МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Кваліфікаційна робота бакалавра

ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ СУПРОВОДЖЕННЯ СПОРТИВНИХ ТРЕНУВАНЬ

Здобувач освіти гр. ІН – 81 Єгор СТЕЦЕНКО

Науковий керівник,
кандидат технічних наук,
доцент Ігор ШЕЛЕХОВ

Завідувач кафедри
доктор технічних наук, професор Анатолій ДОВБИШ

СУМИ 2022

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Затверджую_____
Зав. кафедрою Довбиш А.С.
"_____" _____2022 р.

ЗАВДАННЯ

до кваліфікаційної роботи

здобувача вищої освіти четвертого курсу, групи ІН-81 спеціальності «122 – Комп'ютерні науки» денної форми навчання Стеценко Єгора Юрійовича.

Тема: «ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ СУПРОВОДЖЕННЯ СПОРТИВНИХ ТРЕНУВАНЬ»

Затверджена наказом по СумДУ
№_____ від _____2022 р.

Зміст пояснювальної записки: 1) літературний огляд за обраною тематикою роботи; 2) постановка завдання для розробки; 3) вибір оптимальних інструментів для розробки інформаційної системи; 4) практична реалізація.

Дата видачі завдання «_____»_____ 2022 р.
Керівник роботи _____Ігор ШЕЛЕХОВ
Завдання прийняв до виконання _____ Єгор СТЕЦЕНКО

РЕФЕРАТ

Записка: 50 стор., 18 рис., 3 табл., 1 додаток, 10 джерел.

Об’єкт дослідження — процес проєктування веб- системи супроводження спортивних тренувань

Мета роботи — розробка інформаційного та програмного забезпечення системи супроводження спортивних тренувань

Методи дослідження — технології створення чат ботів

Результати — розроблено інформаційне та програмне забезпечення для супроводження спортивних тренувань у вигляді веб-застосунку. Створений застосунок зручний у користуванні як для тренерів, так і спортсменів, має навігацію та інтерфейс які забезпечать безперебійні тренування в режимі on-line за будь-яких умов. Розробка проводилась на базі мови програмування Typescript. У ході тестування проблем не виявлено.

TYPESCRIPT, HASURA, POSTGRESQL, GRAPHQL, СПОРТ

ЗМІСТ

ВСТУП

В наш час розвиток інформаційних технологій дає потужний поштовх і для розвитку безлічі інших соціальних, культурних та інших галузей. Веб-додатки, соціальні мережі, комп'ютерні ігри-це все те, без чого більшість людей просто не уявляє свого життя. Напрямки в яких можна застосовувати інформаційні технології складно перелічити, а навички програмування можна застосовувати будь-де.

Однією з галузей людської діяльності, яка є дуже популярною в наш час є спорт. Здобутками сучасник технологій спортсмени користуються повсякчасно. Я сам працюю тренером в Українській Федерації Карате і відчуваю, як гаджети та інтернет здатні зробити роботу тренера простішою та більш продуктивною. За допомогою інтернету я замовляю спорядження, викладаю рекламу, спілкуюся зі спортсменами та їх батьками.

Із початком першої пандемії covid-19 інтернет допоміг багатьом тренерам організовувати тренування онлайн, так само як і навчання в школах та ЗВО. При цьому в хід могли іти не найзручніші рішення. Зараз, через війну росії проти України спорт постраждав ще більше. І якщо дорослі спортсмени здатні організовувати свої тренування самостійно, то в дітей через недосвідченість такої можливості немає. Це ускладнює також і роботу тренера.

Для усвідомлення доцільності моєї роботи, мною було зібрано деяку статистику. Мною було опитано декілька знайомих тренерів з Сумської області, а також Харківської, Одеської, Київської та м. Києва. Зібрано було наступну інформацію: з близько 5000 дітей тренування в залах в Україні відвідують приблизно 34% дітей (середнє значення по зазначеним вище областям), близько 12% відвідують тренування у залах в Європі, тобто 44% не тренуються безпосередньо з тренером та іншими дітьми, при цьому з цих 44% лише трохи менше половини (17%) долучаються до тренувань онлайн, або

тренуються самостійно завдяки певним додаткам. В результаті маємо 27% дітей, що не тренуються взагалі, хоча могли б тренуватися онлайн, при цьому найпопулярнішими проблемами в даному випадку є незручний час, розклад, або недоступність безперебійного інтернету.

Саме з цих міркувань, а саме залучити якомога більше тих, хто втратив таку можливість до тренувань, метою моєї дипломної роботи є створення додатку для тренувань, що допоможе тренеру організувати тренувальний процес для всіх без винятків, незважаючи на відстань та час.

1 АНАЛІЗ ВІДОМИХ РІШЕНЬ

Для створення ефективного асистента для занять спортом, потрібно проаналізувати існуючі рішення, виявити в них переваги та недоліки.

1.1 Огляд існуючих рішень

1.1.1 Тренування в режимі on-line

Зазвичай, коли існує проблема, створюються її вирішення. Основним методологічним підходом до тренувань, коли немає доступу до залів та спеціального спорядження є тренування онлайн.

За своєю суттю онлайн тренування це те ж саме дистанційне навчання. Тренер назначає розклад і проводить тренування в зазначений час в Google Meet чи Zoom або подібних платформах.

За допомогою відеозв`язку тренер може відслідковувати як спортсмени виконують завдання. Цей спосіб тренувань онлайн оптимальний для контролю якості виконання завдань, але на практиці залучити до тренувань онлайн всю групу дітей неможливо, оскільки потрібно підібрати час, що підходить як тренеру, так і спортсменам, іноді і їхнім батькам.

У дітей є різні позашкільні заняття, репетитори, в когось друга зміна в школі, є зовсім маленькі спортсмени віком 4-5 років, які не можуть підключитися без батьків, а батьки працюють, тощо.

Мій особистий досвід переконує в цьому: в групі з 25 дітей жодного разу не вдалося долучити до регулярних тренувань онлайн більше 16. При цьому кожен досвідчений тренер знає, що регулярність - це головний аспект результативності. Підведемо підсумок за даним способом на таблиці 1.1.

Таблиця 1.1 – Плюси та мінуси тренувань в режимі on-line

Плюси	Мінуси
Можна відслідковувати якість виконання завдань.	Неможливо долучити до таких тренувань всіх дітей

Як результат, даний метод гарний, але його мінус змушує вирішувати проблему, коли деякі спортсмени не тренуються, а тому не розвиваються.

1.1.2 Додатки з програмами тренувань

Існує безліч додатків для тренувань на платформах Google Play та App Store. Такі додатки пропонують цілі програми тренувань з загальної фізичної підготовки (далі - ЗФП), але не мають баз для спеціальної фізичної підготовки (далі - СФП) та спеціальних тренувань з потрібного виду спорту (наприклад, карате).

До того ж програми і вправи вже вбудовані в додаток, їх створює не сам тренер, а тому важко оцінювати їхню доцільність. Так, наприклад, дорослі спортсмени від 16 років можуть використовувати такі програми для самостійних тренувань, робити за ними прості вправи на кшталт відтискань, підтягувань, присідань і все.

Тобто, мінімальну користь дані додатки принести можуть, але це ніяк не може замінити повноцінну методичну роботу тренера, та написану їм програму.

На таблиці 1.2 представлено плюси та мінуси таких додатків.

Таблиця 1.2 – плюси та мінуси додатків для тренувань

Плюси	Мінуси
Можна тренуватися будь-де та будь-коли.	Відсутня системність дій, неможливість виконання спеціальних фізичних та техніко-тактичних вправ.
Можуть долучитися всі охочі.	Програму пише не тренер.

1.1.3 Інші авторські тренування

Спілкуючись з багатьма іншими тренерами я спостерігаю, як вони вирішують дану проблему. Дехто викладає спортивний контент на відеохостингу YouTube або в соціальній мережі Instagram.

Це допомагає структурувати програми, але незручність виникає, коли до програми необхідно вносити певні вдосконалення. Я, наприклад, як тренер з карате можу відзняти матеріал по певній техніці і викласти його, але ж карате постійно вдосконалюється, я можу подивитися щось цікаве в інших тренерів та вдосконалити викладання техніки певної вправи або удару, що в контексті викладання в тому ж самому Instagram викликатиме незручності.

До того ж платформи YouTube та Instagram в першу чергу розважальні, а не освітні, тому несуть в собі ще й безліч відволікаючого контенту. Є ще схожий варіант: писати окремі тренування в групи спортсменів і батьків в чати, де ми спілкуємося.

Я робив так, але це зовсім незручно, адже треба обирати: або писати окреме тренування, або заповнювати весь чат програмами на кілька днів. Я вважаю, що для цього потрібен окремий простір, а не чат в месенджері (наприклад, Telegram), важливо розділяти спілкування та сам тренувальний процес, вони взаємодіють, але не заважають одне одному.

На таблиці 1.3 представлено плюси та мінуси авторських тренувань.

Таблиця 1.3 – плюси та мінуси авторських тренувань

Плюси	Мінуси
Систематизація.	Невідповідне середовище.
Доступність.	Складно вносити зміни.

Отже, ми розглянули декілька схожих рішень і тому можемо перейти до постановки задачі.

1.2 Постановка задачі

Врахувавши всі мінуси та плюси існуючих методів я ставлю перед собою завдання створити додаток, що об`єднає в собі плюси, та зачепить якомога менше мінусів. Головними вимогами для мене є зручність для тренера та доступність для спортсмена, при цьому орієнтуватися буду на дітей, тобто чим простішою буде взаємодія з додатком-тим краще. З огляду на це мною було висунуто такі вимоги:

1. Реєструватися може тренер, спортсменам реєстрація непотрібна, відкрив додаток-тренуєшся, швидко, просто, зручно.
2. Реєструватися хто-завгодно не може, щоб запобігти появу невідповідного контенту на платформі, тому має існувати код-слово, доступ до якого матимуть знайомі мені тренери, яким я довіряю викладати контент.
3. Тренери мають профіль, створюють програми тренувань і тренування.
4. В тренуванні можна створити відео опис тренування, а також текстовий опис тренування.

Для досягнення поставлених задач та нашої мети в цілому потрібно виконати такі завдання:

1. Огляд та аналіз програмних засобів для реалізації поставленої мети
2. Проектування веб-застосунку та бази даних
3. Розробка веб-застосунку та наповнення бази даних
4. Тестування сторонніми користувачами

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Огляд та вибір мови програмування для веб-застосунку

Розглянувши схожі рішення, переходимо до вибору мови програмування, за допомогою якого ми будемо вести розробку нашого застосунку. Для цього нам потрібно розглянути існуючі мови програмування, які використовуються при розробці веб-застосунків.

- Першою ми розглянемо мову програмування Python. Дана мова являє собою так званий «швейцарський ніж», адже може використовуватися у різноманітних цілях: автоматизація, наука про дані, ігрова індустрія тощо. Не оминула дана мова програмування і веб-розробку: такі фреймворки, як Django, Flask, aiohttp та FastAPI використовуються у популярних сервісах, якими користується численна кількість людей. Мова йде про YouTube, Instagram, Reddit.
- Не менш популярною мовою є JavaScript, яка хоч і не настільки універсальна, проте зосереджена на розробці саме веб-застосунків. Перевагою JavaScript є те, що за допомогою нього можна розробити як front-end частину застосунку, так і back-end. Наприклад, для front-end є такі популярні фреймворки, як React, Angular, Svelte, а для back-end – Node.js, Express.
- Останньою мовою, яку ми розглянемо, є PHP. Популярність даної мови настільки висока, що не потрібно навіть зупинятися на її представленні, а лише зазначити, що не дивлячись на свій вік, мова досі набуває розвитку та має в своєму арсеналі фреймворки на кшталт Laravel, які дозволяють легко та ефективно розробити back-end частину застосунку.

Отже, розглянувши можливі мови програмування для реалізації поставленої мети, було обрано TypeScript. Дана мова програмування

розроблена компанією Microsoft та є своєрідним розширенням можливостей JavaScript.

Перевагами даної мови програмування над традиційним JavaScript є:

1. Статична типізація. Тобто, тепер ми безпосередньо в коді можемо визначати, який саме тип треба приймати та що робити у випадку, якщо на вхід прийшло значення не того типу.
2. Повноцінні класи. Завдяки TypeScript, відтепер в коді можливе втілення класів, як у традиційному об’єктно-орієнтованому програмуванні (ООП).
3. Підтримка підключення модулів.

   Дані особливості дійсно є перевагами, адже вони підвищують ефективність виконання коду, а також впливають на його подальшу підтримку, рефакторинг тощо.

2.2 Вибір допоміжних інструментів для розробки

Окрім використання мови програмування, нам також треба визначити, які допоміжні інструменти будуть використані під час розробки застосунку.

В якості СУБД було обрано PostgreSQL, яка має відкритий вихідний код та переваги над іншими реляційними системами управління базами даних.

PostgreSQL має такі переваги над іншими СУБД:

1. Широкий вибір структур та типів даних, яким не може похизуватися будь-яка інша СУБД. PostgreSQL підтримує uuid, грошовий, перераховуємий, геометричний, бінарний тип даних, мережеві адреси, бітові рядки, масиви, композитні типи та діапазони тощо.
2. Дана СУБД має мінімальні обмеження за розмірами таблиць. Наприклад, у PostgreSQL одна таблиця може займати до 32 ТБ даних, рядок до 1.6 ТБ, а поле – до 1 ГБ.

3. Ну і найголовніша перевага – це те, що PostgreSQL є не просто реляційною, а об’єктно-реляційною СУБД, що робить її гнучкою, на відміну від конкурентів.

Також нам потрібно визначитися, як саме відбуватиметься взаємодія клієнту та серверу веб-застосунку. Звичайно, що традиційно використовується REST API, проте ми використовуватимемо GraphQL – синтаксис, що описує як саме запитувати та передавати дані.

Однією з особливостей GraphQL є те, що дані передаються та отримуються у вигляді Javascript Object Notation (JSON) і це є уніфікованим форматом для GraphQL. На рисунку 2.1 представлено приклад вигляду даних у GraphQL.

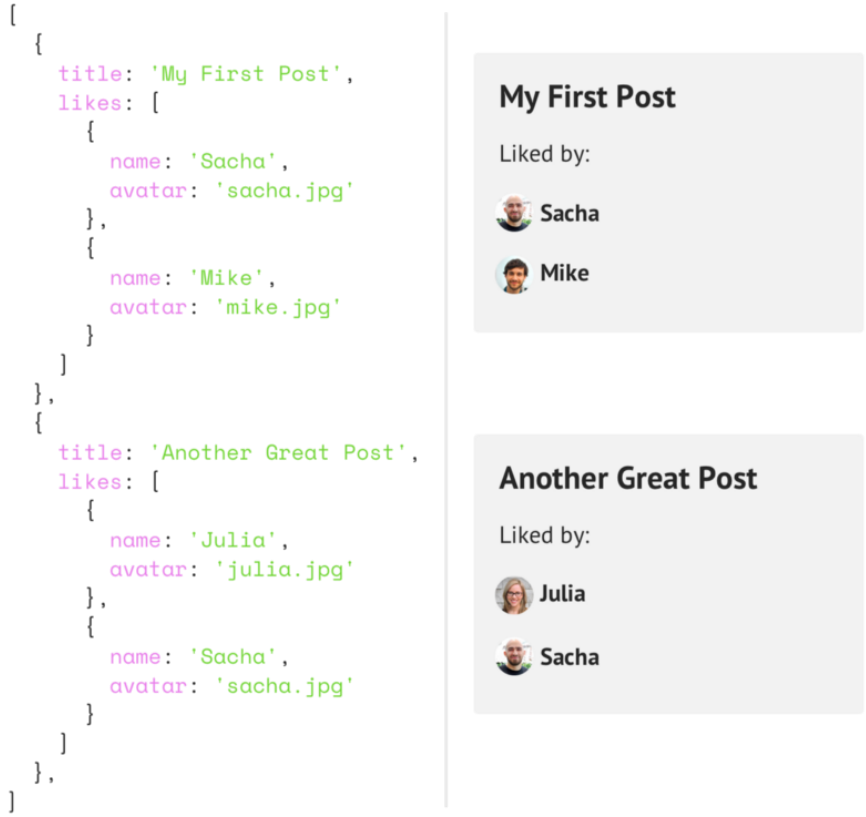

Рисунок 2.1 - Приклад даних у GraphQL

Ще однією особливістю GraphQL є те, що замість декількох end-point, він містить в собі один «розумний» end-point, за допомогою

якого будуть підготовлюватися, оброблюватися дані та передаватися так, як цього потребує клієнт. На рисунку 2.2 представлено приклад роботи з end-point, коли використовується класична архітектура REST. На рисунку 2.3 – коли це GraphQL з «розумним» end-point.

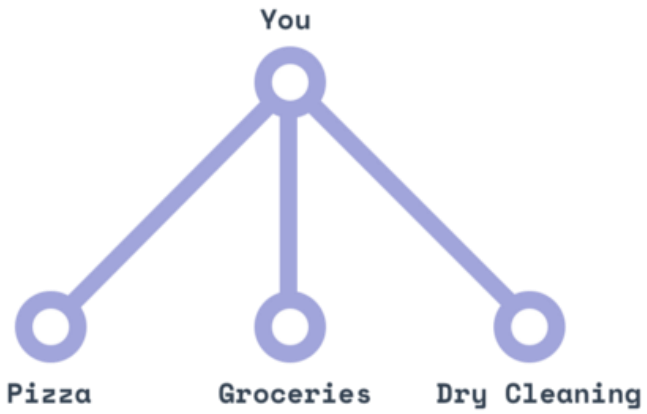

Рисунок 2.2 – End-points у REST

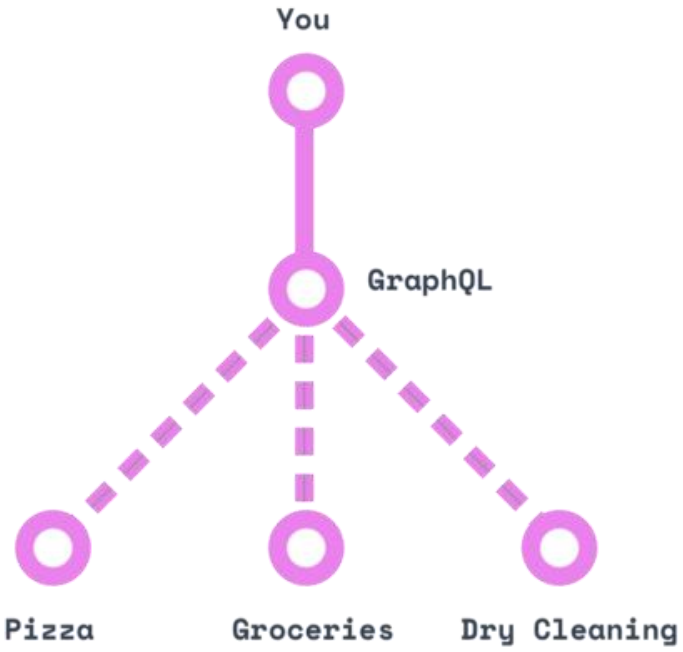

Рисунок 2.3 – «Розумний» end-point у GraphQL

Також під час розробки нашого застосунку буде інтегровано інструмент Hasura – «прошарок» між веб-застосунком, GraphQL та PostgreSQL.

Він дозволяє генерувати схему GraphQL на основі існуючої БД або створити нову. Підтримує GraphQL Subscriptions «з коробки» на основі Postgres-тригерів, динамічний контроль прав доступу, автоматичну генерацію JOIN, а також вирішує проблему N+1 запитів (batching).

За допомогою Hasura, є можливість користуватися foreign keys constraints у PostgreSQL для того, щоб отримати ієрархічні дані в одному запиті. Наприклад, ми можемо виконати такий запит:

```
{
album (where: {year: {_eq: 2018}}) {
title
tracks {
id
title
}
}
    }
```

За допомогою такого запиту ми отримаємо альбоми та відповідні треки, якщо в БД було утворено foreign key, що вказує на таблицю album.

Для розробки front-end частини веб-застосунку буде використано фреймворк Next.js, що побудований на основі React.js та призначений для розробки застосунків, що виходять за рамки «односторінкових застосунків» (SPA).

Основним недоліком SPA є те, що такі застосунки мають проблеми з індексацією сторінок пошуковими роботами, що негативно впливає на популярність ресурсу.

Для вирішення цієї проблеми було розроблено фреймворк Next.js, що має спеціальні інструменти для покращення індексації сторінок застосунку пошуковими роботами.

Тепер потрібно розглянути такі складові, як реєстрація/авторизація та роботу зі сховищем.

Для процесів авторизації та реєстрації ми використаємо Express.js – програмний каркас розробки частини веб-застосунків для Node.js.

Express надає готові абстракції, що полегшують розробку серверної логіки – наприклад, обробка надісланих форм, робота з cookies, Cors-origin resource sharing (CORS). Такі складові це якраз те, що нам потрібно для авторизації та реєстрації.

Якщо казати про роботу зі сховищем, то в даному випадку буде використовуватися мова програмування Go та сховище Amazon S3. Розглянемо дані складові окремо.

Мова програмування Go розроблена компанією Google, є компільованою та переважно використовується у back-end розробці. За допомогою Go ми вибудуємо ефективну роботу веб-застосунку зі сховищем Amazon S3. Розглянемо, чому саме це сховище ми обрали.

Amazon S3 є одним з сервісів Amazon Web Service та сховищем даних, що надає можливість зберігання й отримання будь-якого обсягу даних. Перевагами Amazon S3 над іншими є:

1. Висока масштабованість.
2. Надійність.
3. Висока швидкість.
4. Недорога інфраструктура зберігання даних.

Отже, ми розглянули усі складові для розробки нашого додатку і можемо перейти до його проектування.

2.3 Проектування веб-застосунку

Після того, як ми визначили середовище розробки нашого веб-застосунку, переходимо до його безпосереднього проектування.

Для початку, ми визначимо узагальнений перелік сторінок, які будуть присутні у нашому веб-застосунку. На рисунку 2.4 представлено узагальнену структуру нашого продукту.

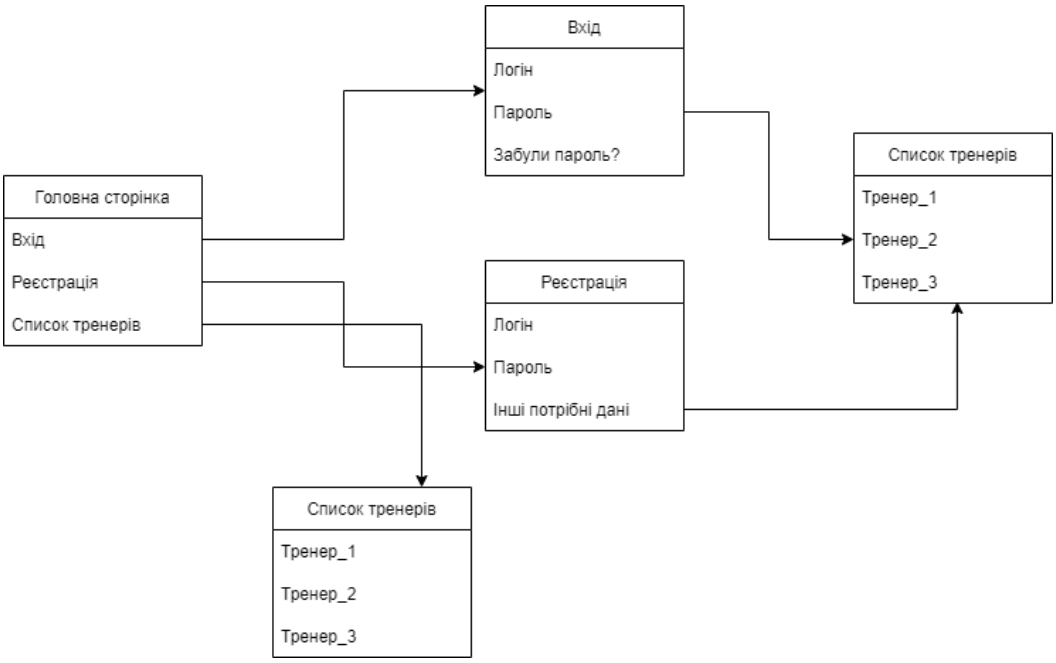

Рисунок 2.4 – Узагальнена структура сайту

Проаналізуємо дану структуру детальніше:

1. Головна сторінка. Вона міститиме у собі початкову інформацію про веб-застосунок, а також посилання на такі функції, як: вхід, реєстрація та список тренерів, які розмістили свої тренувальні програми та за якими вже можна навчатися.
2. Сторінка входу. Вона міститиме у собі поля для вводу логіна та паролю, а також функцію «Забули пароль?», яка дозволить відновити доступ до свого акаунту.
3. Реєстрація. Дана сторінка призначена для реєстрації користувача в системі. На цій сторінці користувач повинен ввести логін та пароль свого майбутнього акаунту, а також інші дані, такі як: e-mail, номер телефону, ім’я тощо.
4. Список тренерів. До даної функції на початкових етапах матимуть доступ як зареєстровані, так і незареєстровані користувачі. Це є основною функцією веб-застосунку. В ній користувач обирає одного з тренерів, після чого він повинен обрати вид фізичної підготовки, за яким планує працювати та вдосконалюватися. Різниця між зареєстрованим та

незареєстрованим користувачем полягає в тому, що перший зможе зберігати прогрес у тренуваннях з будь-якого пристрою, в той час як другому не вдасться зберегти список вже відпрацьованих вправ на різних пристроях.

Тепер перейдемо до деталізації окремих сторінок із загальної структури. На рисунку 2.5 представлено деталізацію функції «Тренування».

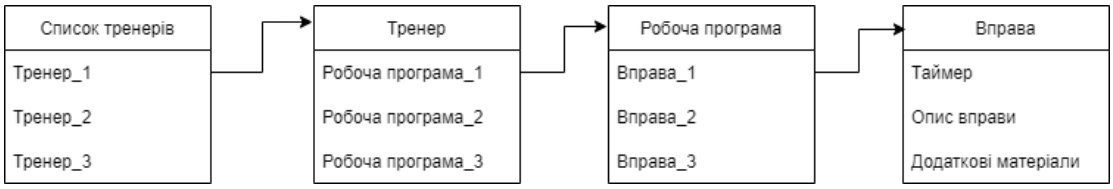

Рисунок 2.5 – Деталізація функції «Тренування»

Розглянемо подробиці даної деталізації.

1. Список тренерів. Сторінка, на якій ми можемо обрати тренера, прочитавши про нього корисну інформацію та зрозумівши, хто саме нам підійде.
2. Тренер. Потрапляючи на сторінку до тренера, ми отримуємо доступ до його робочих програм, які він розмістив у себе на сторінці.
3. Робоча програма. Набір вправ, які потрібно виконати користувачу, щоб досягнути поставлених для нього цілей.
4. Вправа. Сторінка кожної з вправ, які є у робочій програмі. На ній користувач має доступ до таймеру, щоб заміряти, скільки він витрачає на ту чи іншу вправу та згодом міг спостерігати прогрес. Наприклад, користувач на початку виконував 10 відтискань за 60 секунд, а через місяць він виконує ту саму кількість відтискань вже за 40 секунд. Також на цій сторінці присутні опис вправи, тобто що саме користувачу треба робити та додаткові матеріали, в яких може бути наглядно показано, як саме виконувати вправу.

Також нам потрібно розглянути веб-застосунок з боку тренера. Для цього ми також спроектуємо узагальнену структуру сторінок. На рисунку 2.6 представлено узагальнену структуру для тренера.

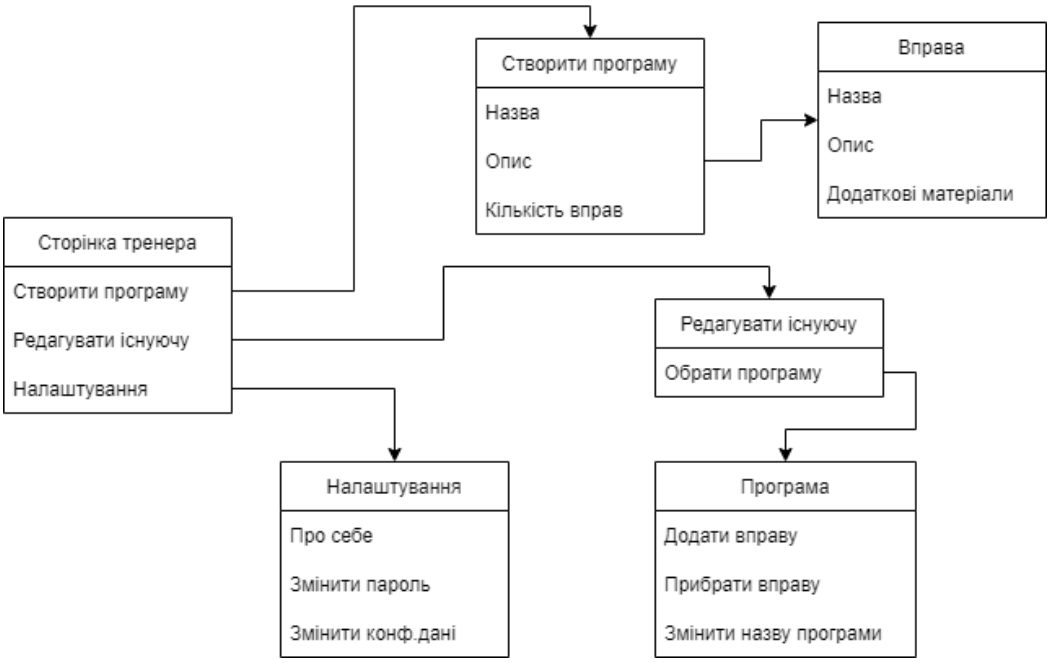

Рисунок 2.6 – Узагальнена структура для користувача-тренера

Розглянемо кожну зі сторінок, представлену в структурі, детальніше.

1. Сторінка тренера. На ній представлено посилання на 3 основні сторінки: «Створити програму», «Редагувати існуючу» та «Налаштування».
2. «Створити програму». Сторінка, за допомогою якої тренер може створити власну робочу програму та розмістити її у застосунку для користувачів. Для створення програми потрібно вказати її назву, опис та кількість вправ, які будуть в ній присутні.
3. Вправа. Це сторінка для створення вправи у робочій програмі. Тут тренер повинен вказати назву вправи, опис як саме її робити, а за бажанням може надати додаткові матеріали, на яких демонструється як правильно виконати праву.

4. «Редагувати існуючу». Дана сторінка представляє з себе список робочих програм, які є у тренера. Одну з них він повинен обрати для редагування і він потрапить на наступну сторінку.
5. Програма. Дана сторінка призначена для редагування робочої програми. На ній тренер може додати або прибрати вправу, змінити назву програми, відредагувати конкретну вправу.
6. Налаштування. На цій сторінці тренер може змінити інформацію «Про себе», змінити пароль, змінити інші конфіденційні дані (пошта, мобільний номер, секретне слово тощо), а також видалити акаунт.

Спроектувавши сторінки та функції веб-застосунку, переходимо до проектування бази даних.

2.4 Проектування бази даних

Для коректної роботи застосунку, нам потрібно зберігати дані у базі. Проте, також нам потрібно спроектувати, в якому саме вигляді будуть присутні дані у нашій базі. Для цього ми виконаємо проектування бази даних нашого веб-застосунку.

На рисунку 2.7 представлено першопочаткову діаграму, яка містить в собі декілька основних таблиць.

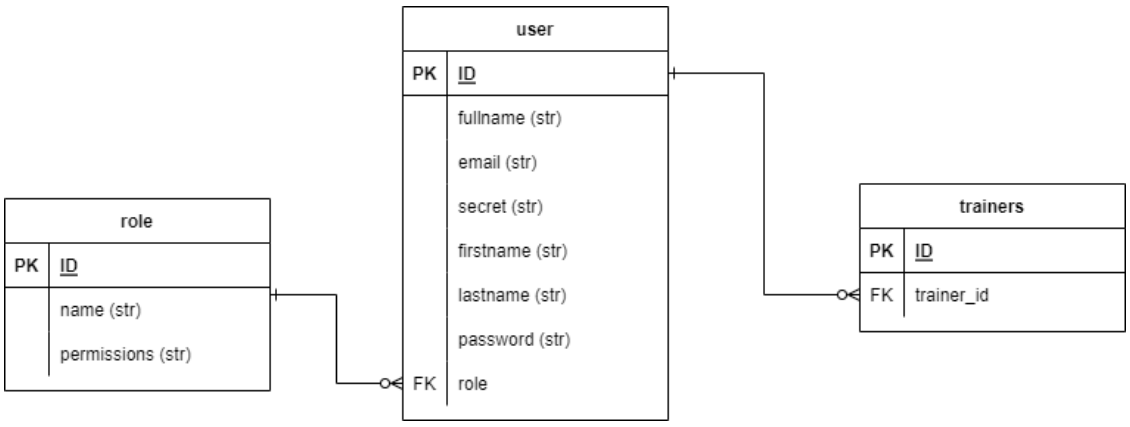

Рисунок 2.7 – Першопочаткова ER-діаграма таблиць role, user та trainers

Тепер розглянемо кожну з таблиць окремо. Почнемо з розгляду таблиці user. Вона містить в собі такі колонки:

1. ID (PK)
2. Fullname (string) – повне ім’я користувача, яке буде відображатися у нього в особистому кабінеті або на сторінці, якщо це тренер.
3. Email (string) – адреса електронної скриньки користувача, на яку він після реєстрації отримає посилання для підтвердження аккаунту, а потім зможе отримувати інформацію про майбутні тренування.
4. Secret (string) – секретне слово користувача, за допомогою якого він зможе відновити доступ до своєї сторінки, в разі якщо було загублено пароль.
5. Firstname (string) – ім’я користувача.
6. Lastname (string) – прізвище користувача.
7. Password (string) – пароль користувача.
8. Role (FK) – роль користувача.

Так як role в цій таблиці у вигляді зовнішнього ключа, то перейдемо до розгляду однойменної таблиці. Вона містить в собі такі дані:

1. ID (PK)
2. Name (string) – назва ролі.
3. Permissions (string) – набір дозволів для певної ролі.

Варто зазначити, що на початковому етапі існує декілька ролей: користувач (user), модератор (moderator), адміністратор (administrator) та тренер (trainer).

Перейдемо до розгляду таблиці trainers. Вона міститиме в собі список тренерів, які зареєструвалися на даному ресурсі. Її колонки виглядають так:

1. ID (PK)
2. Trainer_id (FK) – ідентифікатор користувача, у якого значення role є trainer. Тобто, користувач, який є тренером у системі.

Розглянувши основні таблиці, перейдемо до проектування таблиць, що пов’язані з відносинами між користувачами, тренерами, робочими

програмами та вправами в них. На рисунку 2.8 представлено ER-діаграму таблиць programs, exercises, user_progress, trainer_stats.

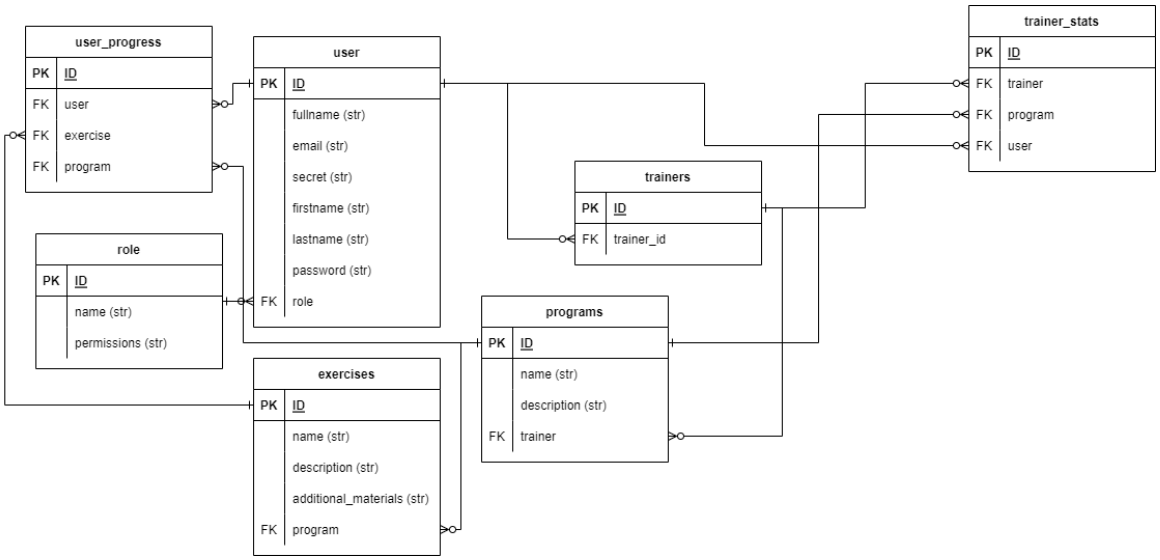

Рисунок 2.8 – ER-діаграма таблиць programs, exercises, user_progress, trainer_stats

Розглянемо кожну нову таблицю з даної діаграми окремо. Почнемо з programs. В цій таблиці будуть зберігатися дані про тренувальні програми, які створюють тренери. Programs містить в собі такі колонки:

1. ID (PK)
2. Name (string) – назва програми.
3. Description (str) – опис тренувальної програми, в якому вказується для кого вона підійде та навіщо вона взагалі.
4. Trainer (FK) – тренер, який створив дану програму.

Таблицю з вправами до кожної робочої програми було створено окремо і вона має назву exercises. В ній будуть зберігатися такі дані:

1. ID (PK)
2. Name (string) – назва вправи.
3. Description (string) – опис вправи, в якій пояснюється як саме виконувати вправу: кількість, послідовність тощо.

4. Additional_materials (string) – додаткові матеріали, які допоможуть користувачу зрозуміти, як саме виконати цю вправу. Це може бути посилання на зображення, на відео або на інструкцію з окремого ресурсу, де більш деталізовано описано перелік дій для виконання вправи.
5. Program (FK) – тренувальна програма, в якій знаходиться вправа.

Тепер перейдемо до опису таблиці user_progress, яка майже повністю побудована на зовнішніх ключах (foreign keys). В ній міститься інформація про користувачів та вправи, які вони виконали. Дана таблиця містить такі колонки:

1. ID (PK)
2. User (FK) – користувач, який виконав вправу.
3. Exercise (FK) – вправа, яка була виконана користувачем.
4. Program (FK) – тренувальна програма, в рамках якої була виконана вправа.

І останньою спроектованою таблицею, яку ми розглянемо, є trainer_stats. Вона зберігатиме в собі статистику для тренерів, на даний момент – скільки користувачів переглянули або пройшли їхні програми. Дана таблиця також майже повністю побудована на зовнішніх ключах і містить в собі такі дані:

1. ID (PK)
2. Trainer (FK) – тренер, до якого належить створена ним тренувальна програма.
3. Program (FK) – тренувальна програма, яка була створена тренером.
4. User (FK) – користувач, який переглянув або пройшов тренувальну програму.

Отже, ми спроектували базу даних майбутнього веб-застосунку і тепер можемо перейти безпосередньо до процесу розробки.

3 КОМП'ЮТЕРНА РЕАЛІЗАЦІЯ ПРОЕКТУ ТА ТЕСТУВАННЯ

3.1 Програмна реалізація веб-застосунку

Коли розглянуто схожі рішення, обрано середовище розробки, допоміжні інструменти для неї, а також спроектовано веб-застосунок та його базу даних, ми можемо перейти безпосередньо до процесу розробки.

Почнемо з найпростішого – розробка процесів реєстрації та авторизації. При розробці даних процесів варто врахувати наступні чинники:

1. E-mail повинен бути вигляду name@example.com. Тобто, повинні допускатися лише дозволені у електронних адресах спеціальні символи, повинні бути присутніми доменне ім'я, де саме розташовано поштову скриньку. Також допустимим варіантом є name.lastname@example.com, інші варіації допускати не можна.
2. Пароль повинен складатися мінімум з 8 символів, 1 спеціального символу, 1 цифри та 1 великої літери.
3. Після реєстрації, користувачу потрібно підтвердити акаунт за допомогою посилання, яке прийде на електронну пошту, адже може бути ймовірність, що користувач вказав невірну адресу і тому при спробі відновити пароль можуть виникнути труднощі.

Для перших двох випадків було передбачено так звані «валідатори значень», які ще на етапі введення користувачем даних перевіряють, чи у вірному форматі він їх надає. Якщо ні, то користувач прямо на сторінці отримує повідомлення про те, що саме потрібно виправити у його реєстраційній формі.

Для третього випадку було розроблено код, який генерує посилання для підтвердження поштової скриньки та надсилає його користувачу. Користувач переходить до своєї поштової скриньки, знаходить повідомлення від застосунку та переходить за посиланням, таким чином підтверджуючи свій

акаунт. Вихідний код даних процесів можна знайти у додатку до дипломної роботи.

Тепер переходимо до розробки функції «Тренування». Потрібно розглянути нюанси даної функції як з боку звичайного користувача, так і з боку тренера. Почнемо з користувача, при розробці функції для якого потрібно врахувати наступне:

1. Один користувач має доступ до декількох тренувальних програм. В той же час, декілька користувачів можуть мати доступ до однієї тренувальної програми.
2. Користувач повинен мати усе під рукою: секундомір, пояснення як виконати вправу, приклад виконання вправи.
3. Користувач може займатися у декількох тренерів. Аналогічно, декілька користувачів можуть займатися у одного тренера.
4. Користувач повинен бачити не перевантажений графічно сервіс. Для нього він повинен бути зручним, а головне адаптивним, щоб він міг користуватися застосунком як з комп’ютера, так і з планшета або телефону.
5. При розробці графічної складової, варто також врахувати user-experience (UX, досвід користувача). Користувач може змінювати вправу безпосередньо в процесі тренування або зупиняти секундомір.

Тепер розглянемо, які саме чинники потрібно врахувати при розробці застосунку для тренера.

1. Одну тренувальну програму можуть проходити декілька користувачів. Аналогічно, декілька тренувальних програм може проходити один користувач.
2. Тренер може мати декілька власних тренувальних програм. В той же час, одна тренувальна програма не може належати декільком тренерам.

3. В тренувальній програмі тренер може розмістити декілька однакових вправ.
4. Тренер повинен мати можливість проводити CRUD-операції з тренувальними програмами. Тобто, він може створювати (create), ознайомлюватися (read), редагувати (update) або видаляти тренувальні програми.
5. Аналогічно, тренер повинен мати змогу відредагувати інформацію про себе: в якому клубі він тепер працює, фотографія яка ідентифікує тренера, місто в якому працює тощо. Окрім цього, він повинен мати доступ до базових операцій: зміна паролю, електронної пошти.

Враховуючи ці чинники, було розроблено основний модуль веб застосунку «Тренування». Вихідний код цього процесу як для тренера, так і для користувача, представлено у додатку до дипломної роботи.

Провівши розробку веб-застосунку, ми можемо перейти до процесу його тестування як зі сторони тренера, так і зі сторони користувача.

3.2 Тестування веб-застосунку

Розробивши наш веб-застосунок для тренувань, потрібно переконатися, що він працює правильно. Почнемо тестування з точки зору тренера.

3.2.1 Тестування в ролі тренера

Для того, щоб розпочати тестування у цій ролі, потрібно зареєструватися як тренер. Переходимо на сторінку реєстрації та заповнюємо її даними (рисунок 3.1).

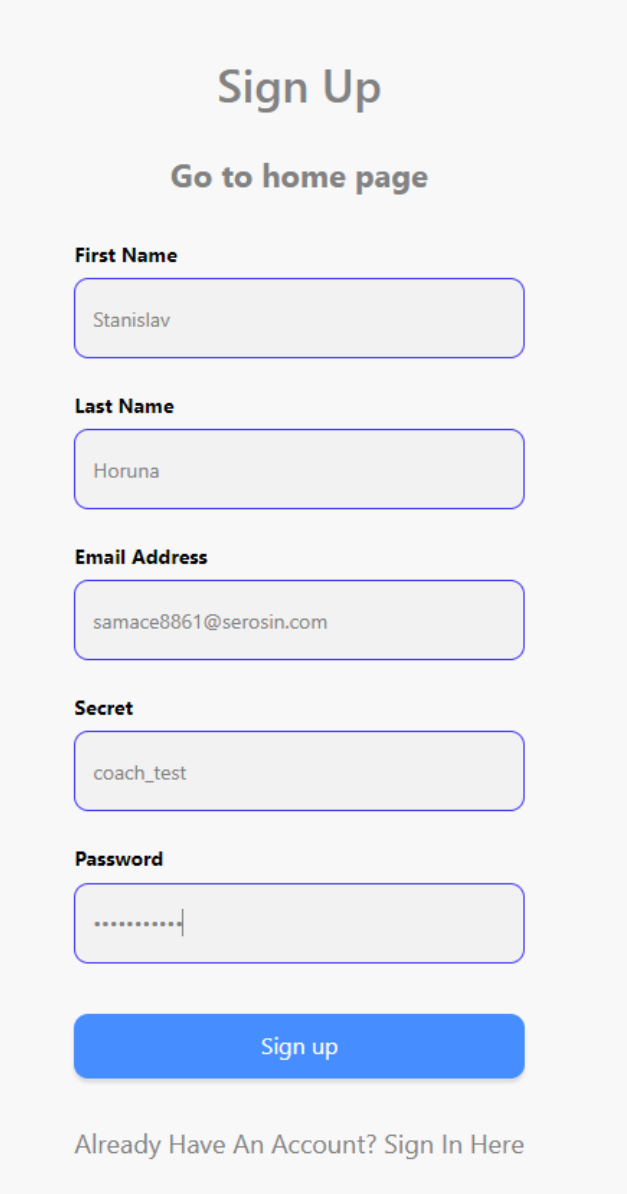

Рисунок 3.1 – Реєстраційна форма для тренера

Натискаємо на кнопку Sign Up та переходимо до поштової скриньки, на яку ми маємо отримати посилання для підтвердження нашого акаунту. Як представлено на рисунку 3.2, бачимо що дане посилання було надіслано та отримано нами успішно.

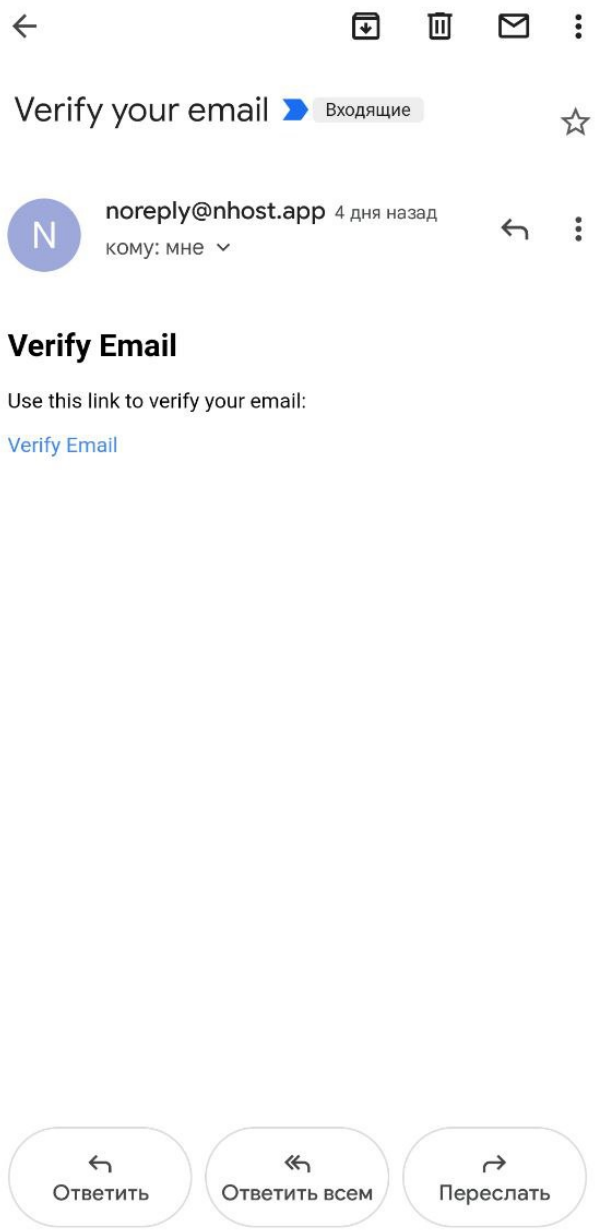

Рисунок 3.2 – Лист з посиланням для підтвердження нашого акаунту

Після того, як ми виконали підтвердження акаунту, перейшовши за посиланням, можемо увійти в систему з даними, які ми надали під час реєстрації. На рисунку 3.3 представлено форму для авторизації.

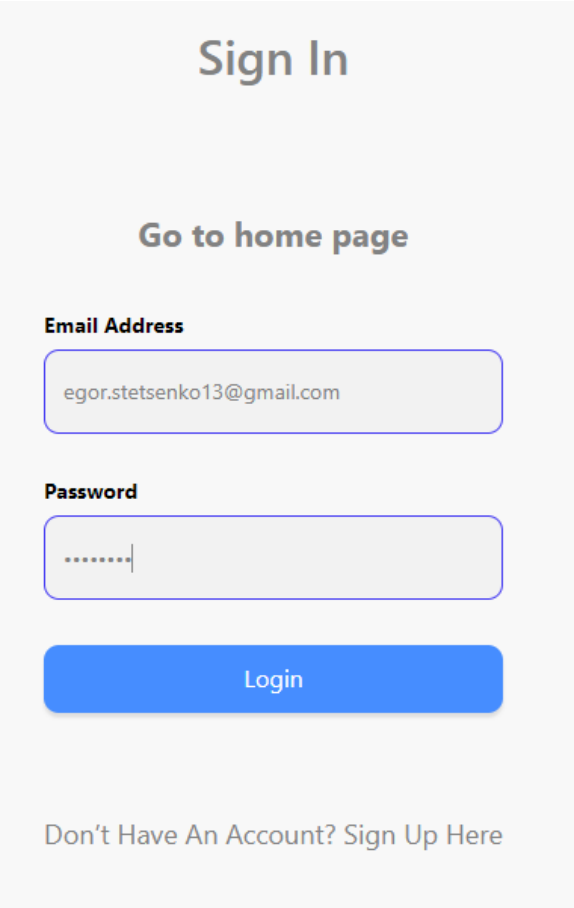

Рисунок 3.3 – Форма для авторизації користувача

Після того, як ми правильно ввели наш логін та пароль, ми потрапляємо до особистого кабінету, який представлений на рисунку 3.4.

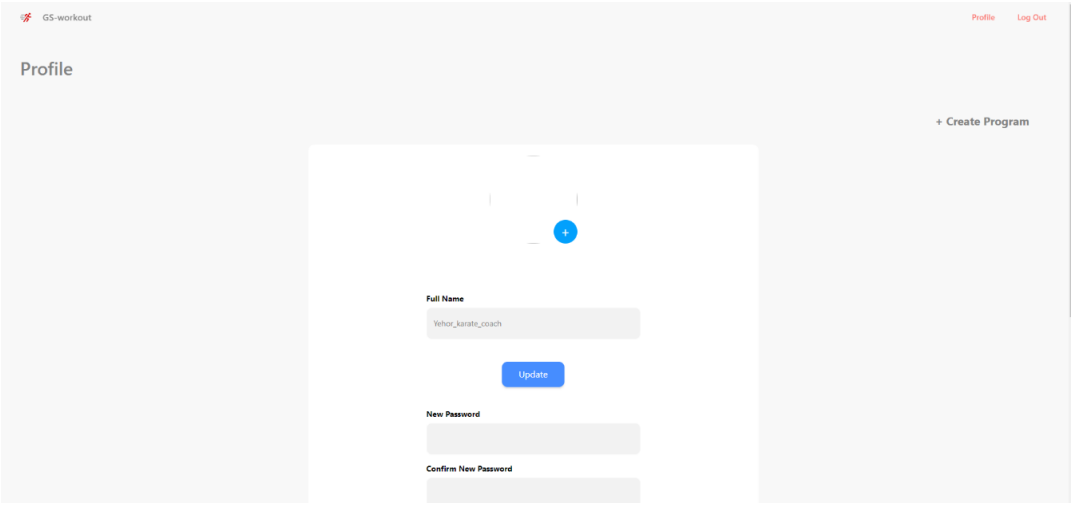

Рисунок 3.4 – Особистий кабінет тренера

На ньому ми можемо побачити такі елементи, як:

1. Кнопка «Create program». За допомогою неї ми можемо створити тренувальну програму для користувачів.
2. Шапка з кнопками Profile, яка поверне нас до нашого профілю та Log Out, за допомогою якої ми можемо вийти з акаунту.
3. Меню налаштування користувача. В ньому ми можемо змінити особисте фото, власне ім’я, а також пароль.

Якщо опуститися нижче, то ми можемо переглянути вже існуючі тренувальні програми. Дане меню представлено на рисунку 3.5.

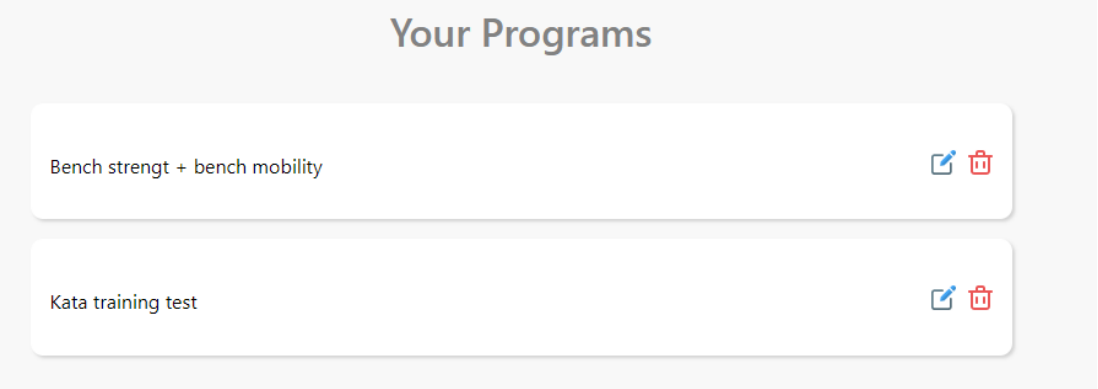

Рисунок 3.5 – Список тренувальних програм, які вже були створені тренером

В даному меню ми можемо побачити наступне:

1. Безпосередньо, список тренувальних програм, які вже були створені тренером.
2. Іконки для редагування тренувальної програми та її видалення.

Перейдемо до редагування однієї з програм. Нас зустрічає зручне меню з потрібними функціями, що представлено на рисунку 3.6.

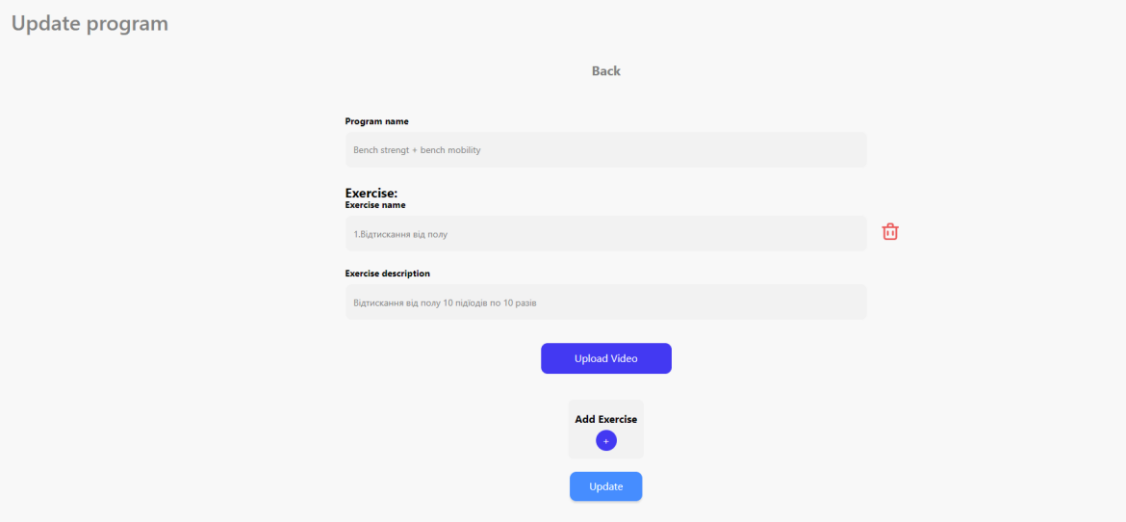

Рисунок 3.6 – Меню редагування тренувальної програми

В даному меню ми спостерігаємо наступні елементи:

1. Поля «Назва програми», «Назва вправи» та «Опис вправи». В них ми вказуємо потрібні дані, які зможуть переглянути користувачі.
2. Кнопка «Завантажити відео», яка дозволяє додати до вправи додаткові матеріали, що показують як саме потрібно виконувати вправу.
3. Кнопка «Додати вправу», яка дозволяє додати ще одну вправу до тренувальної програми.
4. Кнопка «Оновити», яка оновлює дані про програму та її вправи.

Ми виконали тестування веб-застосунку з боку тренера і тепер можемо перейти до тестування в ролі користувача.

3.2.2 Тестування в ролі користувача

Процес реєстрації та входу користувача аналогічний, як і у тренера, тому проведемо тестування як неавторизований користувач. На головній сторінці нас зустрічає список тренерів, у яких ми можемо пройти тренувальні програми (рис.3.7).

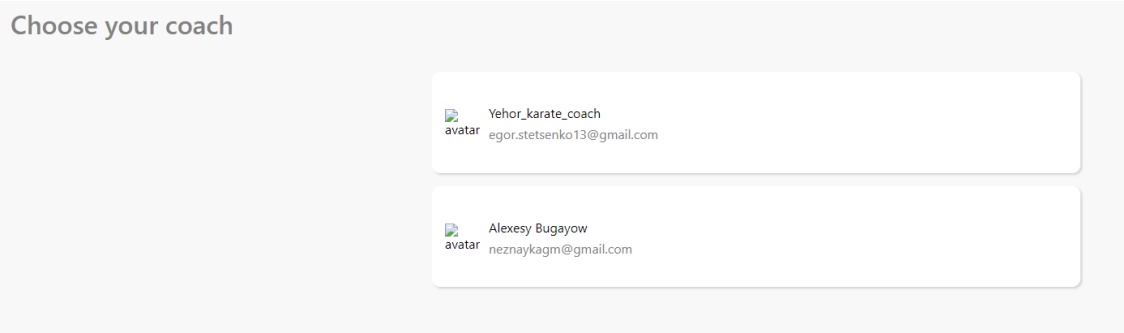

Рисунок 3.7 – Список тренерів, що зустрічає користувача

Переходимо до тренера «Yehor_karate_coach» та бачимо, що він має 2 тренувальні програми – для підвищення витривалості та для відпрацювання ката (рис.3.8).

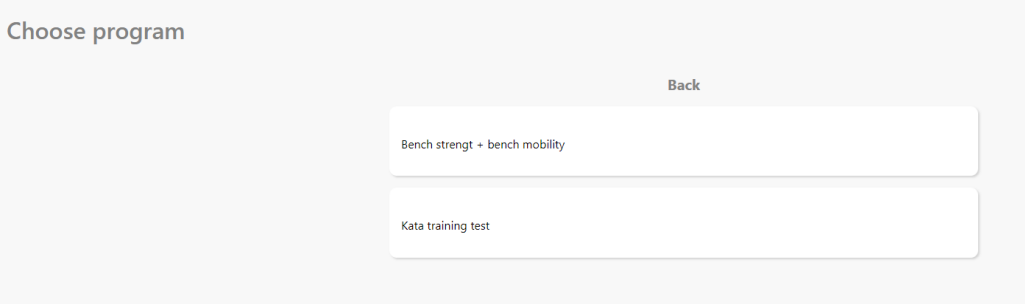

Рисунок 3.8 – Список тренувальних програм тренера «Yehor_karate_coach»

Обираємо одну з програм – наприклад, для підвищення витривалості. На рисунку 3.9 представлено меню з першою вправою даної програми.

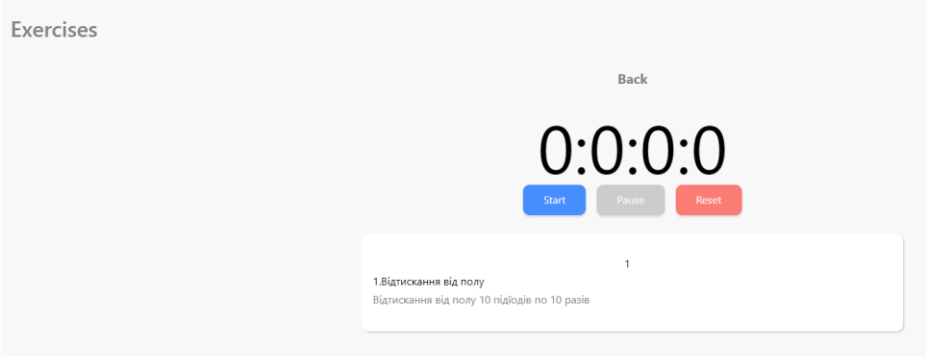

Рисунок 3.9 – Меню вправи у тренувальній програмі для підвищення витривалості

Як ми бачимо, нас зустрічають такі складові:

1. Секундомір. Потрібен для того, щоб користувач міг заміряти, за скільки часу він виконує ту чи іншу вправу.
2. Вправа. Її назва та опис, як саме правильно робити.

Також для наглядності повернемося та оберемо тренувальну програму для відпрацювання ката. На рисунку 3.10 представлено, як виглядає вправа у цій програмі.

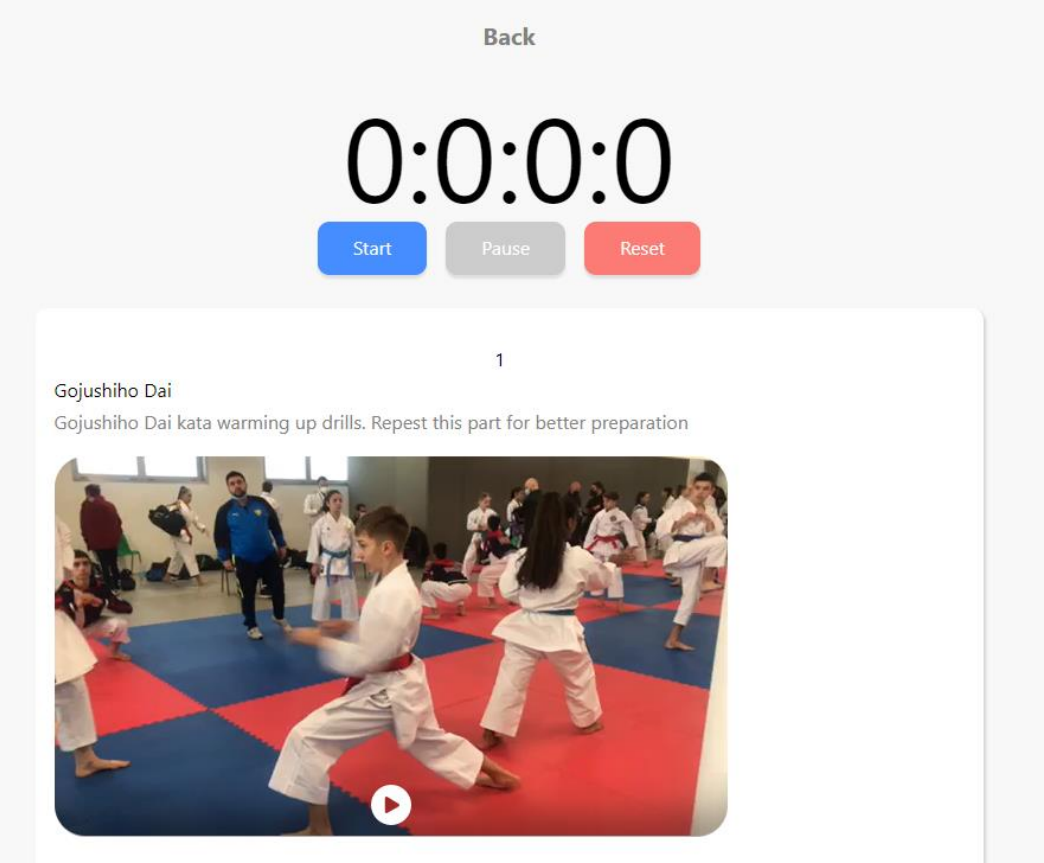

Рисунок 3.10 – Вправа з додатковими матеріалами

Як ми бачимо, окрім звичайних елементів, які були описані в минулій вправі, тут також додане відео для того, щоб наглядно показати як саме виконувати користувачу вправу.

Отже, ми виконали тестування з боку користувача і можемо зробити висновок, що веб-застосунок для тренування працює правильним чином.

ВИСНОВКИ

В ході виконання роботи були вирішені наступні задачі:

- проаналізовано та обрано мову програмування, а також допоміжні засоби для розробки;
- проведено огляд схожих рішень та аналіз доцільності продукту з боку тренера та кінцевого користувача
- спроектовано веб-застосунок та його базу даних
- розроблено веб-застосунок та проведено тестування сторонніми користувачами

Результатом виконання роботи є створений та перевірений веб-застосунок, який допоможе тренерам проводити заняття у будь-яких умовах, на які вони не можуть вплинути; користувачам, які прагнуть розвиватися фізично або займаються спортом на постійній основі.

СПИСОК ЛІТЕРАТУРИ

1. A complete guide to web-development in Python [Електронний ресурс] – Режим доступу: https://www.educative.io/blog/web-development-in-python
2. PHP Practical Example [Електронний ресурс] – Режим доступу: https://www.guru99.com/php-practical-example.html
3. Frontend vs. Backend – What’s the difference? [Електронний ресурс] – Режим доступу: https://www.freecodecamp.org/news/frontend-vs-backend-whats-the-difference/
4. Next.js – детальна інструкція [Електронний ресурс] – Режим доступу : https://habr.com/en/company/timeweb/blog/588498/
5. Кишенькова книжка по Typescript. Частина 1 [Електронний ресурс] – Режим доступу : https://habr.com/en/company/macloud/blog/559902/
6. Hasura Course [Електронний ресурс] – Режим доступу: https://hasura.io/learn/graphql/hasura/introduction/
7. PostgreSQL Tutorial [Електронний ресурс] – Режим доступу: https://www.tutorialspoint.com/postgresql/index.html
8. GraphQL vs. REST [Електронний ресурс] – Режим доступу: https://www.howtographql.com/basics/1-graphql-is-the-better-rest/
9. Bootstrap 3 Tutorial [Електронний ресурс] – Режим доступу: https://www.w3schools.com/bootstrap/
10. Amazon S3. Quick guide [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

ДОДАТОК

Основні фрагменти вихідного коду продукту

Авторизація

```
import React from 'react';
// libs
import * as yup from 'yup';
import { yupResolver } from '@hookform/resolvers/yup';
// hooks
import { useRouter } from 'next/router';
import { useForm } from 'react-hook-form';
import { useSignInEmailPassword } from '@nhost/nextjs';
import { useKeyDown } from '@md-shared/hooks/use-key-down';
import { useUIActions } from '@md-managers/use-ui-actions';
// components
import { Link } from '@md-shared/components/link';
import { Button } from '@md-shared/components/button/main';
import { FormInput } from '@md-shared/components/form';
// views
import { Wrapper, Form, LinkWrapper } from '@md-modules/auth/sign-in/views';

// types
interface FormData {
  email: string;
  password: string;
}

// validation
const schema = yup.object().shape({
  email: yup.string().required('Required').email('Invalid email'),
  password: yup.string().required('Required').min(8, 'Password must be at
least 8 characters')
});

const SignIn = () => {
  const { openToast, startProgress, doneProgress } = useUIActions();
  const { signInEmailPassword, isLoading, isSuccess } =
useSignInEmailPassword();

  const { control, handleSubmit } = useForm<FormData>({
    resolver: yupResolver(schema)
  });

  const router = useRouter();

  const onSubmit = async (formData: FormData) => {
    try {
      startProgress();

      const { isError, error } = await signInEmailPassword(formData.email,
formData.password);

      if (isError) {
        openToast({ type: 'ERROR', error: error?.message });
      }
    } catch (error: any) {
      openToast({ type: 'ERROR', error });
```

```
    } finally {
      doneProgress();
    }
  };

  const { onPress } = useKeyDown({ onSubmit: handleSubmit(onSubmit) });

  if (isSuccess) {
    router.push('/');
    return null;
  }

  return (
    <Wrapper>
      <LinkWrapper>
        <Link preset='large' href='/'>
          Go to home page
        </Link>
      </LinkWrapper>

      <Form onKeyDown={onPress}>
        <FormInput label='Email Address' name='email' control={control} />
        <FormInput name='password' type='password' label='Password'
control={control} />
      </Form>

      <Button isLoading={isLoading} onClick={handleSubmit(onSubmit)}>
        Login
      </Button>

      <Link href='/sign-up'>Don’t Have An Account? Sign Up Here</Link>
    </Wrapper>
  );
};

export default SignIn;
```

Реєстрація

```
import * as React from 'react';
// libs
import * as yup from 'yup';
import { yupResolver } from '@hookform/resolvers/yup';
// components
import { Link } from '@md-shared/components/link';
import { FormInput } from '@md-shared/components/form';
import { Button } from '@md-shared/components/button/main';
// hooks
import { useRouter } from 'next/router';
import { useForm } from 'react-hook-form';
import { useSignUpEmailPassword } from '@nhost/nextjs';
import { useKeyDown } from '@md-shared/hooks/use-key-down';
import { useUIActions } from '@md-managers/use-ui-actions';
// view
import { Form, Wrapper } from './views';

// types
interface FormData {
  fullName: string;
  email: string;
```

```
  secret: string;
  firstName: string;
  lastName: string;
  password: string;
}

type SignInFields = 'firstName' | 'lastName' | 'email' | 'password';

// constants
const fieldsToValidate = ['firstName', 'lastName', 'email', 'password'];

// validation
const schema = yup.object().shape({
  firstName: yup.string().required('Required').min(2, 'Name is too short'),
  lastName: yup.string().required('Required').min(2, 'Name is too short'),
  email: yup.string().required('Required').email('Invalid email'),
  secret: yup.string(),
  password: yup
    .string()
    .required('Required')
    .matches(
      /^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*#?&])[A-Za-z\d@$!%*#?&]{8,}$/,
      'Your password must have a minimum of 8 characters including a number,
uppercase and lowercase and at least one of these @$!%*#?& characters.'
    )
});

const SignUp = () => {
  const { openToast, startProgress, doneProgress } = useUIActions();

  const router = useRouter();

  const { control, handleSubmit, setError } = useForm<FormData>({
    resolver: yupResolver(schema)
  });

  const { signUpEmailPassword, isLoading } = useSignUpEmailPassword();

  const onSubmit = async (formData: FormData) => {
    if (formData.secret === process.env.NEXT_PUBLIC_COACH_SECRET) {
      try {
        startProgress();

        const { isError, error } = await signUpEmailPassword(formData.email,
formData.password, {
          displayName: `${formData.firstName} ${formData.lastName}`.trim(),
          redirectTo: process.env.NEXT_PUBLIC_FRONTEND_URL,
          metadata: {
            firstName: formData.firstName,
            lastName: formData.lastName
          }
        });

        if (!isError) {
          await router.push({
            pathname: '/verify-email',
            query: {
              email: formData?.email
            }
          });
        }
```

```
        if (isError) {
          fieldsToValidate.forEach((name) =>
            setError(name as SignInFields, {
              message: error?.message
            })
          );

          openToast({ type: 'ERROR', error: error?.message });
        }
      } catch (error: any) {
        openToast({ type: 'ERROR', error });
      } finally {
        doneProgress();
      }
    } else {
      openToast({ type: 'ERROR', error: 'Need correct secret!' });
    }
  };

  const { onPress } = useKeyDown({ onSubmit: handleSubmit(onSubmit) });

  return (
    <Wrapper>
      <Link preset='large' href='/'>
        Go to home page
      </Link>

      <Form onKeyDown={onPress}>
        <FormInput label='First Name' control={control} name='firstName' />
        <FormInput label='Last Name' control={control} name='lastName' />
        <FormInput label='Email Address' control={control} name='email' />
        <FormInput label='Secret' control={control} name='secret' />
        <FormInput name='password' type='password' label='Password'
control={control} />
      </Form>
      <Button onClick={handleSubmit(onSubmit)} isLoading={isLoading}>
        Sign up
      </Button>

      <Link href='/sign-in'>Already Have An Account? Sign In Here</Link>
    </Wrapper>
  );
};

export default SignUp;
```

Валідація e-mail

```
import React from 'react';
// hooks
import { useRouter } from 'next/router';
import { useUIActions } from '@md-managers/use-ui-actions';
import { useSendVerificationEmail } from '@nhost/react';
// components
import { Button } from '@md-shared/components/button/main';
// views
import { SubTitle, Wrapper } from '@md-modules/auth/verify-email/view';
import { Link } from '@md-shared/components/link';

const VerifyEmail = () => {
```

```
  const { push, query } = useRouter();
  const { sendEmail } = useSendVerificationEmail();
  const { openToast, startProgress, doneProgress } = useUIActions();

  const sendEmailTimeOut = 5;

  // states
  const [isDisabled, setIsDisabled] = React.useState(false);
  const [timer, setTimer] = React.useState(sendEmailTimeOut);

  // method's
  const onClick = async () => {
    startProgress();

    setTimer(120);
    setIsDisabled(true);

    try {
      if (query?.email) {
        const result = await sendEmail(query.email, { redirectTo:
process.env.NEXT_PUBLIC_FRONTEND_URL });

        if (result.isSent) {
          openToast({ type: 'SUCCESS' });
        }

        if (result.error) {
          openToast({ type: 'ERROR', error: result.error.message });
        }
      } else {
        push('/sign-in');
      }
    } catch (error) {
      openToast({ type: 'ERROR', error: error as Error });

      // eslint-disable-next-line no-console
      console.log('[Error while verify email request]: ', (error as
Error).message);
    } finally {
      doneProgress();
    }
  };

  // Effect's
  React.useEffect(() => {
    if (isDisabled) {
      const intervalID = setInterval(() => {
        if (timer > 0) {
          setTimer((prevState) => prevState - 1);

          return;
        }

        setIsDisabled(false);
      }, 1000);

      return () => clearInterval(intervalID);
    }
  }, [timer, isDisabled]);

  React.useEffect(() => {
    if (timer < 1) {
```

```
      onClick();

      return;
    }

    if (!query.email) {
      push('/');
    }

    setIsDisabled(true);
  }, []);

  return (
    <Wrapper>
      <SubTitle>
        A verification letter was sent to the <strong>{query?.email}</strong>
mail.
      </SubTitle>
      <SubTitle>Please confirm verification.</SubTitle>

      <Button onClick={onClick} disabled={isDisabled}>
        Retry
      </Button>

      {isDisabled && (
        <SubTitle>
          You can try again after <strong>{timer}</strong> seconds
        </SubTitle>
      )}

      <Link href='/sign-up' as='/sign-in'>
        Back to Sign up
      </Link>
    </Wrapper>
  );
};

export default VerifyEmail;
```

Список тренерів

```
import React from 'react';
// hooks
import { useRouter } from 'next/router';
// contexts
import { CoachesAPIContext } from '@md-modules/coaches/layers/api';
// components
import Card from '@md-modules/shared/components/card';
import { ContentLoader } from '@md-shared/components/content-loader';
// views
import { Wrapper } from '@md-modules/coaches/layers/presentation/views';

const CoachesPresentation = () => {
  const { push } = useRouter();
  const { coaches, isLoading, error } = React.useContext(CoachesAPIContext);

  const onClickCard = (id: string) => push(`/programs?id=${id}`);

  return (
    <Wrapper>
```

```
      <ContentLoader isLoading={isLoading} clientError={error}
isEmpty={!coaches.length}>
        {coaches?.map((user) => (
          <Card
            id={user.id}
            key={user.id}
            email={user.email}
            onClick={onClickCard}
            name={user.displayName}
            avatar={user.avatarUrl}
          />
        ))}
      </ContentLoader>
    </Wrapper>
  );
};

export default CoachesPresentation;
```

API для списку тренерів

```
import * as React from 'react';
// libs
import { useQuery } from '@apollo/client';
// utils
import * as U from '@md-utils';
// types
import { ClientError } from '@md-utils/errors/custom';
import { Coach, GetCoachesResponse } from '@md-shared/queries/user/types';
// queries
import { GET_USERS } from '@md-shared/queries/user';

interface Context {
  coaches: Coach[];
  isLoading: boolean;
  error?: ClientError<string>;
}

const CoachesAPIContext = React.createContext<Context>({
  coaches: [],
  isLoading: false
});

const CoachesAPIContextProvider: React.FC = ({ children }) => {
  const { data, loading, error } = useQuery<GetCoachesResponse>(GET_USERS);

  return (
    <CoachesAPIContext.Provider
      value={{
        isLoading: loading,
        coaches: data?.users || [],
        error: error && U.errors.parseAndCreateClientError(error)
      }}
    >
      {children}
    </CoachesAPIContext.Provider>
  );
};

export { CoachesAPIContextProvider, CoachesAPIContext };
```

Створення програми

```
import React from 'react';
// hooks
import { useRouter } from 'next/router';
// components
import { Button } from '@md-shared/components/button/main';
import { ContentLoader } from '@md-shared/components/content-loader';
import { Compose } from '@md-modules/create-program/components/compose';
// views
import { Wrapper } from '@md-modules/coaches/layers/presentation/views';

const CreateProgramPresentation = () => {
  const { back } = useRouter();

  return (
    <Wrapper>
      <Button preset='link' onClick={back}>
        Back
      </Button>

      <ContentLoader>
        <Compose />
      </ContentLoader>
    </Wrapper>
  );
};

export default CreateProgramPresentation;
```

API для створення програми

```
import * as React from 'react';
// hooks
import { useRouter } from 'next/router';
import { useNhostClient } from '@nhost/react';
// libs
import { ApolloError, useLazyQuery, useMutation } from '@apollo/client';
// types
import {
  CreateProgramVariables,
  CreateProgramResponse,
  GetProgramResponse,
  GetProgramVariables,
  Program
} from '@md-shared/queries/programs/types';
// queries
import { CREATE_PROGRAM, GET_PROGRAM } from '@md-shared/queries/programs';
// utils
import * as U from '@md-utils';
import { ClientError, ClientSuccess, clientSuccess, ResponseEither } from
'@md-utils/errors/custom';

interface Context {
  program?: Program;
  isLoading: boolean;
  isUploadingVideo: boolean;
  error?: ClientError<string>;
  setIsUploadingVideo: (isUploadingVideo: boolean) => void;
  createProgram: (
```

```
    object: CreateProgramVariables
  ) => Promise<ResponseEither<CreateProgramResponse | undefined | null,
string>>;
}

const CreateProgramAPIContext = React.createContext<Context>({
  isLoading: false,
  isUploadingVideo: false,
  setIsUploadingVideo: () => {},
  createProgram: () => Promise.resolve({} as
ClientSuccess<CreateProgramResponse>)
});

const CreateProgramAPIContextProvider: React.FC = ({ children }) => {
  const { query } = useRouter();
  const nhost = useNhostClient();
  const [isUploadingVideo, setIsUploadingVideo] = React.useState(false);
  const [createProgramMutation] = useMutation<CreateProgramResponse,
CreateProgramVariables>(CREATE_PROGRAM);

  const [getProgram, { data, loading, error }] =
useLazyQuery<GetProgramResponse, GetProgramVariables>(GET_PROGRAM);

  const createProgram = async (object: CreateProgramVariables) => {
    try {
      const result = await createProgramMutation({ variables: { object:
object.object } });

      return clientSuccess(result.data);
    } catch (error) {
      return U.errors.parseAndCreateClientError(error as ApolloError);
    }
  };

  // Effect's
  React.useEffect(() => {
    nhost.storage.setAdminSecret(process.env.NEXT_PUBLIC_ADMIN_SECRET);
  }, []);

  React.useEffect(() => {
    if (query.id?.[0]) {
      getProgram({
        variables: { id: query.id?.[0] as string }
      });
    }
  }, [getProgram, query.id]);

  return (
    <CreateProgramAPIContext.Provider
      value={{
        createProgram,
        isUploadingVideo,
        isLoading: loading,
        setIsUploadingVideo,
        program: data?.program,
        error: error && U.errors.parseAndCreateClientError(error)
      }}
    >
      {children}
    </CreateProgramAPIContext.Provider>
  );
};
```

```
export { CreateProgramAPIContextProvider, CreateProgramAPIContext };
```

API для відображення профілю

```
import * as React from 'react';
// types
import { ApolloError, useMutation, useQuery } from '@apollo/client';
import {
  GetProgramsResponse,
  GetProgramsVariables,
  Program,
  DeleteProgramResponse,
  DeleteProgramVariables
} from '@md-shared/queries/programs/types';
/// queries
import { GET_PROGRAMS, DELETE_PROGRAM } from '@md-shared/queries/programs';
//  hooks
import { useNhostClient, useUserData } from '@nhost/react';
// utils
import * as U from '@md-utils';
import { ClientError, ClientSuccess, clientSuccess, ResponseEither } from
'@md-utils/errors/custom';

interface Context {
  programs: Program[];
  isProgramsLoading: boolean;
  programsError?: ClientError<string>;
  refetchProgram: () => Promise<ResponseEither<Program[], string>>;
  deleteProgram: (
    data: DeleteProgramVariables
  ) => Promise<ResponseEither<DeleteProgramResponse | undefined | null,
string>>;
}

const SettingsAPIContext = React.createContext<Context>({
  programs: [],
  isProgramsLoading: false,
  refetchProgram: () => Promise.resolve({} as ClientSuccess<Program[]>),
  deleteProgram: () => Promise.resolve({} as
ClientSuccess<DeleteProgramResponse>)
});

const SettingsAPIContextProvider: React.FC = ({ children }) => {
  const nhost = useNhostClient();
  const userId = useUserData();

  const { data: programs, loading: isProgramsLoading, error: programsError,
refetch: refetchProgramM } = useQuery<
    GetProgramsResponse,
    GetProgramsVariables
  >(GET_PROGRAMS, {
    variables: { userId: userId?.id as string }
  });

  const [deleteProgramMutation] = useMutation<DeleteProgramResponse,
DeleteProgramVariables>(DELETE_PROGRAM);

  const refetchProgram = async () => {
    try {
```

```
      const result = await refetchProgramM();

      return clientSuccess(result?.data.programs);
    } catch (error) {
      return U.errors.parseAndCreateClientError(error as ApolloError);
    }
  };

  const deleteProgram = async ({ id }: DeleteProgramVariables) => {
    try {
      const result = await deleteProgramMutation({ variables: { id } });

      return clientSuccess(result.data);
    } catch (error) {
      return U.errors.parseAndCreateClientError(error as ApolloError);
    }
  };

  React.useEffect(() => {
    nhost.storage.setAdminSecret(process.env.NEXT_PUBLIC_ADMIN_SECRET);
  }, []);

  return (
    <SettingsAPIContext.Provider
      value={{
        deleteProgram,
        refetchProgram,
        isProgramsLoading,
        programs: programs?.programs || [],
        programsError: programsError &&
U.errors.parseAndCreateClientError(programsError)
      }}
    >
      {children}
    </SettingsAPIContext.Provider>
  );
};

export { SettingsAPIContextProvider, SettingsAPIContext };
```

Відображення профілю

```
import React from 'react';
// hooks
import { useRouter } from 'next/router';
import { useUIActions } from '@md-managers/use-ui-actions';
// context
import { UserAPIContext } from '@md-shared/providers/user-provider';
import { SettingsAPIContext } from '@md-modules/profile/layers/api';
// components
import Card from '@md-shared/components/card';
import { Button } from '@md-shared/components/button/main';
import NameForm from '@md-modules/profile/components/name-form';
import { AddButton } from '@md-modules/profile/components/add-button';
import { ContentLoader } from '@md-shared/components/content-loader';
import PasswordForm from '@md-modules/profile/components/password-form';
import Loader from '@md-shared/components/content-loader/components/loader';
// views
import {
  MainWrapper,
```

```
  OWrapper,
  IWrapper,
  DataWrapper,
  CardsWrapper,
  Wrapper,
  ButtonWrapper,
  ScreenTitle
} from './views';

const SettingsPresentation = () => {
  const { push } = useRouter();
  const { openToast, doneProgress, startProgress } = useUIActions();

  const { programs, isProgramsLoading, programsError, deleteProgram,
refetchProgram } = React.useContext(
    SettingsAPIContext
  );
  const { user, userError, isUserLoading } =
React.useContext(UserAPIContext);

  const onCreateProgram = () => push('/create-program');

  const isAvatarUploading = false;

  const onClickEdit = (id: string) => push(`/create-program/${id}`);

  const onClickDelete = async (id: string) => {
    startProgress();

    try {
      const data = await deleteProgram({ id });

      if (data) {
        await refetchProgram();

        await openToast({ type: 'SUCCESS' });
      }
    } catch (error: any) {
      openToast({ type: 'ERROR', error: error });
    } finally {
      doneProgress();
    }
  };

  return (
    <Wrapper>
      <ButtonWrapper>
        <Button onClick={onCreateProgram} preset='link'>
          + Create Program
        </Button>
      </ButtonWrapper>

      <ContentLoader isLoading={isUserLoading || isProgramsLoading}
clientError={userError || programsError}>
        <MainWrapper>
          <OWrapper>
            <IWrapper isLoading={isAvatarUploading}>
              {isAvatarUploading ? <Loader /> : <img src={user?.avatarUrl}
/>}
            </IWrapper>
            <AddButton />
          </OWrapper>
```

```
            <DataWrapper>
              <NameForm />
              <PasswordForm />
            </DataWrapper>
          </MainWrapper>
        </ContentLoader>

        <CardsWrapper>
          <ScreenTitle>Your Programs</ScreenTitle>

          <ContentLoader isEmpty={!programs.length && !isProgramsLoading}>
            {programs?.map((program) => (
              <Card
                id={program.id}
                key={program.id}
                name={program.name}
                onClickEdit={onClickEdit}
                onClickDelete={onClickDelete}
              />
            ))}
          </ContentLoader>
        </CardsWrapper>
      </Wrapper>
    );
};

export default SettingsPresentation;
```