МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ФІТНЕС-ЗАСТОСУНКУ ЗАСОБАМИ REACT NATIVE ТА NODE.JS»**

Виконав: студент 2 курсу, групи 8.1211-1іпз
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
І.М. Неділько
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, к.ф.-м.н., Кривохата А.Г.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри фундаментальної та прикладної математики, к.ф.-м.н., Манько Н.І.-В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А. О.
(підпис)

“ ____ ” _____ 2022 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Неділько Івану Миколайовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка мобільного фітнес-застосунку засобами React Native та Node.js

керівник роботи (проєкту) Кривохата Анастасія Григоріївна, к.ф.-м.н.
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 04 » травня 2022 року № 500-с

2. Строк подання студентом роботи

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Аналіз вимог до інформаційної системи.
3. Реалізація інформаційної системи.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
Презентація

6. Консультанти розділів роботи

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

7. Дата видачі завдання 05.05.2022

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	06.05.2022	
2.	Збір вихідних даних.	07.05.2022	
3.	Обробка методичних та теоретичних	27.05.2022	
	джерел.		
4.	Розробка першого та другого розділу.	13.08.2022	
5.	Розробка третього розділу.	02.09.2022	
6.	Оформлення та нормоконтроль	01.12.2022	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	15.12.2022	

Студент _____ І.М. Неділько
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.Г. Кривохата
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка мобільного фітнес-застосунку засобами React Native та Node.js»: 57 с., 29 рис., 5 табл., 12 джерел, 1 додаток.

АСИНХРОННІСТЬ, МОБІЛЬНИЙ ДОДАТОК, СЕРВЕР, ФРЕЙМВОРК, BPMN, DB, ER, POSTGRESQL, IDEF, MVC, NODEJS, REACTJS, REACT NATIVE, UML.

Об’єкт дослідження – процес розробки мобільного застосунку.

Предмет дослідження – фреймворк React Native та платформа Node.js.

Мета роботи: розробка мобільного фітнес-застосунку.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проектування, конструювання та тестування програмного забезпечення.

Під час розробки системи проведений аналіз предметної області, обрана архітектура і платформа реалізації, розроблена функціональна модель системи у нотаціях IDEF0, IDEF1, IDEF3 та BPMN, спроектована архітектура систем за допомогою UML у вигляді наборів діаграм використання, послідовностей, класів, розгортання, реалізована інформаційна система а також мобільний застосунок та аутентифікація до неї з використанням мов програмування Javascript та Typescript, здійснене тестування системи та оформлена супровідна документація.

SUMMARY

Master’s Qualifying Paper «Development of the Mobile Fitness Application using React Native and Node.js»: 57 pages, 29 figures, 5 tables, 12 references, 1 supplement.

ASYNCHRONOUS, MOBILE APP, SERVER, FRAMEWORK, BPMN, DB, ER, POSTGRESQL, IDEF, MVC, NODEJS, REACTJS, REACT NATIVE, UML.

The object of the study is the activity of the mobile fitness application.

The subject of study is the framework React Native and the platform Node.js.

The aim of the study is development and implementation of a mobile fitness application.

The methods of research are methods of collection and analysis of software requirements, modeling, design, construction and testing of software.

During the development of the systems, the analysis of the subject area was carried out, the architecture and the platform for implementation were reconstructed, the functional model of the systems was broken up in the notations IDEF0, IDEF1, IDEF3 and BPMN, the architecture of the system was designed for additional UML in the display of the set , the information system has been implemented with the support of the JavaScript programming language, the system has not been tested and the supervised documentation has been issued.

ЗМІСТ

ВСТУП

У сучасному світі технології розвиваються з блискавичною швидкістю. Однією з головних причин цьому є автоматизація процесів для полегшення життя. Розробка мобільних застосунків не є виключенням. Але разом з автоматизацією зростає й складність програм з технічної точки зору, прикладом чого є реалізація програмної частини кваліфікаційної роботи.

Тему здорового способу життя досліджували ще з давніх давен, коли фізична форма була пріоритетом і головною перевагою людини для виживання. Про фізичний, духовний та ментальний розвиток людини написано чимало статей та проведено тисячі наукових досліджень.

Метою даної роботи є створення мобільного фітнес додатку. З метою реалізації поставленої задачі, було сформульовано такі основні завдання:

- обрати відповідний стек для реалізації програмної частини кваліфікаційної роботи;
- побудувати функціональну структуру для автоматизації основних бізнес-процесів та потоків даних;
- побудувати моделі інформаційної системи;
- спроектувати інформаційну систему;
- реалізувати та протестувати інформаційну систему.

Об'єктом дослідження даної роботи є процес розробки мобільного застосунку. Предметом дослідження є фреймворк React Native та платформа Node.js.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновку, переліку посилань.

Перший розділ роботи присвячений огляду засобів якими реалізовано програмне забезпечення та особливості роботи з ними. У другому розділі проектуються функціональні та інформаційні складові системи за допомогою засобів IDEF0, ER-діаграми, UML-діаграми. Третій розділ присвячений

реалізації і тестуванню програмного продукту. Також представлена документація до використання системи.

1 ОСОБЛИВОСТІ ТЕХНІЧНОГО СТЕКУ ЗАСТОСУНКУ

1.1 Особливості роботи з React Native

Для розробки мобільного застосунку було обрано фреймворк React Native.

React Native – кросплатформний фреймворк з відкритим вихідним кодом для розробки нативних настільних та мобільних додатків на Javascript та Typescript, створений компанією Facebook [1]. У контексті мобільної розробки, кросплатформеність досягається завдяки React Native bridge, який є посередником між Javascript потоком і апартаними потоками системи. Таким чином зберігається продуктивність створених застосунків на цьому фреймворку. Також можна вважати, що React Native працює на базі бібліотеки React.js [2].

Цей фреймворк використовує сумішок однієї з мов програмування Javascript або Typescript та спеціальної мови розмітки JSX [3].

Для того щоб розробляти на React Native, окрім знання Javascript або Typescript, треба розуміти наступні поняття та як їх використовувати. Першочергово треба знати що таке компонент та як з ним працювати. Компонент – самостійний елемент, який є візуальним представленням частини проекту, написаний на мови розмітки JSX, створений спеціально для повторного використання у проекті. По суті весь мобільний додаток кваліфікаційної роботи побудований на компонентах.

Наступним обов’язковим поняттям є стан компонента. Зазвичай використовується для рендеру представлення компонента з оновленими даними. Є невід’ємною частиною розробки на React Native. У кваліфікаційній роботі майже кожен компонент має свій стан.

Взаємодія компонентів відбувається за допомогою властивостей компонента (props) – вхідних даних, які отримуються від батьківського компонента. У мобільному застосунку потрібні для створення компонентів з динамічним представленням.

Не менш важливим поняттям вважається життєвий цикл компонента, який потрібний для декларування та контролю роботи з компонентами. Він має ряд етапів:

– ініціалізація – зазвичай викликається при підключенні компонента у представлення батьківського компоненту. При ініціалізації також створюється стан компонента;
– рендер – створення представлення компонента. Також цей етап викликається кожен раз при оновлені стану компонента;
– оновлення візуального представлення та посилання на елементи представлення;
– демонтування компоненту – викликається при знищенні у представленні батьківського компоненту.

У 16тій версій React.js прийшло нове поняття – React hooks. Основними перевагами реакт хуків є асинхронність та контроль операцій. Завдяки асинхронності будь-які зміни реакт компоненту не блокують основний потік. Контроль операцій дозволяє керувати станом і життєвим циклом реакт компоненту. Також присутня можливість створення власних хуків. Завдяки цьому можна створювати власні хуки на основі хуків реакта, що дозволяє інкапсулювати логіку та покращити підтримку проекту. У кваліфікаційній роботі ця можливість є невід'ємною частиною екранів зі складною логікою, або ж великою кількістю коду бізнес-логіки. Яскравим прикладом є екран створення та проведення тренування, який відповідає за динамічний список вправ з індивідуальними тренуваннями, сам процес проведення тренування.

1.2 Особливості роботи з Node.js

Для розробки серверу як частини програмного забезпечення кваліфікаційної роботи було обрану платформу Node.js.

Node.js – платформа з відкритим кодом для виконання

високопродуктивних мережевих застосунків, написаних мовою JavaScript [4]. Node.js надає можливість виконувати JavaScript код на сервері та відправляти користувачеві результат їх виконання. Node.js має наступні властивості:

– асинхронна однопотокова модель виконання запитів;
– неблокуюче введення/виведення;
– система модулів CommonJS;
– рушій JavaScript Google V8.

Для керування модулями використовується пакетний менеджер npm (node package manager).

Нижче описані основні особливості роботи з Node.js.

Node.js є можливість циклічної залежності. Але через те що підключення файлів «під капотом» – це просто генерація об’єктів з власною областю видимості і властивостями, Node.js дозволить запустити програму, проте написаний код файлу, який повільніше конвертувався в об’єкт стане недоступним, тому що цей об’єкт не матиме властивостей.

Наявність потоків (Streams). По суті основний функціонал Node.js API побудований на пакетах Eventemitter та Streams. Потоки дозволяють зчитувати великий об’єм даних, не блокуючи процес у фоновому режимі. Використання потоків у кваліфікаційній роботі відбувається неявно. Наприклад, читання .env файлу відбувається за допомогою модулів fs та streams.

Основний код на Node.js пишеться асинхронно за допомогою Promise’ів. Якщо забути обробити асинхронність програми або ж якогось конкретного promise’у, то при виникненні помилки може зупинитися програма.

У кваліфікаційній роботі сервер для додатку реалізований на фреймворку Nest.js, основними перевагами якого є наявність інтерфейсу консолі, строгі правила імпорту модулів на основі предметно-орієнтованого проектування та наявність детальної та зрозумілої документації [5, 6].

Інтерфейс консолі для даного фреймворку допомагає достатньо швидко керувати розробкою серверу. Для кваліфікаційної роботи було використано команди nest new [project name], яка ініціалізує репозиторій серверу на основі

підготовленого шаблону та nest g mo [module name], яка генерує директорію з модулем який складається класів контролеру, сутності, data transfer object, сервісу та тесту й імпортує цей модуль у головний.

1.3 Характеристика обраної системи управління базами даних

Оскільки програмне забезпечення має бути гнучким, простим та продуктивним, треба було обрати базу даних, яка найкраще підходить під такі вимоги. Нині популярністю користуються наступні СУБД:

– Mysql;
– Postgresql;
– Mongodb.

Було обрано СУБД Postgresql. Ця СУБД вважається гарним рішенням для малих і середніх застосувань [7]. Серцеві коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX-системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому. Можливості сервера Postgresql наступні:

– простота у встановленні та використанні;
– підтримується необмежена кількість користувачів, що одночасно працюють із БД;
– кількість рядків у таблицях може досягати 50 млн;
– висока швидкість виконання команд;
– наявність простої і ефективної системи безпеки.

Основними перевагами є наявність коректної та швидкої роботи зі зв’язками (чого немає у Mongodb), та різноманітна наявність типів полів, індексів (чого бракує у Mysql). Оскільки система має непросту архітектуру та відповідає всім функціональним вимогам, то база даних повинна відповідати усім кон’юнктивним нормальним формам.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Збір та аналіз вимог

Фітнес програми – це можливість залишатися залученим до процесу тренувань, інтерактивно відстежувати зміни та контролювати результати. Головна проблема людей, які починають займатися спортом – брак мотивації та сили волі. Програми для фітнесу виконують функцію тренера та мотиватора, які ще й контролюють зміни.

З метою аналізу предметної області було досліджено програмні продукти у сфері фітнесу. У магазині додатків для мобільної платформи Android “Play Market” знаходилися застосунки «Home workout» та «Fitness & Bodybuilding» [8, 9].

Спільні функції, які реалізовано в продуктах, такі:

– можливість подивитися деталі вправи;
– наявність історії проведених тренувань;
– можливість проведення тренування;
– наявність готових шаблонів тренувань.

Серед основних переваг даних продуктів можна виділити інтерактивний інтерфейс та наявність аналітики для слідкування за прогресом. До недоліків «Home workout» можна віднести відсутність можливості самостійно створювати програму тренувань і надлишок реклами. Основними недоліками «Fitness & Bodybuilding» є незрозумілий інтерфейс та обмежений функціонал аналітики.

Програмне забезпечення кваліфікаційної роботи реалізовано на основі вище проведеного аналізу. Головною перевагою реалізованого мобільного застосунку перед проаналізованими мобільними додатками є можливість створення, налаштування та проведення тренування. Тобто користувач має змогу створювати індивідуальну програму для кожного тренування.

Список функцій мобільного додатку складається з:

– можливості подивитися деталі вправи;
– наявності історії проведених тренувань;
– можливості створення, налаштування та проведення тренування;
– наявності аналітики за певними критеріями.

У порівнянні з проаналізованими мобільними продуктами основним некритичним недоліком мобільного застосунку кваліфікаційної роботи є відсутність можливості створення та використання шаблонів тренувань.

2.2 Концептуальне проектування інформаційної моделі

Концептуальна модель сховища даних є описом головних (основних) сутностей і відношень між ними. Така модель є відображенням предметних областей, у межах яких планується побудова сховища даних.

При проектуванні концептуальної моделі структурують дані і виявляють взаємозв'язок між ними, без розгляду особливостей реалізації і питань ефективності обробки.

Для розробки концептуальної моделі системи виділено такі інформаційні об'єкти:

– Users;
– Admins;
– Exercises;
– Categories;
– User trainings.

У схемах в квадратах відображені відношення (див. рис. 2.1):

– Categories <відноситься> до Exercises categories, зв'язок один-до-багатьох. Такий тип зв’язку обумовлено тим фактом, що декілька вправ можуть мати одну й ту саму категорію;
– Users <відноситься> до User trainings, зв'язок один-до-багатьох. Такий тип зв’язку обумовлено тим фактом лише один користувач може

провести багато тренувань;

– Exercises <відноситься> до User trainings, зв'язок один-до-багатьох. Такий тип зв’язку обумовлено тим, що одна вправа може бути виконана у багатьох тренуваннях.

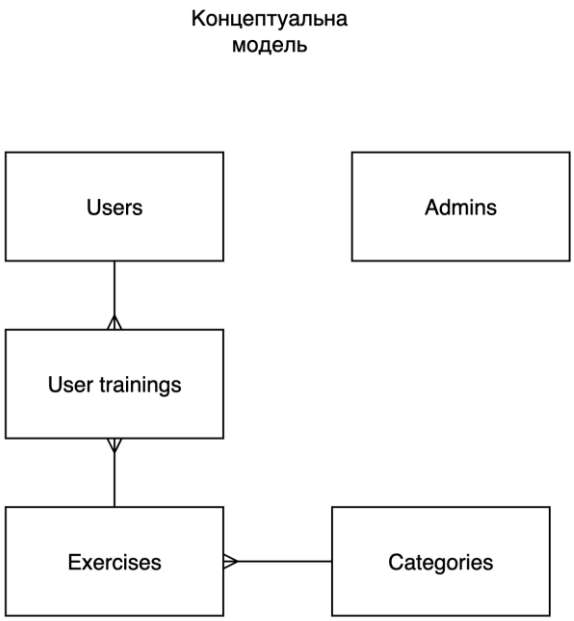

Рисунок 2.1 – Концептуальна модель сховища даних «Фітнес-застосунок»

З метою опису функціональних вимог до системи, для опису предметної області, кращого розуміння функціонування системи наведено діаграму прецедентів (див. рис. 2.2).

Виділено сутність “Користувач”, який взаємодіє із системою за допомогою таких варіантів використання: аутентифікація, авторизація, огляд проведених тренувань, створення та проведення тренування, огляд профілю, огляд аналітики проведених тренувань.

Також виділено сутність “Адмін” який також взаємодіє із системою, проте за допомогою наступних варіантів використання: огляд існуючих категорій, створення категорій, огляд існуючих вправ, створення вправ, аутентифікація, авторизація.

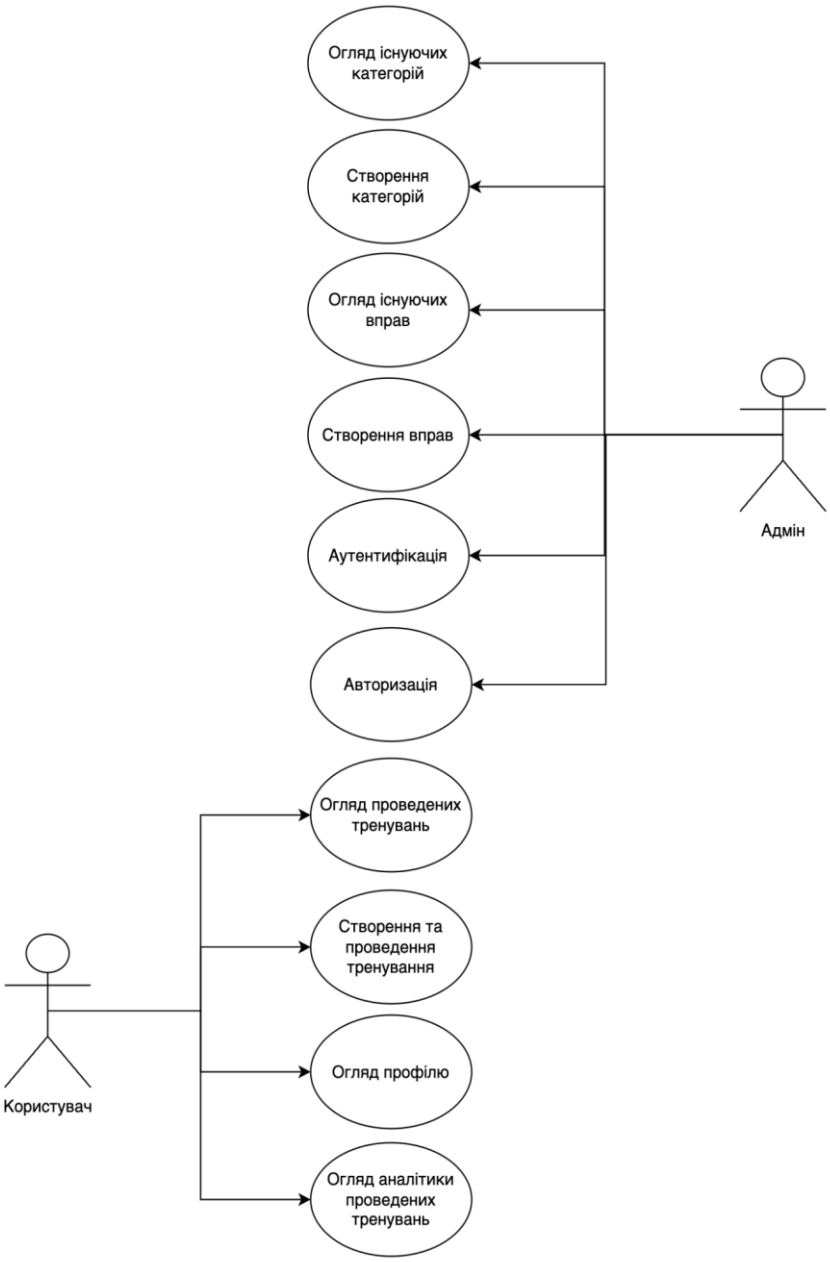

Рис. 2.2 – Діаграма прецедентів

На рисунку 2.3 представлена діаграма послідовності дій бізнес-процесу «Проведення тренування». Спочатку користувач відкриває сторінку проведення тренування, мобільний додаток здійснює запит на сервер список існуючих вправ. Якщо сервер повертає код статусу 500 – це означає, що на стороні сервера існують деякі проблеми, через які він не може відправити як відповідь список вправ. Як тільки додаток отримує від сервера список вправ – користувач отримує змогу створювати налаштування для тренування. Він обирає тип вправи,

кількість виконаних разів за підхід і т.д. Як тільки користувач налаштовує хоча б одне тренування – на екрані з’являється кнопка для старту, проте це не заважає додати та налаштувати ще декілька вправ. Для старту тренування треба натиснути на відповідну кнопку після чого у додатку відкриється екран тренування. Після проведення тренування, мобільний додаток відправляє його на сервер для збереження. Так само як було і у попередньому випадку, якщо сервер повертає код статусу 500 значить має якісь проблеми і не може зберегти проведене тренування користувача зараз. В іншому випадку сервер повертає код статусу 200, що означає що тренування успішно зберіглося, мобільний додаток переводить користувача на домашній екран і показує повідомлення успішного збереження.

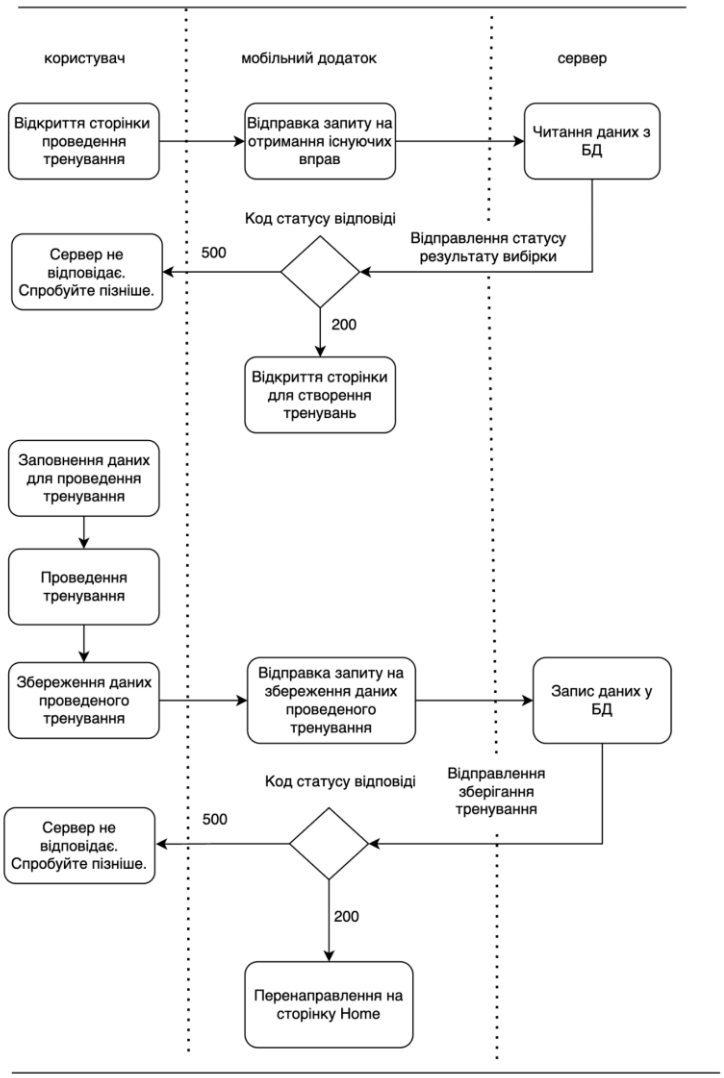

Рисунок 2.3 – Діаграма послідовності бізнес-процесу «Проведення тренування»

2.3 Етап логічного проектування

Для моделювання майбутніх класів, їх атрибутів, методів, а також зв’язків використовуються діаграма класів. Для ІС діаграма класів виглядає наступним чином (рис. 2.4).

Рисунок 2.4 – Діаграма класів компоненту аналізу даних проведених тренувань

Діаграма класів – це основа будь-якої документації програмного проекту. Модель класів лежить в основі об’єктно – орієнтованого програмування. Вона описує як загальний стан всієї програмної системи, так і її поведінку.

На даній діаграмі зображена схема взаємодії класів між собою для аналізу тренувань користувача.

На ній присутні класи TrainingsAnalyticsController та TrainingsAnalyticsService.

Клас TrainingsAnalyticsController. Складається з методу findAll(). Цей метод викликає метод класу TrainingsAnalyticsService findAll().

Клас TrainingsAnalyticsService. Містить методи findAll(), getAnalyticsByExercise(), getAnalyticsByCategory(). У методі findAll() класу TrainingsAnalyticsService, в залежності від вхідних даних, визначається за яким саме параметром потрібно провести аналіз й викликається метод

getAnalyticsByCategory() або getAnalyticsByExercise() відповідно. Метод getAnalyticsByExercise() виконує аналіз для конкретної вправи, для цього він використовує сутності UserTrainingEntity і Exercise. Метод getAnalyticsByCategory() використовує для аналізу наступні сутності UserTrainingEntity, Category та Exercise.

Інформаційна модель даних «Фітнес-застосунок» описує сутності: users, admins, categories, exercises, user_trainings (див. рис. 2.5), а також їх взаємовідносини між собою.

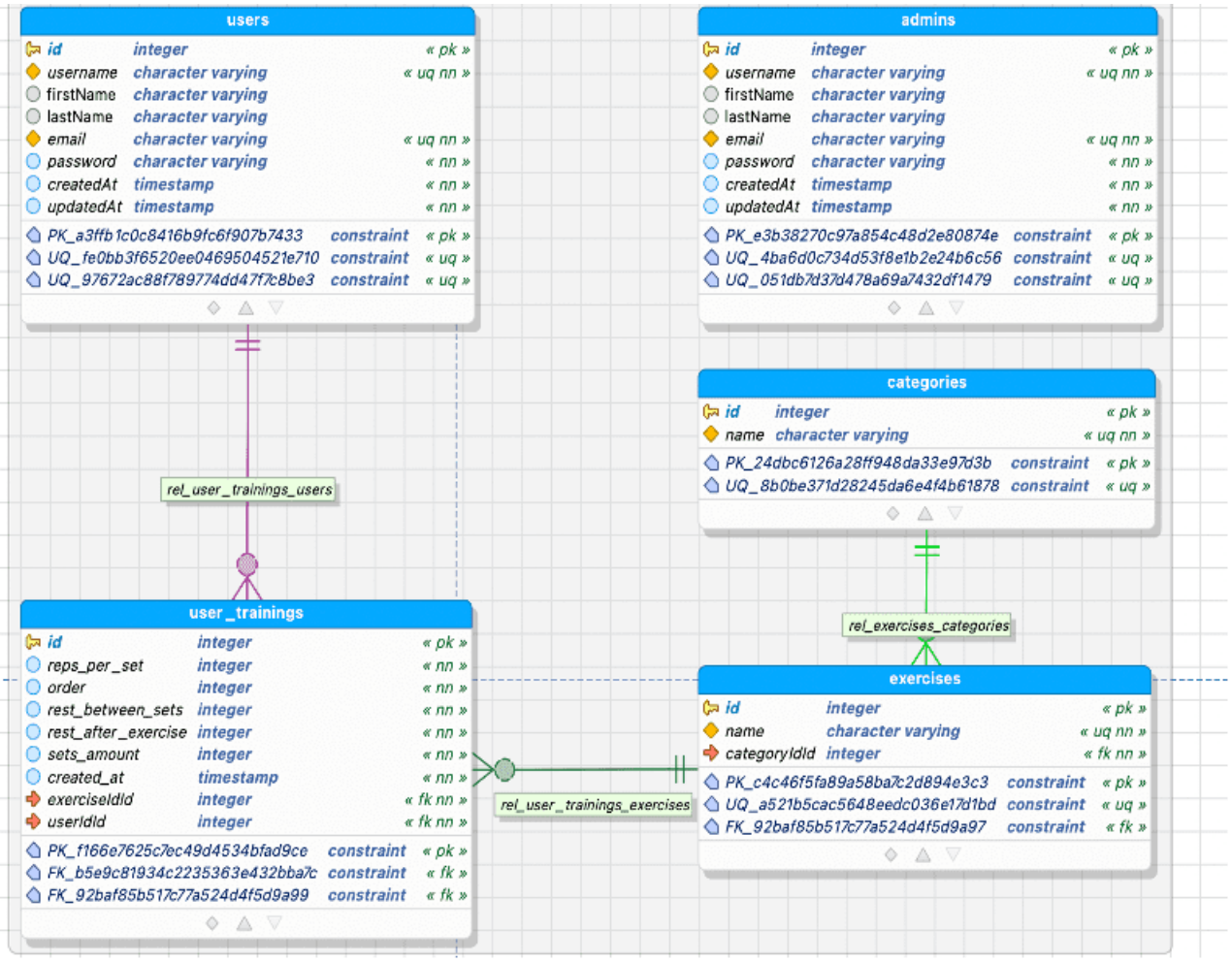

Рисунок 2.5 – Фізична модель підсистеми «Фітнес-застосунок»

Розглянемо детальніше кожну таблицю фізичної моделі підсистеми. У таблиці «Users» (див. табл. 2.1) зберігається інформація про користувачів програмного забезпечення.

Таблиця 2.1 – Користувачі

Назва поля	Тип даних	Властивість	Опис
id	Integer	Primary key (PK)	Ідентифікатор користувача
username	String		Унікальний нікнейм користувача
email	String		Пошта користувача
firstName	String		Ім’я користувача
lastName	String		Прізвище користувача
password	String		Пароль користувача

У таблиці «Admins» (див. табл. 2.2) зберігається інформація про адміністраторів програмного забезпечення.

Таблиця 2.2 – Адміністратори

Назва поля	Тип даних	Властивість	Опис
id	Integer	Primary key (PK)	Ідентифікатор адміністратора
username	String		Унікальний нікнейм адміністратора
email	String		Пошта адміністратора
firstName	String		Ім’я адміністратора

Продовження таблиці 2.2

lastName	String		Прізвище адміністратора
password	String		Пароль адміністратора

У таблиці «Categories» (див. табл. 2.3) зберігається інформація про категорії програмного забезпечення.

Таблиця 2.3 – Категорії

Назва поля	Тип даних	Властивість	Опис
id	Integer	Primary key (PK)	Ідентифікатор категорії
name	String		назва категорії

У таблиці «Exercises» (див. табл. 2.4) зберігається інформація про вправи програмного забезпечення.

Таблиця 2.4 – Вправи

Назва поля	Тип даних	Властивість	Опис
id	Integer	Primary key (PK)	Ідентифікатор вправи
name	String		назва вправи
category_id	Integer	Foreign key	Ідентифікатор категорії

У таблиці «User_trainings» (див. табл. 2.5) зберігається інформація про проведенні тренування користувачів програмного забезпечення.

Таблиця 2.5 – Тренування користувачів

Назва поля	Тип даних	Властивість	Опис
id	Integer	Primary key (PK)	Ідентифікатор тренування
reps_per_set	Integer		кількість повторів на один захід
order	Integer		Порядок виконання вправи
rest_between_sets	Integer		Час відпочинку між заходами вправи у секундах
rest_after_exercise	Integer		Час відпочинку після вправи у секундах
sets_amount	Integer		кількість заходів однієї вправи
exerciseIdId	Integer		Ідентифікатор вправи
categoryIdId	Integer		Ідентифікатор користувача

2.4 Етап фізичного розгортання

З метою відображення фізичного взаємозв'язку між програмними і апаратними компонентами системи та демонстрації маршрутів переміщення

об'єктів і компонентів у розподіленій системі наведено діаграму розгортання. Кожен вузол на діаграмі розгортання є певним типом обчислювального пристрою – у більшості випадків це частина апаратури. Діаграму розгортання наведено на рисунку 2.6.

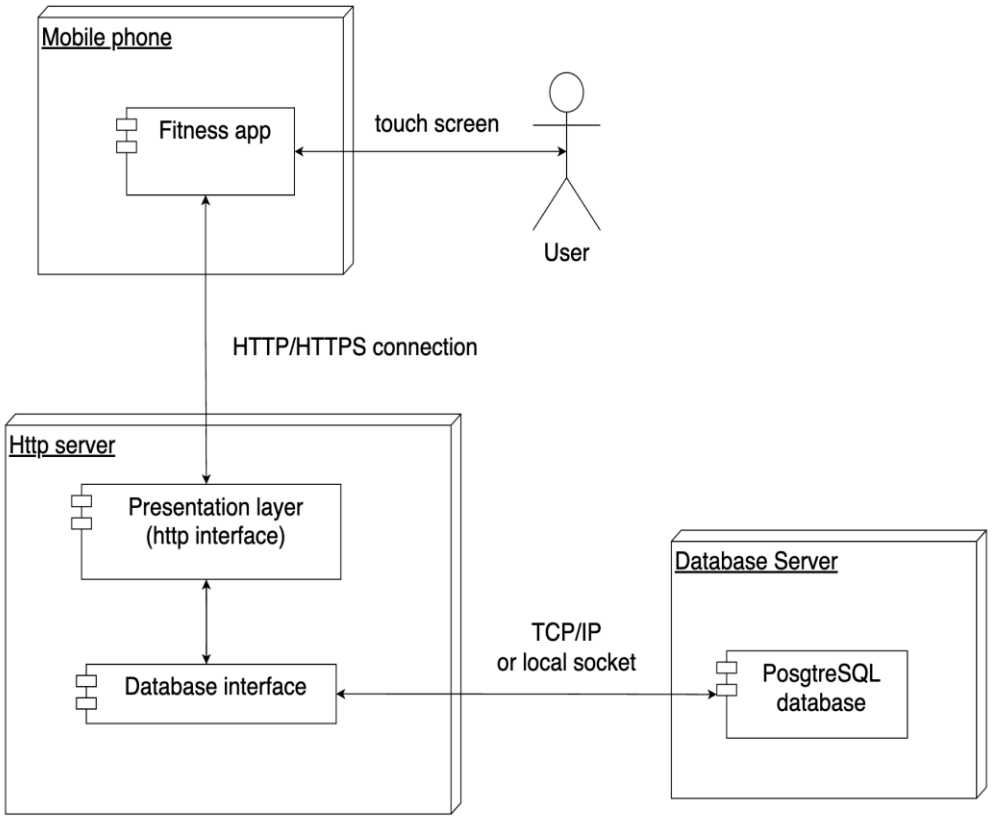

Рисунок 2.6 – Діаграма розгортання

На діаграмі компонентів відображено організацію компонентів і залежності між ними. Компонент – це фізичний елемент реалізації (компоненти вихідного коду, бінарного коду, що виконують компоненти) з чітко визначеним інтерфейсом, призначений для використання в якості змінної частини системи. Дану діаграму наведено на рисунку 2.7.

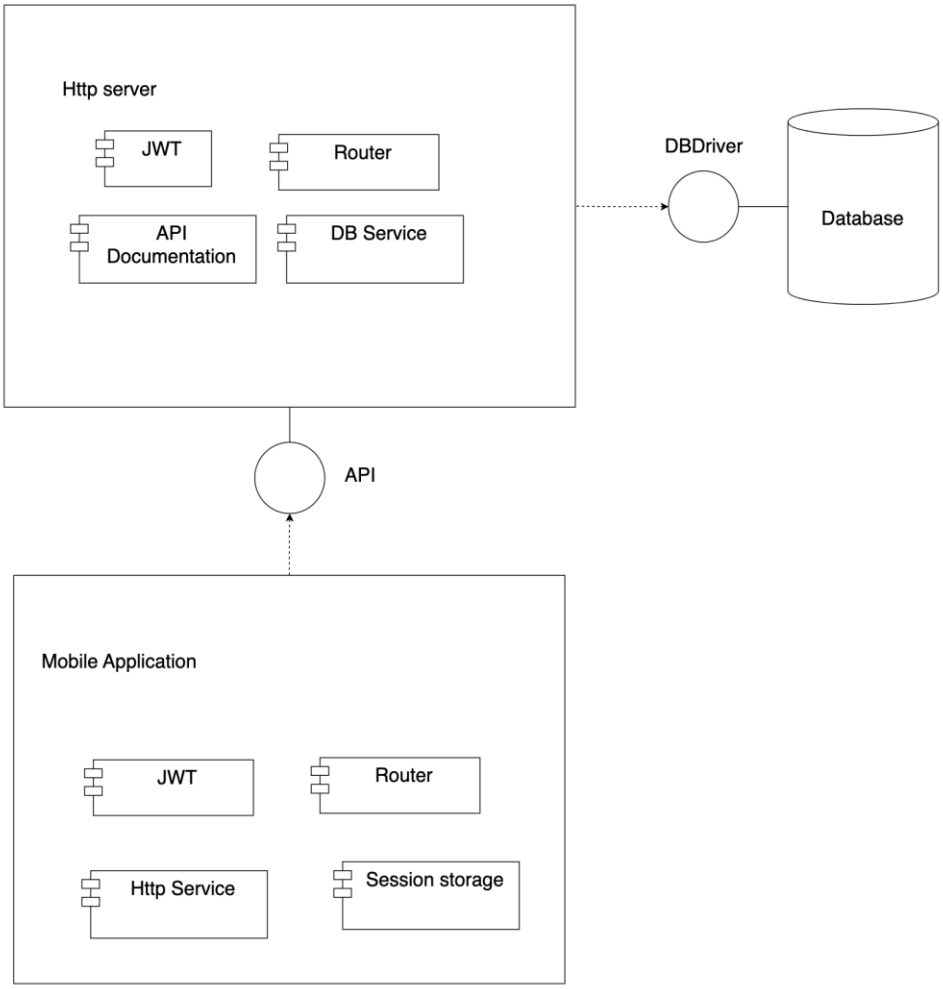

Рисунок 2.7 – Діаграма компонентів

Компонент Backend – є головним, потрібний для обробки http запитів UI компонента. Наприклад, коли користувач відкриває екран проведених тренувань UI компонент відправляє http запит для отримання списку цих тренувань користувача. При отриманні відповідних даних з http запиту, Backend відправляє запиту по tcp протоколу до компоненту DB, де зберігаються усі тренування користувачів застосунку, й відправляє у відповідь отриманий результат з компонента DB компоненту UI.

Компонент UI – відповідає за візуальне уявлення зручного інтерфейсу для взаємодії з інформаційною системою.

Компонент DB – містить в собі всю інформацію програмного додатку, а саме інформацію про користувачів, їх проведенні тренування, категорії, вправи та адмінів додатку.

3 ПРОГРАМУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аутентифікація та захист даних

На сьогоднішній день невід'ємними частинами будь-якого програмного забезпечення є авторизація та аутентифікація.

Аутентифікація – процес перевірки прав доступу до певної інформації чи дій за допомогою ідентифікатора.

В даній кваліфікаційній роботі реалізовано перший вид аутентифікації – однофакторна аутентифікація. Вона є традиційною для більшості програмного забезпечення. Зазвичай це пароль користувача, її ще називають слабкою, бо існують різні способи злому паролю. Найбільш популярними є фішинг та ручний перебір паролю. Так як проект не міститиме критично важливої інформації на кшталт даних кредитної картки, немає необхідності ускладнення процесу аутентифікації.

Авторизація – керування рівнями та засобами доступу до певного захищеного ресурсу, як у фізичному розумінні (доступ до кімнати готелю за карткою), так і в галузі цифрових технологій (наприклад, автоматизована система контролю доступу) та ресурсів системи залежно від ідентифікатора і пароля користувача або надання певних повноважень (особі, програмі) на виконання деяких дій у системі обробки даних.

Найбільш популярним способом авторизація є JWT(json web token) [10].

У кваліфікаційній роботі застосування jwt присутнє у наступному алгоритмі авторизації та аутентифікації:

1) при успішній авторизації користувач отримує унікальну пару jwt токенів – token, refresh token. Кожен токен має певний час життя і зберігає в собі зашифровані дані;
2) для комунікації з сервером у заголовки буде додано основний токен для авторизації користувача.

В роботі, сервіс роботи з токенами реалізований у класі JwtService який зображено на рисунку 3.1.

```
export class JwtService {
  private jwtService: Jwt;
  private SECRET = process.env.JWT_SECRET ?? "asdasd";

  constructor() {
    this.jwtService = new Jwt();
  }

  sign(payload: any, expiresInSeconds: number): string {
    return this.jwtService.sign(payload, options: {
      expiresIn: expiresInSeconds,
      secret: this.SECRET,
    });
  }

  verify(token: string): boolean {
    try {
      this.jwtService.verify(token, options: { secret: this.SECRET });
      return true;
    } catch (e) {
      console.log("error verified token", e);
      return false;
    }
  }

  decode(token):
    | null
    | string
    | {
        [key: string]: any;
      } {
    return this.jwtService.decode(token, options: { json: true });
  }
}
```

Рисунок 3.1 – Клас для роботи з токенами JwtService

На стороні серверу налаштована перевірка за допомогою зчитування та аналізу jwt токена взятого з http запиту.

```
@Injectable()
export class RoleGuardGuard implements CanActivate {
  constructor(
      @Inject(JwtService) private readonly jwtService,
      private reflector: Reflector
  ) {}

  canActivate(
    context: ExecutionContext,
  ): boolean | Promise<boolean> | Observable<boolean> {
    const request = context.switchToHttp().getRequest();
    const {authorization: token} = request.headers
    if (!token) {
      throw new UnauthorizedException( objectOrError: "Unauthorized")
    }

    const ROLE = this.reflector.get<string[]>( metadataKey: 'role', context.getHandler());
    const payload = this.jwtService.decode(token)
    return payload && payload.role && payload.role === ROLE
  }
}
```

Рисунок 3.2 – Перевірка jwt токена http запиту

На рисунку 3.2 спочатку зчитується jwt із заголовку запиту під назвою authorization. Якщо токен не доданий у цей заголовок запиту, тоді сервер відправляє код статусу http «401 Unauthorized», що означає відсутність доступу до цього запиту. Результатом є перевірка чи створений цей токен сервером проекту чи ні, а також чи ідентичні роль з корисного навантаження та роль для конкретної відносної адреси серверу.

```
@UseGuards(RoleGuardGuard)
@Role( role: 'admin')
@Post()
create(@Body() createAdminDto: CreateAdminDto) {
  return this.adminService.create(createAdminDto);
}
```

Рисунок 3.3 – Використання функції авторизації

На рисунку 3.3 зображено захист відносної адреси серверу для створення адміністраторів проекту. Реалізовано це за допомогою декораторів UseGuards, та Role. Спочатку оголошено використання класу захисту по ролям. Далі у якості аргумента декоратора Role() передається метадані, у нашому випадку це роль адміністратора. Тобто лише адміністратор має право створювати інших адміністраторів.

3.2 Огляд основних частин коду

Однією з важливих частин функціоналу є реалізація створення та проведення тренування. Враховуючи те, що тренування може мати довільну кількість вправ з абсолютно різними налаштуваннями і треба написати код який можна буде у майбутньому розширяти та підтримувати, можна зробити висновок що задача досить нетривіальна. Одним з усталених підходів організації коду візуальної частини програмування полягає у тому щоб зберігати у місці головного компонента основну логіку яка використовується на сторінці й через ін'єкцію залежності передавати його у дочірні компоненти делегуючи їм

обов’язки але зберігаючи контроль.

```
const useCreateTrainingHook = ({ navigation }) => {
  const [exercises, setExercises] = useState( initialState: []);
  const [trainingStarted, setTrainingStarted] = useState( initialState: false);
  const [renderFlag, setRenderFlag] = useState( initialState: 1);
  const [availableExercises, setAvailableExercises] = useState( initialState: []);
  const [training, setTraining] = useState(new Training());

  useEffect( effect: () => {
    if (!Array.isArray(exercises) || !exercises.length) return;
    training.setExercises(exercises);
    setTraining(training);
  }, deps: [exercises]);

  const addExercise = () => {
    const newExercise = new Exercise();
    setExercises( value: (prevExercises :[] ) => [...prevExercises, newExercise]);
  };

  const onUpdateExercise = (idx, exercise) => {
    exercises[idx] = new Exercise(exercise);
    setExercises( value: () => [...exercises]);
    setRenderFlag( value: (prevState :number ) => prevState + 1);
  };

  const onDeleteExercise = (idx) => {
    const updatedExercises = exercises.filter((el, ind :number ) => ind !== idx);
    training.removeExerciseById(idx)
    setExercises( value: () => [...updatedExercises]);
    setTraining(training)
  };

  const startTraining = () => {
    training.setCurrentExerciseById(0);
    setTraining(training);
    setTrainingStarted( value: true);
  };

  const clearStateAndOpenHomePage = () => {
    navigation.navigate("Home");
    setExercises( value: []);
    setTraining(new Training());
    setTrainingStarted( value: false);
  };

  const onTrainingEnd = () => {
    startLoading();
    const preparedTraining = prepareTrainingForSaving(training);
    trainingsApi.createTraining(preparedTraining).then(() => clearStateAndOpenHomePage())
      .catch((err) => console.log("Error save training", err)).finally( onFinally: () => stopLoading());
  };

  const onCancelTraining = () => { clearStateAndOpenHomePage() };
  useEffect( effect: () => {
    startLoading();
    exercisesApi
      .getExercises().then((response) => {
        if (Array.isArray(response) && response.length)
          return setAvailableExercises(response);
      }).catch((err) => console.log("err", err)).finally( onFinally: () => {
        stopLoading();
      });
  }, deps: []);

  const isTrainingReadyToStart = training ? !!training.getTotalExercises() : false;
  return {
    exercises, renderFlag, training, trainingStarted, isTrainingReadyToStart, availableExercises,
    startTraining, addExercise, onDeleteExercise, onUpdateExercise, onTrainingEnd, onCancelTraining,
  };
};
```

Рисунок 3.4 – Хук сторінки створення та проведення тренування

На рисунку 3.4 зображена уся логіка для створення та проведення

тренування. Хук зберігає дані про тренування, список вправ, інкапсулює зміну стану повертаючи лише ті методи які дозволено використовувати. Дозволеними методами є редагування вправи, видалення вправи зі списку, додавання вправи у список для тренування а також старт тренування.

```
const CreateTrainingPage = ({ navigation }) => {
  const {
    exercises,
    renderFlag,
    trainingStarted,
    availableExercises,
    isTrainingReadyToStart,
    training,
    addExercise,
    onUpdateExercise,
    onDeleteExercise,
    startTraining,
    onTrainingEnd,
    onCancelTraining,
  } = useCreateTrainingHook( {navigation}: { navigation });

  return (
    <View style={styles.container}>
      {trainingStarted ? (
        <TrainingPage
          initialTraining={training}
          onCancelTraining={onCancelTraining}
          onEndTraining={onTrainingEnd}
        />
      ) : (
        <View style={styles.container}>
          <TrainingExercisesList
            renderFlag={renderFlag}
            onDeleteItem={onDeleteExercise}
            availableExercises={availableExercises}
            exercises={exercises}
            onItemUpdate={onUpdateExercise}
          />
          <TouchableOpacity style={styles.addExerciseBtn} onPress={addExercise}>
            <Text style={styles.addExerciseTitle}>Add exercise</Text>
          </TouchableOpacity>

          {isTrainingReadyToStart && (
            <TouchableOpacity
              onPress={startTraining}
              style={styles.startTrainingBtn}
            >
              <Text style={styles.whiteCenteredText}>Start training</Text>
            </TouchableOpacity>
          )}
        </View>
      )}
    </View>
  );
};
```

Рисунок 3.5 – Сторінка створення та проведення тренування

На рисунку 3.5 зображено приклад використання вищевказаного хука на сторінці створення та проведення тренування. Як ми можемо побачити, нам

доступні лише деякі методі та змінні які були створені в хуку. Далі, в залежності від значення змінної trainingStarted, сторінка показує екран проведення тренування, або ж екран його створення. Розглянемо список огляду списку вправ для тренування з індивідуальними налаштуваннями.

```
const TrainingExercisesList = ({
                                    renderFlag,
                                    exercises,
                                    availableExercises,
                                    onItemUpdate,
                                    onDeleteItem,
                                    onStartTraining
                                }) => {
    if (!exercises) {
        return (null)
    }

    const renderItem = (item) => {
        return (
            <TrainingExerciseItem
                item={item.item}
                ind={item.index}
                availableExercises={availableExercises}
                exercisesAmount={exercises.length}
                onDelete={onDeleteItem}
                onUpdate={onItemUpdate}
                onStartTraining={onStartTraining}
            />
        )
    }

    return (
        <View style={styles.container}>
            {renderFlag && <FlatList
                data={exercises}
                renderItem={renderItem}
                keyExtractor={item => item}
            /> }
        </View>
    )
}
```

Рисунок 3.6 – Список вправ для тренування

На рисунку 3.6 зображено список вправ для тренування. Можна побачити реалізацію ін'єкції залежностей. Компонент приймає як вхідні дані список вправ для тренування, список існуючих вправ у програмі а також методи маніпуляцій зі станом тренування та сторінки. Варто помітити, що відсутнє жодне редагування будь-якого зі списків, що підтверджує повний контроль головного хука useCreateTrainingHook() над роботою та станом сторінки.

3.3 Огляд інтерфейсу мобільного додатку

Оскільки екосистема серверу була підготовлена до швидкого розгортання заздалегідь – досить лише зібрати і запустити docker-compose.

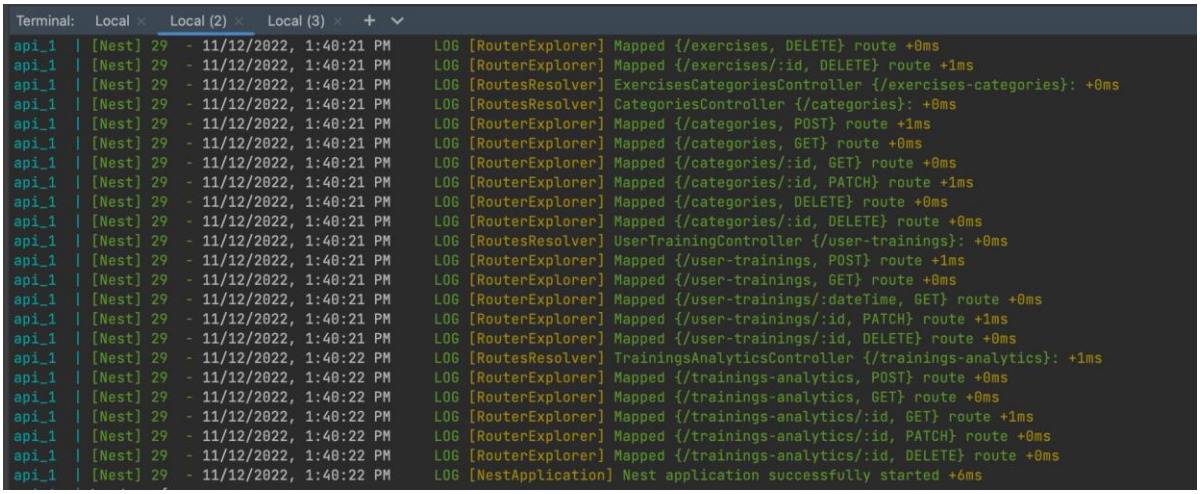

Рисунок 3.7 – Запуск серверу з базою даних

Заповнюємо форму та проходимо аутентифікацію.

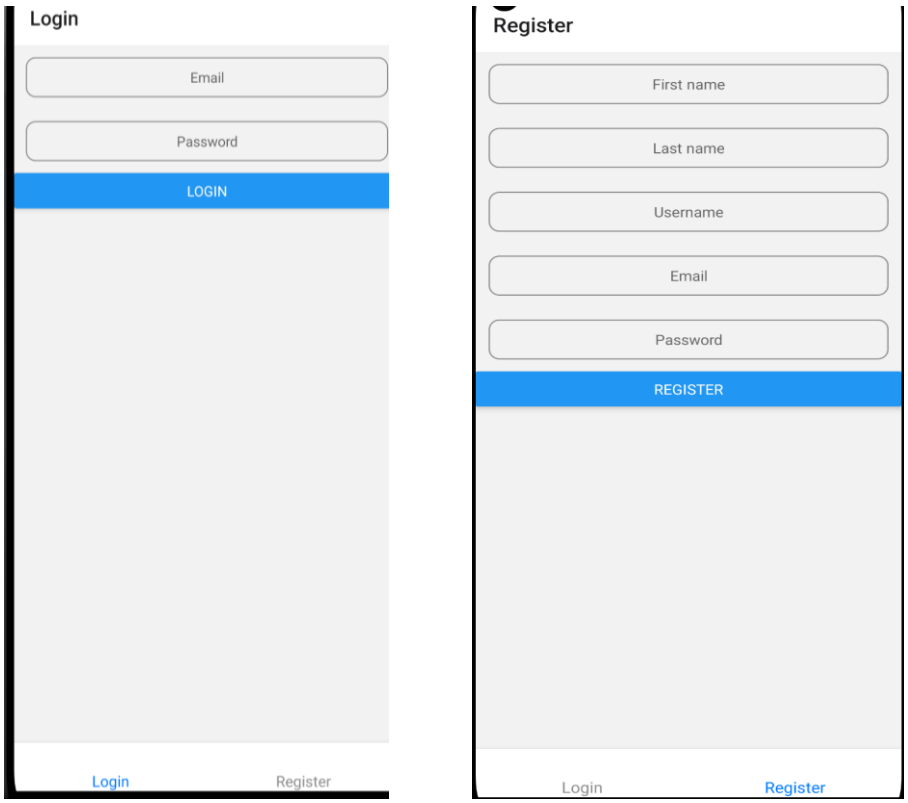

Рисунок 3.8 – Екрани логіну та реєстрації

Після успішного проходження аутентифікації користувач потрапляє на сторінку Home.

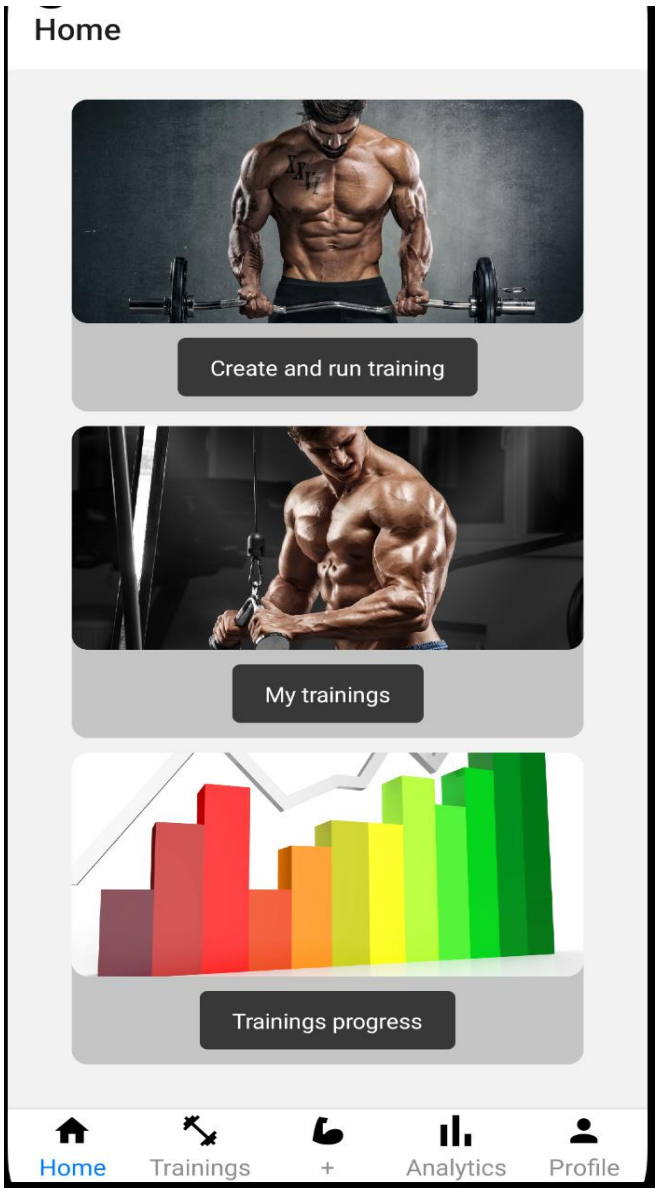

Рисунок 3.9 – Сторінка Home

Як бачимо, застосунок має інтуїтивно-зрозумілий інтерфейс. Перейдемо на сторінку створення та проведення тренування.

Рисунок 3.10 – Сторінка створення та проведення тренування

Щоб провести тренування, треба додати вправи та налаштувати. Для цього натиснемо кнопку Add exercise.

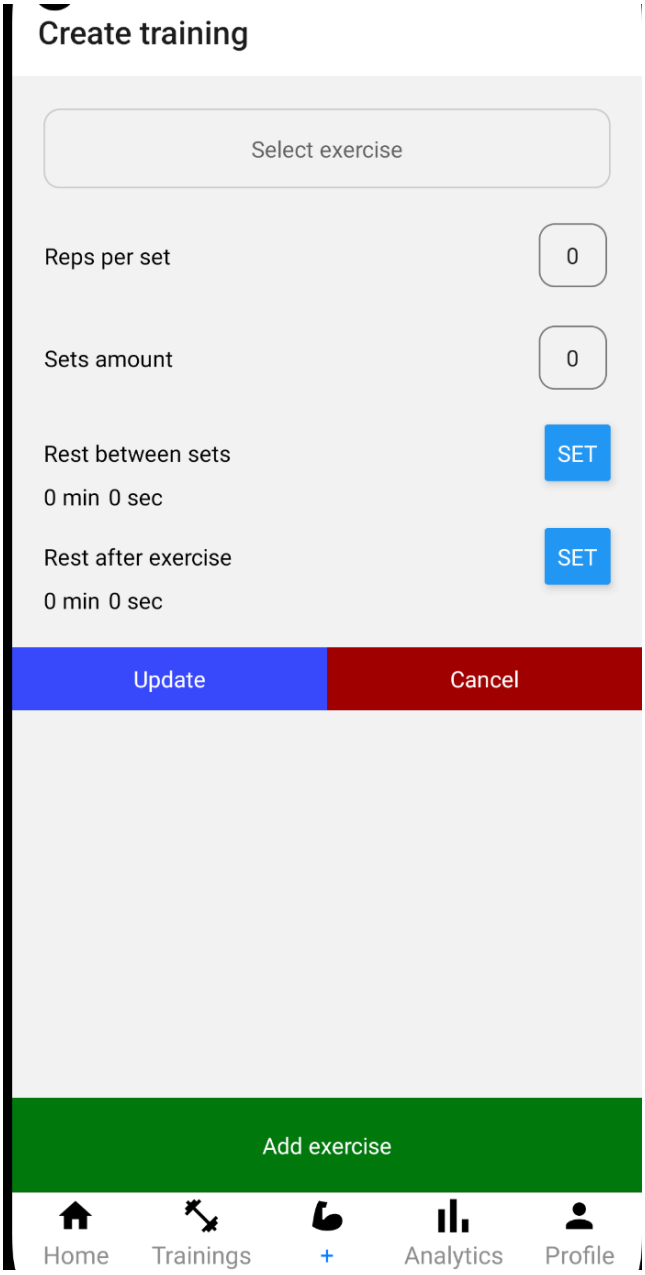

Рисунок 3.11 – Форма додавання та налаштування тренування

На рисунку 3.11 зображено форму для створення та налаштування вправи тренування, у ній можна вказати кількість підходів, повторів за один підхід, час відпочинку між підходами та після вправи.

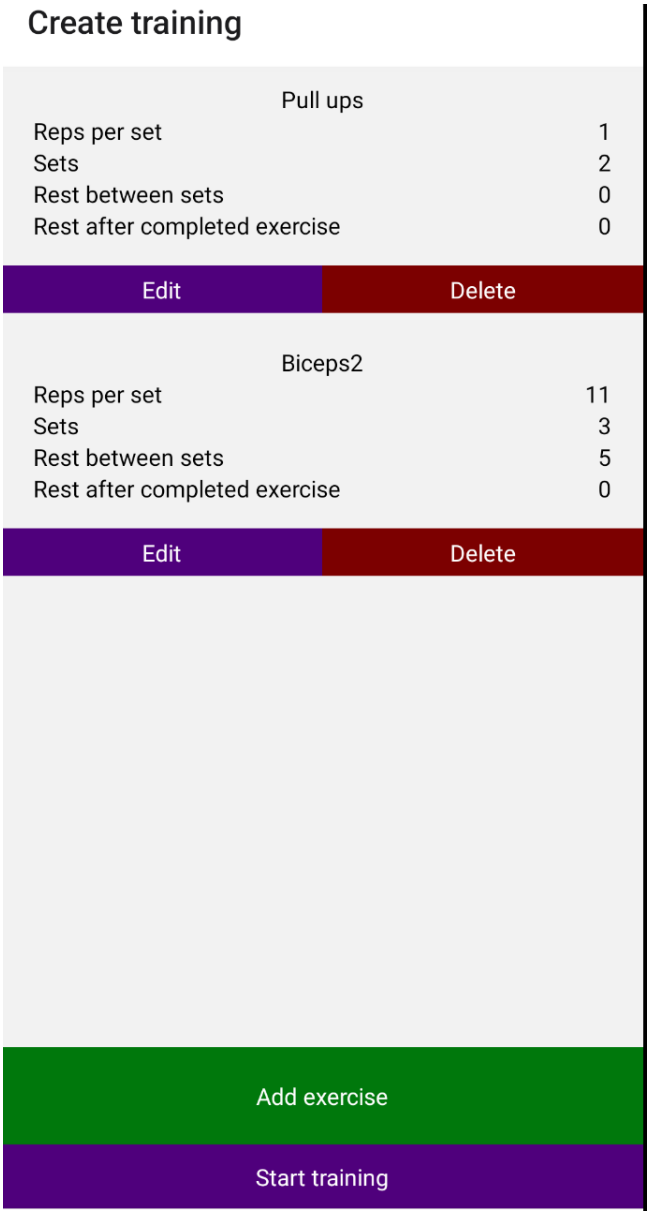

Рисунок 3.12 – Форма додавання та налаштування тренування

На рисунку 3.12 зображено список доданих та налаштованих вправ для тренування. Також можна помітити, що на екрані з’явилася кнопка старту тренування. За бажанням можна видалити або відредагувати будь-яку вправу зі списку. Почнемо тренування.

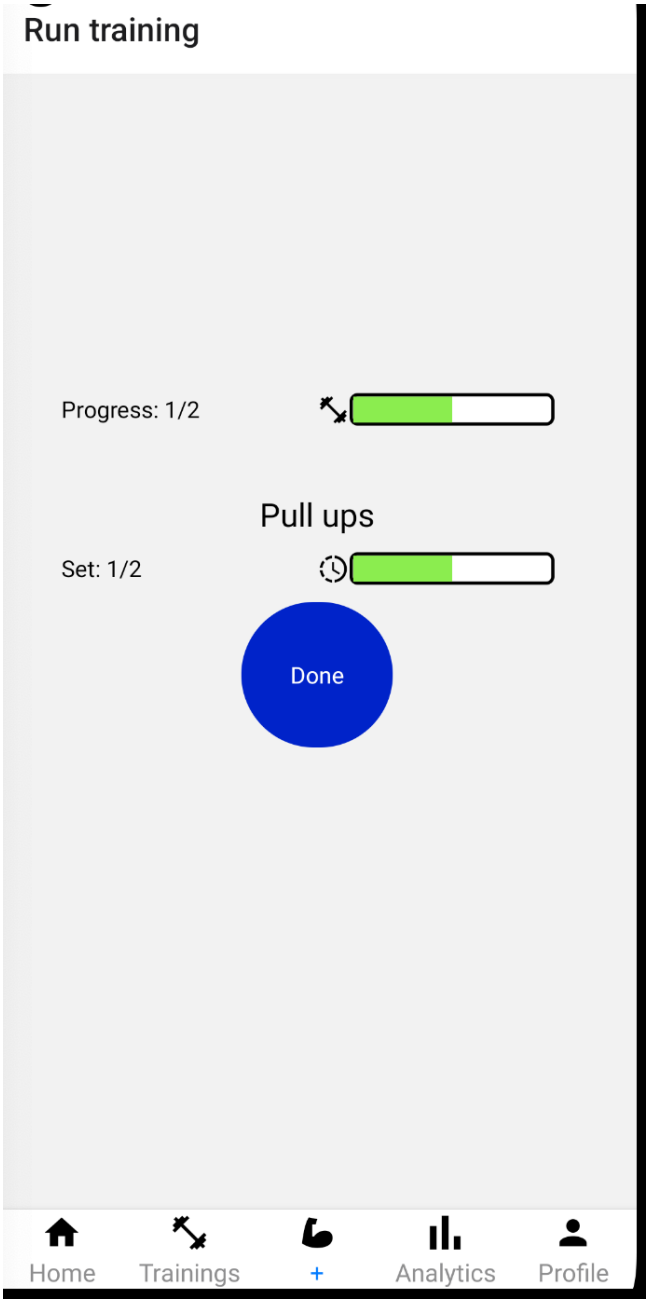

Рисунок 3.13 – Екран проведення тренування

На рисунку 3.13 зображено інформацію про проведення тренування. Відмітка progress показує прогрес виконаних вправ. Відмітка sets показує скільки підходів вже зроблено. Для того щоб завершити підхід достатньо натиснути на кнопку Done.

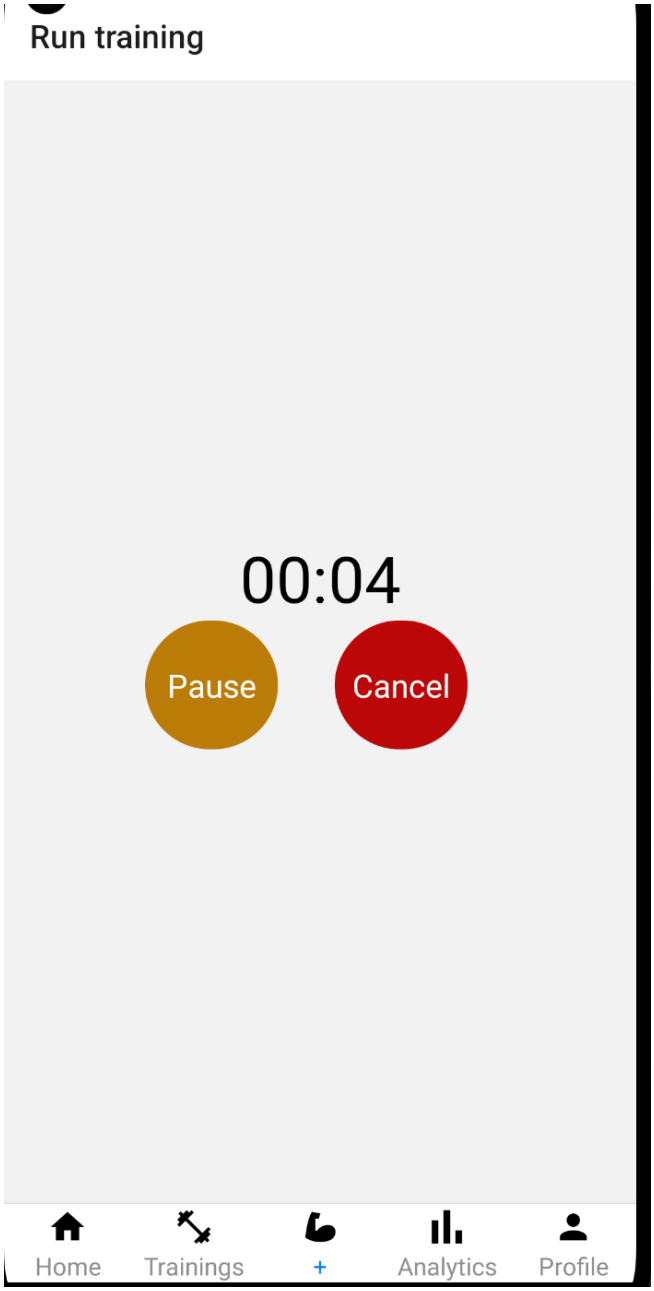

Рисунок 3.14 – Екран відліку часу відпочинку

На рисунку 3.14 зображено екран відліку часу відпочинку. У налаштуваннях другої вправи було встановлено відпочинок 5 секунд між підходами, тому між підходами цієї вправи автоматично буде запускатися лічильник зворотного відліку відпочинку.

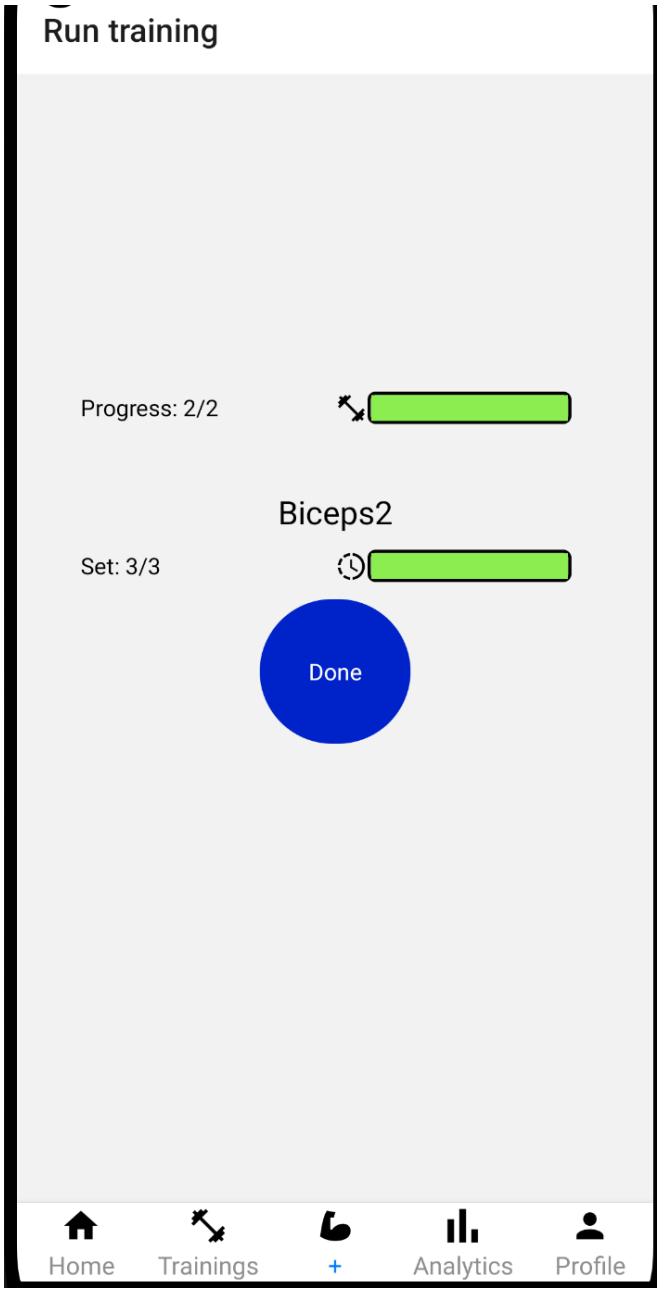

Рисунок 3.15 – Екран відліку часу відпочинку

На рисунку 3.15 зображено екран тренування у якому залишилося зробити останній підхід. Для того щоб завершити тренування достатньо так само натиснути на кнопку Done. Після цього застосунок відправить до серверу дані про проведене тренування. У разі отримання успішного статусу у відповіді сервера користувача автоматично переведе на екран Home. Після цього можна оглянути список проведених тренувань на екрані Trainings що зображено на рисунку 3.16.

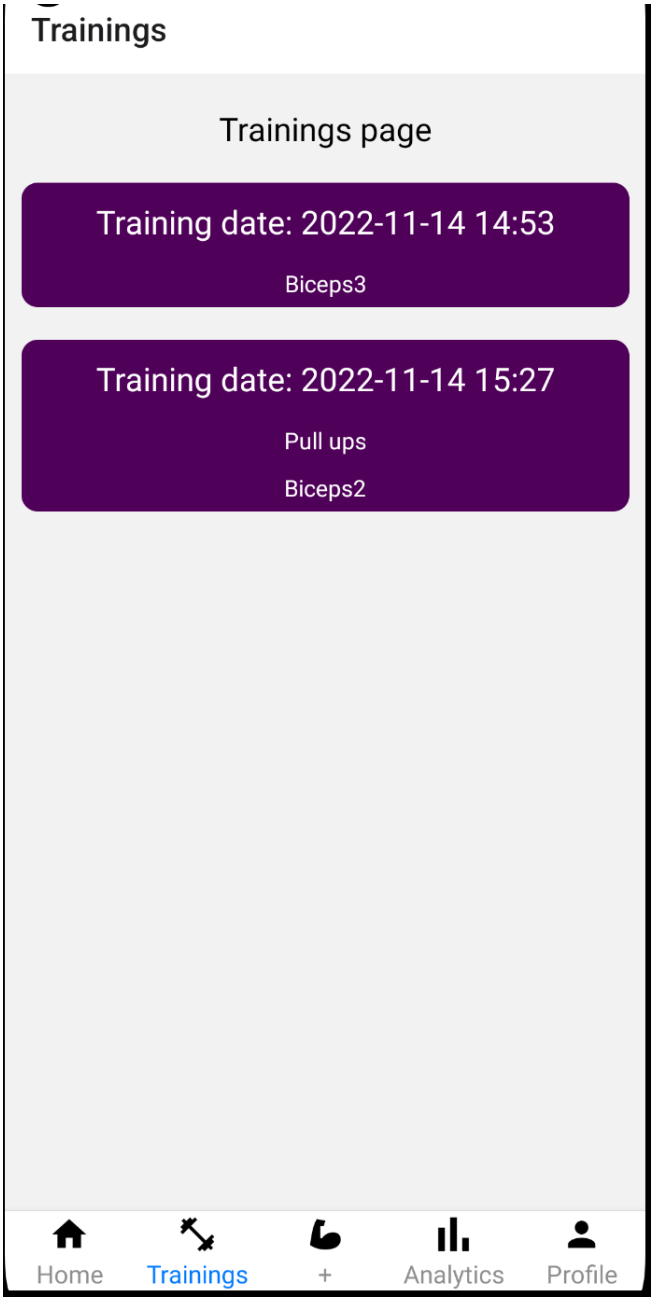

Рисунок 3.16 – Екран проведених тренувань

У мобільному додатку можна побачити більше детальну інформацію про проведене тренування. Ця інформація зображена на рисунку 3.17.

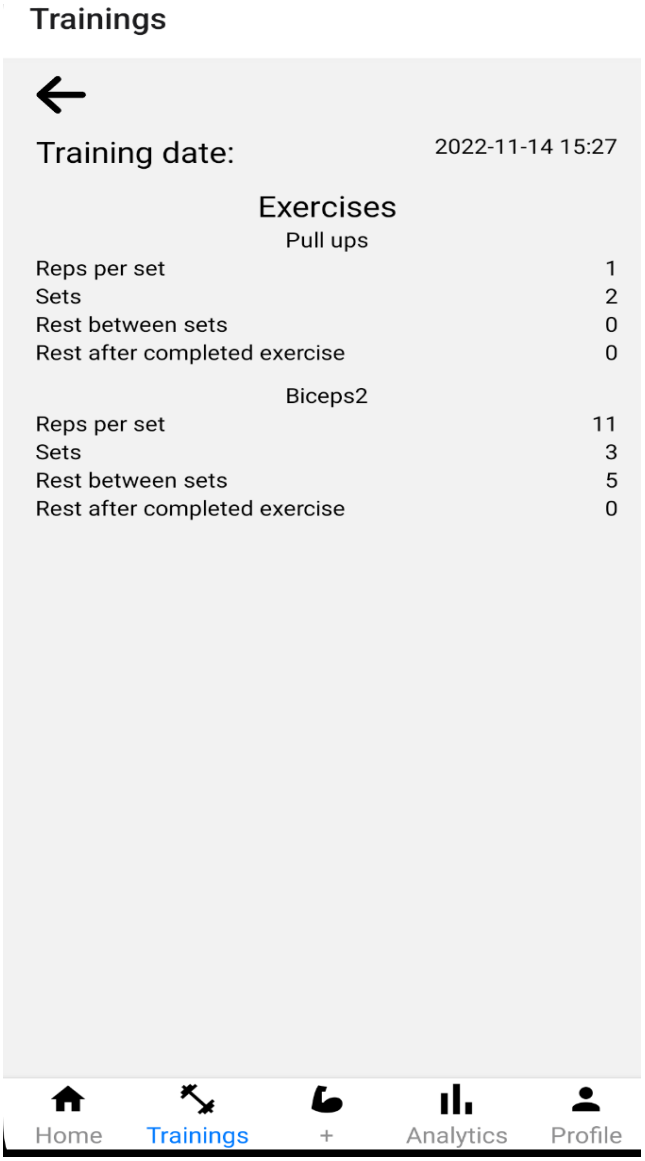

Рисунок 3.17 – Екран проведених тренувань

Після успішного проведення тренування можна розглянути аналітику відкривши натиснувши відповідний пункт у нижньому меню.

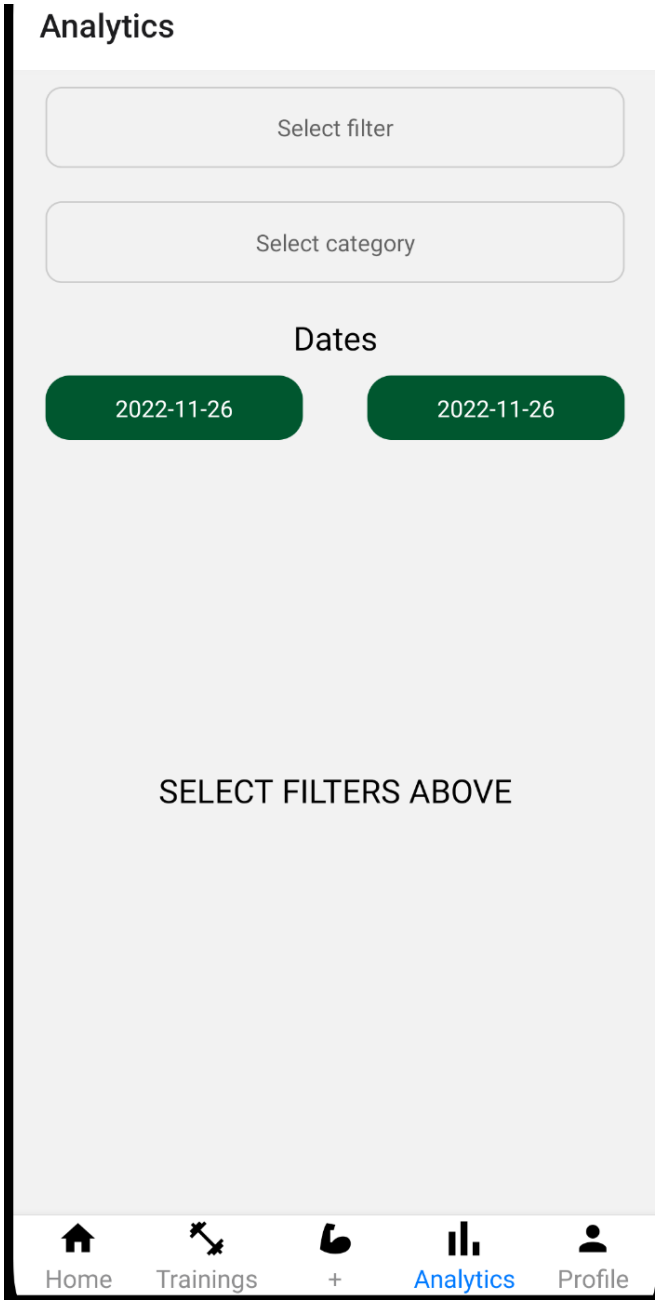

Рисунок 3.18 – Екран аналітики

На рисунку 3.18 зображено екран аналітики. Наразі графік відсутній так як не обрані фільтри для вибору даних.

У кваліфікаційній роботі реалізована аналітика за двома фільтрами – категорією та вправою, що зображено на рисунку 3.19. Також присутні фільтри періоду часу за який були проведені тренування користувачем.

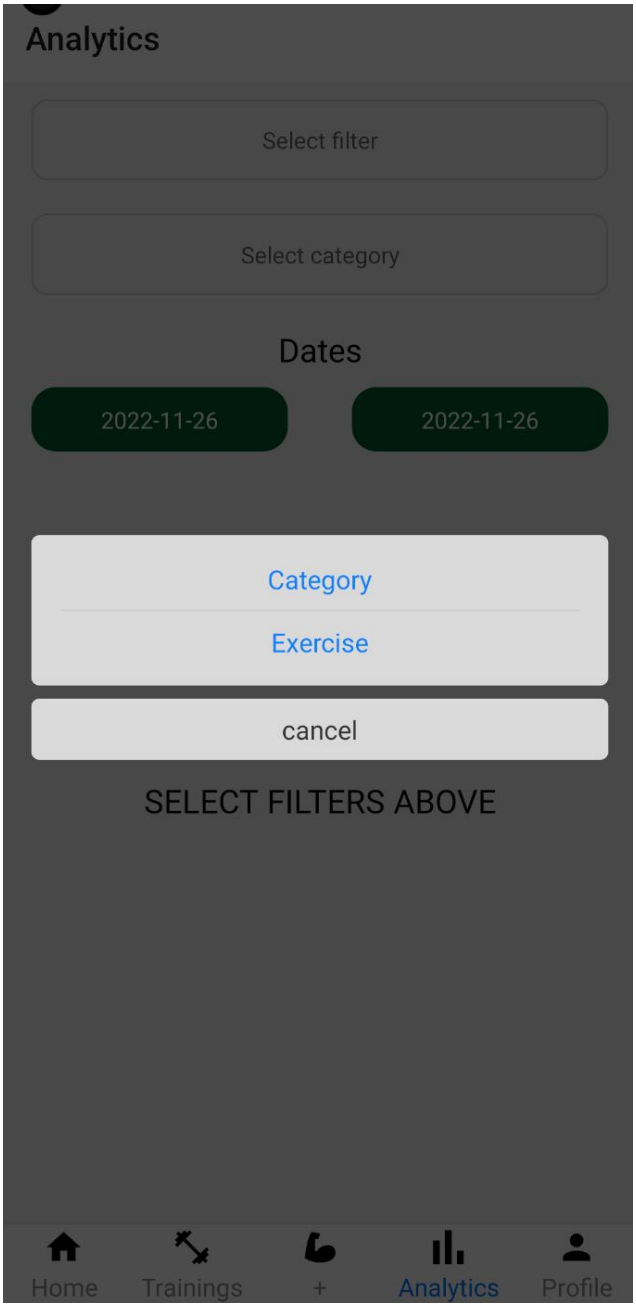

Рисунок 3.19 – Список доступних фільтрів

Після зміни або вибору потрібних фільтрів мобільний застосунок відправляє http запит на сервер з обраними фільтрами. Сервер у свою ж чергу робить SQL запит в залежності від фільтра.

```
SELECT
    (ue.sets_amount * ue.reps_per_set) as total_weight,
    e.name as exercise,
    TO_CHAR(ue.created_at, 'YYYY-MM-DD') as training_date
FROM user_trainings as ue
INNER JOIN exercises as e ON ue."exerciseIdId" = e.id
WHERE e."categoryId" = ${input.value}
    AND ue.created_at > '${input.startDate}'
    AND ue.created_at < '${input.endDate}'
ORDER BY training_date ASC
```

```
select
    (ue.sets_amount * ue.reps_per_set) as total_weight,
    TO_CHAR(ue.created_at, 'YYYY-MM-DD') as training_date,
    e.name as exercise
from user_trainings as ue
inner join exercises as e on ue."exerciseIdId" = e.id
where
    e.id = ${input.value}
    AND ue.created_at > '${input.startDate}'
    AND ue.created_at < '${input.endDate}'
ORDER BY training_date ASC
```

Рисунок 3.20 – SQL запити для отримання аналітики за категорією або вправою

На рисунку 3.20 зображено SQL запити для отримання аналітики в залежності від обраного фільтра. Той чи інший запит робиться в залежності від відправленого фільтра по http від мобільного застосунку. Реалізовано це за допомогою конструкцією switch що зображено на рисунку 3.21.

```
findAll(input: TrainingAnalyticsDto): Promise<AnalyticsResult[]> | [] {
  switch (input.filter) {
    case TrainingFilter.CATEGORY:
      return this.getAnalyticsByCategory(input);
    case TrainingFilter.EXERCISE:
      return this.getAnalyticsByExercise(input);
    default:
      return [];
  }
}
```

Рисунок 3.21 – Метод отримання аналітики

На екрані аналітики доступні фільтри категорії або вправи й фільтри періоду часу. На рисунку 3.22 зображено результат аналізу тренувань за обраними фільтрами.

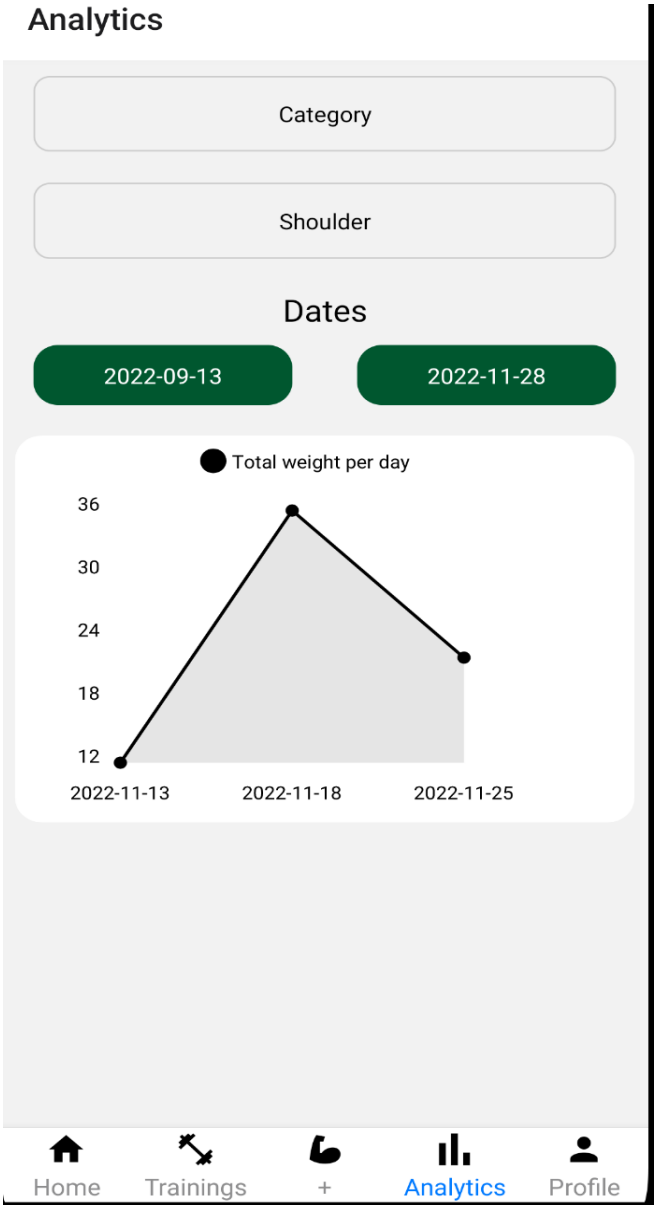

Рисунок 3.22 – Результат аналізу тренувань

ВИСНОВКИ

У кваліфікаційній роботі було розроблено мобільний фітнес застосунок засобами фреймворку React Native та платформи Node.js.

У першому розділі було проаналізовано характеристику предметної області, визначені вимоги до мобільного додатку, проведено аналіз вже існуючих застосунків і огляд використаних технологій. Застосовані технології мають такі переваги, як функціональність, швидкість.

У другому розділі було продемонстровано аналіз бізнес-процесів інформаційної системи, діаграми, які описують розробку проекту, такі як діаграма класів, діаграма прецедентів тощо.

У третьому розділі наведено деталі реалізації інтерфейсу програми для роботи з мобільним додатком. Серед іншого запропоновано систему авторизації та автентифікації користувачів.

На мою думку, інформаційну систему можна поліпшити додавши NoSQL БД для аналітики. Головною причиною додавання цією бази даних є делегування функціоналу аналітики для того щоб покращити стійкість основної бази даних.

Отже, поточний мобільний додаток є відмінною реалізацією й цілком підходить для стартапів.

ПЕРЕЛІК ПОСИЛАНЬ

1. React Native документація. URL: https://reactnative.dev/ (дата звернення: 12.05.2022).

2. React.js документація. URL: https://uk.reactjs.org/ (дата звернення: 09.05.2022).

3. JSX. URL: https://uk.reactjs.org/docs/introducing-jsx.html (дата звернення: 12.05.2022).

4. Node.js документація. nodejs.org. URL: https://nodejs.org/uk/docs/ (дата звернення: 06.03.2022).

5. Nest.js документація. nestjs.com. URL: https://nestjs.com/ (дата звернення: 19.05.2022).

6. Предметно-орієнтоване проектування. URL: https://blog.airbrake.io/blog/software-design/domain-driven-design (дата звернення: 26.07.2022).

7. PostgreSQL документація. postgresql.org. URL: https://www.postgresql.org/ (дата звернення: 15.02.2022).

8. Мобільний застосунок “Home Workout”. URL: https://play.google.com/store/apps/details?id=homeworkout.homeworkouts.noequipment&hl=ru&gl=US (дата звернення: 10.10.2022).

9. Мобільний застосунок “Fitness & Bodybuilding”. URL: https://play.google.com/store/apps/details?id=softin.my.fast.fitness&hl=ru&gl=US (дата звернення: 10.10.2022).

10. JWT. URL: https://habr.com/ru/post/340146/ (дата звернення: 26.03.2022).

11. Роберт М. Чистий код: створення і рефакторинг за допомогою Agile. Харків: «Ранок», 2021. 448 с.

12. Фрімен Е., Робсон Е., Сьєрра К., Бейтс Б. Патерни проєктування. Легкий для сприйняття довідник. Харків: «Фабула», 2020. 672 с.

ДОДАТОК А

Приклад роботи проведення тренування

exercise.model.js

```
export const REST_TYPE = {
  BETWEEN_SETS: "between_sets",
  AFTER_EXERCISE: "after_exercise",
};

export class Exercise {
  constructor(exercise = {}) {
    this.setId(exercise.id);
    this.setName(exercise.name);
    this.setSetsAmount(exercise.setsAmount);
    this.setRepsPerSet(exercise.reps);
    this.setRestBetweenSets(exercise.restBetweenSets);
    this.setRestAfterExercise(exercise.restAfterExercise);
    // training state
    this.isRestNow = false;
    this.isLastRestNow = false;
    this.currentSet = 0;
    this.isExerciseOver = false;
  }

  validateName(name) {
    if (typeof name !== "string") {
      throw new Error("Invalid exercise name");
    }
  }

  validateId(id) {
    if (!Number.isInteger(id)) {
      throw new Error("Invalid exercise name");
```

```
  }
}

setId(id = -1) {
  this.validateId(id);
  this.id = id;
}

setName(name = "") {
  this.validateName(name);
  this.name = name;
}

setRepsPerSet(reps = 0) {
  this.reps = reps;
}

setSetsAmount(amount = 0) {
  this.setsAmount = amount;
}

setRestBetweenSets(seconds = 0) {
  this.restBetweenSets = seconds;
}

setRestAfterExercise(seconds = 0) {
  this.restAfterExercise = seconds;
}

startNextSet() {
  this.increaseCurrentSet();
  this.isExerciseOver = this.checkIsExerciseOver();
}

isRestBetweenSetsExists() {
```

```
  return this.restBetweenSets > 0;
}

isRestAfterExerciseExists() {
  return this.restAfterExercise > 0;
}

increaseCurrentSet() {
  this.currentSet++;
}
getName(){
  return this.name
}

getSetsAmount() {
  return this.setsAmount;
}

getCurrentSet() {
  return this.currentSet;
}

startRestAfterExercise() {
  this.isLastRestNow = true;
}

stopRestAfterExercise() {
  this.isLastRestNow = false;
}

getRestType() {
  if (!this.isRestNow) {
    return null;
  }
  if (this.isLastRestNow) {
```

```
      return REST_TYPE.AFTER_EXERCISE;
    }
    return REST_TYPE.BETWEEN_SETS;
  }

  checkIsExerciseOver() {
    return this.currentSet >= this.setsAmount;
  }

  startRest() {
    this.isRestNow = true;
  }

  endRest() {
    this.isRestNow = false;
  }

  checkIsRestingNow() {
    return this.isRestNow;
  }

  static checkIsExerciseValid(exercise) {
    return exercise instanceof Exercise;
  }
}
```

training.model.js

```
import {Exercise} from "./exercise.model";

export class Training {
    constructor(training = {}) {
        this.setName(training.name)
        this.setExercises(training.exercises)

        this.currentExercise = null
```

```
    this.currentExerciseId = null
}

setName(name = '') {
    this.name = name
}

validateExercise(exercise) {
    if (!(exercise instanceof Exercise)) {
        throw new Error("Invalid exercise")
    }
}

validateExercises(exercises) {
    if (!Array.isArray(exercises)) {
        throw new Error("Exercises not an array")
    }
    exercises.forEach(exercise => {
        this.validateExercise(exercise)
    })
}

validateExerciseAvailability(idx) {
    if (!this.exercises[idx]) {
        throw new Error("Exercise not found")
    }
}

setExercises(exercises = []) {
    this.validateExercises(exercises)
    this.exercises = [...exercises]
}

addExercise(exercise) {
    this.validateExercise(exercise)
```

```
    this.exercises.push(exercise)
}

updateExerciseByIdx(idx, exercise) {
    this.validateExercise(exercise)
    this.validateExerciseAvailability(idx)
    this.exercises[idx] = exercise
}

removeExerciseById(idx) {
    this.validateExerciseAvailability(idx)
    this.exercises.splice(idx, 1)
}

setCurrentExerciseById(id) {
    this.currentExerciseId = id
    this.currentExercise =  this.exercises[id]
}
getCurrentExercise() {
    return this.currentExercise
}
getCurrentExerciseIndex() {
    return this.currentExerciseId
}
getTotalExercises(){
    return this.exercises.length
}

startNextExercise() {
    this.currentExerciseId++
    this.currentExercise = this.exercises[this.currentExerciseId]
    return this.currentExercise
}

checkIsTrainingOver() {
```

```
        return this.currentExerciseId === this.exercises.length -1
    }

    static initTrainingInstance(training) {
        training.exercises = training.exercises.map(el => new Exercise(el))
        return new Training(training)
    }
}
```

training.page.js

```
import CountdownTimer from "../../../components/countdown-timer/countdown-timer";
import { StyleSheet, View } from "react-native";
import { Text } from "react-native";
import userTrainingPageHook from "./userTrainingPage.hook";
import CurrentSetComponent from "../../../components/training/current-set.component";

const TrainingPage = ({ initialTraining, onEndTraining, onCancelTraining }) => {
  const {  onEndTimer,  onCancelled,  onEndSet,  exercise,  totalExercisesAmount,
    currentExerciseIndex,
    restTime,
  } = userTrainingPageHook({  initialTraining,  onEndTraining,  onCancelTraining,  });

  return (
    <View style={styles.container}>
      {restTime ? (
        <CountdownTimer  initialValue={restTime}  onEnd={onEndTimer}
onCancel={onCancelled}  />
      ) : (  <CurrentSetComponent  exercise={exercise}
exerciseOrder={currentExerciseIndex}  totalExercises={totalExercisesAmount}
onEnd={onEndSet}  />
      )}  </View>  );};

const styles = StyleSheet.create({  container: {  position: "relative",  flex: 1,
flexDirection: "column",  },});export default TrainingPage;
```

useTrainingPage.hook.js

```
import { useState } from "react";
import { REST_TYPE } from "../../../models/exercise.model";

const userTrainingPageHook = ({
  initialTraining,
  onEndTraining,
  onCancelTraining,
}) => {
  const [training, setTraining] = useState(initialTraining);
  const [renderFlag, setRenderFlag] = useState(false);
  const [exercise, setExercise] = useState(training.getCurrentExercise());
  const [currentExerciseIndex, setCurrentExerciseIndex] = useState(0);

  const totalExercisesAmount = training.getTotalExercises();
  const render = () => {
    setRenderFlag((val) => !val);
  };

  const startNextExercise = () => {
    const newExercise = training.startNextExercise();
    setCurrentExerciseIndex(training.getCurrentExerciseIndex());
    setTraining(training);
    setExercise(newExercise);
    render();
  };

  const onEndSet = () => {
    exercise.increaseCurrentSet();

    const isExerciseOver = exercise.checkIsExerciseOver();
    // start rest if exercise not completed and rest exists
    if (!isExerciseOver && exercise.isRestBetweenSetsExists()) {
      exercise.startRest();
```

```
    setExercise(exercise);
    render();
    return;
  }

  // start rest after exercise
  if (isExerciseOver && exercise.isRestAfterExerciseExists()) {
    exercise.startRestAfterExercise(); //change flag
    exercise.startRest();
    setExercise(exercise);
    render();
    return;
  }

  const isAllExercisesOver = currentExerciseIndex + 1 >= totalExercisesAmount;
  //if training over
  if (isExerciseOver && isAllExercisesOver) {
    return onEndTraining();
  }

  if (isExerciseOver) {
    startNextExercise()
    return ;
  }

  //update state for rendering component with next set of exercise
  setExercise(exercise);
  render();
};

const onEndTimer = () => {
  const restType = exercise.getRestType();
  if (restType === REST_TYPE.BETWEEN_SETS) {
    exercise.endRest();
  } else {
```

```
    exercise.stopRestAfterExercise();
  }

  const isExerciseOver = exercise.checkIsExerciseOver();
  const isTrainingOver = training.checkIsTrainingOver();

  // is training over without rest after exercise
  if (isExerciseOver && isTrainingOver) {
    return onEndTraining();
  }

  // switch training exercise
  if (isExerciseOver) {
    startNextExercise();
    return;
  }

  // exercise.startNextSet()
  setExercise(exercise);
  render();
};
const onCancelled = () => {
  onCancelTraining();
};

let restTime = getRestTime(exercise);

return {
  exercise,
  renderFlag,
  currentExerciseIndex,
  totalExercisesAmount,
  restTime,
  onEndTimer,
  onCancelled,
```

```
    onEndSet,
  };
};

function getRestTime(exercise) {
  if (!exercise) {
    return null;
  }
  const restType = exercise.getRestType();
  if (restType === REST_TYPE.BETWEEN_SETS) {
    return exercise.restBetweenSets;
  }
  if (restType === REST_TYPE.AFTER_EXERCISE) {
    return exercise.restAfterExercise;
  }

  return null;
}

export default userTrainingPageHook;
```