НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Програмне забезпечення для підтримки діяльності спортивного комплексу

Виконав студент IV курсу, групи ІТ-01
(шифр групи)

Колесник Роман Романович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник ст. викл., Халус О. А. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент проф., проф. каф. ІСТ Корнага Я. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Колеснику Роману Романовичу
(прізвище, ім'я, по батькові)

1. Тема проєкту Програмне забезпечення для підтримки діяльності спортивного комплексу

керівник проєкту Халус Олена Андріївна, ст. викл.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. №2112-с

2. Термін подання студентом проєкту « 17 » червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: аналіз предметної області, огляд ринку програмних продуктів та наявних алгоритмів, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: опис варіантів використань програмного забезпечення, аналіз системних вимог та розроблення функціональних та нефункціональних вимог.
3) Конструювання та розроблення програмного забезпечення: розроблення архітектури програмного забезпечення, обґрунтування засобів розробки, конструювання програмного забезпечення та аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості програмного забезпечення, опис процесів тестування та контрольного прикладу.

5) Розгортання та супровід програмного забезпечення: опис розгортання та супроводу програмного забезпечення.

5. Перелік графічного матеріалу
1) Схема структурна класів програмного забезпечення _____
2) Схема структурна класів програмного забезпечення _____
3) Схема бази даних _____
_____

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	25.03.2024	
3	Постановка та формалізація задачі	31.03.2024	
4	Розробка інформаційного забезпечення	07.04.2024	
5	Алгоритмізація задачі	14.04.2024	
6	Обґрунтування вибору використаних технічних засобів	21.04.2024	
7	Розробка програмного забезпечення	29.04.2024	
8	Налагодження програми	14.05.2024	
9	Виконання графічних документів	21.05.2024	
10	Оформлення пояснювальної записки	02.06.2024	
11	Подання ДП на попередній захист	02.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Роман КОЛЕСНИК
(підпис) (ініціали, прізвище)

Керівник _____ Олена ХАЛУС
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’яти розділів, містить 52 таблиць, 33 рисунків та 49 джерел – загалом 73 сторінки.

Дипломний проєкт присвячений створенню програмного забезпечення для підримки діяльості спортивного комплексу.

Метою розробки є покращення інструментів для обслуговування клієнтів та створення їм дієти, створення звітів, ведення занять в спортивному комплексі, запису та перегляду програми тренувань.

Об'єкт дослідження: програмне забезпечення для підримки діяльості спортивного комплексу.

Предмет дослідження: використання мікросервісної архітектури для вирішення задачі впровадження і супроводження програмного забезпечення для підтримки діяльності спортивного комплексу.

У розділі першому було проаналізовано предметну область, знайдені наявні продуктові та алгоритмічні рішення. Проведено порівняльний аналіз аналогів. Описано бізнес-процеси програмного забезпечення аналізу ігрових даних та поставлено задачу.

Другий розділ присвячений опису варіантів використання та розробленню вимог до програмного забезпечення.

Третій розділ описує розробку архітектури програмного забезпечення, обґрунтування засобів розробки та конструювання програмного забезпечення з подальшим аналіз безпеки даних.

У четвертому розділі було проведено аналіз якості програмного забезпечення. Були описані процеси тестування та контрольний приклад.

П’ятий розділ присвячений опису розгортання та супроводу програмного забезпечення.

Програмне забезпечення розгортуно в інфраструктурі AWS VPC.

КЛЮЧОВІ СЛОВА: СПОРТИВНИЙ КОМПЛЕКС, ДІЄТА, ЗВІТНІСТЬ, ТРЕНУВАННЯ, МІКРОСЕРВІСИ, SPRING, ANGULAR, NODE, MONGO, POSTGRES.

ABSTRACT

The explanatory note of the diploma project consists of five chapters, contains 52 tables, 33 figures, and 49 sources - a total of 73 pages.

The diploma project is dedicated to the creation of software to support the activities of a sports complex.

The purpose of the development is to improve tools for customer service and creating a diet for clients, generating reports, conducting classes in a sports complex, recording and viewing training programs.

Object of research: software to support the activities of a sports complex.

Subject of research: the process of developing, modifying, analyzing, ensuring quality, implementing and maintaining software to support the activities of a sports complex.

In chapter one, we analyzed the subject area and found existing product and algorithmic solutions. A comparative analysis of analogs was conducted. The business processes of the game data analysis software are described and the task is set.

The second section is devoted to the description of use cases and the development of software requirements.

The third section describes the development of the software architecture, the justification of the development tools, and the design of the software, followed by the analysis of data security.

The fourth section analyzes the software quality. Testing processes and a test case were described.

The fifth section is devoted to the description of software deployment and maintenance.

The software is deployed in the AWS VPC infrastructure.

KEYWORDS: SPORTS COMPLEX, DIET, REPORTING, TRAINING, MICROSERVICES, SPRING, ANGULAR, NODE, MONGO, POSTGRES.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІТ-0111.045440.01.91	Технічне завдання	16	
3	А4	КПІ.ІТ-0111.045440.02.81	Пояснювальна записка	73	
4	А4	КПІ.ІТ-0111.045440.03.12	Текст програми	51	
5	А4	КПІ.ІТ-0111.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІТ-0111.045440.05.34	Керівництво користувача	11	
7	А3	КПІ.ІТ-0111.045440.06.99	Графічний матеріал	3	

Змін.	Арк.	№ докум.	Підп.	Дата	КПІ.ІТ-0111.045440.00.90		
Розроб.		Колесник Р.Р			Відомість дипломного проєкту	Літ.	Аркуш 1 / Аркушів
Перевір.		Халус О. А.					
Н.контр.		Вітковська І.І.				КПІ ім. Ігоря Сікорського ФІОТ каф. ІПІ гр. ІТ-01	
Затв.		Жаріков Е.В.					

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Програмне забезпечення для підтримки діяльності спортивного комплексу

Технічне завдання

КПІ.ІТ-0111.045440.01.91

“ПОГОДЖЕНО”

Керівниця проєкту:

_____ Олена ХАЛУС

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Роман КОЛЕСНИК

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Програмне забезпечення для підтримки діяльності спортивного комплексу

Галузь застосування: контроль діяльності спортивного комплексу

Наведене технічне завдання поширюється на розробку веб-сервісу, яке використовується для підтримки діяльності спортивного комплексу та призначена для ведення занять в спортивному комплексі, запису та перегляду програми тренувань, створення дієт, створення звітів.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки Програмного забезпечення для підтримки діяльності спортивного комплексу є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для підтримки роботи спортивного комплексу.

Метою розробки є покращення інструментів для обслуговування клієнтів та створення їм дієти, створення звітів, ведення занять в спортивному комплексі, запису та перегляду програми тренувань.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу:

- авторизація для ролей тренер та користувач;
- реєстрація як користувач чи тренер;
- вибір спеціалізації тренера при реєстрації;
- сторінка зміни паролю;
- сторінка видалення облікового запису;
- сторінка оновлення облікового запису;
- сторінка зміни аватару користувача;
- наявність хедера, футера та навігації між сторінками;
- сторінка користувача - особистий кабінет;
- таблиця тренерів для користувача;
- сторінка тренера - особистий кабінет;
- таблиця користувачів для тренера;
- наявність динамічних сповіщень про стан запитів (тостери);
- підтвердження запитів на сервер (наявність модельних вікон);
- наявність вікна міні-профілю користувача;
- сторінка призначених тренінгів;
- сторінка запису на нові тренінги;
- сторінка для вказання вподобань у харчуванні і створенні дієти;
- сторінка генерації звітів;
- сторінка налаштування отримуваних сповіщень.

На рисунках 4.1 — 4.3 наведено вікна авторизації, реєстрації та зміни паролю.

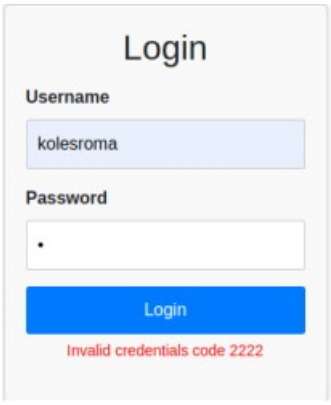


Рисунок 4.1 – Прототип сторінка входу

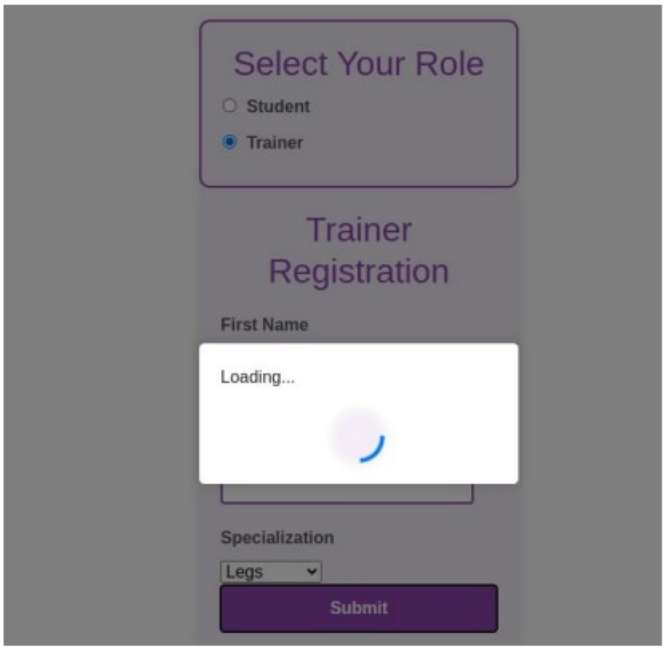


Рисунок 4.2 – Прототип сторінка реєстрації

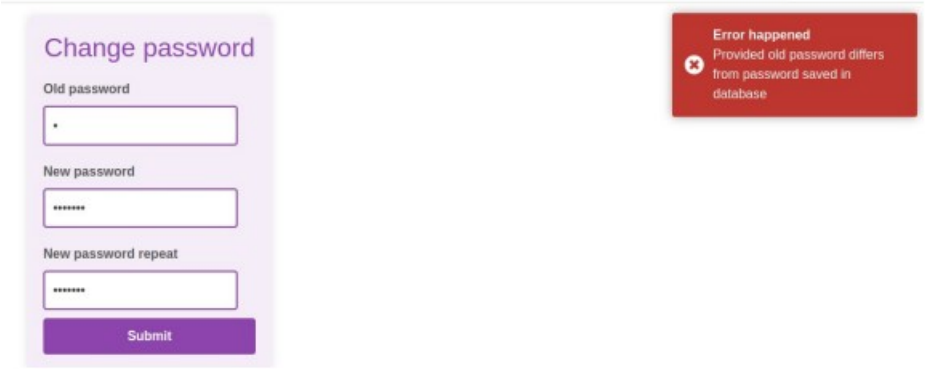


Рисунок 4.3 – Прототип зміни паролю (з валідацією)

На риснуках 4.4 — 4.6 наведено вікна для аміністрування облікового запису корстувача.

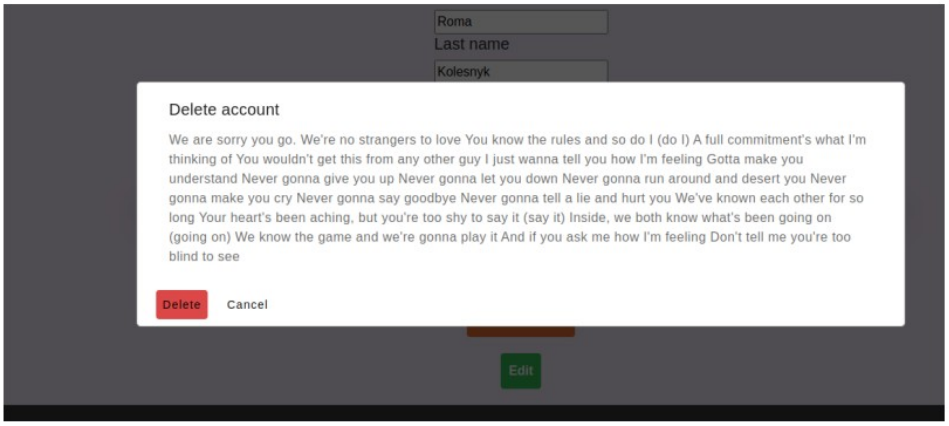


Рисунок 4.4 – Вікно видалення облікового запису

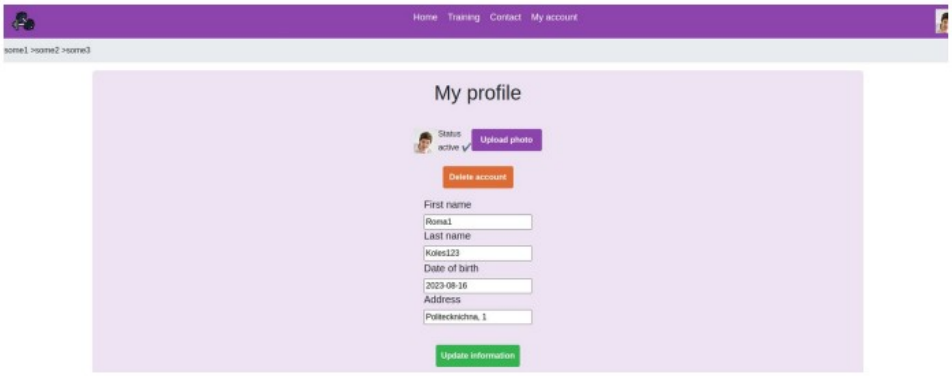


Рисунок 4.5 – Профіль користувача

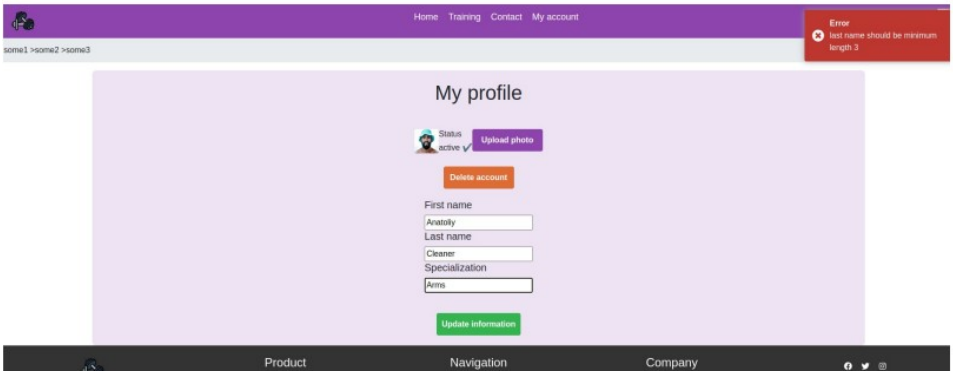


Рисунок 4.6 – Редагування профілю

На рисунках 4.7 — 4.10 наведено вікна для керування тренінгами.

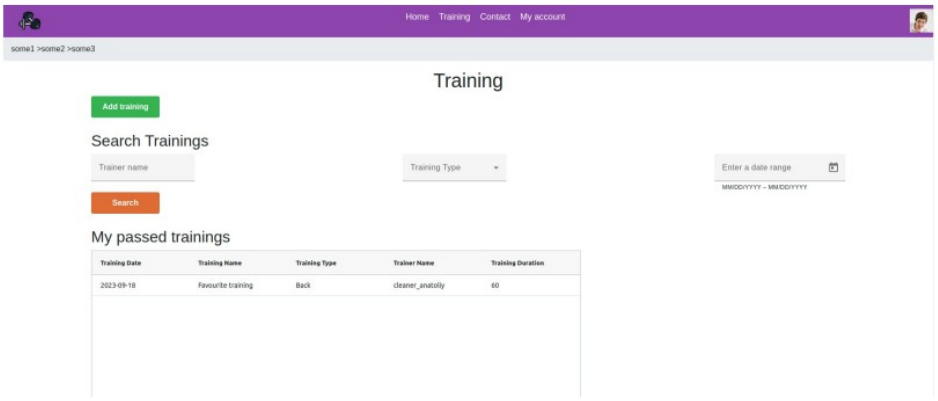


Рисунок 4.7 – Список призначених тренінгів для користувача

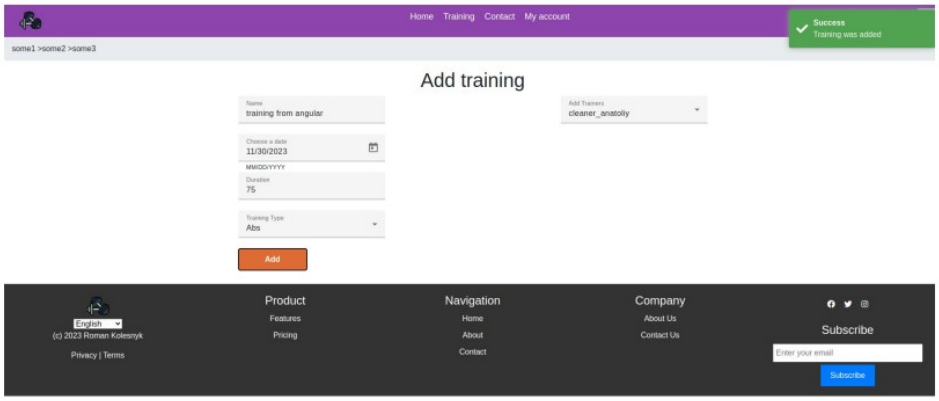


Рисунок 4.8 – Додавання тренengів

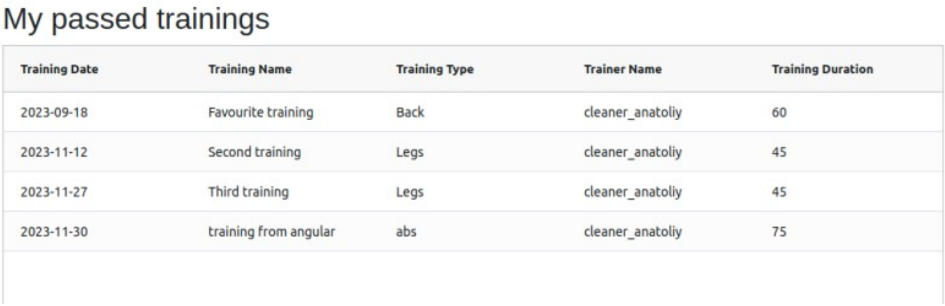


Рисунок 4.9 – Список призначених тренінгів

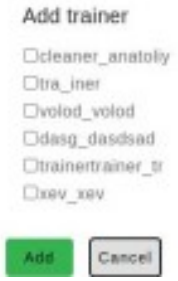


Рисунок 4.10 – Модельне вікно запис до нових тренерів

На рисунку 4.11 наведено вікно для налаштування отримуваних сповіщень, а на рисунку 4.12 – сторінку генерації дієти.

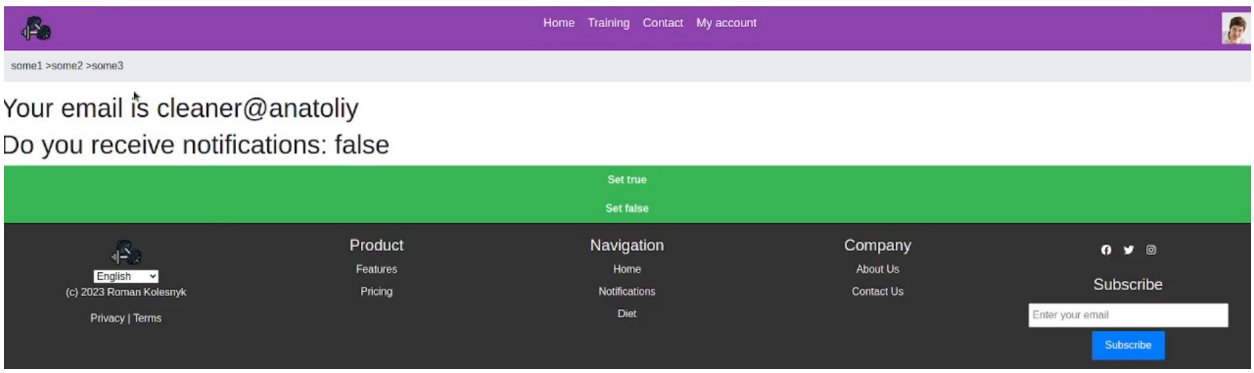


Рисунок 4.11 – Вікно налаштування сповіщень

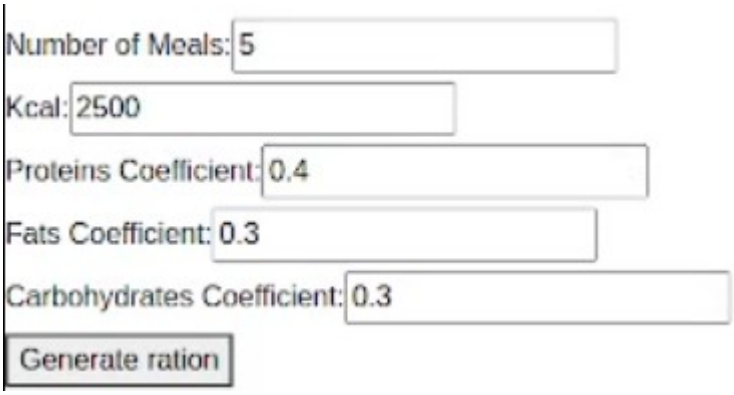


Рисунок 4.12 – Сторінка генерації дієти

4.1.2 Для користувача:

– зареєструватися;
– аторизуватися;
– переглянути свій профіль;
– редагувати профіль;
– оновити аватар користувача;
– видалити профіль;
– переглянути призначених тренерів;
– запис до нових тренерів;
– вибір програми занять (типи м'язів / тип тренувань);

– переглянути тренінги;
– налаштувати сповіщення.

4.1.3 Для тренера:

– авторизуватися;
– зареєструватися;
– переглянути свій профіль;
– редагувати профіль;
– оновити аватар користувача;
– видалити профіль;
– переглянути прив’язаних користувачів;
– переглянути тренінги;
– генерувати звіти роботи;
– згенерувати дієту для клієнта.

4.1.4 Додаткові вимоги:

– наявність адаптивності;
– валідація введених даних перед надсиланням на сервер.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на десктопах

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3;

– об‘єм ОЗП: 2 Гб;

– підключення до мережі Інтернет зі швидкістю від 20 мегабіт. Рекомендована конфігурація технічних засобів

– тип процесору: Intel Core i3;

– об‘єм ОЗП: 4 Гб;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням десктопних операційних систем з підтримкою браузера та доступом до інтеренету.

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: текст.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: веб-сторінка.

4.5.3 Вимоги до мови розробки

Розробку виконати на мовах програмування Java, JavaScript

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища IntelijIdea IDE UE.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у вигляді архіву та/або репозиторію на гітхаб.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати .war файл веб-застосунку

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача.

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема бази даних;
– схема структурна класів програмного забезпечення;
– схема структурна класів програмного забезпечення.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути самодокументовані, тобто тексти програм повинні бути написати декларативно.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1.	Вивчення літератури за тематикою проекту	21.02	
2.	Розробка технічного завдання	03.03	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	19.04	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	30.04	Схема структурна програмного забезпечення та специфікація компонентів (діаграма класів, схема алгоритму)
5.	Програмна реалізація програмного забезпечення	05.05	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	10.05	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	14.05	Пояснювальна записка
8.	Розробка матеріалів графічної частини проекту	20.05	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	29.05	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Пояснювальна записка
до дипломного проєкту

на тему: Програмне забезпечення для підтримки діяльності спортивного комплексу

КПІ.ІТ-0111.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

IDE	–	Integrated Development Environment – інтегроване середовище розробки.
API	–	Application programming interface, прикладний програмний Інтерфейс.
SDK	–	Software development kit.
SSL	–	Secure Sockets Layer, рівень захищених сокетів.
ORM	–	Object relation mapping, мапінг об’єктів.
IT	–	Інформаційні технології.
ER	–	Entity-Relation diagram.
ОС	–	Операційна система.
БД	–	База даних.
СУБД	–	Система управління базами даних
ООП	–	Об’єктно-орієнтоване програмування.

ВСТУП

У сучасному світі важливо забезпечити ефективне функціонування спортивних комплексів для задоволення потреб спортсменів та клієнтів. З метою оптимізації роботи таких комплексів та покращення якості обслуговування створюється програмне забезпечення, яке спрямоване на автоматизацію багатьох процесів управління та організації роботи спортивних закладів.

Розробка програмного забезпечення для підтримки життєдіяльності спортивного комплексу є актуальним завданням у зв'язку зі зростанням популярності здорового способу життя та активного відпочинку [1]. Це програмне забезпечення спрямоване на покращення ефективності управління різними аспектами діяльності спортивного комплексу, такими як реєстрація клієнтів, керування тренуваннями, складання розкладу, ведення обліку відвідуваності, створення програм тренувань та дієт, а також аналіз даних та звітності.

Метою даного проєкту є покращення інструментів для обслуговування клієнтів та створення дієти за рахунок створення програмного забезпечення. Дієта є важливим чинником в спортивному прогресі. Що сприяє досягненню спортивних цілей. У вступному розділі буде надано загальний огляд проекту, описано актуальність та значимість розробки програмного забезпечення для спортивних комплексів, а також сформульовано мету та завдання проекту.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Програмне забезпечення складається з інструкцій, які дозволяють користувачам взаємодіяти з комп'ютером і його периферійними пристроями. Воно включає операційні системи, застосунки, утиліти та інше ПЗ, яке керує функціями комп'ютера, починаючи від основних системних операцій до розширених графічних користувацьких інтерфейсів [2].

API, або прикладний програмний інтерфейс, – це комплекс визначень, протоколів і інструментів для розробки програмного забезпечення. В основі, API надає чітко встановлені методи для спілкування між різними компонентами системи, пропонуючи розробникам інструментарій для ефективного створення ПЗ. API можуть використовуватись у веб-системах, операційних системах, базах даних, а також у пристроях та програмних бібліотеках [3].

Термін "застосунок" може мати різні значення, зокрема в контексті API. Він може вказувати на програмний модуль з конкретною функціональністю, повний сервер або цілий застосунок, а також на окрему частину програми.

Всі частини програмного забезпечення, які можна ідентифікувати як самостійні одиниці, мають потенціал володіти API. Наприклад, коли розробник включає сторонню бібліотеку до програмного коду, вона стає частиною загального застосунку і надає API для взаємодії з іншими частинами програми.

ORM, або інструмент об'єктно-реляційного відображення, є ПЗ, яке допомагає розробникам, що використовують об'єктно-орієнтоване програмування (ООП), ефективно спілкуватися з реляційними базами даних без потреби створювати ORM з нуля, завдяки наявності готових рішень. [4].

Об'єктно-орієнтоване програмування організує дизайн програмного забезпечення навколо даних, або об'єктів, а не процедур чи логіки. Об'єкти в ООП визначаються як дані з унікальними атрибутами та поведінкою. Ця модель

особливо корисна для складних або широко оновлюваних програм, включно з системами для проектування, веб-сервісами та мобільними застосунками.

Розробники висловлювали критику об'єктно-орієнтованого програмування за його зосередженість на компонентах даних, що може затіняти важливість алгоритмів та обчислень, а також через підвищену складність коду та довший час компіляції [5].

Java, мова програмування, розроблена в 1995 році компанією Sun Microsystems, а з 2009 року під керуванням Oracle. Вона використовує байт-код, який інтерпретується віртуальною машиною для різних платформ, заснований на модифікованій об'єктній моделі C++, але зі спрощенням, що усуває деякі потенційні помилки розробників [6].

SSL, криптографічний протокол, розроблений компанією Netscape Communications, є основою для стандарту TLS, що був сформульований на базі SSL 3.0.

MySQL, вільна система управління базами даних, підтримується корпорацією Oracle. Ця система випускається як під вільною GNU General Public License, так і під комерційною ліцензією. Реплікаційні можливості були включені в MySQL майже з самого початку, завдяки попиту від ліцензійних користувачів [7].

Предметна область дипломного проєкту "Програмне забезпечення для підтримки функціонування спортивного комплексу" охоплює ряд процесів, необхідних для ефективної роботи спортивного комплексу за допомогою спеціалізованого ПЗ.

Здебільшого застосунки призначенні для покупки абонементів, бронювання занять. Деякі застосунки мають безлімітні програми чи системи бонусів.

Розробка призначеня для людей, які прагнуть підтримувати своє фізичне здоров’я в тонусі, зосереджені на здоровому харчуванні. Для них важливо вести здоровий спосіб життя.

Часто подібні застосунки зацікавлені лише в тому, щоб клієнт придбав послугу одноразово (абонемент, підписка). Але жодним чином не сприяють продовженню занять та мотивації.

У дипломному проєкті пропонується рішення, що більш зосереджене на клієнтах спортивного клубу. Клієнт — наш друг. Важливою частиною тренувань є стале харчування. Критично важливим є дотримання дієти як підкріплення звички занять спортом.

Регулярність є ключем успіху. У дипломному проєкті є функціонал надсилання повідомлень. Таким чином клієнту можна нагадати про важливість підтримання режиму тренувань чи повідомити про іншу зміну.

У рамках даної роботи програмне забезпечення розглядається як ключовий інструмент для покращення процесів обслуговування клієнтів, підвищення продуктивності та ефективності управління спортивним комплексом, створення дієти та ведення звітності.

1.2 Аналіз існуючих рішень

Проаналізуємо відоме на сьогодні алгоритмічне забезпечення у даній області та технічні рішення, що допоможуть у реалізації програмного забезпечення для підтримки діяльності спортивного комплексу. Далі будуть розглянуті допоміжні програмні засоби, засоби розробки та готові програмні рішення.

1.2.1 Аналіз відомих програмних продуктів

У цьому розділі наведено опис готових програмних продуктів за предметною областю, які реалізують частковий функціонал, описаний у технічному завданні.

Розглянуто приклади для порівняння - спорткомплекс “Галактика” (це спорткомплекс, що має річну систему абонементів та систему надсилання рекламних сповіщень), спортзал “пластилін“ (має сайт для запису на

тренування онлайн), мобільний застосунок “Fitness Buddy” (див. рисунок 1.1 — 1.3) [8-10].

Зазвичай системи з клієнтами використовують авторизацію для розмежування прав та ролей. Для застосунку розроблено систему авторизації.

“Пластилін” має свій сайт, де можна записатися на заняття онлайн. Оскільки це спортзал, то тренери пропонують дієтичні добавки, щоб покращити прогрес у залі. Можливо, клієнт зацікавиться харчуванням, але оскільки це теж нова звичка — добре харчуватися, — то він швидко знехтує звичкою, якщо про неї не нагадувати. Спортзал пропонує знижку жінкам.

Збалансоване харчування — найважливіша складова здорового тренування [11]. В оглянутих аналогах нема функціоналу генерування дієти, але вона присутня в розроблюваному програмному забезпеченні. Для автоматичного підбору продуктів налаштовані гнучкі критерії з можливістю вказати співвідношення білків, жирів та вуглеводів. Згенерований вміст обов’язково валідується дієтологом або тренером-нутриціологом. Використано генетичний алгоритм для генерації дієти.

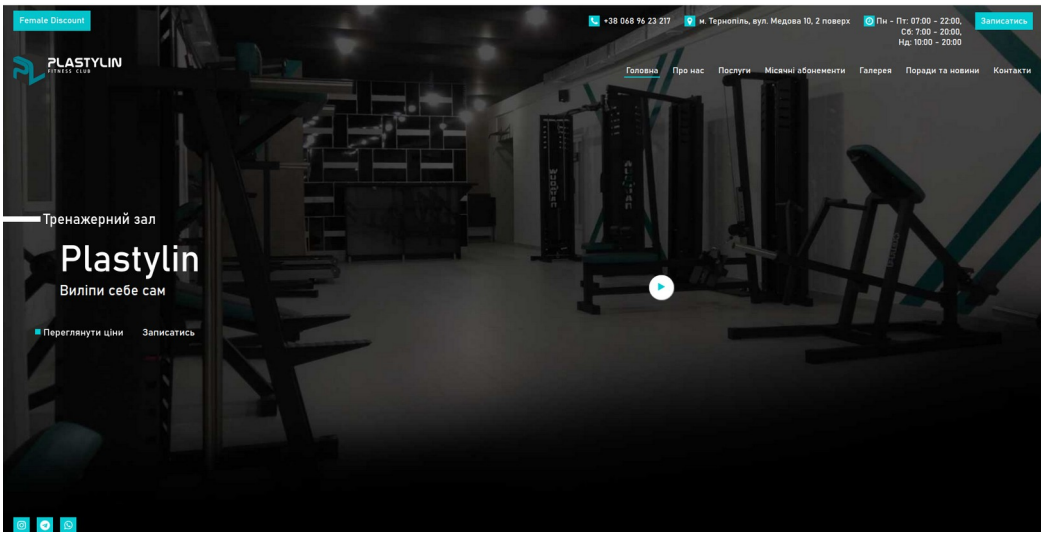


Рисунок 1.1 – Сайт спортзалу “Пластилін”

“Галактика” — це великий спорткомплекс, що має систему абонементів. Абонементи у ньому можна купити лише на рік. Реєстрація на сайті — це місце, де можна залишити свої дані. Є можливість переглянути зони занять.

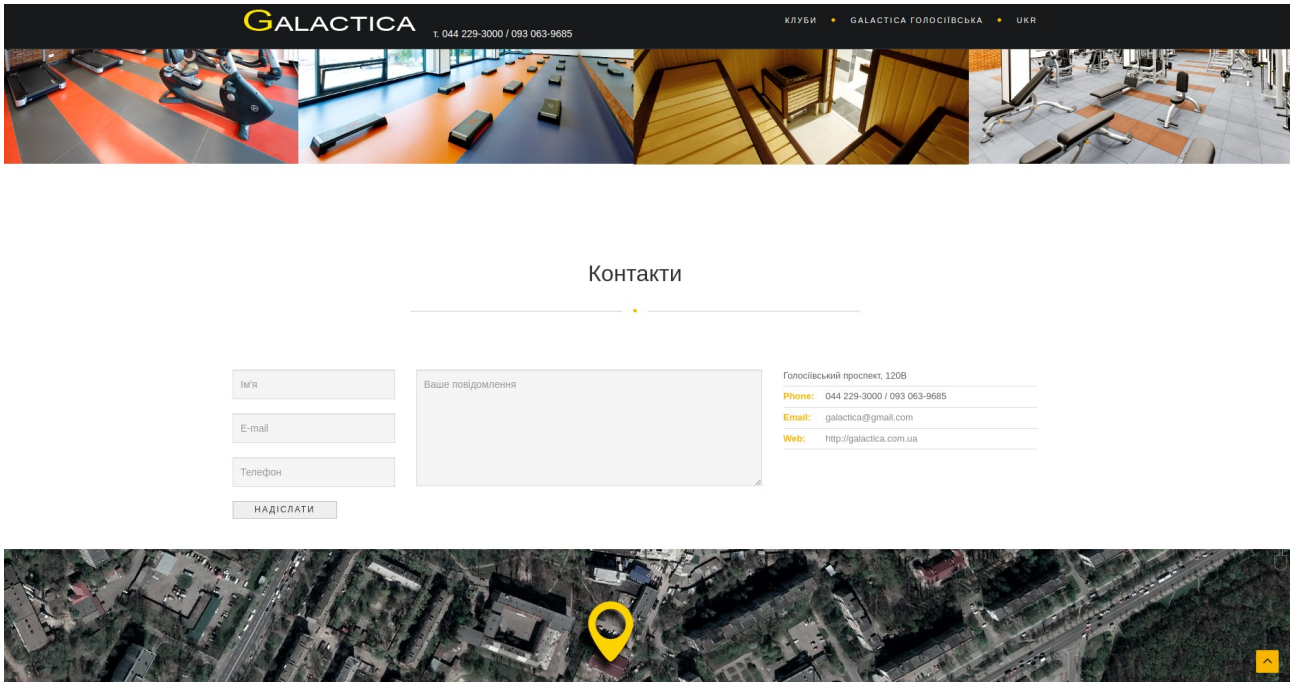


Рисунок 1.2 – Сайт спорткомплексу “Галактика”

Спостерігаються ситуації, коли клієнти купують абонемент та приходять лише кілька разів. Це тому, що підтримувати нову звичко складно та вони не відчувають обов’зку [12]. Помічено, що лише “Галактика” надсилає сповіщення клієнту, але це не того роду повідомлення. Це здебільшого розсилка реклами. Тим паче не вигідно для бізнесу розсилати нагадування прийти на тренування, бо компанія вже і так отримала гроші за абонемент. А розроблюване програмне забезпечення надсилає нагадування прийти на заняття і мотивує клієнта. Його концепція описується фразою “клієнт наш друг”.

Мобільний застосунок “Fitness Buddy” дає змогу переглядати доступні вправи онлайн, займатися не виходячи з дому. Безкошновний, але з рекламою. Може надсилати сповіщення на телефон про заняття, складати звіти тренувань.

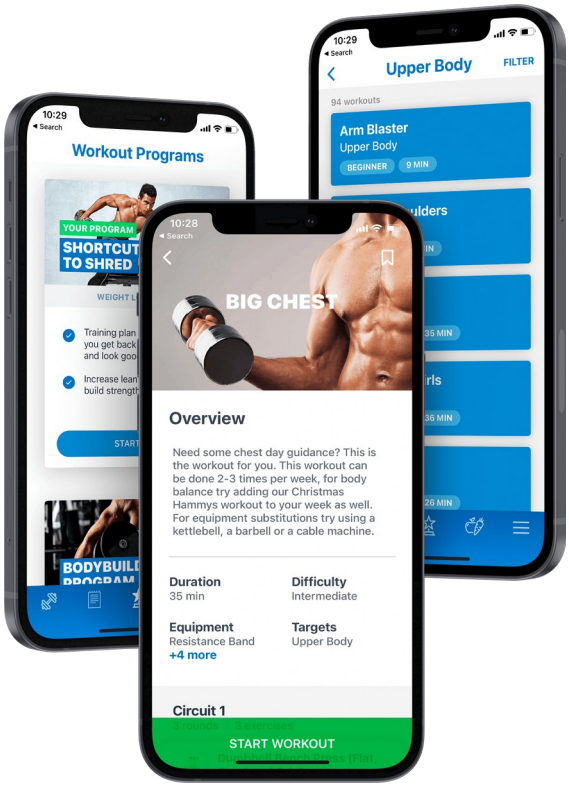


Рисунок 1.3 – Fitness Buddy — мобільний застосунок

У системах, що складаються з незалежних систем, використовують мікросервісну архітектура [13]. Мікросервісна архітектура легко масштабується. Очікується, що розроблене програмне забезпечення розширюватиметься в недалекому майбутньому.

Для порівняння дипломного проєкту з аналогами можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Програмне забезпечення для підтримки діяльності спортивного комплексу	Аналог для порівняння “Галактика”	Аналог для порівняння “Пластилін”	Аналог для порівняння “Fitness Buddy”
Автономність	-	-	-	+
Онлайн запис	+	+	+	-
Особистий кабінет	+	-	+	-
Система знижок	-	-	+	-
Створення дієти	+	-	-	+/-
Функція розсилки повідомлень	+	+/-	-	+
Функція генерації звітів	+	-	-	+

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Подібні сервіси повинні виконувати такі функції як: управління заняттями, облік відвідувачів, комунікація, створення звітів. Отже це застосунки з бізнес-логікою, що мають базу даних для зберігання інформації про користувачів.

Подібні сервіси мають клієнт-серверну архітектуру, що включає рівень представлення даних, тобто інтерфейс користувача, прикладний рівень, тобто реалізація основної логіки, а також рівень управління даними. Часто сервер є монолітним застосунком, що включає всі функції та компоненти в одній великій кодовій базі.

Мікросервісна архітектура стає все більш популярною завдяки своїй здатності забезпечувати гнучкість, масштабованість і незалежність компонентів програмного забезпечення. В контексті клієнт-серверної архітектури, мікросервіси відіграють ключову роль, забезпечуючи модульність та легкість підтримки.

Детальніше про приклади технологій, що використовують при розробці наведено у посиланнях [14-17].

1.3 Опис бізнес процесів

Для опису бізнес процесу програмного забезпечення використовується BPMN модель [18]. Схема бізнес процесу реєстрації та авторизації користувача представлена на рисунку 1.4.

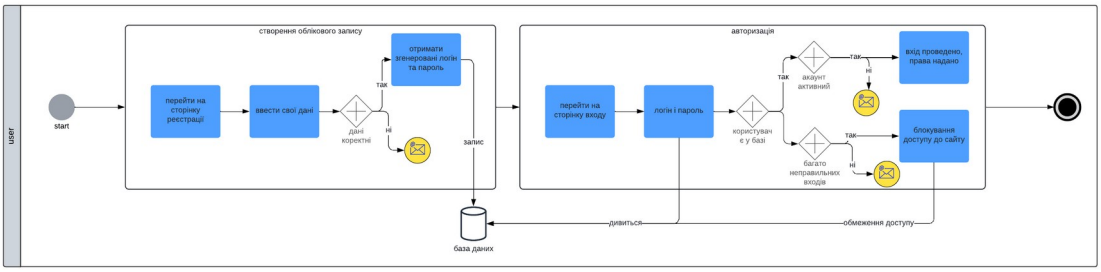


Рисунок 1.4 – Реєстрація та авторизація BPMN

Схема основного бізнес процесу користувача представлена на рисунку 1.5. Процес включає перегляд особистого кабінету, запис на тренування, відмітку відвідуваності, запис до дієтолога, генерацію дієти.

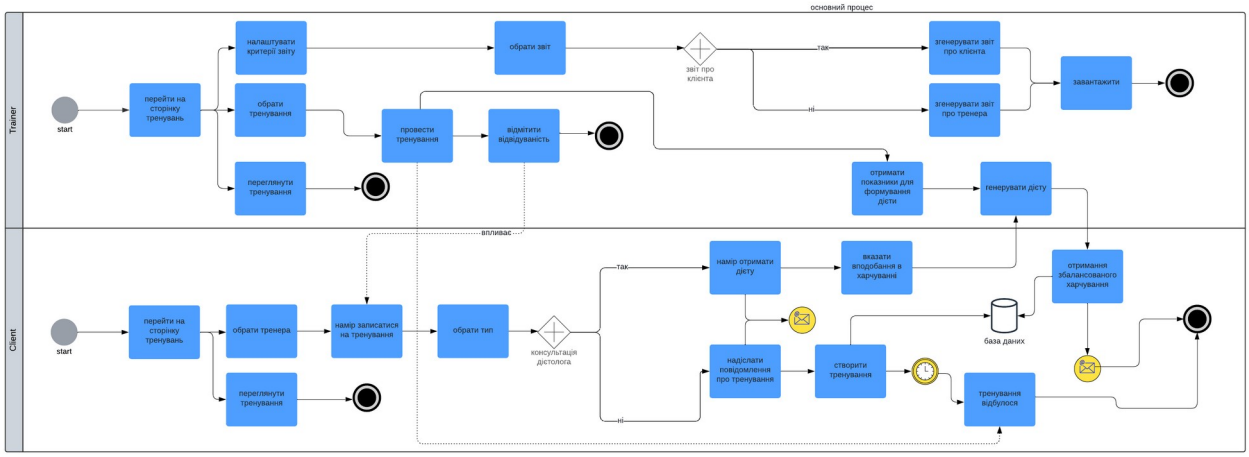


Рисунок 1.5 – Основний процес BPNM

Схема бізнес процесу адміністрування акаунту користувача представлена на рисунку 1.6.

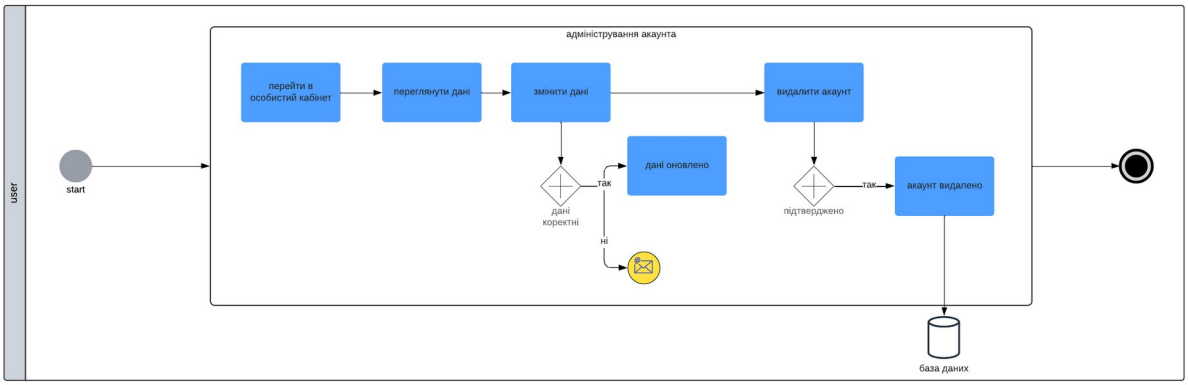


Рисунок 1.6 – Адміністрування акаунту BPNM

Схема бізнес процесу отримання і налаштування сповіщень користувача представлена на рисунку 1.7.

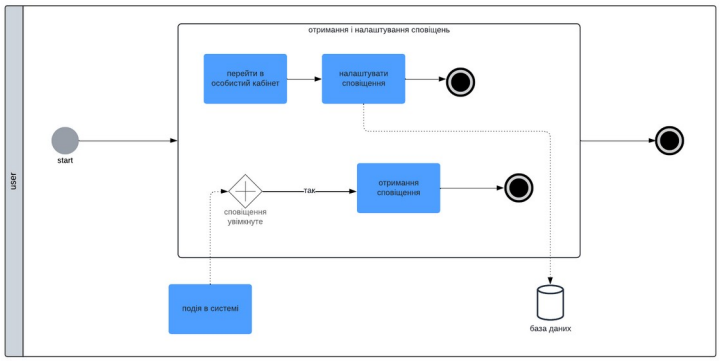


Рисунок 1.7 – Отримання і налаштування сповіщень BPNM

Опис послідовності генерації дієти:

– користувач записується на прийом дієтолога (зазвичай це перше заняття);

– у вказаний день він приходить, а тренер фіксує вподобання у дієті, регулярність прийомів їжі, мету харчування (набрати/скинути масу);

– тренер-дієтолог корегує дієту;

– користувач отримує режим збалансованого харчування.

1.4 Постановка задачі

Цілями цього дипломного проєкту є розробка програмного забезпечення для обслуговування клієнтів, створення дієти та генерації звіту, що є доступним на кожному приладі з підтримкою браузера. Застосунок забезпечує доступ до облікового запису спортивного комплексу.

Під час написання дипломного проєкту необхідно проаналізувати досвід використання вебзастосунків для підтримки спортивних комплексів. Вирішено розробити програмне забезпечення, що дасть можливість користувачу

отримувати доступ до системи за допомогою браузера, з можливістю записатись на тренування не виходячи з дому.

Данна розробка призначена для всіх учаників діяльності спортивного комплексу. Клієнтам вона дозволяє записатися на тренування та отримати дієту харчування. Тренери можуть відмічати відвідуваність та створювати звіти.

Метою дипломного проєкту є покращення інструментів для обслуговування клієнтів.

Для досягення мети необхідно реалізувати наступні задачі:

− авторизація користувачів для забезпечення безпеки та відповідності прав доступу до програмного забезпечення;

− адміністрування облікового запису;

− можливість запису на тренування для клієнтів;

− розсилка повідомлень для ефективного та своєчасного інформування користувачів про події в системі;

− генерація дієти, яка забезпечує клієнтам індивідуальний харчовий режим відповідно до їхніх потреб та побажань;

− генерація звітів про активність користувачів та загруженість тренерів.

Висновки до розділу

Проведено передпроєктне обстеження предметної області, а також розглянуто відомі алгоритмічні рішення та програмні продукти-аналоги.

У розділі з обгрунтуванням проекту зосереджено увагу на сфері спортивних клубів. Детально проаналізовано наявні технології та успішні IT-проєкти, зокрема, технології, використовані для створення серверної та клієнтської частини веб-застосунків. Серед розглянутих прикладів — спортивний комплекс “Галактика”, спортзал “Пластилін” та мобільний застосунок “Fitness Body”.

Використано засоби моделювання BPMN для опису бізнес процесів розробки та їх наочного представлення. Побудовано діаграму компонентів та діаграму послідовності для оцінки зданого завдання.

Сформульовано призначення, цілі та мету розробки програмного забезпечення в даному розділі, що надає зрозуміле та чітке уявлення про напрямки подальшої роботи.

Використання інструментів моделювання дозволяє краще розуміти бізнес-процеси та забезпечити їх ефективну реалізацію у програмному продукті. Сформульовані цілі та мети допомагають у направленні робочих зусиль на досягнення визначених завдань.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Головна функція програмного забезпечення полягає у створенні функціонального веб-застосунку за допомогою покращення інструментів для обслуговування клієнтів та створення дієти, який відповідає потребам цільової аудиторії. Більше функцій можна побачити нижче у таблицях.

Є два актори: тренер (trainer) та клієнт (client). На риснуку 2.1 наведено діаграму варіантів використання.

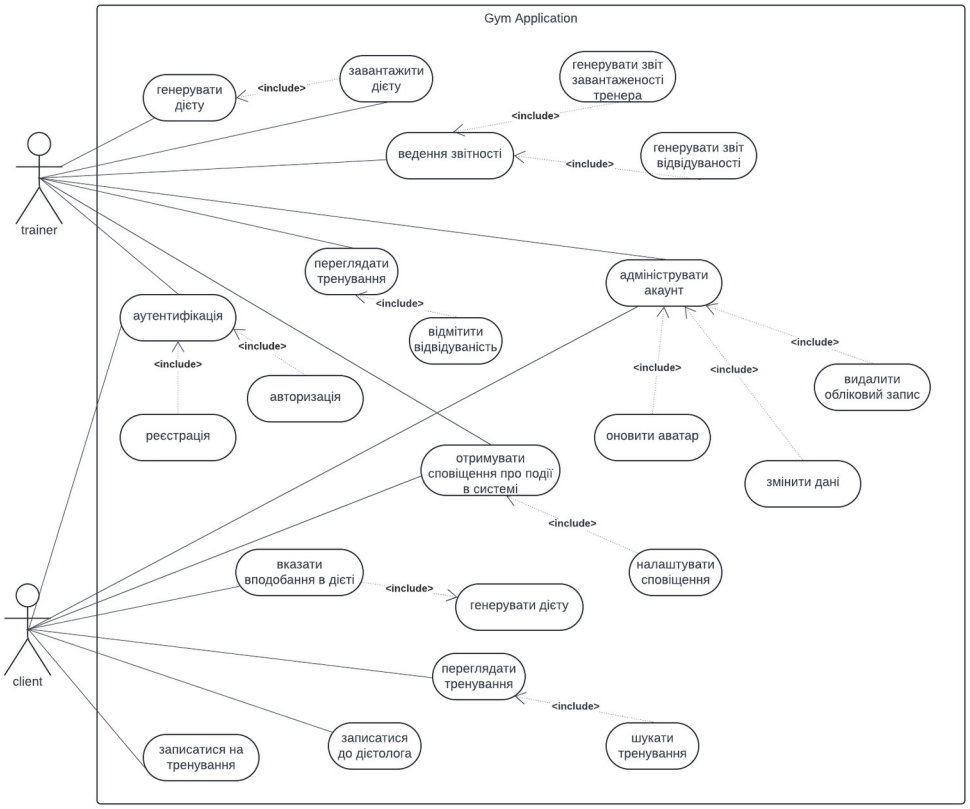


Рисунок 2.1 – Діаграма варіантів використання

2.2 Аналіз системних вимог

У таблицях 2.1 – 2.17 наведені варіанти використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-1

Use case name	Створення нового облікового запису
Use case ID	UC-01
Goals	Створення нового облікового запису
Actors	Новий користувач
Trigger	Користувач бажає створити обліковий запис
Pre-conditions	Користувач відкриває сторінку реєстрації
Flow of Events	Користувач вводить свої дані та натискає кнопку "Зареєструватись". Система перевіряє введені дані та створює новий обліковий запис
Extension	Якщо обліковий запис з такою адресою електронної пошти вже існує, то користувачу виводиться повідомлення про це
Post-Condition	Користувач успішно зареєстрований та може увійти до системи

Таблиця 2.2 – Варіант використання UC-2

Use case name	Вхід до системи
Use case ID	UC-2
Goals	Аутентифікація користувача
Actors	Користувач
Trigger	Користувач бажає увійти до системи
Pre-conditions	Користувач відкриває сторінку входу
Flow of Events	Користувач вводить свої дані для входу (логін та пароль) та натискає кнопку "Увійти". Система перевіряє введені дані та надає доступ користувачеві
Extension	Якщо введені дані не вірні, то користувачу виводиться повідомлення про помилку
Post-Condition	Користувач успішно автентифікований та отримує доступ до особистого кабінету

Таблиця 2.3 – Варіант використання UC-3

Use case name	Перегляд даних особистого кабінету
Use case ID	UC-3
Goals	Перегляд даних особистого кабінету
Actors	Зареєстрований користувач
Trigger	Користувач бажає переглянути свої особисті дані
Pre-conditions	Користувач успішно увійшов до системи
Flow of Events	Користувач переходить до свого особистого кабінету, де відображається його особиста інформація
Extension	-
Post-Condition	Користувач бачить дані про себе

Таблиця 2.4 – Варіант використання UC-4

Use case name	Оновлення особистих даних користувача
Use case ID	UC-4
Goals	Оновлення особистих даних користувача
Actors	Авторизований користувач
Trigger	Користувач відкриває сторінку редагування особистих даних та змінює необхідні дані. Після внесення змін користувач підтверджує їх
Pre-conditions	Користувач успішно увійшов до системи
Flow of Events	Користувач відкриває сторінку редагування особистих даних та змінює необхідні дані. Після внесення змін користувач підтверджує їх
Extension	-
Post-Condition	Особисті дані користувача успішно оновлені

Таблиця 2.5 – Варіант використання UC-5

Use case name	Встановлення аватару
Use case ID	UC-05
Goals	Додавання або зміна аватара користувача
Actors	Зареєстрований користувач
Trigger	Користувач бажає додати або змінити свій аватар
Pre-conditions	Користувач успішно увійшов до системи
Flow of Events	Користувач завантажує або вибирає новий файл з аватаром та підтверджує його встановлення
Extension	-
Post-Condition	Аватар користувача успішно встановлений

Таблиця 2.6 – Варіант використання UC-6

Use case name	Видалення облікового запису
Use case ID	UC-06
Goals	Видалення облікового запису користувача
Actors	Зареєстрований користувач
Trigger	Користувач бажає видалити свій обліковий запис
Pre-conditions	Користувач успішно увійшов до системи
Flow of Events	Користувач переходить до сторінки налаштувань облікового запису та обирає опцію "Видалити обліковий запис". Після цього система запитує підтвердження видалення
Extension	Якщо користувач вирішує скасувати видалення облікового запису, то операція не виконується
Post-Condition	Обліковий запис користувача успішно видалений, та він більше не має доступу до системи

Таблиця 2.7 – Варіант використання UC-7

Use case name	Запис клієнта до тренера на тренування м’язів
Use case ID	UC-07
Goals	Запис клієнта на тренування м'язів з тренером
Actors	Клієнт
Trigger	Клієнт бажає записатися на тренування м'язів з тренером
Pre-conditions	Клієнт та тренер повинні бути зареєстровані у системі
Flow of Events	Клієнт обирає доступну дату та час для тренування, підтверджує запис. Система реєструє запис та повідомляє тренера про новий запис
Extension	-
Post-Condition	Клієнт успішно записаний на тренування м'язів з тренером

Таблиця 2.8 – Варіант використання UC-8

Use case name	Запис клієнта на консультацію дієтолога
Use case ID	UC-8
Goals	Запис клієнта на консультацію з дієтологом
Actors	Клієнт
Trigger	Клієнт бажає записатися на консультацію з дієтологом
Pre-conditions	Клієнт повинен бути зареєстрований у системі
Flow of Events	Клієнт обирає доступну дату та час для консультації, підтверджує запис. Система реєструє запис та повідомляє дієтолога про новий запис
Extension	-
Post-Condition	Клієнт успішно записаний на консультацію з дієтологом

Таблиця 2.9 – Варіант використання UC-9

Use case name	Перегляд своїх тренувань
Use case ID	UC-9
Goals	Перегляд своїх тренувань
Actors	Клієнт
Trigger	Перегляд інформації про власні тренування
Pre-conditions	Клієнт повинен бути зареєстрований у системі та мати записи на тренування
Flow of Events	Клієнт переходить до свого особистого кабінету, де відображається інформація про його тренування
Extension	-
Post-Condition	Клієнт переглядає тренування

Таблиця 2.10 – Варіант використання UC-10

Use case name	Перегляд тренувань своїх клієнтів
Use case ID	UC-10
Goals	Перегляд інформації про тренування клієнтів
Actors	Тренер
Trigger	Тренер бажає переглянути тренування своїх клієнтів
Pre-conditions	Тренер повинен бути зареєстрований у системі
Flow of Events	Тренер переходить до свого особистого кабінету, де відображається інформація про тренування
Extension	-
Post-Condition	Тренер переглядає тренування

Таблиця 2.11 – Варіант використання UC-11

Use case name	Відмітка відвідуваності (для тренера)
Use case ID	UC-11
Goals	Позначення відвідуваності клієнтів на тренуваннях
Actors	Тренер
Trigger	Тренер бажає позначити відвідуваність своїх клієнтів на тренуваннях
Pre-conditions	Тренер повинен бути зареєстрований у системі та мати клієнтів, які записалися на тренування
Flow of Events	Тренер відкриває список тренувань з клієнтами, обирає конкретне тренування та позначає відвідуваність кожного клієнта
Extension	Якщо клієнт не з'явився на тренування, то тренер має можливість зафіксувати це
Post-Condition	Відвідуваність клієнтів на тренуванні відзначена у системі

Таблиця 2.12 – Варіант використання UC-12

Use case name	Отримання сповіщень про події в системі
Use case ID	UC-12
Goals	Отримання нагадувань про заплановані події
Actors	Тренер, клієнт
Trigger	Система автоматично надсилає нагадування про заплановані події (користувач налаштовує нагадування)
Pre-conditions	Користувач повинен бути зареєстрований у системі
Flow of Events	Система автоматично надсилає сповіщення користувачеві за певний час до початку події, повідомляє про зміни тощо
Extension	-
Post-Condition	Користувач отримує нагадування про заплановані події

Таблиця 2.13 – Варіант використання UC-13

Use case name	Налаштування сповіщень
Use case ID	UC-13
Goals	Налаштування сповіщень користувача
Actors	Тренер, клієнт
Trigger	Користувач бажає налаштувати сповіщення
Pre-conditions	Користувач повинен бути зареєстрований у системі
Flow of Events	Користувач встановлює налаштування для отримання сповіщень
Extension	-
Post-Condition	Сповіщення налаштовані відповідно до вказаних користувачем параметрів

Таблиця 2.14 – Варіант використання UC-14

Use case name	Створення та супровід дієти
Use case ID	UC-14
Goals	Створення та контроль дієти користувача
Actors	Тренер, клієнт
Trigger	Клієнт бажає скористатися послугами дієтолога
Pre-conditions	Користувач повинен бути зареєстрований у системі
Flow of Events	Тренер проводить консультацію з клієнтом, може згенерувати раціон харчування згідно з побажаннями клієнта, коригує генеровану дієту та надає рекомендації щодо харчування
Extension	-
Post-Condition	Клієнт отримує індивідуальний план харчування та може слідувати йому

Таблиця 2.15 – Варіант використання UC-15

Use case name	Налаштування побажань для харчування
Use case ID	UC-15
Goals	Налаштування побажань для харчування
Actors	Тренер, клієнт
Trigger	Клієнт бажає вказати свої побажання щодо харчування
Pre-conditions	Користувач повинен бути зареєстрований у системі
Flow of Events	Клієнт обговорює з дієтологом свої побажання щодо харчування, такі як кількість прийомів їжі, мета харчування
Extension	-
Post-Condition	Клієнт вказав свої побажання щодо харчування, які будуть враховані при розробці дієти

Таблиця 2.16 – Варіант використання UC-16

Use case name	Генерація звіту по відвідуваності клієнтів
Use case ID	UC-16
Goals	Генерація звіту по відвідуваності клієнтів
Actors	Тренер
Trigger	Тренер бажає отримати звіт про відвідуваність клієнтів
Pre-conditions	Тренер повинен бути зареєстрований у системі
Flow of Events	Система генерує звіт, що містить інформацію про відвідуваність клієнтів за певний період часу
Extension	-
Post-Condition	Тренер отримує звіт, який можна використовувати для аналізу та прийняття рішень

Таблиця 2.17 – Варіант використання UC-17

Use case name	Генерація звіту по загруженості тренерів
Use case ID	UC-17
Goals	Збір та аналіз інформації про робоче навантаження тренерів
Actors	Тренер
Trigger	Тренер бажає отримати звіт про завантаженість
Pre-conditions	Тренер повинен бути зареєстрований у системі
Flow of Events	Система генерує звіт, що містить інформацію про кількість тренувань, які провів тренер за певний період часу
Extension	-
Post-Condition	Тренер отримує звіт, який можна використовувати для аналізу та прийняття рішень

2.3 Розроблення функціональних вимог

Програмне забезпечення розділене на модулі. Кожен модуль має свій певний набір функцій. У таблицях 2.18 – 2.28 наведений опис функціональних вимог до програмного забезпечення. Матрицю трасування вимог можна побачити на рисунку 2.2.

Таблиця 2.18 – Функціональна вимога FR-1

Назва	Реєстрація як клієнт і як тренер
Опис	Система повинна надавати можливість реєстрації як клієнта і тренера

Таблиця 2.19 – Функціональна вимога FR-2

Назва	Авторизація
Опис	Система повинна надавати можливість користувачеві авторизуватися в системі.

Таблиця 2.20 – Функціональна вимога FR-3

Назва	Захист від брут форс
Опис	Система повинна мати механізми захисту, які унеможливлюють атаки брут форс

Таблиця 2.21 – Функціональна вимога FR-4

Назва	Перегляд особистого кабінету
Опис	Користувач повинен мати можливість переглядати свій особистий кабінет у системі.

Таблиця 2.22 – Функціональна вимога FR-5

Назва	Адміністрування облікового запису
Опис	Користувач повинен вміти керувати обліковим записом у системі

Таблиця 2.23 – Функціональна вимога FR-6

Назва	Створення тренувань
Опис	Клієнт повинен мати можливість створювати тренування та записуватися на них.

Таблиця 2.24 – Функціональна вимога FR-7

Назва	Перегляд тренувань
Опис	Користувач (клієнт або тренер) повинен мати можливість переглядати доступні тренування.

Таблиця 2.25 – Функціональна вимога FR-8

Назва	Ведення звітності по тренуваннях
Опис	Тренер повинен мати можливість вести звітність про проведені тренування

Таблиця 2.26 – Функціональна вимога FR-9

Назва	Отримання та адміністрування отримуваних сповіщень
Опис	Користувач повинен мати можливість отримувати та керувати отримуваними сповіщеннями.

Таблиця 2.27 – Функціональна вимога FR-10

Назва	Створення дієти
Опис	Система повинна надавати можливість клієнтові створити і зберегти індивідуальний харчовий режим.

Таблиця 2.28 – Функціональна вимога FR-11

Назва	Генерація звіту
Опис	Система повинна мати можливість генерувати звіти тренерів і клієнтів

	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17
FR-1	1																
FR-2		1															
FR-3		1															
FR-4			1														
FR-5				1	1	1											
FR-6							1	1									
FR-7									1	1							
FR-8											1						
FR-9												1	1				
FR-10														1	1		
FR-11																1	1

Рисунок 2.2 – Матриця трасування вимог

2.4 Розроблення нефункціональних вимог

Система має відповідати наступним нефункціональним вимогам, описаних у таблицях 2.29 – 2.30.

Таблиця 2.29 – Нефункціональна вимога NFR-1

Назва	Повинна мати відкриту архітектуру
Опис	Система повинна володіти відкритою архітектурою. Таким чином, система може масштабуватися горизонтально. Усі процеси повинні бути розділені на логічні рівні

Таблиця 2.30 – Нефункціональна вимога NFR-2

Назва	Інтерфейс користувача має бути зручним та інтуїтивно-зрозумілим
Опис	Інтерфейс користувача повинен бути побудованим за правилами, визначеними у додатку А

Висновки до розділу

Проаналізовано структуру проекту, включаючи загальні принципи його функціонування та взаємодію компонентів. Проведено аналіз вимог до програмного забезпечення, що включає діаграму варіантів використання та опис функціональних вимог у формі таблиць. Додатково були визначені нефункціональні вимоги для забезпечення високої якості та ефективності програмного продукту.

Розглянуто варіанти використання з двома основними акторами — тренер та клієнт. Основні функції включають в себе запис на тренування до тренера, перегляд історії тренувань, відмітку відвідуваності, налаштування вподобань в їжі, генерацію дієти, отримання та налаштування сповіщень, а також генерацію звітів для тренерів.

Описано функціонально та нефункціональні вимоги до системи. Наведено матрицю трасування функціональних вимог.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

Розроблене програмне забезпечення має клієнт-серверну структуру, що означає, що воно складається з двох основних частин: клієнтської та серверної [19]. Серверна частина відповідає за обробку запитів на доступ до ресурсів та логіку роботи з даними системи. Клієнтська частина відповідає за представлення інтерфейсу користувача та доступ до ресурсів, які надає сервер.

Вибір мікросервісної архітектури над монолітною може бути вигідним завдяки гнучкості, масштабованості, можливості використання різноманітних технологій та полегшенню супроводження та розвитку застосунку [20-21]. Мікросервіси дозволяють легше розподіляти ризики та забезпечувати відокремленість помилок. На противагу ця архітектура вносить складність у керування застосунком та потребує більше уваги до аспектів розгортання та моніторингу.

Для розробки серверної частини застосунку використано архітектуру мікросервісів. Діаграму взаємодії мікросервісів зображено на рисунку 3.1.

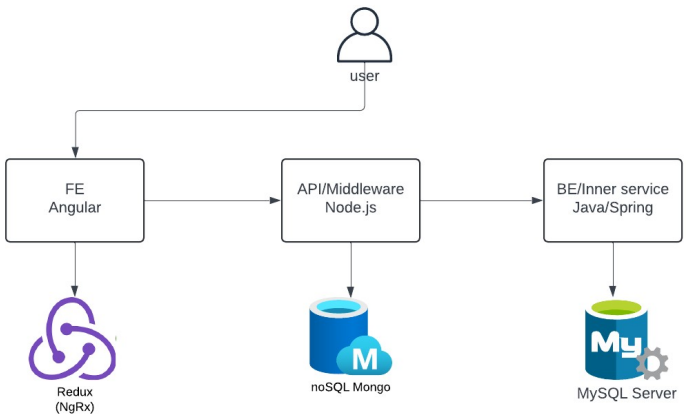


Рисунок 3.1 – Діаграма взаємодії мікросервісів

Усі запити користувача обробляються за допомогою Angular, який зберігє стани за допомогю підходу Redux та бібліотеки NgRx [22-23].

Мікросервіс FE Angular спілкується з мікросервісами за допомогою протоколу HTTP.

Мікросервіс API/Middleware Node.js виконує роль прослойки між фронтендом та основним застосунком Java/Spring у деяких запитах (Middleware) [24]. Його роль зменшити нагрузку під час авторизації та зберігання аватарів користувачів.

Мікросервіс BE/Inner service Java/Spring обробляє основну бізнес логіку. До нього звертається Node.js мікросервіс та FE angular.

Діаграму компонентів та їх взаємодію мікросервіса FE angular показано на рисунку 3.2. Є два логічні рівні — вигляд та сервіс. Сервіс звертається до зовнішніх мікросервісів і повертає отримані дані на рівень вигляду.

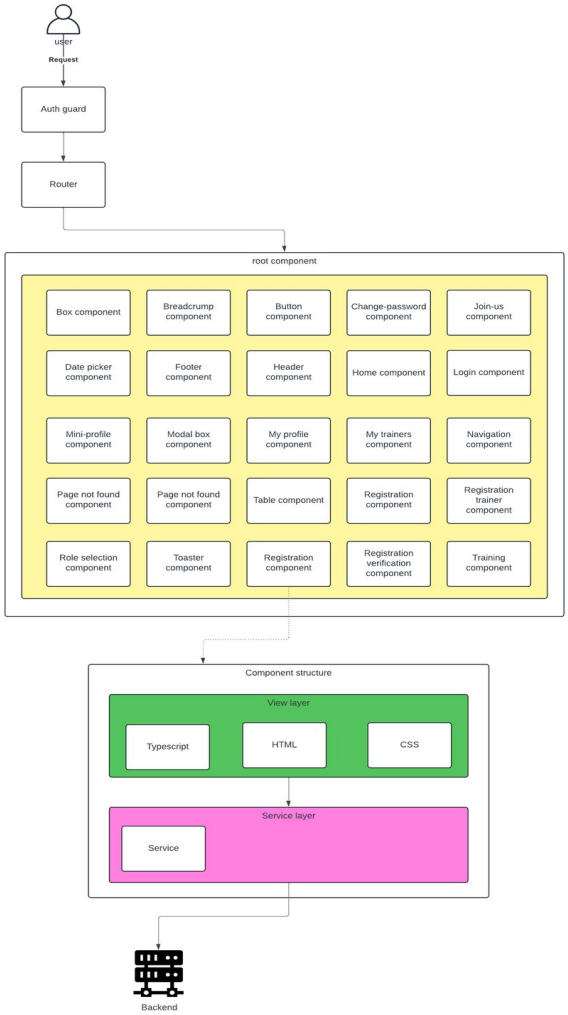


Рисунок 3.2 – Діаграма взаємодії компонентів мікросервісу FE Angular

Діаграму компонентів та їх взаємодію мікросервіса Node.js показано на рисунку 3.3. Є три логічні рівні — контролер (обробники маршрутів), сервіс (робить запити до репозиторія та на Java мікросервіс), репозиторій (json моделі Mongoose, що записують у Mongo DB).

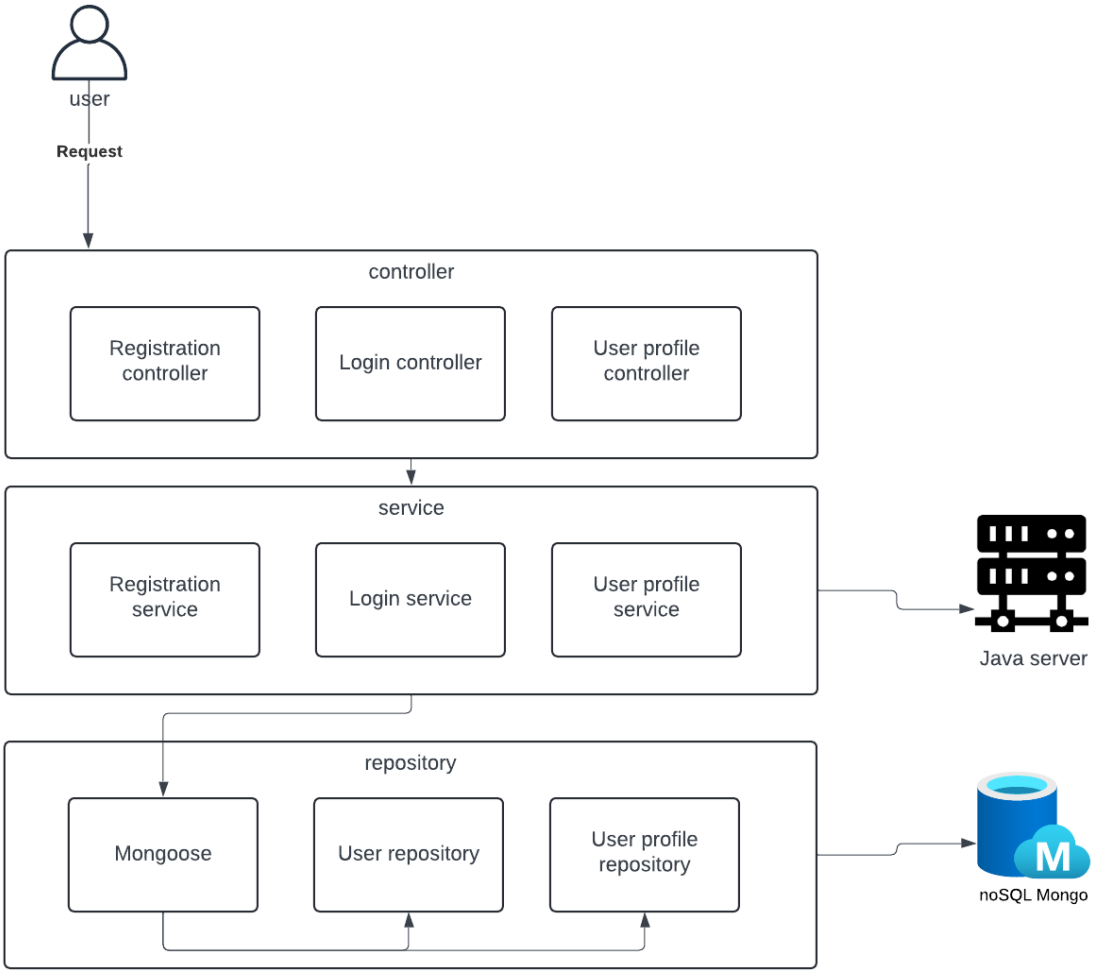


Рисунок 3.3 – Діаграма взаємодії компонентів мікросервісу Node.js

Діаграму компонентів та їх взаємодію мікросервіса Java/Spring показано на рисунку 3.4. Серверну частину розділено на 4 логічні рівні. Контролер приймає усі запити клієнта і надсилає в обробку бізнес-логіці. Шар бізнес-логіки містить в собі сервіси, що опрацьовують запит і взаємодіють з persistence шаром. Останній в свою чергу робить запити до бази даних. Клієнту повертається json, який обробляє frontend частина застосунку.

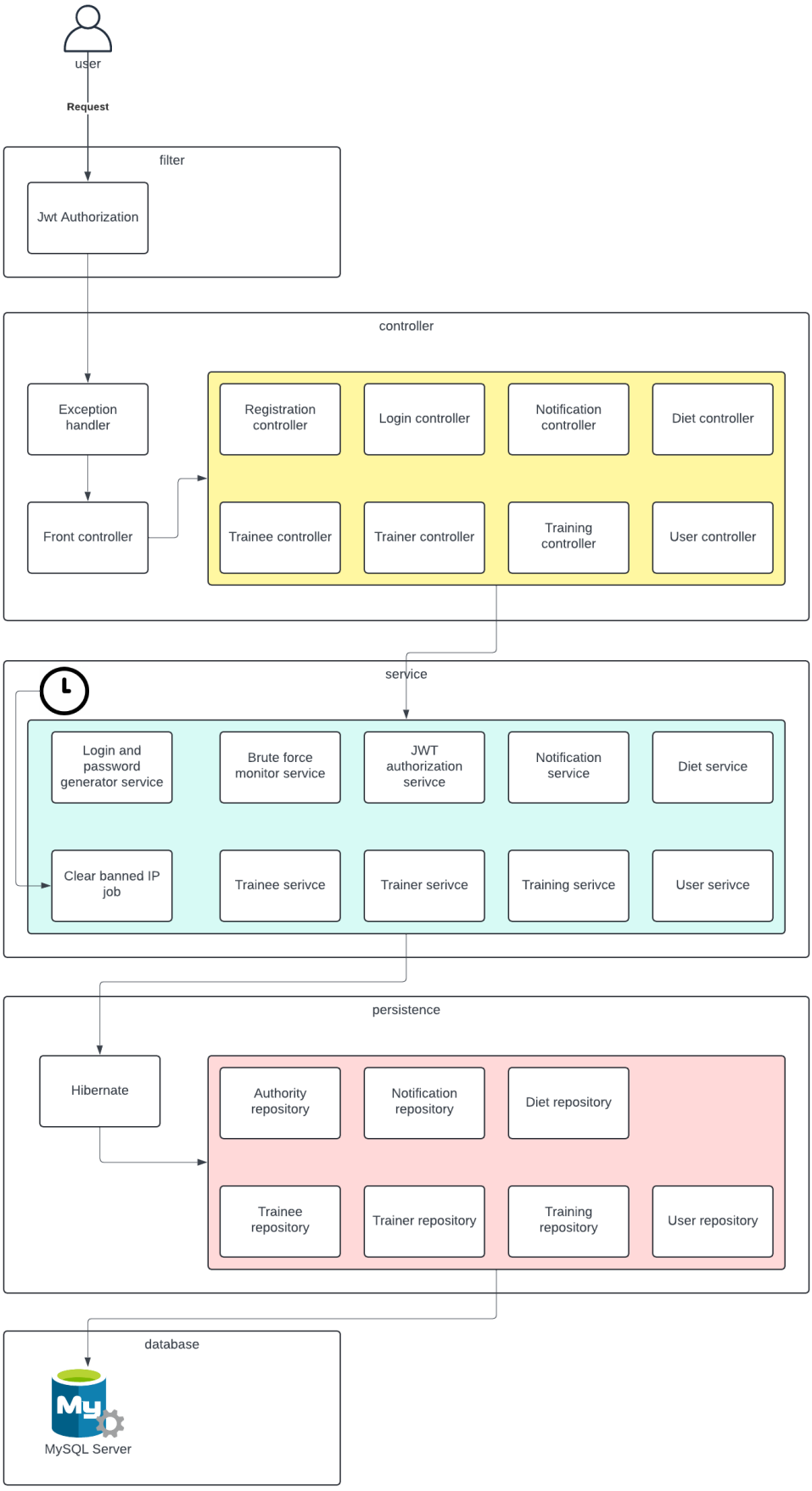


Рисунок 3.4 – Діаграма взаємодії компонентів мікросервісу Java/Spring

3.2 Обґрунтування вибору засобів розробки

Мови програмування можна розмежувати на дві категорії: мови високого рівня та низького. Низькорівневі мови (Assembler, C, C++) напряму працюють з залізом комп’ютера, натомість високорівневі мови передбачають приховання цієї реалізації і надання зручного API користувачеві (Java, C#) [14]. Java є однією з найпопулярніших мов високого рівня, є кросплатформеною і має велике ком’юніті.

Дипломний проєкт є веб-застосунком, і не передбачає роботи з залізом, прямого доступу до пам’яті. Java є однією з найпопулярніших мов високого рівня. Реалізація її віртуальної машини є на більшості десктопних операційних систем, тобто є кросплатформеною. Мова підтримує багатопоточність і має велику кількість бібліотек, зокрема для реалізації веб-застосунку [14].

Отже, використано високорівневу мову програмування Java та парадигму ООП, де об’єкти у java відповідають об’єктам у реальному житті.

Таким чином розробку основного мікросервісу проведено на мові програмування Java 11, що має довготривалу підтримку.

У якості інтегрованого середовища розробки обрано Intelij Idea, адже вона має багато вбудованих сервісів та плагінів, зокрема Tomcat та JPA builder, що їх використано в роботі [17].

Для реалізації запитів до API було використано фреймворк Spring [25-26]. За своєю суттю — це контейнер для ін’єкцій залежностей із парою зручних рівнів (доступ до бази даних, проксі). Цей фреймфорк має багато дочірніх, зокрема Hybernate [4, 27], що використовується для мапінгу об’єктів з бази даних в java сутності (ORM). Це спрощує процес написання коду, тому його і обрано.

Hybernate – це фреймворк у Java, який постачається з рівнем абстракції та обробляє реалізацію внутрішньо. Він дозволяє взаємодіяти з рівнем бази даних не напряму, а через рівень persistence, що надається цим фреймворком. Підтримує діалект MySQL та більшості інших СУБД. Реалізації включають такі завдання, як написання запиту для операцій створення, отримання, оновлення

чи видалення інформації з бази даних. Він дозволяє будувати відношення між об’єктами сутностей БД. Таким чином екземпляри цих класів можуть бути полями інших сутностей, а діаграма не містить таблиць багато-до-багатьох.

Для фронтенду обрано Angular [15]. Адже він є одним з найвживаніших фреймворків для розробки клієнтської частини веб-застосунків. Він надає розробникам структуроване середовище для побудови односторінкових застосунків з великою кількістю компонентів. Однією з його переваг є механізм двостороннього зв'язку даних, який автоматизовано оновлює інтерфейс при зміні даних на сервері. Крім того, Angular забезпечує велику кількість вбудованих функціональностей, таких як маршрутизація, форми, аутентифікація тощо, що робить його привабливим для розробки великих та складних застосунків.

Middleware/API було створено за допомогою Node.js, який є потужною серверною платформою на базі JavaScript [16]. Характерною рисою Node.js є його асинхронність, що дозволяє ефективно обробляти численні запити одночасно без зупинки виконання коду. Така особливість робить Node.js відмінним варіантом для розробки сучасних веб-серверів, які потребують високої продуктивності та можливості масштабування. Додатково, Node.js пропонує широкий асортимент модулів та пакетів, що полегшує розробку та надає можливості використання різних інструментів для вирішення специфічних задач.

У якості бази даних обрано MySQL. Адже вона багатоплатформена, масштабована (реплікація чи кластеризація даних), безпечна (передавання даних за допомогою протоколу SSL) та має велике коло користувачів.

Lombok – це бібліотека Java на основі анотацій, яка дозволяє скоротити шаблонний код. Lombok пропонує різні анотації, спрямовані на заміну коду Java, який добре відомий як шаблонний, повторюваний або виснажливий для написання [28]. За допомогою цієї бібліотеки реалізовано паттерн будівельник [29].

Maven – популярний інструмент збірки з відкритим вихідним кодом, розроблений для створення, публікації та розгортання проектів [30]. Цей інструмент дозволяє розробникам створювати та документувати структуру життєвого циклу.

Логування є важливим аспектом розробки програмного забезпечення. Це Допомагає усунути помилки роботи системи, покращує моніторинг проектів у виробничих середовищах та полегшує налагодження [31]. Log4j є частиною проекту Apache Logging Services – проекту з відкритим кодом у рамках Apache Software Foundation. Він включає кілька варіантів фреймворку журналювання Log4j для різних розгортань програмування та випадків використання. На рисунку 3.5 показано логування подій системи при авторизації користувача. У логах не фігурує ані пароль користувача, ані його хеш.

```
2023-01-26T13:33:11.971208+00:00 heroku[router]: at=info method=GET path="/login" host=practicum-estimation.herokuapp.com request_id=a0bd93d4-ff17-421f-bc9d-
3b9ae872407b fwd="00.000.00.00" dyno=web.1 connect=0ms service=96ms status=200 bytes=1083 protocol=https
2023-01-26T13:34:24.130721+00:00 heroku[router]: at=info method=POST path="/login" host=practicum-estimation.herokuapp.com request_id=55fbd040-1fec-4c6c-bec0-
f92ad52c07b8 fwd="00.000.00.00" dyno=web.1 connect=0ms service=690ms status=302 bytes=477 protocol=https
2023-01-26T13:34:24.037198+00:00 app[web.1]: 2023-01-26 13:34:24.036  INFO 4 --- [io-40989-exec-3] c.k.k.p.service.impl.UserServiceImpl     : logged in:
UserEntity(id=3, username=dima, name=dima, surname=goncharenko, email=gon@dima.com, birthday=2002-10-15, role=RoleEntity(id=1, authority=ROLE_STUDENT),
group=GroupEntity(id=1, description=IT-01))
```

Рисунок 3.5 – Логування бібліотеки Log4j

3.3 Конструювання програмного забезпечення

Побудовано діаграму відношення сутностей рівня persistence для бази даних (див. рисунок 3.6).

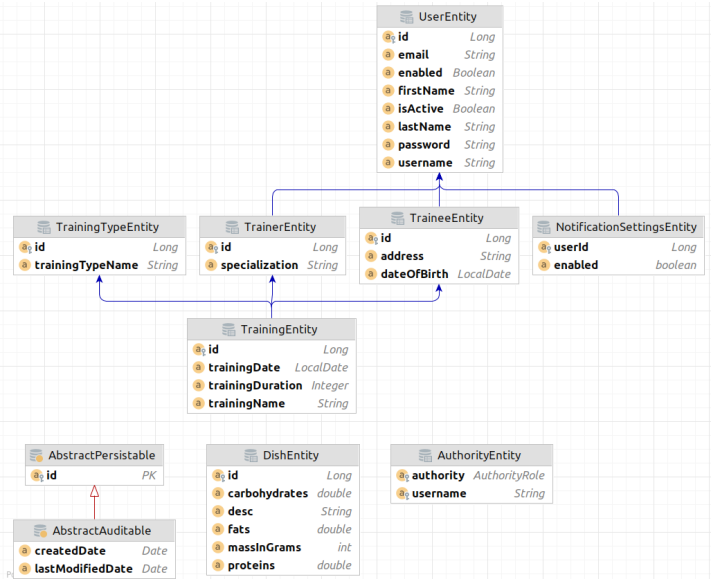


Рисунок 3.6 – Діаграма відношення persistence сутностей

В якості системи управління базами даних використовується MySql. База даних серверу призначена для зберігання користувачів, а також даних про їх тренування, дієту, налаштування сповіщень. Опис таблиць бази даних наведено у таблицях 3.1 — 3.8. Схема бази даних наведена у графічному матеріалі.

Таблиця 3.1 – Опис таблиці trainers

Таблиця	Назва поля	Тип даних	Опис
trainers	id	int	Ідентифікаційний номер користувача
	specialization	varchar	Спеціалізація тренера
	user_id	int	Ід користувача

Таблиця 3.2 – Опис таблиці trainees

Таблиця	Назва поля	Тип даних	Опис
trainees	id	int	Ідентифікаційний номер тренера
	address	varchar	адреса
	date_of_birth	date	День народження
	user_id	int	Ід користувача

Таблиця 3.3 – Опис таблиці authorites

Таблиця	Назва поля	Тип даних	Опис
authorites	username	varchar	ім’я
	authority	varchar	Привілеї ролі

Таблиця 3.4 – Опис таблиці users

Таблиця	Назва поля	Тип даних	Опис
users	id	int	Номер користувача
	username	varchar	Юзернейм користувача
	first_name	varchar	ім’я
	is_active	boolean	Акаунт не заблокований
	enabled	boolean	Чи увімкнений акаунт
	last_name	varchar	прізвище
	password	varchar	Пароль (зашифрований)
	email	varchar	Електронна пошта

Таблиця 3.5 – Опис таблиці trainings

Таблиця	Назва поля	Тип даних	Опис
trainings	id	int	Ідентифікаційний номер тренінгу
	training_date	date	Дата проведення заняття
	training_duration	int	Тривалість заняття
	training_name	varchar	Назва заняття
	trainee_id	int	Ід клієнта
	trainer_id	int	Ід тренера
	training_type_id	int	Ід типу тренування
	visited	boolean	Відвідуваність

Таблиця 3.6 – Опис таблиці training_types

Таблиця	Назва поля	Тип даних	Опис
training_types	id	int	Ідентифікаційний номер типу тренування
	training_type_name	varchar	Назва типу

Таблиця 3.7 – Опис таблиці notification_settings

Таблиця	Назва поля	Тип даних	Опис
notification_settings	user_id	int	Ідентифікаційний номер користувача
	enabled	boolean	Включені чи вимкнені сповіщення

Таблиця 3.8 – Опис таблиці dish

Таблиця	Назва поля	Тип даних	Опис
dish	id	int	Ідентифікаційний номер страви
	mass_in_grams	int	Маса страви
	description	varchar	Назва страви
	proteins	float	Коефіцієнт білків
	fats	float	Коефіцієнт жирів
	carbohydrates	float	Коефіцієнт вуглеводів

У таблицях 3.9 — 3.11 наведено призначення класів мікросервісів Angular, Node.js та Java/Spring.

Таблиця 3.9 — Призначення класів Angular

AuthEffects, AuthService	Відповідає за стан авторизації. Потрібен для відокремлення відображення різним типам користувачів (анонім, клієнт чи тренер)
AuthGuardLoggedIn Component	Відповідає за доступи до маршурітв авторизованим користувачам
BoxComponent	Компонент Box
ButtonComponent	Компонент Кнопка
ChangePasswordComponent	Компонент для зміни паролю
DatePickerComponent	Компонент для вибору дати
DietComponent	Компонент для вказання вподобань і створення дієти
DietService	Сервіс, що надсилає запит на бекенд і парсить отримані дані
FooterComponent	Компонент Footer
HeaderComponent	Компонент Header
HomeComponent	Компонент Домашня сторінка
LoginComponent	Компонент Входу в акаунт
MiniProfileComponent	Компонент міні профіль

Продовження таблиці 3.9

ModalBoxComponent, DialogUpdatePhoto, DialogAddTrainer	Модальне вікно (абстрактна сутність). Наявні імплементації для видалення акаунту та додавання тренерів.
MyProfileService	Сервіс, що отримує інформацію про поточного користувача, дозволяє змінити дані та надіслати на бекенд
MyTrainersComponent	Компонент мої тернери
NavigationComponent	Компонент навігації
NotificationSettings Component	Компонент налаштування сповіщень. Дозволяє вказати налаштування та зберегти їх
NotificationService	Сервіс налаштування сповіщень
PageNotFoundComponent	Компонент сторінка не знайдена
RegistrationStudent Component	Компонент реєстрації як клієнт
StudentPageComponent, StudentEffects, Student, StudentRegistrationState, StudentService	Компонет студент. Ефекти для збереження стану студента та сервіс
ToasterComponent	Компонент Toaster
TrainingComponent, TrainingService	Компонент та сервіс для отримання і відображення тренінгів

Таблиця 3.10 — Призначення класів Node.js

Registration Controller	Відповідає за реєстрацію. Надсилає запит на Java/Spring
Login Controller	Відповідає за авторизацію. Надсилає запит на Java/Spring лише якщо знаїодитьзнаходить користувача у своїй базі
Profile Controller	Дозволяє оновити профіль користувача
Token Service	Цей сервіс має звіт JWT токен, відмінний від Java сервіса. Усі запити до Java сервіса йдуть через Token Service
Mongoose	Репозиторій для mongo db. Зберігає дані користувачів та їх фото профілі
Crypt Serivce	Шифрує паролі користувачів (згортки паролів у базах mongo та mysql відрізнаються)

Таблиця 3.11 — Призначення класів Java/Spring

MethodSecurityConfig	Конфігурація увімкнути MethodSecurityConfig
ScheduleConfig	Конфігурація увімкнути розклад
WebSecurityConfig	Конфігурація веб секюріті
AccessUtil	Допоміжний клас для контролю доступів

Продовження таблиці 3.11

DietController	Контролер для взаємодії сутності дієт
LoginController	Контролер для авторизації
NotificationController	Контролер для взаємодії сутності повідомлень
RegistrationController	Контролер для реєстрації різних ролей
ReportController	Контролер для взаємодії сутності звітів
TraineeController	Контролер для взаємодії сутності клієнт
TrainerController	Контролер для взаємодії сутності тренер
TrainingController	Контролер для взаємодії сутності занять
UserController	Контролер для взаємодії сутності користувач
ChangePasswordDto	Клас зміни паролю
CreateDietDto	Клас створення дієти
DishDto	Клас створення страви
MealDto	Клас створення раціону
TraineeDto	Клас створення клієнта
TrainerDto	Клас створення тренера
TrainingDto	Клас створення заняття
UpdateUserDto	Клас оновлення інформації про користувача

Продовження таблиці 3.11

AuthorityEntity	Клас збереження інформації про роль користувача JPA
DishEntity	Клас для взаємодії з базою даних JPA
NotificationSettingsEntity	Клас для взаємодії з базою даних JPA
TraineeEntity	Клас для взаємодії з базою даних JPA
BruteForceTryingException	Виняткова ситуація брут форс
EntityNotFound	Виняткова ситуація не знайдено сутність
NoJwtInHeadersException	Виняткова ситуація немає JWT токета
JwtAuthorizationFilter	Фільтр, що опрацьовує кожен вхідних запит, відповідає за авторизацію
AuthorizationException Handler	Хендлер виняткових ситуацій
ClearBannedIpJob	Сервіс, що запускається за розкладом, очищує забанені ip адреси
TraineeMapper	Мапить сутності в dto класи
TrainerMapper	Мапить сутності в dto класи
TrainingMapper	Мапить сутності в dto класи
AuthorityRepository	Репозиторій для взаємодії з базою даних, спілкується через JPA класи
NotificationSettings Repository	Репозиторій для взаємодії з базою даних, спілкується через JPA класи

Продовження таблиці 3.11

TrainingRepository	Репозиторій для взаємодії з базою даних, спілкується через JPA класи
UserRepository	Репозиторій для взаємодії з базою даних, спілкується через JPA класи
BruteForceMonitorService	Сервіс, що логує всі спроби авторизації — він визначає чи підозрілою є дія авторизації. Якщо так, то блокує доступ до сайту
DietService	Сервіс генерації дієти за вподобаннями користувача. Використовує страви з бази даних. Для генерації використовує генетичний алгоритм. Спершу проведено кілька ітерацій для генерації раціонів. Найліпший раціон обирається найменшим відхиленням кілокалорій від вказаних користувачем. При цьому співвідношення білків, жирів та вуглеводів такі, як вказав користувач
ReportService	Сервіс генерації звітів. Звіт може бути про загруженість тренера або відвідуваність клієнта
JwtAuthorizationService	Сервіс перевірки авторизації. Викликається з фільтра
LoginPasswordGenerator	Генерує тимчасові паролі при реєстрації
NotificationService	Дозволяє налаштувати сповіщення, надслає сповіщення при тригерах
TrainingService	Створює заняття

Схема структурна класів програмного забезпечення наведена у графічному матеріалі (опис ключових класів мікросервісу Angular, а також класів мікросервісу Java/Spring).

Опис утиліт, бібліотек та іншого стороннього програмного забезпечення, що використовується у розробці наведено в таблиці 3.10.

Таблиця 3.10 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	IntelliJ IDEA	Головне середовище розробки програмного забезпечення серверної частини.
2	Curl	Програмне забезпечення необхідне для тестування запитів. Використовувалось для тестування прав ролей.
3	MySQL	Вільна система керування реляційними базами даних.
4	Data Grip	Програмне забезпечення, яке надає легкий графічний інтерфейс для доступу до бази даних.
5	Log4j	Бібліотека, призначена для логування.
6	Lombok	Бібліотека, спрямована на заміну коду Java.
7	Maven	Інструмент збірки.
8	Jakarta EE persistence	Плагін для відображення persistence сутностей.
9	Bootstrap	CSS framework, що призначений для спрощення процесу створення стилів для користувацького інтерфейсу

3.4 Аналіз безпеки даних

Небезпечно зберігати паролі у відкритому вигляді. Якщо зловмисники отримають доступ до БД і там зберігатимуться паролі, то вони зможуть авторизуватися від чужого імені. Тому паролі зберігаються в зашифрованому вигляді. Для хешування паролю використовується алгоритм BCrypt [32].

BCrypt — це одностороння хеш-функція, заснована на шифрі Blowfish [33]. Це означає, що пароль можна зашифрувати, але розшифрувати неможливо.

Міжсайтовий скриптинг (XSS) - це вид атаки, при якому зловмисник внедрює шкідливі скрипти на безпечні та надійні веб-сайти. Це стає можливим тоді, коли зловмисник використовує вразливість у веб-застосунку для впровадження шкідливого коду, зазвичай у вигляді скрипту браузера, на комп'ютер користувача. Зловмисник може використовувати XSS для надсилання шкідливих скриптів користувачам, які не підозрюють нічого поганого. Браузер кінцевого користувача виконує ці скрипти, вважаючи їх надійними, тому що вони приходять з відомого джерела. В результаті шкідливий скрипт може отримати доступ до конфіденційної інформації, такої як файли cookie або токени сеансу [34].

Angular має вбудовані заходи безпеки, які допомагають запобігти атакам XSS такі як: санітизація вхідних даних, безпечне вставлення HTML, екарнування [35].

Захист від атаки брутфорсу є критично важливим аспектом в забезпеченні безпеки даних [36]. Брутфорс атака полягає у спробах зловмисників вгадати пароль, шляхом послідовної спроби всіх можливих комбінацій символів. Оскільки така атака ґрунтується на переборі всіх можливих варіантів, вона може бути дуже ефективною, особливо для слабких або недостатньо складних паролів.

Саме тому в системі додано функціонал бану користувачів, що намагаються брут форсом отримати доступ до чужого облікового запису.

Висновки до розділу

Розроблене програмне забезпечення має клієнт-серверну структуру, що складається з серверної та клієнтської частин. Серверна частина відповідає за обробку запитів та логіку роботи з даними, а клієнтська - за представлення інтерфейсу користувача та доступ до ресурсів сервера.

Вибір мікросервісної архітектури перед монолітною може бути вигідним завдяки гнучкості, масштабованості та полегшенню розвитку та супроводження застосунку. Проте ця архітектура вимагає уваги до аспектів розгортання та моніторингу.

Обгрунтовано вибір конкретних технологій, що використовуватимуться в розробці програмного забезпечення.

Для розробки серверної частини застосунку використано архітектуру мікросервісів. Мікросервіс FE Angular спілкується з мікросервісами за допомогою протоколу HTTP, мікросервіс API/Middleware Node.js виконує роль прослойки, а мікросервіс BE/Inner service Java/Spring обробляє основну бізнес логіку.

Описано класи сервісів, їхнє призначення подано за допомогою таблиці.

Для імплементації використано фреймворк Spring – контейнер для ін’єкцій залежностей із доступом до бази даних, проксі тощо.

Використано базу даних MySql, описано її структуру та сутності. Паролі не зберігаються у відкритому вигляді, а шифрують алгоритмом BCrypt.

Використано бібліотеку логування Log4j, плагін для відображення persistence сутностей Jakarta EE persistence, інструмент збірки проект Maven, бібліотеку, спрямована на заміну коду Java – Lombok.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Якість програмного забезпечення є важливою характеристикою продукту, оскільки це компонент, актуальний для кінцевого користувача, зокрема, є важливий для багатокористувацьких служб клієнт-сервер [37].

Щоб забезпечити якість одним із важливих процесів у розробці програмного забезпечення, необхідно провести його тестування. Тому він є одним із основних етапи створення програмних продуктів.

Тестування дозволяє покращити неякісний продукт на наступних етапах розгортання програмного забезпечення продукт, раннє виявлення помилок у життєвому циклі розробки [38].

Було проаналізовано програмне забезпечення за допомогою сервісу Sonar Cloud (див. рисунок 4.1), адже він має авторизацію з GitHub [39]. Для цього використані наступні метрики:

- надійність коду (баги);
- підтримуваність коду (неточності);
- безпека;
- дублювання коду;
- часовий борг програміста.

Покриття кодом всього проекту складає Java – 1.9 тисячі рядків. Було перевірено лише покриття частини застосунку Java. Результати аналізу наведено на рисунках 4.1 – 4.2. Приклад вирішення неточності – пропозиція визначити константу замість дублювання слова кілька разів.

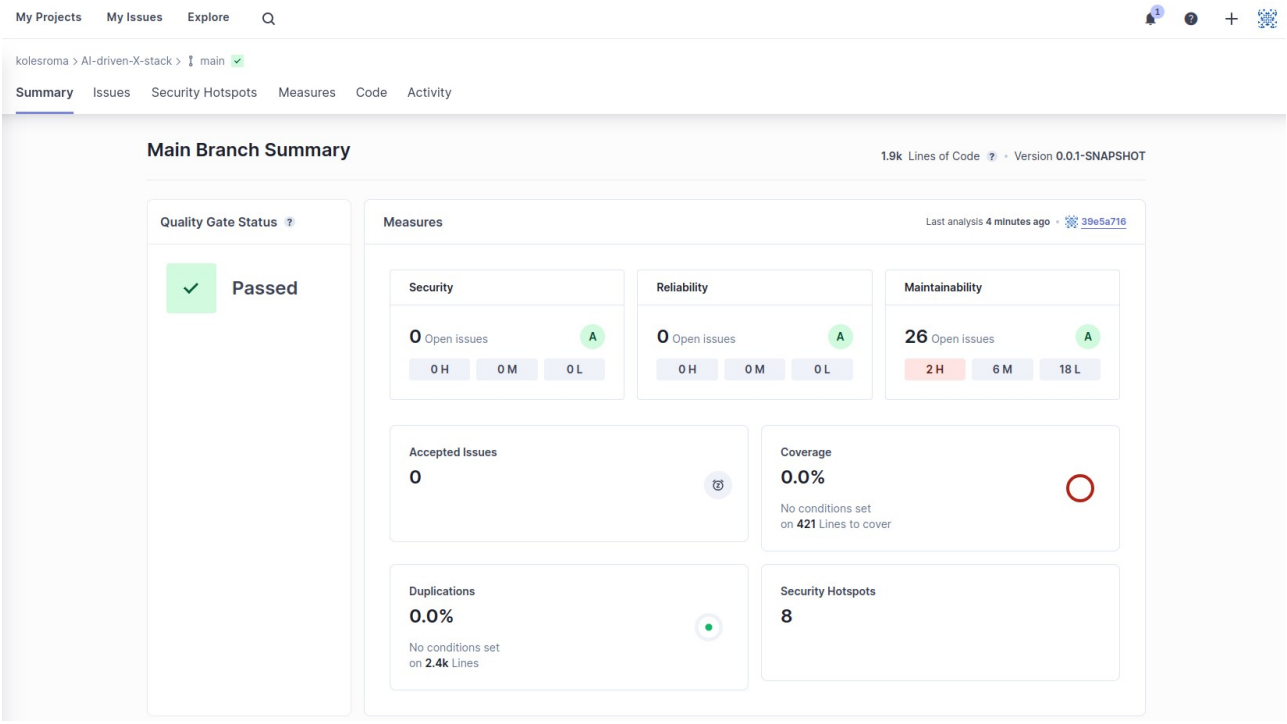


Рисунок 4.1 – Результат аналізу якості ПЗ у Sonar [40]

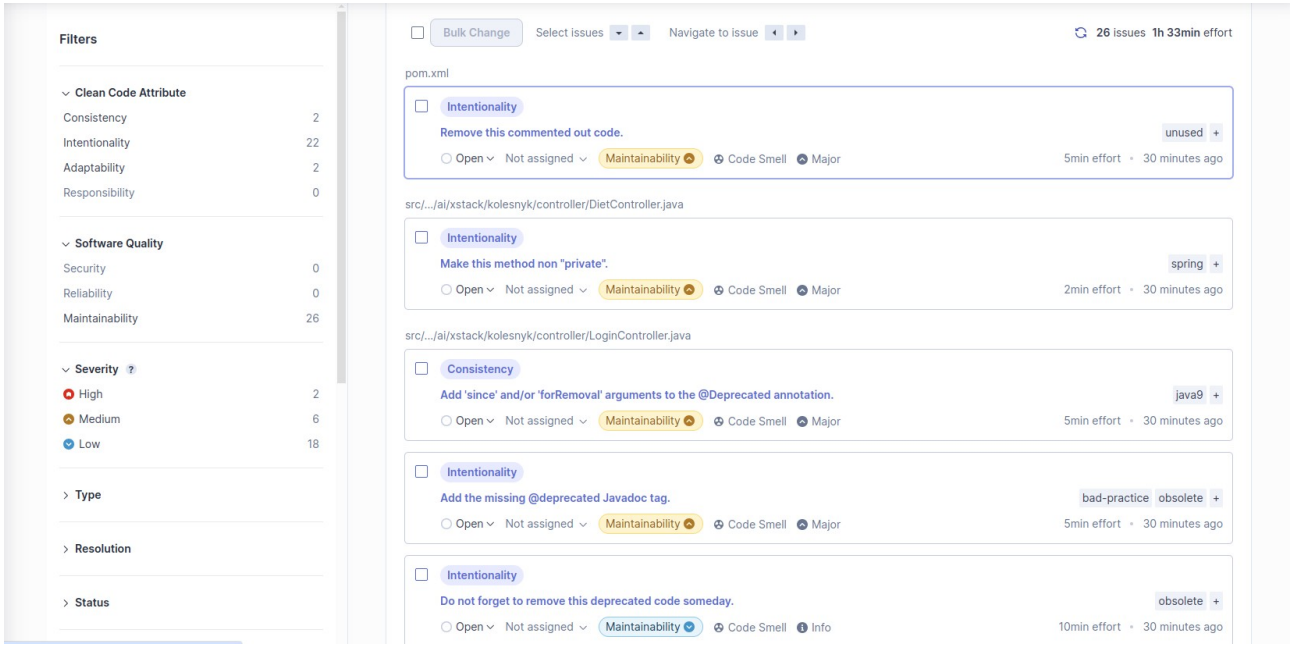


Рисунок 4.2 – Таблиця критеріїв Java коду

На рисунку 4.3 наведено часовий борг програміста. Це час, що потрібно витратити, щоб проект не мав багів, вразливостей чи неточностей. Цей критерій свідчить про те, що код потрібно рефакторити і витратити стільки часу, скільки

дорівнює цей борг. За мірками Sonar, 93 хвилини є незначним боргом для проекту такого обсягу.

Не було виявлено критичних помилок.

Рисунок 4.3 – Часовий борг програміста

На рисунку 4.4 наведено процес перевірки у Sonar Cloud.

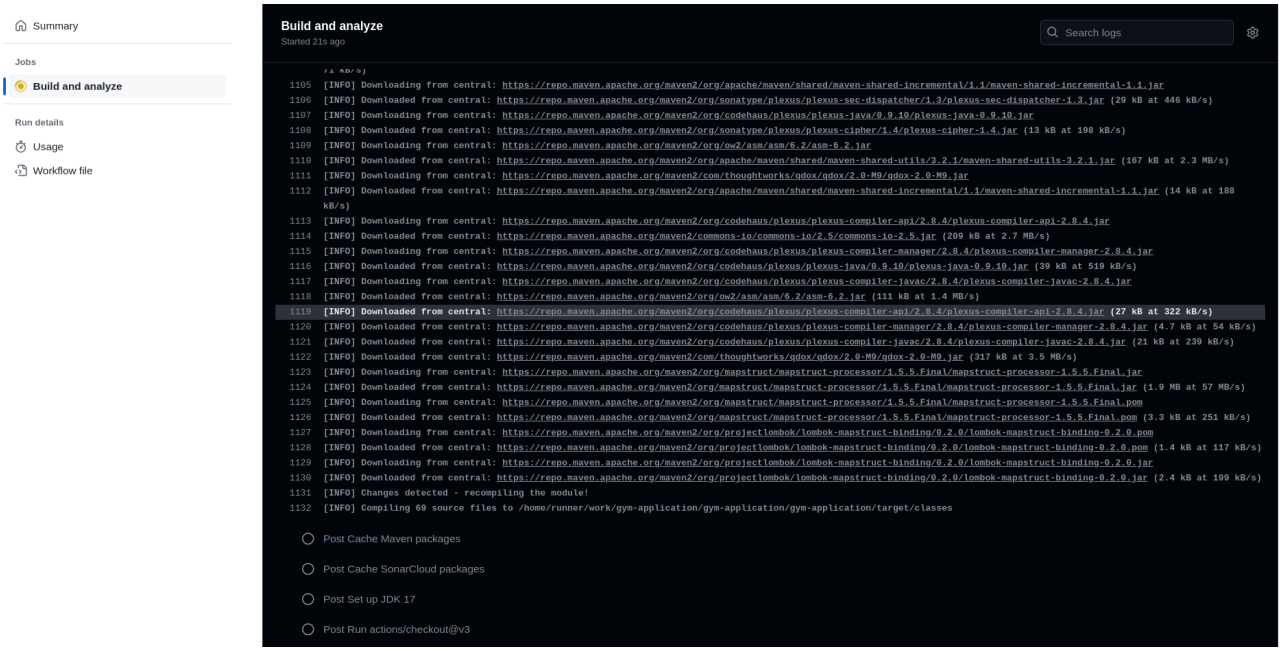


Рисунок 4.4 – Пайплайн перевірки проекту у Sonar Cloud

4.2 Опис процесів тестування

За мету поставлено протестувати ключові особливості програмного забезпечення. Саме тому було виконане мануальне інтеграційне тестування програмного забезпечення через UI, опис відповідних тестів наведено у таблицях 4.1 – 4.8.

Таблиця 4.1 – Тест 1

Мета тесту	Перевірка можливості аутенифікації
Початковий стан	Неавторизований користувач переходить на сторінку входу чи будь-яку сторінку сервісу і перенаправляється на сторінку входу
Вхідні дані	Логін та пароль зареєстрованого користувача
Схема проведення	Ввід даних та надсилання форми
Результат, що очікується	Успішна аутентифікація

Таблиця 4.2 – Тест 2

Мета тесту	Перевірка можливості запису на тренінг
Початковий стан	Користувач, що є лієнтом, переходить на сторінку запису
Вхідні дані	Дата тренінгу
Схема проведення	Надсилання даних на сервер
Результат, що очікується	Отримання тренінгу

Таблиця 4.3 – Тест 3

Мета тесту	Перевірка можливості отримання сповіщення
Початковий стан	Користувач налаштував відповідні опції сповіщень
Вхідні дані	Налаштування сповіщень
Схема проведення	Відправити тестове сповіщення (наприклад при записі на тренінг)
Результат, що очікується	Сповіщення успішно отримано

Таблиця 4.4 – Тест 4

Мета тесту	Перевірка можливості налаштування сповіщень
Початковий стан	Вимкнути сповіщення, що при записі на тренінг його більше не буде отримано
Вхідні дані	Вибір типів отримуваних сповіщень
Схема проведення	Зберегти налаштування сповіщень
Результат, що очікується	Налаштування успішно збережено, а при записі на тренінг більше не отримано сповіщення

Таблиця 4.5 – Тест 5

Мета тесту	Перевірка можливості генерації дієти
Початковий стан	Користувач увійшов у систему і вказав подобання
Вхідні дані	Відсоткове співвідношення білків, жирів та вуглеводів
Схема проведення	Відправити запит на генерацію дієти
Результат, що очікується	Отримано результат і його можна зберегти

Таблиця 4.6 – Тест 6

Мета тесту	Перевірка адміністрування акаунту
Початковий стан	Користувач увійшов у свій обліковий запис
Вхідні дані	Новий пароль користувача
Схема проведення	Внести зміни у профіль користувача
Результат, що очікується	Пароль успішно змінено

Таблиця 4.7 – Тест 7

Мета тесту	Перевірка можливості генерації звітів
Початковий стан	Тренер увійшов у систему
Вхідні дані	Параметри для звіту (обрати тип)
Схема проведення	Відправити запит на генерацію звіту
Результат, що очікується	Звіт згенеровано на сервері та його можна завантажити

Таблиця 4.8 – Тест 8

Мета тесту	Перевірка можливості вказань подобань в дієті
Початковий стан	Стандартні коефіцієнти страв, але вони можуть бути встановлені користувачем
Вхідні дані	Відсоткове співвідношення білків, жирів та вуглеводів
Схема проведення	Надіслати вибір харчування і підібрати дієту, що задовольняє критерії харчування
Результат, що очікується	Вподобання враховано і дієта покриває необхідні кілокалорії

4.3 Опис контрольного прикладу

У якості контрольного прикладу буде наведено процес генерації дієти (див. рисунок 4.5). Наведено початковий вигляд при заході на сторінку дієти. Кнопка завантаження недоступна, поки не згенерувати дієту.

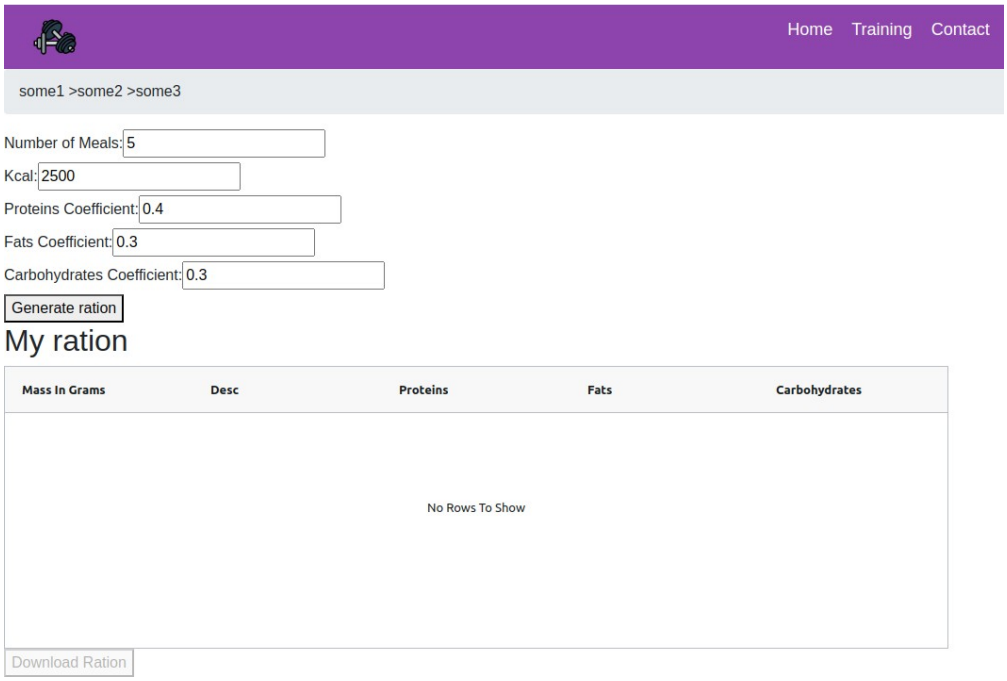


Рисунок 4.5 – Сторінка генерування дієти

Наявний захист від надмірних обрахунків, якщо ввести надто велику кількість кілокалорій. Тоді форму не можна надіслати. Так же, коли сума коефіцієнтів не дорівнює 1 (див. рисунок 4.6).

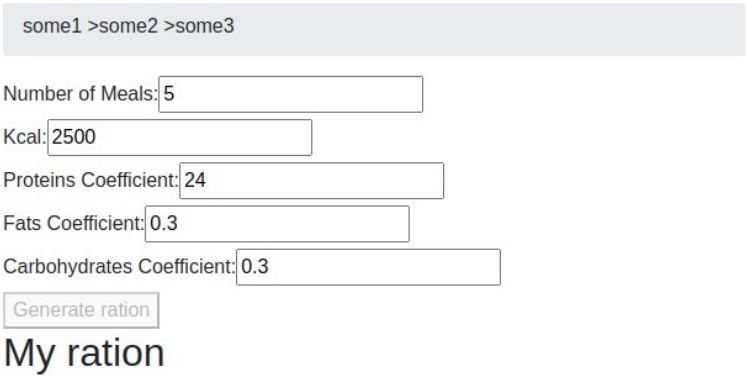


Рисунок 4.6 – Кнопка підтвердження стає недоступною

Коли натиснуто згенерувати дієту, отримано відповідь від сервера раціону на перший день. Кнопка завантаження стає доступною (див. рисунок 4.7).

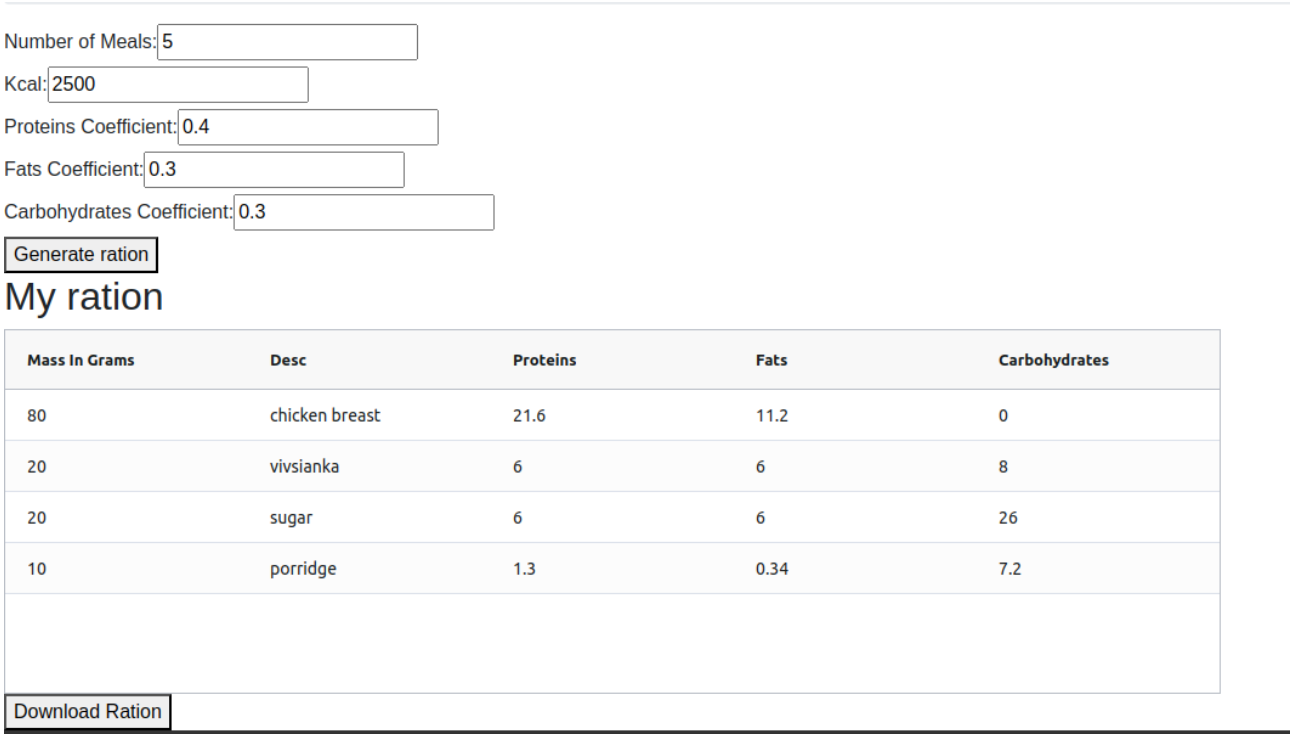


Рисунок 4.7 – Раціон харчування на перший день

Збереження раціону локально наведено на рисунку 4.8. Збережену дієту тренер роздруковує і надає клієнту.

Рисунок 4.8 – Збереження дієти

Висновки до розділу

Проведено повний цикл для аналізу програмного продукту, цілісний аналіз предметної області, вимог до програмного продукту та їх ролі в проекті.

Таблично описані контрольні приклади, що містять мету тесту, початковий стан, вхідні дані (повинні бути валідними), схему проведення та очікуваний результат.

Описано протестовані програмні об’єкти, тип тестів, ресурси та план необхідний для виконання тестування.

У рамках цього плану виконано тестування частини продукту, що відповідає за основну логіку сайту: Перевірка можливості отримання сповіщення, перевірка можливості налаштування сповіщень, перевірка можливості генерації дієти, перевірка адміністрування акаунту, перевірка можливості генерації звітів, перевірка можливості вказань подобань в дієті.

Наведено контрольний приклад процесу генерації дієти. Показані та протестовані можливі варіанти дій користувача.

Програмне забезпечення було проаналізовано за допомогою сервісу Sonar Cloud. Унаслідок перевірки не було знайдено критичних багів та не було виявлено дублікацій коду. Запропоновано внести деякі змінні, що покращать ідтримку продукту.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Вибір AWS для розгортання архітектури має кілька обґрунтованих причин [41]. По-перше, AWS є одним з трьох найбільших хмарних провайдерів, що забезпечує великий вибір сервісів для будь-яких потреб (Google Cloud, Azure) [42]. Також, AWS надає можливість розгорнути всі мікросервіси та бази даних, що робить його відмінним варіантом для проекту.

ДевОпс може налаштувати для автоматичне розгортання, що дозволить швидко виконувати оновлення без необхідності ручних дій [43]. Це спростить роботу команди та забезпечить швидкий розвиток продукту.

Оскільки система складається з різних мікросервісів, необхідно регулярно виконувати технічне обслуговування та оновлення. AWS дозволяє виділити час для цих операцій, а також забезпечує можливість швидко виконати розгортання всіх змінних компонентів системи.

У архітектурі використані різні сервіси AWS для різних складових системи, що дозволяє максимально оптимізувати роботу кожного компонента. Frontend частина розгорнута за допомогою S3 та CloudFront, що забезпечує швидке та надійне розповсюдження статичного контенту [44].

Node.js та Java застосунки розгорнуті за допомогою EC2, що дозволяє отримати повний контроль над серверами, на яких вони працюють [45].

Бази даних MongoDB та MySQL розгорнуті відповідно за допомогою DocumentDB та Amazon RDS, що надає повну готовність та масштабованість баз даних.

Всі ці компоненти об'єднані в одну Virtual Private Cloud, що забезпечує ізольоване середовище для роботи системи та максимальний рівень безпеки (див. рисунок 5.1) [46]. Такий підхід дозволить вам створити надійний та масштабований продукт на основі AWS.

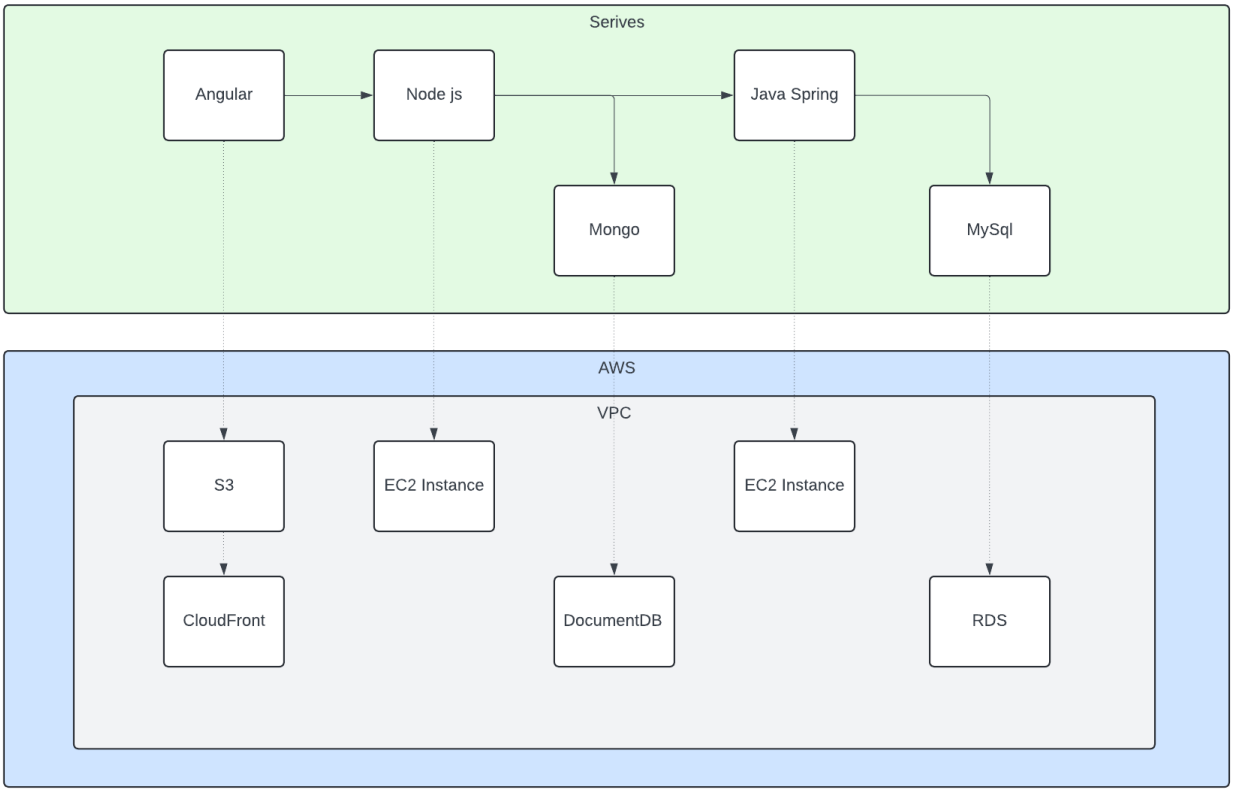


Рисунок 5.1 – Розгортання сервісів у VPC AWS

AWS CodePipeline дозволяє автоматизувати процеси поставки програмного забезпечення. Є змога налаштувати пайплайн для автоматичного розгортання коду на AWS, і виконання тестів (якщо потрібно). Після кожного пушу в репозиторій, CodePipeline автоматично запустить пайплайн, що призведе до автоматичного оновлення додаткової інфраструктури в AWS [47].

Отже, розгортання починається коли новий код застосунку доставляється у репозиторій у гілку main.

Наведено приклад мануального розгортання застосунку Angular в S3 та CloudFront. Для цього потрібно збудувати застосунок та загрузити на S3 Bucket (див. рисунок 5.2).

```
ngroman@laptop:~/IdeaProjects/AI-driven-X-stack/angular$ ng build
.: Generating browser application bundles (phase: building)...
```

Рисунок 5.2 – Будування артефакту застосунку Angular

Спершу потрібно налаштувати правила для отримання об’єктів з S3 бакета (див. рисунок 5.3).

Рисунок 5.3 – Налаштування правил S3 на читання

Загрузка файлів у бакет наведена на риснку 5.4.

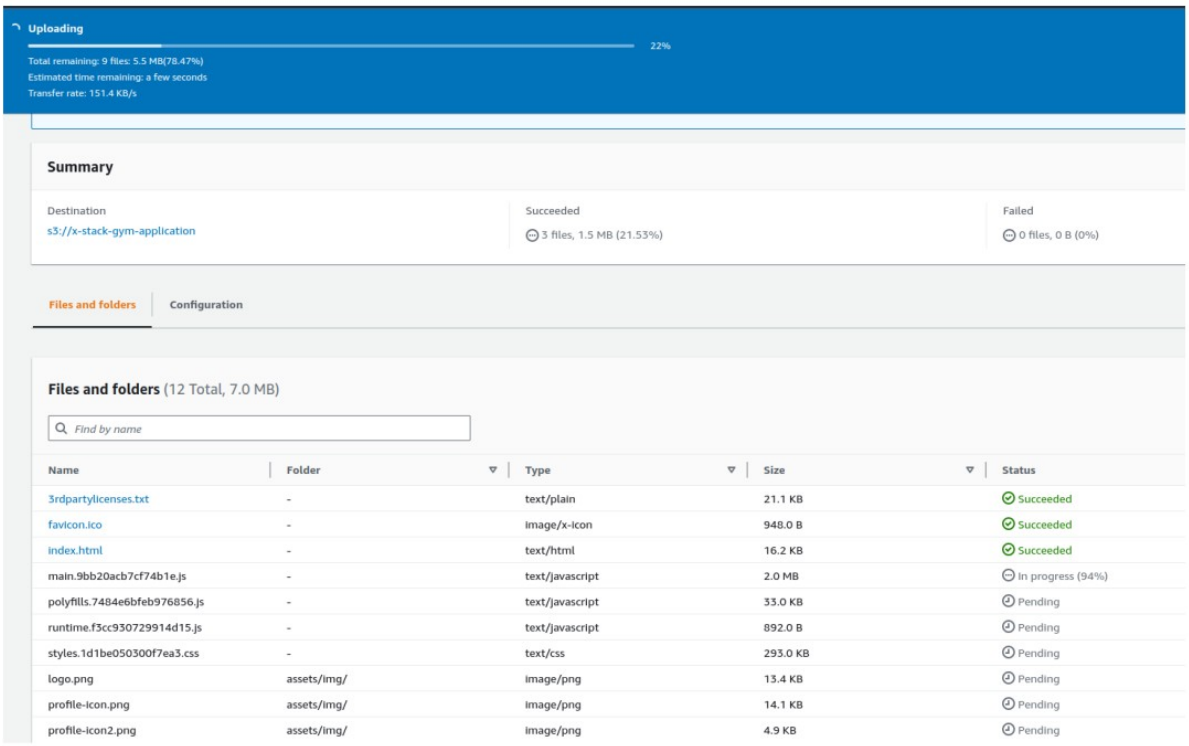


Рисунок 5.4 – Загрузка в S3 бакет

Для доступу користувачам, потрібно завантажити дані з бакета на CloudFront (див. рисунок 5.5).

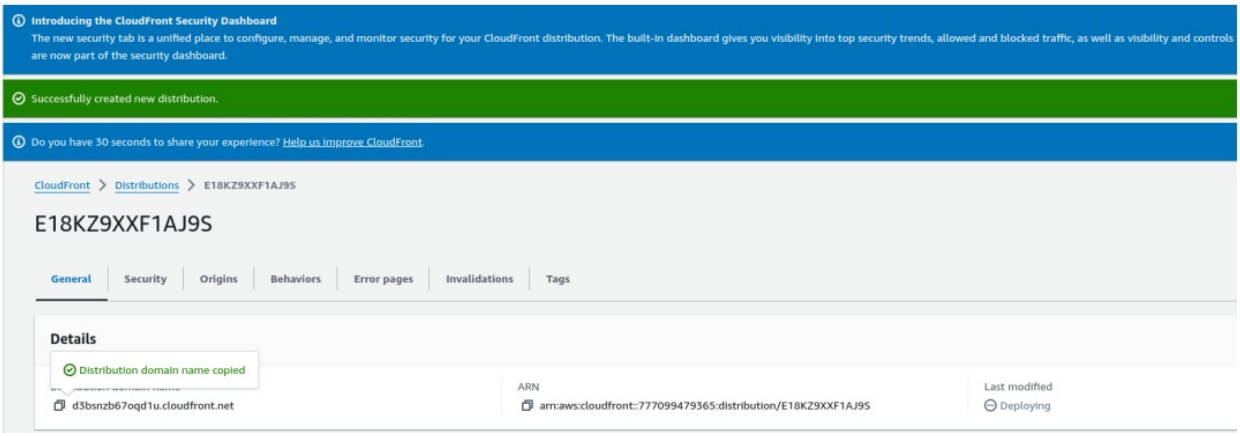


Рисунок 5.5 – Налаштування CloudFront

Тепер за посиланням користувачі мають доступ до сайту (див. рисунок 5.6).

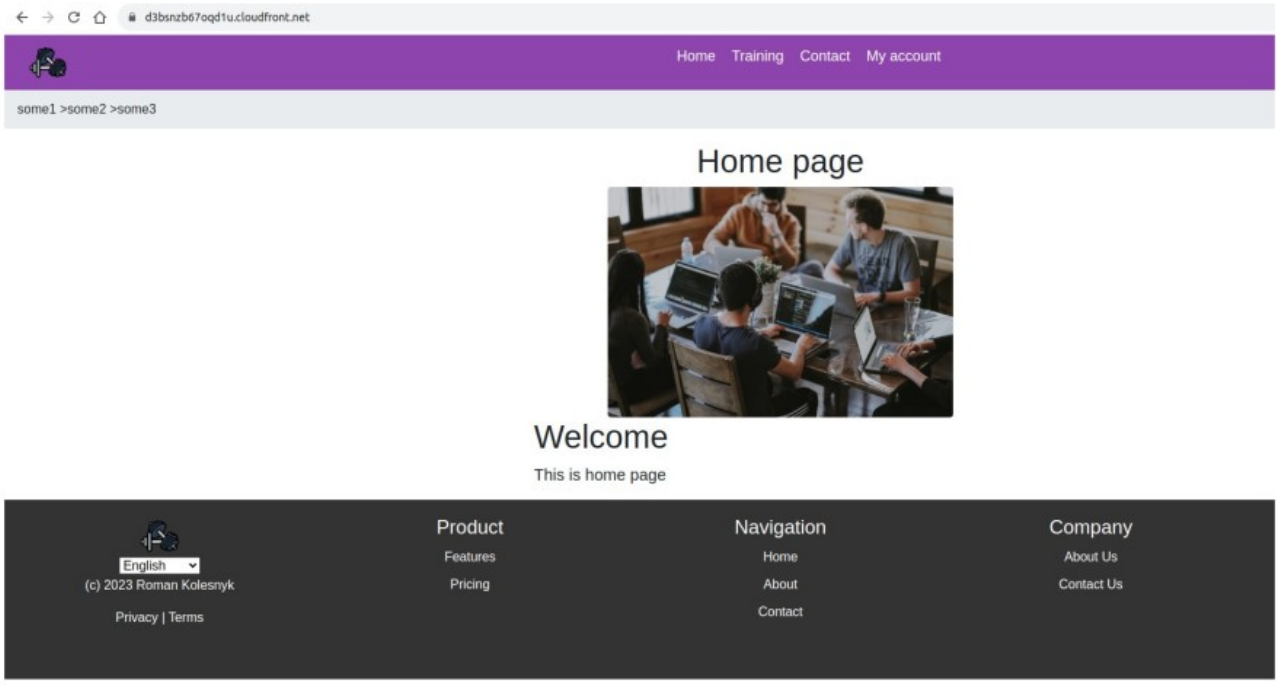


Рисунок 5.6 – Доступ у CloudFront

5.2 Супровід програмного забезпечення

Проект має два декілька середовищ — dev (середовище, де відбуваються зміни та проводиться тестування) та prod (середовище, куди кінцевому користувачеві доставляються протестовані зміни) [48]. Він автоматично розгортається лише на середовище dev. Для середовища prod його повинен мануально розгорнути девОпс інженер після того, як ПЗ буде протестовано.

Для середовища dev створено тригер на push чи pull request в гілку main [49]. Щоразу коли це відбувається, проект будується і розгортається на AWS за допомогою CodePipeline. Це дозволяє синхронізувати розгортання мікросервісів, адже робота відбувається одразу у трьох різних репозиторіях.

Висновки до розділу

Користувачі повинні мати можливість отримати актуальну версію продукту. Розгорнуто проект на хостинг за допомогою сервісу AWS. Сплановано тригери на поступ в гілку main. При push та pull request у main проект автоматично будується та нова версія розгортається у хмарі.

Налаштування автоматичного розгортання за допомогою DevOps інструментів дозволяє швидко впроваджувати оновлення без необхідності ручних дій. Це спрощує роботу команди та забезпечує ефективний розвиток продукту.

Функціонування всіх сервісів всередині однієї AWS VPC має декілька переваг. Безпека - усі ресурси в межах однієї VPC знаходяться в одній логічній мережі, що дозволяє керувати доступом та налаштуваннями безпеки на рівні мережі. Ізоляція - використання однієї VPC дозволяє створювати ізольовані середовища для системи, що забезпечує легше керування та управління ресурсами. Ефективність мережі - використання однієї VPC дозволяє оптимізувати використання ресурсів мережі (є можливість визначення правил маршрутизації). Простота управління — використання однієї консолі для керування всіма сервісами. Це також полегшує моніторинг.

ВИСНОВКИ

Проведено передпроєктне обстеження предметної області, а також розглянуто відомі алгоритмічні рішення та програмні продукти-аналоги.

Сформульовано призначення, цілі та мету розробки програмного забезпечення в даному розділі, що надає зрозуміле та чітке уявлення про напрямки подальшої роботи.

Розглянуто варіанти використання з двома основними акторами — тренер та клієнт. Основні функції включають в себе запис на тренування до тренера, перегляд історії тренувань, відмітку відвідуваності, налаштування вподобань в їжі, генерацію дієти, отримання та налаштування сповіщень, а також генерацію звітів для тренерів.

Данна розробка призначена для всіх учаників діяльності спортивного комплексу. Клієнтам вона дозволяє записатися на тренування та отримати дієту харчування. Тренери можуть відмічати відвідуваність та створювати звіти.

Розроблене програмне забезпечення має клієнт-серверну структуру, що складається з серверної та клієнтської частин. Серверна частина відповідає за обробку запитів та логіку роботи з даними, а клієнтська - за представлення інтерфейсу користувача та доступ до ресурсів сервера.

Для розробки серверної частини застосунку використано архітектуру мікросервісів. Мікросервіс FE Angular спілкується з мікросервісами за допомогою протоколу HTTP, мікросервіс API/Middleware Node.js виконує роль прослойки, а мікросервіс BE/Inner service Java/Spring обробляє основну бізнес логіку.

Проведено повний цикл для аналізу програмного продукту, цілісний аналіз предметної області, вимог до програмного продукту та їх ролі в проекті.

Таблично описані контрольні приклади, що містять мету тесту, початковий стан, вхідні дані (повинні бути валідними), схему проведення та очікуваний результат. Описано протестовані програмні об’єкти, тип тестів, ресурси та план необхідний для виконання тестування.

Програмне забезпечення було проаналізовано за допомогою сервісу Sonar Cloud. Унаслідок перевірки не було знайдено критичних багів та не було виявлено дублікацій коду. Запропоновано внести деякі змінні, що покращать ідтримку продукту.

Програмне забезпечення розгорнто в хмарному середовищі AWS VPC. Об'єднання всіх сервісів в одну AWS VPC має декілька переваг, таких як забезпечення безпеки, ізоляції, ефективності мережі та простоти управління. Цей підхід сприяє ефективній роботі та масштабуванню системи, забезпечуючи її надійність та безпеку.

Досягнуто поставленої мети. Реалізовано наступні задачі: авторизація, адміністрування облікового запису, можливість запису на тренування, розсилка повідомлень про події в системі, генерація дієти згідно з вподобаннями клієнта, генерація звітів про активність користувачів та загруженість тренерів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Why is software engineering important [Електронний ресурс] – Режим доступу до ресурсу: https://www.institutedata.com/blog/why-is-software-engineering-important.

2. Dave Johnson. What is software [Електронний ресурс] / Dave Johnson – Режим доступу до ресурсу: https://www.businessinsider.com/guides/tech/what-is-software.

3. Measuring API Usability [Електронний ресурс] – Режим доступу до ресурсу: https://web.archive.org/web/20220303235859/http://www.drdobbs.com/windows/measuring-api-usability/184405654.

4. Samuel Kyere Samuel Kyere. ORM [Електронний ресурс] / Samuel Kyere Samuel Kyere – Режим доступу до ресурсу: https://medium.com/@mr.kyere.s/simplifying-database-interactions-with-object-relational-mapping-orm-a27a9cbc8160.

5. OOP Criticism [Електронний ресурс] – Режим доступу до ресурсу: https://www.oocities.org/tablizer/oopbad.htm.

6. Advantages of Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.ibm.com/docs/sl/aix/7.2?topic=monitoring-advantages-java.

7. What is MySQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.oracle.com/mysql/what-is-mysql.

8. Спорткомплекс "Галкатика" [Електронний ресурс] – Режим доступу до ресурсу: https://www.galactica.com.ua/index.php.

9. Спортзал "Пластилін" [Електронний ресурс] – Режим доступу до ресурсу: https://plastylin.te.ua.

10. Мобільний застосунок "Fitness body" [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=fitness.buddy.app&hl=uk&gl=US.

11. Що таке збалансоване харчування і чому воно важливе для здоров’я [Електронний ресурс] – Режим доступу до ресурсу:

https://healthyfood.org.ua/ua/sho-take-zbalansovane-harchuvannya-i-chomu-vono-vazhlive-dlya-zdorovya.

12. Athlete Doesn’t Seem Motivated? Here Are 4 Reasons Why [Електронний ресурс] – Режим доступу до ресурсу: https://completeperformancecoaching.com/2021/09/25/athlete-doesnt-seem-motivated-4-reasons.

13. What are microservices [Електронний ресурс] – Режим доступу до ресурсу: https://www.logicmonitor.com/blog/what-are-microservices.

14. Bill Kinnersley. The Language List [Електронний ресурс] / Bill Kinnersley – Режим доступу до ресурсу: https://webcitation.org/6HZt8VX9t?url=http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm.

15. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://www.bacancytechnology.com/blog/why-use-angular.

16. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://relevant.software/blog/why-and-when-to-use-node-js.

17. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea.

18. BPMN [Електронний ресурс] – Режим доступу до ресурсу: https://www.bpmn.org.

19. Клієнт-серверна архітектура [Електронний ресурс] – Режим доступу до ресурсу: https://training.qatestlab.com/blog/technical-articles/client-server-architecture.

20. Chandler Harris. Microservices vs. monolithic architecture [Електронний ресурс] / Chandler Harris – Режим доступу до ресурсу: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith.

21. Mehmet Ozkaya. Architecture Comparison: Monolithic vs Microservices [Електронний ресурс] / Mehmet Ozkaya – Режим доступу до ресурсу: https://medium.com/design-microservices-architecture-with-patterns/architecture-comparison-monolithic-vs-microservices-4109265c4806.

22. Redux [Електронний ресурс] – Режим доступу до ресурсу: https://redux.js.org.

23. NgRx [Електронний ресурс] – Режим доступу до ресурсу: https://ngrx.io.

24. What is middleware [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-middleware.

25. Spring [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/why-spring.

26. Kumar Chandrakant. Why Choose Spring as Your Java Framework? [Електронний ресурс] / Kumar Chandrakant – Режим доступу до ресурсу: https://www.baeldung.com/spring-why-to-choose.

27. Hibernate [Електронний ресурс] – Режим доступу до ресурсу: https://hibernate.org/orm.

28. Lombok [Електронний ресурс] – Режим доступу до ресурсу: https://projectlombok.org.

29. Патерни проєктування [Електронний ресурс] – Режим доступу до ресурсу: https://refactoring.guru/uk/design-patterns.

30. Welcome to Apache Maven [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org.

31. Shubham Gupta. Application Logging and its importance [Електронний ресурс] / Shubham Gupta – Режим доступу до ресурсу: https://medium.com/ula-engineering/application-logging-and-its-importance-c9e788f898c0.

32. Bill O\'Neil. Hashing Passwords in Java With BCrypt [Електронний ресурс] / Bill O\'Neil – Режим доступу до ресурсу: https://dzone.com/articles/hashing-passwords-in-java-with-bcrypt.

33. B. Schneier. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) [Електронний ресурс] / B. Schneier – Режим доступу до ресурсу: https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html.

34. Jim Manico. Cross Site Scripting (XSS) [Електронний ресурс] / Jim Manico, Jeff Williams, Dave Wichers – Режим доступу до ресурсу: https://owasp.org/www-community/attacks/xss/.

35. Angular XSS Guide: Examples and Prevention [Електронний ресурс] – Режим доступу до ресурсу: https://www.stackhawk.com/blog/angular-xss-guide-examples-and-prevention.

36. Брутфорс: що це таке? [Електронний ресурс] – Режим доступу до ресурсу: https://ukeywaf.com/baza/brutfors.

37. Коломоєць Д.А. Особливості автоматизованого тестування якості програмного забезпечення / Коломоєць Д.А., Шаров С.В.. – Мелітополь, 2023.

38. Введення в тестування програмного забезпечення [Електронний ресурс] – Режим доступу до ресурсу: https://qalearning.com.ua/theory/lectures/material/testing-intro.

39. Sonar Cloud [Електронний ресурс] – Режим доступу до ресурсу: https://sonarcloud.io.

40. Gym application [Електронний ресурс] – Режим доступу до ресурсу: https://sonarcloud.io/project/overview?id=kolesroma_gym-application.

41. AWS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com.

42. Порівняння табличка: AWS vs Azure vs GCP [Електронний ресурс] – Режим доступу до ресурсу: https://smartnet.ua/porivnjannja-tablichka-aws-vs-azure-vs-gcp.

43. What is DevOps automation? [Електронний ресурс] – Режим доступу до ресурсу: https://www.redhat.com/en/topics/automation/what-is-devops-automation.

44. Amazon S3 [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/s3.

45. Amazon EC2 [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ec2.

46. What is Amazon VPC? [Електронний ресурс] – Режим доступу до ресурсу: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html.

47. AWS CodePipeline [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/codepipeline.

48. Everything You Need to Know About Environments [Електронний ресурс] – Режим доступу до ресурсу: https://www.qovery.com/blog/everything-you-need-to-know-about-deployment-environments.

49. GitHub code [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/kolesroma/gym-application.

ДОДАТКИ

Додаток А

Зробити інтерфейс користувача консистентним

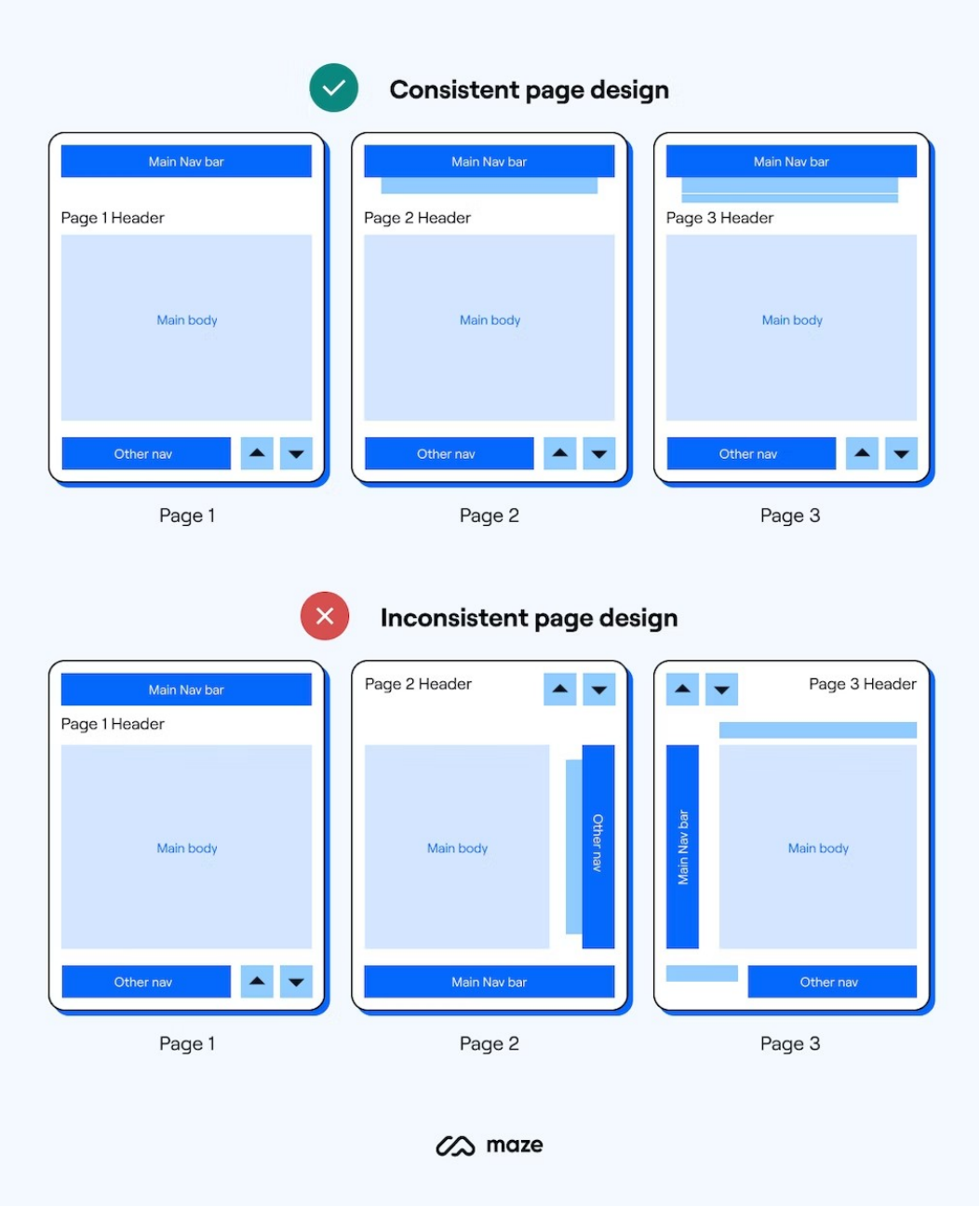
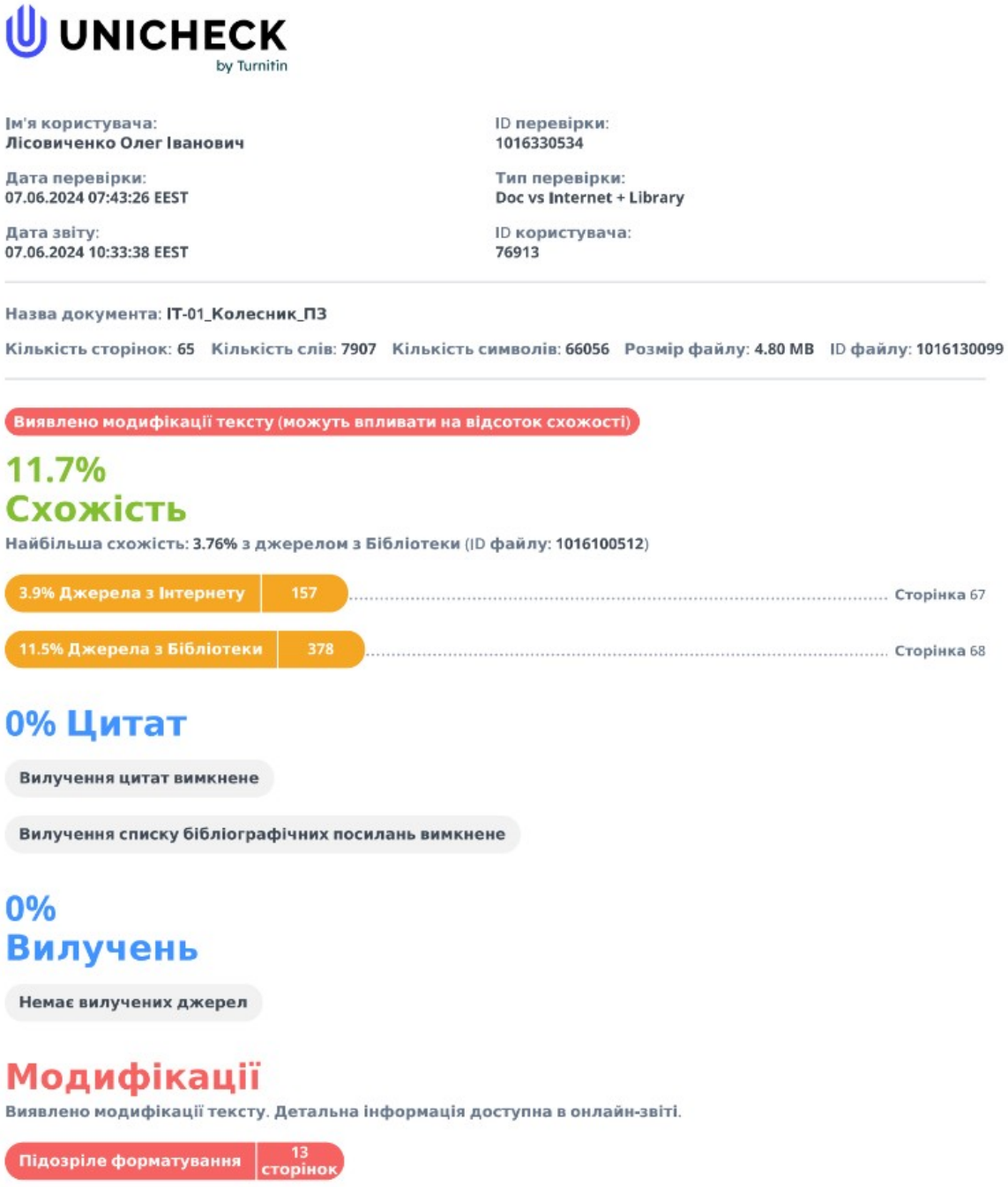


Рисунок А.1 – Критерії до дизайну інтерфейсу користувача


UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016330534

Дата перевірки:
07.06.2024 07:43:26 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
07.06.2024 10:33:38 EEST

ID користувача:
76913

Назва документа: IT-01_Колесник_ПЗ

Кількість сторінок: 65 Кількість слів: 7907 Кількість символів: 66056 Розмір файлу: 4.80 MB ID файлу: 1016130099

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

11.7%
Схожість

Найбільша схожість: 3.76% з джерелом з Бібліотеки (ID файлу: 1016100512)

3.9% Джерела з Інтернету 157 ..... Сторінка 67

11.5% Джерела з Бібліотеки 378 ..... Сторінка 68

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 13 сторінок

Рисунок Б.2 – Звіт подібності

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення для підтримки діяльності спортивного комплексу

Текст програми

КПІ.ІТ-0111.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Роман КОЛЕСНИК

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду

https://github.com/kolesroma/gym-application

файл ScheduleConfig**.java**

```
@Configuration
@EnableScheduling
public class ScheduleConfig {
}
```

файл WebSecurityConfig**.java**

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

   @Bean
   public SecurityFilterChain securityFilterChain(HttpSecurity http, JwtAuthorizationFilter jwtAuthorizationFilter)
throws Exception {
      return http
            .csrf(csrf -> csrf
                  .ignoringRequestMatchers("/registration/**")
                  .ignoringRequestMatchers("/auth/**")
                  .ignoringRequestMatchers("/users/**")
                  .ignoringRequestMatchers("/trainings")
                  .ignoringRequestMatchers("/diet")
                  .ignoringRequestMatchers("/notification/**")
                  .ignoringRequestMatchers("/trainees/trainers"))
            .cors(cors -> cors
                  .configurationSource(corsConfigurationSource()))
            .authorizeHttpRequests(auth -> auth
                  .requestMatchers("/notification/trigger-sending").anonymous()
                  .requestMatchers("/auth/**").permitAll()
                  .requestMatchers("/home").hasRole("TRAINER")
                  .requestMatchers("/trainees/**").authenticated()
                  .requestMatchers("/users/**").authenticated()
                  .requestMatchers("/registration/**").permitAll()
                  .anyRequest().authenticated())
            .addFilterBefore(jwtAuthorizationFilter, UsernamePasswordAuthenticationFilter.class)
            .logout(httpSecurityLogoutConfigurer -> httpSecurityLogoutConfigurer
                  .logoutUrl("/logout")
                  .permitAll()
                  .logoutSuccessUrl("/?successLoggedOut")
                  .addLogoutHandler(new HeaderWriterLogoutHandler(
                        new ClearSiteDataHeaderWriter(CACHE, COOKIES, STORAGE))))
            .build();
   }

   @Bean
   public JdbcUserDetailsManager jdbcUserDetailsManager(DataSource dataSource) {
      return new JdbcUserDetailsManager(dataSource);
   }

   @Bean
   public UserDetailsService userDetailsService(JdbcUserDetailsManager jdbcUserDetailsManager) {
      return jdbcUserDetailsManager;
   }

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

```
    private CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOriginPatterns(Collections.singletonList("*"));
        configuration.addAllowedMethod("*");
        configuration.addAllowedHeader("*");
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);

        return source;
    }

}
```

файл DietController**.java**
реалізація функціональної задачі генерації дієти

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/diet")
public class DietController {

    private final DietService dietService;

    @PostMapping
    private List<MealDto> generateDayRation(@RequestBody CreateDietDto createDietDto) {
        return dietService.generateDayRation(createDietDto);
    }

}
```

файл LoginController**.java**
реалізація функціональної задачі авторизації

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/auth")
public class LoginController {

    private final JwtAuthorizationService jwtAuthorizationService;
    private final UserService userService;

    @PostMapping("/login")
    public Map<String, String> login(@RequestBody UserEntity auth,
                                     HttpServletRequest request,
                                     HttpServletResponse response) {
        UserEntity userDb =
                userService.getByUsernameAndPassword(auth.getUsername(), auth.getPassword(),
request.getRemoteAddr());

        if (!userDb.getEnabled()) {
            throw new AccessDeniedException("Your account is disabled");
        }

        String token = jwtAuthorizationService.createToken(userDb);
        response.addHeader("Authorization", String.format("Bearer %s", token));
        return Map.of("token", token);
    }

    @PostMapping("/change-password")
```

```
    public Map<String, String> changePassword(@RequestBody @Validated ChangePasswordDto
changePasswordDto,
                                              Principal principal) {
        userService.changePassword(changePasswordDto);
        return Map.of("message", "Password was changed successfully", "username", principal.getName());
    }

    @PostMapping("/disable-account")
    public Map<String, String> disableAccount(Principal principal) {
        userService.disableAccount();
        return Map.of("message", "Account was disabled", "username", principal.getName());
    }

    @Deprecated
    @GetMapping("/logout")
    public String logout(HttpServletResponse response) {
        response.addHeader("Authorization", "");
        return "logged out";
    }

}
```

файл NotificationController.**java**
реалізація функціональної задачі надсилання сповіщень

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/notification")
public class NotificationController {

    private final NotificationService notificationService;
    private final UserService userService;

    @GetMapping("/status/me")
    public NotificationSettingsEntity getStatus() {
        UserEntity user = userService.getUserFromCurrentSecurityContext();
        return notificationService.getNotificationSettingsByEmail(user.getEmail());
    }

    @PostMapping("/status")
    public void setNotificationStatus(@RequestParam boolean status) {
        notificationService.setNotificationStatusTo(status);
    }

    @GetMapping("/trigger-sending")
    public void triggerSending() {
        notificationService.sendNotificationForTomorrowTraining();
    }

}
```

файл ReportController.java
реалізація функціональної задачі генерації звітів

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/report")
public class ReportController {

    private final ReportService reportService;
    private final UserService userService;
```

```
    @GetMapping("/visited")
    public List<TrainingVisitedReportDto> generateTrainingVisitedReport() {
        UserEntity user = userService.getUserFromCurrentSecurityContext();
        return reportService.generateTrainingVisitedReport(user.getUsername());
    }

    @GetMapping("/velocity")
    public List<TrainingVelocityReportDto> generateTrainingVelocityReport() {
        UserEntity user = userService.getUserFromCurrentSecurityContext();
        return reportService.generateTrainingVelocityReport(user.getUsername());
    }

}
```

файл TrainingController.java

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/trainings")
public class TrainingController {
    private final TrainingService trainingService;

    @GetMapping
    public List<TrainingDto> getAllTrainingsForUser(Authentication authentication) {
        if (AccessUtil.hasRole(authentication, "ROLE_TRAINEE")) {
         return trainingService.getTrainingListForTrainee(authentication.getName());
        }
        if (AccessUtil.hasRole(authentication, "ROLE_TRAINER")) {
            return trainingService.getTrainingListForTrainer(authentication.getName());
        }
        throw new AccessDeniedException("provided authentication neither has ROLE_TRAINEE nor
ROLE_TRAINER");
    }

    @PostMapping
    public TrainingDto createTraining(@RequestBody TrainingDto trainingDto, Authentication authentication) {
        if (AccessUtil.hasRole(authentication, "ROLE_TRAINEE")) {
            trainingDto.setTraineeName(authentication.getName());
        } else if (AccessUtil.hasRole(authentication, "ROLE_TRAINER")) {
            trainingDto.setTrainerName(authentication.getName());
        }
        return trainingService.create(trainingDto);
    }

}
```

файл CreateDietDto.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CreateDietDto {

    private int numberOfMeals;

    private int kcal;

    private double proteinsCoefficient;

    private double fatsCoefficient;

    private double carbohydratesCoefficient;
```

```
}
```

файл DishDto.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class DishDto {

    private int massInGrams;

    private String desc;

    private double proteins;

    private double fats;

    private double carbohydrates;

    public double getCalories() {
        return proteins * 4d + fats * 9d + carbohydrates * 4d;
    }

}
```

файл MealDto.java

```
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class MealDto {

    private List<DishDto> dishes;

    private int kcal;

}
```

файл TrainingVelocityReportDto.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class TrainingVelocityReportDto {

    private LocalDate trainingDate;

    private Integer trainingCount;

    private Integer totalDuration;

    private Double velocity;

}
```

файл TrainingVisitedReportDto.java

реалізація функціональної задачі генерації звітів

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class TrainingVisitedReportDto {
```

```
    private String traineeUsername;

    private Integer totalDuration;

    private Integer visitedCount;

    private Integer totalCount;

}
```

файл UpdateUserDto.java

```
@Data
public class UpdateUserDto {
    private TrainerDto trainer;

    private TraineeDto trainee;
}
```

файл AuthorityRole.java

```
public enum AuthorityRole {
    ROLE_TRAINEE,
    ROLE_TRAINER
}
```

файл AuthorityEntity.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "authorities")
@IdClass(AuthorityEntityId.class)
public class AuthorityEntity {
    @Id
    private String username;

    @Id
    @Enumerated(EnumType.STRING)
    private AuthorityRole authority;
}
```

файл DishEntity.java

реалізація функціональної задачі генерації дієти

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "dish")
public class DishEntity {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int massInGrams;

    @Column(name = "description")
    private String desc;

    private double proteins;
```

```
    private double fats;

    private double carbohydrates;

}
```

файл NotificationSettingsEntity.java
реалізація функціональної задачі надсилання сповіщень

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "notification_settings")
public class NotificationSettingsEntity {

    @Id
    @Column(name = "user_id")
    private Long userId;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id", insertable = false, updatable = false)
    private UserEntity user;

    private boolean enabled;

}
```

файл TraineeEntity.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "trainees")
public class TraineeEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private LocalDate dateOfBirth;

    private String address;

    @OneToOne(fetch = FetchType.LAZY, cascade={CascadeType.MERGE, CascadeType.PERSIST})
    @JoinColumn(name = "user_id")
    private UserEntity user;
}
```

файл TrainerEntity.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "trainers")
public class TrainerEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String specialization;

    @OneToOne(fetch = FetchType.LAZY, cascade={CascadeType.MERGE, CascadeType.PERSIST})
    @JoinColumn(name = "user_id")
```

```
    private UserEntity user;
}
```

файл TrainingEntity.java

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "trainings")
public class TrainingEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    @ManyToOne
    @JoinColumn(name = "trainee_id")
    private TraineeEntity trainee;

    @EqualsAndHashCode.Exclude
    @ToString.Exclude
    @ManyToOne
    @JoinColumn(name = "trainer_id")
    private TrainerEntity trainer;

    private String trainingName;

    private LocalDate trainingDate;

    private Integer trainingDuration;

    @ManyToOne(cascade={CascadeType.PERSIST})
    @JoinColumn(name = "training_type_id")
    private TrainingTypeEntity trainingType;
}
```

файл BruteForceTryingException.java

```
@StandardException
public class BruteForceTryingException extends RuntimeException {
}
```

файл JwtAuthorizationFilter.java

```
@Component
@RequiredArgsConstructor
public class JwtAuthorizationFilter extends OncePerRequestFilter {
    private final JwtAuthorizationService authorizationService;
    private final ObjectMapper mapper;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain) throws ServletException, IOException {
        try {
            String accessToken = authorizationService.getTokenFromUserRequest(request);
            Claims claims = authorizationService.parseTokenToClaims(accessToken);

            Authentication authentication = new UsernamePasswordAuthenticationToken(
                    claims.getSubject(),
                    null,
                    AuthorityUtils.commaSeparatedStringToAuthorityList(claims.get("roles", String.class))
```

```
            );
            SecurityContextHolder.getContext().setAuthentication(authentication);
        } catch (NoJwtInHeadersException ignored) {
        } catch (Exception e) {
            Map<String, Object> errorDetails = new HashMap<>();
            errorDetails.put("message", "Authentication Error");
            errorDetails.put("details", e.getMessage());
            response.setStatus(HttpStatus.FORBIDDEN.value());
            response.setContentType(MediaType.APPLICATION_JSON_VALUE);
            mapper.writeValue(response.getWriter(), errorDetails);
            return;
        }
        filterChain.doFilter(request, response);
    }
}
```

файл ClearBannedIpJobImpl.java

```
@Component
@RequiredArgsConstructor
public class ClearBannedIpJobImpl implements ClearBannedIpJob {
    private static final int RELOAD_BANNED_IP_IN_MILLIS = 300_000;

    private final BruteForceMonitorService bruteForceMonitorService;

    @Override
    @Scheduled(fixedRate = RELOAD_BANNED_IP_IN_MILLIS)
    public void clearBannedIpList() {
        bruteForceMonitorService.clearAll();
    }
}
```

файл TraineeMapper.java

```
@Mapper(componentModel = "spring")
public interface TraineeMapper {

    @Mapping(source = "user.username", target = "username")
    @Mapping(source = "user.firstName", target = "firstName")
    @Mapping(source = "user.lastName", target = "lastName")
    @Mapping(source = "dateOfBirth", target = "dateOfBirth")
    @Mapping(source = "address", target = "address")
    @Mapping(source = "user.isActive", target = "isActive")
    @Mapping(target = "trainers", ignore = true)
    @Mapping(source = "user.email", target = "email")
    TraineeDto toDto(TraineeEntity traineeEntity);

    @Mapping(source = "username", target = "user.username")
    @Mapping(source = "firstName", target = "user.firstName")
    @Mapping(source = "lastName", target = "user.lastName")
    @Mapping(source = "dateOfBirth", target = "dateOfBirth")
    @Mapping(source = "address", target = "address")
    @Mapping(source = "email", target = "user.email")
    TraineeEntity toEntity(TraineeDto traineeDto);
}
```

файл NotificationSettingsRepository.java

```
public interface NotificationSettingsRepository extends JpaRepository<NotificationSettingsEntity, Long> {

    Optional<NotificationSettingsEntity> findByUserEmail(String email);

}
```

файл TraineeRepository.java

```
public interface TraineeRepository extends JpaRepository<TraineeEntity, Long> {
    Optional<TraineeEntity> findByUserUsername(String username);
}
```

файл TrainerRepository.java

```
public interface TrainerRepository extends JpaRepository<TrainerEntity, Long> {
    Optional<TrainerEntity> findByUserUsername(String username);
}
```

файл TrainingRepository.java

```
public interface TrainingRepository extends JpaRepository<TrainingEntity, Long> {
    List<TrainingEntity> findByTraineeUserUsername(String traineeUsername);

    List<TrainingEntity> findByTrainerUserUsername(String trainerUsername);

    @Query(value = "select sum(training_duration) from trainings where trainee_id = ?", nativeQuery = true)
    int calculateTotalDurationForTraineeWithId(Long traineeId);

    @Query(value = "select count(visited) from trainings where trainee_id = ?", nativeQuery = true)
    int calculateTotalCount(Long traineeId);

    @Query(value = "select count(visited) from trainings where trainee_id = ? and visited", nativeQuery = true)
    int calculateVisitedCount(Long traineeId);

}
```

файл UserRepository.java

```
public interface UserRepository extends JpaRepository<UserEntity, Long> {
    Optional<UserEntity> findByUsernameAndPassword(String username, String password);

    Optional<UserEntity> findByUsername(String username);

    Boolean existsByUsername(String username);
}
```

файл BruteForceMonitorServiceInMemory.java

```
@Service
public class BruteForceMonitorServiceInMemory implements BruteForceMonitorService {
    private static final int THRESHOLD = 5;

    private final Map<String, Integer> failedAttemptsIpMap = new ConcurrentHashMap<>();

    @Override
    public void trackFailedAttempt(String ip) {
        failedAttemptsIpMap.put(ip, failedAttemptsIpMap.getOrDefault(ip, 0) + 1);
    }

    @Override
    public boolean isSuspiciousIp(String ip) {
        int failedAttempts = failedAttemptsIpMap.getOrDefault(ip, 0);
        return failedAttempts >= THRESHOLD;
    }

    @Override
    public void clearIp(String ip) {
        failedAttemptsIpMap.remove(ip);
    }
```

```
    @Override
    public void clearAll() {
        failedAttemptsIpMap.clear();
    }
}
```

файл DietServiceImpl.java

```
@Service
реалізація функціональної задачі генерації дієти
public class DietServiceImpl implements DietService {

    private static final double KCAL_PER_GRAM_FOR_PROTEINS = 4d;
    private static final double KCAL_PER_GRAM_FOR_FATS = 9d;
    private static final double KCAL_PER_GRAM_FOR_CARBOHYDRATES = 4d;

    private static final List<DishDto> MENU = new ArrayList<>();

    static {
        MENU.add(new DishDto(10, "porridge", 1.3, .34, 7.2));
        MENU.add(new DishDto(10, "vivsianka", 3, 3, 4));
        MENU.add(new DishDto(10, "sugar", 3, 3, 13));
        MENU.add(new DishDto(10, "chicken breast", 2.7, 1.4, 0));
        MENU.add(new DishDto(10, "salo", .14, 9.67, 0));
    }

    @Override
    public List<MealDto> generateDayRation(CreateDietDto createDietDto) {
        int kcal = createDietDto.getKcal();
        int numberOfMeals = createDietDto.getNumberOfMeals();
        double proteinsCoefficient = createDietDto.getProteinsCoefficient();
        double fatsCoefficient = createDietDto.getFatsCoefficient();
        double carbohydratesCoefficient = createDietDto.getCarbohydratesCoefficient();
        int kcalPerMeal = kcal / numberOfMeals;

        ArrayList<MealDto> meals = new ArrayList<>();
        for (int i = 0; i < numberOfMeals; i++) {
            meals.add(MealDto.builder().kcal(kcalPerMeal).build()); // breakfast
        }

        meals.forEach(meal -> setDishesForMeal(meal, proteinsCoefficient, fatsCoefficient,
carbohydratesCoefficient));
        return meals;
    }

    /**
     * using genetic algorithm.
     * create bestCombination.
     * set meal.setDishes(bestCombination) -> List<DishDto>
     * @param mealDto
     * @param proteinsCoefficient
     * @param fatsCoefficient
     * @param carbohydratesCoefficient
     */
    private static void setDishesForMeal(MealDto mealDto, double proteinsCoefficient, double fatsCoefficient,
double carbohydratesCoefficient) {
        int totalKcal = mealDto.getKcal(); // 500kcal
        double gramProteins = totalKcal * proteinsCoefficient / KCAL_PER_GRAM_FOR_PROTEINS; //200kcal =
50g
        double gramFats = totalKcal * fatsCoefficient / KCAL_PER_GRAM_FOR_FATS; //150kcal = 37.5g
        double gramCarbohydrates = totalKcal * carbohydratesCoefficient /
KCAL_PER_GRAM_FOR_CARBOHYDRATES; //150kcal = 16.7g
```

```
        List<DishDto> bestCombination = findBestCombination(totalKcal, gramProteins, gramFats,
gramCarbohydrates);
        List<DishDto> groupedDishesForMeal = groupDishesByDescriptionWithTotalMass(bestCombination);
        mealDto.setDishes(groupedDishesForMeal);
    }

    private static List<DishDto> groupDishesByDescriptionWithTotalMass(List<DishDto> bestCombination) {
        Map<String, List<DishDto>> groupedByDescription = bestCombination.stream()
                .collect(Collectors.groupingBy(DishDto::getDesc));

        List<DishDto> dishesList = new ArrayList<>();
        for (Map.Entry<String, List<DishDto>> entry : groupedByDescription.entrySet()) {
            String desc = entry.getKey();
            List<DishDto> dishes = entry.getValue();
            int totalMass = dishes.stream().mapToInt(DishDto::getMassInGrams).sum();
            double totalProteins = calculateTotalProteins(dishes);
            double totalFats = calculateTotalFats(dishes);
            double totalCarbohydrates = calculateTotalCarbohydrates(dishes);
            dishesList.add(new DishDto(totalMass, desc, totalProteins, totalFats, totalCarbohydrates));
        }

        return dishesList;
    }

    private static List<DishDto> findBestCombination(int totalKcal, double gramProteins, double gramFats, double
gramCarbohydrates) {
        List<DishDto> bestCombination = new ArrayList<>();
        double bestDifference = Double.MAX_VALUE;

        for (int i = 0; i < 1000; i++) {
            List<DishDto> currentCombination = generateRandomCombination(totalKcal);
            double currentProteins = calculateTotalProteins(currentCombination);
            double currentFats = calculateTotalFats(currentCombination);
            double currentCarbohydrates = calculateTotalCarbohydrates(currentCombination);

            double difference = Math.abs(gramProteins - currentProteins) +
                    Math.abs(gramFats - currentFats) +
                    Math.abs(gramCarbohydrates - currentCarbohydrates);

            if (difference < bestDifference) {
                bestDifference = difference;
                bestCombination = currentCombination;
            }
        }

        return bestCombination;
    }

    private static List<DishDto> generateRandomCombination(int totalKcal) {
        List<DishDto> combination = new ArrayList<>();
        int remainingKcal = totalKcal;

        Random random = new Random();

        while (remainingKcal > MENU.stream().mapToDouble(DishDto::getCalories).min().orElse(0)) {
            DishDto randomDish = MENU.get(random.nextInt(MENU.size()));
            combination.add(randomDish);
            remainingKcal -= randomDish.getCalories();
        }

        return combination;
    }
```

```
    private static double calculateTotalProteins(List<DishDto> dishes) {
        return dishes.stream().mapToDouble(DishDto::getProteins).sum();
    }

    private static double calculateTotalFats(List<DishDto> dishes) {
        return dishes.stream().mapToDouble(DishDto::getFats).sum();
    }

    private static double calculateTotalCarbohydrates(List<DishDto> dishes) {
        return dishes.stream().mapToDouble(DishDto::getCarbohydrates).sum();
    }

}
```

файл JwtAuthorizationServiceImpl.java

```
@Component
@Log4j2
public class JwtAuthorizationServiceImpl implements JwtAuthorizationService {
    private static final long TOKEN_EXPIRATION_IN_MILLIS = 86_400_000L;
    private static final String TOKEN_HEADER = "Authorization";
    private static final String TOKEN_PREFIX = "Bearer ";

    private final String secretKey;
    private final JwtParser jwtParser;
    private final UserDetailsService userDetailsService;

    public JwtAuthorizationServiceImpl(@Value("${security.jwt.token.secret}") String secretKey,
UserDetailsService userDetailsService) {
        this.secretKey = secretKey;
        this.jwtParser = Jwts.parser().setSigningKey(secretKey);
        this.userDetailsService = userDetailsService;
    }

    @Override
    public String createToken(UserEntity user) {
        String username = user.getUsername();
        UserDetails userDetails = userDetailsService.loadUserByUsername(username);
        Claims claims = Jwts.claims().setSubject(username);
        claims.put("id", user.getId());
        claims.put("firstName", user.getFirstName());
        claims.put("lastName", user.getLastName());
        claims.put("roles", getRolesCommaSeparatedString(userDetails));
        Date now = new Date();
        String jwtToken = Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(new Date(now.getTime() + TOKEN_EXPIRATION_IN_MILLIS))
                .signWith(SignatureAlgorithm.HS256, secretKey)
                .compact();
        log.debug(String.format("created token for user %s", username));
        return jwtToken;
    }

    @Override
    public String getTokenFromUserRequest(HttpServletRequest request) {
        String authorizationToken = request.getHeader(TOKEN_HEADER);
        if (authorizationToken != null && authorizationToken.startsWith(TOKEN_PREFIX)) {
            return authorizationToken.substring(TOKEN_PREFIX.length());
        }
        throw new NoJwtInHeadersException();
```

```
    }

    @Override
    public Claims parseTokenToClaims(String token) {
        Claims claims = jwtParser.parseClaimsJws(token).getBody();
        if (!isValidClaims(claims)) {
            throw new IllegalArgumentException("claims are invalid");
        }
        return claims;
    }

    private static String getRolesCommaSeparatedString(UserDetails userDetails) {
        return userDetails.getAuthorities()
                .stream()
                .map(GrantedAuthority::getAuthority)
                .collect(Collectors.joining(","));
    }

    private boolean isValidClaims(Claims claims) throws AuthenticationException {
        return claims != null
                && claims.getExpiration().after(new Date());
    }
}
```

файл LoginPasswordGeneratorImpl.java

```
@Service
public class LoginPasswordGeneratorImpl implements LoginPasswordGenerator {

    @Override
    public String generateLogin(String firstName, String lastName) {
        if (notMatches(firstName) || notMatches(lastName)) {
            throw new IllegalArgumentException("Name must contain only letters");
        }
        return firstName.toLowerCase() + "_" + lastName.toLowerCase();
    }

    @Override
    public String generatePassword() {
        PasswordGenerator passwordGenerator = new PasswordGenerator();

        CharacterRule lowerCaseRule = new CharacterRule(EnglishCharacterData.LowerCase);
        lowerCaseRule.setNumberOfCharacters(4);

        CharacterRule upperCaseRule = new CharacterRule(EnglishCharacterData.UpperCase);
        upperCaseRule.setNumberOfCharacters(4);

        CharacterRule digitRule = new CharacterRule(EnglishCharacterData.Digit);
        digitRule.setNumberOfCharacters(2);

        CharacterData specialChars = new CharacterData() {
            @Override
            public String getErrorCode() {
                return "ERROR";
            }

            @Override
            public String getCharacters() {
                return "!@#$%^&*()_+";
            }
        };
        CharacterRule specialCharsRule = new CharacterRule(specialChars);
        specialCharsRule.setNumberOfCharacters(2);
```

```
        return passwordGenerator.generatePassword(16, specialCharsRule, lowerCaseRule, upperCaseRule,
digitRule);
    }

    private static boolean notMatches(String string) {
        return !string.matches("^[a-zA-Z]+$");
    }
}
```

файл NotificationServiceImpl.java
реалізація функціональної задачі надсилання сповіщень

```
@Service
@Log4j2
@RequiredArgsConstructor
public class NotificationServiceImpl implements NotificationService {

    private final JavaMailSender emailSender;
    private final NotificationSettingsRepository notificationSettingsRepository;
    private final UserService userService;
    private final TrainingRepository trainingRepository;

    @Override
    public void sendMessage(String email, String subject, String text) {
        NotificationSettingsEntity settings = getNotificationSettingsByEmail(email);
        if (!settings.isEnabled()) {
            log.info("Email {} disable notification", email);
            return;
        }
        sendSimpleMessage(email, subject, text);
        log.info("Sent letter to {}", email);
    }

    @Override
    public NotificationSettingsEntity getNotificationSettingsByEmail(String email) {
        return notificationSettingsRepository.findByUserEmail(email)
                .orElse(new NotificationSettingsEntity(0L, null, true));
    }

    @Override
    public void setNotificationStatusTo(boolean status) {
        UserEntity user = userService.getUserFromCurrentSecurityContext();
        Long userId = user.getId();
        Optional<NotificationSettingsEntity> settingsOptional = notificationSettingsRepository.findById(userId);
        if (settingsOptional.isEmpty()) {
            var newSettings = new NotificationSettingsEntity(userId, user, status);
            notificationSettingsRepository.save(newSettings);
        } else {
            NotificationSettingsEntity settings = settingsOptional.get();
            settings.setEnabled(status);
            notificationSettingsRepository.save(settings);
        }
    }

    @Override
    @Scheduled(cron = "0 1 0 * * *") // Runs at 00:01 every day
    public void sendNotificationForTomorrowTraining() {
        log.info("Started job to send notifications");
        List<TrainingEntity> filteredTrainings = trainingRepository.findAll()
                .stream()
                .filter(training -> LocalDate.now().equals(training.getTrainingDate()))
```

```
                .toList();
        filteredTrainings
                .forEach(training -> sendSimpleMessage(training.getTrainee().getUser().getEmail(), "Reminder", "You
have training soon: " + training.getTrainingName() + ".\nWith date: " + training.getTrainingDate()));
        log.info("Ended job to send notifications. Sent to count of users: " + filteredTrainings.size());
    }

    private void sendSimpleMessage(String to, String subject, String text) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("kolesgym@gmail.com");
        message.setTo(to);
        message.setSubject(subject);
        message.setText(text);
        emailSender.send(message);
    }

}
```

файл ReportServiceImpl.java
реалізація функціональної задачі генерації звітів

```
@Service
@Log4j2
@RequiredArgsConstructor
public class ReportServiceImpl implements ReportService {

    private final TrainingRepository trainingRepository;

    @Override
    @Transactional
    public ArrayList<TrainingVisitedReportDto> generateTrainingVisitedReport(String trainerUsername) {
        List<TrainingEntity> trainings = trainingRepository.findByTrainerUserUsername(trainerUsername);
        ArrayList<TrainingVisitedReportDto> accumulator = new ArrayList<>();

        trainings.forEach(training -> {
            String traineeUsername = training.getTrainee().getUser().getUsername();
            if (!accumulatorContainsUsername(accumulator, traineeUsername)) {
                Long traineeId = training.getTrainee().getId();
                int totalDuration = trainingRepository.calculateTotalDurationForTraineeWithId(traineeId);
                int visitedCount = trainingRepository.calculateVisitedCount(traineeId);
                int totalCount = trainingRepository.calculateTotalCount(traineeId);
                TrainingVisitedReportDto visitedReportDto = new TrainingVisitedReportDto(traineeUsername,
totalDuration, visitedCount, totalCount);
                accumulator.add(visitedReportDto);
            }
        });

        return accumulator;
    }

    @Override
    @Transactional
    public ArrayList<TrainingVelocityReportDto> generateTrainingVelocityReport(String trainerUsername) {
        LocalDate now = LocalDate.now();
        LocalDate start = now.withDayOfMonth(1);
        LocalDate end = now.withDayOfMonth(now.lengthOfMonth());

        List<TrainingEntity> trainings = trainingRepository.findByTrainerUserUsername(trainerUsername)
                .stream()
                .filter(training -> training.getTrainingDate().isAfter(start))
                .toList();
        ArrayList<TrainingVelocityReportDto> accumulator = new ArrayList<>();
```

```
        for (LocalDate date = start; date.isBefore(end); date = date.plusDays(1)) {
            LocalDate dateCopy = date;
            List<TrainingEntity> trainingEntities = trainings.stream().filter(x ->
x.getTrainingDate().equals(dateCopy)).toList();
            int totalDuration = trainingEntities.stream().mapToInt(TrainingEntity::getTrainingDuration).sum();
            TrainingVelocityReportDto trainingVelocityReportDto = new TrainingVelocityReportDto(dateCopy,
trainingEntities.size(), totalDuration, totalDuration / 480d);
            accumulator.add(trainingVelocityReportDto);
        }

        return accumulator;
    }

    private static boolean accumulatorContainsUsername(ArrayList<TrainingVisitedReportDto> accumulator,
                                       String traineeUsername) {
        return accumulator.stream()
                .map(TrainingVisitedReportDto::getTraineeUsername)
                .anyMatch(x -> x.equals(traineeUsername));
    }

    @Bean
    CommandLineRunner f(ReportService reportService) {
        return x -> reportService.generateTrainingVelocityReport("cleaner_anatoliy");
    }

}
```

файл TraineeServiceImpl.java

```
@Service
@RequiredArgsConstructor
@Log4j2
public class TraineeServiceImpl implements TraineeService {
    private final TrainerRepository trainerRepository;
    private final UserRepository userRepository;
    private final UserService userService;
    private final TraineeRepository traineeRepository;
    private final TrainingRepository trainingRepository;
    private final TraineeMapper traineeMapper;
    private final TrainerMapper trainerMapper;
    private final LoginPasswordGenerator loginPasswordGenerator;
    private final PasswordEncoder passwordEncoder;
    private final AuthorityRepository authorityRepository;

    @Override
    public List<TraineeDto> getAll() {
        return traineeRepository.findAll()
                .stream()
                .map(traineeMapper::toDto)
                .toList();
    }

    @Override
    @Transactional
    public TraineeDto getByUsername(String username) {
        TraineeEntity trainee = traineeRepository.findByUserUsername(username)
                .orElseThrow(() -> new EntityNotFound("no trainee with such username"));

        List<TrainerDto> trainersOfTrainee = trainingRepository.findByTraineeUserUsername(username)
                .stream()
                .map(TrainingEntity::getTrainer)
                .map(trainerMapper::toDto)
```

```
                .toList();

        TraineeDto traineeDto = traineeMapper.toDto(trainee);
        traineeDto.setTrainers(trainersOfTrainee);

        return traineeDto;
    }

    @Override
    @Transactional
    public Map<String, String> create(TraineeDto trainee) {
        String username = loginPasswordGenerator.generateLogin(trainee.getFirstName(), trainee.getLastName());
        if (userRepository.existsByUsername(username)) {
            throw new IllegalArgumentException(String.format("User with username %s already registered",
username));
        }
        String passwordRaw = loginPasswordGenerator.generatePassword();

        traineeRepository.save(getTraineeEntity(trainee, username, passwordRaw));
        authorityRepository.save(getTraineeAuthority(username));

        log.info("Registered new trainee: {}", trainee);
        return Map.of("username", username, "password", passwordRaw);
    }

    @Override
    public TraineeDto update(TraineeDto traineeDto, String username) {
        if (traineeDto == null) {
            throw new IllegalArgumentException("provided trainee is null");
        }
        TraineeEntity trainee = traineeRepository.findByUserUsername(username)
                .orElseThrow(() -> new EntityNotFound("no trainee with such username"));
        prepareTrainee(trainee, traineeDto);
        traineeRepository.save(trainee);
        log.info("Updated trainee profile: {}", trainee);
        return traineeMapper.toDto(trainee);
    }

    @Override
    public void createEmptyTrainings(List<String> trainerUsernameList) {
        TraineeEntity trainee = getTraineeFromSecurityContext();
        List<TrainingEntity> trainingEntities = trainerUsernameList.stream()
                .map(trainerUsername -> prepareEmptyTraining(trainerUsername, trainee))
                .toList();
        trainingRepository.saveAll(trainingEntities);
    }

    private TraineeEntity getTraineeFromSecurityContext() {
        String usernameTrainee = userService.getUserFromCurrentSecurityContext().getUsername();
        return traineeRepository.findByUserUsername(usernameTrainee)
                .orElseThrow(() -> new EntityNotFound("no trainee with such username"));
    }

    private TrainingEntity prepareEmptyTraining(String trainerUsername, TraineeEntity trainee) {
        TrainerEntity trainer = trainerRepository.findByUserUsername(trainerUsername)
                .orElseThrow(() -> new EntityNotFound("no trainer with such username"));
        TrainingEntity training = new TrainingEntity();
        training.setTrainer(trainer);
        training.setTrainee(trainee);
        return training;
    }

    private static void prepareTrainee(TraineeEntity entity, TraineeDto dto) {
```

```
        entity.setAddress(dto.getAddress());
        entity.setDateOfBirth(dto.getDateOfBirth());
        UserEntity user = entity.getUser();
        user.setFirstName(dto.getFirstName());
        user.setLastName(dto.getLastName());
    }

    private static AuthorityEntity getTraineeAuthority(String login) {
        AuthorityEntity authorityEntity = new AuthorityEntity();
        authorityEntity.setUsername(login);
        authorityEntity.setAuthority(AuthorityRole.ROLE_TRAINEE);
        return authorityEntity;
    }

    private TraineeEntity getTraineeEntity(TraineeDto traineeDto, String login, String passwordRaw) {
        TraineeEntity traineeEntity = traineeMapper.toEntity(traineeDto);
        UserEntity userEntity = traineeEntity.getUser();
        userEntity.setUsername(login);
        userEntity.setPassword(passwordEncoder.encode(passwordRaw));
        userEntity.setEnabled(true);
        userEntity.setIsActive(true);
        return traineeEntity;
    }

}
```

файл TrainerServiceImpl.java

```
@Service
@RequiredArgsConstructor
@Log4j2
public class TrainerServiceImpl implements TrainerService {
    private final UserRepository userRepository;
    private final TrainerRepository trainerRepository;
    private final TrainerMapper trainerMapper;
    private final LoginPasswordGenerator loginPasswordGenerator;
    private final PasswordEncoder passwordEncoder;
    private final AuthorityRepository authorityRepository;

    @Override
    public TrainerDto getByUsername(String username) {
        TrainerEntity trainer = trainerRepository.findByUserUsername(username)
                .orElseThrow(() -> new EntityNotFound("no trainer with such username"));
        return trainerMapper.toDto(trainer);
    }

    @Override
    @Transactional
    public Map<String, String> create(TrainerDto trainer) {
        String username = loginPasswordGenerator.generateLogin(trainer.getFirstName(), trainer.getLastName());
        if (userRepository.existsByUsername(username)) {
            throw new IllegalArgumentException(String.format("User with username %s already registered",
username));
        }
        String passwordRaw = loginPasswordGenerator.generatePassword();

        trainerRepository.save(getTrainerEntity(trainer, username, passwordRaw));
        authorityRepository.save(getTrainerAuthority(username));

        log.info("Registered new trainer: {}", trainer);
        return Map.of("username", username, "password", passwordRaw);
    }
```

```
    @Override
    public TrainerDto update(TrainerDto trainerDto, String username) {
        if (trainerDto == null) {
            throw new IllegalArgumentException("provided trainer is null");
        }
        TrainerEntity trainer = trainerRepository.findByUserUsername(username)
                .orElseThrow(() -> new EntityNotFound("no trainer with such username"));
        prepareTrainer(trainer, trainerDto);
        trainerRepository.save(trainer);
        log.info("Updated trainer profile: {}", trainer);
        return trainerMapper.toDto(trainer);
    }

    @Override
    public List<TrainerDto> getAllFreeTrainers(String usernameTrainee) {
        return trainerRepository.findAll()
                .stream()
                .filter(trainerEntity -> notInTraineeList(trainerEntity, usernameTrainee))
                .map(trainerMapper::toDto)
                .toList();
    }

    private boolean notInTraineeList(TrainerEntity trainer, String usernameTrainee) {
        return true;
    }

    private static void prepareTrainer(TrainerEntity entity, TrainerDto dto) {
        entity.setSpecialization(dto.getSpecialization());
        UserEntity user = entity.getUser();
        user.setFirstName(dto.getFirstName());
        user.setLastName(dto.getLastName());
    }

    private static AuthorityEntity getTrainerAuthority(String login) {
        AuthorityEntity authorityEntity = new AuthorityEntity();
        authorityEntity.setUsername(login);
        authorityEntity.setAuthority(AuthorityRole.ROLE_TRAINER);
        return authorityEntity;
    }

    private TrainerEntity getTrainerEntity(TrainerDto trainerDto, String login, String passwordRaw) {
        TrainerEntity trainerEntity = trainerMapper.toEntity(trainerDto);
        UserEntity userEntity = trainerEntity.getUser();
        userEntity.setUsername(login);
        userEntity.setPassword(passwordEncoder.encode(passwordRaw));
        userEntity.setEnabled(true);
        userEntity.setIsActive(true);
        return trainerEntity;
    }

}
```

файл TrainingServiceImpl.java
реалізація функціональної задачі запису на тренінг

```
@Service
@RequiredArgsConstructor
@Log4j2
public class TrainingServiceImpl implements TrainingService {
    private final TrainingRepository trainingRepository;
    private final TrainingMapper trainingMapper;
    private final TrainerRepository trainerRepository;
```

```
    private final TraineeRepository traineeRepository;
    private final NotificationService notificationService;

    @Override
    public List<TrainingDto> getTrainingListForTrainer(String trainerUsername) {
        return trainingRepository.findByTrainerUserUsername(trainerUsername)
                .stream()
                .map(trainingMapper::toDto)
                .toList();
    }

    @Override
    public List<TrainingDto> getTrainingListForTrainee(String traineeUsername) {
        return trainingRepository.findByTraineeUserUsername(traineeUsername)
                .stream()
                .map(trainingMapper::toDto)
                .toList();
    }

    @Override
    public TrainingDto create(TrainingDto trainingDto) {
        TrainerEntity trainer = trainerRepository.findByUserUsername(trainingDto.getTrainerName())
                .orElseThrow(() -> new EntityNotFound("no trainer with such username"));
        TraineeEntity trainee = traineeRepository.findByUserUsername(trainingDto.getTraineeName())
                .orElseThrow(() -> new EntityNotFound("no trainee with such username"));
        TrainingEntity trainingEntity = trainingMapper.toEntity(trainingDto);
        trainingEntity.setTrainer(trainer);
        trainingEntity.setTrainee(trainee);
        trainingRepository.save(trainingEntity);
        TrainingDto returnedDto = trainingMapper.toDto(trainingEntity);
        sendCreatedTrainingEmail(trainingDto, trainee);
        return returnedDto;
    }

    private void sendCreatedTrainingEmail(TrainingDto trainingDto, TraineeEntity trainee) {
        notificationService.sendMessage(trainee.getUser().getEmail(),
                "Created Trainings",
                "You created new training for date: " + trainingDto.getTrainingDate() +
                        ".\nWith name: " + trainingDto.getTrainingName() +
                        ".\nWith trainer: " + trainingDto.getTrainerName() +
                        ".\nWith type: " + trainingDto.getTrainingType() + "."
        );
    }

}
```

файл UserServiceImpl.java

```
@Service
@RequiredArgsConstructor
@Log4j2
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final BruteForceMonitorService bruteForceMonitorService;
    private final JdbcUserDetailsManager jdbcUserDetailsManager;

    @Override
    public UserEntity getByUsernameAndPassword(String username, String password, String ip) {
        if (bruteForceMonitorService.isSuspiciousIp(ip)) {
            throw new AccessDeniedException("Your IP is suspicious and was banned because of many unsuccessful
attempts to log in");
        }
```

```
        UserEntity userDb;
        try {
            userDb = getByUsername(username);
            if (!isPasswordMatches(userDb, password)) {
                throw new BruteForceTryingException();
            }
        } catch (BruteForceTryingException e) {
            log.debug(String.format("Brute force for username %s", username));
            bruteForceMonitorService.trackFailedAttempt(ip);
            throw new IllegalArgumentException(e);
        }
        return userDb;
    }

    @Override
    public UserEntity getByUsername(String username) {
        return userRepository.findByUsername(username)
                .orElseThrow(BruteForceTryingException::new);
    }

    @Override
    public void changePassword(ChangePasswordDto changePasswordDto) {
        String oldPassword = changePasswordDto.getOldPassword();
        String newPassword = changePasswordDto.getNewPassword();
        if (!isPasswordMatches(getUserFromCurrentSecurityContext(), oldPassword)) {
            throw new IllegalArgumentException("Provided old password differs from password saved in database");
        }
        if (!newPassword.equals(changePasswordDto.getNewPasswordRepeat())) {
            throw new IllegalArgumentException("New password and password repeat are not equal");
        }
        jdbcUserDetailsManager.changePassword(
                passwordEncoder.encode(oldPassword),
                passwordEncoder.encode(newPassword)
        );
    }

    @Override
    public UserEntity getUserFromCurrentSecurityContext() {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        return userRepository.findByUsername(authentication.getName())
                .orElseThrow(() -> new EntityNotFound("Authentication not provided"));
    }

    @Override
    public void disableAccount() {
        UserEntity user = getUserFromCurrentSecurityContext();
        user.setEnabled(false);
        userRepository.save(user);
    }

    @Override
    public boolean isPasswordMatches(UserEntity userDb, String rawPassword) {
        return passwordEncoder.matches(rawPassword, userDb.getPassword());
    }
}
```

файл AiDrivenXStackApplication.java

```
@SpringBootApplication
public class AiDrivenXStackApplication {

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

```
    }

}
```

файл V1__init.sql

```
create table trainees
(
    id            bigint auto_increment,
    address       varchar(255),
    date_of_birth date,
    user_id       bigint,
    primary key (id)
);

create table trainers
(
    id             bigint auto_increment,
    specialization varchar(255),
    user_id        bigint,
    primary key (id)
);

create table training_types
(
    id                 bigint auto_increment,
    training_type_name varchar(255),
    primary key (id)
);

create table trainings
(
    id               bigint auto_increment,
    training_date    date,
    training_duration integer,
    training_name    varchar(255),
    trainee_id       bigint,
    trainer_id       bigint,
    training_type_id bigint,
    primary key (id)
);

create table users
(
    id         bigint auto_increment,
    first_name varchar(255),
    is_active  boolean,
    last_name  varchar(255),
    password   varchar(255),
    username   varchar(255),
    primary key (id)
);

alter table trainees
    add constraint UK_trainees_user_id unique (user_id);

alter table trainers
    add constraint UK_trainers_user_id unique (user_id);


alter table trainees
    add constraint FK_trainees_user_id foreign key (user_id) references users (id);

alter table trainers
```

```
    add constraint FK_trainers_user_id foreign key (user_id) references users (id);

alter table trainings
    add constraint FK_trainings_trainee_id foreign key (trainee_id) references trainees (id);

alter table trainings
    add constraint FK_trainings_trainer_id foreign key (trainer_id) references trainers (id);

alter table trainings
    add constraint FK_trainings_training_type_id foreign key (training_type_id) references training_types (id);
```

файл V2__add_authorities_modified_user.sql

```
create table authorities
(
    username  varchar(50) not null references users (username),
    authority varchar(50) not null
);

create unique index ix_auth_username on authorities (username, authority);

alter table users
    add `enabled` boolean not null default '1'
```

файл V3__username_unique.sql

```
create unique index ix_user_username on users (username);
```

файл V4__notification_settings.sql

```
CREATE TABLE notification_settings
(
    user_id BIGINT,
    enabled BOOLEAN DEFAULT TRUE,
    FOREIGN KEY (user_id) REFERENCES users (id),
    UNIQUE (user_id)
);
```

файл V5__add_email.sql

```
alter table users add column email varchar(100);
```

файл V6__add_dish.sql

```
create table dish
(
    id           bigint auto_increment,
    mass_in_grams int,
    description   varchar(100),
    proteins      float,
    fats          float,
    carbohydrates float,
    primary key (id)
);
```

файл V7__add_visited.sql

```
alter table trainings add column visited tinyint;
```

файл index.js

```
const express = require('express');
```

```
const bodyParser = require('body-parser');
const uuid = require('uuid');
const jwt = require('jsonwebtoken');
const multer = require('multer');
const fs = require('fs');
const mongoose = require('mongoose');
const cors = require('cors');

const crypt = require('./crypt.js');
const mongo = require('./mongo.js');
const jwtModule = require('./jwt.js');
const axios = require("axios");
const {findUserByLogin, insertUser, updatePasswordForUsername, updateImageForUser} = require("./mongo");

const app = express();
const PORT = process.env.PORT || 3000;

const JWT_SECRET = 'mysecretkey';
const MONGO_DB_URL = "mongodb://127.0.0.1:27017/nodegym";

const JAVA_HOST_URL = 'http://localhost:8080';
const JAVA_LOGIN_URL = JAVA_HOST_URL + '/auth/login';
const JAVA_REGISTRATION_STUDENT = JAVA_HOST_URL + '/registration/trainee';
const JAVA_REGISTRATION_TRAINER = JAVA_HOST_URL + '/registration/trainer';
const JAVA_CHANGE_PASSWORD = JAVA_HOST_URL + '/auth/change-password';

app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));

app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something went wrong!');
});

app.listen(PORT, () => {
   console.log(`Server is running on port ${PORT}`);
});

// app.use((req, res) => {
//     return res.status(500).json({message: 'Something broke!'});
// });

// routes

async function generateJavaToken(login, password) {
   return axios.post(JAVA_LOGIN_URL, {username: login, password: password})
      .then(response => {
         return response.data;
      })
      .catch(error => {
         return error.response.data
      });
}

app.post('/login', (req, res) => {
   const {login, password} = req.body;
   mongo.findUserByLogin(login)
      .then(async user => {
         if (!user) {
            return res.status(401).json({message: 'Invalid credentials'});
         }
         const isValidPassword = crypt.comparePasswords(user, password);
```

```
            if (isValidPassword) {
                user.token = jwt.sign({login, "role": "user"}, JWT_SECRET, {expiresIn: '24h'});
                const javaResponse = await generateJavaToken(login, password);
                if (javaResponse.token) {
                    res.status(200).json({
                        message: 'Login successful',
                        tokenNode: user.token,
                        tokenJava: javaResponse.token
                    });
                } else {
                    res.status(403).json(javaResponse);
                }
            } else {
                res.status(401).json({message: 'Invalid credentials code 2222'});
            }
        });
});

function createNewJavaStudent(student) {
    return axios.post(JAVA_REGISTRATION_STUDENT, student)
        .then(response => {
            return response.data;
        })
        .catch(error => {
            return error.response.data;
        });
}

function createNewJavaTrainer(trainer) {
    return axios.post(JAVA_REGISTRATION_TRAINER, trainer)
        .then(response => {
            return response.data;
        })
        .catch(error => {
            return error.response.data;
        });
}

function createNewJavaUser(type, user) {
    if (type === 'trainee') {
        return createNewJavaStudent(user);
    } else if (type === 'trainer') {
        return createNewJavaTrainer(user);
    } else {
        throw Error("Invalid registration type provided: " + type);
    }
}

app.post('/register/:type', (req, res) => {
    const javaUser = createNewJavaUser(req.params.type, req.body);
    javaUser.then(response => {
        const message = response.message;
        const username = response.username;
        const password = response.password;
        if (username && password) {
            const newUser = {
                id: uuid.v4(),
                login: username,
                password: crypt.cryptPassword(password)
            };
            mongo.insertUser(newUser);
            res.status(201).json({
                message: `User ${username} registered successfully`,
```

```
                username: username,
                password: password
            });
        } else {
            return res.status(401).json({message: message});
        }
    });
});

function changePassword(changePasswordModel, config) {
    return axios.post(JAVA_CHANGE_PASSWORD, changePasswordModel, config)
        .then(response => {
            return {status: 'ok', container: response.data};
        })
        .catch(error => {
            return {status: 'bad', container: error.response.data};
        });
}

function updateUserInMongo(username, newPassword) {
    updatePasswordForUsername(crypt.cryptPassword(newPassword), username);
}

app.post('/change-password', (req, res) => {
    const {newPassword} = req.body;
    const body = req.body;
    const config = {
        headers: {authorization: `${req.headers.authorization}`}
    };
    const changedPassword = changePassword(body, config);
    changedPassword.then(response => {
        if (response.status === 'ok') {
            updateUserInMongo(response.container.username, newPassword);
            return res.status(200).json(response.container);
        } else {
            return res.status(400).json(response.container);
        }
    });
});

app.get('/logout', (req, res) => {
    res.setHeader('Authorization', '');
    res.status(200).json({message: 'Logged out'});
});

// filter

app.get('/some', (req, res) => {
    jwtModule.verifyToken(req, res);
    res.status(200).json({message: 'norm'});
});

// images base64

const Photo = mongoose.model('Photo', {data: Buffer, contentType: String});

Photo.on('error', (error) => {
  console.error('Mongoose error:', error);
});

mongoose.connect(MONGO_DB_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
```

```
});

const db = mongoose.connection;
db.on('error', console.error.bind(console, 'MongoDB connection error:'));

const storage = multer.diskStorage({
   destination: (req, file, cb) => {
      cb(null, 'uploads/');
   },
   filename: (req, file, cb) => {
      cb(null, Date.now() + '-' + file.originalname);
   },
});

const upload = multer({storage});

function decodeToken(rawJwtToken) {
   try {
      return jwt.verify(rawJwtToken, JWT_SECRET);
   } catch (error) {
      console.error('JWT verification failed:', error.message);
   }
}

app.post('/upload-photo', upload.single('photo'), (req, res) => {
   const file = req.file;
   if (!file) {
      return res.status(400).json({error: 'No file uploaded.'});
   }

   const tokenNode = req.headers.tokennode;
   const claims = decodeToken(tokenNode);
   const username = claims.login;

   const fileData = fs.readFileSync(file.path);
   const base64Data = fileData.toString('base64');

   const photo = new Photo({
      data: Buffer.from(base64Data, 'base64'),
      contentType: file.mimetype,
      uploadTimestamp: new Date(),
   });

   photo.save((err, result) => {
      if (err) {
         console.error(err);
         return res.status(500).json({error: 'Error saving the photo.'});
      }
      updateImageForUser(username, result._id + "");
      res.json({message: 'Photo uploaded successfully', photoId: result._id});
   });
});

app.get('/user-photo', (req, res) => {
   const username = decodeToken(req.headers.tokennode).login;
   findUserByLogin(username)
      .then(user => {
         if (!user) {
            return res.status(401).json({message: 'Invalid credentials'});
         }
         Photo.findById(user.photoId, (err, photo) => {
            if (err) {
               console.error(err);
```

```
                return res.status(500).json({error: 'Error retrieving the photo.'});
            }
            res.contentType(photo.contentType);
            res.send(photo.data);
        });
    });
});
```

файл jwt.js

```
const jwt = require('jsonwebtoken');

const JWT_SECRET = 'mysecretkey';

function verifyToken(req, res) {
    const authHeader = req.headers['authorization'];
    if (!authHeader.startsWith("Bearer")) {
        return res.status(401).json({message: 'Unauthorized'});
    }
    const token = authHeader.split(' ')[1];

    if (!token) {
        return res.status(401).json({message: 'Unauthorized'});
    }
    jwt.verify(token, JWT_SECRET, (err, user) => {
        if (err) {
            return res.status(401).json({message: 'Unauthorized'});
        }
        req.user = user;
    });
}

module.exports = {
    verifyToken
};
```

файл mongo.js

```
const {MongoClient} = require('mongodb');
const uri = 'mongodb://localhost:27017';

const client = new MongoClient(uri);

client.connect()
    .then(() => {
        console.log('Connected to the MongoDB server');
    })
    .catch((err) => {
        console.error('Error connecting to the MongoDB server:', err);
    });


const db = client.db('nodegym');
const users = db.collection('users');

function insertUser(user) {
    users.insertOne(user)
        .then((result) => {
            console.log('Registered user:', result.insertedId);
        })
        .catch((err) => {
            console.error('Error inserting document:', err);
        });
}
```

```
function updatePasswordForUsername(newPasswordHashed, username) {
   users.updateOne({login: username}, {$set: {password: newPasswordHashed}})
      .then((result) => {
         console.log('Updated user password');
      })
      .catch((err) => {
         console.error('Error updating document:', err);
      });
}

function updateImageForUser(username, photoId) {
   users.updateOne({login: username}, {$set: {photoId: photoId}})
      .then((result) => {
         console.log('Updated user photo');
      })
      .catch((err) => {
         console.error('Error updating document:', err);
      });
}

async function findUserByLogin(login) {
   return await users.findOne({login: login});
}

module.exports = {
   insertUser,
   findUserByLogin,
   updatePasswordForUsername,
   updateImageForUser
};
```

файл crypt.js

```
const bcrypt = require("bcrypt");

const saltRounds = 10;

function cryptPassword(password) {
   try {
      return bcrypt.hashSync(password, saltRounds, (err, hash) => {
         if (err) {
            throw Error("Cannot hash password");
         } else {
            return hash;
         }
      });
   } catch (e) {
      console.error(e);
   }
}

function comparePasswords(user, password) {
   return bcrypt.compareSync(password, user.password);
}

module.exports = {
   cryptPassword,
   comparePasswords
};
```

файл auth.ts

```
import {createAction, props} from '@ngrx/store';

export const loginRequest = createAction('[Auth] Login Request', props<{ username: string, password: string }>());
```

```
export const loginSuccess = createAction('[Auth] Login Success', props<{ tokenNode: string, tokenJava: string
}>());
export const loginFailure = createAction('[Auth] Login Failure', props<{ error: any }>());
export const logout = createAction('[Auth] Log out');


import { Injectable } from '@angular/core';
import { Actions, createEffect, ofType } from '@ngrx/effects';
import { of } from 'rxjs';
import {catchError, map, mergeMap} from 'rxjs/operators';
import * as AuthActions from './auth.actions';
import { AuthService } from './auth.service';

@Injectable()
export class AuthEffects {

  login$ = createEffect(() =>
    this.actions$.pipe(
      ofType(AuthActions.loginRequest.type),
      mergeMap((action: {username: string, password: string} ) =>
        this.authService.login(action.username, action.password).pipe(
          map((response) => AuthActions.loginSuccess({tokenNode: response.tokenNode, tokenJava:
response.tokenJava})),
          catchError(error => of(AuthActions.loginFailure(error)))
        )
      )
    )
  );

  constructor(private actions$: Actions, private authService: AuthService) {}
}


import { createReducer, on } from '@ngrx/store';
import * as AuthActions from './auth.actions';

export interface AuthState {
  tokenNode: string | null;
  tokenJava: string | null;
  error: string | null;
}

export const initialState: AuthState = {
  tokenNode: null,
  tokenJava: null,
  error: null,
};

export const authReducer = createReducer(
  initialState,
  on(AuthActions.loginSuccess, (state, { tokenNode, tokenJava }) => ({
    ...state,
    tokenNode,
    tokenJava,
    error: null,
  })),
  on(AuthActions.loginFailure, (state, { error }) => ({
    ...state,
    error: error.message
  })),
  on(AuthActions.logout, (state) => ({
    ...state,
    tokenNode: null,
```

```
    tokenJava: null,
    error: null,
  }))
);

import {HttpClient, HttpHeaders} from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import {ChangePasswordModel} from "../student/student.model";
import {Store} from "@ngrx/store";
import {AuthState} from "./auth.reducer";

@Injectable()
export class AuthService {
  private hostUrl = 'http://localhost:3000';
  private loginUrl = this.hostUrl + '/login';
  private changePasswordUrl = this.hostUrl + '/change-password';

  private disableAccountUrl = 'http://localhost:8080/auth/disable-account';

  constructor(private http: HttpClient, private store: Store<{ auth: AuthState }>) {
    this.store.select("auth")
      .subscribe((auth) => {
        this.javaToken = auth.tokenJava;
      });
  }

  private javaToken: string | null = null;

  login(username: string, password: string): Observable<{message: string, tokenNode: string, tokenJava:string}> {
    return this.http.post<{message: string, tokenNode: string, tokenJava:string}>(this.loginUrl, { login: username,
password: password });
  }

  changePassword(changePasswordModel: ChangePasswordModel) {
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${this.javaToken}`
    });
    const requestOptions = {
      headers: headers
    };
    return this.http.post<{message: string}>(this.changePasswordUrl, changePasswordModel, requestOptions);
  }

  disableAccount() {
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${this.javaToken}`
    });
    const requestOptions = {
      headers: headers
    };
    return this.http.post<{message: string}>(this.disableAccountUrl, {}, requestOptions);
  }
}
```

файл auth-guard.ts

```
import {Injectable} from '@angular/core';
import {
  ActivatedRouteSnapshot,
  CanActivate,
  Router,
  RouterStateSnapshot,
```

```
  UrlTree
} from "@angular/router";
import {Observable} from "rxjs";
import {Store} from "@ngrx/store";
import {AuthState} from "../auth/auth.reducer";
import {loginRequest, logout} from "../auth/auth.actions";

@Injectable()
export class AuthGuardComponent implements CanActivate {

  isAuthenticated: boolean = false;

  constructor(private router: Router, private store: Store<{ auth: AuthState }>) {
    this.store.select("auth")
      .subscribe((auth) => {
        this.isAuthenticated = auth.tokenJava != null
          && auth.tokenNode != null;
      });
  }

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean | UrlTree> |
Promise<boolean | UrlTree> | boolean | UrlTree {
    if (this.isAuthenticated) {
      return true;
    } else {
      console.error("access denied");
      this.router.navigate(["/login"]);
      return false;
    }
  }

  logOut() {
    this.store.dispatch(logout());
    localStorage.clear();
  }
}


import {Injectable} from '@angular/core';
import {
  ActivatedRouteSnapshot,
  CanActivate,
  Router,
  RouterStateSnapshot,
  UrlTree
} from "@angular/router";
import {Observable} from "rxjs";
import {Store} from "@ngrx/store";
import {AuthState} from "../auth/auth.reducer";
import {loginRequest, logout} from "../auth/auth.actions";

@Injectable()
export class AuthGuardLoggedInComponent implements CanActivate {

  isAnonymous: boolean = false;

  constructor(private router: Router, private store: Store<{ auth: AuthState }>) {
    this.store.select("auth")
      .subscribe((auth) => {
        this.isAnonymous = auth.tokenJava == null
          || auth.tokenNode == null;
      });
  }
```

```
  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean | UrlTree> |
Promise<boolean | UrlTree> | boolean | UrlTree {
    if (this.isAnonymous) {
      return true;
    } else {
      console.error("logged in access denied");
      this.router.navigate(["/home"]);
      return false;
    }
  }
}
```

файл diet.ts

```
import {Injectable} from "@angular/core";
import {HttpClient, HttpHeaders} from "@angular/common/http";
import {Store} from "@ngrx/store";
import {AuthState} from "../auth/auth.reducer";
import {Observable} from "rxjs";

@Injectable()
export class DietService {

  private creteDietUrl = 'http://localhost:8080/diet';

  private javaToken: string | null = null;

  constructor(private http: HttpClient, private store: Store<{ auth: AuthState }>) {
    this.store.select("auth")
      .subscribe((auth) => {
        this.javaToken = auth.tokenJava;
      });
  }

  createDiet(createDietDto: any): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${this.javaToken}`
    });
    const requestOptions = {
      headers: headers
    };
    return this.http.post(this.creteDietUrl, createDietDto, requestOptions);
  }

}


import {Component} from '@angular/core';
import {DietService} from "./diet.service";
import {Router} from "@angular/router";

@Component({
  selector: 'app-diet',
  templateUrl: './diet.component.html',
  styleUrls: ['./diet.component.css']
})
export class DietComponent {

  rowData: any = [];
  rowDataAll: any = [];

  columnDataStudent = [
```

```
    {field: 'massInGrams'},
    {field: 'desc'},
    {field: 'proteins'},
    {field: 'fats'},
    {field: 'carbohydrates'},
  ];

  createDietDto: { numberOfMeals: number, kcal: number, proteinsCoefficient: number, fatsCoefficient: number,
carbohydratesCoefficient: number } = {
    numberOfMeals: 5,
    kcal: 2500,
    proteinsCoefficient: 0.4,
    fatsCoefficient: 0.3,
    carbohydratesCoefficient: 0.3
  }

  constructor(private dietService: DietService) {
  }

  onSubmit() {
    this.dietService.createDiet(this.createDietDto)
      .subscribe(response => {
        console.log(response);
        this.rowDataAll = response;
        this.rowData = response[0].dishes;
      });
  }

  invalidCreateDto(): boolean {
    return this.createDietDto.proteinsCoefficient + this.createDietDto.fatsCoefficient +
this.createDietDto.carbohydratesCoefficient !== 1
      || this.createDietDto.kcal > 5000;
  }

  downloadRation() {
    const json = JSON.stringify(this.rowDataAll);
    const blob = new Blob([json], {type: 'application/json'});
    const link = document.createElement('a');
    link.href = URL.createObjectURL(blob);
    link.download = "daily_ration";
    link.click();
  }

}


<form (ngSubmit)="onSubmit()">
  <div>
    <label>Number of Meals:</label>
    <input type="number" name="numberOfMeals" [(ngModel)]="createDietDto.numberOfMeals">
  </div>
  <div>
    <label>Kcal:</label>
    <input type="number" name="kcal" [(ngModel)]="createDietDto.kcal">
  </div>
  <div>
    <label>Proteins Coefficient:</label>
    <input type="number" step="0.01" name="proteinsCoefficient"
[(ngModel)]="createDietDto.proteinsCoefficient">
  </div>
  <div>
    <label>Fats Coefficient:</label>
    <input type="number" step="0.01" name="fatsCoefficient" [(ngModel)]="createDietDto.fatsCoefficient">
  </div>
```

```
  <div>
    <label>Carbohydrates Coefficient:</label>
    <input type="number" step="0.01" name="carbohydratesCoefficient"
[(ngModel)]="createDietDto.carbohydratesCoefficient">
  </div>
  <button type="submit" [disabled]="invalidCreateDto()">Generate ration</button>
</form>

<section>
  <h2>My ration</h2>
  <ag-grid-angular
    style="width: 101%; max-width: 1002px; height: 300px;"
    class="ag-theme-alpine"
    [rowData]="rowData"
    [columnDefs]="columnDataStudent">
  </ag-grid-angular>
  <button [disabled]="rowDataAll.length === 0" (click)="downloadRation()">Download Ration</button>
</section>
```

файл mini-profile.ts

```
import {Component, ElementRef, HostListener} from '@angular/core';
import {Router} from "@angular/router";
import {AuthGuardComponent} from "../auth-guard/auth-guard.component";
import {HeaderComponent} from "../header/header.component";
import {MyProfileService} from "../my-profile/my-profile.service";

@Component({
  selector: 'app-mini-profile',
  templateUrl: './mini-profile.component.html',
  styleUrls: ['./mini-profile.component.css']
})
export class MiniProfileComponent {
  user: { firstName: string, lastName: string, username: string, dateOfBirth: string, address: string, email: string,
specialization: string } = {
    firstName: 'Roma',
    lastName: 'Kolesnyk',
    username: 'kolesroma',
    dateOfBirth: '27-11-2002',
    email: 'koles@roma.com',
    address: 'Shevchenka, 20a',
    specialization: ''
  };

  constructor(private router: Router, private authGuard: AuthGuardComponent, private eRef: ElementRef, private
header: HeaderComponent, private profileService: MyProfileService) {
  }

  isNightMode = false;

  profilePhotoUrl: string = this.profileService.getProfilePhotoOrDefault()

  toggleNightMode() {
    this.isNightMode = !this.isNightMode;
  }

  navigateToMyAccount() {
    this.closeMiniProfile();
    this.router.navigate(['/my-account']);
  }

  logOut() {
    this.closeMiniProfile();
```

```
    this.authGuard.logOut();
    this.router.navigate(['/login']);
  }

  private firstTimeOpenMiniProfile = true;

  @HostListener('document:click', ['$event'])
  clickOutMiniProfile(event: MouseEvent) {
    if (this.firstTimeOpenMiniProfile) {
      this.firstTimeOpenMiniProfile = false;
    } else if (!this.eRef.nativeElement.contains(event.target)) {
      this.closeMiniProfile();
    }
  }

  private closeMiniProfile() {
    this.header.isMiniProfileOpened = false;
    this.firstTimeOpenMiniProfile = true;
  }

}
```

файл my-profile.ts

```
import {Injectable} from "@angular/core";
import {HttpClient, HttpHeaders} from "@angular/common/http";
import {fromEvent, Observable} from "rxjs";
import {Store} from "@ngrx/store";
import {AuthState} from "../auth/auth.reducer";
import {map} from "rxjs/operators";
import {FullUserModel} from "./full-user.model";

@Injectable()
export class MyProfileService {
  private profilePhotoUrl = 'http://localhost:3000/user-photo';
  private uploadPhotoUrl = 'http://localhost:3000/upload-photo';

  private getMeUrl = 'http://localhost:8080/users/me';
  private getUserRoleUrl = 'http://localhost:8080/users/me/role';
  private updateUserUrl = 'http://localhost:8080/users';

  private profilePhotoLocationDefault = 'assets/img/profile-icon.png';

  constructor(private http: HttpClient, private store: Store<{ auth: AuthState }>) {
    this.store.select("auth")
      .subscribe((auth) => {
        this.nodeToken = auth.tokenNode;
        this.javaToken = auth.tokenJava;
      });
  }

  private nodeToken: string | null = null;
  private javaToken: string | null = null;

  getProfilePhoto(): Observable<Blob> {
    const headers = new HttpHeaders({
      'tokenNode': `${this.nodeToken}`
    });
    return this.http.get(this.profilePhotoUrl, {headers: headers, responseType: 'blob'});
  }

  getMe(): Observable<any> {
    const headers = new HttpHeaders({
```

```
    'Authorization': `Bearer ${this.javaToken}`
  });
  const requestOptions = {
    headers: headers
  };
  return this.http.get(this.getMeUrl, requestOptions);
}

getUserRole(): Observable<{ authority: string }> {
  const headers = new HttpHeaders({
    'Authorization': `Bearer ${this.javaToken}`
  });
  const requestOptions = {
    headers: headers
  };
  return this.http.get<{ authority: string }>(this.getUserRoleUrl, requestOptions);
}

updateUser(fullUser: FullUserModel, role: string): Observable<any> {
  const headers = new HttpHeaders({
    'Authorization': `Bearer ${this.javaToken}`
  });
  const requestOptions = {
    headers: headers
  };
  if (role === 'ROLE_TRAINEE') {
    return this.http.put(this.updateUserUrl, {trainee: fullUser}, requestOptions);
  }
  if (role === 'ROLE_TRAINER') {
    return this.http.put(this.updateUserUrl, {trainer: fullUser}, requestOptions);
  }
  throw new Error(`provided role ${role} is not supported`)
}

uploadProfilePhoto(file: File): Observable<any> {
  const headers = new HttpHeaders({
    'tokenNode': `${this.nodeToken}`
  });
  const formData = new FormData();
  formData.append("photo", file);
  const protoResponse = this.http.post(this.uploadPhotoUrl, formData, {headers: headers});
  protoResponse
    .subscribe(() => {
      this.saveProfilePhotoLocally();
    });
  return protoResponse;
}

saveProfilePhotoLocally() {
  this.getProfilePhoto()
    .subscribe(img => {
      this.toBase64(img)
        .subscribe(base64image => {
          localStorage.setItem("user-photo", base64image + "");
        });
    });
}

getProfilePhotoOrDefault(): string {
  const userPhotoBase64 = localStorage.getItem("user-photo");
  if (userPhotoBase64) {
    return userPhotoBase64;
  } else {
```

```
      return this.profilePhotoLocationDefault;
    }
  }

  toBase64(blob: Blob): Observable<string> {
    const reader = new FileReader();
    reader.readAsDataURL(blob);
    return fromEvent(reader, 'load')
      .pipe(map(() => (reader.result as string)))
  }
}


import {Component} from '@angular/core';
import {DialogDeleteAccount, DialogUpdatePhoto, ModalBoxComponent} from "../modal-box/modal-
box.component";
import {MyProfileService} from "./my-profile.service";
import {FullUserModel} from "./full-user.model";
import {ToastrService} from "ngx-toastr";

@Component({
  selector: 'app-my-profile',
  templateUrl: './my-profile.component.html',
  styleUrls: ['./my-profile.component.css']
})
export class MyProfileComponent {
  isEditing: boolean = false;
  role: string = '';

  profilePhotoUrl: string = this.profileService.getProfilePhotoOrDefault()

  user: FullUserModel = {
    firstName: null,
    lastName: null,
    username: null,
    dateOfBirth: null,
    address: null,
    specialization: null
  };

  constructor(private modalBox: ModalBoxComponent, private profileService: MyProfileService, private toastr:
ToastrService) {
    this.profileService.getMe()
      .subscribe(response => {
        this.user.firstName = response.firstName;
        this.user.lastName = response.lastName;
        this.user.username = response.username;
        this.user.dateOfBirth = response.dateOfBirth;
        this.user.address = response.address;
        this.user.specialization = response.specialization;
      });
    this.profileService.getUserRole()
      .subscribe(response => {
        this.role = response.authority;
      });
  }

  openDeleteAccountBox() {
    this.modalBox.openDialog(DialogDeleteAccount, '0ms', '100ms');
  }

  openUpdatePhotoBox() {
    this.modalBox.openDialog(DialogUpdatePhoto, '0ms', '100ms');
  }
```

```
  updateProfile() {
    if (!this.isUserFieldsValid()) {
      return;
    }
    this.toggleEditingMode();
    this.profileService.getUserRole()
      .subscribe(response => {
        this.profileService.updateUser(this.user, response.authority)
          .subscribe(response => {
            this.toastr.success("Profile info was updated", "Success");
          }, error => {
            this.toastr.error("", "Error");
          });
      });
  }

  isUserFieldsValid(): boolean {
    if (this.user.firstName && this.user.firstName.length < 3){
      this.toastr.error("first name should be minimum length 3", "Error");
      return false;
    }
    if (this.user.lastName && this.user.lastName.length < 3){
      this.toastr.error("last name should be minimum length 3", "Error");
      return false;
    }
    if (this.user.address && this.user.address.length < 3){
      this.toastr.error("address should be minimum length 3", "Error");
      return false;
    }
    if (this.user.specialization && this.user.specialization.length < 3){
      this.toastr.error("specialization should be minimum length 3", "Error");
      return false;
    }
    return true;
  }

  toggleEditingMode() {
    this.isEditing = !this.isEditing;
  }
}
```

файл my-trainers.ts

```
import {Component} from '@angular/core';
import {TrainingService} from "../training/training.service";
import {DialogAddTrainer, ModalBoxComponent} from "../modal-box/modal-box.component";

@Component({
  selector: 'app-my-trainers',
  templateUrl: './my-trainers.component.html',
  styleUrls: ['./my-trainers.component.css', '../button/button.component.css']
})
export class MyTrainersComponent {

  rowData: any = [];

  columnData = [
    {field: 'trainerName'},
    {field: 'trainingType'}
  ];

  constructor(private trainingService: TrainingService, private modalBox: ModalBoxComponent,) {
```

```
    this.trainingService.getTrainings()
      .subscribe(response => {
        this.rowData = this.filterUnique(response);
      });
  }

  filterUnique(rowData: any[]): any[] {
    const uniquePairs: any[] = [];

    rowData.forEach(training => {
      const existingPair = uniquePairs.find(pair =>
        pair.trainerName === training.trainerName && pair.trainingType === training.trainingType
      );

      if (!existingPair) {
        uniquePairs.push({
          trainerName: training.trainerName,
          trainingType: training.trainingType,
        });
      }
    });

    return uniquePairs;
  }

  openAddTrainerBox() {
    this.modalBox.openDialog(DialogAddTrainer, '0ms', '100ms');
  }
}
```

```
<div class="trainer-list">
  <h1 class="text-center">My Trainers</h1>
  <app-table [rowData]="rowData"
             [columnDefs]="columnData"
  ></app-table>
  <div class="common-button-container">
    <button class="important"
            (click)="openAddTrainerBox()">Add Trainer
    </button>
  </div>
</div>
```

```
.trainer-list {
  width: 400px;
}

.common-button-container {
  width: 120px;
  margin-top: 28px;
}
```

файл navigation.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-navigation',
  templateUrl: './navigation.component.html',
  styleUrls: ['./navigation.component.css']
})
export class NavigationComponent {
  isHorizontalNavOpened: boolean = false;
```

```
}

<nav id="nav">
  <ul>
    <li><a routerLink="/" routerLinkActive="active">Home</a></li>
    <li><a routerLink="/training" routerLinkActive="active">Training</a></li>
    <li><a routerLink="/contact" routerLinkActive="active">Contact</a></li>
    <li><a routerLink="/my-account" routerLinkActive="active">My account</a></li>
  </ul>
</nav>

@import "src/app/header/header.component.css";
```

файл notification-settings.ts

```
import {Component} from '@angular/core';
import {MyProfileService} from "../my-profile/my-profile.service";
import {NotificationService} from "./notification.service";
import {ToastrService} from "ngx-toastr";

@Component({
  selector: 'app-notification-settings',
  templateUrl: './notification-settings.component.html',
  styleUrls: ['./notification-settings.component.css']
})
export class NotificationSettingsComponent {

  email: string = '';
  status: string = '';

  constructor(private notificationService: NotificationService, private profileService: MyProfileService, private
toastr: ToastrService) {
    this.profileService.getMe()
      .subscribe(response => {
        this.email = response.email;
      });
    this.notificationService.getNotificationStatus()
      .subscribe(response => {
        this.status = response.enabled;
      });
  }

  toggleStatusTo(newStatus: boolean) {
    this.notificationService.toggleNotificationStatus(newStatus)
      .subscribe(response => {
        this.toastr.success("Status was updated to " + newStatus, "Success");
        this.status = newStatus.toString();
      });
  }

}

<h1>Your email is {{email}}</h1>
<h1>Do you receive notifications: {{status}}</h1>

<app-button type="secondary" text="Set true" (click)="toggleStatusTo(true)"></app-button>
<app-button type="secondary" text="Set false" (click)="toggleStatusTo(false)"></app-button>

import {Injectable} from "@angular/core";
import {HttpClient, HttpHeaders, HttpParams} from "@angular/common/http";
import {Store} from "@ngrx/store";
```

```
import {AuthState} from "../auth/auth.reducer";
import {Observable} from "rxjs";
import {MyProfileService} from "../my-profile/my-profile.service";

@Injectable()
export class NotificationService {

  private getNotificationStatusUrl = 'http://localhost:8080/notification/status/me';
  private toggleNotificationStatusUrl = 'http://localhost:8080/notification/status';

  private javaToken: string | null = null;

  constructor(private http: HttpClient, private store: Store<{ auth: AuthState }>, private profileService:
MyProfileService) {
    this.store.select("auth")
      .subscribe((auth) => {
        this.javaToken = auth.tokenJava;
      });
  }

  getNotificationStatus(): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${this.javaToken}`
    });
    const requestOptions = {
      headers: headers
    };
    return this.http.get(this.getNotificationStatusUrl, requestOptions);
  }

  toggleNotificationStatus(newStatus: boolean): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${this.javaToken}`
    });
    let params = new HttpParams();
    params = params.append('status', newStatus.toString());
    const requestOptions = {
      headers: headers,
      params: params
    };
    return this.http.post(this.toggleNotificationStatusUrl, {}, requestOptions);
  }

}
```

файл trainings.ts

```
import {Injectable} from "@angular/core";
import {HttpClient, HttpHeaders} from "@angular/common/http";
import {Store} from "@ngrx/store";
import {AuthState} from "../auth/auth.reducer";
import {BehaviorSubject, Observable} from "rxjs";

@Injectable()
export class TrainingService {
  private getTrainingsUrl = 'http://localhost:8080/trainings';
  private saveTrainingUrl = 'http://localhost:8080/trainings';

  constructor(private http: HttpClient, private store: Store<{ auth: AuthState }>) {
    this.store.select("auth")
      .subscribe((auth) => {
        this.javaToken = auth.tokenJava;
      });
  }
```

```
  private rowDataSubject: BehaviorSubject<any[]> = new BehaviorSubject<any[]>([]);
  rowData$: Observable<any[]> = this.rowDataSubject.asObservable();

  setRowData(rowData: any[]): void {
    this.rowDataSubject.next(rowData);
  }

  getRowData(): Observable<any[]> {
    return this.rowData$;
  }

  private javaToken: string | null = null;

  getTrainings(): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${this.javaToken}`
    });
    const requestOptions = {
      headers: headers
    };
    return this.http.get(this.getTrainingsUrl, requestOptions);
  }

  saveTraining(training: any): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': `Bearer ${this.javaToken}`
    });
    const requestOptions = {
      headers: headers
    };
    return this.http.post(this.saveTrainingUrl, training, requestOptions);
  }
}

import {Component} from '@angular/core';
import {MatFormFieldModule} from "@angular/material/form-field";
import {MatInputModule} from "@angular/material/input";
import {MatSelectModule} from "@angular/material/select";
import {FormsModule} from "@angular/forms";
import {MatDatepickerModule} from "@angular/material/datepicker";
import {MatNativeDateModule} from "@angular/material/core";
import {DatePipe} from "@angular/common";
import {TrainingService} from "./training.service";
import {MyProfileService} from "../my-profile/my-profile.service";
import {Router} from "@angular/router";

@Component({
  selector: 'app-training',
  templateUrl: './training.component.html',
  styleUrls: ['./training.component.css']
})
export class TrainingComponent {
  role: string = '';

  rowData: any = [];

  columnDataStudent = [
    {field: 'trainingDate'},
    {field: 'trainingName'},
    {field: 'trainingType'},
    {field: 'trainerName'},
    {field: 'trainingDuration'},
```

```
  ];

  columnDataTrainer = [
    {field: 'trainingDate'},
    {field: 'trainingName'},
    {field: 'trainingType'},
    {field: 'traineeName'},
    {field: 'trainingDuration'},
  ];

  constructor(private trainingService: TrainingService, private profileService: MyProfileService, private router:
Router) {
    this.profileService.getUserRole()
      .subscribe(response => {
        this.role = response.authority;
      });
    this.trainingService.getRowData().subscribe(rowData => {
      this.rowData = rowData;
    });
  }

  addTrainingStudent() {
    this.router.navigate(["training/add/student"]);
  }

  addTrainingTrainer() {
    this.router.navigate(["training/add/trainer"]);
  }
}

@Component({
  selector: 'app-form-filed-trainings-student',
  templateUrl: 'form-field-trainings-student.html',
  styleUrls: ['training.component.css', '../button/button.component.css'],
  standalone: true,
  imports: [MatFormFieldModule, MatInputModule, MatSelectModule, MatDatepickerModule,
MatNativeDateModule, FormsModule],
})
export class FormFieldTrainingsStudent {
  trainerName: string = '';
  trainingType: string = '';
  dateFrom: string = '';
  dateTo: string = '';

  constructor(private datePipe: DatePipe, private trainingService: TrainingService) {
  }

  reformatDate(dateRaw: string) {
    return this.datePipe.transform(dateRaw, 'yyyy/MM/dd') || '';
  }

  searchTrainingsForStudent() {
    this.trainingService.getTrainings()
      .subscribe(response => {
        this.trainingService.setRowData(this.filterTrainings(response));
      });
  }

  filterTrainings(response: any) {
    if (this.trainerName) {
      response = response.filter((training: any) => (training.trainerName ??= '').toLowerCase() ===
this.trainerName.toLowerCase());
    }
```

```
    if (this.trainingType) {
      response = response.filter((training: any) => (training.trainingType ??= '').toLowerCase() ===
this.trainingType.toLowerCase());
    }
    if (this.dateFrom && this.dateTo) {
      response = response.filter((training: any) => this.isDateInRange(
        new Date(training.trainingDate),
        new Date(this.dateFrom),
        new Date(this.dateTo)
      ));
    }
    return response;
  }

  isDateInRange(date: Date, dateFrom: Date, dateTo: Date): boolean {
    return date >= dateFrom && date <= dateTo;
  }
}

@Component({
  selector: 'app-form-filed-trainings-trainer',
  templateUrl: 'form-field-trainings-trainer.html',
  styleUrls: ['training.component.css', '../button/button.component.css'],
  standalone: true,
  imports: [MatFormFieldModule, MatInputModule, MatSelectModule, MatDatepickerModule,
MatNativeDateModule, FormsModule],
})
export class FormFieldTrainingsTrainer {
  studentName: string = '';
  trainingType: string = '';
  dateFrom: string = '';
  dateTo: string = '';

  constructor(private datePipe: DatePipe, private trainingService: TrainingService) {
  }

  reformatDate(dateRaw: string) {
    return this.datePipe.transform(dateRaw, 'yyyy/MM/dd') || '';
  }

  searchTrainingsForTrainer() {
    this.trainingService.getTrainings()
      .subscribe(response => {
        this.trainingService.setRowData(this.filterTrainings(response));
      });
  }

  filterTrainings(response: any) {
    if (this.studentName) {
      response = response.filter((training: any) => (training.traineeName ??= '').toLowerCase() ===
this.studentName.toLowerCase());
    }
    if (this.trainingType) {
      response = response.filter((training: any) => (training.trainingType ??= '').toLowerCase() ===
this.trainingType.toLowerCase());
    }
    if (this.dateFrom && this.dateTo) {
      response = response.filter((training: any) => this.isDateInRange(
        new Date(training.trainingDate),
        new Date(this.dateFrom),
        new Date(this.dateTo)
      ));
    }
```

```
    return response;
  }

  isDateInRange(date: Date, dateFrom: Date, dateTo: Date): boolean {
    return date >= dateFrom && date <= dateTo;
  }
}
```

файл app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';
import { AgGridModule } from 'ag-grid-angular';
import { RouterModule } from '@angular/router';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { ToastrModule } from 'ngx-toastr';

import { AppComponent } from './app.component';
import { LoginComponent } from './login/login.component';
import {CommonModule, DatePipe} from "@angular/common";
import { RoleSelectionComponent } from './role-selection/role-selection.component';
import { RegistrationStudentComponent } from './registration-student/registration-student.component';
import { RegistrationTrainerComponent } from './registration-trainer/registration-trainer.component';
import { HeaderComponent } from './header/header.component';
import { FooterComponent } from './footer/footer.component';
import { ButtonComponent } from './button/button.component';
import { JoinUsBoxComponent } from './join-us-box/join-us-box.component';
import { BoxComponent } from './box/box.component';
import { MyProfileComponent } from './my-profile/my-profile.component';
import { TableComponent } from './table/table.component';
import { NavigationComponent } from './navigation/navigation.component';
import { BreadcrumbComponent } from './breadcrumb/breadcrumb.component';
import { ModalBoxComponent } from './modal-box/modal-box.component';
import {MatButtonModule} from "@angular/material/button";
import { ToasterComponent } from './toaster/toaster.component';
import { DatePickerComponent } from './date-picker/date-picker.component';
import {MatInputModule} from "@angular/material/input";
import {MatDatepickerModule} from "@angular/material/datepicker";
import { MiniProfileComponent } from './mini-profile/mini-profile.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';
import { HomeComponent } from './home/home.component';
import { RegistrationVerificationComponent } from './registration-verification/registration-verification.component';
import {
  FormFieldTrainingsStudent,
  FormFieldTrainingsTrainer,
  TrainingComponent
} from './training/training.component';
import { ChangePasswordComponent } from './change-password/change-password.component';
import { AuthGuardComponent } from './auth-guard/auth-guard.component';
import {StudentPageComponent} from "./student/student.component";
import {HttpClientModule} from "@angular/common/http";
import {ActionReducer, ActionReducerMap, MetaReducer, StoreModule} from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import {StudentEffects} from "./student/student.effects";
import {StudentService} from "./student/student.service";
import {studentListReducer, studentRegistrationReducer} from "./student/student.reducer";
import {AuthService} from "./auth/auth.service";
import {authReducer} from "./auth/auth.reducer";
import { localStorageSync } from 'ngrx-store-localstorage';
import {AuthEffects} from "./auth/auth.effects";
import {AuthGuardLoggedInComponent} from "./auth-guard/auth-guard-logged-in.component";
import {TrainerEffects} from "./trainer/trainer.effects";
import {trainerRegistrationReducer} from "./trainer/trainer.reducer";
```

```
import {TrainerService} from "./trainer/trainer.service";
import {MyProfileService} from "./my-profile/my-profile.service";
import {MatSelectModule} from "@angular/material/select";
import {TrainingService} from "./training/training.service";
import { TrainingAddStudentComponent } from './training-add-student/training-add-student.component';
import { TrainingAddTrainerComponent } from './training-add-trainer/training-add-trainer.component';
import { MyTrainersComponent } from './my-trainers/my-trainers.component';
import { DietComponent } from './diet/diet.component';
import {DietService} from "./diet/diet.service";
import { NotificationSettingsComponent } from './notification-settings/notification-settings.component';
import {NotificationService} from "./notification-settings/notification.service";

export function localStorageSyncReducer(reducer: ActionReducer<any>): ActionReducer<any> {
  return localStorageSync({ keys: ['isAuthenticated', 'auth'], rehydrate: true })(reducer);
}
const metaReducers: Array<MetaReducer<any, any>> = [localStorageSyncReducer];

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    RoleSelectionComponent,
    RegistrationStudentComponent,
    RegistrationTrainerComponent,
    HeaderComponent,
    FooterComponent,
    ButtonComponent,
    JoinUsBoxComponent,
    BoxComponent,
    MyProfileComponent,
    TableComponent,
    NavigationComponent,
    BreadcrumbComponent,
    ToasterComponent,
    MiniProfileComponent,
    PageNotFoundComponent,
    HomeComponent,
    RegistrationVerificationComponent,
    TrainingComponent,
    ChangePasswordComponent,
    StudentPageComponent,
    TrainingAddStudentComponent,
    TrainingAddTrainerComponent,
    MyTrainersComponent,
    DietComponent,
    NotificationSettingsComponent,
  ],
  imports: [
    BrowserModule,
    FormsModule,
    CommonModule,
    AgGridModule,
    RouterModule.forRoot([
      {path: '', component: HomeComponent},
      {path: 'home', component: HomeComponent},
      {path: 'login', component: LoginComponent, canActivate: [AuthGuardLoggedInComponent]},
      {path: 'my-account', component: MyProfileComponent, canActivate: [AuthGuardComponent]},
      {path: 'training', component: TrainingComponent, canActivate: [AuthGuardComponent]},
      {path: 'join-us', component: JoinUsBoxComponent},
      {path: 'change-password', component: ChangePasswordComponent, canActivate: [AuthGuardComponent]},
      {path: 'registration', component: RoleSelectionComponent, canActivate: [AuthGuardLoggedInComponent]},
      {path: 'registration/student', component: RegistrationStudentComponent, canActivate:
[AuthGuardLoggedInComponent]},
```

```
    {path: 'registration/trainer', component: RegistrationTrainerComponent, canActivate:
[AuthGuardLoggedInComponent]},
    {path: 'training/add/student', component: TrainingAddStudentComponent, canActivate:
[AuthGuardComponent]},
    {path: 'training/add/trainer', component: TrainingAddTrainerComponent, canActivate:
[AuthGuardComponent]},
    {path: 'registration-verification', component: RegistrationVerificationComponent, canActivate:
[AuthGuardComponent]},
    {path: 'diet', component: DietComponent, canActivate: [AuthGuardComponent]},
    {path: 'notification-settings', component: NotificationSettingsComponent, canActivate:
[AuthGuardComponent]},
    {path: '**', component: PageNotFoundComponent}
   ]),
   MatButtonModule,
   ModalBoxComponent,
   BrowserAnimationsModule,
   ToastrModule.forRoot(),
   MatInputModule,
   MatDatepickerModule,
   DatePickerComponent,
   HttpClientModule,
   StoreModule.forRoot(
    {
     students: studentListReducer,
     studentRegistration: studentRegistrationReducer,
     trainerRegistration: trainerRegistrationReducer,
     auth: authReducer
    },
    {metaReducers}
   ),
   EffectsModule.forRoot([
    StudentEffects,
    TrainerEffects,
    AuthEffects
   ]),
   MatSelectModule,
   FormFieldTrainingsStudent,
   FormFieldTrainingsTrainer
  ],
  providers: [
   AuthGuardComponent,
   AuthGuardLoggedInComponent,
   StudentService,
   TrainerService,
   AuthService,
   ModalBoxComponent,
   MyProfileService,
   DatePipe,
   TrainingService,
   DietService,
   NotificationService
  ],
  exports: [
   ButtonComponent,
   ButtonComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

файл index.html

```
<!doctype html>
<html lang="en">
```

```
<head>
  <meta charset="utf-8">
  <title>Angular</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення для підтримки діяльності спортивного комплексу

Програма та методика тестування

КПІ.ІТ-0111.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

Виконавець:
_____ Роман КОЛЕСНИК

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є розроблена програмне забезпечення для підтримки діяльності спортивного комплексу. Програмне забезпечення доступне на всіх платформах, що мають підтримку веб-брауера.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

- перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;
- перевірка збереження даних;
- перевірка сумісності веб-додатку з останніми версіями сучасних браузерів (Chrome, Opera, Firefox, …);
- перевірка сумісності застосунку з різними операційними системами (Windows, Linux, …).

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

- статичне тестування перевіряється програма разом з усією документацією, яка аналізується на предмет дотримання стандартів програмування;
- функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;
- мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення.

Обрано типи тестування виходячи з потреб та обставин проекту. Sonar Cloud дозволяє перевірити код на наявність критичних проблем. Статичне тестування допомагає забезпечити відповідність стандартам програмування. За мету поставлено протестувати ключові особливості програмного забезпечення. Функціональне тестування та мануальне тестування гарантують коректну роботу програми з урахуванням реальних потреб користувачів та особливостей проекту.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з використанням наскрізного тестування (End-to-end, E2E, Chain testing) та написанням unit-тестів, з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

- тестування на пристроях з різною роздільною здатністю екрану;
- тестування на виведення повідомлень про помилку, коли це необхідно;
- тестування зміни орієнтації екрану;
- тестування інтерфейсу користувача.

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

Програмне забезпечення для підтримки діяльності спортивного комплексу

Керівництво користувача

КПІ.ІТ-0111.045440.05.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Олена ХАЛУС

Нормоконтроль:

_____ Ірина ВІТКОВСЬКА

Виконавець:

_____ Роман КОЛЕСНИК

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

Метою програмного забезпечення для підтримки діяльності спортивного комплексу є покращення інструментів для обслуговування клієнтів та створення дієти за рахунок створення програмного забезпечення. Дієта є важливим чинником в спортивному прогресі. Це сприяє досягнунню спортивних цілей.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Програмне забезпечення повинно функціонувати на десктопах

Мінімальна конфігурація технічних засобів:

– тип процесору: Intel Core i3;

– об‘єм ОЗП: 2 Гб;

– підключення до мережі Інтернет зі швидкістю від 20 мегабіт.

Рекомендована конфігурація технічних засобів

– тип процесору: Intel Core i3;

– об‘єм ОЗП: 4 Гб;

– підключення до мережі Інтернет зі швидкістю від 100 мегабіт.

2.2 Завантаження застосунку

Технічні деталі:

- наш серверний застосунок використовує Java 17 і фреймворк SpringBoot для розробки веб-застосунків, для відображення користувацького інтерфейсу використовується Angular, для middleware використовується Node.js;
- щоб запустити серверну програму локально, завантажте та налаштуйте java на своєму ПК. Це можна перевірити, виконавши команду java --version у cmd;
- щоб запустити серверну програму локально, завантажте та налаштуйте angular на своєму ПК. Це можна перевірити, виконавши команду ng --version у cmd;
- щоб запустити серверну програму локально, завантажте та налаштуйте node на своєму ПК. Це можна перевірити, виконавши команду node --version у cmd.

Кроки виконання:

- відкрийте CLI за допомогою cmd;
- перейдіть до папки, куди ви завантажили архів .jar;
- напишіть java -jar {шлях_до папки}/{назва_архіву}.jar;
- напишіть node index.js;
- напишіть ng serve.

2.3 Перевірка коректної роботи

Переконайтеся, що серверна програма працює:

- відкрийте браузер і перейдіть на http://localhost:4200/ ;
- ваш локальний сервер буде розгорнуто на порту 4200 локального хосту.

3 ВИКОНАННЯ ПРОГРАМИ

При запуску програмного застосунку користувачу буде відображено два поля для вводу логіну та паролю (див. рисунок 3.1).

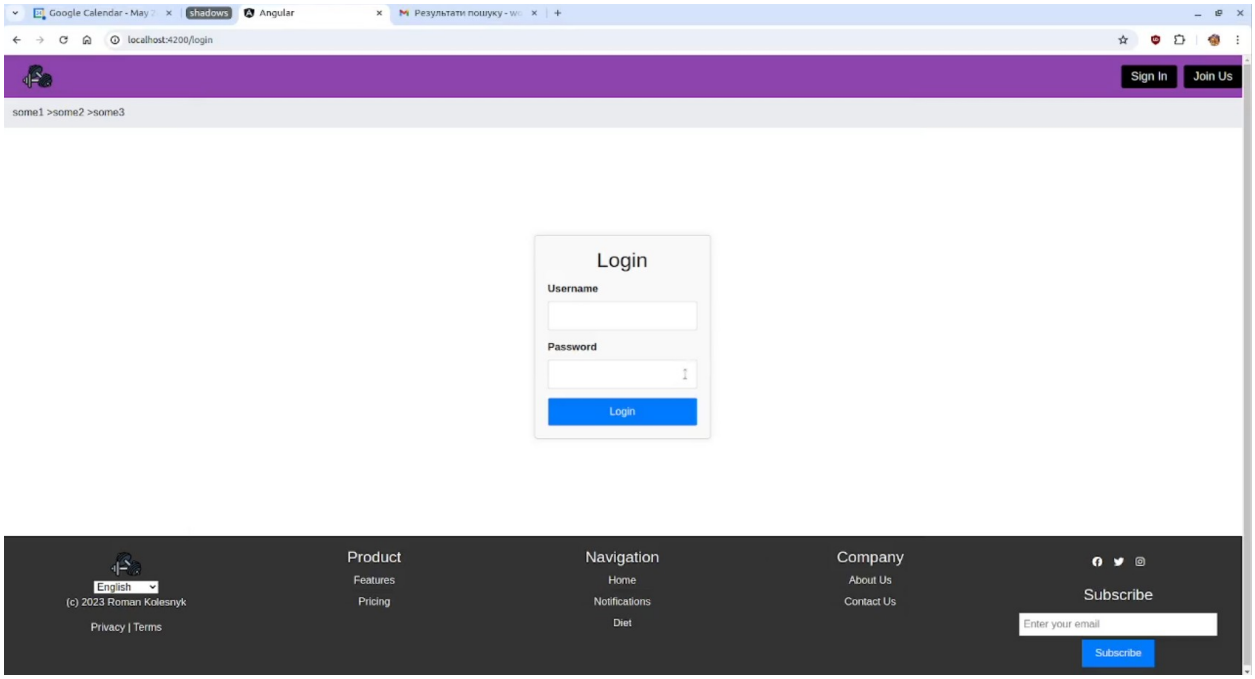


Рисунок 3.1 – Сторінка авторизації

За неуспішної авторизації буде перенаправлено на сторінку з помилкою.

За успішної авторизації користувач отримує доступ до ресурсів сервісу і перенаправляєтьсяна головну сторінку.

На рисунку 3.2 наведено домашню сторінку.

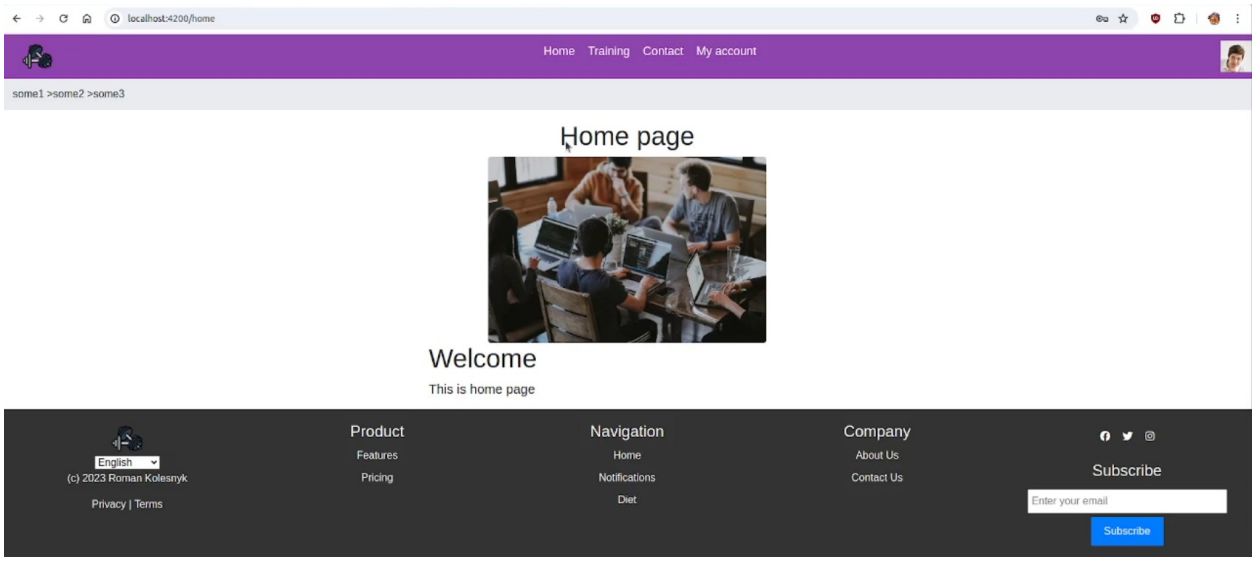


Рисунок 3.2 – Домашня сторінка застосунку

Для навігації по сайту доцільно використовувати хедер та футер. На рисунку 3.3 наведено особистий кабінет користувача. Перейшовши в особистий кабінет, можна переглянути інформацію про поточного користувача, його тренерів та змінити особисту інформацію. Можна оновити дані про себе.

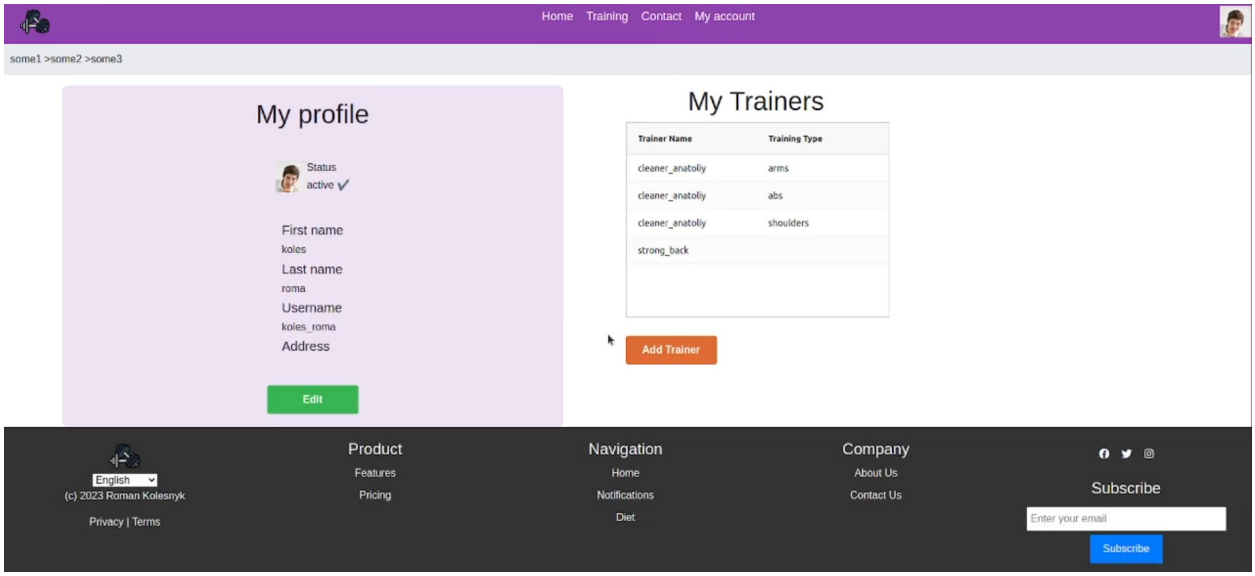


Рисунок 3.3 – Особистий кабінет користувача

Модальне вікно видалення облікового запису наведено на рисунку 3.4.

Можна видалити обліковий запис, при цьому буде попереджувальне модальне вікно.

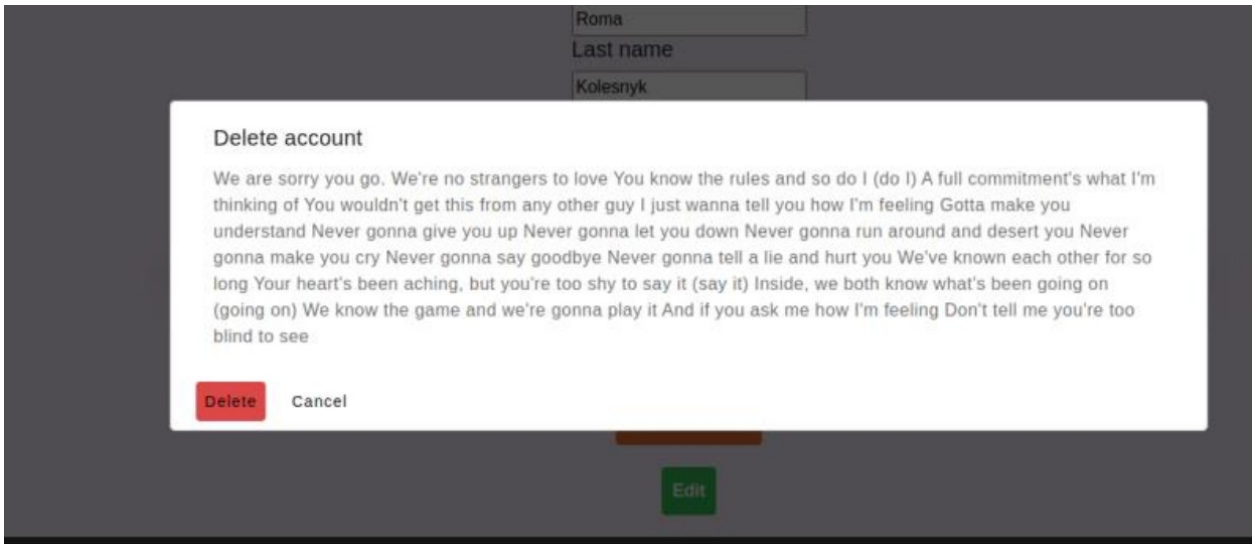


Рисунок 3.4 – Особистий кабінет користувача

Перехід на сторінку тренінгів наведено на рисунку 3.5.

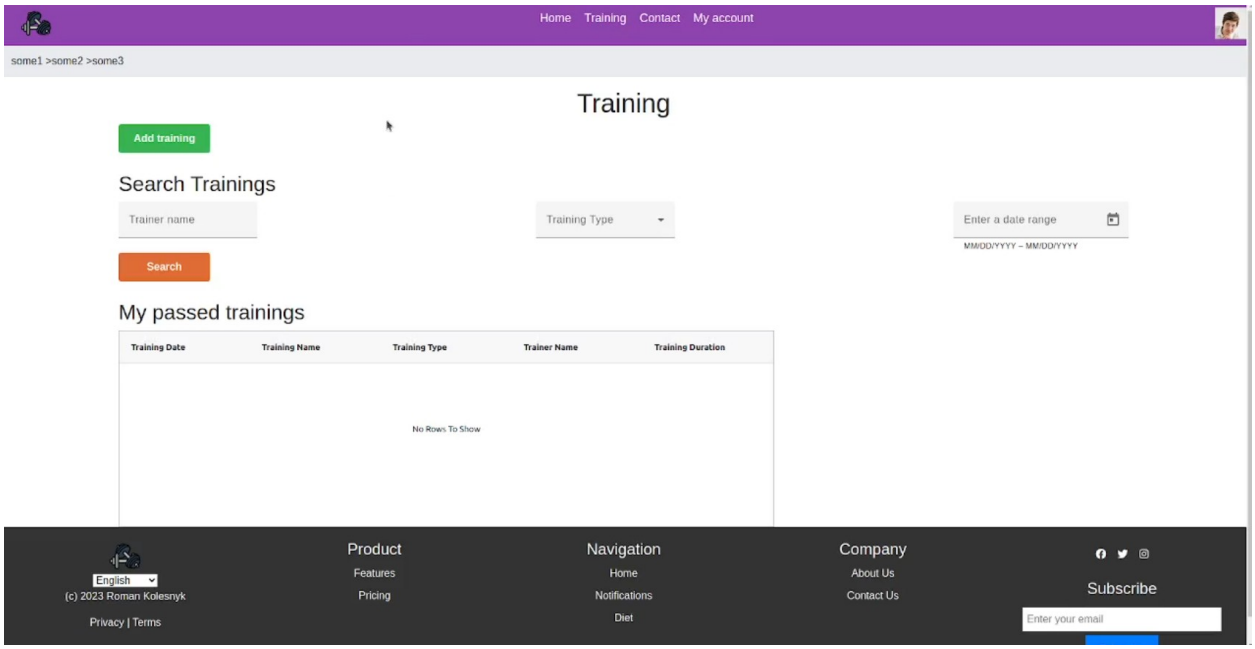


Рисунок 3.5 – Сторінка тренінгів

Створення нового тренінга наведено на рисунку 3.6.

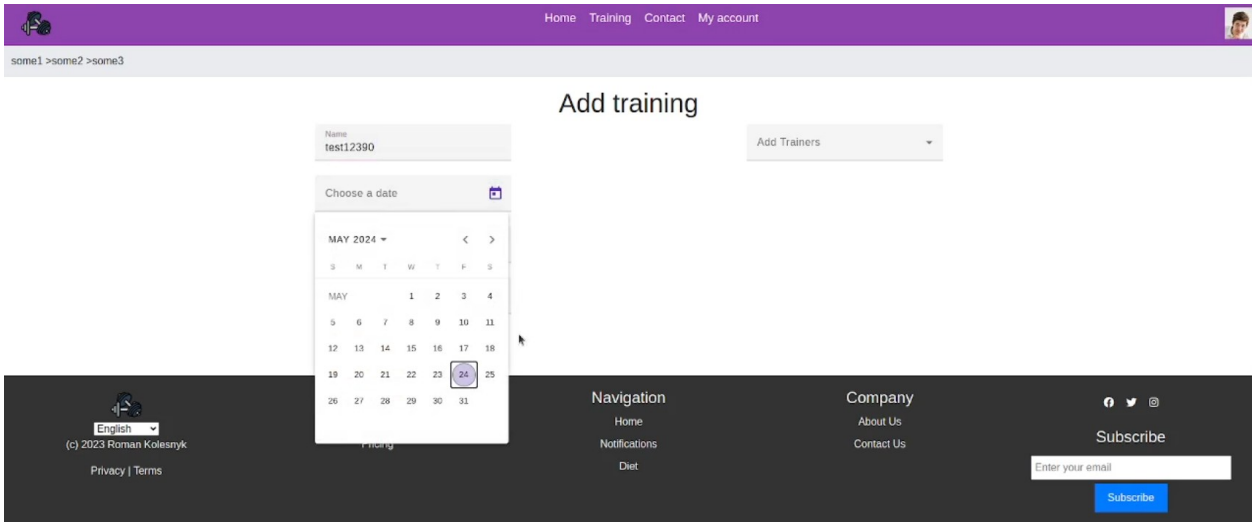


Рисунок 3.6 – Створення тренінга

При подіях в системі відбувається отримання повідомлення на пошту, наприклад при записі на тренування (див. рисунок 3.7).

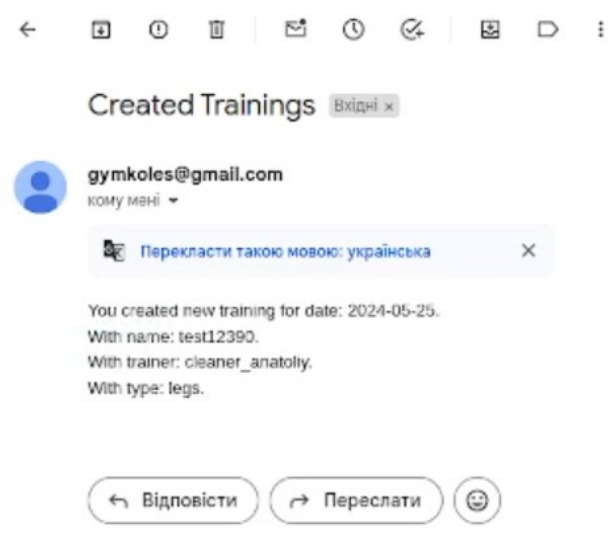


Рисунок 3.7 – Отримання повідомлень

Повідомлення можна налаштувати на сторінці налаштування сповіщень (див. рисунок 3.8).

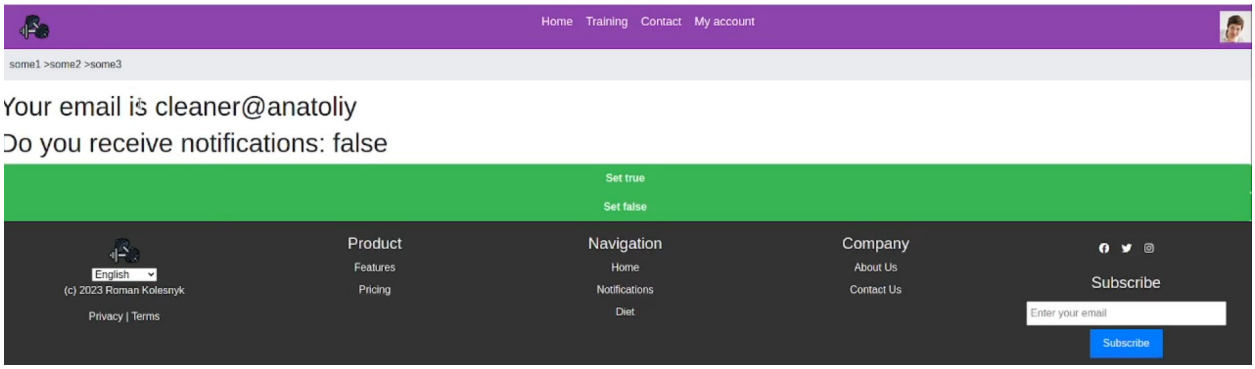


Рисунок 3.8 – Налатшування повідомлень

На сторінці генерації дієти можна поставити мету тренувань, вказати кількість білків, жирів та вуглеводів і кількість кілокалорій на день. Тренер може допомогти з пропорціями (див. рисунок 3.9).

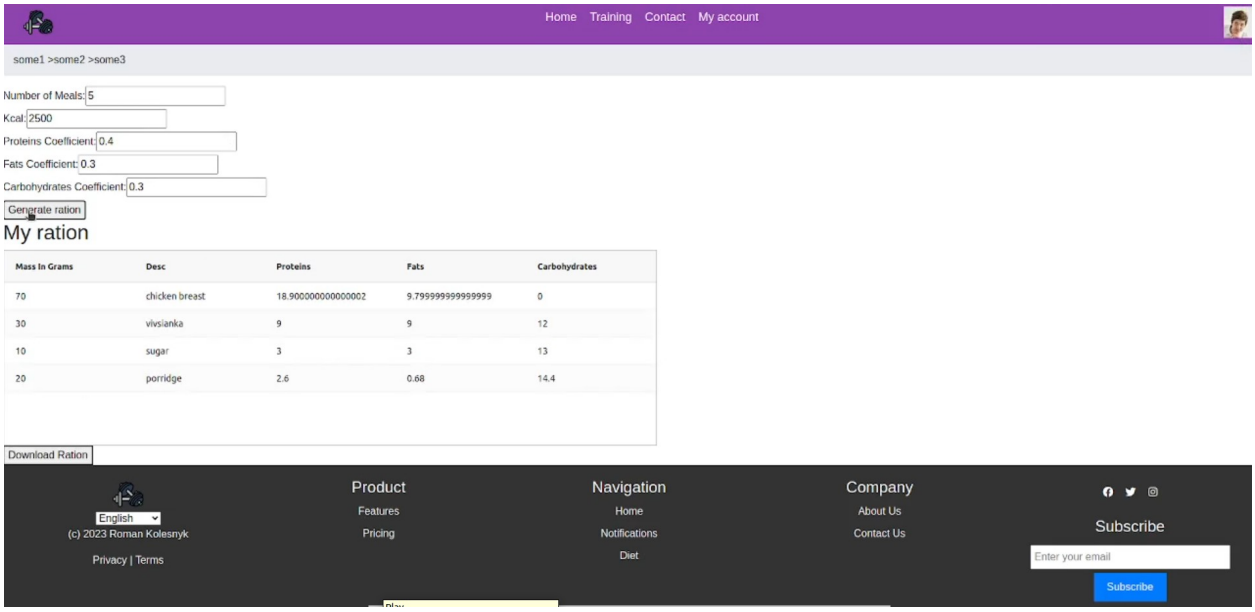


Рисунок 3.9 – Генерація дієти

Є можливість завантажити дієту.

Також про неправильні дії користувача передбачені помилки (див. рисунок 3.10).

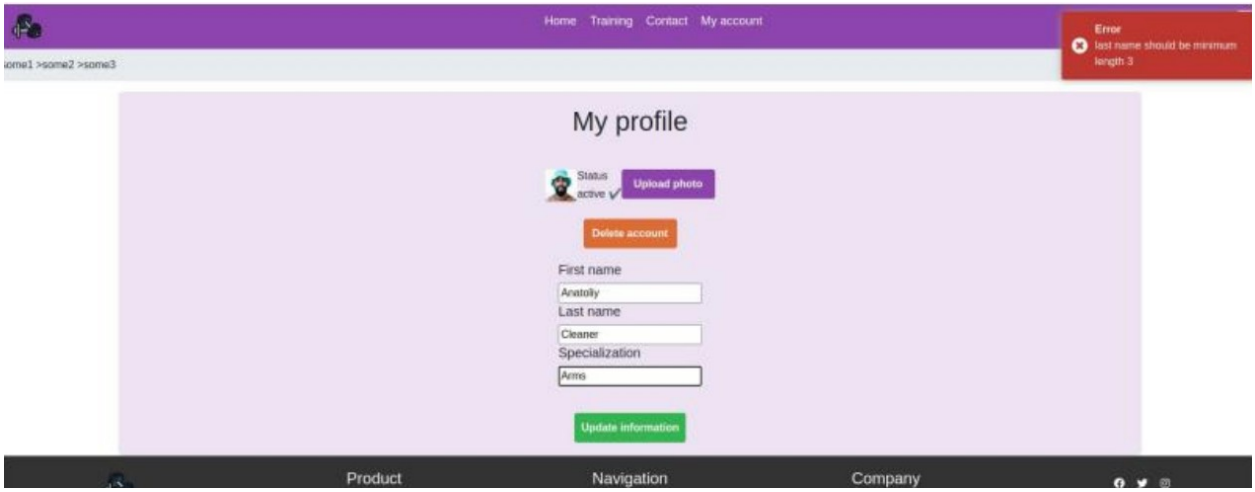


Рисунок 3.10 – Оновлення даних, помилка — мінімальна довжина слова повинна бути три символи

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

Програмне забезпечення для підтримки діяльності спортивного комплексу

Графічний матеріал

КПІ.ІТ-0111.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Олена ХАЛУС

Нормоконтроль:
_____ Ірина ВІТКОВСЬКА

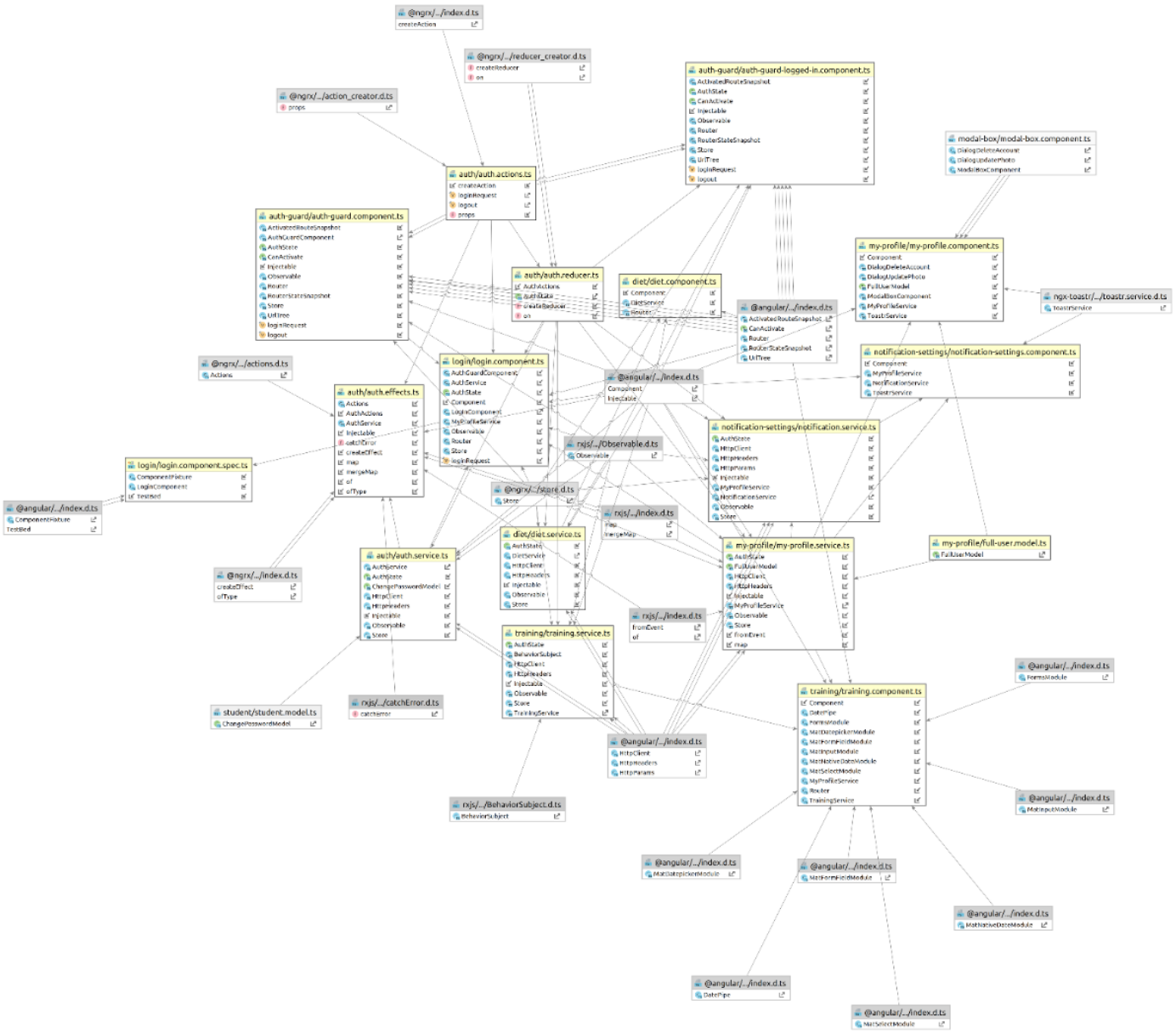
Виконавець:
_____ Роман КОЛЕСНИК

Київ – 2024

КПІ.ІТ-0111.045440.06.99.СБД

					КПІ.ІТ-0111.045440.06.99.СБД
Зм.	Арк.	№ докум.	Підп.	Дата	Схема бази даних
Розробив		Колесник Р.Р.			Лит. / Маса / Масштаб
Перевірив		Халус О.А.			Аркуш 1 / Аркушів 3
Т. контр.					Програмне забезпечення для підтримки діяльності спортивного комплексу
Н. контр.		Вітковська І.І.			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-01
Затвердив		Жаріков Е.В.			

КПІ.ІТ-0111.045440.06.99.ССК



					КПІ.ІТ-0111.045440.06.99.ССК		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розробив		Колесник Р.Р.			Схема структурна класів програмного забезпечення	Лит.	Маса / Масштаб
Перевірив		Халус О.А.					
Т. контр.						Аркуш 2	Аркушів 3
Н. контр.		Вітковська І.І.			Програмне забезпечення для підтримки діяльності спортивного комплексу	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-01	
Затвердив		Жаріков Е.В.					

КПІ.ІТ-0111.045440.06.99.ССК

					КПІ.ІТ-0111.045440.06.99.ССК		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розробив		Колесник Р.Р.			Схема структурна класів програмного забезпечення	Лит.	Маса / Масштаб
Перевірив		Халус О.А.					
Т. контр.						Аркуш 3	Аркушів 3
Н. контр.		Вітковська І.І.			Програмне забезпечення для підтримки діяльності спортивного комплексу	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІТ-01	
Затвердив		Жаріков Е.В.					