task<IActionResult> Post(NoteDTO noteDTO)
{
	if (noteDTO == null)
	{
```

```
            return BadRequest();
        }
        try
        {
            await _noteService.CreateNoteAsync(noteDTO);
            return Ok(noteDTO);
        }
        catch (Exception e)
        {
            return BadRequest(e.Message);
        }

}
```

PUT

```
// PUT api/notes/5
[HttpPut("{id}")]
public async Task<IActionResult> Put(Guid id, Note note)
{
        if (note == null)
            return BadRequest();
        note.Id = id;
        var existingNote = await
        _noteService.GetNoteByIdAsNoTrackingAsync(note.Id);
        if (existingNote == null)
            return NotFound();
        var updated = await _noteService.UpdateNoteAsync(note);
        if (updated)
            return Ok(updated);
        return BadRequest();
}
```

DELETE

```
// DELETE api/notes/5
[HttpDelete("{id}")]
public async Task<IActionResult> Delete(Guid id)
{
        var deleted = await _noteService.DeleteNoteAsync(id);
        if (deleted)
            return NoContent();
        return NotFound();
}
```

Сервіс для підтримки цих операцій також містить всі необхідні методи:

CREATE

```
public async Task<bool> CreateNoteAsync(NoteDTO noteDTO)
{
        Note note = _mapper.Map<Note>(noteDTO);

        await _dataContext.Notes.AddAsync(note);
        var created = await _dataContext.SaveChangesAsync();

        return created > 0;
}
```

READ

```
public async Task<Note> GetNoteByIdAsync(Guid noteId) =>
      await _dataContext.Notes.FirstOrDefaultAsync(n => n.Id == noteId);
```

UPDATE

```
public async Task<bool> UpdateNoteAsync(Note note)
{
      _dataContext.Notes.Update(note);

      var updated = await _dataContext.SaveChangesAsync();
      return updated > 0;
}
```

DELETE

```
public async Task<bool> DeleteNoteAsync(Guid noteId)
{
      Note note = await _dataContext.Notes.FirstOrDefaultAsync(n => n.Id ==
noteId);

      if (note == null)
            return false;

      _dataContext.Notes.Remove(note);
      var deleted = await _dataContext.SaveChangesAsync();

      return deleted > 0;
}
```

3.2 Реалізація клієнтської частини

Для написання клієнтської частини, було використано наступний стек технологій: React, TypeScript. Також, як вже зазначалося раніше, такі технології зв’язку та комунікації як JWT та WebSocket використовуються і в клієнтській і в серверній частині додатка.

3.2.1 Створення проекту та його структура

Для створення клієнтської частини додатка було використано IDE Visual Studio Code. Так як додаток буде написаний з використанням фреймворка React, то і сам проект одразу створюється з використанням відповідного шаблону:

npx create-react-app my-app --template typescript

Проект має типову для react-додатків структуру. Основна директорія проекту містить каталоги node_modules, public, src та різні файли конфігурації, такі як: package.json, tsconfig.json та інші:

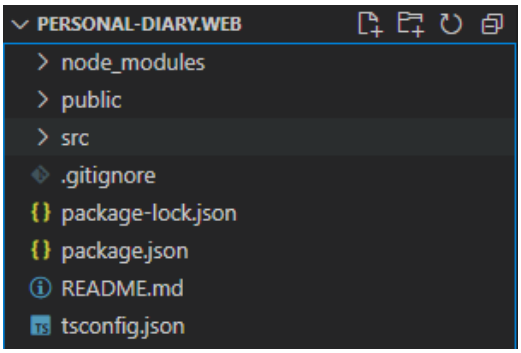


Рис. 3.3 Структура проекту.

Каталог node_modules містить всі підключені до нашого додатка бібліотеки та модулі.

Каталог public містить загальнодоступні для всього додатка елементи. Це якісь картинки, зображення, іконки та інше.

Основним же для проекту являється каталог src. Це власне і є вже сам проект.

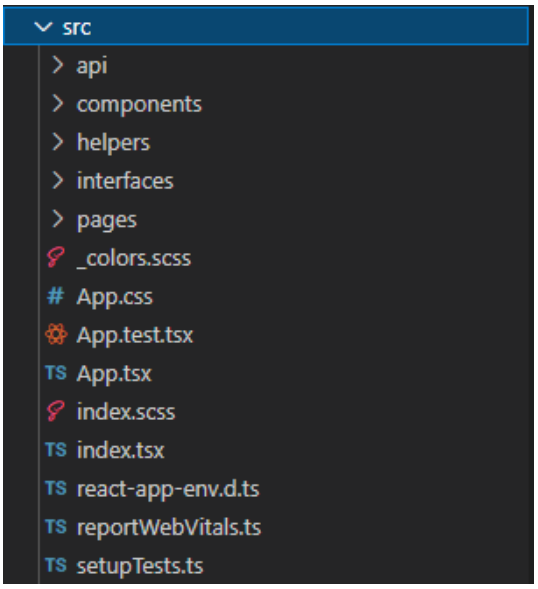


Рис.3.3. Каталог src та його вміст.

Цей каталог містить основні виконувані файли додатка. Це такі файли як index.tsx, який є головним файлом веб-додатка та основна компонента App.tsx. Далі для створення зручної структури я створив наступні каталоги:

- pages – містить компоненти сторінки;
- components – інші візуальні компоненти;
- interfaces – містить опис інтерфейсів та моделей, які будуть необхідні в розробці;
- helpers – функціональні компоненти, які не мають ніякої візуальної частини, а слугують тільки для виконання певних допоміжних цілей;
- api – компоненти, які представляють API функціонал.

3.2.2 Реалізація API

Для реалізації основних HTTP запитів і відповідно базових функцій CRUD я створив модуль API в якому представлені наступні функції:

GET

```
const get = async (url: string, data?: any) => {
	const response = await axios.get(BASE_URL + url, {
		params: data
	});

	return response;
};
```

POST

```
const post = async (url: string, data?: any) => {
	const response = await axios.post(BASE_URL + url, data, {
		headers: {
			"Accept": "application/json",
			"Content-Type": 'application/json',
		},
	});

	return response;
};
```

PUT

```
const put = async (url: string, data?: any) => {
	const response = await axios.put(BASE_URL + url, data, {
		headers: {
			"Accept": "application/json",
			"Content-Type": 'application/json',
		},
	});

	return response;
};
```

DELETE

```
const remove = async (url: string) => {
	const response = await axios.delete(BASE_URL + url);

	return response;
};
```

Всі ці функції використовують систему axios запитів. Такий підхід дуже спрощує та полегшує написання системи HTTP запитів.

Модуль API також містить додаткові перехоплювачі запитів та відповідей. Перехоплювач для запитів використовується для вставлення в заголовок запиту схеми авторизації та відповідно JWT токена:

```
axios.interceptors.request.use(
	config => {
		const token = LocalStorageApi.getToken()
		if (token) {
			config.headers!.Authorization = 'Bearer ' + token;
		}
		return config;
	},
	error => {
		Promise.reject(error)
	}
);
```

Перехоплювач для відповідей призначений для того, щоб одразу на етапі отримання відповіді перехоплювати неавторизованих та неавтентифікованих користувачів (401 та 403 помилки) та перенаправляти їх на сторінку логінування:

```
const setupResponseInterceptors = (navigate: any) => {
	axios.interceptors.response.use(
```

```
            response => response,
            error => {
                  if (error.response) {
                        if (error.response.status === 401 ||
                  error.response.status === 403) {
                              navigate('/loginSwitcher');
                        }
                  }

                  return Promise.reject(error);
            }
      );
};
```

Далі для кожного API-сервісу в додатку були створені окремі модулі, які використовуючи функціонал базового API модулю реалізовують всі необхідні HTTP-запити.

Наприклад, для роботи з записами користувача (Notes) був створений модуль NoteApi. Цей модуль містить наступні функції:

getNotes() – витягнення всіх записів для певного користувача:

```
const getNotes = async (): Promise<INote[]> => {
      const response = await Api.get('notes');

      return response.data;
};
```

getNoteById() – витягнення певного запису за його ідентифікатором:

```
const getNoteById = async (noteId: string | undefined): Promise<INote> => {
      const response = await Api.get(`notes/${noteId}`);

      return response.data;
}
```

createNote() – створення нового запису:

```
const createNote = async (newNote: INoteDTO) => {
      const response = await Api.post(`notes`, newNote);

      return response.data;
}
```

changeNote() – зміна певного запису:

```
const changeNote = async (id: string, note: INoteDTO) => {
      const response = await Api.put(`notes/${id}`, note);

      return response.data;
}
```

deleteNote() – видалення запису:

```
const deleteNote = async (noteId: string | undefined): Promise<void> => {
      const response = await Api.remove(`notes/${noteId}`);
}
```

І така ж структура зроблена для кожного Api-сервісу. Це дає змогу побудувати зручну, ефективну та просту систему HTTP запитів з одночасним додаванням JWT токена до запиту та відловлюванням помилок авторизації та автентифікації.

3.3 Реалізація Python сервера

Для реалізації власне NLP Text Summarization було використано Python бібліотеки. Тож для доступу і користування цими наявними засобами було створено простий HTTP сервер на мові Python.

3.3.1 Створення HTTP сервера

Для створення HTTP сервера було використано бібліотеку Flask. Спочатку було створено сам сервер за допомогою команд Flask:

```
from flask import Flask, request
from flask_cors import CORS

app = Flask(__name__)
CORS(app)
```

Після чого було визначено кінцеву точку сервера для доступу із зовнішніх ресурсів:

```
@app.route("/summarize", methods=['POST'])
def summarize_text():
    content_type = request.headers.get('Content-Type')
    if (content_type == 'application/json'):
        json = request.json
        summary = summarize(json["text"])
        return summary
    else:
        return 'Content-Type not supported!'
```

Таким чином, це звичайний POST запит, до якого можна звертатися за адресою */summarize*. Під час обробки цього запиту перевіряється чи є тіло запиту (контент) і якщо воно є, то цей контент обробляється далі безпосередньо методом підсумовування тексту.

3.3.2 Реалізація методу із функцією підсумовування тексту

Для реалізації функціоналу із генерування анотації до вмісту блогу, було використано бібліотеку *bert-extractive-summarizer*. Засобами цієї бібліотеки було створено метод, який як вхідний параметр приймає текст, обробляє його і на виході видає коротку анотацію до цього тексту:

```
from summarizer import Summarizer, TransformerSummarizer

def summarize(text):
    bert_model = Summarizer()
    bert_summary = ''.join(bert_model(text, min_length=60))
    return(bert_summary)
```

4. Опис додатка

Так як додаток має клієнт-серверну архітектуру, то щоб скористатися ним, немає необхідності встановлювати щось додатково. Сам додаток може бути запущений на сервері і щоб використати його можливості, потрібно просто перейти на сайт за посиланням. Проте, так як зараз додаток ще перебуває на стадії розробки, він не виставлений на публічний хостинг. Тому всі дії будуть показані на локально запущеному додатку.

4.1 Інструкція з використання

Після того як користувач перейшов на сайт, він попадає на сторінку авторизації користувача:

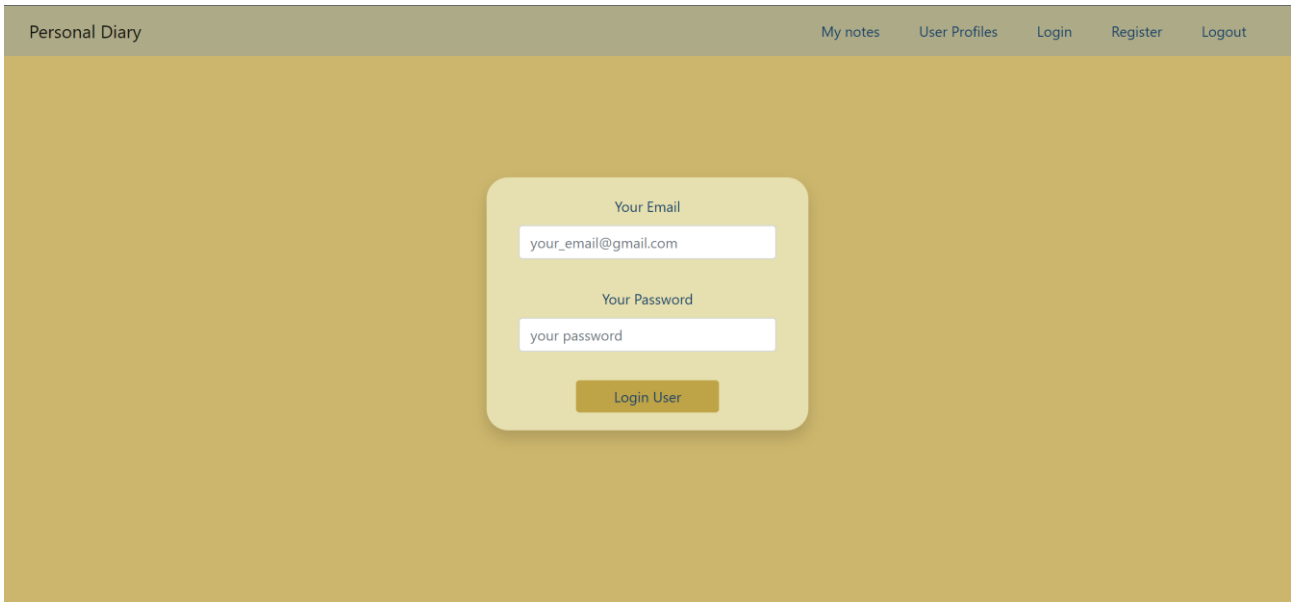


Рис. 4.1. Сторінка логінування.

Якщо людина ще не має свого акаунта на сайті, вона може перейти на сторінку реєстрації з верхньої панелі навігації натиснувши кнопку Register. Сторінка реєстрації виглядає наступним чином:

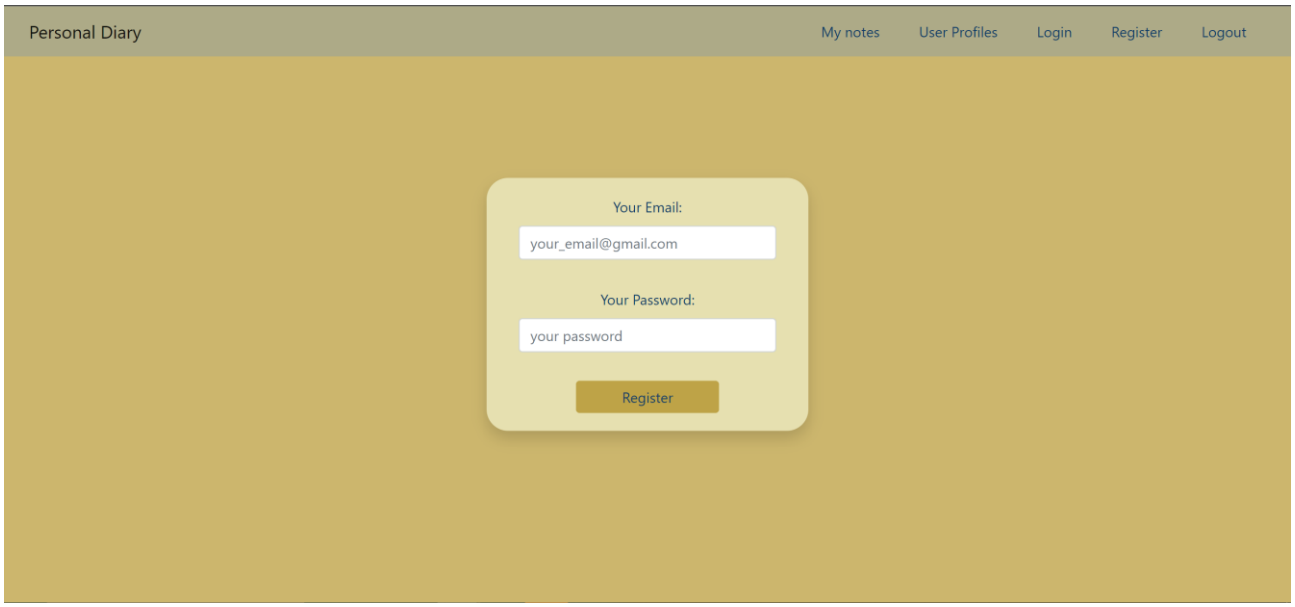


Рис. 4.2. Сторінка реєстрація користувачів.

На сторінці є форма з полями для адреси електронної пошти, імені користувача та пароля. Після введення всіх необхідних даних треба натиснути на кнопку “Register”. Таким чином, користувач буде зареєстрований в системі.

Якщо реєстрація користувача в базі пройде успішно, то ця кнопка зміниться на іншу – “Create Profile”, яка призначено для того, щоб створити профіль користувача:

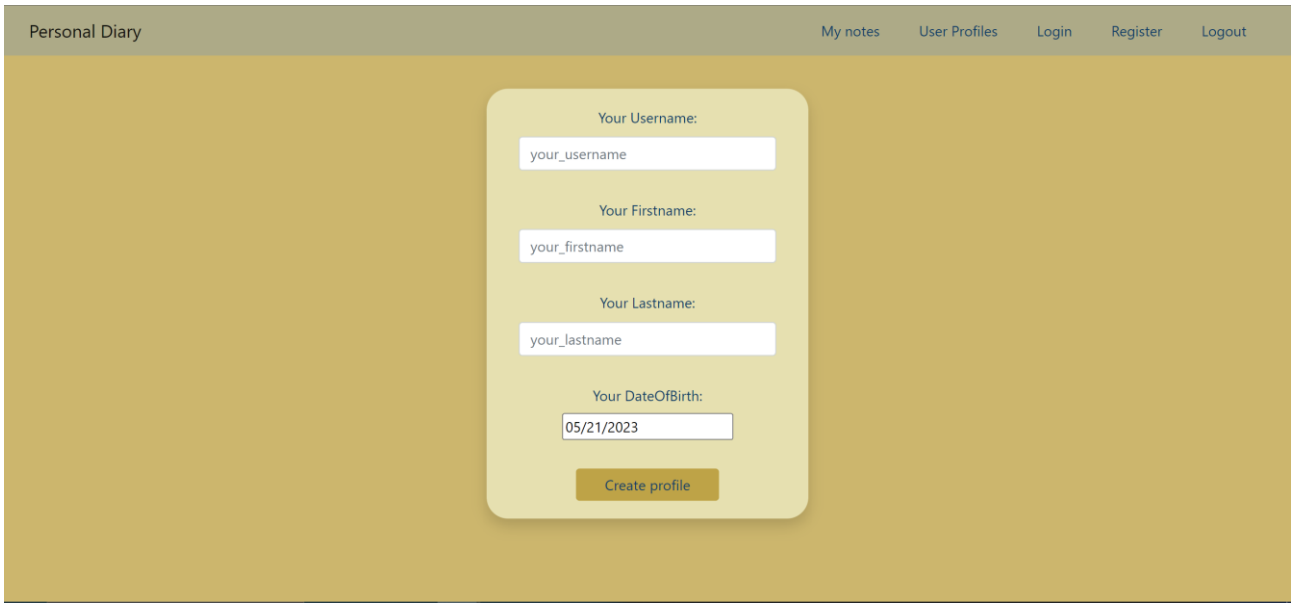


Рис. 4.3. Форма створення профіля користувача.

При натисканні кнопки “Create Profile”, якщо профіль успішно створений, то користувач попадає на сторінку авторизації користувача за допомогою логіна та пароля.

Після введення ваших тільки що створених логіна та пароля, користувач попаде на основну сторінку блогу – стрічку із публікаціями (записами ваших підписників). Якщо користувач ще ні на кого не підписався, то ця сторінка буде пустою. Для пошуку інших користувачів та оформлення підписки на них, користувач може натиснути кнопку User Profiles у верхньому меню додатку.

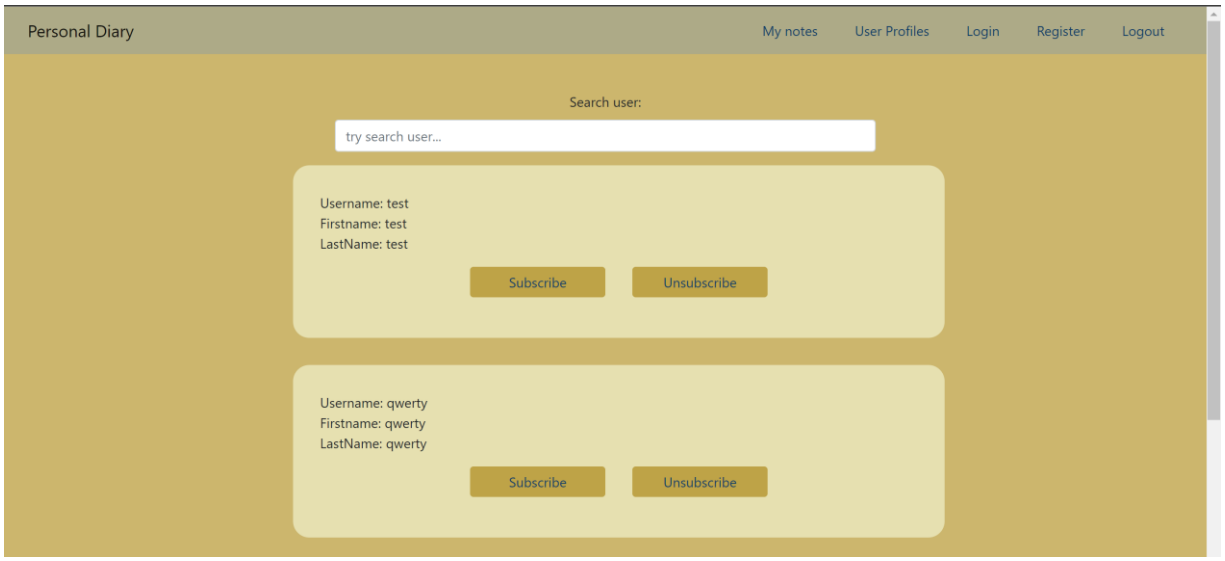


Рис. 4.4. Сторінка пошуку користувачів.

На цій сторінці користувач буде бачити інших користувачів які зареєстровані в блозі. Вгорі є поле для пошуку. Пошук проводиться по імені користувача (Firstname), його прізвищу (Lastname) та нікнейму (Username). Біля кожного знайденого користувача є дві кнопки: Subscribe для підписки на цього користувача та Unsubscribe відповідно для скасування підписки.

Після того як користувач підпишеться на інших користувачів блогу, то його стрічка публікацій наповниться різними записами інших користувачів.

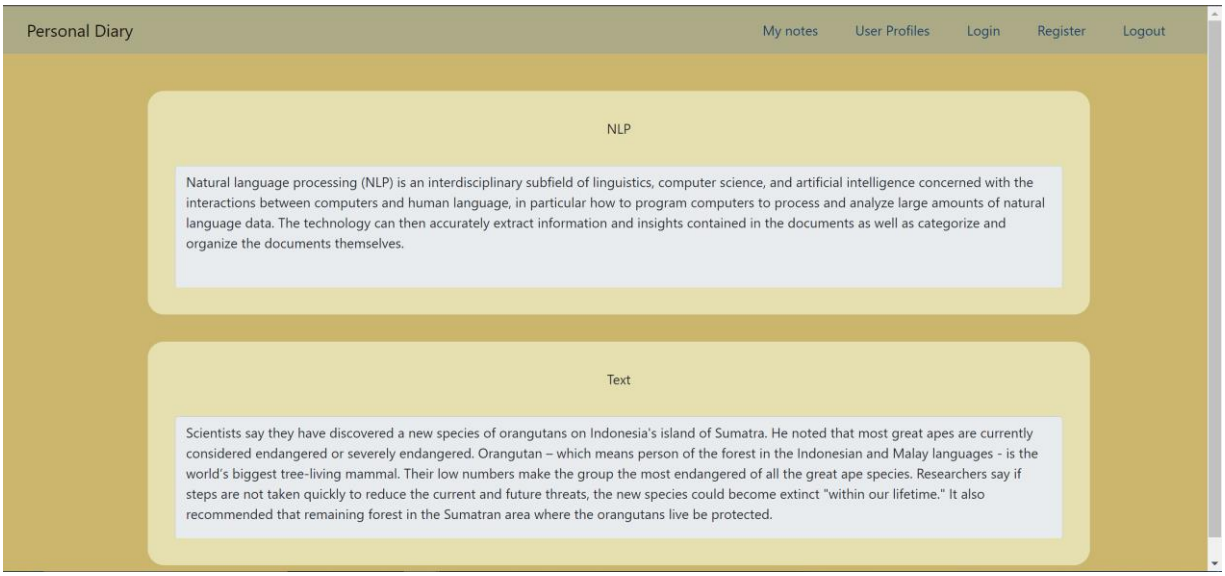


Рис. 4.4. Стрічка записів ваших підписників.

На цій сторінці у публікаціях відображається якраз коротка анотація написаного тексту, а не власне весь зміст певної публікації.

Використовуючи кнопку “My notes” вгорі сторінки можна перейти до сторінки перегляду власних створених публікацій. Вгорі сторінки є кнопка “Create new note”, відповідно для створенні нових записів. Весь інший простір сторінки займають самі записи, які оформлені у вигляді блоків.

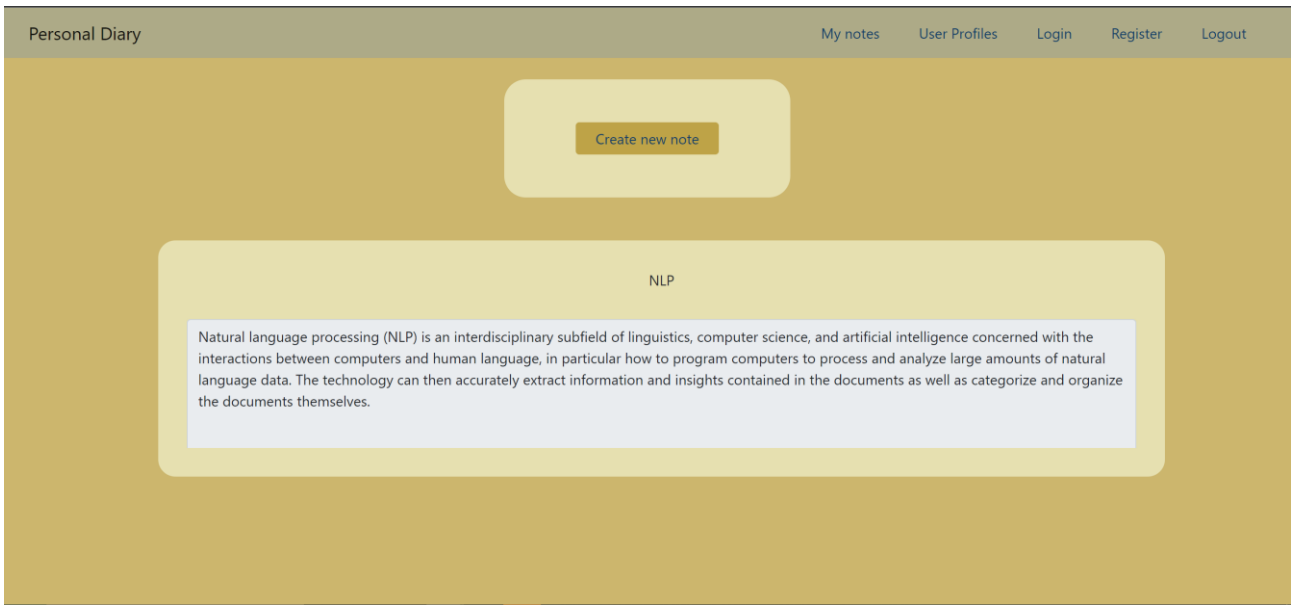


Рис. 4.5. Стрічка ваших особистих записів.

При натисканні на кнопку “Create new note” користувач попадає на сторінку створення нового запису:

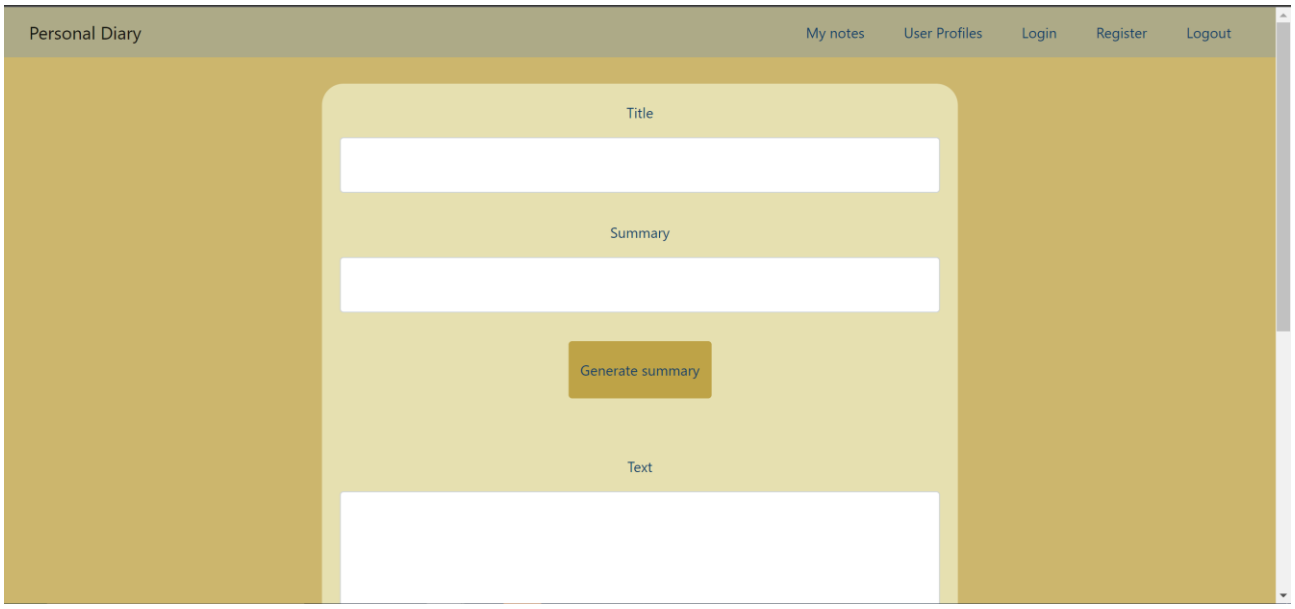


Рис. 4.6. Сторінка створення нового запису.

Під полем “Summary” є кнопка “Generate summary” натиснувши на яку, система автоматично згенерує коротку анотацію всієї публікації і збереже цю анотацію як Summary.

Внизу сторінки є дві кнопки: “Save” та “Discard”:

Рис 4.7. Кнопки “Save” та “Discard”.

При натисканні кнопки “Save”, новий створений запис буде збережений в системі і користувач буде перенаправлений на сторінку перегляду всіх записів, де вже буде його новий запис. А при натисканні кнопки “Discard”, запис, який ми робили, не буде збережений і користувач просто перейде на сторінку перегляду всіх свої записів.

На сторінці (стрічці) перегляду всіх ваших потсів, представлені скорочені варіанти цих записів, щоб все було компактно і не займало багато місця. Тут в

кожному записі відображається не сам текст всього запису, а саме згенерована анотація, яку автор запису згенерував під час його створення. Для перегляду відповідно повного запису, треба натиснути в будь якому місці на блоці того запису, який ми хочемо переглянути. При цьому користувач буде перенаправлений на сторінку перегляду запису:

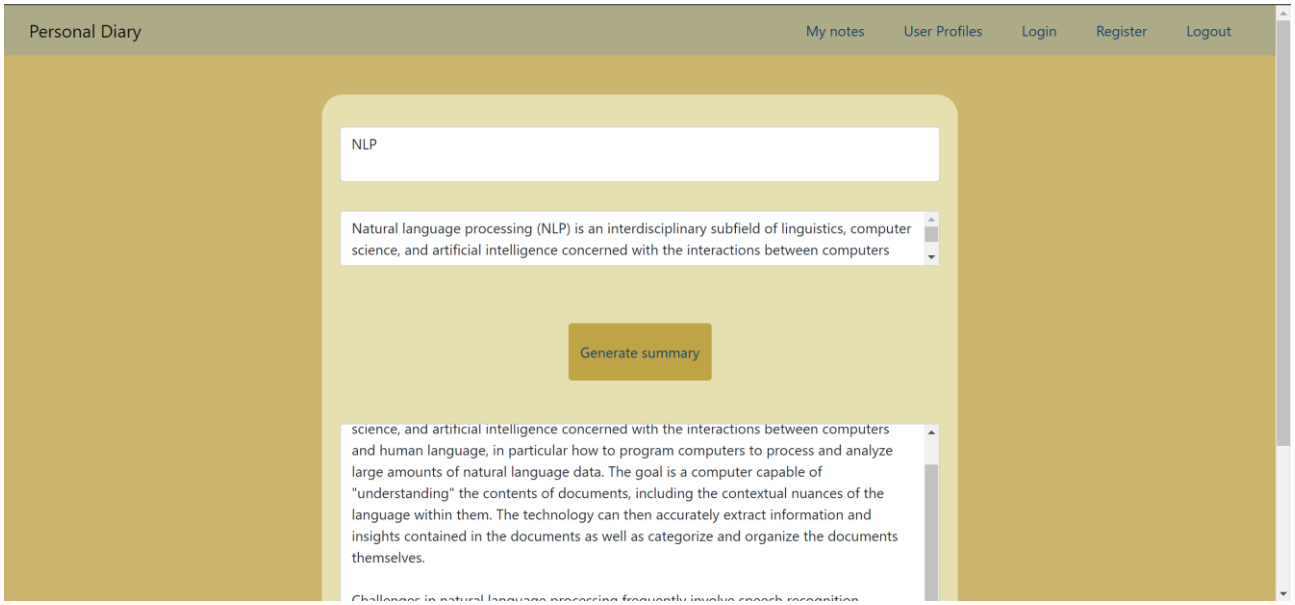


Рис. 4.8. Сторінка перегляду та редагування певного запису.

Вона дуже схожа на сторінку створення запису. Вгорі є поле для вводу заголовка. Далі йде блок для вводу короткої анотації. Тут також є кнопка “Generate summary” за допомогою якої можна згенерувати коротку анотацію Після цих полів вже йде блок самого тексту публікації. Єдине, що відрізняється, це те, що всі ці поля вже будуть заповнені, так як ми переглядаємо вже наявний запис. Тут ми можемо змінити наш запис будь яким чином. Також внизу сторінки є 3 кнопки: “Save”, “Discard changes” та “Delete”:

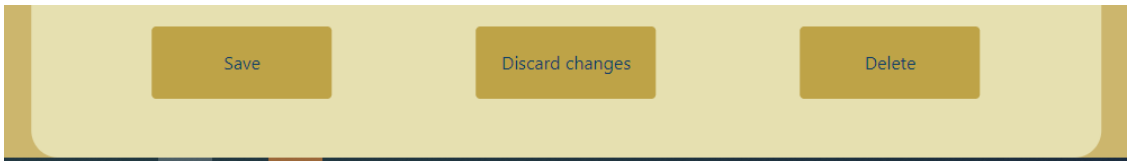


Рис. 4.9. Кнопки “Save”, “Discard changes” та “Delete”.

При натисканні кнопки “Save”, всі внесені зміни до обраного запису будуть збережені. Після натискання кнопки “Discard changes”, навпаки, внесені зміни будуть відмінені, та запис повернеться до його початкового стану. А при натисканні кнопки “Delete”, обраний запис відповідно буде видалений. Для всіх цих кнопок, після їх натискання, користувач перенаправляється на сторінку перегляду всіх записів.

Окремо треба згадати навігаційне меню вгорі сторінки:

Рис. 4.22. Навігаційне меню.

Воно весь час прикріплене догори, тому при прокручуванні сторінки буде зберігати своє місцеположення. В цьому меню наявні п’ять пунктів: “My notes”, “User Profiles”, “Login”, “Register” та “Logout”. При натисканні “My notes” користувач попаде на сторінку перегляду свої особистих публікацій, при натисканні “User Profiles” він буде перенаправлений на сторінку перегляду і пошуку користувачів. При виборі “Login”, користувач перенаправляється на сторінку логінування, при виборі “Register”, відповідно на сторінку реєстрації. А при натисканні кнопки “Logout”, користувач вилогується із системи зі свого акаунта. І так як у незалогінених користувачів немає доступу до сторінки списку всіх записів, то користувач буде перенаправлений на сторінку логінування.

Також при натисканні на назву додатку “Personal Diary” у лівому боці навігаційної панелі, користувач буде перенаправлений на домашню сторінку, тобто на сторінку перегляду всіх записів.

4.2 Напрямки подальшого вдосконалення додатка

Цей додаток має ще деякі аспекти, які можна в подальшому вдосконалювати. По-перше, це покращення власне основної функції додатка – ведення блогу. В цьому напрямку можна розширювати функціонал самого блогу,

додавати різні функції для профілю користувача, додати аватарки користувачів і тд.

Іншим напрямком по вдосконаленню проекту може бути дослідження різних алгоритмів автоматичного аналізу тексту. Наразі в системі для генерування короткої анотації було використано алгоритм BERT та одну з його реалізацій в мові Python. Проте існує ще багато інших методів як для автоматичного генерування анотації так і для інших напрямків автоматичного аналізу тексту. При цьому інші методи і алгоритми можуть бути набагато швидшими і точнішими ніж використаний. Тому цей напрям дослідження та подальшого вдосконалення є дуже важливий для успішного функціонування додатку в цілому.

Висновки

Під час виконання роботи був розроблений доволі зручний та функціональний додаток, який реалізовує систему ведення блогу із автоматичним аналізом тексту. Розроблений додаток виконує всі поставлені перед ним завдання: він реалізовує функціонал блогу. Тобто користувачі мають змогу створювати свої публікації (записи) в ньому, редагувати їх та видаляти. Цей додаток має систему автоматичного аналізу тексту – користувачі під час створення нової публікації мають змогу згенерувати коротку анотацію всього тексту який є в публікації. І третє, створений додаток має клієнт-серверну архітектуру. Це означає, що веб-сайт, яким буде користуватися людина є тільки клієнтом і не несе великого навантаження на ПК користувача. Всі обрахунки та інтелектуальний аналіз текстів будуть відбуватися на серверній частині. Також внаслідок цього, всі збережені дані користувача, а саме: його публікації, дані профілю та інше будуть зберігатися в базі даних на сервері. Тобто, користувач матиме доступ до своїх даних з будь-якого пристрою, треба тільки ввійти в систему під своїми даними. Згаданий застосунок був написаний з використанням таких технологій як .Net, Asp.NET Core, Microsoft SQL, Entity Framework, React, Python. Для його написання я використовував наступні IDE: Microsoft Visual Studio, Visual Studio Code та Microsoft SQL Server Management Studio. Подальші дослідження на тематику цієї роботи можуть проводитися в декількох напрямках. По-перше, це покращення функціоналу самого блогу. По-друге, це дослідження та покращення системи автоматичного аналізу тексту. Тому, розроблений додаток має широкі можливості свого розвитку і продовження розробки в майбутньому. Що, зі свого боку, може перетворитися в чудовий стартап-проект.

Використана література

1. “A COMPLETE GUIDE TO Natural Language Processing” [Електронний ресурс] – Режим доступу до ресурсу: https://www.deeplearning.ai/resources/natural-language-processing/
2. “natural language processing (NLP)” [Електронний ресурс] – Режим доступу до ресурсу: https://www.techtarget.com/searchenterpriseai/definition/natural-language-processing-NLP
3. “Text summarization using NLP” [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/analytics-vidhya/text-summarization-using-nlp-3e85ad0c6349
4. “Text Summarization Approaches for NLP – Practical Guide with Generative Examples” [Електронний ресурс] – Режим доступу до ресурсу: https://www.machinelearningplus.com/nlp/text-summarization-approaches-nlp-example/
5. “TEXT SUMMARIZATION IN NLP” [Електронний ресурс] – Режим доступу до ресурсу: https://www.topcoder.com/thrive/articles/text-summarization-in-nlp
6. “What is summarizing?” [Електронний ресурс] – Режим доступу до ресрусу: https://www.oneai.com/learn/text-summarizer
7. “C Sharp” [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/C_Sharp_(programming_language)
8. “ASP.NET Core” [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/dotnet/aspnetcore
9. “Вибір між ASP.NET 4.x і ASP.NET Core” [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/aspnet/core/fundamentals/choose-aspnet-framework?view=aspnetcore-6.0

10. “What’s New in ASP.NET Core in .NET 6” [Електронний ресурс] – Режим доступу до ресурсу: https://www.codemag.com/Article/2111062/What%E2%80%99s-New-in-ASP.NET-Core-in-.NET-6
11. “Entity Framework” [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Entity_Framework
12. “What is Entity Framework?” [Електронний ресурс] – Режим доступу до ресурсу: https://www.entityframeworktutorial.net/what-is-entityframework.aspx
13. “React (JavaScript library)” [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/React_(JavaScript_library)
14. “React. JavaScript-бібліотека для створення користувацьких інтерфейсів” [Електронний ресурс] – Режим доступу до ресурсу: https://uk.reactjs.org/
15. “Flask” [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Flask
16. “How To Make a Web Application Using Flask in Python 3” [Електронний ресрус] – Режим доступу до ресрусу: https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3
17. “Extractive Summarization with BERT Extractive Summarizer” [Електронний ресрус] – Режим доступу до ресрусу: https://www.holisticseo.digital/python-seo/summarize/
18. “Bert Extractive Summarizer” [Електронний ресрус] – Режим доступу до ресрусу: https://pypi.org/project/bert-extractive-summarizer/

Додатки

Посилання на репозиторії з вихідним кодом проекту

- Репозиторій з вихідним кодом серверної .NET частини проекту: https://github.com/Vinnytile/PersonalDiary.Server;
- Репозиторій з вихідним кодом серверної Python частини проекту: https://github.com/Vinnytile/blog.python.server;
- Репозиторій з вихідним кодом клієнтської частини проекту: https://github.com/Vinnytile/personal-diary.web.

Додаток 1

Файл NotesController.cs

```
using BusinessLogic.Services;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using SharedData.Models;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace PersonalDiary.Server.Controllers
{
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("api/[controller]")]
    [ApiController]
    public class NotesController : ControllerBase
    {
        private readonly INoteService _noteService;

        public NotesController(INoteService noteService)
        {
            _noteService = noteService;
        }

        [HttpGet("{userIdentityId}")]
        public async Task<IActionResult> GetNotes(Guid userIdentityId)
        {
            List<Note> notes = await
_noteService.GetAllNotesByUserIdentityIdAsync(userIdentityId);

            return Ok(notes);
        }

        [HttpGet("observed/{userIdentityId}")]
        public async Task<IActionResult> GetObservedNotes(Guid userIdentityId)
        {
            List<Note> notes = await
_noteService.GetObservedNotesByUserIdentityIdAsync(userIdentityId);

            return Ok(notes);
        }
```

```
        // GET api/notes/5
        [HttpGet("note/{id}")]
        public async Task<IActionResult> Get(Guid id)
        {
            var note = await _noteService.GetNoteByIdAsync(id);

            if (note is null)
                return NotFound();

            return Ok(note);
        }

        // POST api/notes
        [HttpPost]
        public async Task<IActionResult> Post(NoteDTO noteDTO)
        {
            if (noteDTO == null)
            {
                return BadRequest();
            }

            try
            {
                await _noteService.CreateNoteAsync(noteDTO);
                return Ok(noteDTO);
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

        // PUT api/notes/5
        [HttpPut("{id}")]
        public async Task<IActionResult> Put(Guid id, Note note)
        {
            if (note == null)
                return BadRequest();

            note.Id = id;

            var existingNote = await
_noteService.GetNoteByIdAsNoTrackingAsync(note.Id);

            if (existingNote == null)
                return NotFound();

            var updated = await _noteService.UpdateNoteAsync(note);

            if (updated)
                return Ok(updated);

            return BadRequest();
        }

        // DELETE api/notes/5
        [HttpDelete("{id}")]
        public async Task<IActionResult> Delete(Guid id)
        {
            var deleted = await _noteService.DeleteNoteAsync(id);

            if (deleted)
                return NoContent();

            return NotFound();
```

```
        }
    }
}
```

Додаток 2

Файл NoteService.cs

```
using AutoMapper;
using DataAccess.Context;
using Microsoft.EntityFrameworkCore;
using SharedData.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BusinessLogic.Services
{
    public class NoteService : INoteService
    {
        private readonly DataContext _dataContext;
        private readonly IMapper _mapper;

        public NoteService(DataContext dataContext, IMapper mapper)
        {
            _dataContext = dataContext;
            _mapper = mapper;
        }

        public async Task<List<Note>> GetAllNotesByUserIdentityIdAsync(Guid
userIdentityId) =>
            await _dataContext.Notes.Where(x => x.UserIdentityFID ==
userIdentityId).ToListAsync();

        public async Task<List<Note>> GetObservedNotesByUserIdentityIdAsync(Guid
userIdentityId)
        {
            var observedUserIdentititesId = await _dataContext.Subscriptions.Where(s
=> s.SubscriberFID == userIdentityId).Select(s => s.ObservableFID).ToListAsync();

            var observedNotes = await _dataContext.Notes.Where(n =>
observedUserIdentititesId.Contains(n.UserIdentityFID)).ToListAsync();

            return observedNotes;
        }

        public async Task<Note> GetNoteByIdAsync(Guid noteId) =>
            await _dataContext.Notes.FirstOrDefaultAsync(n => n.Id == noteId);

        public async Task<Note> GetNoteByIdAsNoTrackingAsync(Guid noteId) =>
            await _dataContext.Notes.AsNoTracking().FirstOrDefaultAsync(n => n.Id ==
noteId);

        public async Task<bool> CreateNoteAsync(NoteDTO noteDTO)
        {
            Note note = _mapper.Map<Note>(noteDTO);
            note.CreatedAt = DateTime.Now;

            await _dataContext.Notes.AddAsync(note);
            var created = await _dataContext.SaveChangesAsync();

            return created > 0;
        }
```

```
        public async Task<bool> UpdateNoteAsync(Note note)
        {
            note.ChangedAt = DateTime.Now;
            _dataContext.Notes.Update(note);
            var updated = await _dataContext.SaveChangesAsync();

            return updated > 0;
        }

        public async Task<bool> DeleteNoteAsync(Guid noteId)
        {
            Note note = await _dataContext.Notes.FirstOrDefaultAsync(n => n.Id ==
noteId);

            if (note == null)
                return false;

            _dataContext.Notes.Remove(note);
            var deleted = await _dataContext.SaveChangesAsync();

            return deleted > 0;
        }
    }
}
```

Додаток 3

Файл UserProfileController.cs

```
using BusinessLogic.Services.Interfaces;
using Microsoft.AspNetCore.Mvc;
using SharedData.Models.User;
using System.Threading.Tasks;
using System.Collections.Generic;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using System;

namespace PersonalDiary.Server.Controllers
{
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [Route("api/[controller]")]
    [ApiController]
    public class UserProfileController : ControllerBase
    {
        private readonly IUserProfileService _userProfileService;

        public UserProfileController(IUserProfileService userProfileService)
        {
            _userProfileService = userProfileService;
        }

        [HttpGet]
        public async Task<IActionResult> GetAllUserProfiles()
        {
            List<UserProfile> notes = await _userProfileService.GetAllUserProfiles();

            return Ok(notes);
        }

        [HttpPost("subscribeUser")]
        public async Task<IActionResult> SubscribeUser(SubscriptionDTO
subscriptionDTO)
        {
```

```
            if (subscriptionDTO == null)
            {
                return BadRequest();
            }

            try
            {
                await _userProfileService.SubscribeUserAsync(subscriptionDTO);
                return Ok(subscriptionDTO);
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }

        [HttpPost("unsubscribeUser")]
        public async Task<IActionResult> UnsubscribeUser(SubscriptionDTO
subscriptionDTO)
        {
            if (subscriptionDTO == null)
            {
                return BadRequest();
            }

            try
            {
                await _userProfileService.UnsubscribeUserAsync(subscriptionDTO);
                return Ok(subscriptionDTO);
            }
            catch (Exception e)
            {
                return BadRequest(e.Message);
            }
        }
    }
}
```

Додаток 4

Файл UserProfileService.cs

```
using AutoMapper;
using BusinessLogic.Services.Interfaces;
using DataAccess.Context;
using Microsoft.EntityFrameworkCore;
using SharedData.Models.User;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace BusinessLogic.Services.Implementation
{
    public class UserProfileService : IUserProfileService
    {
        private readonly DataContext _dataContext;
        private readonly IMapper _mapper;

        public UserProfileService(DataContext dataContext, IMapper mapper)
        {
            _dataContext = dataContext;
            _mapper = mapper;
        }
```

```
        public async Task<List<UserProfile>> GetAllUserProfiles() =>
            await _dataContext.UserProfiles.ToListAsync();

        public async Task<UserProfile> GetUserProfileByIdAsync(Guid userProfileId) =>
            await _dataContext.UserProfiles.FirstOrDefaultAsync(u => u.Id ==
userProfileId);

        public async Task<bool> CreateUserProfileAsync(UserProfileDTO UserProfileDTO)
        {
            UserProfile userProfile = _mapper.Map<UserProfile>(UserProfileDTO);
            userProfile.CreatedAt = DateTime.Now;

            await _dataContext.UserProfiles.AddAsync(userProfile);
            var created = await _dataContext.SaveChangesAsync();

            return created > 0;
        }

        public async Task<bool> UpdateUserProfileAsync(UserProfile userProfile)
        {
            userProfile.ChangedAt = DateTime.Now;
            _dataContext.UserProfiles.Update(userProfile);
            var updated = await _dataContext.SaveChangesAsync();

            return updated > 0;
        }

        public async Task<bool> SubscribeUserAsync(SubscriptionDTO subscriptionDTO)
        {
            Subscription subscription = _mapper.Map<Subscription>(subscriptionDTO);

            var observableIdentitityFID = _dataContext.UserProfiles.FirstOrDefault(up
=> up.Id == subscriptionDTO.ObservableFID).UserIdentityFID;
            subscription.ObservableFID = observableIdentitityFID;
            subscription.CreatedAt = DateTime.Now;

            await _dataContext.Subscriptions.AddAsync(subscription);
            var subscribed = await _dataContext.SaveChangesAsync();

            return subscribed > 0;
        }

        public async Task<bool> UnsubscribeUserAsync(SubscriptionDTO subscriptionDTO)
        {
            Subscription subscription = _mapper.Map<Subscription>(subscriptionDTO);

            var observableIdentitityFID = _dataContext.UserProfiles.FirstOrDefault(up
=> up.Id == subscriptionDTO.ObservableFID).UserIdentityFID;
            subscription.ObservableFID = observableIdentitityFID;

            var deleteSubscription = _dataContext.Subscriptions.FirstOrDefault(s =>
s.SubscriberFID == subscription.SubscriberFID && s.ObservableFID ==
subscription.ObservableFID);

            var subscriptionList =
_dataContext.Subscriptions.Remove(deleteSubscription);
            var unsubscribed = await _dataContext.SaveChangesAsync();

            return unsubscribed > 0;
        }
    }
}
```

Додаток 6

Файл Api.ts

```
import axios from "axios";
import LocalStorageApi from './LocalStorageApi'

const BASE_URL = "https://localhost:5001/api/";
const PYTHON_URL = "http://localhost:5003/"

axios.interceptors.request.use(
  config => {
    const token = LocalStorageApi.getToken()
    if (token) {
      config.headers!.Authorization = 'Bearer ' + token;
    }
    return config;
  },
  error => {
    Promise.reject(error)
  }
);

const setupResponseInterceptors = (navigate: any) => {
  axios.interceptors.response.use(
    response => response,
    error => {
      if (error.response) {
        if (error.response.status === 401 || error.response.status === 403) {
          navigate('/login');
        }
      }

      return Promise.reject(error);
    }
  );
};

const get = async (url: string, data?: any) => {
    const response = await axios.get(BASE_URL + url, {
      params: data
    });

    return response;
};

const post = async (url: string, data?: any) => {
  const response = await axios.post(BASE_URL + url, data, {
    headers: {
      "Accept": "application/json",
      "Content-Type": 'application/json',
    },
```

```
  });

  return response;
};

const put = async (url: string, data?: any) => {
  const response = await axios.put(BASE_URL + url, data, {
    headers: {
      "Accept": "application/json",
      "Content-Type": 'application/json',
    },
  });

  return response;
};

const remove = async (url: string) => {
    const response = await axios.delete(BASE_URL + url);

    return response;
};

const postPython = async (url: string, data?: any) => {
  const response = await axios.post(PYTHON_URL + url, data, {
    headers: {
      "Accept": "application/json",
      "Content-Type": 'application/json',
    },
  });

  return response;
};

const exportedObject = {
    get,
    post,
    put,
    remove,
    postPython,
    setupResponseInterceptors
};

export default exportedObject;
```

Додаток 7

Файл NoteApi.ts

```
import { INote, INoteDTO } from "../interfaces/interfaces";
import Api from "./Api";
```

```
const getNotes = async (userIdentityId: string | undefined): Promise<INote[]> => {
    const response = await Api.get(`notes/${userIdentityId}`);

    return response.data;
};

const getObservedNotes = async (userIdentityId: string | undefined):
Promise<INote[]> => {
    const response = await Api.get(`notes/observed/${userIdentityId}`);

    return response.data;
};

const getNoteById = async (noteId: string | undefined): Promise<INote> => {
    const response = await Api.get(`notes/note/${noteId}`);

    return response.data;
}

const createNote = async (newNote: INoteDTO) => {
    const response = await Api.post(`notes`, newNote);

    return response.data;
}

const changeNote = async (id: string, note: INoteDTO) => {
    const response = await Api.put(`notes/${id}`, note);

    return response.data;
}

const deleteNote = async (noteId: string | undefined): Promise<void> => {
    const response = await Api.remove(`notes/${noteId}`);
}

const generateNoteSummary = async (noteJson: string) => {
    const response = await Api.postPython(`summarize`, noteJson);
    //const response = await Api.getPython(`test`);

    return response.data;
}

const exportedObject = {
    getNotes,
    getObservedNotes,
    getNoteById,
    createNote,
    changeNote,
    deleteNote,
    generateNoteSummary
```

```
};

export default exportedObject;
```

Додаток 8

Файл UserProfileApi.ts

```
import { ISubscriptionDTO, IUserProfile } from "../interfaces/interfaces"
import Api from "./Api"

const getUserProfiles = async (): Promise<IUserProfile[]> => {
    const response = await Api.get(`userProfile`);

    return response.data;
};

const subscribeUser = async (subscription: ISubscriptionDTO) => {
    const response = await Api.post(`userProfile/subscribeUser`, subscription);

    return response.data;
}

const unsubscribeUser = async (subscription: ISubscriptionDTO) => {
    const response = await Api.post(`userProfile/unsubscribeUser`, subscription);

    return response.data;
}

const exportedObject = {
    getUserProfiles,
    subscribeUser,
    unsubscribeUser
};
export default exportedObject;
```

Додаток 9

Файл App.tsx

```
import React from 'react';
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
import { SelfNotesLinePage } from './pages/SelfNotesLinePage'
import { NoteViewPage } from './pages/NoteViewPage'
import { NewNoteFormPage } from './pages/NewNoteFormPage';
import { LoginPage } from './pages/LoginPage'
import { RegisterPage } from './pages/RegisterPage'
import { ErrorPage } from './pages/ErrorPage'
import { NavigateFunction } from './helpers/NavigateFunction'
import { RegisterFacePage } from './pages/RegisterFacePage';
import { LoginFacePage } from './pages/LoginFacePage';
```

```
import { LoginSwitcherPage } from './pages/LoginSwitcherPage';
import { Navbar } from './components/Navbar/Navbar';
import { RegisterProfilePage } from './pages/RegisterProfilePage';
import { UserProfilesLinePage } from './pages/UserProfilesLinePage';
import { GeneralNotesLinePage } from './pages/GeneralNotesLinePage';
import { NotePreviewPage } from './pages/NotePreviewPage';

const App: React.FC = () => {
  return (
    <BrowserRouter>
      <NavigateFunction />
      <Navbar />
      <Routes>
        <Route element={<Navigate replace to="/generalNotes" />} path="/" />
        <Route element={<SelfNotesLinePage />} path="/selfNotes" />
        <Route element={<GeneralNotesLinePage />} path="/generalNotes" />
        <Route element={<NoteViewPage />} path="/note/:id" />
        <Route element={<NotePreviewPage />} path="/notepreview/:id" />
        <Route element={<NewNoteFormPage />} path="/newNote" />
        <Route element={<LoginPage />} path="/login" />
        <Route element={<LoginFacePage />} path="/loginFace" />
        <Route element={<RegisterPage />} path="/register" />
        <Route element={<RegisterFacePage />} path="/registerFace/:userId" />
        <Route element={<RegisterProfilePage />}
path="/registerProfile/:userIdentityId" />
        <Route element={<LoginSwitcherPage />} path="/loginSwitcher" />
        <Route element={<UserProfilesLinePage />} path="/userProfiles" />
        <Route element={<ErrorPage />} path="*" />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

Додаток 10

Файл server.py

```
from flask import Flask, request
from flask_cors import CORS
from summarizer import Summarizer, TransformerSummarizer

app = Flask(__name__)
CORS(app)

@app.route("/summarize", methods=['POST'])
def summarize_text():
    content_type = request.headers.get('Content-Type')
    if (content_type == 'application/json'):
        json = request.json
        summary = summarize(json["text"])
        return summary
    else:
        return 'Content-Type not supported!'
```

```python
def summarize(text):
    bert_model = Summarizer()
    bert_summary = ''.join(bert_model(text, min_length=60))
    return(bert_summary)
```