Вінницький національний технічний університет/
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Бакалаврська дипломна робота на тему: «Інтерактивний вебдодаток Training Club»

Виконав: студент 4 курсу, групи КІ-22мсз
спеціальності 123 – Комп'ютерна інженерія
_____ Ткачук М. О.

Керівник: к.т.н., доц., доцент каф. ОТ
_____ Черняк О. І.
«12» 06 2024 р.

Рецензент: к.т.н., доц., доцент каф. ПЗ
_____ Хошаба О. М.
«14» 06 2024 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф Азаров О.Д.
"12" 06 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр Галузь
знань – 12 «Інформаційні технології» Спеціальність
– 123 «Комп'ютерна інженерія»
Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф. Азаров О. Д.
«23» квітня 2024 року

З А В Д А Н Н Я

НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Ткачуку Максиму Олександровичу

1 Тема роботи: «Інтерактивний вебдодаток Training Club», керівник роботи Черняк О. І. к.т.н., доц., доцент кафедри ОТ затверджені наказом вищого навчального закладу від «11» березня 2024 року № 80.

2 Термін подання студентом роботи 17.06.2024 р.

3 Вихідні дані до роботи: React, Nest js, TypeScript, монолітний back end сервіс, Github, CI/CD, MongoDB.

4 Зміст розрахунково-пояснювальної записки: вступ, аналіз сучасних веб технологій для реалізації вебдодатків, розробка вебдодатку, розробка API, висновки, перелік посилань, додатки.

5 Перелік текстового матеріалу — лістинг програми.

6 Консультанти розділів роботи в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ 1-4	к.т.н., доц., доцент кафедри ОТ Черняк О.І.		

7 Дата видачі завдання 12.03.2024 р. Календарний план етапів роботи наведено в таблиці 2.

Таблиця 2 — Календарний план

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Вибір, узгодження та затвердження теми БДР	03.02.2024	10.02. 2024	виконано
2.	Аналіз літературних джерел. Попередня розробка основних розділів	15.02. 2024	01.03. 2024	виконано
3.	Розробка технічного завдання (ТЗ)	05.03. 2024	06.03. 2024	виконано
4.	Аналіз вирішення поставленої задачі на даному етапі	07.03. 2024	19.03. 2024	виконано
5.	Реалізація веб додатку	19.03. 2024	30.04. 2024	виконано
6.	Оформлення пояснювальної записки та графічної частини	02.05. 2024	17.05. 2024	виконано
7.	Нормоконтроль	20.05. 2024	20.05. 2024	виконано
8.	Попередній захист БДР, доопрацювання, рецензування БДР	20.05. 2024	12.06. 2024	виконано
9.	Захист БДР ЕК	17.06. 2024	17.06. 2024	виконано

Студент _____ Ткачук М. О.

Керівник роботи _____ Черняк О. І.

АНОТАЦІЯ

УДК 004.4

Ткачук М. О. Інтерактивний вебдодаток Training Club. Бакалаврська дипломна робота зі спеціальності 123 «Комп`ютерна інженерія», освітня програма «Комп`ютерна інженерія».Вінниця: ВНТУ, 2024. 93с.

На укр. мові. Бібліогр.: 18 назв; рис.: 18.

У бакалаврській дипломній роботі було розроблено інтерактивний вебдодаток Training Club, який надасть можливість користувачам займатися активною фізичною діяльністю через індивідуальні тренування, розроблені професійними тренерами. В основній частині роботи проаналізовано сучасні технології для реалізації вебдодатку, опис реалізації вебдодатку, опис реалізації серверної частини та опис налаштування CI/CD для розгортання сервера на VPS.

Ключові слова: Nginx, React, Nest js, CORS, CI/CD, VPS.

ABSTRACT

Tkachuk M. O. Training Club interactive web application. Bachelor thesis on specialty 123 "Computer Engineering", educational program "Computer Engineering". Vinnytsia: VNTU, 2024. 93p.

In Ukrainian speech Bibliography: 18 titles; Fig.: 18.

In the bachelor's thesis, an interactive web application Training Club was developed, which will provide the opportunity for users to engage in active physical activity through individual training sessions developed by professional trainers. In the main part of the work, modern technologies for the implementation of the web application, the description of the implementation of the web application, the description of the implementation of the server part and the description of the CI/CD configuration for deploying the server on a VPS are analyzed.

Keywords: Nginx, React, Nest js, CORS, CI/CD, VPS.

ЗМІСТ

ВСТУП

В сучасному світі здоров'я та фізична активність стали невід'ємною частиною нашого щоденного життя. Запит на зручні та доступні інструменти для підтримки активного способу життя та віртуальні рішення у сфері фітнесу дедалі більше завойовують популярність. В цьому контексті, розробка інтерактивного вебдодатку Training Club є актуальною ідеєю.

Актуальність роботи полягає в розробці фітнес-платформи, що дозволяє інтеграцію різних типів контенту та віддалених тренерів, а також їх вибору користувачами.

Метою дипломної роботи є інтеграція різноманітного фітнес-контенту і тренерів для надання вільного вибору методик фізичних тренувань. Вебдодаток Training Club створений для широкого кола аудиторії, надаючи доступ до багатьох різноманітних вправ та можливість створення власних тренувальних програм.

Однією з ключових функцій цього вебдодатку є можливість тренерам розробляти власні комплекси вправ та ділитися ними з іншими користувачами. Це дозволить тренерам виявити свою креативність та рівень професійності, а також надихнути користувачів на досягнення своїх фітнес–цілей. Кожен користувач зможе знайти комплекс тренувань, які відповідають їхнім потребам і можливостям, завдяки різноманітним фільтрам та персоналізованим рекомендаціям. Все це сприяє популяризації активного способу життя та збереженню здоров'я серед різних верств населення. Крім того проект має забезпечити поєднання сучасних технологій та високого рівня фаховості тренерів.

Для досягнення поставленої мети створення інтерактивного вебдодатку Training Club із зазначеними функціями, необхідно виконати наступні задачі:

1) проаналізувати сучасні веб-технології;
— провести загальний огляд веб–технологій;
— провести аналіз використаних технологій;
2) виконати порівняльну характеристику фітнес додатків;
3) розробити вимоги до структури вебдодатку;

4) розробити клієнтську частину;

— реалізувати навігацію по вебдодатку;

— налаштувати логіку push-повідомлень;

— написати тести;

5) розробити серверну частину.

— вибрати архітектуру проекту;

— вибрати та налаштувати базу даних;

— налаштувати CI/CD;

— написати тести.

Об'єктом дослідження є процес створення інтерактивного вебдодатку для фітнесу, який включає в себе розробку дизайну, розробку клієнтської частини та її інтеграцію з серверною частиною вебдодатку.

Предметом дослідження є методи та інструменти розробки вебдодатків, зокрема, інтерфейс користувача, алгоритми персоналізації тренувань та засоби забезпечення безпеки даних користувачів.

В процесі роботи використовувалися такі методи дослідження, як аналіз літературних джерел та сучасних трендів у сфері фітнесу, розробка прототипів та тестування вебдодатку, а також збір та аналіз відгуків користувачів для покращення функціоналу.

АНАЛІЗ СУЧАСНИХ ВЕБ–ТЕХНОЛОГІЙ

1.1 Загальний огляд веб–технологій

Веб технології — це фундамент, на якому будується сучасний інтернет. Вони включають у себе мови програмування, інструменти та протоколи, які використовуються для розробки веб-сайтів та додатків. Загальний огляд веб технологій охоплює широкий спектр тем, від базових HTML-сторінок до складних веб-додатків, що використовують JavaScript, CSS, різноманітні фреймворки та бібліотеки.

На зорі інтернету веб-сторінки були статичними, але з часом вони стали динамічними та інтерактивними, завдяки розвитку технологій. HTML (HyperText Markup Language) залишається основою веб-контенту, дозволяючи структурувати інформацію та забезпечувати основу для стилізації та інтерактивності [1]. CSS (Cascading Style Sheets) використовується для стилізації веб-сторінок, дозволяючи розробникам створювати візуально привабливі дизайни [2]. JavaScript, як мова програмування вебу, дозволяє створювати інтерактивні елементи та динамічні веб-додатки [3].

Сучасні веб технології також включають різноманітні фреймворки та бібліотеки, такі як React, Angular та Vue.js [4,5,6], які спрощують процес розробки та підвищують продуктивність. Вони надають готові компоненти та інструменти для швидкого розгортання веб-додатків.

Безпека інтернету є ще однією критичною темою. Розробники повинні бути обізнані з потенційними загрозами, такими як XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) та інші види атак. Використання HTTPS та SSL/TLS сертифікатів є стандартом для забезпечення безпечного обміну даними.

У майбутньому веб технології продовжать розвиватися, інтегруючи новітні досягнення у сфері штучного інтелекту, машинного навчання та інших передових областей. Це відкриває безмежні можливості для створення ще більш інтелектуальних та зручних веб-додатків. Веб–технології залишаються в центрі інновацій, що формують спосіб, яким ми взаємодіємо з цифровим світом.

1.1.1 Клієнтські технології

Клієнтські технології — це інструменти та мови програмування, які використовуються для створення та інтеграції з користувацьким інтерфейсом веб–додатків. Вони визначають, як веб–сторінки відображаються та взаємодіють з користувачами. Основними клієнтськими технологіями є HTML, CSS та JavaScript, які разом формують тріаду веб-розробки.

HTML (HyperText Markup Language) є основою будь-якої веб-сторінки. Він використовується для структурування контенту та елементів на сторінці. З появою HTML5, стандарту HTML, введено нові функції, такі як відео та аудіо теги, які дозволяють вбудовувати мультимедійний контент безпосередньо у веб-сторінки.

CSS (Cascading Style Sheets) відповідає за візуальний стиль веб-сторінок. Він дозволяє розробникам створювати багаті та адаптивні дизайни, які можуть автоматично адаптуватися до різних пристроїв та розмірів екранів. CSS3, остання версія CSS, включає розширені можливості, такі як анімації, переходи та гнучкі макети (flexbox).

JavaScript є мовою програмування, яка додає інтерактивність до веб-сторінок. Вона дозволяє розробникам створювати динамічні веб-додатки, які можуть реагувати на дії користувачів у реальному часі. Сучасні JavaScript фреймворки, такі як React, Angular та Vue.js, надають потужні інструменти для побудови складних користувацьких інтерфейсів.

Ці технології разом створюють основу для розробки веб-додатків, які є красивими, швидкими та інтерактивними. Вони дозволяють розробникам створювати досвід, який відповідає очікуванням сучасних користувачів інтернету, які шукають швидкі та зручні веб-сервіси.

З розвитком технологій, клієнтська сторона веб-розробки продовжує еволюціонувати. Наприклад, Progressive Web Apps (PWA) використовують сучасні веб-технології для створення додатків, які працюють як нативні додатки на мобільних пристроях. WebAssembly відкриває можливості для запуску коду, написаного на інших мовах програмування, безпосередньо у веб-браузері, що розширює можливості веб-додатків.

Клієнтські технології продовжують бути важливою частиною веб-розробки, оскільки вони безпосередньо впливають на користувацький досвід та задоволеність користувачів. Вони є ключовими для створення інтерактивних, зручних та доступних веб-сервісів.

1.1.2 Серверні технології

Серверні технології є невід'ємною частиною веб–розробки, оскільки вони обробляють запити від клієнтів, виконують бізнес-логіку та управляють базами даних. Вони включають у себе мови програмування, фреймворки та серверне програмне забезпечення, які працюють на веб-серверах.

Node.js є однією з найпопулярніших серверних платформ, яка використовує JavaScript для серверної розробки [7]. Це дозволяє розробникам використовувати одну мову програмування як на клієнтській, так і на серверній стороні, що спрощує розробку та підтримку веб-додатків. Express.js є легким та гнучким фреймворком для Node.js, який надає потужні інструменти для створення веб-додатків та API [8].

PHP є ще однією широко використовуваною мовою для серверної розробки. Вона проста у вивченні та має велику спільноту. Laravel є сучасним PHP фреймворком, який надає багатий набір функцій для веб-розробки, включаючи аутентифікацію, маршрутизацію, сесії та кешування [9].

Серверні технології також включають у себе системи управління базами даних (СУБД), такі як MySQL, PostgreSQL та MongoDB, які забезпечують зберігання та управління даними. Вони дозволяють розробникам ефективно працювати з даними, виконувати запити та забезпечувати інтеграцію з веб-додатками.

Безпека є ключовим аспектом серверної розробки. Розробники повинні впроваджувати заходи безпеки, такі як аутентифікація, авторизація, шифрування та захист від вразливостей, щоб забезпечити захист даних користувачів.

Серверні технології постійно розвиваються, і нові інструменти та підходи з'являються регулярно. Це включає в себе тенденції, такі як мікросервіси, контейнеризація та хмарні обчислення, які змінюють спосіб розгортання та

управління серверними додатками. Вони дозволяють розробникам створювати більш гнучкі, масштабовані та надійні системи.

Загалом, серверні технології є фундаментом для будь-якого веб-додатка, і їх розуміння є критично важливим для будь-якого веб-розробника. Вони дозволяють створювати потужні, безпечні та ефективні веб-сервіси, які можуть обслуговувати мільйони користувачів по всьому світу.

1.2 Аналіз використаних технологій

Для розробки інтерактивного вебдодатку Training Club обрано сучасний стек технологій, який включає React, Material UI, TypeScript для клієнтської частини та NestJS та MongoDB для серверної.

React обрано як основу клієнтської частини через його гнучкість та ефективність у створенні інтерактивних користувацьких інтерфейсів. Використання JSX та компонентного підходу дозволяє легко інтегрувати динамічні елементи інтерфейсу та оптимізувати процес розробки.

Material UI надає готовий набір компонентів, що базуються на принципах Material Design від Google, що забезпечує високу якість дизайну та користувацького досвіду [10].

TypeScript використовується для додання строгих типів до JavaScript, що підвищує надійність коду та спрощує його масштабування та підтримку [11].

NestJS є прогресивним Node.js фреймворком для побудови ефективних та надійних серверних додатків. Він використовує TypeScript за замовчуванням та надає широкі можливості для модульності та масштабування.

MongoDB є NoSQL базою даних, яка відома своєю гнучкістю та легкістю у використанні. Вона ідеально підходить для сучасних веб-додатків, які потребують швидкого доступу до даних та їх гнучкої структури [12].

Для забезпечення якості коду та його надійності обрано фреймворк для тестування Jest [13]. Jest є одним з найпопулярніших фреймворків для тестування JavaScript-коду, зокрема, завдяки таким перевагам:

— Jest оптимізований для швидкого виконання тестів, що дозволяє ефективно інтегрувати тестування у процес розробки;

— можливість використання snapshot-тестів дозволяє автоматично відстежувати зміни у відображенні компонентів;

— кожен тест виконується в ізольованому середовищі, що забезпечує точність результатів та виключає побічні ефекти;

— вбудовані засоби для створення моків та шпигунів дозволяють тестувати компоненти у відокремленні від зовнішніх залежностей;

— Jest пропонує готові до використання налаштування, що значно спрощує процес налаштування тестів.

Використання Jest у проекті дозволило забезпечити високу якість коду, виявити та виправити помилки на ранніх етапах розробки, а також спростити процес рефакторингу та додавання нових функцій.

Для документування API обрано Swagger — це набір інструментів для розробки, побудови, документування та використання RESTful веб-сервісів [16]. Він дозволяє розробникам легко визначати структуру API, генерувати інтерактивну документацію, а також створювати SDK та клієнтські бібліотеки для API на різних мовах програмування. Основним форматом, який використовується Swagger, є OpenAPI Specification (OAS), яка описує структуру API у зрозумілому для людей та машин вигляді.

Переваги використання Swagger:

— Swagger автоматично генерує зрозумілу та інтерактивну документацію, яку можна використовувати для тестування API безпосередньо з браузера;

— завдяки автоматичному генеруванню документації розробники можуть швидко ознайомитися з API, зрозуміти його структуру та параметри, що зменшує час на інтеграцію;

— інтерактивна документація Swagger дозволяє тестувати API прямо з браузера, що значно спрощує процес відлагодження;

— Swagger може використовуватись для автоматичного генерування клієнтських бібліотек та серверних каркасів, що скорочує час на написання коду.

Nest.js — це сучасний фреймворк для Node.js, який побудований на основі TypeScript і дозволяє створювати масштабовані та підтримувані серверні додатки. Використання Swagger з Nest.js має ряд переваг:

— завдяки типізації TypeScript Swagger може автоматично генерувати документацію, що мінімізує можливість помилок;

— Nest.js надає декоратори, які дозволяють легко налаштувати документацію для кожного методу контролера;

— документація Swagger автоматично оновлюється при зміні коду, що зменшує потребу в ручному редагуванні документації;

— Nest.js дозволяє легко налаштувати аутентифікацію для API, використовуючи декоратори Swagger.

Використання цих технологій дозволяє створити надійний та масштабований веб-додаток, який забезпечить високу продуктивність та користувацький комфорт.

2 ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА ФІТНЕС-ДОДАТКІВ

2.1 Аналіз сучасних рішень

MyFitnessPal — це один з найбільш популярних додатків для контролю за харчуванням та фізичною активністю, який допомагає мільйонам користувачів досягати своїх цілей щодо здоров'я та фітнесу. Додаток пропонує різноманітні функції для відстеження харчування, фізичних вправ та здоров'я, що робить його універсальним інструментом для підтримки активного способу життя. У цьому розділі розглянуто основні функції, переваги та недоліки MyFitnessPal, а також його вплив на користувачів. Фітнес–додаток MyFitnessPal представлений на рисунку 2.1.

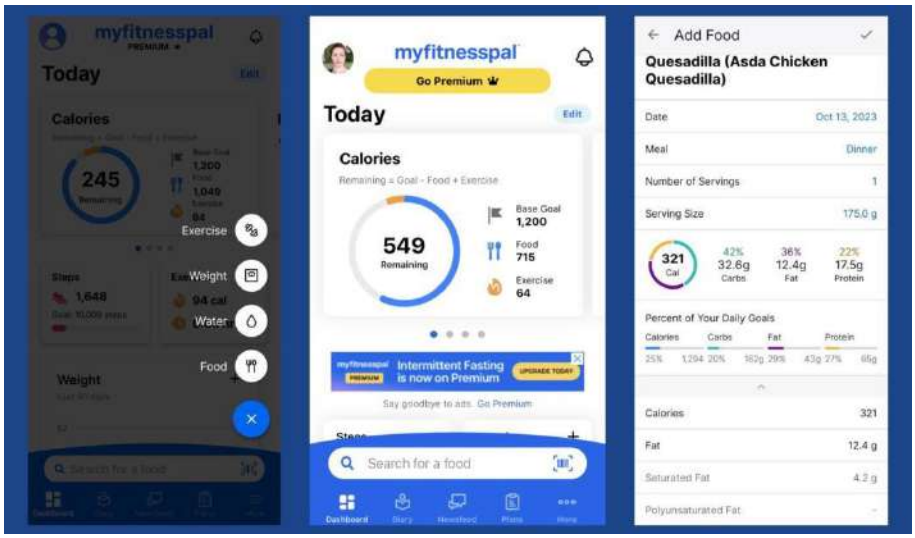

Рисунок 2.1 — Фітнес–додаток MyFitnessPal

Переваги MyFitnessPal:

— величезна база даних продуктів харчування, яка значно полегшує процес відстеження харчування;

— інтерфейс додатка є інтуїтивно зрозумілим та зручним, що дозволяє користувачам швидко та легко додавати дані про харчування та фізичну активність;

— надає персоналізовані рекомендації та звіти, що допомагає користувачам досягати своїх цілей більш ефективно;

— активна спільнота користувачів створює підтримуючу атмосферу, де можна знайти мотивацію та отримати поради від інших людей;

— додаток інтегрується з багатьма іншими сервісами та додатками, що забезпечує комплексний підхід до відстеження здоров'я.

Недоліки MyFitnessPal:

— у безкоштовній версії додатка є реклама, що може бути відволікаючим фактором;

— іноді трапляються помилки в інформації про продукти, що може призводити до неточностей у відстеженні харчування;

— може не мати деяких спеціалізованих функцій, які є у інших фітнес-додатків, таких як деталізовані тренувальні плани або інструкції до вправ.

Sworkit – це популярний фітнес-додаток, який пропонує користувачам персоналізовані тренування, що можуть бути виконані в будь-якому місці без спеціального обладнання. Завдяки своїм адаптивним тренуванням та зручному інтерфейсу, Sworkit став незамінним інструментом для багатьох людей, які прагнуть підтримувати активний спосіб життя. Фітнес–додаток Sworkit представлений на рисунку 2.2.

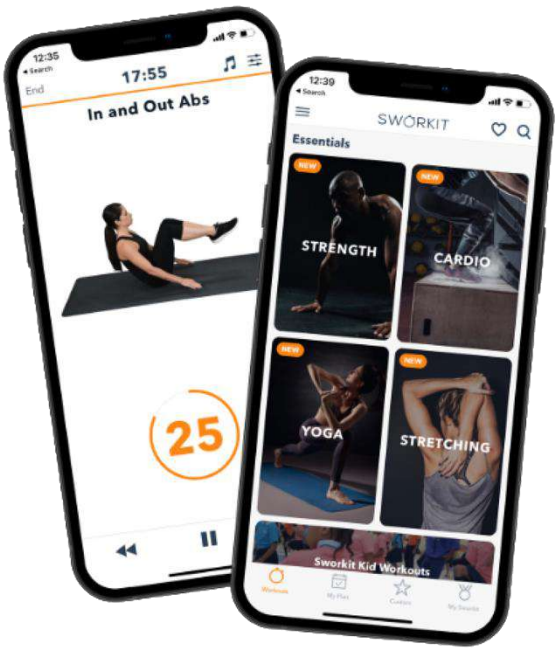

Рисунок 2.2 — Фітнес–додаток Sworkit

Переваги Sworkit:

— однією з найбільших переваг Sworkit є можливість виконувати тренування будь-де та будь-коли без необхідності спеціального обладнання;

— Sworkit створює індивідуальні тренувальні плани на основі цілей та уподобань користувача, що дозволяє досягати максимальних результатів;

— професійні відеоуроки допомагають користувачам правильно виконувати вправи, що знижує ризик травм та підвищує ефективність тренувань;

— додаток інтегрується з популярними фітнес-трекерами, що дозволяє автоматично записувати дані про тренування та забезпечує комплексний підхід до відстеження здоров'я;

— Sworkit пропонує тренування для користувачів різного рівня підготовки, що робить його доступним для широкої аудиторії.

Недоліки Sworkit:

— деякі функції та тренування доступні лише в преміум-версії, що може бути незручним для користувачів, які не готові платити за додаток;

— хоча тренування без обладнання є перевагою для багатьох користувачів, ті, хто прагне більш інтенсивних тренувань з використанням спеціалізованого обладнання, можуть знайти Sworkit обмеженим;

— незважаючи на можливість отримання консультацій від тренерів, ця функція не завжди доступна в режимі реального часу, що може бути незручним для деяких користувачів.

2.2 Порівняльна характеристика вебдодатку Training Club

На ринку вже існує чимало фітнес-додатків, які пропонують широкий спектр можливостей для користувачів. Серед них виділяються такі відомі платформи, як MyFitnessPal і Sworkit, які здобули значну популярність завдяки своїм унікальним функціям та зручності використання.

У цьому розділі проведемо порівняльну характеристику вебдодатку Training Club з цими двома додатками, щоб визначити його сильні та слабкі сторони, а також розглянути можливості для вдосконалення та масштабування. Це дозволить

зрозуміти конкурентні переваги Training Club та його потенціал у розвитку фітнес-індустрії.

Перевагами Training Club є можливість тренерам розробляти власні колекції вправ та ділитися ними з іншими користувачами. Це дозволяє тренерам виявити свою креативність та експертність, а також надихнути інших на досягнення своїх фітнес-цілей. На відміну від MyFitnessPal, який більше зосереджений на відстеженні харчування, Training Club ставить акцент на тренуваннях і розвитку фізичних навичок користувачів.

Кожен користувач зможе знайти тренування, які відповідають їхнім потребам і можливостям, завдяки різноманітним фільтрам. Це дозволяє користувачам отримувати більш персоналізований досвід, ніж в Sworkit, де тренування більш універсальні;

Вебдодаток пропонує безкоштовний режим з обмеженим функціоналом без реклами на початку старту, що забезпечує привабливість для нових користувачів. Це відрізняє Training Club від багатьох конкурентів, які активно використовують рекламу для монетизації.

Постійне оновлення вмісту та врахування змін у сфері фітнесу та здоров'я дозволять Training Club залишатися актуальним і конкурентоспроможним. Це забезпечує довгострокову привабливість платформи для користувачів.

Недоліками Training Club є хоча безкоштовний режим є привабливим для нових користувачів, обмежений функціонал може не задовольнити всіх потреб користувачів, що може стати перешкодою для утримання користувачів на довготривалій основі.

На відміну від MyFitnessPal, який пропонує детальний трекінг харчування, Training Club не має такої функції. Відсутність можливості відстежувати калорії та макроелементи може стати значним недоліком для користувачів, які прагнуть комплексного підходу до свого здоров'я та фітнесу.

Для підтримки актуальності та конкурентоспроможності Training Club необхідно постійно інвестувати в оновлення контенту, розробку нових функцій та маркетинг. Це може стати значним фінансовим тягарем.

Також є шляхи вдосконалення Training Club, а саме додати функції відстеження харчування та інтеграція з відомими харчовими трекерами, такими як MyFitnessPal, може значно підвищити привабливість Training Club для користувачів, які шукають комплексний підхід до здоров'я та фітнесу.

Збільшення кількості безкоштовних функцій може допомогти залучити більше користувачів та утримати їх на платформі. Це може включати додавання базових тренувальних програм, доступ до обмеженої кількості тренувань від тренерів та інші корисні функції.

Впровадження елементів гейміфікації, таких як досягнення, нагороди та лідерборди, може зробити процес тренувань більш захоплюючим та мотивуючим для користувачів.

Додавання соціальних функцій, таких як форуми, групи підтримки та можливість ділитися досягненнями, може допомогти створити активне ком'юніті навколо Training Club. Це сприятиме зростанню лояльності користувачів та збільшенню їх залученості.

Розширення інтеграції з популярними фітнес-трекерами, такими як Fitbit, Apple Watch та інші, дозволить користувачам отримувати більш точні дані про свою фізичну активність та покращити їх тренувальний процес.

3 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ

3.1 Обґрунтування та розробка структури вебдодатку

Головна мета полягає в створенні якісного продукту, який надасть можливість людям будь–якого віку займатися спортом будь де. Це означає, що вебдодаток пропонує спортивні вправи, тренування та рекомендації, які дозволять користувачам залишатися активними та підтримувати своє здоров'я, незалежно від їхнього місцезнаходження. Важливо забезпечити зручний та доступний спосіб доступу до фітнес–ресурсів, щоб створити ефективний інструмент для збереження фізичної активності та сприяння здоровому способу життя.

Цільова аудиторія цього сайту — це люди будь–якого віку. Основний акцент робиться на універсальність та доступність для всіх користувачів. Це означає, що від дітей до пенсіонерів, будь–яка людина може знайти корисний та привабливий контент для занять спортом. Розробка повинна враховувати різні потреби та очікування цільової аудиторії.

Структура вебдодатку, яка включає основну сторінку, панель адміністратора для створення вправ та тренувань, а також сторінку для відображення всіх тренувань, відповідає цілям проекту. Основна сторінка може включати важливі інформаційні блоки, привабливий дизайн та просту навігацію, щоб зацікавити користувачів і залучити їх до інтеракції з ресурсом. Панель адміністратора дозволить тренерам та адміністраторам додавати нові вправи та тренування, а сторінка з відображенням тренувань наддасть можливість користувачам легко знаходити та вибирати вправи, які їм підходять.

Загалом, така структура відповідає меті сайту, надає доступ до корисного контенту для різних категорій користувачів і сприяє популяризації активного та здорового способу життя.

Дизайн є ключовим аспектом в розробці інтерактивного вебдодатку Training Club. Вибір колірної гами та загального стилю має бути обґрунтованим та відповідати основним цілям та концепції проекту.

Однією з важливих вихідних ідей для вибору жовтої колірної гами є інспірація від існуючих ресурсів, зокрема TRX. TRX — відомий бренд у сфері

фітнесу, і вибір жовтої гами був інтегрований для створення асоціацій з цим відомим брендом. Жовтий колір також асоціюється з енергією, оптимізмом і активністю, що ідеально підходить для вебдодатку, який спонукає до здорового способу життя та активних фізичних навантажень. Крім того, жовтий колір привертає увагу та створює візуальний контраст з іншими кольорами, що дозволить виділити ключові елементи, такі як кнопки для реєстрації на тренування, вибору програм тощо.

Дизайн повинен бути сучасним та привабливим для цільової аудиторії, яка включає людей будь–якого віку. Сучасний стиль включає в себе мінімалістичний дизайн, який спрощує взаємодію користувачів з вебдодатком, та велику увагу до деталей.

Оформлення має бути інтуїтивно зрозумілим та легким у навігації. Це означає використання зрозумілих іконок, чіткого розташування елементів і зручний шрифт.

Обрана колірна гама та загальний стиль дизайну повинні підтримувати цілі та концепцію проекту, роблячи вебдодаток привабливим та зручним для користувачів будь–якого віку, що бажають займатися спортом та підтримувати активний спосіб життя.

3.2 Розробка макету сторінок вебдодатка в сервісі Figma

3.2.1 Головна сторінка

Головна сторінка, доступна користувачам після реєстрації, є центральним пунктом фітнес–сервісу. Вона призначена не лише для сприяння ознайомленню з можливостями платформи, але й для створення простору, де ви можете розпочати свій фітнес–шлях. Все, що потрібно для активного та здорового способу життя, знаходиться прямо тут. Головна сторінка представлена на рисунку 3.1.

На головній сторінці відразу доступна ключова можливість — кнопка "Розпочати Тренування". Кожен раз, коли відчуваєте потребу у новому виклику чи просто бажаєте випадкового підходу до тренувань, ця кнопка стане найкращим другом.

Вона веде до випадкового тренування, дозволяючи урізноманітнити свої фізичні зусилля та випробувати щось нове.

Рисунок 3.1 — Головна сторінка

Меню в верхньому лівому куті головної сторінки слугує навігаційним компасом. Тут можна швидко переходити до інших розділів які надає платформа, таких як "Мої Тренування", "Колекції Вправ", "Профіль", чи "Налаштування". Ваша фітнес–подорож тепер легка та доступна за лічені клацання.

Чотири кнопки з додатковим функціоналом розширюють можливості. "Створити Тренування" дозволяє самостійно формувати тренувальні програми, виходячи з уподобань та цілей. "Мої Улюблені" зберігає улюблені тренування на одному місці для швидкого доступу. "Загальні Досягнення" відкриває можливості поділитися своїми успіхами та спілкуватися з іншими учасниками фітнес–спільноти. Четверта кнопка "Нещодавні", надає можливість переглянути самі нові тренування, які були додані самою платформою чи тренерами.

З головною сторінкою фітнес–досвід стає не лише ефективним, але й цікавим та захоплюючим.

3.2.2 Панель адміністратора

Панель адміністратора Training Club — це місце, де ви, як адміністратор або тренер, можете створювати та управляти вправами та тренуваннями. Сторінка для створення тренувань та вправ представлена на рисунку 3.2 — 3.3. Тут у вас є повний контроль над контентом, що надається користувачам сайту.

Рисунок 3.2 — Редактор тренувань

Ви можете додавати нові вправи, редагувати існуючі, створювати тренувальні програми та ділитися ними зі своєю аудиторією.

3.2.3 Сторінка виконання вправ

Сторінка виконання вправ розроблена з однією метою – надати структурований та ефективний підхід до фізичної активності. Для отримання можливості побудувати свої тренування, враховуючи ваші особисті потреби та фітнес–цілі. Кожен етап: "Підготовка", "Робота", та "Відпочинок" — має своєрідну

роль у досягненні оптимальних результатів, підтримуючи на кожному етапі фітнес–подорожі. Сторінка виконання вправ представлена на рисунку 3.4.

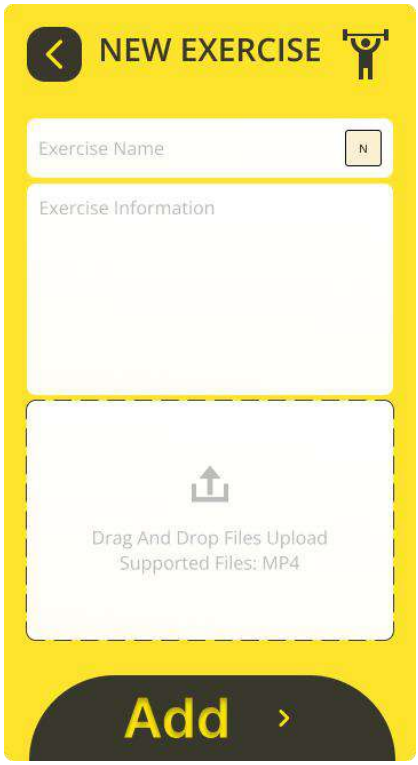

Рисунок 3.3 — Редактор вправ

Ця сторінка призначена для всіх, хто шукає простий та зручний спосіб удосконалити своє тіло та покращити своє фізичне здоров'я. Незалежно від досвіду чи рівня підготовки, кожен знайде тут інструменти для створення персоналізованих тренувань, які відповідають кожному.

Під час використання сторінки виконання вправ, користувачі переживають троє основних станів, кожен з яких виконує ключову роль у покращенні фізичної форми та досягненні фітнес–цілей.

Підготовка — в цьому стані користувачі зосереджують свою увагу на підготовці організму до майбутніх фізичних зусиль. Використовуючи легкі аеробні вправи та розтяжки, вони активують м'язи та підготовлюють суглоби до інтенсивної фізичної активності. Цей етап сприяє підвищенню гнучкості та зменшенню ризику травм.

Робота — в цій фазі користувачі виконують інтенсивні вправи для досягнення своїх фітнес–цілей. Кожен рух зорієнтований на конкретний м'яз чи групу м'язів,

щоб підвищити силу, витривалість чи інші фізичні показники. Важливо дотримуватися правильної техніки виконання та контролювати своє навантаження для досягнення максимальних результатів.

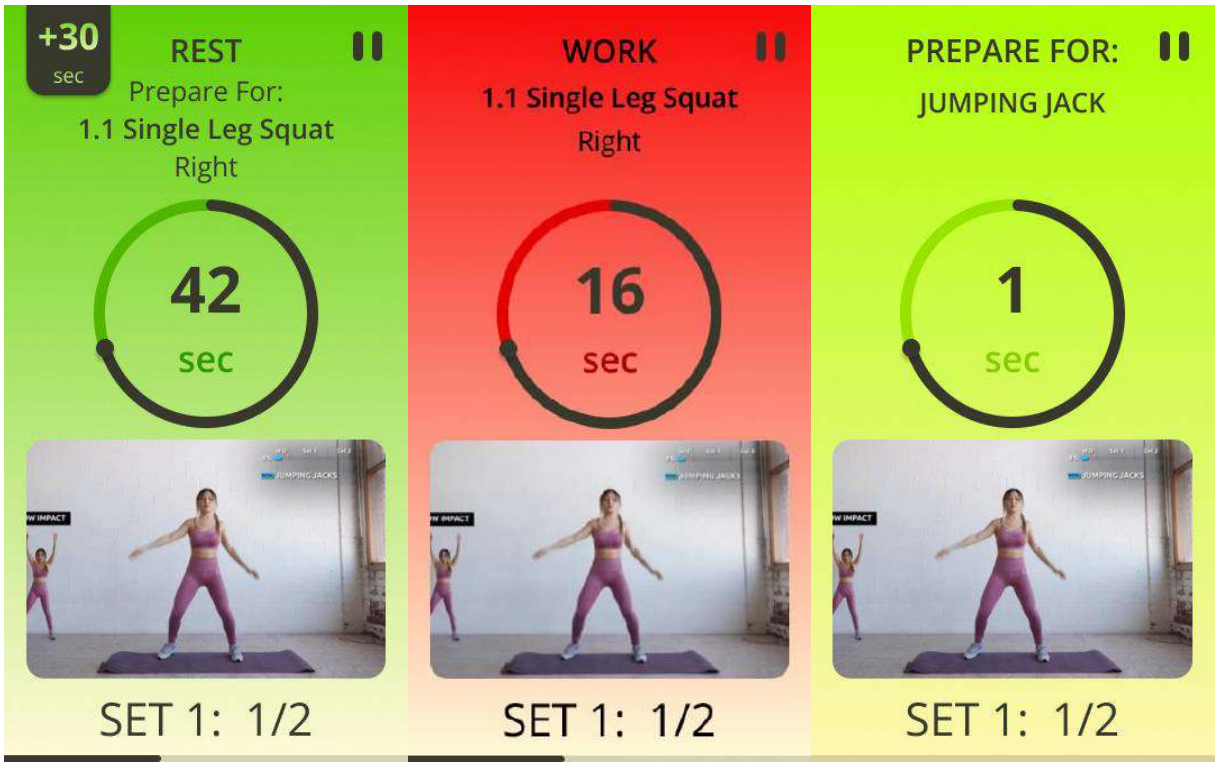

Рисунок 3.4 — Сторінка тренувань

Відпочинок — це період, коли організм відновлює свою енергію після інтенсивного фізичного навантаження. Використовуючи короткі інтервали відпочинку, користувачі допомагають м'язам відновити енергію та готуються до наступного циклу тренувань. Цей етап зменшує втому, сприяє оптимальному відновленню та підготовці до наступної фази робочого навантаження. Загальний результат кожного тренування залежить від управління цими трьома станами, дозволяючи користувачам насолоджуватися ефективним та безпечним процесом тренувань, щоб досягти найкращих результатів у покращенні свого фізичного здоров'я та форми.

3.3 Реалізація навігації по вебдодатку

У процесі створення інтерактивного вебдодатку Training Club, особлива увага приділялася забезпеченню користувачам зручної та ефективної навігації.

Використання бібліотеки react–router–dom у поєднанні з createBrowserRouter дозволило створити структуровану систему маршрутів та навігаційного функціоналу.

Вебдодаток має ряд визначених маршрутів, кожен з яких пов'язаний із відповідним компонентом. Нижче детально розібрано логіку необхідну для налаштування навігації по вебдодатку. Розібрано лише частину коду, весь код для навігації представлено на ДОДАТКУ Б.

Оголошується змінна Router та експортується. Змінна Router містить конфігурацію для маршрутизатора, створеного за допомогою функції createBrowserRouter:

```
export const Router = createBrowserRouter([ ... ]);
```

Оголошується маршрут для кореневої сторінки. Компонент AuthenticationLayout, який відображається при доступі до кореневої сторінки. Також для випадків при яких виникатиме помилка вказується компонент Error, який відображає користувачам певну інформацію про те, що сталося. Останнім параметром передається масив дочірніх сторінок:

```
[{ path: '/',
element: <AuthenticationLayout />,
errorElement: <Error />,
children: [ ... ] }
```

В масив children передається схожий об`єкт з тим, що був описаний раніше. Перший параметр "path" приймає посилання на вказану сторінку в параметрі element. В параметрах лише відрізняються значення для доступу до сторінки HOME:

```
children: [{
path: RoutesPath.HOME,
element: ( <NavigateMenuProvider> <Home /> </NavigateMenuProvider> ),
}]
```

У контексті адміністративної частини вебдодатку, зручність навігації для адміністраторів досягнута завдяки використанню відповідних маршрутів, які спрощують доступ до функціоналу та можливості редагування.

Крім того, для забезпечення додаткового зручного функціоналу, реалізована функція comeBack, яка використовує функцію navigate зі стороньої бібліотеки React Router Dom для переходу на головну сторінку:

```
const comeback = (): void => {
 navigate(RoutesPath.HOME)
)}
```

Це може бути використано для реалізації кнопок або елементів керування, які дозволяють адміністраторам швидко повертатися на головну сторінку під час редагування або перегляду даних.

Візуальна реалізації кнопки, яка використовує вище наведений код представлена на рисунку 3.5.

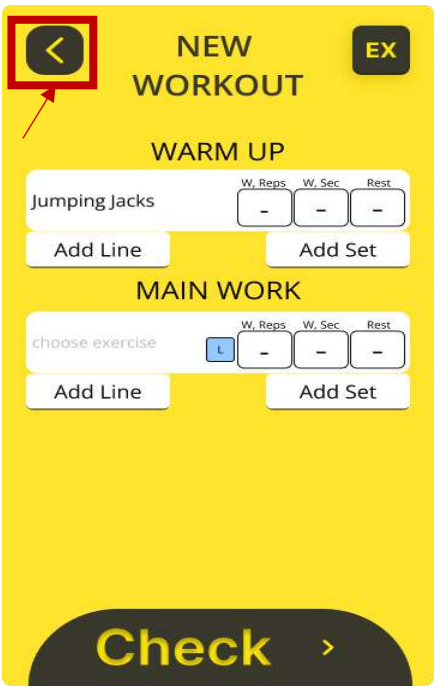

Рисунок 3.5 — Вигляд кнопки для переходу на попередню сторінку

Таким чином, інфраструктура навігації, вбудована для забезпечення максимальної зручності та ефективності користувачам у використанні різних частин та функціоналу фітнес-платформи. Блок-схема навігації по вебдодатку представлена на рисунку 3.6.

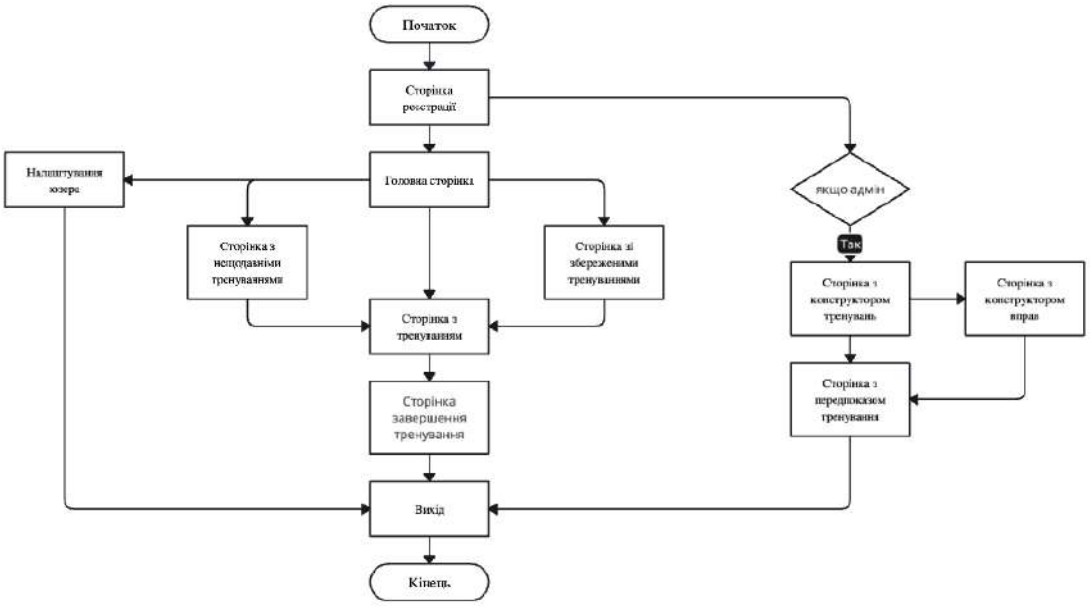

Рисунок 3.6 — Блок-схема навігації по вебдодатку

3.4 Push–повідомлення

Push-повідомлення є важливим інструментом для залучення користувачів та підтримки їх активності. Вони дозволяють надсилати короткі повідомлення прямо на пристрої користувачів, навіть коли вебдодаток або мобільний додаток не використовується. Це може бути корисно для різноманітних цілей, наприклад, для сповіщення про нові повідомлення у соціальних мережах, оновленнях статусу замовлення, або нагадуваннях про події.

Для реалізації push-повідомлень у дипломному проекті використано сервіс OneSignal. OneSignal — це платформа для залучення клієнтів, яка дозволяє компаніям різних розмірів збільшувати утримання та доходи за допомогою швидких, персоналізованих повідомлень. OneSignal підтримує електронну пошту, push-повідомлення, повідомлення в додатку, SMS, Live Activities від Apple та інші канали через Webhooks.

У проекті використовується функція usePushNotifications для ініціалізації та управління push-повідомленнями, код якої детально розібраний нижче, а весь код представлений на ДОДАТКУ В.

Оголошується та експортується функція usePushNotifications, яка є власною реалізацією для керування push-сповіщеннями:

```
export const usePushNotifications = () => {}
```

Перевіряється, чи користувач підписаний на сповіщення, зчитуючи значення з локального сховища:

```
const isSubscribed = getFromLocalStorage(LocalStorageKey.NOTIFICATION) === 'true'
```

Перевіряється, чи ініціалізовано сповіщення, зчитуючи значення з локального сховища:

```
const isInitNotification = getFromLocalStorage(LocalStorageKey.IS_INIT_NOTIFICATION) === 'true'
```

Оголошується асинхронна функція init, яка виконує ініціалізацію push-сповіщень. Функція init перевіряє чи вже ініціалізовано сповіщення, якщо користувач вперше зареєструвався, то йому автоматично увімкнено та збережено усю необхідну інформацію в localStorage:

```
const init = async (): Promise<void> => {}
```

Оголошується асинхронна функція subscribe для підписки на push-сповіщення. Ця функція ініціалізує користувача в сервісі OneSignal для подальшої взаємодії з ним та зберігає в localStorage прапорець для зазначення ініціалізації користувача:

```
const subscribe = async (): Promise<void> => {
OneSignal.User.PushSubscription.optIn()
setToLocalStorage(LocalStorageKey.NOTIFICATION, true)
}
```

Оголошується асинхронна функція unSubscribe для відписки від push-сповіщень. Функція вказує в localStorage по ключу, що юзер відмінив надсилання push-повідомлень та одночасно вимикає їх в сервісі OneSignal:

```
const unSubscribe = async (): Promise<void> => {
OneSignal.User.PushSubscription.optOut()
setToLocalStorage(LocalStorageKey.NOTIFICATION, false)
}
return { subscribe, unSubscribe, isSubscribed, init }
}
```

Push-повідомлення виглядають як спливаючі віконця на екрані пристрою, які містять коротке повідомлення та можуть включати зображення або кнопки для взаємодії. Приклад як саме виглядає повідомлення представлений на рисунку 3.7.

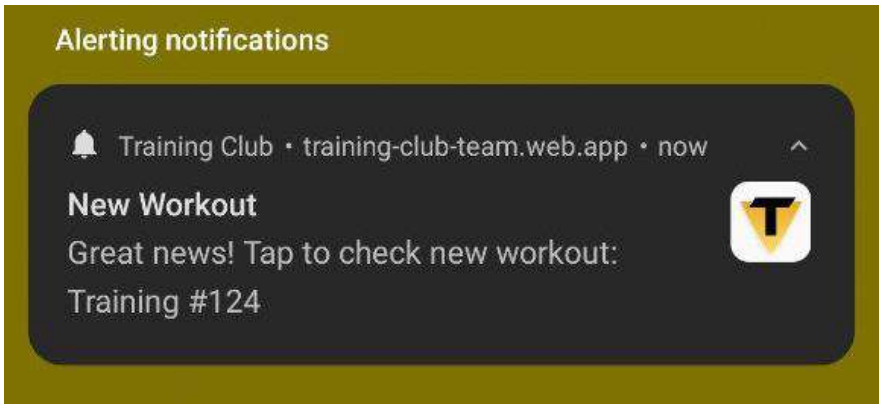

Рисунок 3.7 — Push-повідомлення нового тренування

Користувачі мають можливість увімкнути або вимкнути отримання push-повідомлень у налаштуваннях свого акаунта. Це дає їм контроль над тим, які повідомлення вони бажають отримувати, та забезпечує кращий досвід користування. Сторінка налаштувань для push-повідомлень представлена на рисунку 3.8.

3.5 Панель адміністратора

Панель адміністратора є ключовою частиною платформи Training Club, яка

надається виключно користувачам зі статусом "admin". Ця панель дозволяє адміністраторам створювати та керувати тренуваннями та вправами, що є невід'ємною частиною функціональності платформи. Завдяки інтуїтивно зрозумілому інтерфейсу, адміністратори мають змогу легко та ефективно управляти контентом, що сприяє кращій організації тренувальних процесів.

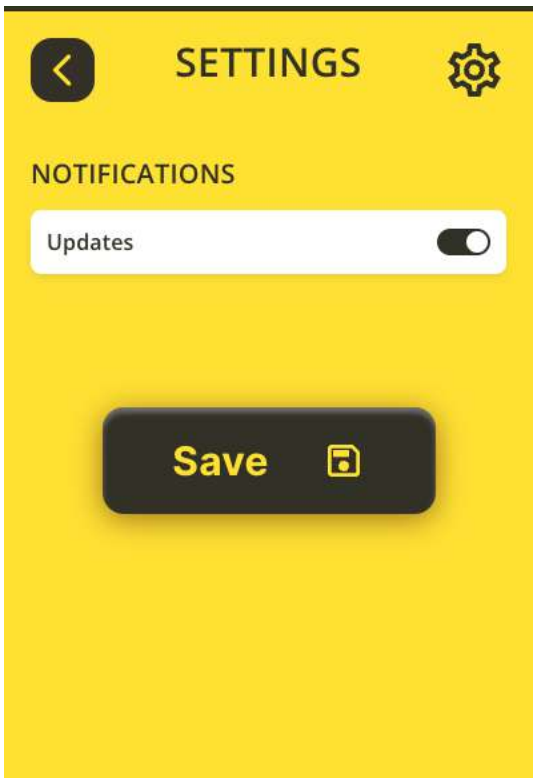

Рисунок 3.8 – Налаштування Push-повідомлень

Сторінка для створення тренування (див. рис. 3.2) містить форму, реалізовану за допомогою бібліотеки React Hook Form, яка забезпечує максимальну гнучкість та ефективність. React Hook Form - це високопродуктивна, гнучка та розширювана бібліотека для форм, яка оптимізує кількість перерендерів, мінімізує обчислення валідації та прискорює монтування компонентів. Використання цієї бібліотеки дозволяє створювати форми з усіма необхідними валідаціями, що попереджає помилки користувачів та підвищує якість введених даних.

Нижче представлено код та детальний аналіз коду форми, яка використовується для створення тренувань.

Створюється спеціальна обгортка за допомогою компонента FormFieldContainer, що спрощує взаємодію коду та даних, які користувачі будуть вводити в надані поля:

<FormFieldContainer spacing={2} direction="row" alignItems="end">

Використовується компонент Controller з бібліотеки React Hook Form для управління станом форми. Він приймає параметрами назву поля в яке записується значення введене юзером, функцію яка відслідковує зміни та елемент, який відображатиметься користувачу для взаємодії.

<Controller name={`${formName} mode`} control={control} render={({ field: { value } }) => <Box>{value && <MarkerMode />}</Box>} />

Натиснувши на кнопку "EX" у верхньому правому куті панелі, адміністратори можна перейти на сторінку для створення вправ. На цій сторінці юзери можуть вибрати вправу для редагування з випадаючого списку або створити нову. Це забезпечує легкість управління та можливість швидкого оновлення контенту, що є критичним для динамічного та актуального тренувального процесу. Діалогове вікно вибору вправ представлене на зображені 3.9.

Функціональність панелі адміністратора включає управління користувачами, управління контентом, налаштування вебдодатку, управління аналітикою. Ці можливості дозволяють адміністраторам ефективно керувати платформою та забезпечувати високий рівень сервісу для користувачів.

3.6 Реалізація логіки тренування

Основна логіка доступу до тренувань у додатку реалізована через два інтуїтивно зрозумілі механізми. Перший дозволяє користувачу розпочати тренування негайно, натиснувши кнопку на головному екрані, що ініціює випадково вибране тренування. Цей метод сприяє різноманітності та спонтанності у фітнес-рутині користувача, дозволяючи відкрити нові види вправ без попереднього планування (див. рис. 3.5).

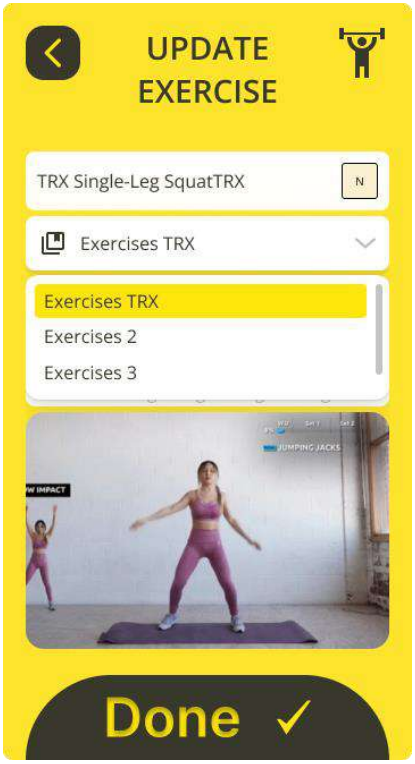

Рисунок 3.9 — Діалогове вікно вибору вправ

Другий метод передбачає використання сторінки "Нещодавні тренування", де користувач може переглянути історію своїх тренувань та обрати конкретну вправу для повторення. Це дозволяє підтримувати послідовність у тренуваннях та фокусуватися на певних фізичних цілях. Сторінка з нещодавніми тренуваннями представлена на рисунку 3.10.

Після вибору тренування користувач переходить на сторінку перед показу тренування, де представлена вся необхідна інформація: опис вправ, очікувані результати, необхідне обладнання та інструкції. Такий підхід забезпечує користувачу повний контекст тренування перед його початком і допомагає підготуватися до виконання вправ.

Процес тренування поділений на три основні режими: "Робота", "Відпочинок", та "Підготовка" (див. рис. 3.4). Кожен режим має унікальний колір та звукове супроводження, що створює чітке розуміння поточної фази тренування та допомагає користувачу зосередитися на виконанні вправ.

Рисунок 3.10 — Сторінка з нещодавнми тренуваннями

По завершенню тренування користувач потрапляє на сторінку фінішу, де відображається загальний час, проведений на тренуванні, та кількість спалених калорій.

Також на цій сторінці пропонується можливість поділитися результатами у соціальних мережах за допомогою згенерованої карточки, яка включає всю інформацію про тренування, що може слугувати мотивацією для користувача та його друзів. Картка з інформацією про тренування представлена на рисунку 3.11.

3.7 Тестування роботи вебдодатку

Під час розробки інтерактивного вебдодатку Training Club виникла потреба в тестуванні, яке грає ключову роль у забезпеченні якості та стабільності додатку. В даному проекті для здійснення тестування використовуються одиничні, інтеграційні та функціональні тести. Для їхнього написання та виконання обрана бібліотека Jest, яка є потужним інструментом для тестування веб–застосунків.

.

Рисунок 3.11 — Картка з інформацією про тренування

Є різні можливості для тестування написаногоного коду, а саме одиничні тести використовуються для перевірки правильності роботи окремих функцій чи модулів. Наприклад, для перевірки коректності функції serializeTrainings, яка відповідає за обробку тренувань.

Інтеграційні тести використовуються для перевірки взаємодії між різними компонентами та модулями системи, що в свою чергу дозволяє переконатися, що вони працюють разом безперебійно.

функціональні тести охоплюють перевірку функціональності веб–додатку на рівні користувацького інтерфейсу.

Нижче наведено код та проаналізовано частину коду, а саме тест для функції serializeTrainings. Увесь лістинг коду функції serializeTrainings представлений на ДОДАТКУ И. Результати тестування функції serializeTraining представлені на рисунку 3.12.

Оголошується блок тестів для функції serializeTrainings за допомогою методу describe:

describe('serializeTrainings', () => {

Оголошується перший тест, який перевіряє кількість елементів, повернутих функцією:

test('The number of elements returned by the function', () => {

Виконується виклик функції з вхідними даними і перевіряється, чи повернутий об'єкт відповідає об'єкту, що був вказаний для перевірки правильності результатів за структурою та значеннями:

expect(
serializeTrainings(trainingResponse.warmUp[0].trainingSessions),
).toMatchObject(serializeTrainingsResult)})

Оголошується другий тест, який перевіряє, чи повертає функція точний очікуваний об'єкт:

test('The object that the function should return', () => {
expect(serializeTrainings(trainingResponse.warmUp[0].trainingSessions)).toEqua
(serializeTrainingsResult)})

Оголошується третій тест, який перевіряє, чи не повертає функція неправильне значення, наприклад порожній об'єкт:

test('The function returns an invalid value', () => {
expect(erializeTrainings(trainingResponse.warmUp[0].trainingSessions)).not.toE
qual({})})

Рисунок 3.12 — Результат тестування функції serializeTrainings

Використання Jest дозволяє нам ефективно виявляти та виправляти помилки, а також підтримувати стабільність та надійність функціоналу вебдодатку.

Тестування один із основних етапів в розробці, що допомагає забезпечити високу якість та надійність роботи фітнес-платформи.

4 РОЗРОБКА СЕРВЕРНОЇ ЧАСИНИ

4.1 Ініціалізація та налаштування NestJS

Для реалізації серверної частини обрано Nest.js через його масштабованість, гнучкість та підтримку TypeScript, що забезпечує сильну типізацію та об'єктно-орієнтований підхід. Це дозволяє створювати надійні та легко підтримувані вебдодатки.

Нижче наведений код основного файлу усієї платформи, який налаштовує поведінку роботи усієї API сервера. В цьому файлі імпортуються необхідні модулі та сервіси для ініціалізації та налаштування NestJS додатку. ValidationPipe використовується для валідації вхідних даних запитів, ConfigService для доступу до конфігураційних змінних, NestFactory для створення екземпляру додатку, а SwaggerModule для документації API. CookieParser дозволяє легко обробляти cookies. Фільтри HttpExceptionFilter та AuthExceptionFilter використовуються для глобальної обробки помилок та винятків. Використання useContainer дозволяє використовувати залежності в межах модуля AppModule. Функція bootstrap ініціалізує та запускає додаток. Нижче наведений код та детальний аналіз основного файлу серверної частини вебдодатку. Лістинг коду ініціалізації програми представлений на ДОДАТКУ Ж.

Оголошується асинхронна функція bootstrap:

```
async function bootstrap(): Promise<void> {}
```

Створюється екземпляр додатку за допомогою NestFactory, використовуючи AppModule як кореневий модуль, а також отримується екземпляр ConfigService для доступу до конфігураційних параметрів додатку:

```
const app = await NestFactory.create(AppModule);
const config = app.get(ConfigService);
```

Вмикається CORS з налаштуваннями, що дозволяються запити з URL, отриманого з конфігурації та з доменів, що відповідають заданому регулярному виразу, а також дозволяється передача облікових даних:

```
app.enableCors({
origin: [config.get('CLIENT_URL'), 'http://localhost:3001'],
credentials: true,
});
app.use(cookieParser());
```

Використовується глобальний пайп в якому налаштовується чи будуть автоматично видаляються властивості, які не є частиною DTO, чи автоматично перетворюються типи даних відповідно до типів, зазначених у DTO, а також чи припиниться валідація при першій знайденій помилці:

```
app.useGlobalPipes(
new ValidationPipe({
whitelist: true,
transform: true,
stopAtFirstError: true,
}),
);
```

Відбувається ініціалізація Swagger при умові, що розробка відбувається в development режимі:

```
if (config.get('ENV_MODE') === 'development') {
const swagger = new DocumentBuilder()
.setTitle('Faktastisch API')
.setVersion('2.0')
.addBearerAuth()
.build();
const document = SwaggerModule.createDocument(app, swagger);
SwaggerModule.setup('api', app, document);}
```

Додаток починає прослуховувати з'єднання на зазначеному порту:

```
await app.listen(config.get('PORT'));
```

4.2 Вибір архітектури проекту

Nest.js є фреймворком для створення ефективних, масштабованих Node.js додатків на основі TypeScript. Основною ідеєю Nest.js є модульна структура, що дозволяє зручно організовувати код та забезпечує високу підтримуваність і масштабованість проекту. Основними компонентами є модулі сервіси та контролери.

Компоненти в Nest.js пов'язуються через механізм залежностей, який дозволяє автоматично інжектувати необхідні сервіси в контролери або інші сервіси. Це забезпечує високу модульність і можливість повторного використання коду.

За допомогою переваг модульної структури серверна частина вебдодатку Training Club була представлена окремим модулем, який ізольований від інших модулів, що дає змогу максимальної гнучкості в розробці. Структура сервера представлена на рисунку 4.1.

Нижче наведено приклад коду підключення усіх модулів проекту в один потік для подальшого запуску сервера. Лістинг коду головного модулю сервера представлений на ДОДАТКУ К:

```
@Module({
ExerciseCollectionModule,
ExerciseModule,
WorkoutModule,
….
]})
```

Для того щоб була змога використовувати в модулі сервіси на контролери їх потрібно правильно заінджектити в сам модуль де ми хочемо їх використовувати. Приклад коду представлений нижче. Для створення модуля використано декоратор

Module, який приймає три параметра. Перший параметр приймає інші модулі, які ми хочемо використати в обраному модулі, без цього вони працювати не будуть, другий параметр призначений для контролера, а третій для сервіса. Якщо використати їх в модулі напряму – виникне помилка, яка сповістить про некоректне використання та неможливість подальшої роботи обраного модуля:

```
@Module({
 imports: [
  MongooseModule.forFeature([{ name: User.name, schema: UserSchema }]),
  MongooseModule.forFeature([{ name: Token.name, schema: TokenSchema
}]),
 ],
 controllers: [AuthController],
 providers: [TokenService, JwtService, AuthService, MailService],
})
```

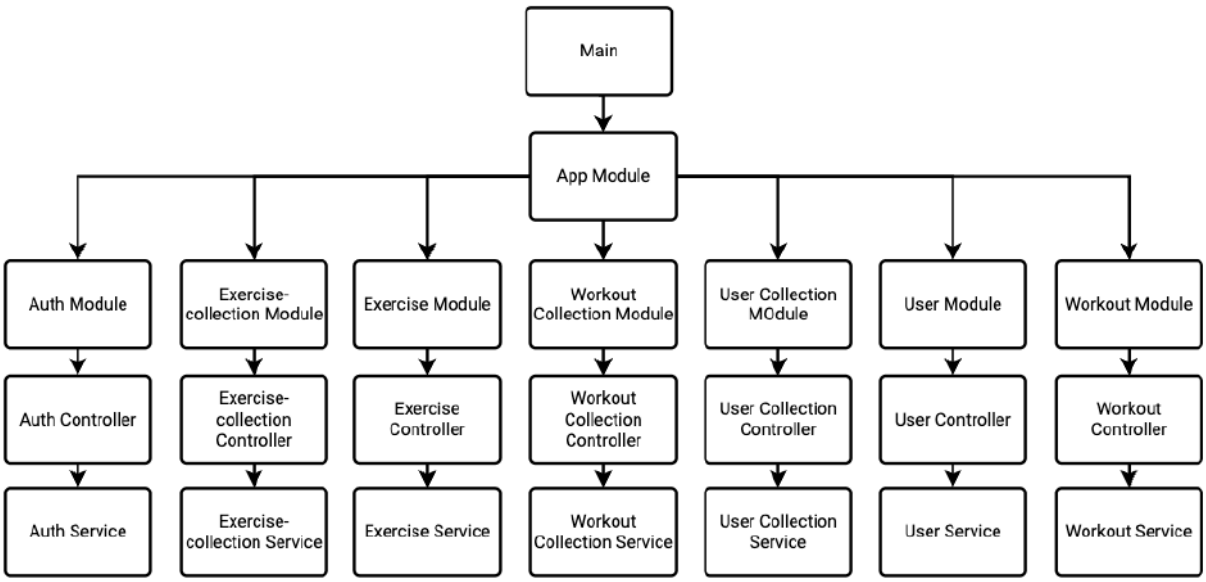

Рисунок 4.1 — Структура сервера

4.3 Вибір та налаштування бази даних

Для збереження текстової інформації обрано базу даних MongoDB. Цей вибір був зумовлений наступними перевагами даної бази даних:

— Дозволяє зберігати дані у форматі, схожому на JSON, що робить можливим легке додавання нових полів або типів даних без переробки всієї бази;

— підтримує горизонтальне масштабування, що дозволяє ефективно розширювати базу даних з ростом потреб проекту;

— швидкість читання та запису в MongoDB висока завдяки оптимізованим запитам та індексації.

База даних була розміщена на сервісі MongoDB Atlas, що дозволило використовувати готовий до використання кластер, забезпечуючи надійність та доступність даних. Для збереження всієї необхідної інформації створено сім колекцій, які включають дані про користувачів, тренування, вправи, а також аналітичні дані для користувачів. Структура цих колекцій представлена на рисунку 4.2.

Рисунок 4.2 — Структура колекції в MongoDB

Нижче наведений код з прикладом підключення бази даних MongoDB до проекту та його пояснення.

Викликається метод forRootAsync модуля MongooseModule для асинхронного налаштування підключення до MongoDB з використанням Mongoose, що дозволяє налаштовувати підключення динамічно:

MongooseModule.forRootAsync({

Вказується, що модуль ConfigModule має бути імпортований. Це дозволяє використовувати конфігураційний сервіс ConfigService для отримання параметрів конфігурації:

imports: [ConfigModule]

Оголошується асинхронна фабрична функція useFactory, яка приймає configService як параметр. configService є екземпляром ConfigService, який використовується для доступу до конфігураційних параметрів. Ця функція повертає об'єкт конфігурації для підключення до MongoDB:

useFactory: async (configService: ConfigService)

Внутрішньо фабрична функція повертає об'єкт конфігурації, що містить uri. Uri отримується за допомогою методу get сервісу ConfigService, який отримує значення параметра MONGODB_URI з конфігураційного файлу або середовища. Цей параметр містить URI для підключення до бази даних MongoDB:

({uri: configService.get('MONGODB_URI')})

Вказується, що фабричній функції необхідно інжектувати ConfigService. Це означає, що ConfigService буде автоматично наданий NestJS і переданий як аргумент фабричної функції:

inject: [ConfigService]

Для збереження медіа файлів обрано сервіс Amazon S3 (Simple Storage Service), який є частиною хмарної платформи AWS. S3 Bucket обрано через наступні переваги:

— Amazon S3 забезпечує 99.9% довговічність даних, що робить його одним з найнадійніших рішень для збереження даних;

— S3 пропонує розширені можливості управління доступом та шифрування даних;

— S3 дозволяє зберігати та обробляти необмежену кількість даних, що робить його ідеальним для проектів будь-якого розміру.

Нижче наведено приклад коду для завантаження медіа файлів у S3 та його повний розбір по стрічково.

Оголошується асинхронний метод upload, який приймає два параметри file, що містить файл, який завантажиться та bucketName, що містить назву сховища в Amazon S3, куди завантажиться файл в подальшому:

async upload(file: Express.Multer.File, bucketName: string)

Створюється об'єкт params, який містить параметри для завантаження файлу в Amazon S3. В параметрах вказується назва сховища, унікальний ідентифікатор файл, вказується сам файл та тип файлу:

```
const params = {
Bucket: bucketName,
Key: uuidv4(),
Body: file.buffer,
ContentType: file.mimetype,
};
```

Створюється нова команда PutObjectCommand з пакету AWS SDK для завантаження об'єкта в S3, використовуючи параметри з об`єкту params, який був заповнений вище:

const command = new PutObjectCommand(params);

Виконується команда PutObjectCommand для завантаження файлу в S3 за допомогою методу send з конфігурації AWS. Оскільки метод асинхронний, використовується await для очікування завершення завантаження:

await this.config.send(command);

Створюється URL завантаженого файлу в S3, використовуючи назву сховища та унікальний ключ файлу:

const imageUrl = `https://${bucketName}.s3.amazonaws.com/${params.Key}`;

Повертається URL завантаженого файлу в S3 як результат виконання методу upload:

return imageUrl;

4.4 Конфігурація Swagger

Swagger — це інструмент, що дозволяє автоматично генерувати документацію для API, забезпечуючи зручний і зрозумілий інтерфейс для розробників та інших користувачів. Інтеграція Swagger в проект є важливим кроком для забезпечення ефективної комунікації між командами розробників, а також для полегшення тестування та налагодження API.

Нижче розібрано код підключення Swagger до проекту з построковим поясненням.

Спочатку створюється асинхронна функція bootstrap, яка є точкою входу для запуску додатка. Використовується NestFactory для створення екземпляра додатка на основі AppModule, який в подальшому використається для підключення Swagger. Функція create є асинхронною, тому використовується await:

const app = await NestFactory.create(AppModule);

Створюється об'єкт конфігурації для Swagger за допомогою класу DocumentBuilder:

const swaggerConfig = new DocumentBuilder()

Створюється об'єкт конфігурації де задається назва API, версія API, додається підтримка аунтефікації за допомогою Bearer токенів:

.setTitle('Faktastisch API')
.setVersion('2.0')
.addBearerAuth()

Використовується SwaggerModule для створення документації Swagger на основі додатку app та конфігурації swaggerConfig:

const document = SwaggerModule.createDocument(app, swaggerConfig);

Налаштовується маршрут "/api" для доступу до документації Swagger. Коли користувач відкриває цей маршрут - відкриється сторінка з документацією API:

SwaggerModule.setup('api', app, document);

Запускається додаток і він починає прослуховувати підключення на порту 3000:

await app.listen(3000);

Для того, щоб Swagger автоматично створював документацію для портів, потрібно в DTO добавляти декоратор ApiProperty. Код з прикладом цієї реалізації представлений нижче.

Оголошується експортований клас LogoutBodyDto. Клас використовується для опису структури даних, що передаються в тілі запиту для операції виходу із системи:

export class LogoutBodyDto {

Декоратор "@IsMongoId" з пакету class-validator вказує, що поле userID повинно бути валідним MongoDB ObjectId. Це забезпечує перевірку даних на етапі валідації, щоб впевнитися, що userID є дійсним MongoDB ObjectId: @IsMongoId().

Декоратор "@ApiProperty" з пакету nestjs/swagger описує властивість userID для документації Swagger. Вказується приклад значення, що допомагає розробникам зрозуміти, яке значення очікується у цьому полі:

@ApiProperty({ example: '645013e9a6a1948d3b16fdd1' })

Оголошується поле userID типу ObjectId в класі LogoutBodyDto. Це поле зберігає ідентифікатор користувача, який передано в тілі запиту для операції виходу з системи:

userID: ObjectId;

Приклад структури згенерованої документації API за допомогою Swagger представлений на рисунку 4.3. Інтерфейс дозволяє переглядати доступні кінцеві точки, параметри запитів та відповіді сервера. Користувач може виконувати запити безпосередньо через веб-інтерфейс, що значно спрощує процес тестування та інтеграції.

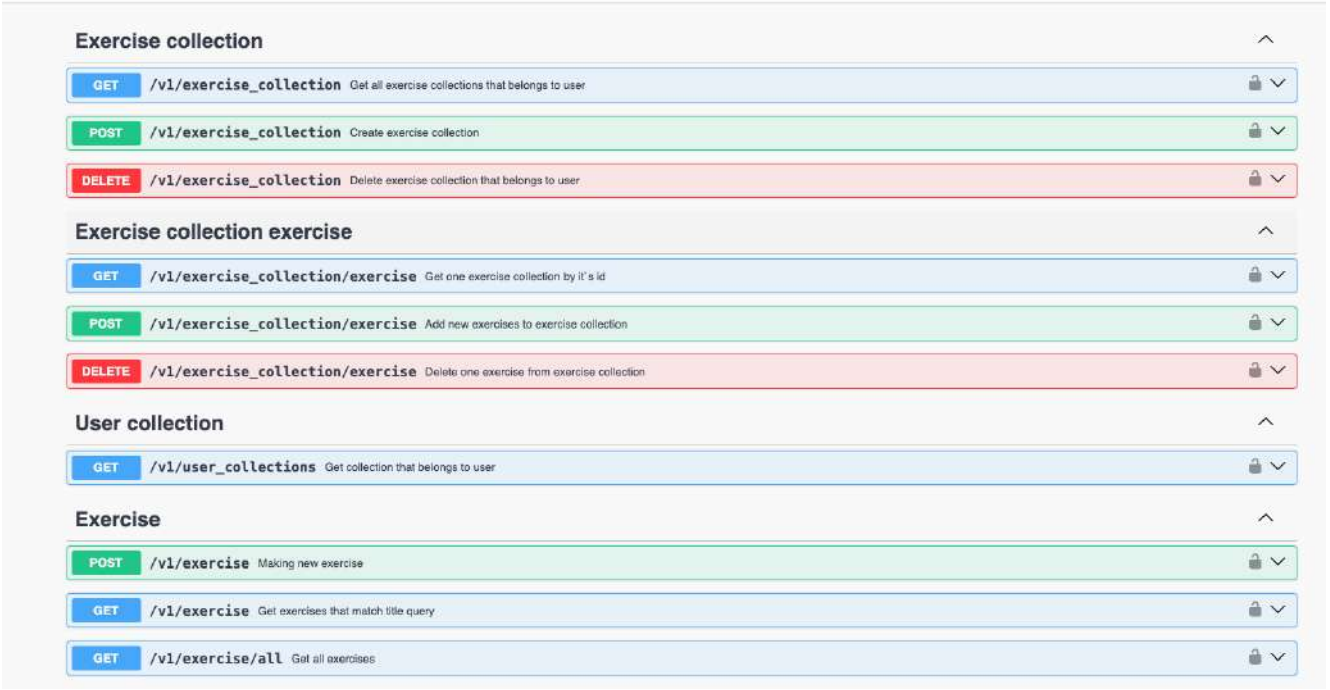

Рисунок 4.3 — Структура Swagger документації

4.5 Налаштування CI/CD

CI/CD (Continuous Integration/Continuous Deployment) — це набір практик та інструментів, що використовуються для автоматизації процесів інтеграції та розгортання коду. Метою CI/CD є покращення якості програмного забезпечення, прискорення процесу розробки та зменшення ризиків, пов'язаних з інтеграцією та розгортанням змін.

CI — це практика регулярного інтегрування змін у кодову базу. Основні переваги CI:

— Автоматичне тестування кожного коміту, що допомагає швидко виявляти та виправляти помилки;

— швидке виявлення конфліктів між різними гілками коду;

— поліпшення якості коду завдяки регулярному тестуванню.

CD — це практика автоматичного розгортання змін у виробниче середовище після успішного проходження тестів. сновні переваги CD:

— Швидке та безпечне розгортання нових версій додатку;

— зменшення ризиків завдяки автоматизації процесів розгортання;

— можливість частого випуску нових функцій та виправлень помилок.

Для реалізації CI/CD використано GitLab, який надає вбудовані інструменти для налаштування конвеєрів автоматизації. Нижче наведено приклад файлу .gitlab-ci.yml, який визначає етапи та завдання для побудови та розгортання додатку, а також проведено детальний розбір коду, повний лістинг коду налаштування CI/CD представлений на ДОДАТКУ Е.

Використовується офіційний образ Docker версії latest як базовий образ для виконання скриптів:

```
image: docker:latest
```

Визначаються етапи для цього CI/CD процесу,а саме збірка та розгортання:

```
stages:
  - build
  - deploy
```

Налаштовується етап збірки. Спочатку виконується запуск допоміжних скриптів, які виконуються перед основною логікою. Копіюється файл з оточення розробки до .env файлу та виконується вхід до Docker Hub за допомогою логіна та токену. Виконується основна логіка збірки проекту. Спочатку створюється Docker образ з тегом, що містить короткий хеш коміту. Потім цей образ відправляється до репозиторію Docker:

```
build:
  stage: build
  before_script:
    - cp ${DEV_ENV_FILE} .env
```

```
    - echo $DOCKER_TOKEN | docker login -u $DOCKER_USER --password-
  stdin
  script:
    - docker build -t ${DOCKER_DEV_REPO}:${CI_COMMIT_SHORT_SHA}
    - docker push ${DOCKER_DEV_REPO}:${CI_COMMIT_SHORT_SHA}
```

Виконуються команди для розгортання додатку. Спочатку оновлюється система та встановлюється програмний пакет sshpass. Потім виконується SSH з'єднання на віддалений сервер та запуск команди для розгортання нової версії за допомогою нового Docker образу. Після цього видаляються невикористані Docker образи:

```
deploy:
  stage: deploy
  image: ruby:latest
  script:
    - apt-get update && apt-get install -y sshpass
    - sshpass -p "$HOSTINGER_PASSWORD" ssh -o StrictHostKeyChecking=no
  root@$HOSTINGER_IP "cd /home/dev_training_back &&
  IMAGE_TAG=$CI_COMMIT_SHORT_SHA docker-compose up -d && docker
  image prune -af"
```

Для розгортання додатку на VPS використано Docker. Docker — це платформа для автоматизації розгортання додатків у контейнерах. Контейнер — це легкий, автономний виконуваний пакет, що містить усе необхідне для запуску додатку: код, runtime, бібліотеки та налаштування.

Переваги Docker:

— контейнери можна запускати на будь-якій платформі, що підтримує Docker, що забезпечує однакове середовище для розробки, тестування та виробництва;

— кожен контейнер працює ізольовано від інших, що забезпечує високу безпеку та стабільність додатків;

— контейнери запускаються значно швидше, ніж віртуальні машини, що дозволяє швидко масштабувати додаток;

— Docker використовує ресурси хоста більш ефективно, ніж традиційні віртуальні машини.

Нижче наведено приклад коду з файлу Dockerfile та його детальним аналізом, який забезпечує створення контейнера, що містить усі необхідні залежності та файли для запуску додатку у середовищі виробництва, повний лістинг коду представлений на ДОДАТКУ Г.

Використовується офіційний образ Node.js як базовий образ. Це забезпечує всі необхідні залежності для роботи з Node.js: FROM node:18.20.

Встановлюється робочий каталог для контейнера. Це означає, що всі наступні команди будуть виконуватися відносно цього каталогу: WORKDIR /app.

Копіюються всі файли з поточного локального каталогу в робочий каталог контейнера: COPY . ..

Виконується встановлення усіх необхідних залежностей для повноцінної роботи серверної частини на Nest.js:

RUN yarn

RUN yarn add @nestjs/cli

Виконується команда для збірки проекту. Це включає компіляцію TypeScript коду, мініфікацію та інші процеси збірки:

RUN yarn build

Встановлюється команда за замовчуванням для запуску контейнера. Це означає запуск Node.js програми в режимі продакшену:

CMD ["yarn", "start:prod"]

4.6 Тестування API

У сучасній розробці програмного забезпечення критичним аспектом є тестування API. Одним з ефективних методів є використання E2E (end-to-end) тестів, які дозволяють перевірити функціональність системи як єдиного цілого. У

цьому розділі я розгляну використання E2E тестів для тестування API на базі фреймворку Nest.js, використовуючи бібліотеку supertest для відправки HTTP запитів.

Для тестування API я використав фреймворк Nest.js, який забезпечує структуровану архітектуру для побудови серверних додатків. Для відправки HTTP запитів під час тестів використовується бібліотека supertest, яка дозволяє легко здійснювати запити до сервера і перевіряти їхні відповіді.

Нижче наведено приклад коду E2E тесту для модуля авторизації та його детальний аналіз. Повний лістинг коду представлений на ДОДАТКУ Д. Цей тест перевіряє функціональність реєстрації нового користувача, що дає можливість завжди мати інформацію про надійність написаної логіки в майбутньому.

Оголошується тестовий блок для реєстрації користувача:

```
describe('User registration', () => {})
```

Визначається окремий тестовий випадок для перевірки успішного створення нового користувача:

```
it('should successfully create a new user', async () => {})
```

Створюється об'єкт, який містить очікувану структуру відповіді з токенами та даними користувача для перевірки у тесті:

```
const userResponse: AuthUserResponseDto = {
...tokens,
user: expect.objectContaining({
avatar: '',
email: userData.email,
nickname: userData.email,
role: UserRoles.USER,
}),
}
```

Виконується HTTP-запит на реєстрацію користувача з відповідними налаштуваннями та очікуваннями. Для успішного запиту задається метода та

шлях, встановлюється заголовок з даними юзера для успішної авторизації на сервері:

```
request(app.getHttpServer())
.post('/v1/auth/sign_up')
.set('user-agent', userAgent)
.send(userData)
```

Після отримання відповіді з сервера йде перевірка даних на відповідність об`єкту, що був створений раніше в тесті. Якщо хоча б одна перевірка вертатиме негативне значення, то тест не пройде і поверне помилку:

```
const { refreshToken } = parseCookies(res.headers['set-cookie'][0])
userID = res.body.user._id
expect(res.body).toEqual(userResponse)
expect(refreshToken).toBe(tokens.refreshToken)
```

Результат тестування auth модуля представлений на рисунку 4.4.

Використання E2E тестів для тестування API дозволяє перевірити роботу додатку в цілому, включаючи взаємодію між різними модулями і компонентами системи. Використання бібліотеки supertest спрощує процес написання тестів для перевірки HTTP запитів та відповідей. Наведений приклад коду демонструє, як можна ефективно організувати і виконувати тести для забезпечення якості та надійності API на базі Nest.js.

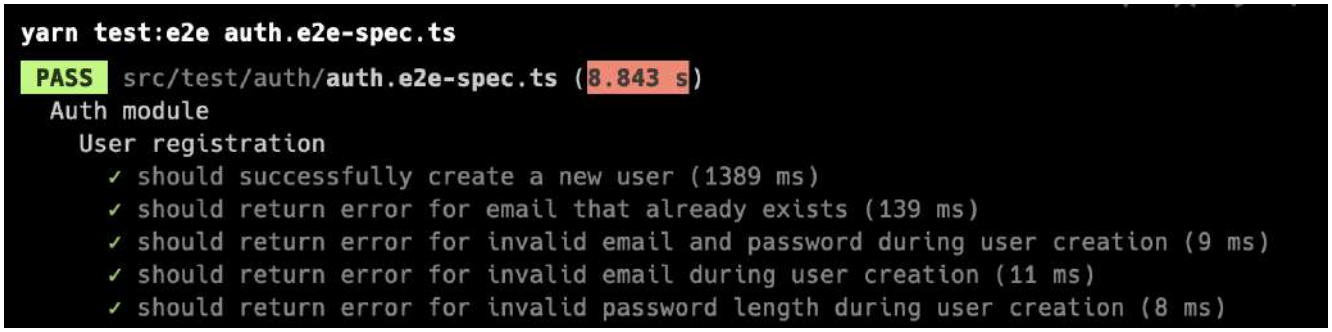

Рисунок 4.4 — Результат тестування auth модуля

ВИСНОВКИ

Для досягнення поставленої мети створення інтерактивного вебдодатку Training Club із зазначеними функціями було виконано наступні задачі.

Поставлену задачу проведення загального огляду веб-технологій було виконано. Визначено, що HTML5, CSS3, та ECMAScript 6 є основними інструментами для розробників. Клієнтські фреймворки та бібліотеки (React, Angular, Vue.js) забезпечують високий рівень інтерактивності та динаміки вебдодатків. На серверному боці технології на основі Node.js (Express, Nest.js) забезпечують високу продуктивність і гнучкість. Використання TypeScript у поєднанні з цими фреймворками покращує якість коду та спрощує підтримку проектів.

Поставлену задачу аналізу сучасних рішень у сфері фітнес-додатків було виконано. Проведено порівняння MyFitnessPal та Sworkit з вебдодатком Training Club, що дозволило виявити його сильні сторони та недоліки. Training Club виділяється можливістю тренерам розробляти власні колекції вправ. Разом з тим, було виявлено обмежений функціонал у безкоштовному режимі та відсутність фокусування на відстеженні харчування.

Поставлену задачу розробки вимог до структури вебдодатку було виконано. Визначено, що вебдодаток повинен мати інтуїтивну систему навігації, динамічну маршрутизацію, естетичний дизайн, зручність для користувачів та адміністративну панель для ефективного керування контентом та користувачами.

Поставлену задачу розробки клієнтської частини було виконано. Навігацію по вебдодатку було реалізовано за допомогою бібліотеки React Router, що покращило користувацький досвід. Логіку push-повідомлень було налаштовано для оперативного інформування користувачів про важливі події та оновлення. Тестування клієнтської частини було проведено для виявлення та виправлення помилок, що забезпечило високу якість кінцевого продукту.

Поставлену задачу розробки серверної частини було виконано. Вибрано архітектуру проекту на основі фреймворка NestJS, що забезпечило структурованість коду та гнучкість. Базу даних було вибрано та налаштовано для

надійного та ефективного зберігання даних. Систему CI/CD було налаштовано для автоматизації процесів тестування та розгортання вебдодатку. Проведено тестування API, що включало юніт-тести, інтеграційні та функціональні тести, забезпечивши стабільність та надійність серверної частини вебдодатку.

Загалом, реалізація вебдодатку Training Club була успішно виконана завдяки комплексному підходу до розробки, використанню сучасних технологій та ретельному тестуванню. Це дозволило створити зручний, функціональний та надійний вебдодаток, що задовольняє потреби користувачів та відповідає високим стандартам якості.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. HTML. Вікіпедія. [Електронний ресурс]. Доступно за адресою: https://uk.wikipedia.org/wiki/HTML (дата звернення: 10.05.2024). – Назва з екрана.

2. CSS. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://uk.wikipedia.org/wiki/HTML (дата звернення: 10.05.2024). – Назва з екрана.

3. JavaScript Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide (дата звернення: 10.05.2024). – Назва з екрана.

4. React. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 10.05.2024). — Назва з екрана.

5. Vue.js. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://vuejs.org/guide/introduction (дата звернення: 10.05.2024). — Назва з екрана.

6. Angular. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://angular.io/ (дата звернення: 11.05.2024). — Назва з екрана.

7. Node js. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://nodejs.org/docs/latest/api/ (дата звернення: 11.05.2024). — Назва з екрана.

8. Express. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://expressjs.com/ (дата звернення: 12.05.2024). — Назва з екрана.

9. Laravel. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://laravel.com/docs/11.x/readme (дата звернення: 12.05.2024). — Назва з екрана.

10. Material UI. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://mui.com/ (дата звернення: 12.05.2024). – Назва з екрана.

11. TypeScript. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://www.typescriptlang.org/ (дата звернення: 14.05.2024). – Назва з екрана.

12. MongoDB. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://www.mongodb.com/docs/ (дата звернення: 14.05.2024). – Назва з екрана.

13. Jest для тестування. Документація. [Електронний ресурс]. Доступно за адресою: https://jestjs.io/ (дата звернення: 14.05.2024). – Назва з екрана.

14. React Router. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://reactrouter.com/en/main (дата звернення: 14.05.2024). – Назва з екрана.

15. OneSignal. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://onesignal.com/unlock-growth-ga-lp (дата звернення: 14.05.2024). – Назва з екрана.

16. React Hook Form. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://react-hook-form.com/ (дата звернення: 16.05.2024). – Назва з екрана.

17. Swagger. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://swagger.io/docs/ (дата звернення: 17.05.2024). – Назва з екрана.

18. Docker. Офіційна документація. [Електронний ресурс]. Доступно за адресою: https://docs.docker.com/ (дата звернення: 17.05.2024). – Назва з екрана.

ДОДАТОК А

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп’ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., проф Азаров О.Д.
«06» травня 2024 року

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання бакалаврської дипломної роботи
«Інтерактивний вебдодаток Training Club»
08-54.БДР.004.00.000.ТЗ

Керівник роботи: к.т.н., ст. викл. каф.ОТ
_____ Черняк О. І.

Виконав: студент гр. КІ-22мсз
_____Ткачук М. О.

ВНТУ 2024

1 Підстави для виконання бакалаврської дипломної роботи (БДР)

1.1 Актуальність дослідження обумовлена зростанням попиту на інструменти для підтримки активного способу життя та віртуальні фітнес-рішення. У сучасному світі онлайн-сервіси стали невід'ємною частиною повсякденного життя. Вебдодаток Training Club, що інтегрує різноманітний фітнес-контент і тренерів, надає користувачам можливість обирати методики фізичних тренувань, тим самим сприяючи популяризації здорового способу життя та збереженню здоров'я серед різних верств населення. Це робить проект актуальним та важливим, враховуючи сучасні тенденції в сфері фітнесу та технологій.

1.2 Наказ про затвердження теми БДР

2 Мета БДР і призначення розробки

2.1 Мета роботи — розробка інтерактивного вебдодатку Training Club, який інтегрує різноманітний фітнес-контент і тренерів для надання користувачам можливості вільного вибору методик фізичних тренувань, створення власних тренувальних програм та отримання персоналізованих рекомендацій;

2.2 Призначення розробки — виконання бакалаврської дипломної роботи із подальшою підтримкою та розвитком;

3 Вихідні дані для виконання БДР

3.1 Обґрунтування доцільності створення програмного засобу.

3.2 Проведення аналізу сучасних технологій для реалізації вебдодатків.

3.3 Огляд існуючих аналогів фітнес-додатків.

3.4 Розробка структурної схеми вебдодатку.

3.5 Розробка функціоналу вебдодатку.

4 Вимоги до виконання БДР

Програмний засіб із вебінтерфейсом має:

— коректно відповідати на запити користувачів;

— надавати можливість тренерам розробляти та ділитися комплексами вправ;

— забезпечувати користувачам зручний доступ до тренувальних програм;

— обробляти помилки та забезпечувати безпеку даних користувачів.

5 Етапи БДР та очікувані результати

Етапи роботи та очікувані результати приведено в таблиці А.1.

Таблиця А.1 — Етапи БДР

№ з/п	Назва етапів дипломної роботи	Термін виконання		Очікувані результати
		початок	закінчення	
1	Аналіз завдання	05.03.2024	07.03.2024	Задачі дослідження
2	Аналіз сучасних веб–технологій	8.03.2024	11.03.2024	Розділ 1
3	Порівняльна характеристика фітнес–додатків	12.03.2024	18.03.2024	Розділ 2
4	Розробка клієнтської частини	18.03.2024	25.04.2024	Розділ 3
5	Розробка серверної частини	25.04.2024	19.05.2024	Розділ 4
6	Тестування та розгортання прогамного засобу	19.05.2024	22.05.2024	Розділ 4
7	Оформлення пояснювальної записки та ілюстративного матеріалу	22.05.2024	10.06.2024	ПЗ, додатки, презентація

6 Матеріали, що подаються до захисту БДР

До захисту подаються: пояснювальна записка БДР, текстові матеріали, протокол попереднього захисту роботи на кафедрі, відгук наукового керівника, рецензія опонента, анотації до БДР українською та іноземною мовами, нормоконтроль про відповідність оформлення БДР діючим вимогам;

7 Порядок контролю виконання та захисту БДР

Виконання етапів графічної та розрахункової документації БДР контролюється науковим керівником згідно зі встановленими термінами. Захист БДР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлювання та порядок виконання БДР

8.1 При оформлювання БДР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання бакалаврських кваліфікаційних робіт зі спеціальності 123 – «Комп’ютерна інженерія» (освітня програма «Комп’ютерна інженерія»). Кафедра обчислювальної техніки ВНТУ 2022;

8.2 Порядок виконання БДР викладено в «Положення про кваліфікаційні роботи на першому (бакалаврському) рівні вищої освіти СУЯ ВНТУ-03.02.02-П.001.01:21».

ДОДАТОК Б

Лістинг коду навігації по вебдодатку

```
export const Router = createBrowserRouter([
  {
    path: '/',
    element: <AuthenticationLayout />,
    errorElement: <Error />,
    children: [
      {
        path: RoutesPath.HOME,
        element: (
          <NavigateMenuProvider>
            <Home />
          </NavigateMenuProvider>
        ),
      },
      { path: RoutesPath.PRE_WORKOUT, element: <PreWorkout /> },
      { path: RoutesPath.WORKOUT, element: <WorkoutPage /> },
      { path: RoutesPath.WORKOUT_BUILDER, element: <WorkoutBuilder /> },
      { path: RoutesPath.NEW_EXERCISE, element: <NewExercise /> },
      { path: RoutesPath.TRAINING_CONFIRMATION, element:
<WorkoutConfirmation /> },
      { path: RoutesPath.RECENT_WORKOUTS, element: <RecentWorkouts /> },
      { path: RoutesPath.EXERCISE_LIBRARY, element: <ExerciseLibrary /> },
      { path: RoutesPath.SIGN_UP, element: <SignUp /> },
      { path: RoutesPath.SIGN_IN, element: <SignIn /> },
      { path: RoutesPath.AUTH, element: <Auth /> },
      { path: RoutesPath.SAVED_WORKOUTS, element: <SavedWorkouts /> },
      { path: RoutesPath.GREETING, element: <Greeting /> },
      { path: RoutesPath.CONTACT_US, element: <ContactUs /> },
```

```
{ path: RoutesPath.ADMIN, element: <Admin /> },
{
  path: RoutesPath.SETTINGS,
  element: <SettingsLayout />,
  children: [
    { path: RoutesPath.SETTINGS, element: <Settings /> },
    { path: RoutesPath.USER_PROFILE, element: <UserProfile /> },
    { path: RoutesPath.NOTIFICATIONS, element: <Notifications /> },
  ],
},
{
  path: RoutesPath.ADMIN_STORE_WORKOUT_UPDATE,
  element: <UpdateWorkout />,
},
{
  path: RoutesPath.ADMIN_STORE_WORKOUT_CHECK,
  element: <WorkoutCheck />,
},
{
  path: RoutesPath.ADMIN,

  element: <AdminLayout />,
  children: [
    {
      path: RoutesPath.ADMIN,
      element: <Admin />,
    },
    {
      path: RoutesPath.ADMIN_STORE,
      children: [
```

```
    {
      path: RoutesPath.ADMIN_STORE,
      element: <AdminStore />,
    },
    {
      path: RoutesPath.ADMIN_STORE_WORKOUT_COLLECTIONS,
      element: <AdminStoreWorkouts />,
    },
    {
      path: RoutesPath.ADMIN_STORE_WORKOUT_COLLECTION,
      element: <AdminStoreWorkoutCollection />,
    },
    {
      path: RoutesPath.ADMIN_STORE_WORKOUT_PROFILE,
      element: <WorkoutProfile />,
    },
    {
      path: RoutesPath.ADMIN_STORE_EXERCISE_COLLECTIONS,
      element: <AdminStoreExercises />,
    },
    {
      path: RoutesPath.ADMIN_STORE_EXERCISE_COLLECTION,
      element: <AdminStoreExerciseCollection />,
    },
    {
      path: RoutesPath.ADMIN_STORE_TRASH,
      element: <AdminStoreTrash />,
    },
  ],
},
```

```
        ],
      },
    ],
  },
])
```

ДОДАТОК В

Лістинг коду push-повідомлень

```
export const usePushNotifications = () => {
  const isSubscribed = getFromLocalStorage(LocalStorageKey.NOTIFICATION) === 'true'
  const isInitNotification =
    getFromLocalStorage(LocalStorageKey.IS_INIT_NOTIFICATION) === 'true'

  const init = async (): Promise<void> => {
    if (!isInitNotification) {
      OneSignal.init({
        appId: ONE_SIGNAL_APP_ID,
        allowLocalhostAsSecureOrigin: true,
      })

      setToLocalStorage(LocalStorageKey.IS_INIT_NOTIFICATION, true)
      setToLocalStorage(LocalStorageKey.NOTIFICATION, true)

      const subscriptionCallback = async (e: SubscriptionChangeEventT) => {
        if (!e.current.id) return

        await User.sendNotification(e.current.id)

        OneSignal.User.PushSubscription.removeEventListener(
          'change',
          subscriptionCallback,
        )
      }

      OneSignal.User.PushSubscription.addEventListener(
```

```
        'change',
        subscriptionCallback,
      )
    }
  }

  const subscribe = async (): Promise<void> => {
    OneSignal.User.PushSubscription.optIn()

    setToLocalStorage(LocalStorageKey.NOTIFICATION, true)
  }

  const unSubscribe = async (): Promise<void> => {
    OneSignal.User.PushSubscription.optOut()
    setToLocalStorage(LocalStorageKey.NOTIFICATION, false)
  }

  return { subscribe, unSubscribe, isSubscribed, init }
}
```

ДОДАТОК Г

Лістинг коду створення docker образу

```
FROM node:18.20
WORKDIR /app
COPY . .
RUN yarn
RUN yarn add @nestjs/cli
RUN yarn build
COPY ./src/cdn/fonts /usr/share/fonts/truetype/outfit
RUN fc-cache -f -v
CMD ["yarn", "start:prod"]
```

ДОДАТОК Д

Лістинг коду тестів модуля auth

```
describe('Auth module', () => {
  let app: INestApplication
  let tokenService: TokenService

  beforeAll(async () => {
    const moduleFixture: TestingModule = await Test.createTestingModule({
      imports: [AppModule],
    }).compile()

    app = await initApplication(moduleFixture)

    tokenService = moduleFixture.get<TokenService>(TokenService)
  })

  afterAll(async () => {
    await app.close()
  })

  afterEach(() => {
    jest.restoreAllMocks()
  })

  const userData = {
    email: `${uuidv4()}@example.com`,
    password: '123456789',
  }
  const userAgent =
```

```
    'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.0.0 Safari/537.36'

  let userID = ''

  let tokens = {} as GenerateTokensT

  describe('User registration', () => {
    it('should successfully create a new user', async () => {
      tokens = tokenService.generateTokens(userData)

      const userResponse: AuthUserResponseDto = {
        ...tokens,
        user: expect.objectContaining({
          avatar: '',
          email: userData.email,
          nickname: userData.email,
          role: UserRoles.USER,
        }),
      }
      jest
        .spyOn(TokenService.prototype, 'generateTokens')
        .mockReturnValue(tokens)

      jest
        .spyOn(MailService.prototype, 'sendActivationMain')
        .mockImplementationOnce(() => Promise.resolve())

      return request(app.getHttpServer())
        .post('/v1/auth/sign_up')
```

```
      .set('user-agent', userAgent)
      .send(userData)
      .expect('Content-Type', /json/)
      .expect(HttpStatus.CREATED)
      .then(res => {
        const { refreshToken } = parseCookies(res.headers['set-cookie'][0])

        userID = res.body.user._id

        expect(res.body).toEqual(userResponse)
        expect(refreshToken).toBe(tokens.refreshToken)
      })
  })

  it('should return error for email that already exists', async () => {
    return request(app.getHttpServer())
      .post('/v1/auth/sign_up')
      .set('user-agent', userAgent)
      .send(userData)
      .expect('Content-Type', /json/)
      .expect(HttpStatus.BAD_REQUEST)
      .then(res => {
        expect(res.body.message).toBe(
          `The user with the current address ${userData.email} already exists`,
        )
      })
  })

  it('should return error for invalid email and password during user creation', async ()
=> {
```

```
  const errors = [
    'email must be an email',
    'password must be longer than or equal to 8 characters',
  ]

  return request(app.getHttpServer())
    .post('/v1/auth/sign_up')
    .set('user-agent', userAgent)
    .expect('Content-Type', /json/)
    .expect(HttpStatus.BAD_REQUEST)
    .then(res => {
      expect(res.body.errors).toEqual(errors)
    })
})

it('should return error for invalid email during user creation', async () => {
  const errors = ['email must be an email']

  return request(app.getHttpServer())
    .post('/v1/auth/sign_up')
    .set('user-agent', userAgent)
    .send({ password: '12345678' })
    .expect('Content-Type', /json/)
    .expect(HttpStatus.BAD_REQUEST)
    .then(res => {
      expect(res.body.errors).toEqual(errors)
    })
})

it('should return error for invalid password length during user creation', async () => {
```

```
    const errors = ['password must be longer than or equal to 8 characters']

    return request(app.getHttpServer())
      .post('/v1/auth/sign_up')
      .set('user-agent', userAgent)
      .send({ email: 'test@gmail.com' })
      .expect('Content-Type', /json/)
      .expect(HttpStatus.BAD_REQUEST)
      .then(res => {
        expect(res.body.errors).toEqual(errors)
      })
  })
})

describe('User authorization', () => {
  it('should successfully to authorize a user', async () => {
    tokens = tokenService.generateTokens(userData)

    const userResponse: AuthUserResponseDto = {
      ...tokens,
      user: expect.objectContaining({
        avatar: '',
        email: userData.email,
        nickname: userData.email,
        role: UserRoles.USER,
      }),
    }
    jest
      .spyOn(TokenService.prototype, 'generateTokens')
      .mockReturnValue(tokens)
```

```
  return request(app.getHttpServer())
    .post('/v1/auth/sign_in')
    .set('user-agent', userAgent)
    .send(userData)
    .expect('Content-Type', /json/)
    .expect(HttpStatus.CREATED)
    .then(res => {
      const { refreshToken } = parseCookies(res.headers['set-cookie'][0])

      userID = res.body.user._id

      expect(res.body).toEqual(userResponse)
      expect(refreshToken).toBe(tokens.refreshToken)
    })
})

it('should return error if user nonexistent', async () => {
  return request(app.getHttpServer())
    .post('/v1/auth/sign_in')
    .set('user-agent', userAgent)
    .send({ ...userData, email: 'nonexistent@mail.com' })
    .expect('Content-Type', /json/)
    .expect(HttpStatus.BAD_REQUEST)
    .then(res => {
      expect(res.body.message).toBe('User with such email was not found')
    })
})
```

```
    it('should return error for invalid email and password during user authorization',
async () => {
      const errors = [
        'email must be an email',
        'password must be longer than or equal to 8 characters',
      ]

      return request(app.getHttpServer())
        .post('/v1/auth/sign_in')
        .set('user-agent', userAgent)
        .expect('Content-Type', /json/)
        .expect(HttpStatus.BAD_REQUEST)
        .then(res => {
          expect(res.body.errors).toEqual(errors)
        })
    })

    it('should return error for invalid email during user authorization', async () => {
      const errors = ['email must be an email']

      return request(app.getHttpServer())
        .post('/v1/auth/sign_in')
        .set('user-agent', userAgent)
        .send({ password: '12345678' })
        .expect('Content-Type', /json/)
        .expect(HttpStatus.BAD_REQUEST)
        .then(res => {
          expect(res.body.errors).toEqual(errors)
        })
    })
```

```
  it('should return error for invalid password length during authorization', async () => {
    const errors = ['password must be longer than or equal to 8 characters']

    return request(app.getHttpServer())
      .post('/v1/auth/sign_in')
      .set('user-agent', userAgent)
      .send({ email: 'test@gmail.com' })
      .expect('Content-Type', /json/)
      .expect(HttpStatus.BAD_REQUEST)
      .then(res => {
        expect(res.body.errors).toEqual(errors)
      })
  })
})

describe('User refresh token', () => {
  it('should successfully refresh token for user', async () => {
    tokens = tokenService.generateTokens({ ...userData, _id: userID })

    const userResponse: AuthUserResponseDto = {
      ...tokens,
      user: expect.objectContaining({
        avatar: '',
        email: userData.email,
        nickname: userData.email,
        role: UserRoles.USER,
      }),
    }
```

```
    jest
      .spyOn(TokenService.prototype, 'generateTokens')
      .mockReturnValue(tokens)

    return request(app.getHttpServer())
      .get('/v1/auth/refresh_token')
      .set('user-agent', userAgent)
      .set('Cookie', `refreshToken=${tokens.refreshToken}`)
      .expect('Content-Type', /json/)
      .expect(HttpStatus.OK)
      .then(res => {
        const { refreshToken } = parseCookies(res.headers['set-cookie'][0])

        expect(res.body).toEqual(userResponse)
        expect(refreshToken).toBe(tokens.refreshToken)
      })
  })

  it('should return error if refresh does not pass', async () => {
    jest
      .spyOn(TokenService.prototype, 'generateTokens')
      .mockReturnValue(tokens)

    return request(app.getHttpServer())
      .get('/v1/auth/refresh_token')
      .set('user-agent', userAgent)
      .expect(HttpStatus.UNAUTHORIZED)
      .then(res => {
        expect(res.body.message).toBe('Token is undefined')
      })
```

```
  })
})

describe('User logout', () => {
  it('should successfully logout a user', async () => {
    return request(app.getHttpServer())
      .post('/v1/auth/logout')
      .send({ userID })
      .expect(HttpStatus.CREATED)

      .then(res => {
        const { refreshToken } = parseCookies(res.headers['set-cookie'][0])

        expect(res.body).toEqual({})
        expect(refreshToken).toBe('')
      })
  })

  it('should return error if user nonexistent', async () => {
    return request(app.getHttpServer())
      .post('/v1/auth/logout')
      .send({ userID: '6624e20209cb98f1f009ff30' })
      .expect(HttpStatus.NOT_FOUND)
      .then(res => {
        expect(res.body.message).toBe('User not found')
      })
  })

  it('should return error if invalid userID ', async () => {
    const errors = ['userID must be a mongodb id']
```

```
    return request(app.getHttpServer())
      .post('/v1/auth/logout')
      .send({ userID: '1111' })
      .expect(HttpStatus.BAD_REQUEST)
      .then(res => {
        expect(res.body.errors).toEqual(errors)
      })
  })
})

describe('Activation link ', () => {
  // not urgent
})
                                                    })
```

ДОДАТОК Е

Лістинг коду CI/CD

```
image: docker:latest
stages:
  - build
  - deploy

services:
  - docker:dind

develop_build:
  stage: build
  before_script:
    - cp ${DEV_ENV_FILE} .env
    - echo $DOCKER_TOKEN | docker login -u $DOCKER_USER --password-stdin
  script:
    - docker build -t ${DOCKER_DEV_REPO}:${CI_COMMIT_SHORT_SHA} .
    - docker push ${DOCKER_DEV_REPO}:${CI_COMMIT_SHORT_SHA}
  only:
    - dev



production_build:
  stage: build
  before_script:
    - cp ${PROD_ENV_FILE} .env
    - echo $DOCKER_TOKEN | docker login -u $DOCKER_USER --password-stdin
  script:
    - docker build -t ${DOCKER_PROD_REPO}:${CI_COMMIT_SHORT_SHA} .
    - docker push ${DOCKER_PROD_REPO}:${CI_COMMIT_SHORT_SHA}
```

```
  only:
    - main

dev_deploy:
  stage: deploy
  image: ruby:latest
  script:
    - apt-get update && apt-get install -y sshpass
    - sshpass -p "$HOSTINGER_PASSWORD" ssh -o StrictHostKeyChecking=no
root@$HOSTINGER_IP "cd /home/dev_training_back &&
IMAGE_TAG=$CI_COMMIT_SHORT_SHA docker-compose up -d && docker
image prune -af"
  only:
    - dev

prod_deploy:
  stage: deploy
  image: ruby:latest
  script:
    - apt-get update && apt-get install -y sshpass
    - sshpass -p "$HOSTINGER_PASSWORD" ssh -o StrictHostKeyChecking=no
root@$HOSTINGER_IP "cd /home/trx_training_back &&
IMAGE_TAG=$CI_COMMIT_SHORT_SHA docker-compose up -d && docker
image prune -af"
  only:
    - main
```

ДОДАТОК Ж

Лістинг коду ініціалізації програми

```
async function bootstrap(): Promise<void> {
  const app = await NestFactory.create(AppModule);
  const config = app.get(ConfigService);

  app.enableCors({
    origin: [
      config.get('CLIENT_URL'),
      'http://localhost:3001',
      /https:\/\/training-club-.+-trainingclubs-projects\.vercel\.app/,
    ],
    credentials: true,
  });

  app.use(cookieParser());

  app.useGlobalPipes(
    new ValidationPipe({
      whitelist: true,
      transform: true,
      stopAtFirstError: true,
    }),
  );

  app.useGlobalFilters(new HttpExceptionFilter(), new AuthExceptionFilter());
```

```
  if (config.get('ENV_MODE') === 'development') {
    const swagger = new DocumentBuilder()
      .setTitle('Faktastisch API')
      .setVersion('2.0')
      .addBearerAuth()
      .build();

    const document = SwaggerModule.createDocument(app, swagger);
    SwaggerModule.setup('api', app, document);
  }

  useContainer(app.select(AppModule), {
    fallbackOnErrors: true,
    fallback: true,
  });

  await app.listen(config.get('PORT'));
}
```

ДОДАТОК И

Лістинг коду тестів функції trainingConversion

```
beforeAll(() => {
  vi.mock('src/utils', () => {
    return {
      getUniqueId: () => 1111,
    }
  })
})

describe('parseTraining', () => {
  test('The number of elements returned by the function', () => {
    expect(parseTraining(trainingResponse)).toMatchObject(parseTrainingResult)
  })

  test('The object that the function should return', () => {
    expect(parseTraining(trainingResponse)).toEqual(parseTrainingResult)
  })

  test('The function returns an invalid value', () => {
    expect(parseTraining(trainingResponse)).not.toEqual({})
  })
})

describe('prepareTrainingSessions', () => {
  test('The number of elements returned by the function', () => {
    expect(prepareTrainingSessions(trainingResponse.warmUp)).toMatchObject(
      prepareTrainingSessionsResult,
    )
  })
```

```
  test('The object that the function should return', () => {
    expect(prepareTrainingSessions(trainingResponse.warmUp)).toEqual(
      prepareTrainingSessionsResult,
    )
  })

  test('The function returns an invalid value', () => {
    expect(prepareTrainingSessions(trainingResponse.warmUp)).not.toEqual({})
  })
})

describe('serializeTrainings', () => {
  test('The number of elements returned by the function', () => {
    expect(
      serializeTrainings(trainingResponse.warmUp[0].trainingSessions),
    ).toMatchObject(serializeTrainingsResult)
  })

  test('The object that the function should return', () => {
    expect(serializeTrainings(trainingResponse.warmUp[0].trainingSessions)).toEqual(
      serializeTrainingsResult,
    )
  })

  test('The function returns an invalid value', () => {
    expect(
      serializeTrainings(trainingResponse.warmUp[0].trainingSessions),
    ).not.toEqual({})
  })
```

```
})

describe('prepareTrainingForSetsWithReps', () => {
  const training = serializeTrainings(trainingResponse.warmUp[0].trainingSessions)

  test('The number of elements returned by the function', () => {
    expect(
      prepareTrainingForSetsWithReps(training, trainingResponse.warmUp[0].sets),
    ).toMatchObject(prepareTrainingForSetsWithRepsResult)
  })

  test('The object that the function should return', () => {
    expect(
      prepareTrainingForSetsWithReps(training, trainingResponse.warmUp[0].sets),
    ).toEqual(prepareTrainingForSetsWithRepsResult)
  })

  test('The function returns an invalid value', () => {
    expect(
      prepareTrainingForSetsWithReps(training, trainingResponse.warmUp[0].sets),
    ).not.toEqual({})
  })
})

describe('separateOnWorkAndRest', () => {
  const training = serializeTrainings(trainingResponse.warmUp[0].trainingSessions)

  test('The number of elements returned by the function', () => {
    expect(separateOnWorkAndRest(training)).toMatchObject(
      separateOnWorkAndRestResult,
```

```
    )
  })

  test('The object that the function should return', () => {
    expect(separateOnWorkAndRest(training)).toEqual(separateOnWorkAndRestResult)
  })

  test('The function returns an invalid value', () => {
    expect(separateOnWorkAndRest(training)).not.toEqual({})
  })
})

describe('setRestIfSetHaveRest', () => {
  const training = separateOnWorkAndRest(
    serializeTrainings(trainingResponse.warmUp[0].trainingSessions),
  )

  test('The number of elements returned by the function', () => {
    expect(
      setRestIfSetHaveRest(training, trainingResponse.warmUp[0].sets),
    ).toMatchObject(setRestIfSetHaveRestResult)
  })

  test('The object that the function should return', () => {
    expect(setRestIfSetHaveRest(training, trainingResponse.warmUp[0].sets)).toEqual(
      setRestIfSetHaveRestResult,
    )
  })

  test('The function returns an invalid value', () => {
```

```
    expect(
      setRestIfSetHaveRest(training, trainingResponse.warmUp[0].sets),
    ).not.toEqual({})
  })
})

describe('addUniqIdForTrainingsResult', () => {
  // @ts-ignore
  const workout = addUniqIdForTrainings(removeRestPageAtEndOfListResult)

  test('The number of elements returned by the function', () => {

expect(addUniqIdForTrainings(workout)).toMatchObject(addUniqIdForTrainingsResult
)
  })

  test('The object that the function should return', () => {
    expect(addUniqIdForTrainings(workout)).toEqual(addUniqIdForTrainingsResult)
  })

  test('The function returns an invalid value', () => {
    expect(addUniqIdForTrainings(workout)).not.toEqual({})
  })
})

describe('setRightPrepareForField', () => {
  const training = prepareTrainingSessions(trainingResponse.warmUp)

  test('The number of elements returned by the function', () => {
    expect(setRightPrepareForField(training)).toMatchObject(
```

```
    setRightPrepareForFieldResult,
   )
  })

  test('The object that the function should return', () => {
   expect(setRightPrepareForField(training)).toEqual(setRightPrepareForFieldResult)
  })

  test('The function returns an invalid value', () => {
   expect(setRightPrepareForField(training)).not.toEqual({})
  })
})

describe('setPreparePage', () => {
  const training = setRightPrepareForField(
   prepareTrainingSessions(trainingResponse.warmUp),
  )

  test('The number of elements returned by the function', () => {
   expect(setPreparePage(training)).toMatchObject(setPreparePageResult)
  })

  test('The object that the function should return', () => {
   expect(setPreparePage(training)).toEqual(setPreparePageResult)
  })

  test('The function returns an invalid value', () => {
   expect(setPreparePage(training)).not.toEqual({})
  })
})
```

```
describe('removeRestPageAtEndOfList', () => {
  const training = setPreparePage(
    setRightPrepareForField(prepareTrainingSessions(trainingResponse.warmUp)),
  )

  test('The number of elements returned by the function', () => {
    expect(removeRestPageAtEndOfList(training)).toMatchObject(
      removeRestPageAtEndOfListResult,
    )
  })

  test('The object that the function should return', () => {
    expect(removeRestPageAtEndOfList(training)).toEqual(
      removeRestPageAtEndOfListResult,
    )
  })

  test('The function returns an invalid value', () => {
    expect(removeRestPageAtEndOfList(training)).not.toEqual({})
  })
})
```

ДОДАТОК К

Лістинг коду головного модуля

```
import * as path from 'path'
import process from 'process'

import { Module } from '@nestjs/common'
import { ConfigModule, ConfigService } from '@nestjs/config'
import { MongooseModule } from '@nestjs/mongoose'
import { ServeStaticModule } from '@nestjs/serve-static'

import { AuthModule } from './modules/auth/auth.module'
import { ExerciseModule } from './modules/exercise/exercise.module'
import { ExerciseCollectionModule } from './modules/exercise-collection/exercise-
collection.module'
import { FeedbackModule } from './modules/feedback/user-collection.module'
import { ConfigValidationService } from './modules/shared/services/config-validation-
service'
import { TokenModule } from './modules/token/token.module'
import { UserModule } from './modules/user/user.module'
import { UserAnalyticsModule } from './modules/user-analytics/user-collection.module'
import { UserCollectionModule } from './modules/user-collection/user-
collection.module'
import { WorkoutModule } from './modules/workout/workout.module'
import { WorkoutCollectionModule } from './modules/workout-collection/workout-
collection.module'

@Module({
  imports: [
    ConfigModule.forRoot({
      isGlobal: true,
```

```
      validationSchema: ConfigValidationService.createSchema(),
    }),
    MongooseModule.forRootAsync({
      imports: [ConfigModule],
      useFactory: async (configService: ConfigService) => ({
        uri: configService.get('MONGODB_URI'),
      }),
      inject: [ConfigService],
    }),
    ServeStaticModule.forRoot({
      rootPath: path.join(process.cwd(), 'src', 'cdn'),
      serveRoot: '/static/cdn',
    }),
    ExerciseCollectionModule,
    ExerciseModule,
    WorkoutModule,
    UserAnalyticsModule,
    AuthModule,
    TokenModule,
    UserCollectionModule,
    UserModule,
    FeedbackModule,
    WorkoutCollectionModule,
  ],
})
export class AppModule {}
```

ДОДАТОК Л

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: "Інтерактивний вебдодаток Training Club"

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,4% Схожість 1,6%

Аналіз звіту подібності (відмітити потрібне):

☐ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Ткачук М. О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Черняк О. І.
(підпис) (прізвище, ініціали)