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

КУРСОВА РОБОТА

на тему: «Програмне забезпечення для організації та зберігання спортивних тренувань»

Виконав:
студент групи ФеІм-12с
Бурець Валентин

Науковий керівник:
асистент
Юзевич В.М.

Львів – 2023

АНОТАЦІЯ

Мій навчальний проект передбачав розробку та впровадження спеціалізованого програмного забезпечення, яке ефективно організовує та архівує дані про спортивну підготовку.

Програма призначена для використання як тренерами, так і їхніми учнями, що є комплексним рішенням для управління спеціальними спортивними тренуваннями.

Система містить функції для організації та зберігання спортивних тренувань, а моє програмне забезпечення складається з двох частин, які реалізують клієнт-серверну архітектуру з використанням різних технологій.

Клієнтська сторона програмується за допомогою TypeScript як мови програмування, а також бібліотек React.JS, таблиць стилів SASS і бібліотек компонентів графічного інтерфейсу React Bootstrap.

По суті, ця система використовує такі передові технології, як остання версія мови програмування C#, що підтримується .Net 6 і Visual Studio 2022 для операцій на стороні сервера, а для керування базами даних використовується MS SQL Server.

Внутрішній або клієнтський інтерфейс розроблений для роботи з кількома сучасними веб-браузерами, що робить його сумісним між платформами на кількох пристроях із попередньо встановленими браузерами. Для розгортання серверної сторони необхідно встановити веб-сервери SQL Server і IIS.

Загальний обсяг роботи 97 сторінок.

Крім того, ця курсова робота містить економічний розділ, який розраховує та визначає потенційні витрати на розробку цього програмного забезпечення. Крім того, було зроблено оцінку можливого доходу, який може виникнути від продажу цієї веб-системи. Крім того, було розроблено план, що деталізує підхід до його розробки.

ABSTRACT

My academic project entailed designing and implementing specialized software that efficiently organizes and archives athletic preparation data.

The application is designed to be used by both coaches and their trainees providing a comprehensive solution for sport specific training management.

The system includes functions for organizing and storing sports training, and my software consists of two parts that implement a client-server architecture using various technologies.

The client-side is programmed using TypeScript as the programming language, along with React.JS libraries, SASS style sheets and React Bootstrap graphic interface component libraries.

At its core, this system benefits from advanced technology such as C# programming language's latest version supported by .Net 6 and Visual Studio 2022 for server-side operations while opting for MS SQL Server to manage databases.

The front-end or client side interface is designed to accommodate several modern web-browsers - rendering it cross-platform compatible across multiple devices with pre-installed browsers. To deploy the server side, it is necessary to have SQL Server and IIS web servers installed.

The total amount of work is 97 pages.

Additionally, this thesis contains an economic section that calculates and identifies potential expenses for developing this software. Furthermore, an estimation was made regarding the possible revenue that could arise from vending this web system. Additionally, a plan detailing the approach for executing its development was devised.

ЗМІСТ

ВСТУП 9

РОЗДІЛ 1. АНАЛІЗ ЗАСТОСУВАННЯ СУЧАСНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У СПОРТИВНІЙ ІНДУСТРІЇ 10

1.1. Тренування 10

1.2. Аналіз цільової аудиторії. 10

1.2.1. Person 1 11

1.2.2. Person 2 12

1.3. SWOT аналіз 12

1.4. Продукти-аналоги 13

1.5. Висновки 16

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДЛЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОРГАНІЗАЦІЇ ТА ЗБЕРІГАННЯ СПОРТИВНИХ ТРЕНУВАНЬ 17

2.1. Постановка задачі для програмного забезпечення для організації та зберігання спортивних тренувань 17

2.2. Специфікація вимог до програмного забезпечення для організації та зберігання спортивних тренувань 17

2.2.1. Вступ 17

2.2.2. Загальний опис 17

2.2.3. Класи користувачів та їхні характеристики 18

2.2.4. Середовище функціонування 19

2.2.5. Характеристики системи 20

2.3.6 Атрибути якості програмного продукту 24

2.3.7. Вимоги зовнішніх інтерфейсів 24

2.3.8. Інші нефункціональні вимоги 25

2.4. Висновки 25

РОЗДІЛ 3. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОРГАНІЗАЦІЇ ТА ЗБЕРІГАННЯ СПОРТИВНИХ ТРЕНУВАНЬ 26

3.1. Проєктування архітектури та об’єктної моделі програмного забезпечення для організації та зберігання спортивних тренувань. 26

3.1.1. Діаграма прецедентів (Додаток А.1). 26

3.1.2. Діаграма послідовності (Додаток Б.2). 27

3.1.3. Діаграма діяльності (Додаток Б.3)..... 28

3.1.4. Діаграма класів. 28

3.1.5. Діаграма інтерфейсів. 29

3.1.6. Архітектурна діаграма. 30

3.1.7. Діаграма розгортання..... 30

3.2. Проєктування бази даних до програмного забезпечення для організації та зберігання спортивних тренувань 31

3.3. Висновки. 35

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОРГАНІЗАЦІЇ ТА ЗБЕРІГАННЯ СПОРТИВНИХ ТРЕНУВАНЬ..... 37

4.1. Реалізація серверної частини програмної системи для організації та зберігання спортивних тренувань..... 37

4.2. Реалізація клієнтської частини прорамного забезпечення для органіхзації та зберігання спортивних тренувань..... 39

4.3. Опис роботи системи 41

4.4. Проведення тестових випадків для програмного забезпечення для організації та зберігання спортивних тренувань..... 44

4.4.1. Функціональне тестування програмного забезпечення для організації та зберігання спортивних тренувань..... 45

4.4.2. Тестування інтерфейсу. 47

4.5. Висновки. 47

ВИСНОВКИ..... 49

СПИСОК ЛІТЕРАТУРИ..... 50

ДОДАТОК А. 52

Додаток Г. Інструкція користувача 57

1. Компоненти програмного забезпечення 57

2. Встановлення програмного забезпечення 57

3. Базові функції ПЗ 57

4. Аналіз помилок та можливих проблем 58

Додаток В. Код 59

ВСТУП

Спорт має дивовижні властивості. Він об’єднує людей, знайомить їх між собою, спорт зміцнює здоров’я, характер і навіть розумові здібності людей, які ним займаються, розвиває в них такі навички як швидкість, спритність, реакція, координація, витривалість, терпіння і сила. Спорт робить людей більш стійкими до негативних факторів зовнішнього середовища. Він часто приносить людям масу позитивних емоцій.

На жаль, сьогодні як ніколи раніше люди можуть забути про фізичну активність і жити майже не рухаючись. Навіть ті хто займаються спортом роблять це не оптимально і не зберігають свої результатів. Спортивна індустрія переповнена непорозумінням та не правдивою інформацією. Час є одним з найдорожчих людських ресурсів, тому ніхто не хоче витратити сотні годин в спортивному залі так і не досягнувши хоч якогось вагомого результату результату, але чомусь багато людей так роблять.

На ринку є лише декілька продуктів, які намагаються вирішити цю проблему, але більшість з них є не універсальними і покривають потреби лише певної частини можливих користувачів. Метою даної курсової роботи є створення додатку, який збільшить ефективність атлета, спрощенням пошуку оптимальних спортивних програм в залежності від цілі та максимально спростить збереження та моніторинг результатів. Продукт буде мати максимально простий дизайн та буде доступний як вебсайт за допомогою JavaScript бібліотеки React на клієнтській частині та ASP.Net на серверній частині.

РОЗДІЛ 1. АНАЛІЗ ЗАСТОСУВАННЯ СУЧАСНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У СПОРТИВНІЙ ІНДУСТРІЇ

1.1. Тренування

Тренування - це навчання, вправи для вдосконалення навиків, майстерності; розвиток, удосконалення шляхом систематичного виконання відповідних вправ [1]. Тепер розуміючи значення слова, ми можемо усвідомити наскільки часто люди використовують його хибно. Якщо ціллю активності не є покращення її не можна назвати тренуванням.

Звідси фізичні тренування - активність спрямована на розвиток або покращення можливостей тіла. Є багато видів таких активностей. Розрізняють:

- Аеробне — спрямоване на поліпшення витривалості.
- Анаеробне — спрямоване на розвиток сили.
- Інтервальне — покликане поєднати найкраще від анаеробних та аеробних тренувань
- Гіпоксичне — призначене для зменшення періоду акліматизації.

Кожен вид також має градацію по рівню. Очевидно, що атлет буде мати різні плани занять на різних етапах своєї кар’єри. Це означає, що одна людина за своє життя може спробувати багато різних видів спортивних занять. Багато досягнень в одному виду спорті веде до покраще в іншому. Таким чином людина, яка бігла п'ять кілометрові забіги, буде швидше їздити на велосипеді або навіть присідати більше ніж людина без попереднього досвіді або довгої перерви.

Зараз як ніколи раніше цінується час, таким чином більшість людей хочуть досягти своїх цілей, якомога швидше не втрачаючи при цьому досягнень, які вони здобули. Така потреба досить сильно відчувається у індустрії, через що харчові добавки майже на піку своєї популярності. Але, нажаль, зараз є ще багато простору для оптимізації і збільшення свідомості спортсменів про ефективні методи занять.

1.2. Аналіз цільової аудиторії.

Прогрес у спорті як і, майже, у будь-чому іншому йде не по прямій(рис. 1.1)[2]. Початківець стає сильнішим швидко, він буде ставати кращим навіть при не оптимальних умовах. Що означає, що йому абсолютно не потрібно зберігати свої

та аналізувати свої результати. Проблеми починаються на середньому рівні. Коли це стається більшість людей або закидають тренування або шукають кращі й ефективніші методи занять. Таким чином нашою цільовою аудиторією будуть люди, які або вже в пошуках.

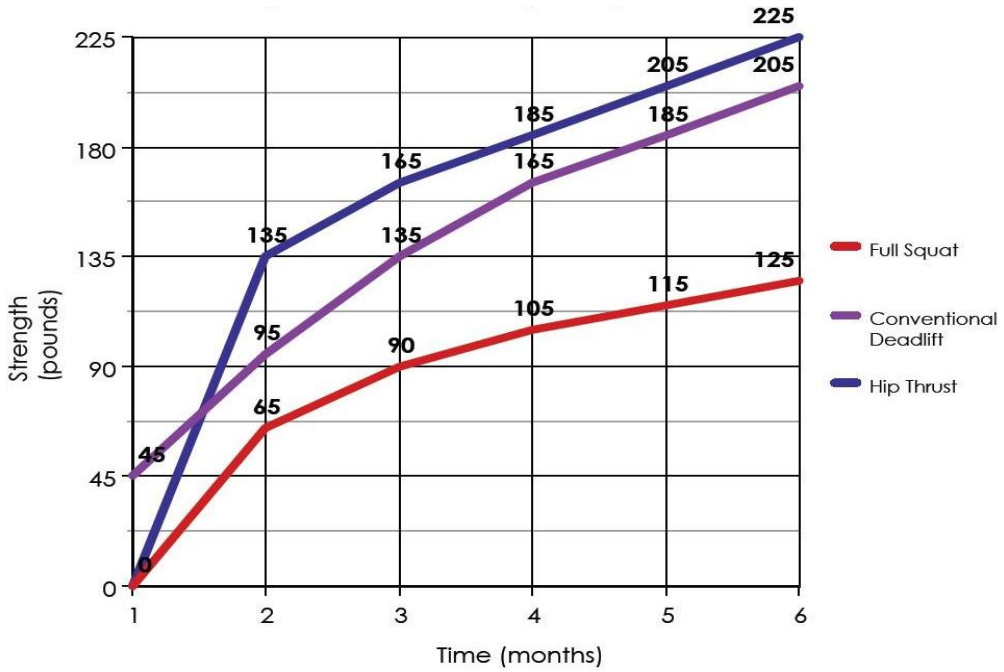

Рис. 1.1. Приклад не лінійної прогресії атлета в перші місяці тренувань

1.2.1. Person 1

Рис. 1.2. Точик Петро Іванович

Вік: 20 рік

Стать: чоловік

Місто: Львів

Сімейний стан: неодружений

Фінансовий дохід: середній

Захоплення і інтереси:

- Спілкування з цікавими людьми
- Відеоігри
- Біг
- Настільні ігри

Передісторія: Петро, завжди хотів стати сильним і більшим, але йому так і не поталанило знайти хорошого тренера або спортивну програму для цього.

Цілі: стати більшим і сильнішим

Мотивація: реалізувати свій фізичний потенціал.

Бар'єри: складність знаходження хорошої спортивної програми в індустрії де дуже багато брехні

1.2.2. Person 2

Рис. 1.3. Левченко Ігор Іванович

Вік: 33 рік

Стать: чоловік

Місто: Львів

Сімейний стан: не одружений

Фінансовий дохід: високий

Захоплення і інтереси:

- Історія
- Бойові мистецтва
- Культуризм
- Рибалка
- Політика

Передісторія: Ігор завжди користувався зошитом для запису своїх тренувань. Нажаль, він його загубив. На щастя, він має хорошу пам'ять, але йому все-рівно доведеться зберігати свої тренування.

Цілі: покращити безпеку логів тренувань.

Мотивація: перемога на містер олімпіаді.

Бар'єри: відсутність додатків, які б найкраще підходили під його стиль тренувань

1.3. SWOT аналіз

SWOT-аналіз (або SWOT-матриця) - це стратегія стратегічного планування, яка використовується для того, щоб допомогти людині чи організації визначити сильні, слабкі сторони, можливості та загрози, пов'язані з бізнес-конкуренцією чи плануванням проекту. На таблиці 1.1 зображені результати аналізу.

Після проведення SWOT-аналізу зрозуміло, що основною перевагою нашого сервісу є універсальність. Будь-який спортсмен зможе додати свої вправи та спортивну програму у систему. Також він легко зможе змінити свій план за необхідності, що до речі може ставитися досить часто, так як при досягненні певного рівня старі методи можуть перестати бути ефективними. Нажаль, ми також

виявили, що програми конкурентів є досить хорошими і деякі з них мають майже весь функціонал, хоч більшість з них орієнтовані на певну категорію спортсменів, наприклад є окремі додатки для бігунів, паверліфтерів(кожна програма має окремий додаток) і культуристів.

Таблиця 1.1 Результати SWOT-аналізу

Strength	Weaknesses
● Зміна існуючих спортивних програм під себе ● Розширена класифікація вправ ● Можливість експортувати логи тренувань в електронну таблицю ● Додавання власних вправ	● Відсутність додатку для годинників ● Відсутність великої кількості тренувань на початку
Opportunities	**Threats**
● Створення додатку для годинників. ● Синхронізація з різними спортивними додатками та обладнанням. ● Залучення професійних спортсменів на платформу. ● Збільшення розуміння важливості запису та аналізу результатів спортивних тренувань. ● Використання штучного інтелекту для підбору робочого навантаження та програми спортсмену.	● Зменшення інтересу в спорті, через пандемію. ● Висока конкуренція з боку вже існуючих додатків.

1.4. Продукти-аналоги

Світовий ринок додатків такого типу є досить широким але більшість існуючих рішень мають певний напрям і орієнтовані на певну групу користувачів.

Проведений аналіз дає змогу виокремити наступні продукти:

● JEFIT Workout Planner - додаток, який доступний на Android, IOS і WEB. Має функцію рутинного планування, за допомогою якої ви можете побудувати свої тренувальні процедури на тиждень і далі. Існують також заздалегідь побудовані процедури підйому, якщо вам не хочеться займатися своїми руками. У сфері відстеження ваших тренувань Jefit пропонує функцію відстеження,

яка дозволяє реєструвати вправи, повторення, підходи та вагу, а також таймер тренувань, який триматиме вас на шляху, відлічуючи до наступного сету. Основним недоліком додатку є його не захищеність, він неодноразово розкривав дані своїх користувачів. Дизайн додатку зображений на рис. 1.4.

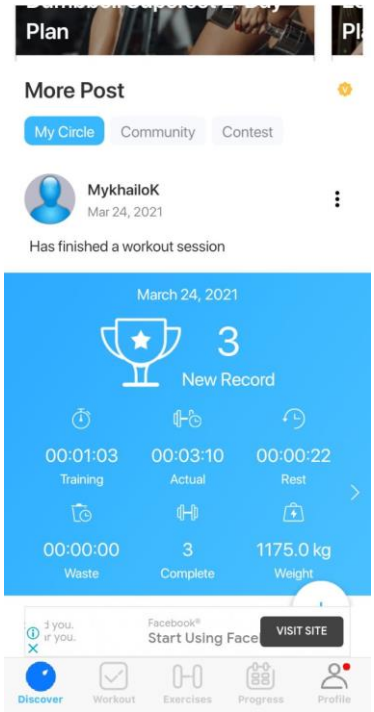

Рис. 1.4. Дизайн JEFIT

- Simple Workout Log - ідеально підходить для атлета, який хоче витратити більше часу на підйом і менше часу, возиться з додатком. Ця програма має саме ті функції, на які ви могли б очікувати: її мінімалістичний підхід дозволяє вводити вправу, вагу, яку ви використовували, і скільки повторень та сетів. Ви можете організувати свої вправи, додаючи їх до категорій або створюючи процедури. Рутинна функція дуже корисна, якщо ви часто виконуєте одні і ті ж тренування, наприклад, базовий сеанс пауерліфтингу. Дизайн додатку зображений на рис. 1.5. Додаток дуже простий і точно не підходить початківцям.

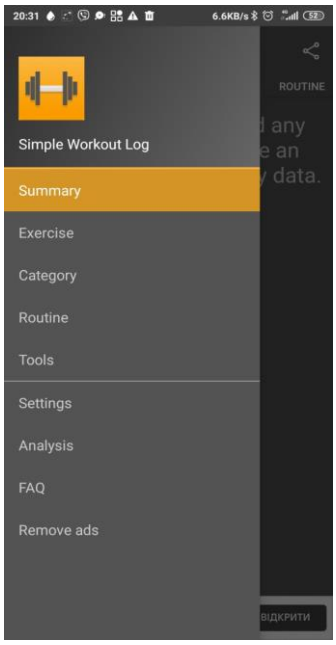

Рис. 1.5. Дизайн Simple Workout Log

Gymaholic Training - ще один трекер тренувань, але він не дозволяє додавати власні не перераховані вправи, що сильно знижує гнучкість цього додатку. Дизайн додатку зображений на рис. 1.6.

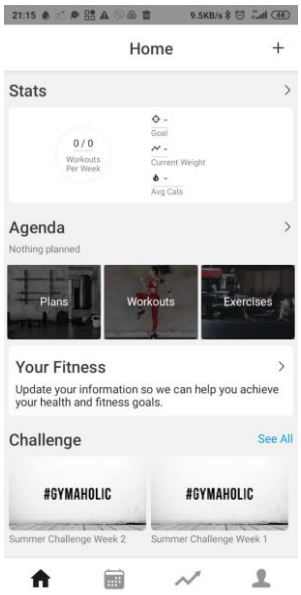

Рис. 1.6. Дизайн Simple Workout Log

1.5. Висновки

Після проведення аналізу предметної області ми зрозуміли, що серед є досить велика проблема у обізнаності середнього спортсмена. Більшість людей навіть не знають, що означає слово тренування і, що постійний, систематичний і запланований прогрес є його основною частиною.

Також навіть спортсмени, які знають, часто не планують свої тренування через відсутність хорошого рішення для цього. Більшість додатків конкурентів орієнтуються на окремі програми або є недостатньо зручними.

РОЗДІЛ 2. ПОСТАНОВКА ЗАДАЧІ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДЛЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОРГАНІЗАЦІЇ ТА ЗБЕРІГАННЯ СПОРТИВНИХ ТРЕНУВАНЬ

2.1. Постановка задачі для програмного забезпечення для організації та зберігання спортивних тренувань

Реалізувати програмне забезпечення, за допомогою якого користувач зможе шукати спортивні програми для досягнення своїх цілей, адаптувати їх під себе, а також зберігати результати своїх тренувань та ділитися ними з іншими. Для інтерфейсу користувача буде використаний React.JS, а для серверної частини Net 6. Тобто це повинна бути веб-система на основі вище згаданих технологій.

2.2. Специфікація вимог до програмного забезпечення для організації та зберігання спортивних тренувань.

2.2.1. Вступ

2.2.1.1. Призначення та мета.

Метою кваліфікаційної роботи є програмне забезпечення за допомогою якого користувач зможе шукати спортивні програми для досягнення своїх цілей, адаптувати їх під себе, а також зберігати результати своїх тренувань та ділитися ними з іншими.

Програма дає змогу створювати та вносити власні вправи та плани. Додаток спрощує процес збереження та поширення власних результатів.

2.2.2. Загальний опис

2.2.2.2. Характеристики продукту

Додаток буде простим трекером тренувань, але зручним як для початківців так і для прокачаних атлетів. Веб система буде мати велику бібліотеку тренувальних програм, яку будуть писати та вдосконалювати самі користувачі. Також програмна система буде спрощувати зберігання результатів за рахунок шаблонів занять та зберігання списків вправ для побудови тренувальних планів, зручного та інтуїтивного інтерфейсу користувача та зручних допоміжних функцій системи.

Клієнтська частина має наступні функції:

- Перегляд, фільтрація за властивостями списку вправ
- Перегляд списку планів для тренувань
- Перегляд детальної інформації про тренування
- Перегляд детальної інформації про план тренування
- Перегляд рекомендованих подібних тренувань на вікні детальної інформації про тренування
- Створення нових вправ
- Створення нових планів для тренування, як список необхідних вплрав
- Створення розширених тренувальних планів з можливістю додавати вправи до дня тижні, тобто робити певний графік занять із встановленою тривалістю занять та кількістю підходів
- Підтримка локалізації інтерфейсу
- Реєстрація та створення власного профіля користувача
- Перегляд статистики

Також повинна бути серверна частина проекту, яка відповідає за бізнес-логіку проекту та надає певні послуги клієнтській частині. Вона надає наступні функції:

- Доступ до бази даних
- Цілісність даних
- Збереження, модифікація даних про користувачів, вплави та тренувальні плани в базі даних

Іще як частина серверної частини є база даних. Вона надає наступні функції:

- Сховище даних

2.2.3. Класи користувачів та їхні характеристики

Користувач системи повинен мати можливість знаходити спортивні програми, які можуть задовольнити його цілі. Для цього йому буде доступний розширений пошук який включатиме в себе такі характеристики як фокус, середня

оцінка та загальна кількість занять. Додаток містить два види користувачів: атлета та тренера. Тренер буде мати всі можливості атлета.

Класи користувачів:

Атлет – має можливість переглядати та фільтрувати вправи та трейнінг плани. Після авторизації користувач має можливість створювати нові трейнінг плани для себе. Цьому типу користувачів доступні два типи створюваних трейнінг планів (простий список вправ та розширений з можливістю створювати графік тренувань на кожен день тижня та встановлювати тривалість та кількість підходів для кожного тренування). Також у користувача є власний особистий кабінет, де він може переглянути інформацію про себе. Також можна переглянути статистику що до тренувань.

Тренер – Має можливість редагувати вправи та трейнінг плани. Також у тренера, як у адміністратора доступна можливість редагувати список користувачів.

2.2.4. Середовище функціонування

Розроблювана програмна система складається з двох частин і відповідно для кожної з частин є свої певні вимоги до технічного та програмного забезпечення.

Вимоги до технічної частини:

- Стабільне інтернет з’єднання зі швидкістю від 500 Кбіт/с або краще
- Будь-який веб переглядач
- Для серверної частини програмного забезпечення необхідно:
- Процесор Intel Core i3-2100 з кешем не менше аніж 4Мб або аналогічні рішення від AMD або краще
- Оперативна пам’ять в об’ємі 16 ГБ або більше
- HDD диск на 2 ТБ або більше
- Інтернет з’єднання зі швидкістю 1 Гбіт/с або швидше

З програмного забезпечення повинно бути встановлено:

- Операційна система Windows 11 Pro
- Microsoft .Net 6
- Microsoft SQL Server 2019

2.2.5. Характеристики системи

Детальний опис класів користувачів та випадками використання системи можна ознайомитися на діаграмі випадків використання(Див. Додаток Б.).

2.3.5.1. Вхід та реєстрація

Опис: Система забезпечує процес автентифікації та ідентифікації користувачів.

Пріоритет: Високий.

Послідовність дій/відгук:

У випадку, якщо користувач ще не зареєстрований в системі:

Натиснувши кнопку "Увійти", користувач активує модальне вікно для реєстрації. Після цього користувач вводить особисті дані (логін, пароль, електронну адресу та номер телефону). Потім натискає кнопку "Зареєструватися", як результат користувач успішно авторизується в системі.

У випадку, якщо користувач вже має обліковий запис:

Після натискання кнопки "Увійти", користувач переходить до модального вікна для входу. Він вказує свій логін та пароль у відповідні поля. Після натискання кнопки "Увійти", користувач успішно авторизується в системі.

Функціональні вимоги:

REQ-1: Перевірка правильності введених даних у полях автентифікації та реєстрації;

REQ-2: Відображення відповідних повідомлень у разі помилок під час спроби входу в систему;

REQ-3: Забезпечення доступу до елементів системи залежно від ролі користувача;

REQ-4: Автоматичне приховування введених даних під час входу в систему;

2.3.5.2 Створення нових простих тренувальних планів

Опис: Створення нових простих тренувальних планів на основі вправ, які наявні в системі.

Пріоритет: середній.

Послідовності дія/відгук: Користувач може створити новий тренінг план шляхом натискання відповідної кнопки і вибору опції "Простий трейнінг план". Він вводить необхідні дані у відповідні поля та вибирає вправи зі списку для додавання до свого тренінг плану. Після заповнення всієї необхідної інформації, користувач завершує процес створення нового тренінг плану, натискаючи відповідну кнопку. Якщо створення пройшло успішно, тренінг план з'явиться у списку доступних тренінг планів і буде відображатися в особистому кабінеті користувача. У разі невдачі відобразиться повідомлення про помилку.

Функціональні вимоги

REQ-1: Перевірка правильності введених даних у формі створення нового тренінг плану;

REQ-2: Відображення відповідного повідомлення у разі невдачі при створенні нового тренінг плану;

REQ-3: Можливість вказувати дату початку та дату завершення тренінг плану;

REQ-4: Після успішного створення тренінг плану, можливість перегляду його в особистому кабінеті;

REQ-5: Після успішного створення тренінг плану, можливість перегляду його у списку всіх доступних тренінг планів на сайті;

2.3.5.3 Створення нових складних тижневих тренувальних планів

Опис: Створення нових складних тренувальних планів на основі вправ, які наявні в системі з можливістю створювати набір вправ на кожен день.

Пріоритет: середній.

Послідовності дія/відгук: Користувач може створити новий тренінг план шляхом натискання відповідної кнопки і вибору опції "тижневий план". Він вводить необхідні дані у відповідні поля та додає вправи на у кожен день зі списку для додавання до свого тренінг плану. Також Користувач може додати тривалість, кількість підходів та час початку для кожної прави. Після заповнення всієї необхідної інформації, користувач завершує процес створення нового тренінг плану, натискаючи відповідну кнопку. Якщо створення пройшло успішно, тренінг

план з'явиться у списку доступних тренінг планів і буде відображатися в особистому кабінеті користувача. У разі невдачі відобразиться повідомлення про помилку.

Функціональні вимоги

REQ-1: Перевірка правильності введених даних у формі створення нового тренінг плану;

REQ-2: Відображення відповідного повідомлення у разі невдачі при створенні нового тренінг плану;

REQ-3: Можливість вказувати дату початку та дату завершення тренінг плану;

REQ-4: Можливість додавати вправи до днів тижнів;

REQ-5: Можливість задавати кількість підходів для вправ;

REQ-6: Після успішного створення тренінг плану, можливість перегляду його в особистому кабінеті;

REQ-7: Після успішного створення тренінг плану, можливість перегляду його у списку всіх доступних тренінг планів на сайті;

2.3.5.4 Перегляд та фільтрація тренувальних планів

Опис: Користувач має можливість переглядати список тренувальних планів з можливістю застосування фільтрів та сортування.

Пріоритет: Високий.

Послідовності дія/відгук: Користувач відвідує сторінку, де відображається список доступних тренувальних планів. Він бачить картки з описом кожного тренувального плану, які доступні в системі. Для зручності користувача присутні кнопки, за допомогою яких можна сортувати та фільтрувати список тренувальних планів. При використанні цих кнопок зміни відображаються на екрані, а список оновлюється відповідно до обраних налаштувань.

Функціональні вимоги:

REQ-1: Можливість перегляду списку тренувальних планів без обов'язкового входу в систему;

REQ-2: Можливість сортування списку тренувальних планів відповідно до певного критерію;

REQ-3: Можливість застосування фільтрів до списку тренувальних планів для відображення лише певних типів тренувальних планів або згідно з обраними параметрами;

REQ-4: Валідація даних, які вводяться для сортування та фільтрації списку тренувальних планів, щоб забезпечити правильність введених значень;

REQ-5: В разі некоректних даних, які вводяться для фільтрації або сортування, відображати відповідне повідомлення або сповіщення;

2.3.5.5 Перегляд та фільтрація списку вправ

Опис: Користувач має можливість переглядати список вправ з можливістю застосування фільтрів та сортування.

Пріоритет: Високий.

Послідовності дія/відгук: Користувач відвідує сторінку, де відображається список доступних вправ. Він бачить картки з описом кожної вправи, які доступні в системі. Для зручності користувача присутні кнопки, за допомогою яких можна сортувати та фільтрувати список вправ. При використанні цих кнопок зміни відображаються на екрані, а список оновлюється відповідно до обраних налаштувань.

Функціональні вимоги:

REQ-1: Можливість перегляду списку вправ без обов'язкового входу в систему;

REQ-2: Можливість сортування списку вправ відповідно до певного критерію;

REQ-3: Можливість застосування фільтрів до списку вправ для відображення лише певних типів вправ або згідно з обраними параметрами;

REQ-4: Валідація даних, які вводяться для сортування та фільтрації списку вправ, щоб забезпечити правильність введених значень;

REQ-5: В разі некоректних даних, які вводяться для фільтрації або сортування, відображати відповідне повідомлення або сповіщення;

2.3.5.6 Управління користувачами в системі

Опис: Здійснення дій з обліковими записами користувачів в системі.

Пріоритет: Високий.

Послідовності дія/відгук: Адміністратор увійшов у систему та перейшов на сторінку управління користувачами. На цій сторінці відображається таблиця зі списком користувачів та докладною інформацією про кожного з них. Адміністратор має можливість додавати нових користувачів, видаляти або змінювати дані існуючих користувачів.

Функціональні вимоги:

REQ-1: Можливість перегляду списку користувачів системи без обов'язкового входу в систему;

REQ-2: Можливість вносити зміни в список користувачів, включаючи додавання нових користувачів та зміну існуючих даних;

REQ-3: В разі невдалих спроб змінити список користувачів відображати відповідне повідомлення або сповіщення;

2.3.6 Атрибути якості програмного продукту

- Придатність – забезпечення виконання набору функцій, передбачених на етапі розробці.
- Супроводжуваність
- Надійність
- Зручність використання – важливим пунктом є «правильний» інтерфейс, який буде зрозумілий користувачу, також до цього пункту можна включити документацію користувача, що має бути простою для сприйняття.

2.3.7. Вимоги зовнішніх інтерфейсів

2.3.7.1. Користувацькі інтерфейси

При створенні даного продукту передбачається, що основні особливості інтерфейсу будуть відповідати загальним стандартам оформлення типових програмних продуктів.

2.3.8. Інші нефункціональні вимоги

2.3.8.1. Вимоги продуктивності

Система повинна швидко запускатися. Перехід між сторінками повинен бути швидким без зайвих переходів.

2.3.8.2. Вимоги безпеки

Система повинна бути достатньо захищеною, щоб особисті дані про здоров'я можуть не розголошуватися недоречно або несанкціоновано.

2.4. Висновки

У даному розділі курсової роботи, було установлено мету та ціль розроблюваної системи. Були визначено початкові та кінцеві цілі, а також було вибрано інструменти для реалізації веб-системи.

Було визначено потенційні класи користувачів та їх ролі та доступний функціонал системи для них.

Також була визначена основна бізнес-логіка програмної системи та сформульовані функціональні та нефункціональні вимоги. Серед найважливіших вимог можна виділити ті, що випливають з бізнес-логіки для реалізації головного сценарію використання веб-системи, зокрема, створення та перегляд трейнінг планів.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОРГАНІЗАЦІЇ ТА ЗБЕРІГАННЯ СПОРТИВНИХ ТРЕНУВАНЬ

3.1. Проєктування архітектури та об’єктної моделі програмного забезпечення для організації та зберігання спортивних тренувань.

Під час розробки веб-системи на різних рівнях було використано UML діаграми, як зазначено в [17]. Під час самого проектування були прийняті важливі архітектурні рішення та була визначена структура веб-системи. Для розробки основної бізнес-логіки була створена детальна діаграма, яка містить всі необхідні пояснення.

Проектування відбувалося в інструменті Draw.io [14].

Розроблені UML діаграм (Додаток А):

- прецедентів (use case diagram) (Додаток А.1);
- розгортання (deployment diagram);
- послідовності (sequence diagram) (Додаток А.2);
- діяльності (activity diagram) (Додаток А.3);
- класів (class diagram);
- інтерфейсів (interface diagram);
- архітектурна діаграма (architecture diagram);
- кооперації (collaboration diagram) (Додаток А.4);
- станів (statechart diagram) (Додаток А.5);
- компонентів (component diagram) (Додаток А.6);

3.1.1. Діаграма прецедентів (Додаток А.1).

У діаграмі прецедентів, яка була розроблена, можна виділити три ролі користувачів: атлет (user), інструктор (admin) та неавторизований користувач. Кожна з цих ролей має різні можливості в системі.

Опис функціональності:

- Атлет, який входить до системи, може переглядати головну сторінку, список тренувальних планів та список вправ. Він може сортувати і

фільтрувати ці дані, а також здійснювати пошук конкретної вправи або тренувального плану. Атлет може створювати новий тренувальний план на основі наявних вправ у системі, а також переглядати створені ним плани у списку тренувальних планів. Крім того, атлет має можливість створювати складений тренувальний план, додаваючи вправи на кожен день тижня. В розширеному тренувальному плані атлет може також вказувати кількість підходів та час початку виконання вправи.

- Адміністратор системи має можливості керування обліковими записами користувачів, зокрема додавання, видалення та зміна облікових записів. Він також може керувати вправами, включаючи їх додавання, видалення та модифікацію.

Крім того, всім користувачам доступна інформація про систему, а також список вправ і рекомендовані вправи, які можна переглянути.

3.1.2. Діаграма послідовності (Додаток Б.2).

Діаграма послідовності ілюструє послідовність дій, які користувачі системи виконують для досягнення своїх цілей. Ця діаграма надає детальний опис кожного кроку виконання, утворюючи своєрідний алгоритм.

На створеній діаграмі відображено послідовність дій, які виконує користувач системи. Лінії життя, що представлені на діаграмі, включають користувача, клієнтську частину системи, API та базу даних.

Для повноцінної роботи з системою, спочатку користувач повинен успішно авторизуватися. У випадку, якщо користувач не має облікового запису, йому необхідно створити його, заповнивши відповідні форми під час процедури авторизації.

Після успішної авторизації, користувач має можливість переглянути список оголошень та ознайомитися з деталізованою інформацією про вправи. Крім того, він може створити тренувальний план, використовуючи наявні вправи у системі.

По завершенні роботи з системою, користувач може вийти, завершивши сеанс.

3.1.3. Діаграма діяльності (Додаток Б.3).

Представлено діаграму діяльності, яка відображає процеси, що відбуваються під час взаємодії відвідувача з системою. У даному контексті, ця діаграма є виразним графічним зображенням поведінки користувача. На діаграмі представлені етапи авторизації, перегляду вправ та трейнінг планів, а також створення нових трейнінг планів.

3.1.4. Діаграма класів.

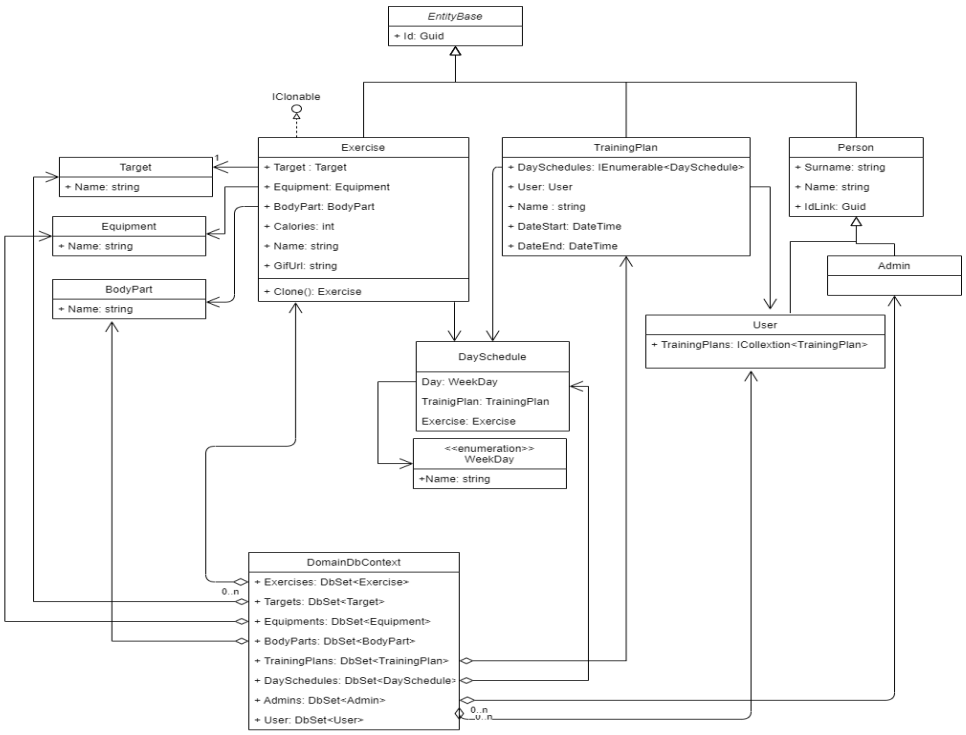

Рис. 3.1. Діаграма класів

На графічній діаграмі класів намалювана ієрархія класів, яка представляє серверну частину системи. Основним класом у цій системі є клас Exercise, що містить в собі повну інформацію про вправи. До того ж, клас Exercise реалізує інтерфейс IClonable, який застосовує паттерн прототипу і дозволяє швидко створювати багато подібних екземплярів вправ, економлячи ресурси. Допоміжними класами до цього відносяться Target, Equipment та BodyPart.

TrainingPlan відповідає за збереження тренувальних планів і є класом, що містить колекції вправ або колекції днів тижня для складних тренувальних планів.

User і Admin є користувачами системи, причому User є класом-нащадком від Admin.

DomainDbContext є контекстом, який об'єднує всі класи для зручного доступу до них з інших рівнів системи. Крім цього, всі класи успадковані від базового класу EntityBase, в якому визначені основні властивості для всіх класів.

Класи User і Admin наслідують клас Person, який визначає основні персональні властивості, такі як прізвище, ім'я, номер телефону та інші.

3.1.5. Діаграма інтерфейсів.

У відповідності до представленої діаграми інтерфейсів, розроблювана веб-система включає репозиторії, що забезпечують доступ до даних моделей у багатьох класах. Усі ці репозиторії успадковуються від загального батьківського класу EntityRepository, який визначає основні операції для забезпечення доступу до даних. Батьківський інтерфейс використовує шаблон типу для конкретизації типів даних у методах, які безпосередньо визначені в дочірніх інтерфейсах.

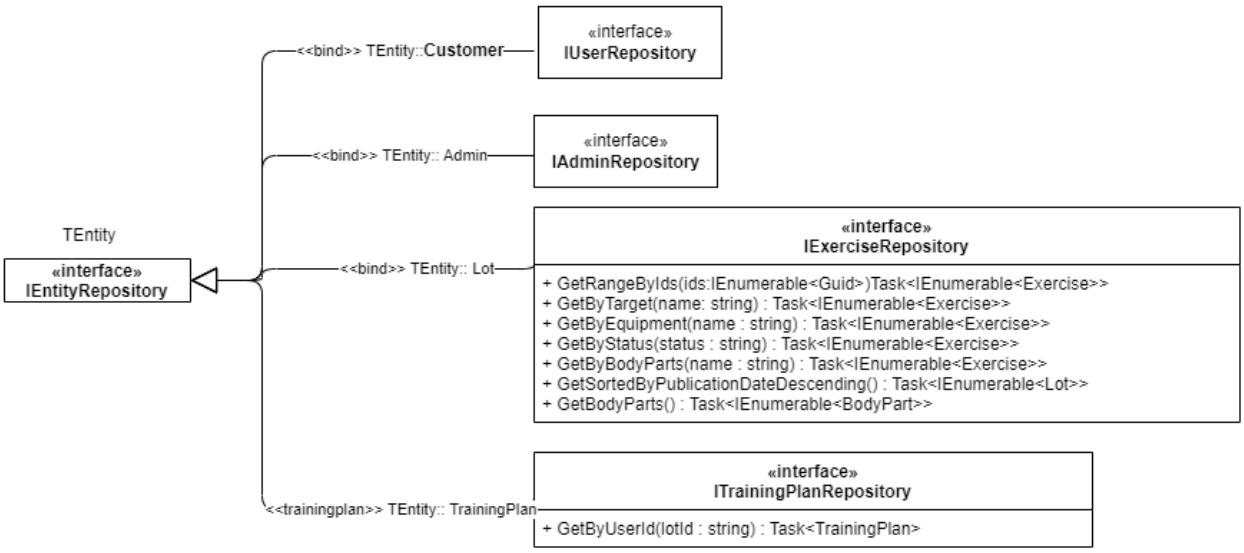

Рис. 3.2. Діаграма інтерфейсів

В контексті Data.Contract визначено інтерфейси, що використовуються для роботи з моделями. Цей рівень визначає контракти для взаємодії з контекстом даних. Реалізація цих інтерфейсів здійснюється на рівні Data.

3.1.6. Архітектурна діаграма.

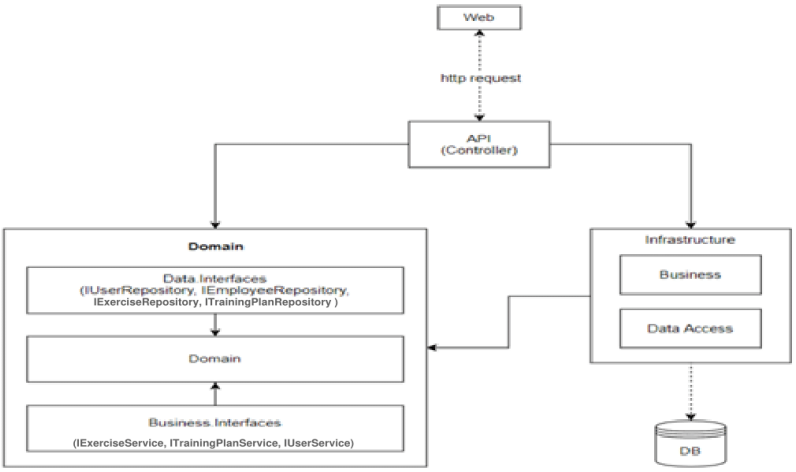

Рис. 3.3. Архітектурна діаграма

У ході розробки серверної частини була використана onion-архітектура з метою досягнення кращої організації компонентів. На нижчому рівні розташовуються моделі, що містять повну інформацію про систему. Для отримання доступу до цих даних використовується рівень доступу до даних, представлений набором репозиторіїв з методами для роботи з моделями. З метою зручного використання, ці репозиторії об'єднані в єдину робочу одиницю (unit of work). Цей архітектурний шаблон сприяє спрощенню роботи з репозиторіями та легкому впровадженню залежностей до них у сервісах. Бізнес-логіка додатку розміщена на рівні бізнесу, де відбуваються логічні операції, що ґрунтуються на вищих рівнях, і використання репозиторіїв з робочої одиниці. На найвищому рівні розташовані контролери, які обробляють запити від клієнтської частини та викликають функції з рівня бізнес-логіки [17].

3.1.7. Діаграма розгортання.

На діаграмі розгортання видно основні компоненти, які будуть використовуватись у системі. Згідно з діаграмою, система має три основних модулі:

1. Клієнтський модуль - відповідає за веб-інтерфейс користувача.
2. Модуль API - обробляє запити від клієнта та забезпечує доступ до бази даних.
3. Модуль бази даних - служить як сховище для даних системи.

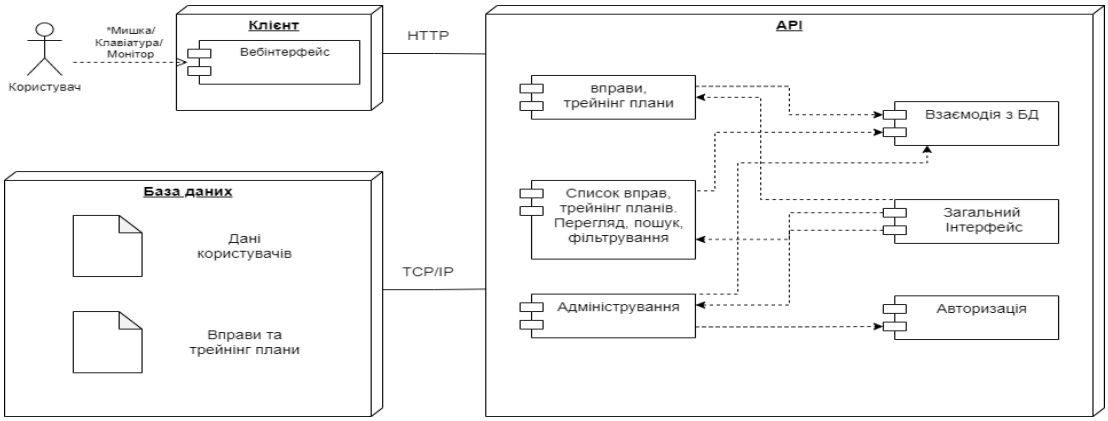

Рис. 3.4. Діаграма розгортання

Крім того, кожен з перерахованих модулів має додаткові підмодулі. В модулі API виявляється наступне:

- Підмодуль вправ, що відповідає за обробку вправ.
- Підмодуль взаємодії з базою даних, який забезпечує взаємодію з базою даних.
- Підмодуль тренувальних планів, який відповідає за обробку тренувальних планів.
- Підмодуль адміністрування, що забезпечує адміністративні функції системи.

Таким чином, розглянуті модулі та їх підмодулі складають основну структуру системи, яка відображена на діаграмі розгортання.

3.2. Проєктування бази даних до програмного забезпечення для організації та зберігання спортивних тренувань

Зважаючи на тематику та поставлені завдання, необхідно мати реляційну базу даних, яка передбачається користувачеві як нормалізовані набори відношень різного рівня. Для збереження даних у системі було обрано Microsoft SQL Server 2022 як реляційну базу даних [19]. Цей тип бази даних ідеально підходить для виконання поставлених завдань відповідно до тематики. Ознайомитися з діаграмою бази даних можна на рисунку Рис. 3.5. Далі буде надано опис бази даних з деталями реалізації кожної таблиці та призначенням кожної колонки у таблиці.

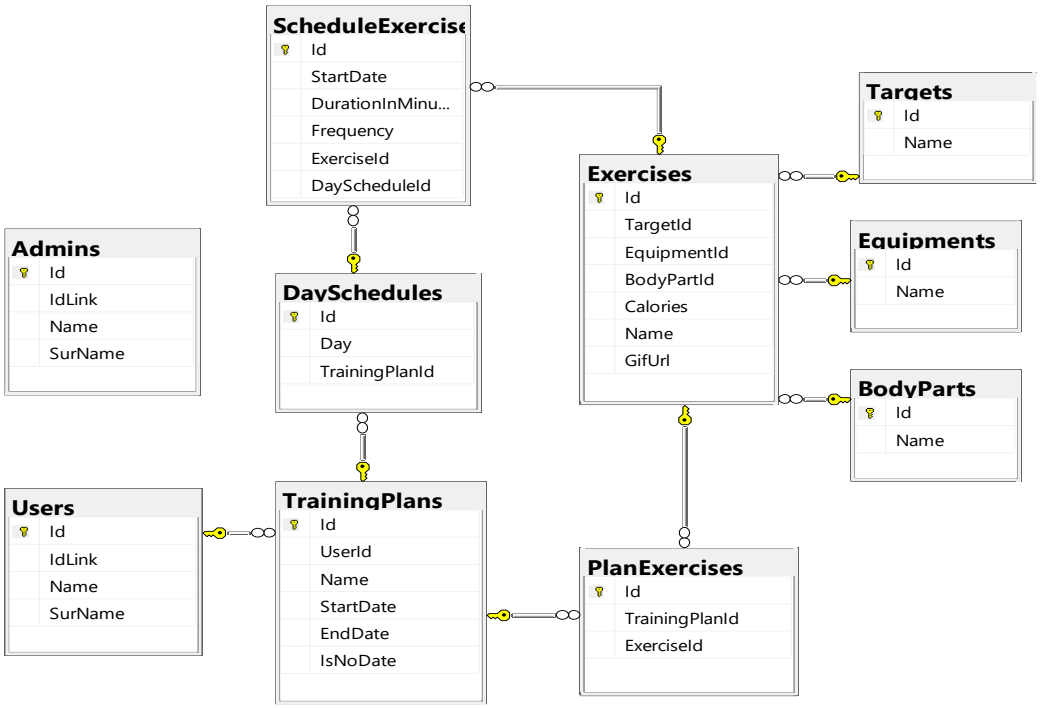

Рис. 3.5. Діаграма бази даних

Таблиця Exercises – відповідає за збереження даних про вправу.

Містить такі поля:

- Id – унікальний ідентифікатор вправи в системі;
- TargetId(FK) – ідентифікатор мети вправи (посилання на таблицю Target);
- EquipmentId(FK) – ідентифікатор спорядження для вправи(посилання на таблицю Equipment);
- BodyPartId(FK) – ідентифікатор частини тіла, тобто яка частина тіла тренувається виконуючи цю вправу(посилання на таблицю BodyPart);
- Calories – кількість калорій які можуть бути спалені за 10хв виконання цієї вправи;
- Name – назва вправи;
- GifUrl – анімація виконання вправи;

Таблиця ScheduleExercises – забезпечує зв’язок багато до багатьох для таблюць Exercises та DaySchedule, а також має додаткові властивості для вправи.

Містить такі поля:

- Id - унікальний ідентифікатор в системі;

- ExerciseId - ідентифікатор вправи (посилання на таблицю Exercises);
- DayScheduleId - ідентифікатор дня до якого додаються вправи (посилання на таблицю DaySchedule);
- StartDate – дата початку заняття;
- DurationInMinutes – тривалість виконання вправи;
- Frequency – кількість підходів або частота.

Таблиця DaySchedule – відповідає за збереження даних про додані заняття на день тижня

Містить такі поля:

- Id – унікальний ідентифікатор в системі;
- TrainingPlanId(FK) – ідентифікатор (посилання на таблицю TrainingPlan);
- Days – день тижня;

Таблиця TrainingPlans – відповідає за збереження даних про створені трейнінг плани. Ця таблиця може слугувати основою для створення простих або складних трейнінг планів.

Містить такі поля:

- Id – унікальний ідентифікатор в системі;
- UserId(FK) – ідентифікатор атлета (посилання на таблицю User);
- Name – назва трейнінг плану;
- StartDate – дата початку трейнінг плану
- EndDate – дата закінчення трейнінг плану
- isNoDate – прапорець, який інформує що трейнінг план немає дати початку чи закінчення;

Таблиця PlanExercises – створює відношення багато до багатьох між таблицями Exercises та TrainingPlan. За допомогою цієї таблиці створюються прості трейнінг плани.

Містить такі поля:

- Id – унікальний ідентифікатор у системі;
- TrainingPlanId(FK) – ідентифікатор трейнінг плану (посилання на таблицю TrainingPlan);
- ExerciseId(FK) – ідентифікатор вправи (посилання на таблицю PlanExercise);

Таблиця Target – відповідає за збереження даних про ціль виконання вправи.

Містить такі поля:

- Id - унікальний ідентифікатор в системі;
- Name – ім'я цілі;

Таблиця Equipment – відповідає за збереження даних про ціль спорядження, яке необхідне для виконання вправи.

Містить такі поля:

- Id - унікальний ідентифікатор в системі;
- Name – ім'я обладнання;

Таблиця BodyPart – відповідає за збереження даних про частину тіла, яку повинна зміцнювати вправа.

Містить такі поля:

- Id - унікальний ідентифікатор в системі;
- Name – ім'я частини тіла;

Таблиця User – відповідає за збереження даних про користувачів системи.

Містить такі поля:

- Id – унікальний ідентифікатор в системі;
- IdLink– ідентифікатор персони (посилання на таблицю бази даних Identity).

- Name – ім’я;
- SurName – прізвище.

Таблиця Admin – відповідає за збереження даних про адміністраторів системи.

Містить такі поля:

- Id – унікальний ідентифікатор в системі;
- IdLink– ідентифікатор персони (посилання на таблицю бази даних Identity).
- Name – ім’я;
- SurName – прізвище;

Таблиця Person – відповідає за збереження даних про персону.

Містить такі поля:

- Id – унікальний ідентифікатор в системі;
- PassWord – пароль;
- PhoneNumber – номер телефону;
- Email – адреса електронної пошти.

Для забезпечення управління користувачами у системі була впроваджена додаткова база даних з назвою WorkOutLogIdentityDbContext. В основі цієї бази даних лежить бібліотека аутентифікації Microsoft Identity [21]. Використання цього підходу значно спрощує взаємодію з користувачами системи, особливо що стосується реалізації аутентифікаційних процесів. Цей крок дозволяє забезпечити зручну та безпечну ідентифікацію користувачів у системі.

3.3. Висновки.

У цьому розділі було здійснено розробку та проектування UML-діаграм для програмного забезпечення, яке спрямоване на організацію та збереження спортивних тренувань. Розроблені діаграми відіграють важливу роль у процесі

написання програмного коду та дозволяють зекономити значну кількість часу. Завдяки цим діаграмам було спроектовано програмну систему на різних рівнях та в різних сценаріях.

В рамках цього проекту було розроблено архітектурну основу системи та здійснено проектування основних рішень, необхідних для реалізації заданої системи відповідно до предметної області. Кожну діаграму було ретельно описано, пояснено основні архітектурні рішення, втілені на них.

Також була розроблена діаграма бази даних для зберігання даних системи. Вона включає основні таблиці та зв'язки між ними. Кожна сутність та зв'язок в таблиці були детально описані, їх призначення було чітко визначено.

Під час проектування UML-діаграм було використано десктопну версію інструменту Draw.io [14], який забезпечує зручний та ефективний процес моделювання діаграм.

РОЗДІЛ 4. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОРГАНІЗАЦІЇ ТА ЗБЕРІГАННЯ СПОРТИВНИХ ТРЕНУВАНЬ

4.1. Реалізація серверної частини програмної системи для організації та зберігання спортивних тренувань.

В рамках цього проекту було створено серверну частину, яка складається з кількох проєктів, об'єднаних у одне рішення. Таке об'єднання проектів відповідає багатошаровій архітектурі серверної частини і дозволяє ефективно розділити код на різні рівні, створюючи чітку і зрозумілу ієрархію.

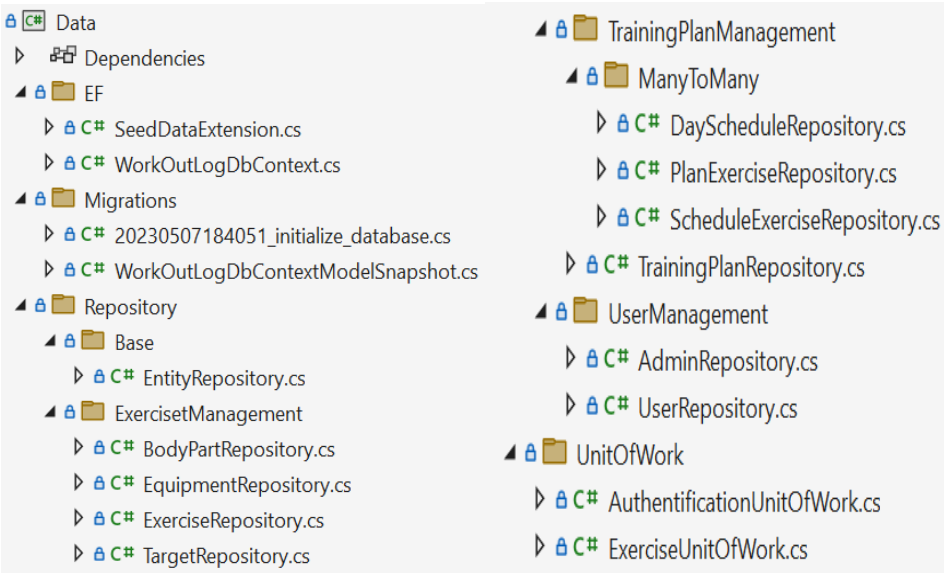

Рис. 4.1. Структура проекту з шаром, який відповідає за оступ до бази даних

У складі серверної частини використовуються дві бази даних. База даних WorkOutLogDbContext використовується для збереження інформації про оголошення та їх управління, тоді як база даних WorkOutLogIdentityDbContext зберігає дані користувачів системи.

На найнижчому рівні розташований проект Domain, який містить опис моделей. Ці моделі використовуються для побудови бази даних, де кожна модель має свої властивості і є нащадком базової моделі, що містить загальні властивості для всіх моделей.

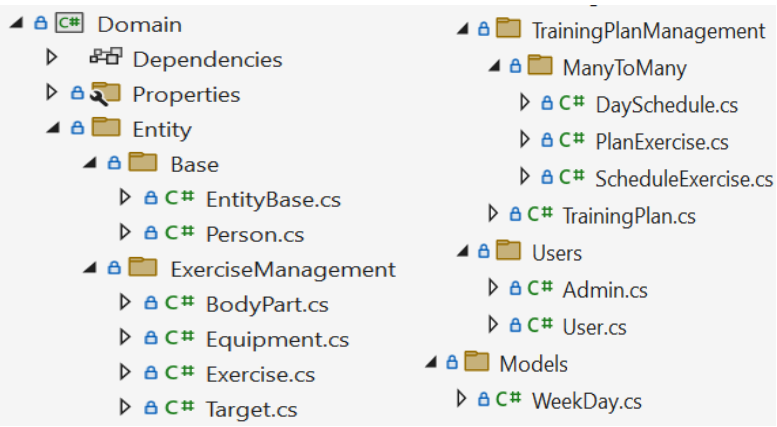

Рис. 4.2. Структура проекту з моделями бази даних

Рівень Data забезпечує доступ до моделей через контекст бази даних (репозиторії) і включає класи та методи для реалізації цього доступу. Також на цьому рівні присутній клас "одиниця роботи", який є архітектурним патерном для збереження посилань на репозиторії і дозволяє використовувати інверсію залежностей у бізнес-логіці додатка. Крім того, на рівні Data знаходиться контекст бази даних для роботи з оголошеннями.

Рівень Data.Contract визначає інтерфейс, який повинен бути реалізований на рівні Data. Це забезпечує виконання "контракту" рівня Data, визначеного на рівні Data.Contract.

Проєкт Data.Identity містить контекст бази даних для роботи з користувачами системи.

Над рівнем Data знаходиться рівень Business - рівень бізнес-логіки. Тут реалізована логіка для виконання операцій CRUD і інших необхідних логічних операцій, необхідних для роботи веб-системи.

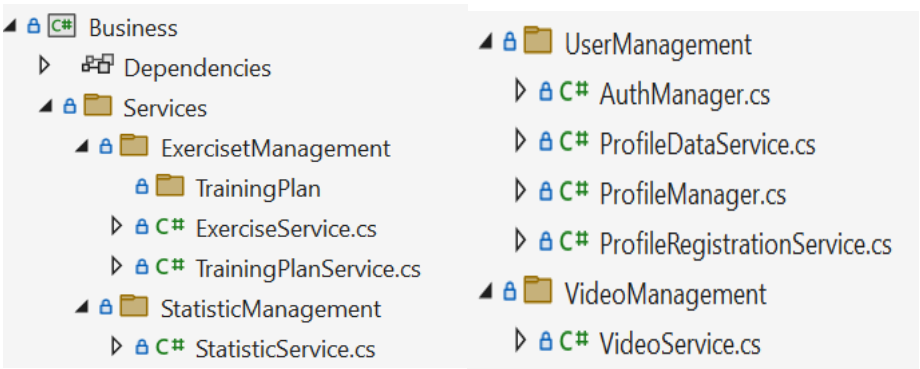

Рис. 4.3. Структура проекту з бізнес логікою

Рівень бізнес-логіки включає інтерфейси рівня Business.Contract, де описані методи, необхідні на рівні бізнес-логіки. Також на цьому рівні визначені моделі представлення, які служать допоміжними моделями для рівня бізнес-логіки. Вони використовуються випадках, коли для функціональності не потрібні зайві властивості з "оригінальних моделей" на рівні Domain.

Найвищим рівнем у рішенні серверної частини є рівень кінцевих точок, відомий як WebAPI. Тут розміщені всі контролери, які отримують запити від клієнтської частини. У контролерах викликаються функції з рівня бізнес-логіки. Крім того, на цьому рівні знаходяться файли конфігурації проекту, включаючи файли мапування між моделями з рівня Domain та моделями, визначеними на рівні бізнес-логіки, файли для ініціалізації інверсії залежностей, файли з налаштуваннями безпеки та інші файлы конфігурації.

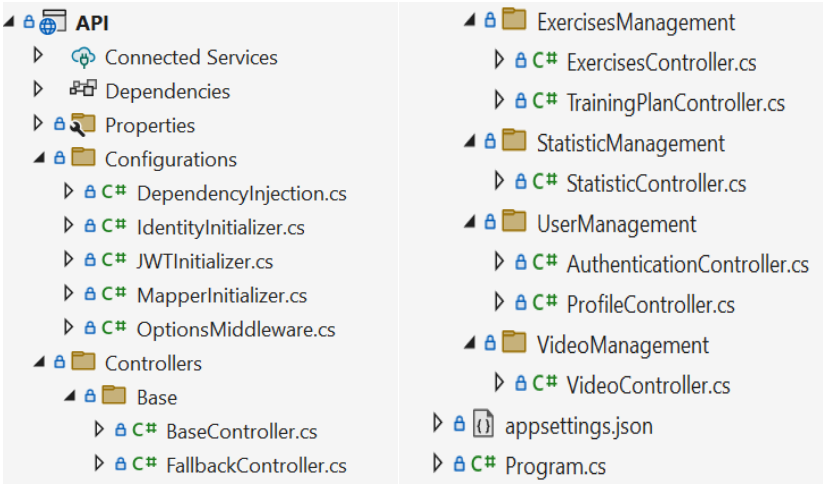

Рис. 4.4. Структура проекту з кінцевими точками

4.2. Реалізація клієнтської частини прорамного забезпечення для органіхзації та зберігання спортивних тренувань

Початковий етап розробки фронтенду включав створення початкової сторінки зі списком вправ і окремої сторінки для створення нових вправ. З метою досягнення цієї мети було необхідно використати відповідні бібліотеки для інтеграції зі сторонніми API та розробити інтуїтивно зрозумілий інтерфейс для зручного користування.

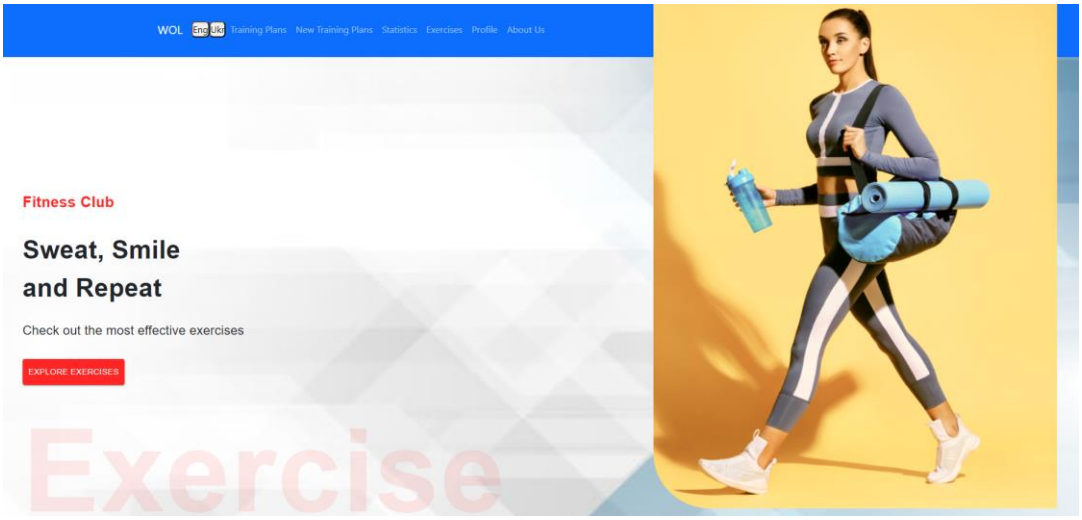

Рис. 4.5. Початкова сторінка клієнтської частини

Була розроблена логіка, що забезпечує взаємодію з серверною частиною шляхом відправки відповідних запитів. За цією метою були створені спеціальні реактивні компоненти, які відповідають вимогам поставлених завдань. Після успішної реалізації початкових сторінок була продовжена розробка аналогічних сторінок.

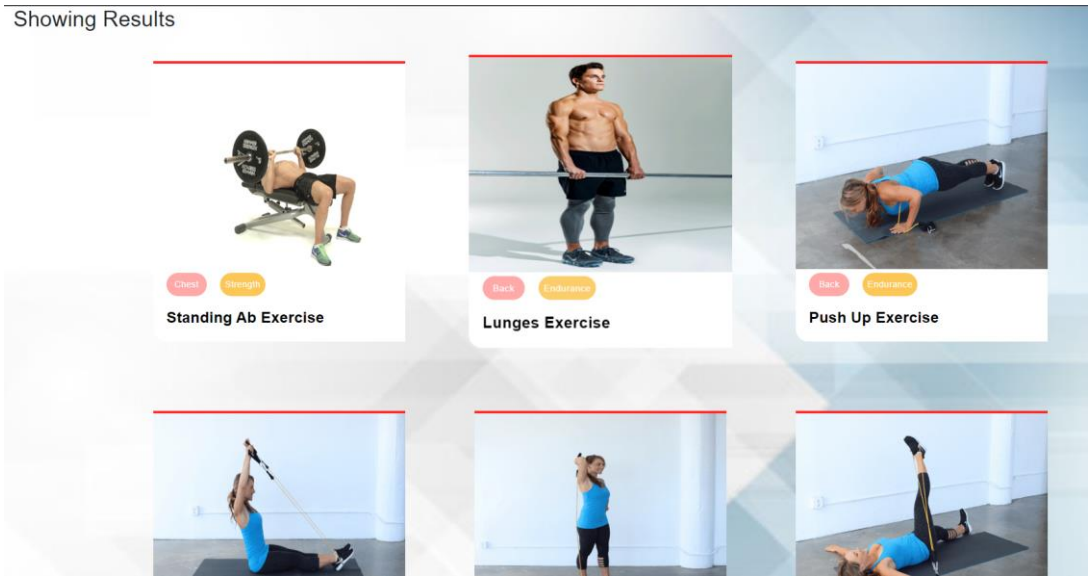

Рис. 4.6. Початкова сторінка клієнтської частини зі списком доступних вправ

Для забезпечення взаємодії з серверними кінцевими точками та відповідними бізнес-логіками були створені сервісні компоненти, які забезпечують виконання різних типів HTTP-запитів. У системі існують дві категорії

користувачів: адміністратори та звичайні користувачі. Для досягнення цього були реалізовані механізми контролю доступу як на клієнтській, так і на серверній частині системи.

4.3. Опис роботи системи

При вступі до клієнтської частини системи, користувача вітає головна сторінка, на якій представлений перелік доступних вправ. На цій сторінці користувач має можливість ознайомитися з вправами, які доступні йому, а також виконати сортування списку за різними критеріями.

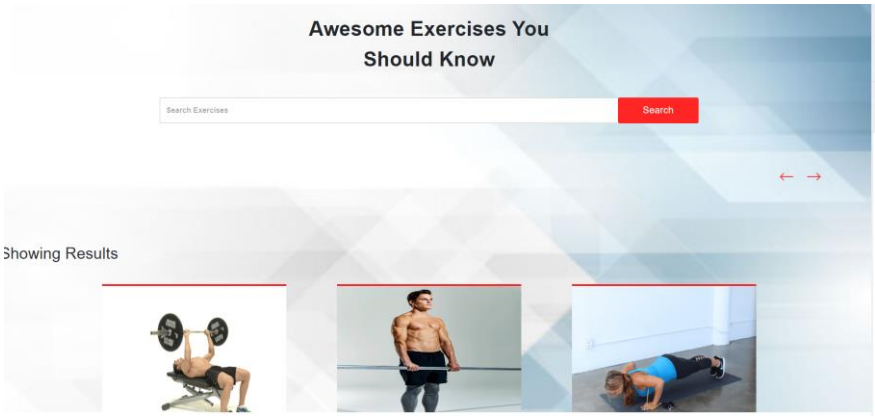

Рис. 4.7. Початкова сторінка клієнтської частини зі списком доступних вправ

При натисненні на будь-яку вправу користувач переходить на сторінку з детальним описом цієї вправи. На цій сторінці користувач може ознайомитися з детальною інформацією про вправу.

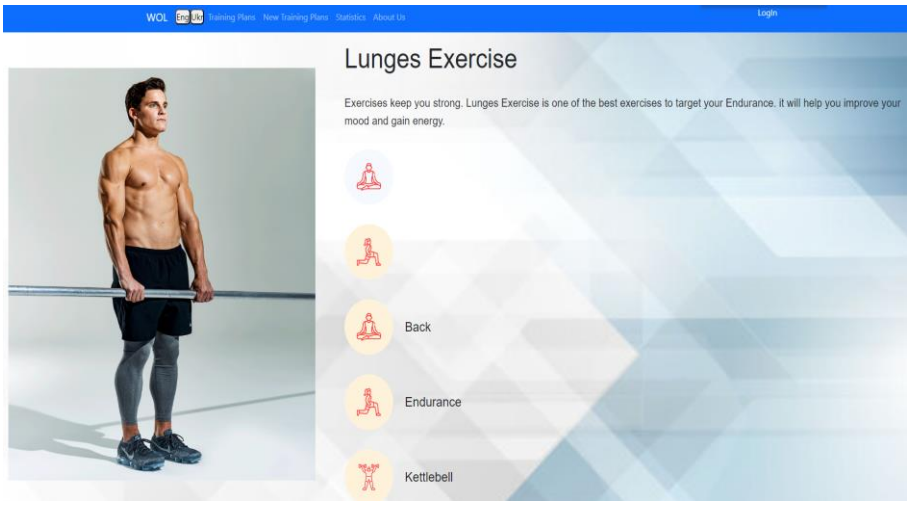

Рис. 4.8. сторінка з детальною інформацією про вибрану вправу

У разі, якщо користувач ще не має авторизованого доступу до системи, він матиме можливість виконати процедуру авторизації. При натисканні кнопки "Увійти" користувачу буде відображено вікно, де він зможе ввести свої облікові дані та увійти до системи.

Під час процедури авторизації будуть вимагатися введення логіну та пароля користувача. Якщо користувач ще не має облікового запису в системі, він матиме можливість зареєструватися через відповідне вікно або звернутися до адміністратора сайту для створення нового облікового запису.

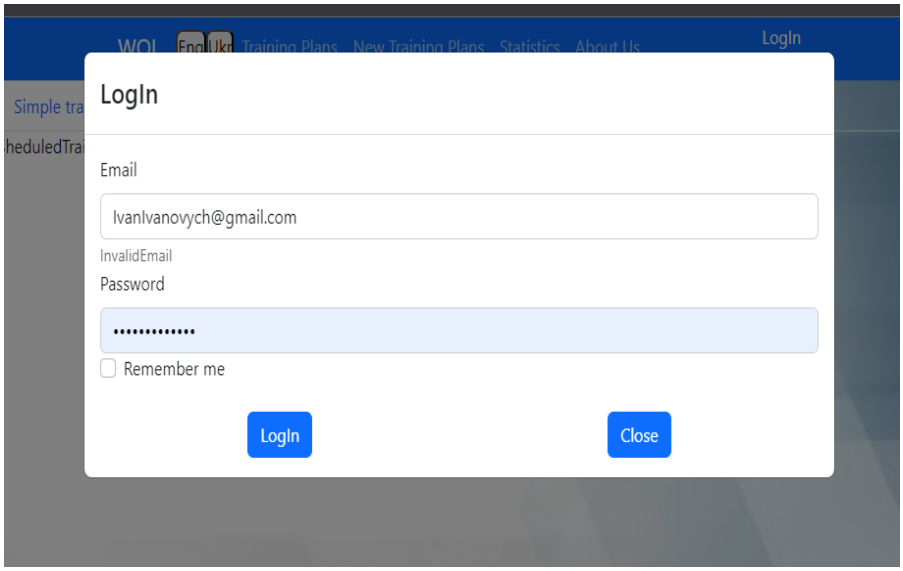

Рис. 4.9. сторінка з детальною інформацією про вибрану вправу

Для авторизованого користувача доступна можливість створювати власні тренувальні плани. Для цього необхідно перейти на сторінку створення нового тренувального плану та вибрати одну з вкладок. Всього доступно три вкладки. На першій доступна можливість створити простий тренуваьлний план. На другій вкладці доступна можливість створити складний тренувальний план. На третій доступна інструкції з описом як можна створити новий тренувальний план.

Для того щоб створити простий тренувальний план необхідно перейти на його сторінку та увести дані у поля та сформувати список вправ. Після додавання всієї необхідної інформацію користувач повинен натисну кнопку “Create training plan”. Створений план відобразиться у списку доступних планів.

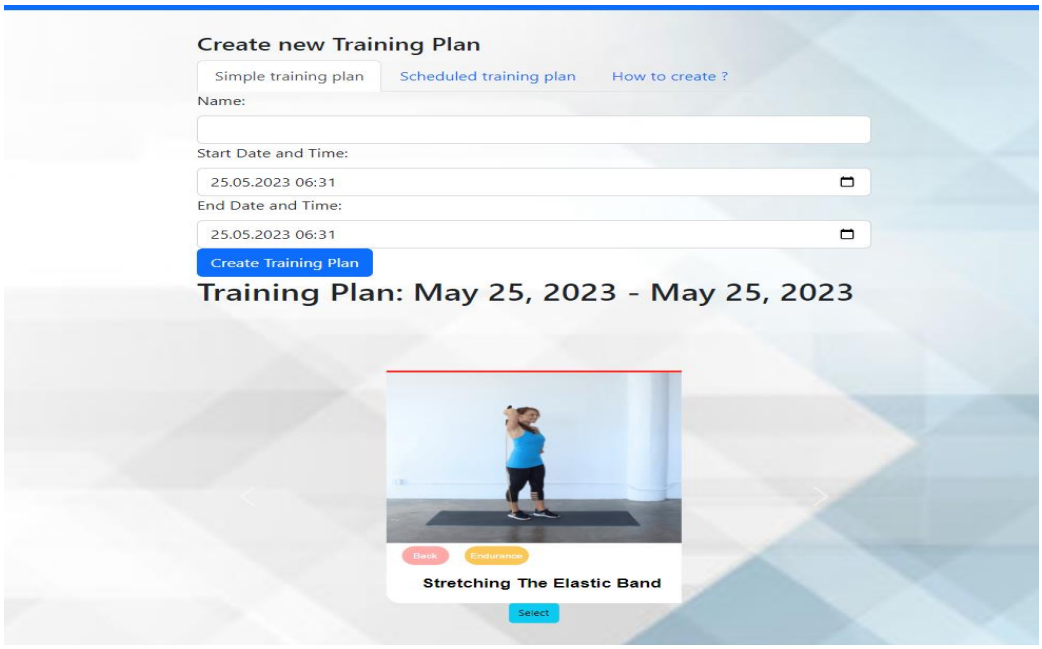

Рис. 4.10. сторінка створення простого тренувального плану

Для того щоб створити складний тренувальний план необхідно перейти на його сторінку, увести дані нового плану та додати до кожного дня тижня його набір вправ із додатковими параметрами.

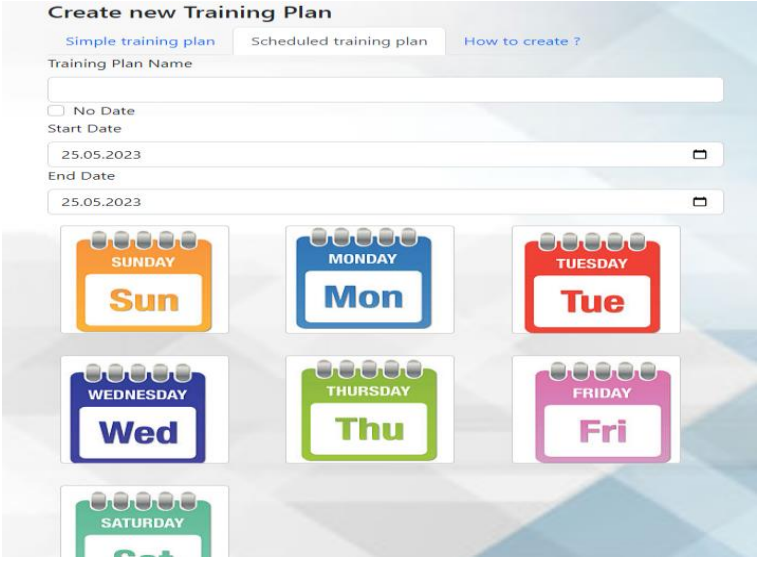

Рис. 4.10. сторінка створення складного тренувального плану

Важливою частиною складного тренувального плану є його можливість створювати тренувальний план на кожен день тижня. Для цього необхідно вибрати день тижня. У відповідь на цю дію відобразиться контексне вікно зі списком доступних вплав та можливістю створити список вправ на день. Також для кожної

вправи можна задати кількість підходів та тривалість виконання вправи та час початку виконня.

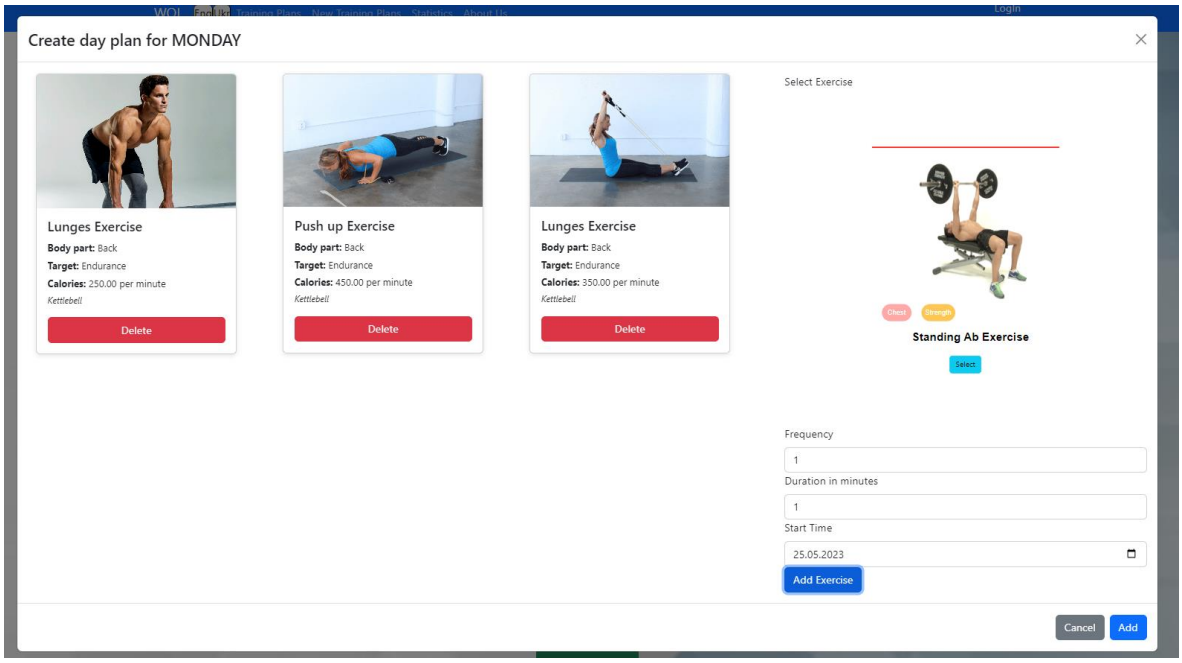

Рис. 4.11. Додавання вправ до дня тижня

Після внесення всіх даних користувач повинен натиснути кнопку “Створити тренувальний план”. По завершенню операції новий тренувальний план буду доступний в системі і інформацію про нього можна переглянути у списку доступних тренувальний планів.

Для того щоб ознайомитися зі списком доступних тренувальних планів користувачу необхідно перейти на сторінку зі списком тренувальних планів. На цій сторінці відображено всі тренувальні плани, які були створені користувачами. Користувач може відфільтрувати список наявних тренувальних планів. Також при натисненні на один з планів можна перейти на сторінку цього плану та ознайомитися з його детальною інформацією.

4.4. Проведення тестових випадків для програмного забезпечення для організації та зберігання спортивних тренувань.

У процесі перевірки та оцінки ефективності веб-системи, призначеної для організації та зберігання спортивних тренувань, використовувалися різні методи тестування. Зокрема, були застосовані функціональні тести, які дозволяють

перевірити, чи працюють окремі функції системи відповідно до встановлених вимог. Для оцінки продуктивності були використані інструменти Visual Studio та Task Manager, що дозволило здійснити аналіз ресурсів та продуктивності системи. Крім того, було проведено ручне тестування користувацького інтерфейсу, де спеціалісти перевіряли зручність та коректність роботи інтерфейсу з точки зору користувача.

4.4.1. Функціональне тестування програмного забезпечення для організації та зберігання спортивних тренувань.

Для перевірки правильності роботи серверної частини розробленої веб-системи було створено набір юніт-тестів. Ці тести охоплювали основний функціонал бізнес-шару серверної частини. Було розроблено кілька сценаріїв використання з різними тестовими даними, що були визначені у (Таблиці 4.1.).

Таблиця 4.1.

Варіанти використання для функціонального тестування

Варіанти використання	Тестові випадки	Тестові дані
Створення нових вправ	1	3
Редагування, видалення, отримання даних про вправу	1	4
Створення нового простого тренувального плану	1	3
Створення складного тренувального плану	1	5
Перегляд детальної інформації про вправу	1	3
Перегляд детальної інформації про тренувальний план	1	3
Разом	6	21

Для створення юніт-тестів, спрямованих на перевірку функціональності бізнес-логіки, був створений окремий проект під назвою Business.Tests. Більшість тестів були розроблені за шаблоном AAA з використанням бібліотеки Moq для перевірки поведінки та стану системи, а також бібліотек Xunit або Nunit.

Структура тестів та їх опис також були враховані. Зважаючи на високий рівень бізнес-логіки, спочатку була проведена ініціалізація нижчих рівнів системи. Згідно з шаблоном, спочатку ініціалізувалися рівень даних (репозиторії та об'єднання репозиторіїв в одиницю роботи), а потім створювався новий екземпляр сервісного класу.

```
[Test]
0 references
public void CreateScheduledTrainingPlan_WhenDaySchedulesIsNull_ShouldReturnNewGuid()
{
    // Arrange
    var createScheduledPlanExercise = new CreateScheduledTrainingPlanDto();
    var userId = Guid.NewGuid();

    var mapperMock = new Mock<IMapper>();
    mapperMock.Setup(m => m.Map<TrainingPlan>(createScheduledPlanExercise))
        .Returns(new TrainingPlan { UserId = userId });

    var userRepositoryMock = new Mock<IUserRepository>();
    userRepositoryMock.Setup(u => u.GetByIdLink(userId))
        .ReturnsAsync(new User(new Guid("86b32f01-322b-43fe-836c-06fefa4c55b4")) { Id = new Guid("9029d27c-eada-45ab-9e9a-897aa05d2d85") });

    var trainingPlanRepositoryMock = new Mock<ITrainingPlanRepository>();
    trainingPlanRepositoryMock.Setup(t => t.AddSync(It.IsAny<TrainingPlan>()))
        .Returns(Guid.NewGuid());

    var dayScheduleRepositoryMock = new Mock<IDayScheduleRepository>();
    var scheduleExerciseRepositoryMock = new Mock<IScheduleExerciseRepository>();

    var exerciseUnitOfWorkMock = new Mock<IExerciseUnitOfWork>();
    exerciseUnitOfWorkMock.SetupGet(uow => uow.UserRepository)
        .Returns(userRepositoryMock.Object);
    exerciseUnitOfWorkMock.SetupGet(uow => uow.TrainingPlanRepository)
        .Returns(trainingPlanRepositoryMock.Object);
    exerciseUnitOfWorkMock.SetupGet(uow => uow.DayScheduleRepository)
        .Returns(dayScheduleRepositoryMock.Object);
    exerciseUnitOfWorkMock.SetupGet(uow => uow.ScheduleExerciseRepository)
        .Returns(scheduleExerciseRepositoryMock.Object);
```

Рис. 4.7. Приклад ініціалізації у тесті

Після ініціалізації нижчих рівнів і рівня бізнес-логіки настала фаза дій. На цьому етапі викликався метод з рівня бізнес-логіки, який перевірявся.

```
public class ExerciseServiceTests
{
    private Mock<IMapper> _mapperMock;
    private Mock<IExerciseUnitOfWork> _exerciseUowMock;
    private Mock<IExerciseRepository> _exerciseRepositoryMock;
    private ExerciseService _exerciseService;

    [SetUp]
    0 references
    public void SetUp()
    {
        // Mock dependencies
        _mapperMock = new Mock<IMapper>();
        _exerciseUowMock = new Mock<IExerciseUnitOfWork>();
        _exerciseRepositoryMock = new Mock<IExerciseRepository>();
        _exerciseService = new ExerciseService(_mapperMock.Object, _exerciseUowMock.Object);
    }

    [Test]
    0 references
    public void Create_WhenExerciseNameExists_ShouldThrowException()
    {
        // Arrange
        var createExercise = new CreateExerciseDto { Name = "Test Exercise" };

        _exerciseRepositoryMock
            .Setup(x => x.IsExist(createExercise.Name))
            .Returns(true);

        _exerciseUowMock
            .Setup(x => x.ExerciseRepository)
            .Returns(_exerciseRepositoryMock.Object);

        // Act & Assert
        Assert.ThrowsAsync<Exception>(() => _exerciseService.Create(createExercise));
    }
```

Рис. 4.8. Приклад виклику тестового метода та перевірка на правильність результатів

Останнім етапом була перевірка правильності результатів після виклику методу з рівня бізнес-логіки. Для цього використовувався відповідний статичний метод з області видимості класу Assert, що дозволяв порівнювати значення або здійснювати інші операції перевірки.

4.4.2. Тестування інтерфейсу.

Було проведено тестування користувацького інтерфейсу, а також протестовано основні та критичні функціональні можливості клієнтської частини веб-системи (див. Таблицю 4.2). Тестування охопило ключові сценарії використання системи користувачем, включаючи процес авторизації, перегляд доступних вправ та тренувальних планів, вибір конкретних вправ або планів, створення складних або простих тренувальних планів.

Таблиця 4.2.

Варіанти використання для тестування інтерфейсу

Варіант використання	Тестові дані
Перевірка екрану профілю	4
Створення нових вправ	5
Перевірка можливості сортування та перегляду списку вправ	2
Перегляд детальної інформації стосовно вправи	3
Створення нового простого тренувального плану	2
Перегляд та редагування профіля користувача	3
Перегляд створених тренувальних планів	2
Перегляд списку створених вправ	3
Створення нового складного тренувального плану	1
Разом	25

4.5. Висновки.

У цьому розділі було виконано розробку веб-системи, спрямованої на тренувальні вправи та тренувальні плани. Було розроблено код для реалізації серверної та клієнтської частин веб-системи. На серверній частині була розроблена

база даних з використанням методології "спочатку код". На основі цього рівня та встановлених залежностей у файлі контексту була згенерована база даних. Були реалізовані архітектурні та проектні рішення, розроблені на попередніх етапах розробки, для обох частин системи. Також було проведено опис роботи програми та її основних компонентів. З метою забезпечення стабільності роботи системи було проведено тестування серверної та клієнтської частин веб-системи. Для серверної частини були написані юніт-тести, а для клієнтської частини було проведено ручне тестування. На даному етапі також була розроблена інструкція користувача (див. Додаток Г).

ВИСНОВКИ

В рамках курсової роботи було успішно розроблено програмне забезпечення, спрямоване на організацію та зберігання спортивних тренувань. Цей процес включав в себе всі етапи, що характеризують життєвий цикл розробки програмного забезпечення.

Спочатку був проведений аналіз предметної області та визначено цілі та важливість реалізації даної системи. Також були вивчені і проаналізовані аналогічні проекти, доступні на ринку.

Після аналізу предметної області були встановлені та сформульовані завдання, а також вибрані інструменти для реалізації проекту. Весь процес був розбитий на два рівні - серверний та клієнтський, і були чітко визначені вимоги до розроблюваної системи.

На наступному етапі - проєктуванні, була розроблена структура системи, її поведінка та взаємодія. Також були розроблені UML діаграми та прийняті ключові архітектурні рішення. Важливою частиною цього етапу було проектування бази даних, необхідної для функціонування системи.

Нарешті, на етапі реалізації було написано програмний код, який відповідав завданням, сформульованим на етапі проєктування. Були реалізовані клієнтська та серверна частини веб-системи. Після цього було проведено широке тестування як на сервері, так і на клієнтській стороні.

В результаті мого дослідження та виконання курсової роботи я отримав значний досвід у розробці складних програмних систем. Цей проект дав мені можливість пройти всі етапи життєвого циклу програмного забезпечення, що надає мені цілісну картину процесу створення програмного забезпечення. Завдяки цьому досвіду я здатний ефективно аналізувати предметну область, поставляти чіткі задачі, проектувати структуру системи та реалізовувати її за допомогою відповідного програмного коду. Весь цей досвід відкриває для мене нові можливості в майбутніх проектах і підвищує мою компетентність в галузі розробки програмного забезпечення.

СПИСОК ЛІТЕРАТУРИ

1. Білодід І. СУМ-11 – Наукова думка. 2008. – Том 10, 246 с. , ліва колонка.
2. Interface. User & Reference Guide for emWin V5.26 –August 21, 2014. – 213 с.
3. Arnold S. Berger. Embedded Systems Design: An Introduction to Processes, Tools, and ‘Techniques. CMP Books © 2002. ISBN: 1-57820-073-3. – 654с.
4. Понцер Г. Наш метаболізм. Калорії, вага та секрети міцного здоров’я / Герман Понцер. – New York: КСД, 2010. – 416 с. – (The New York Times).
5. Stefanov S. React: Up & Running: Building Web Applications / Stoyan Stefanov., 2021. – 222 с. – (Програмування).
6. Price M. J. C# 10 and .NET 6 – Modern Cross-Platform Development: Build apps, websites, and services with ASP.NET Core 6, Blazor, and EF Core 6 using Visual Studio 2022 and Visual Studio Code, 6th Edition 6th ed. Edition / Mark Price., 2023. – 1328 с. – (програмування). – (Для професіоналів; кн. 6).
7. Richter J. CLR via C# / Jeffrey Richter. – Redmond, Washington 98052-6399: Microsoft prePress, 2012. – 813 с. – (A Division of Microsoft Corporation). – (Full Coverage of Multicore Programming; вип. 4).
8. SQL Server Tutorial [Електронний ресурс]– 2022. – Режим доступу до ресурсу: https://www.sqlservertutorial.net/ (2022).
9. React JavaScript-бібліотека для створення користувацьких інтерфейсів [Електронний ресурс] // FaceBookOpenSource. – 2022. – Режим доступу до ресурсу: https://uk.reactjs.org/ (2022). 62
10. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale 1st Edition / Boris Cherny., 2019. – 342 с. – (Programming TypeScript).
11. Draw.io [Електронний ресурс] Режим доступу до ресурсу: https://www.draw.io/ (2022).
12. Figma [Електронний ресурс] Режим доступу до ресурсу: https://www.figma.com/ (2022).
13. VC Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com/ (2022).

14. MS Visual Studio [Електронний ресурс] – Режим доступу до ресурсу: https://visualstudio.microsoft.com/ (2022).
15. Грицюк Ю. Аналіз вимог до програмного забезпечення / Юрій Грицюк. – Львів: Львівська політехніка, 2018. – 456 с.
16. Booch G. The Unified Modeling Language User Guide / G. Booch, J. Rumbaugh, I. Jacobson., 2005. – 407 с. – (Addison Wesley). – (3nd Edition).
17. Software Architecture — The Onion Architecture [Електронний ресурс] // Shivendra Odean. – 2019. – Режим доступу до ресурсу: https://medium.com/@shivendraodean/software-architecture-the-onion-architecture-1b235bec1dec.
18. Левус Є. Вступ до інженерії програмного забезпечення / Є. Левус, Н. Мельник. – Львів: Львівська політехніка, 2018. – 248 с.
19. MS SQL Server [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/ru-ru/sql-server/sql-server-2019 (2022).
20. MS Identity Web authentication library [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/active-directory/develop/microsoft-identity-web(2022)

ДОДАТОК А.

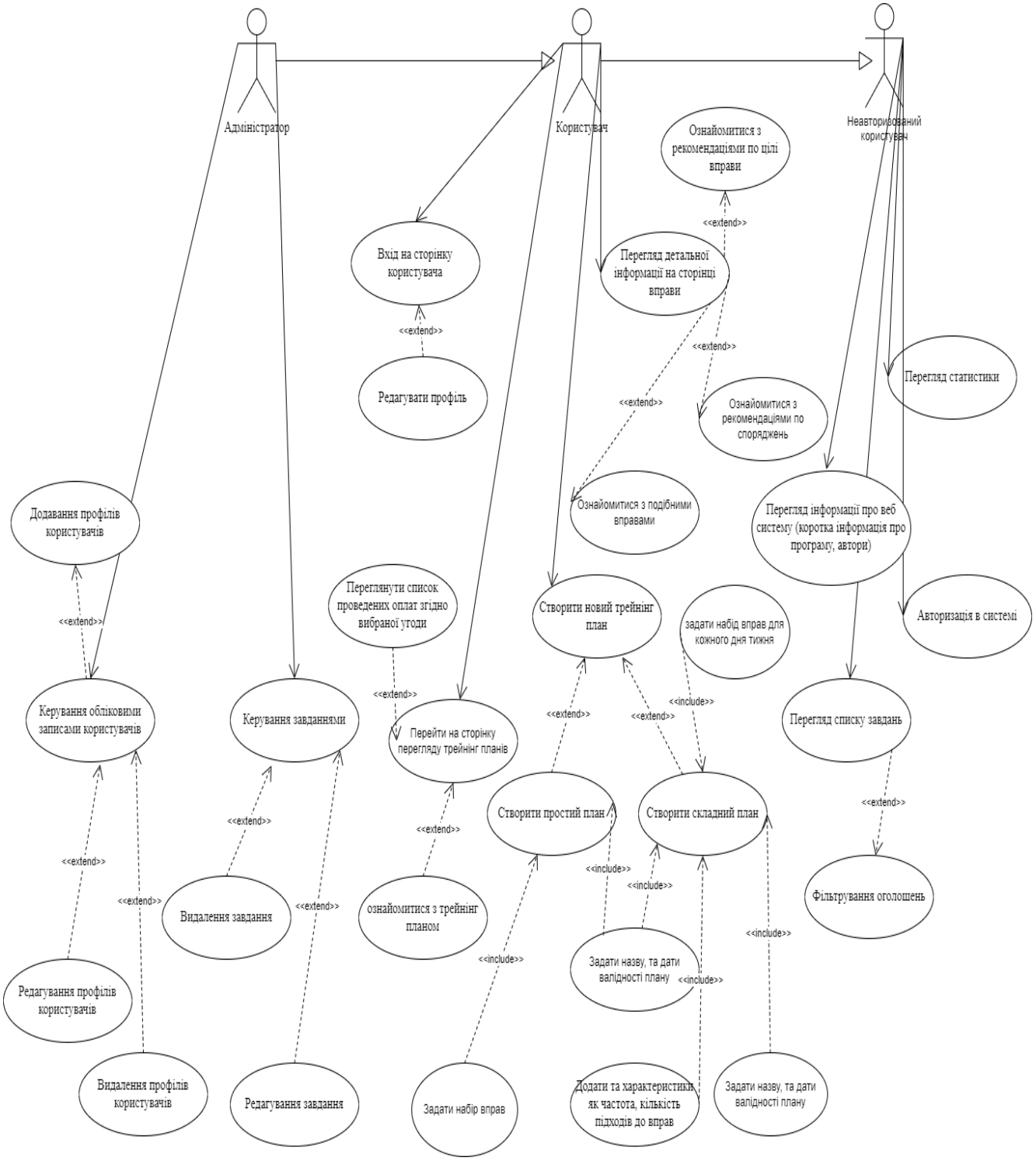

Рис. А.1. Діаграма прецедентів

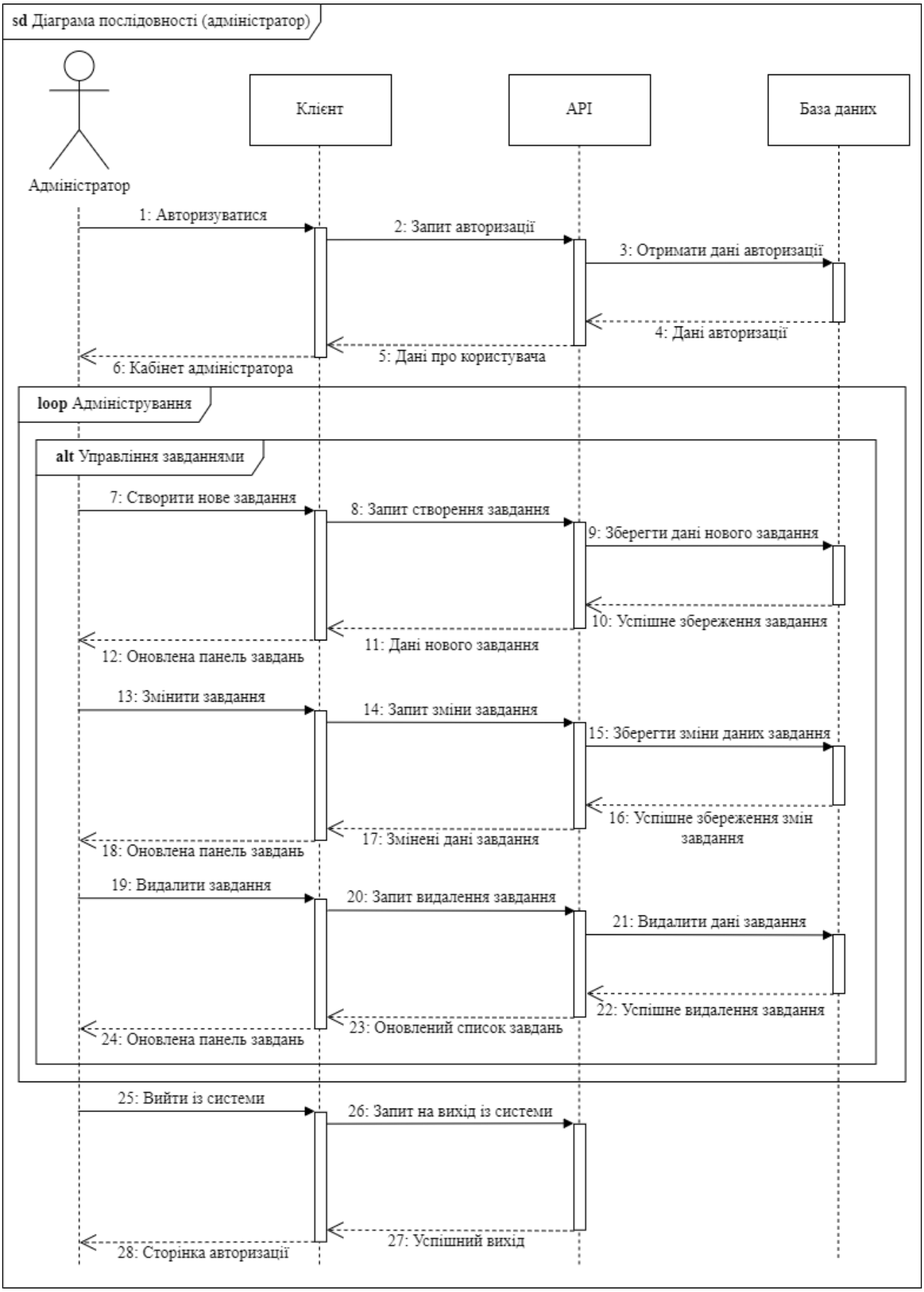

Рис. А.2. Діаграма послідовності

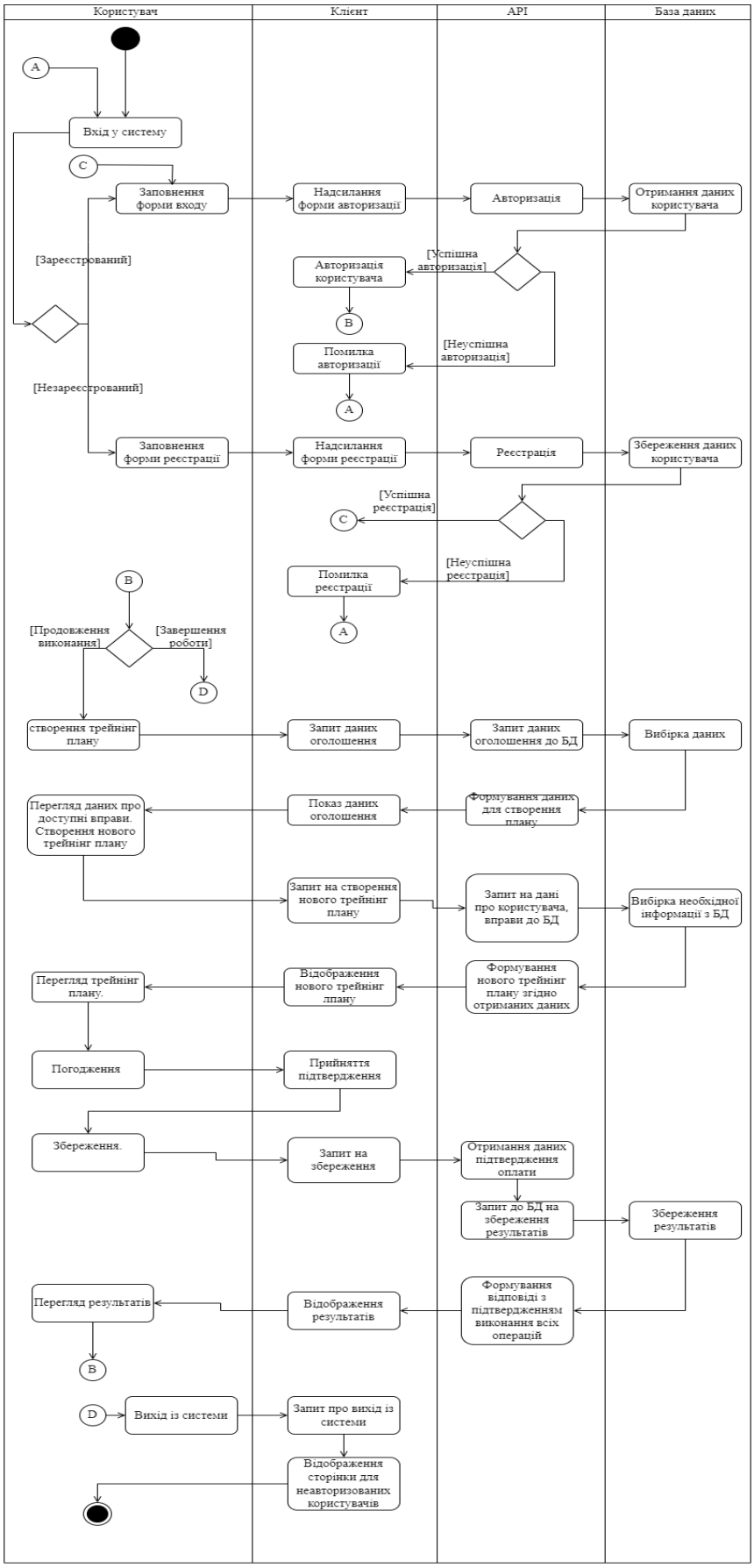

Рис. А.3. Діаграма діяльності

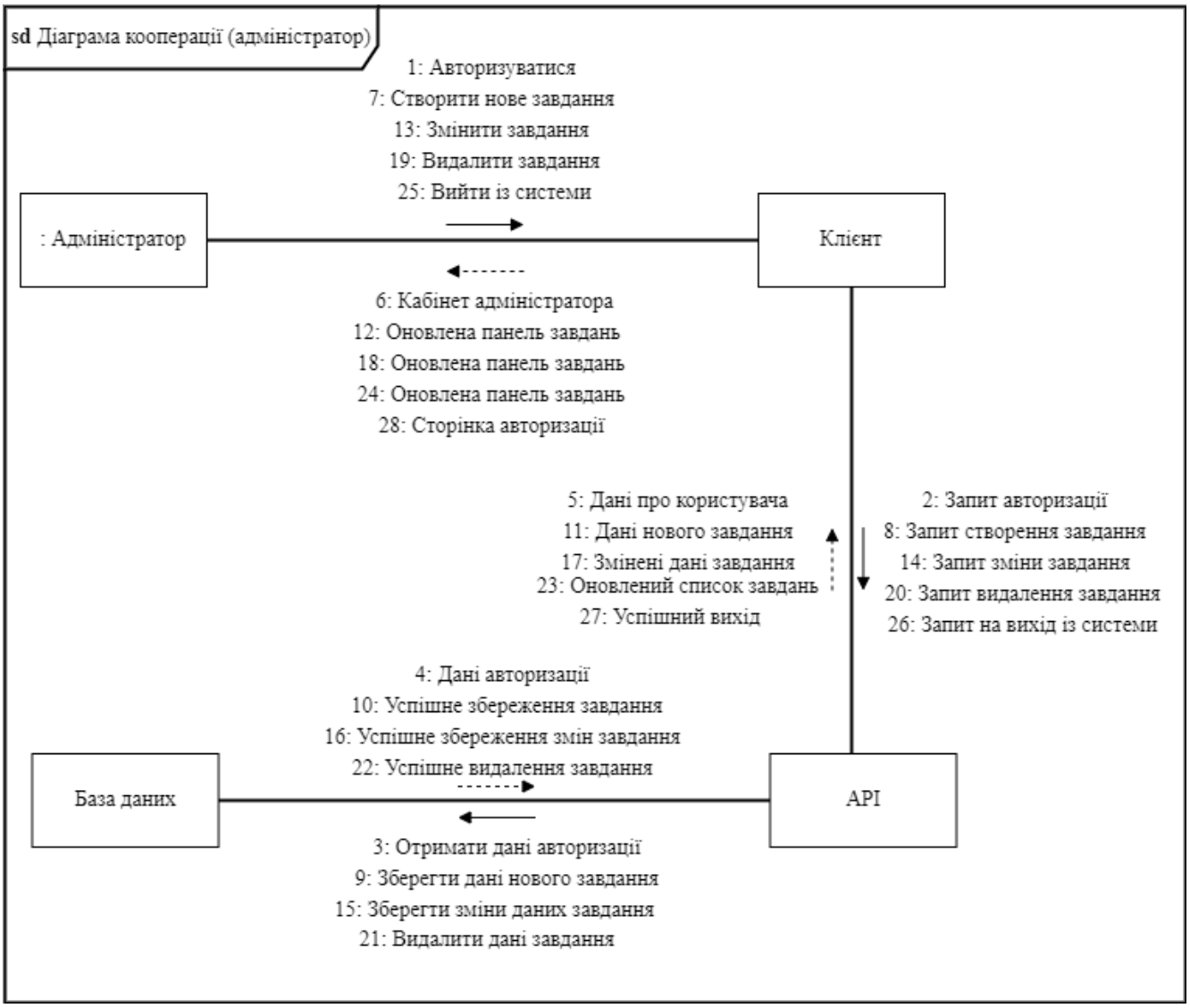

Рис. А.4. Діаграма кооперації

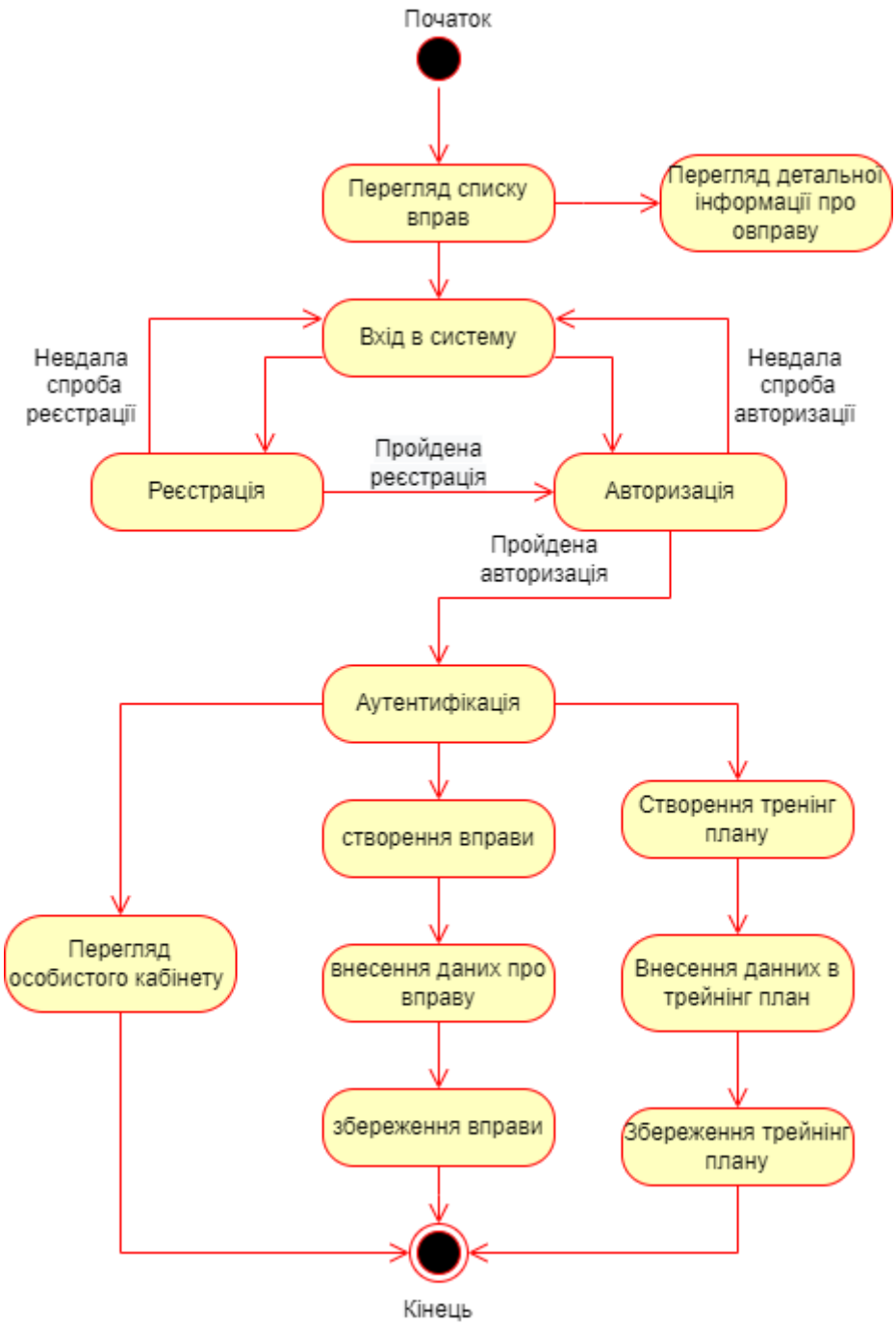

Рис. А.5. Діаграма станів

Додаток Г. Інструкція користувача

1. Компоненти програмного забезпечення

Розроблювана веб система хоститься в середовищі Azure DevOps. Відповідні всі необхідні компоненти для її роботи розміщені у цьому середовищі. Проект має серверну частину, написану на C# 9.0 у середовищі MS Visual Studio 2022. Для збереження даних на серверній частині використовується база даних MS SQL Server. Клієнтська частина написана з використанням бібліотеки React на мові програмування TypeScript у середовищі програмування MS Visual Studio Code.

2. Встановлення програмного забезпечення

Для запуску веб системи необхідно відкрити веб переглядач та ввести доменне ім’я WorOutLog. У результаті правильного введення доменного імені відобразиться головна сторінка веб системи.

3. Базові функції ПЗ

Вхід в систему. Щоб увійти в систему потрібно натиснути кнопку для входу в систему. Відобразиться модальне вікно із полями пошти та паролю. Користувач вводить дані і натискає кнопку підтвердження. Відбується вхід в систему.

Створення нових вправ або тренувальний планів. Для цього необхідно увійти в систему. Далі вибрати у верхній частині сторінки посилання на створення нового тренувального плану або вправи. Відбудеться перехід на сторінку створення нового тренувального плану або вправи. Користувач вводить дані і натискає кнопку для створення нової вправи. Новий трейнінг план або вправа зберігається в системі. І як результат відображається відповідне сповіщення і якщо перейти на сторінку зі списком вправ, новостворена вправа чи тренувальний план з’явиться у списку.

Перегляд вправ може виконати будь-який користувач, для цього не потрібен вхід в систему. Також будь-якой користувач може використати функцію фільтрування та сторування списку вправ для пролегшення пошуку необхідної вправи. Ця функція доступна на верхній частині головної сторінки. Для перегляду

детальної інформації стосовно вправи достатньо перейти на головну сторінку і вибрати зі списку будь-яку вправу. При виборі вправи відбудеться перехід на сторінку з детальною інформацією про вправу.

4. Аналіз помилок та можливих проблем

У випадку неправильної роботи веб системи необхідно дотримуватися підказок, які відображаються у випадку проблем у роботі системи.

Можна перевірити правильність налаштування середовища роботи.

Додаток В. Код

ExercisesManagement

```
using API.Controllers.Base;
using Business.Contract.Model.ExerciseManagement;
using Business.Contract.Services.ExerciseManagement;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers.ExercisesManagement
{
    [Route("api/[controller]")]
    [ApiController]
    public class ExercisesController : BaseController
    {
        private readonly IExerciseService _exerciseService;
        public ExercisesController(IExerciseService exerciseService)
        {
            _exerciseService = exerciseService;
        }

        [HttpPost]
        [Route("[action]")]
        [Authorize(AuthenticationSchemes = "Bearer")]
        public async Task<ActionResult> Create(CreateExerciseDto newExercise)
        {
            try
            {
                return Ok(await _exerciseService.Create(newExercise));
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPut]
        [Route("[action]")]
        [Authorize(AuthenticationSchemes = "Bearer")]
        public async Task<ActionResult> Update(ExerciseDto exercise)
        {
            try
            {
                await _exerciseService.Update(exercise);
                return Ok(exercise.Id + " exercise updated");
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpDelete]
        [Route("[action]")]
        [Authorize(AuthenticationSchemes = "Bearer")]
        public async Task<ActionResult> Delete(Guid exerciseId)
        {
            try
            {
                await _exerciseService.Delete(exerciseId);
                return Ok(exerciseId + " exercise deleted");
            }
```

```
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }

    [HttpGet]
    [Route("[action]")]
    [AllowAnonymous]
    public async Task<ActionResult> GetById(Guid exerciseId)
    {
        try
        {
            return Ok(await _exerciseService.GetById(exerciseId));
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }

    [HttpGet]
    [Route("[action]")]
    [AllowAnonymous]
    public async Task<ActionResult> GetByTarget(string name)
    {
        try
        {
            return Ok(await _exerciseService.GetByTarget(name));
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }

    [HttpGet]
    [Route("[action]")]
    [AllowAnonymous]
    public async Task<ActionResult> GetByEquipment(string name)
    {
        try
        {
            return Ok(await _exerciseService.GetByEquipment(name));
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
    }

    [HttpGet]
    [Route("[action]")]
    [AllowAnonymous]
    public async Task<ActionResult> GetByBodyParts(string name)
    {
        try
        {
            return Ok(await _exerciseService.GetByBodyParts(name));
        }
        catch (Exception ex)
        {
            return BadRequest(ex.Message);
        }
```

```
        }

        [HttpGet]
        [Route("[action]")]
        [AllowAnonymous]
        public async Task<ActionResult> GetBodyParts()
        {
            try
            {
                return Ok(await _exerciseService.GetBodyParts());
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet]
        [Route("[action]")]
        [AllowAnonymous]
        public async Task<ActionResult> GetAll()
        {
            try
            {
                return Ok(await _exerciseService.GetAll());
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }
}



using API.Controllers.Base;
using Business.Contract.Model.ExerciseManagement;
using Business.Contract.Model.ExerciseManagement.TrainingPlan;
using Business.Contract.Model.ExerciseManagement.TrainingPlan.ScheduledTrainingPlan;
using Business.Contract.Services.ExerciseManagement;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace API.Controllers.ExercisesManagement
{
    [Route("api/[controller]")]
    [ApiController]
    public class TrainingPlanController : BaseController
    {
        private readonly ITrainingPlanService _trainingPlanService;
        public TrainingPlanController(ITrainingPlanService trainingPlanService)
        {
            _trainingPlanService = trainingPlanService;
        }

        [HttpPost]
        [Route("[action]")]
        [Authorize(AuthenticationSchemes = "Bearer")]
        public ActionResult CreateSimpleTrainingPlan(CreateTrainingPlanDto
newTrainingPlan)
        {
            try
            {
```

```
                return Ok(_trainingPlanService.CreateSimpleTrainingPlan(newTrainingPlan,
GetUserId()));
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPost]
        [Route("[action]")]
        [Authorize(AuthenticationSchemes = "Bearer")]
        public ActionResult CreateScheduledTrainingPlan(CreateScheduledTrainingPlanDto
newTrainingPlan)
        {
            try
            {
                return
Ok(_trainingPlanService.CreateScheduledTrainingPlan(newTrainingPlan, GetUserId()));
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPut]
        [Route("[action]")]
        [Authorize(AuthenticationSchemes = "Bearer")]
        public async Task<ActionResult> Update(UpdateTrainingPlanDto
updateTrainingPlanDTO)
        {
            try
            {
                await _trainingPlanService.Update(updateTrainingPlanDTO);
                return Ok(updateTrainingPlanDTO.Id + " training plan updated");
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpDelete]
        [Route("[action]")]
        [Authorize(AuthenticationSchemes = "Bearer")]
        public async Task<ActionResult> Delete(Guid trainingPlanId)
        {
            try
            {
                await _trainingPlanService.Delete(trainingPlanId);
                return Ok(trainingPlanId + " exercise deleted");
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet]
        [Route("[action]")]
        [AllowAnonymous]
        public async Task<ActionResult> GetById(Guid trainingPlanId)
        {
```

```
            try
            {
                return Ok(await _trainingPlanService.GetById(trainingPlanId));
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet]
        [Route("[action]")]
        [AllowAnonymous]
        public async Task<ActionResult> GetAll()
        {
            try
            {
                return Ok(await _trainingPlanService.GetAll());
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet]
        [Route("[action]")]
        [AllowAnonymous]
        public async Task<ActionResult> GetAllByUserId()
        {
            try
            {
                return Ok(await _trainingPlanService.GetAll(GetUserId()));
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }
}
```

ExercisetManagement

```
using System;
using AutoMapper;
using Business.Contract.Model.ExerciseManagement;
using Business.Contract.Services.ExerciseManagement;
using Data.Contract.UnitOfWork;
using Domain.Entity.ExerciseManagement;
```

```
namespace Business.Services.ExercisetManagement
{
    public class ExerciseService : IExerciseService
    {
        private IMapper _mapper;
        private readonly IExerciseUnitOfWork _exerciseUnitOfWork;

        public ExerciseService(IMapper mapper, IExerciseUnitOfWork
exerciseUnitOfWork)
        {
            _mapper = mapper;
            _exerciseUnitOfWork = exerciseUnitOfWork;
        }

        public async Task<Guid> Create(CreateExerciseDto createExercise)
        {
            Exercise newExercise = _mapper.Map<Exercise>(createExercise);

            if (_exerciseUnitOfWork.ExerciseRepository.IsExist(createExercise.Name))
            {
                throw new Exception("Exercise with with name is exist");
            }

            if( createExercise.EquipmentName != null && createExercise.EquipmentName
!= "")
            {
                newExercise.EquipmentId = await AddEquipment(newExercise,
createExercise.EquipmentName);
                await _exerciseUnitOfWork.Save();
            }
```

```
        if (createExercise.TargetName != null && createExercise.EquipmentName != "")
        {
            newExercise.TargetId = await AddTarget(newExercise,
createExercise.TargetName);
            await _exerciseUnitOfWork.Save();
        }

        if (createExercise.BodyPartName != null && createExercise.BodyPartName !=
"")
        {
            newExercise.BodyPartId = await AddBodyPart(newExercise,
createExercise.BodyPartName);
            await _exerciseUnitOfWork.Save();
        }

        return await _exerciseUnitOfWork.ExerciseRepository.Add(newExercise);
    }

    public async Task Delete(Guid id)
    {
        if(id == Guid.Empty)
        {
            throw new Exception("Bad Id");
        }

        Exercise exercise = await
_exerciseUnitOfWork.ExerciseRepository.GetById(id);

        await _exerciseUnitOfWork.ExerciseRepository.Remove(exercise);
```

```
        await _exerciseUnitOfWork.Save();
    }

    public async Task<IEnumerable<ExerciseDto>> GetAll()
    {
        IEnumerable<Exercise> exercises = await
_exerciseUnitOfWork.ExerciseRepository.GetAll();
        List<ExerciseDto> exerciseDTOs= new List<ExerciseDto>();

        foreach(Exercise exercise in exercises)
        {
            ExerciseDto exerciseDTO = _mapper.Map<ExerciseDto>(exercise);

            if(exercise.EquipmentId != null)
            {
                Equipment equipment = await
_exerciseUnitOfWork.EquipmentRepository.GetById((Guid)exercise.EquipmentId);
                exerciseDTO.Equipment = equipment.Name;
            }
            if(exercise.TargetId != null)
            {
                Target target = await
_exerciseUnitOfWork.TargetRepository.GetById((Guid)exercise.TargetId);
                exerciseDTO.Target = target.Name;
            }
            if (exercise.BodyPartId != null)
            {
                BodyPart bodyPart = await
_exerciseUnitOfWork.BodyPartRepository.GetById((Guid)exercise.BodyPartId);
                exerciseDTO.BodyPart = bodyPart.Name;
```

```
            }

            exerciseDTOs.Add(exerciseDTO);
        }

        return exerciseDTOs;
    }

    public async Task<ExerciseDto> GetById(Guid exerciseId)
    {
        Exercise exercise = await _exerciseUnitOfWork.ExerciseRepository.GetById(exerciseId);
        ExerciseDto exerciseDTO = _mapper.Map<ExerciseDto>(exercise);

        if(exercise.EquipmentId != null)
        {
            Equipment equipment = await _exerciseUnitOfWork.EquipmentRepository.GetById((Guid)exercise.EquipmentId);
            exerciseDTO.Equipment = equipment.Name;
        }

        if(exercise.TargetId != null)
        {
            Target target = await _exerciseUnitOfWork.TargetRepository.GetById((Guid)exercise.TargetId);
            exerciseDTO.Target = target.Name;
        }

        if (exercise.BodyPartId != null)
        {
```

```
            BodyPart bodyPart = await _exerciseUnitOfWork.BodyPartRepository.GetById((Guid)exercise.BodyPartId);
            exerciseDTO.BodyPart = bodyPart.Name;
        }

        return exerciseDTO;
    }

    public async Task<IEnumerable<ExerciseDto>> GetByTarget(string name)
    {
        IEnumerable<Exercise> exercises = await _exerciseUnitOfWork.ExerciseRepository.GetByTarget(name);
        List<ExerciseDto> exerciseDTOs = new List<ExerciseDto>();

        foreach (Exercise exercise in exercises)
        {
            ExerciseDto exerciseDTO = _mapper.Map<ExerciseDto>(exercise);

            if(exercise.EquipmentId != null)
            {
                Equipment equipment = await _exerciseUnitOfWork.EquipmentRepository.GetById((Guid)exercise.EquipmentId);
                exerciseDTO.Equipment = equipment.Name;
            }

            if (exercise.TargetId != null)
            {
                Target target = await _exerciseUnitOfWork.TargetRepository.GetById((Guid)exercise.TargetId);
                exerciseDTO.Target = target.Name;
```

```
            }

            exerciseDTOs.Add(exerciseDTO);
        }

        return exerciseDTOs;
    }

    public async Task<IEnumerable<ExerciseDto>> GetByEquipment(string name)
    {
        IEnumerable<Exercise> exercises = await
_exerciseUnitOfWork.ExerciseRepository.GetByEquipment(name);
        List<ExerciseDto> exerciseDTOs = new List<ExerciseDto>();

        foreach (Exercise exercise in exercises)
        {
            ExerciseDto exerciseDTO = _mapper.Map<ExerciseDto>(exercise);

            if(exercise.EquipmentId != null)
            {
                Equipment equipment = await
_exerciseUnitOfWork.EquipmentRepository.GetById((Guid)exercise.EquipmentId);
                exerciseDTO.Equipment = equipment.Name;
            }
            if(exercise.TargetId != null)
            {
                Target target = await
_exerciseUnitOfWork.TargetRepository.GetById((Guid)exercise.TargetId);
                exerciseDTO.Target = target.Name;
            }
```

```
            exerciseDTOs.Add(exerciseDTO);
        }

        return exerciseDTOs;
    }

    public async Task<IEnumerable<ExerciseDto>> GetByBodyParts(string name)
    {
        IEnumerable<Exercise> exercises = await
_exerciseUnitOfWork.ExerciseRepository.GetByBodyParts(name);
        List<ExerciseDto> exerciseDTOs = new List<ExerciseDto>();

        foreach (Exercise exercise in exercises)
        {
            ExerciseDto exerciseDTO = _mapper.Map<ExerciseDto>(exercise);

            if (exercise.EquipmentId != null)
            {
                Equipment equipment = await
_exerciseUnitOfWork.EquipmentRepository.GetById((Guid)exercise.EquipmentId);
                exerciseDTO.Equipment = equipment.Name;
            }
            if (exercise.TargetId != null)
            {
                Target target = await
_exerciseUnitOfWork.TargetRepository.GetById((Guid)exercise.TargetId);
                exerciseDTO.Target = target.Name;
            }
```

```
            exerciseDTOs.Add(exerciseDTO);
        }

        return exerciseDTOs;
    }

    public async Task<IEnumerable<string>> GetBodyParts()
    {
        var bodyParts = await
_exerciseUnitOfWork.ExerciseRepository.GetBodyParts();
        List<string> parts = new List<string>();

        foreach(var bodyPart in bodyParts) {
            string newBopartString = bodyPart.Name;
            parts.Add(newBopartString);
        }

        return parts.ToArray();
    }

    public async Task Update(ExerciseDto updateExerciseDto)
    {
        if (updateExerciseDto.Id == null)
        {
            throw new Exception("invalid exercise Id");
        }

        Exercise exercise = await
_exerciseUnitOfWork.ExerciseRepository.GetById(updateExerciseDto.Id);
```

```
        if (exercise == null)
        {
            await Create(_mapper.Map<CreateExerciseDto>(updateExerciseDto));
        }
        else
        {
            //update Equipment
            if(updateExerciseDto.Equipment != null)
            {
                if (exercise.Equipment != null)
                {
                    if (exercise.Equipment.Name != updateExerciseDto.Equipment)
                    {
                        exercise.EquipmentId = await AddEquipment(exercise,
updateExerciseDto.Equipment);
                        await _exerciseUnitOfWork.Save();
                    }
                    //else if equipmnt is equal carrent value dont change carrent value
                }
                else
                {
                    exercise.EquipmentId = await AddEquipment(exercise,
updateExerciseDto.Equipment);
                    await _exerciseUnitOfWork.Save();
                }
            }

            //update Target
            if (updateExerciseDto.Target != null)
            {
```

```
            if (exercise.Target != null)
            {
                if (exercise.Target.Name != updateExerciseDto.Target)
                {
                    exercise.TargetId = await AddTarget(exercise,
updateExerciseDto.Target);
                    await _exerciseUnitOfWork.Save();
                }
                //else if target is equal carrent value dont change carrent value
            }
            else
            {
                exercise.TargetId = await AddTarget(exercise,
updateExerciseDto.Target);
                await _exerciseUnitOfWork.Save();
            }
        }

        //update BodyPart
        if (updateExerciseDto.BodyPart != null)
        {
            if (exercise.BodyPart != null)
            {
                if (exercise.BodyPart.Name != updateExerciseDto.BodyPart)
                {
                    exercise.BodyPartId = await AddBodyPart(exercise,
updateExerciseDto.BodyPart);
                    await _exerciseUnitOfWork.Save();
                }
                //else if bodyPart is equal carrent value dont change carrent value
```

```
            }
            else
            {
                exercise.BodyPartId = await AddBodyPart(exercise, updateExerciseDto.BodyPart);
                await _exerciseUnitOfWork.Save();
            }
        }



        //update name
        if (exercise.Name != updateExerciseDto.Name && updateExerciseDto.Name != null)
        {
            exercise.Name = updateExerciseDto.Name;
        }
        //update GifUrl
        if (exercise.GifUrl != updateExerciseDto.GifUrl && updateExerciseDto.GifUrl != null)
        {
            exercise.GifUrl = updateExerciseDto.GifUrl;
        }
        //update Calories
        if (exercise.Calories != updateExerciseDto.Calories && updateExerciseDto.Calories != 0)
        {
            exercise.Calories = updateExerciseDto.Calories;
        }

        await _exerciseUnitOfWork.ExerciseRepository.Update(exercise);
```

```
            await _exerciseUnitOfWork.Save();
        }
    }

    private async Task<Guid> AddEquipment(Exercise exercise, string equipment)
    {
        Guid equipmentId = Guid.NewGuid();

        if (_exerciseUnitOfWork.EquipmentRepository.IsExist(equipment))
        {
            equipmentId = (await _exerciseUnitOfWork.EquipmentRepository.GetByName(equipment)).Id;
        }
        else
        {
            Equipment newEquipment = new Equipment() { Id = equipmentId, Name = equipment };
            await _exerciseUnitOfWork.EquipmentRepository.Add(newEquipment);
        }
        return equipmentId;
    }

    private async Task<Guid> AddTarget(Exercise exercise, string target)
    {
        Guid targetId = Guid.NewGuid();

        if (_exerciseUnitOfWork.TargetRepository.IsExist(target))
        {
            targetId = (await _exerciseUnitOfWork.TargetRepository.GetByName(target)).Id;
```

```
            }
            else
            {
                Target newTarget = new Target() { Id = targetId, Name = target };
                await _exerciseUnitOfWork.TargetRepository.Add(newTarget);
            }
            return targetId;
        }

        private async Task<Guid> AddBodyPart(Exercise exercise, string bodyPart)
        {
            Guid bodyPartId = Guid.NewGuid();

            if (_exerciseUnitOfWork.BodyPartRepository.IsExist(bodyPart))
            {
                bodyPartId = (await
_exerciseUnitOfWork.BodyPartRepository.GetByName(bodyPart)).Id;
            }
            else
            {
                BodyPart newBodyPart = new BodyPart() { Id = bodyPartId, Name = bodyPart
};
                await _exerciseUnitOfWork.BodyPartRepository.Add(newBodyPart);
            }
            return bodyPartId;
        }
    }
}

using AutoMapper;
```

```
using Business.Contract.Model.ExerciseManagement;
using Business.Contract.Model.ExerciseManagement.TrainingPlan;
using Business.Contract.Model.ExerciseManagement.TrainingPlan.ReturnModel;
using
Business.Contract.Model.ExerciseManagement.TrainingPlan.ReturnModel.ScheduleTra
iningPlan;
using
Business.Contract.Model.ExerciseManagement.TrainingPlan.ScheduledTrainingPlan;
using Business.Contract.Services.ExerciseManagement;
using Data.Contract.UnitOfWork;
using Domain.Entity.ExerciseManagement;
using Domain.Entity.TrainingPlanManagement;
using Domain.Entity.TrainingPlanManagement.ManyToMany;
using System.Diagnostics;

namespace Business.Services.ExercisetManagement
{
    public class TrainingPlanService : ITrainingPlanService
    {
        private readonly IMapper _mapper;
        private readonly IExerciseUnitOfWork _exerciseUnitOfWork;

        public TrainingPlanService(IMapper mapper, IExerciseUnitOfWork
exerciseUnitOfWork)
        {
            _mapper = mapper;
            _exerciseUnitOfWork = exerciseUnitOfWork;
        }
```

```
public Guid CreateScheduledTrainingPlan(CreateScheduledTrainingPlanDto
createScheduledPlanExercise, Guid userId)
{
    TrainingPlan trainingPlan =
_mapper.Map<TrainingPlan>(createScheduledPlanExercise);
    trainingPlan.UserId =
(_exerciseUnitOfWork.UserRepository.GetByIdLink(userId).GetAwaiter().GetResult())
.Id;

    _exerciseUnitOfWork.TrainingPlanRepository.AddSync(trainingPlan);
    _exerciseUnitOfWork.SaveSync();

    if (createScheduledPlanExercise.DaySchedules != null)
    {
        foreach (var daySchedule in createScheduledPlanExercise.DaySchedules)
        {
            DaySchedule newDaySchedule =
_mapper.Map<DaySchedule>(daySchedule);
            newDaySchedule.TrainingPlanId = trainingPlan.Id;



_exerciseUnitOfWork.DayScheduleRepository.AddSync(newDaySchedule);
            _exerciseUnitOfWork.SaveSync();

            List<ScheduleExercise> newScheduleExercises = new
List<ScheduleExercise>();

            foreach(var sheduledExercise in daySchedule.ScheduleExercises)
            {
```

```
                var scheduleExercise =
_mapper.Map<ScheduleExercise>(sheduledExercise);
                scheduleExercise.DayScheduleId = newDaySchedule.Id;

                newScheduleExercises.Add(scheduleExercise);
            }



_exerciseUnitOfWork.ScheduleExerciseRepository.AddRangeSync(newScheduleExerci
ses);
            _exerciseUnitOfWork.SaveSync();
        }
    }

    return trainingPlan.Id;
}

public Guid CreateSimpleTrainingPlan(CreateTrainingPlanDto createPlanExercise,
Guid userId)
{
    TrainingPlan trainingPlan = _mapper.Map<TrainingPlan>(createPlanExercise);
    trainingPlan.UserId =
(_exerciseUnitOfWork.UserRepository.GetByIdLink(userId).GetAwaiter().GetResult())
.Id;

    if(createPlanExercise.PlanExerciseIds != null)
    {
        foreach (var itemId in createPlanExercise.PlanExerciseIds)
        {
```

```
                PlanExercise planExercise = new PlanExercise()
                {
                    ExerciseId = itemId
                };

                trainingPlan.PlanExercises.Add(planExercise);
            }
        }

        Guid trainingPlanId = _exerciseUnitOfWork.TrainingPlanRepository.AddSync(trainingPlan);

        _exerciseUnitOfWork.SaveSync();

        return trainingPlanId;
    }

    public async Task Delete(Guid id)
    {
        if (id == Guid.Empty)
        {
            throw new Exception("Bad Id");
        }

        await _exerciseUnitOfWork.PlanExerciseRepository.RemoveRange(id);

        TrainingPlan trainingPlan = await _exerciseUnitOfWork.TrainingPlanRepository.GetById(id);

        await _exerciseUnitOfWork.TrainingPlanRepository.Remove(trainingPlan);
```

```
        await _exerciseUnitOfWork.Save();
    }

    public async Task<GetAllTrainingPlansModel> GetAll()
    {
        var trainingPlans = await _exerciseUnitOfWork.TrainingPlanRepository.GetAll();
        var getAllTrainingPlansModelReply = new GetAllTrainingPlansModel();

        if (trainingPlans != null)
        {
            foreach (var trainingPlan in trainingPlans.Where(item => item.DaySchedules.Count == 0))
            {
                var returnTrainingPlanDTO = _mapper.Map<ReturnSimpleTrainingPlanDto>(trainingPlan);

                IEnumerable<Guid> exerciseIds = (await _exerciseUnitOfWork.PlanExerciseRepository.GetAllByTrainingPlanId(trainingPlan.Id)).Where(item => item.ExerciseId != null).Select(e => (Guid)e.ExerciseId).ToArray();

                if (!exerciseIds.Any())
                {
                    continue;
                }

                IEnumerable<Exercise> exercises = await _exerciseUnitOfWork.ExerciseRepository.GetRangeByIds(exerciseIds);
                returnTrainingPlanDTO.Exercises = _mapper.Map<ICollection<ExerciseDto>>(exercises);
```

```
getAllTrainingPlansModelReply.SimpleTrainingPlans.Add(returnTrainingPlanDTO);
            }
        }

        if (trainingPlans != null)
        {
            foreach (var trainingPlan in trainingPlans.Where(item => item.DaySchedules.Count != 0))
            {
                var returnTrainingPlanDTO = _mapper.Map<ReturnScheduleTrainingPlanDto>(trainingPlan);

                var daySchedules = await _exerciseUnitOfWork.DayScheduleRepository.GetDayScheduleRangeById(trainingPlan.Id);

                foreach (var daySchedule in daySchedules)
                {
                    try
                    {
                        var scheduleExercises = await _exerciseUnitOfWork.ScheduleExerciseRepository.GetAllByDayScheduleId(daySchedule.Id);

                        var dayScheduleDto = _mapper.Map<DayScheduleDto>(daySchedule);

                        foreach (var scheduleExercise in scheduleExercises)
```

```
                    {
                        var scheduleExerciseDto =
_mapper.Map<ScheduleExerciseDto>(scheduleExercise);
                        if (scheduleExercise.ExerciseId != null)
                        {
                            var exercise = await
_exerciseUnitOfWork.ExerciseRepository.GetById((Guid)scheduleExercise.ExerciseId)
;
                            scheduleExerciseDto.Exercise =
_mapper.Map<ExerciseDto>(exercise);
                        }
                    }
                    returnTrainingPlanDTO.DaySchedules.Add(dayScheduleDto);
                }
                catch (Exception ex)
                {
                    Debug.WriteLine(ex);
                }
            }


getAllTrainingPlansModelReply.ScheduleTrainingPlans.Add(returnTrainingPlanDTO);
            }
        }

        return getAllTrainingPlansModelReply;
    }
    public async Task<GetAllTrainingPlansModel> GetAll(Guid userId)
    {
```

```
var trainingPlans = await _exerciseUnitOfWork.TrainingPlanRepository.GetAllByUserId(userId);
var getAllTrainingPlansModelReply = new GetAllTrainingPlansModel();

if (trainingPlans != null)
{
    foreach (var trainingPlan in trainingPlans)
    {
        var returnTrainingPlanDTO = _mapper.Map<ReturnSimpleTrainingPlanDto>(trainingPlan);

        IEnumerable<Guid> exerciseIds = (await _exerciseUnitOfWork.PlanExerciseRepository.GetAllByTrainingPlanId(trainingPlan.Id)).Where(item => item.ExerciseId != null).Select(e => (Guid)e.ExerciseId).ToArray();

        if (!exerciseIds.Any())
        {
            continue;
        }

        IEnumerable<Exercise> exercises = await _exerciseUnitOfWork.ExerciseRepository.GetRangeByIds(exerciseIds);
        returnTrainingPlanDTO.Exercises = _mapper.Map<ICollection<ExerciseDto>>(exercises);


getAllTrainingPlansModelReply.SimpleTrainingPlans.Add(returnTrainingPlanDTO);
    }

    foreach (var trainingPlan in trainingPlans)
```

```
        {
            var returnTrainingPlanDTO = _mapper.Map<ReturnScheduleTrainingPlanDto>(trainingPlan);

            var daySchedules = await _exerciseUnitOfWork.DayScheduleRepository.GetDayScheduleRangeById(trainingPlan.Id);

            foreach (var daySchedule in daySchedules)
            {
                ScheduleExercise scheduleExercise = await _exerciseUnitOfWork.ScheduleExerciseRepository.GetByDayScheduleId(daySchedule.Id);

                if (scheduleExercise == null)
                {
                    continue;
                }

                var dayScheduleDto = _mapper.Map<DayScheduleDto>(daySchedule);

                var scheduleExerciseDto = _mapper.Map<ScheduleExerciseDto>(scheduleExercise);

                scheduleExerciseDto.Exercise = _mapper.Map<ExerciseDto>(_exerciseUnitOfWork.ExerciseRepository.GetById((Guid)scheduleExercise.ExerciseId));

                returnTrainingPlanDTO.DaySchedules.Add(dayScheduleDto);
            }
```

```
getAllTrainingPlansModelReply.ScheduleTrainingPlans.Add(returnTrainingPlanDTO);
            }
        }

        return getAllTrainingPlansModelReply;
    }

    public async Task<ReturnTrainingPlanDto> GetById(Guid trainingPlanId)
    {
        var trainingPlan = await
_exerciseUnitOfWork.TrainingPlanRepository.GetById(trainingPlanId);

        var returnTrainingPlanDTO =
_mapper.Map<ReturnTrainingPlanDto>(trainingPlan);

        IEnumerable<Guid> planExercises = (await
_exerciseUnitOfWork.PlanExerciseRepository.GetAllByTrainingPlanId(trainingPlanId)
).Select(e => e.Id).ToArray();

        IEnumerable<Exercise> exercises = await
_exerciseUnitOfWork.ExerciseRepository.GetRangeByIds(planExercises);

        returnTrainingPlanDTO.Exercises =
_mapper.Map<ICollection<ExerciseDto>>(exercises);

        var daySchedules = await
_exerciseUnitOfWork.DayScheduleRepository.GetDayScheduleRangeById(trainingPla
n.Id);
```

```
foreach (var daySchedule in daySchedules)
{
    try
    {
        var scheduleExercises = await _exerciseUnitOfWork.ScheduleExerciseRepository.GetAllByDayScheduleId(daySchedule.Id);

        var dayScheduleDto = _mapper.Map<DayScheduleDto>(daySchedule);

        foreach(var scheduleExercise in scheduleExercises)
        {
            var scheduleExerciseDto = _mapper.Map<ScheduleExerciseDto>(scheduleExercise);

            scheduleExerciseDto.Exercise = _mapper.Map<ExerciseDto>(_exerciseUnitOfWork.ExerciseRepository.GetById((Guid)scheduleExercise.ExerciseId));

            returnTrainingPlanDTO.DaySchedules.Add(dayScheduleDto);
        }
    }
    catch(Exception ex)
    {
        Debug.WriteLine(ex);
    }
}

return returnTrainingPlanDTO;
```

```
    }

    public async Task Update(UpdateTrainingPlanDto updateTrainingPlan)
    {
        TrainingPlan trainingPlan = await
_exerciseUnitOfWork.TrainingPlanRepository.GetById(updateTrainingPlan.Id);

        if(trainingPlan != null)
        {
            if (trainingPlan.StartDate != updateTrainingPlan.StartDate)
            {
                trainingPlan.StartDate = updateTrainingPlan.StartDate;
            }
            if (trainingPlan.EndDate != updateTrainingPlan.EndDate)
            {
                trainingPlan.EndDate = updateTrainingPlan.EndDate;
            }
            if (trainingPlan.Name != updateTrainingPlan.Name)
            {
                trainingPlan.Name = updateTrainingPlan.Name;
            }
            if (trainingPlan.UserId != updateTrainingPlan.UserId)
            {
                trainingPlan.UserId = updateTrainingPlan.UserId;
            }

            // check every exercise in trainig plan that they exist in updated item
            foreach(var PlanExercise in trainingPlan.PlanExercises)
            {
                if(PlanExercise != null)
```

```
                {
                    if (PlanExercise.ExerciseId != null &&
updateTrainingPlan.PlanExerciseIds.Contains((Guid)PlanExercise.ExerciseId))
                    {
                        continue;
                    }
                    await
_exerciseUnitOfWork.PlanExerciseRepository.Remove(PlanExercise);
                    await _exerciseUnitOfWork.Save();
                }
            }

            // check training plan that it contains updated exercises
            foreach (var PlanExerciseId in updateTrainingPlan.PlanExerciseIds)
            {
                if (trainingPlan.PlanExercises.Any( i => i.ExerciseId == PlanExerciseId))
                {
                    continue;
                }

                PlanExercise planExercise = new PlanExercise()
                {
                    TrainingPlanId = trainingPlan.Id,
                    ExerciseId = PlanExerciseId
                };

                await _exerciseUnitOfWork.PlanExerciseRepository.Add(planExercise);
                await _exerciseUnitOfWork.Save();
            }
        }
```

```
    }
  }
}
```

Domain.Entity.ExerciseManagement

```
using Domain.Entity.Base;

namespace Domain.Entity.ExerciseManagement
{
    public class BodyPart : EntityBase
    {
        public BodyPart()
        {
            Exercises = new HashSet<Exercise>();
            Name = "";
        }
        public string Name { get; set; }
        public virtual ICollection<Exercise> Exercises { get; set; }
    }
}
using Domain.Entity.Base;

namespace Domain.Entity.ExerciseManagement
{
    public partial class Equipment: EntityBase
    {
        public Equipment() {
            Exercises = new HashSet<Exercise>();
            Name = "";
        }
        public string Name { get; set; }
        public virtual ICollection<Exercise> Exercises { get; set; }
    }
}
using Domain.Entity.Base;
using Domain.Entity.TrainingPlanManagement.ManyToMany;

namespace Domain.Entity.ExerciseManagement
{
    public partial class Exercise : EntityBase
    {
        public Exercise() {
            PlanExercises = new HashSet<PlanExercise>();
            ScheduleExercises = new HashSet<ScheduleExercise>();
            Name = "";
            GifUrl = "";
        }

        public Guid? TargetId { get; set; }
        public Guid? EquipmentId { get; set; }
        public Guid? BodyPartId {  get; set; }
        public int Calories { get; set; }
        public string Name { get; set; }
        public string GifUrl { get; set; }
        public virtual Target? Target { get; set; }
        public virtual Equipment? Equipment { get; set; }

        public virtual BodyPart? BodyPart { get; set; }
```

```
        public virtual ICollection<PlanExercise> PlanExercises { get; set; }
        public virtual ICollection<ScheduleExercise> ScheduleExercises { get; set; }
    }
}
using Domain.Entity.Base;

namespace Domain.Entity.ExerciseManagement
{
    public partial class Target : EntityBase
    {
        public Target() {
            Exercises = new HashSet<Exercise>();
            Name = "";
        }
        public string Name { get; set; }
        public virtual ICollection<Exercise> Exercises { get; set; }

    }
}
using Domain.Entity.Base;
using Domain.Models;

namespace Domain.Entity.TrainingPlanManagement.ManyToMany
{
    public class DaySchedule: EntityBase
    {
        public DaySchedule()
        {
            Day = WeekDay.Monday;
            ScheduleExercises = new HashSet<ScheduleExercise>();
        }

        public WeekDay Day { get; set; }
        public Guid TrainingPlanId {  get; set; }

        public virtual ICollection<ScheduleExercise> ScheduleExercises { get; set; }
        public virtual TrainingPlan? TrainingPlan { get; set; }
    }
}
using Domain.Entity.Base;
using Domain.Entity.ExerciseManagement;

namespace Domain.Entity.TrainingPlanManagement.ManyToMany
{
    public class PlanExercise : EntityBase
    {
        public PlanExercise() { }
        public Guid? TrainingPlanId { get; set; }
        public Guid? ExerciseId { get; set; }

        public virtual TrainingPlan? TrainingPlan { get; set; }
        public virtual Exercise? Exercise { get; set; }
    }
}
using Domain.Entity.Base;
using Domain.Entity.ExerciseManagement;

namespace Domain.Entity.TrainingPlanManagement.ManyToMany
{
    public class ScheduleExercise: EntityBase
    {
        public ScheduleExercise() {
            StartDate = DateTime.Now;
        }
```

```
        public DateTime StartDate { get; set; }
        public int DurationInMinutes { get; set; }
        public int Frequency { get; set; }
        public Guid? ExerciseId { get; set; }
        public Guid? DayScheduleId { get; set; }
        public virtual Exercise? Exercise { get; set; }
        public virtual DaySchedule? DaySchedule { get; set; }
    }
}
using Domain.Entity.Base;
using Domain.Entity.TrainingPlanManagement.ManyToMany;
using Domain.Entity.Users;

namespace Domain.Entity.TrainingPlanManagement
{
    public partial class TrainingPlan : EntityBase
    {
        public TrainingPlan()
        {
            PlanExercises = new HashSet<PlanExercise>();
            Name = string.Empty;
            DaySchedules = new HashSet<DaySchedule>();
            IsNoDate = false;
        }
        public Guid? UserId { get; set; }
        public string? Name { get; set; }
        public DateTime StartDate { get; set; } = DateTime.Now;
        public DateTime EndDate { get; set; } = DateTime.Now;
        public virtual User? User { get; set; }
        public bool IsNoDate { get; set; }
        public virtual ICollection<PlanExercise> PlanExercises { get; set; }
        public virtual ICollection<DaySchedule> DaySchedules { get; set; }
    }
}
```

Клієнтська частини

```
import * as React from "react";
import { useState, useEffect } from "react";
import { TrainingPlanListContextType } from "./Types/TrainingPlanListType";
import { TrainingPlanList } from
"../../../../../../types/TrainingPlan/TrainingPlanList";
import { requestDefaultState } from
"../../../../../../components/Message/RequestMessage";
import { RequestResult } from "../../../../../../types/RequestResult";
import { fetchData } from "../../../../../../utils/fetchData";
import ConnectionConfig from
"../../../../../../assets/jsonData/ConnectionConfig/ConnectionConfig.json";

export const TrainingPlanListContext =
  React.createContext<TrainingPlanListContextType | null>(null);

const TrainingPlanListProvider: React.FC<React.PropsWithChildren> = ({
```

```
  children,
}) => {

  const [request, setRequest] =
useState<RequestResult>(requestDefaultState)

  const [trainingPlanList, setTrainingPlanList] =
useState<TrainingPlanList>({
    scheduledTrainingPlan: [],
    simpleTrainingPlanModel: [],
  });

  useEffect(() => {
    const GetAllTrainingPlans = async () => {
      let data: TrainingPlanList;
      data = await fetchData(
        ConnectionConfig.ServerUrl +
ConnectionConfig.Routes.Exercises.GetAll
      );
      setTrainingPlanList(data);
    };

    GetAllTrainingPlans();
  }, []);

  return (
    <TrainingPlanListContext.Provider
      value={{
        request,
        setRequest,
        trainingPlanList,
        scheduledTrainingPlans: trainingPlanList.scheduledTrainingPlan,
        simpleTrainingPlans: trainingPlanList.simpleTrainingPlanModel,
      }}
    >
      {children}
    </TrainingPlanListContext.Provider>
  );
};

export default TrainingPlanListProvider;

export const useTrainingPlanListContext = () =>
```

```
  React.useContext(TrainingPlanListContext);



import { useState } from "react";
import { Button, Form } from "react-bootstrap";
import styles from "../TrainingPlanStyle.module.sass";
import { AddExerciseModal } from "../AddExerciseModal/AddExerciseModal";
import { DayList } from "../Components/DayCarousel/DayList";
import { BadRequest, GoodRequest, requestDefaultState } from
"../../../../../../components/Message/RequestMessage";
import { useTrainingPlanContext } from
"../Components/Context/TrainingPlanContext";
import { TrainingPlanContextType } from
"../Components/Context/Types/TrainingPlanType";
import { fetchData } from "../../../../../../utils/fetchData";
import ConnectionConfig from
"../../../../../../assets/jsonData/ConnectionConfig/ConnectionConfig.json";
import PostService from
"../../../../../../components/Services/PostService";

const TrainingPlanForm: React.FC = () => {
  //daySchedule={state.daySchedules.length !== 0 ?
state.daySchedules.filter(item => item.day === day)[0] : null}
  const { state, dispatch, request, setRequest } = useTrainingPlanContext()
as TrainingPlanContextType;
  const [showModal, setShowModal] = useState(false);

  const handleSubmit = async() => {
    // Handle form submission
    let data = await PostService(
      ConnectionConfig.Routes.TrainingPlan.Create.ScheduledTrainingPlan,
      state
    );

    if (data) {
      setRequest(
        (prev: any) =>
          prev && {
            ...prev,
            good: { show: true, message: "training plan is created" },
          }
      );
```

```
    } else {
      setRequest(
        (prev: any) =>
          prev && {
            ...prev,
            bad: { show: true, message: "training plan isn't created" },
          }
      );
    }

    setTimeout(setRequest(requestDefaultState), 3000);
  };

  return (
    <div>
      <GoodRequest show={request.good.show} text={request.good.message} />
      <BadRequest show={request.bad.show} text={request.bad.message} />
      <Form>
        <Form.Group controlId="name">
          <Form.Label>Training Plan Name</Form.Label>
          <Form.Control
            type="text"
            value={state.name}
            onChange={(e) =>
              dispatch({ type: "SET_NAME", payload: e.target.value })
            }
          />
        </Form.Group>
        <Form.Group controlId="isNoDate">
          <Form.Check
            type="checkbox"
            label="No Date"
            checked={state.isNoDate}
            onChange={(e) =>
              dispatch({ type: "SET_IS_NO_DATE", payload: e.target.checked
})
            }
          />
        </Form.Group>
        {!state.isNoDate && (
          <>
            <Form.Group controlId="startDate">
              <Form.Label>Start Date</Form.Label>
```

```
                <Form.Control
                  type="date"
                  value={state.startDate.toISOString().split("T")[0]}
                  onChange={(e) =>
                    dispatch({
                      type: "SET_START_DATE",
                      payload: new Date(e.target.value),
                    })
                  }
                />
              </Form.Group>
              <Form.Group controlId="endDate">
                <Form.Label>End Date</Form.Label>
                <Form.Control
                  type="date"
                  value={state.endDate.toISOString().split("T")[0]}
                  onChange={(e) =>
                    dispatch({
                      type: "SET_END_DATE",
                      payload: new Date(e.target.value),
                    })
                  }
                />
              </Form.Group>
            </>
          )}
        </Form>
        <DayList
          setShowModal={setShowModal}
        />
        <div className={styles["save-training-plan-style"]}>
          <Button variant="success" onClick={()=>{handleSubmit()}}>
            Save Training Plan
          </Button>
        </div>
        <AddExerciseModal
          show={showModal}
          onHide={() => setShowModal(false)}
        />
      </div>
    );
  };
```

```
export default TrainingPlanForm;

import * as React from "react";
import { useState, useEffect, useReducer } from "react";
import {
  ScheduledTrainingPlanReducer,
  initialState,
} from "../Reducer/ScheduledTrainingPlanReducer";
import { TrainingPlanContextType } from "./Types/TrainingPlanType";
import { RequestResult } from "../../../../../../../types/RequestResult";
import { requestDefaultState } from
"../../../../../../../components/Message/RequestMessage";
import { CreateDaySchedule } from
"../../../../../../../types/TrainingPlan/ScheduledTrainingPlan/DaySchedule/
CreateDaySchedule";
import { WeekDay } from
"../../../../../../../types/TrainingPlan/ScheduledTrainingPlan/DaySchedule/
WeekDay";

export const TrainingPlanContext =
  React.createContext<TrainingPlanContextType | null>(null);

const TrainingPlanProvider: React.FC<React.PropsWithChildren> = ({
  children,
}) => {
  const [state, dispatch] = useReducer(
    ScheduledTrainingPlanReducer,
    initialState
  );

  const [day, setDay] = useState<WeekDay>(WeekDay.FRIDAY);
  const [dayName, setDayName] = useState<string>("");

  const [request, setRequest] =
useState<RequestResult>(requestDefaultState);

  const handleAddDaySchedule = (daySchedule: CreateDaySchedule) => {
    if (daySchedule.scheduleExercises.length === 0) {
      setRequest(
        (prev: any) =>
          prev && {
            ...prev,
            bad: {
```

```
            show: true,
            message: "day plan doesn't contain any exercise",
          },
        }
    );
    return;
  }

  dispatch({
    type: "ADD_DAY_SCHEDULE",
    payload: daySchedule,
  });
};

const dayScheduleInitialState = {
  day: day,
  scheduleExercises: [],
};

const [daySchedule, setDaySchedule] = useState<CreateDaySchedule>(
  dayScheduleInitialState
);

useEffect(() => {
  if (daySchedule.day !== day) {
    if (state.daySchedules.length === 0) {
      // console.log("test 0")
      setDaySchedule(dayScheduleInitialState);
    } else {
      let tempDaySchedule = state.daySchedules.find(
        (item: CreateDaySchedule) => item.day === day
      );
      // console.log(tempDaySchedule, day, state);
      setDaySchedule(
        tempDaySchedule !== undefined
          ? tempDaySchedule
          : dayScheduleInitialState
      );
    }
  }
}, [day]);

useEffect(() => {
```

```
    console.log(state);
  }, [state]);

  return (
    <TrainingPlanContext.Provider
      value={{
        state,
        dispatch,
        request,
        setRequest,
        handleAddDaySchedule,
        daySchedule,
        dayScheduleInitialState,
        setDaySchedule,
        day,
        setDay,
        dayName,
        setDayName,
      }}
    >
      {children}
    </TrainingPlanContext.Provider>
  );
};

export default TrainingPlanProvider;

export const useTrainingPlanContext = () =>
  React.useContext(TrainingPlanContext);
```