МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЦЕНТР ЗАОЧНОЇ, ДИСТАНЦІЙНОЇ ТА ВЕЧІРНЬОЇ ФОРМ НАВЧАННЯ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «Мобільний додаток моніторингу результатів тренувань спортсмена»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТдн-84жт Стрельченко Андрій Петрович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою _____«___» _____2022 р.

Науковий керівник _____ к.т.н., доц., Чибіряк Я. І.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент_____
(підпис)

Суми-2022

Сумський державний університет
Центр заочної та дистанційної форми навчання
Кафедра інформаційних технологій
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. кафедри ІТ

_____
В. В. Шендрик
«06» жовтня 2021 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Стрельченко Андрій Петрович

1 Тема роботи *Мобільний додаток моніторингу результатів тренувань спортсмена*

керівник роботи *Чибіряк Яна Іваівна, к.т.н., доцент*,

затверджені наказом по університету від «____» ____202_ р. №_____

2 Строк подання студентом роботи «20» червня 2022 р.

3 Вхідні дані до роботи Результати аналізу предметної області, вимоги державних стандартів

____

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Аналіз предметної області, актуальність досліджуваної задачі, аналіз сучасних тенденцій, постановка задачі, моделювання

предметної області, засоби розробки системи, опис програмної реалізації додатку, висновки, використані джерела, додаток А, додаток Б, додаток В.

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Ілюстрації з інтерфейсом, кодом та функціоналом програми

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання 6 жовтня 2021

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	06.12.21	
1.1	Актуальність досліджуваної задачі	17.12.21	
1.2	Аналіз сучасних тенденцій	22.12.21	
1.3	Постановка задачі	26.12.21	
2	Моделювання предметної області	02.01.22	
2.1	UML діаграми	12.01.22	
3	Засоби розробки системи	18.01.22	
4	Опис програмної реалізації вебдодатку	20.01.22	
4.1	Реалізація клієнтської частини	05.04.22	
4.2	Реалізація серверної частини	15.05.22	

Студент _____ Стрельченко А. П.
(підпис)

Керівник роботи _____ к.т.н., доц. Чибіряк Я. І.
(підпис)

ЗМІСТ

ВСТУП **4**

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ **6**

1.1 Актуальність досліджуваної задачі 6
1.2. Огляд існуючих програмних продуктів для вирішення поставлених задач 8
1.3. Перелік вимог до програмного додатку 10

РОЗДІЛ 2 ПОСТАНОВКА ЗАДАЧІ **12**

2.1 Мета та задачі 12
2.2 Програмні засоби реалізації 12
2.3 Планування робіт 13
2.3.1 Ідентифікація мети IT-проекту методом SMART 13
2.3.2 Планування змісту структури робіт IT-проекту (WBS) 14
2.3.3 Розробка матриці відповідальності 16

РОЗДІЛ 3 ПРОЕКТУВАННЯ ЗАСТОСУНКУ **18**

3.1 Моделювання використання застосунку 18
3.2 Моделювання застосунку «Мобільний додаток моніторингу результатів тренувань спортсмена» 19
*3.2.1 Моделювання застосунку «Мобільний додаток моніторингу результатів тренувань спортсмена» ..... **Error! Bookmark not defined.***
3.2.1 Функціональне моделювання мобільного додатку в IDEF0. 22
3.2.2 Діаграма 1-го рівня декомпозиції 23
3.3. Модель проектування. 24

4. РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ **25**

4.1 Розробка візуальної частини 26
3.4.2 Розробка логіки для віджетів 29
3.4.3 Розробка контролерів для взаємодії з API 30

ВИСНОВКИ **35**

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ **36**

ДОДАТКИ **37**

Додаток А Технічне завдання 37
Додаток Б Програмний код 41

ВСТУП

У зв’язку зі швидкою інтеграцією техніки у всі аспекти людського життя, навіть звичні раніше речі, такі як записування списку покупок чи залишання нагадувань в вигляді папірців на робочому місці, знаходять себе в електронній, цифровій формі. Звичайні записники перейшли в програми, які синхронізуються через хмарні сховища та надають доступ людям, які знаходяться в різних місцях. Всі ці програми зараз можна тримати в своєму телефоні, так як їх розвинутість стоїть поряд з настільними комп’ютерами і вирішує всі ті ж задачі, але в портативному вигляді. Саме тому найактуальнішою програмою я вважаю мобільний застосунок.

Для кожної людини важливо тримати своє тіло в тонусі. Для цього потрібно займатися спортом, а для фіксації своїх досягнень та візуального відображення прогресу зручно було б записувати його десь та аналізувати після кожного тренування результати. Звісно добре мати якийсь блокнот щоб не тримати всі результати в голові та записувати їх під час, або після тренувань. Ще було б непогано мати таймер для відрахування часу до наступного підходу/вправи. Ну і ідеальним варіантом це складати графіки після тренування та за певні періоди, щоб бачити зміни у власних показниках. Саме ці цілі я переслідував при постановці задачі яку мав би виконувати додаток.

Актуальність роботи можна розглянути в консенсусі важливості спорту в житті людини. Так як програма напряму мотивує займатися спортом та вести здоровий спосіб життя. Її застосування полягає в збереженні показників та відображенню змін протягом певного періоду спортивного життя людини.

Мета роботи – створення мобільного застосунку для збереження даних про тренування. Для досягнення мети було поставлено наступні завдання:

1. Перевірка існуючих додатків
2. Порівняння їх між собою
3. Відбір кращих реалізацій

4. Збір необхідної інформації для функціонування програми
5. Підготування середовища розробки

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність досліджуваної задачі

«Рух – це життя» – так звучить одне з найпопулярніших тверджень сучасного світу, котре прикрашає банери та різну рекламну продукцію спортивних центрів та секцій. Типова річ, яку знає кожен з нас, проте мало хто розуміє суть цього поняття та в чому саме перевага рухливості.

Чому ж справді так важливо рухатись, щоб добре почуватись? В чому користь активності та занять спортом? Яким чином це впливає на стан вашого здоров'я?

По-перше, регулярно займаючись спортом це сприятимете покращенню обмінних процесів, які відповідають за добре самопочуття та хороший стан здоров'я. Крім цього, тренування добре впливають на травну систему, допомагають швидше виводити шкідливі речовини з організму. Крім цього, ті, хто регулярно займається спортом, рідко мають відчуття нудоти.

Заняття спортом «піднімають» імунітет. Це захищає організм від респіраторних хвороб та інших захворювань, які атакують організм зі зміною температур та сезонів. Таким чином можна впевнено сказати, що спорт – це панацея від хвороб, яка не має побічних ефектів і здійснює позитивний вплив на організм.

Доведено, спорт – одне з найбільш ефективних знеболюючих. Перш за все, тренування. Вони допомагають м'язам розслабитися і зміцнити суглоби, що може знімати болі та спазми різного характеру. Крім цього, фізичні навантаження сильно сприяють виробці ендорфіну – природному гормону щастя, що ефективно бореться з болем різного типу.

Хроші тренування – це здоровий сон. Це зовсім і не дивно, адже після великого фізичного навантаженя наше тіло, в особливості великі м'язи потребують відпочинку та відновлення. Люди, котрі регулярно займаються

спортом, сплять краще, піддаються стресу менше та рідше або взагалі не страждають безсонням.

Хочете покращити настрій? Просто завітайте до спортзалу або ж проведіть більш активний день на свіжому повітрі. Фізична активність усуває депресію, допомагаючи прибрати з голови погані думки та підводить до ефективних рішень щоденних задач, доромагає з появою нових ідей для роботи, навчання.

Зниження ваги – одна з найбільш популярних переваг спорту. Але варто пам’ятати - досягнення позитивного ефекту можливе тільки при умові поєднання фізичних навантажень з правильним сном та правильним харчуванням.

З тим що спорт важлива складова нашого життя ми ознайомились. Чому ж важливо фіксувати свої досягнення в спорті?

Фіксація результатів надасть вам характеристику ваших занять та досягнень. Також записавши всі моменти під час тренування ви зможете побачити прогрес та зміни в організмі. Звісно записувати все не вийде, та і не потрібно. Вистачить зафіксувати кількість підходів, вагу з якою працював, кількість повторів за підхід.

Записуючи дані про кожне тренування ви зможете побачити свої кращі та гірші сторони в спорті та вдосконалювати те що вам хочеться. Також ви зможете через деякий період перевірити наскільки змінилися ваші показники порівняно з аналогічним періодом попереднього року та скорегувати тренування.

1.2. Огляд існуючих програмних продуктів для вирішення поставлених задач

FitNote24 – частинно схожа по функціоналу програма, але має деякі недоліки

- Відсутність моніторингу відпочинку
- Відсутність таймеру з відліком до початку наступного підходу
- Має всередині заблокований функціонал, який працює по підписці
- Не має можливасті переключення kg→lbs та навпаки

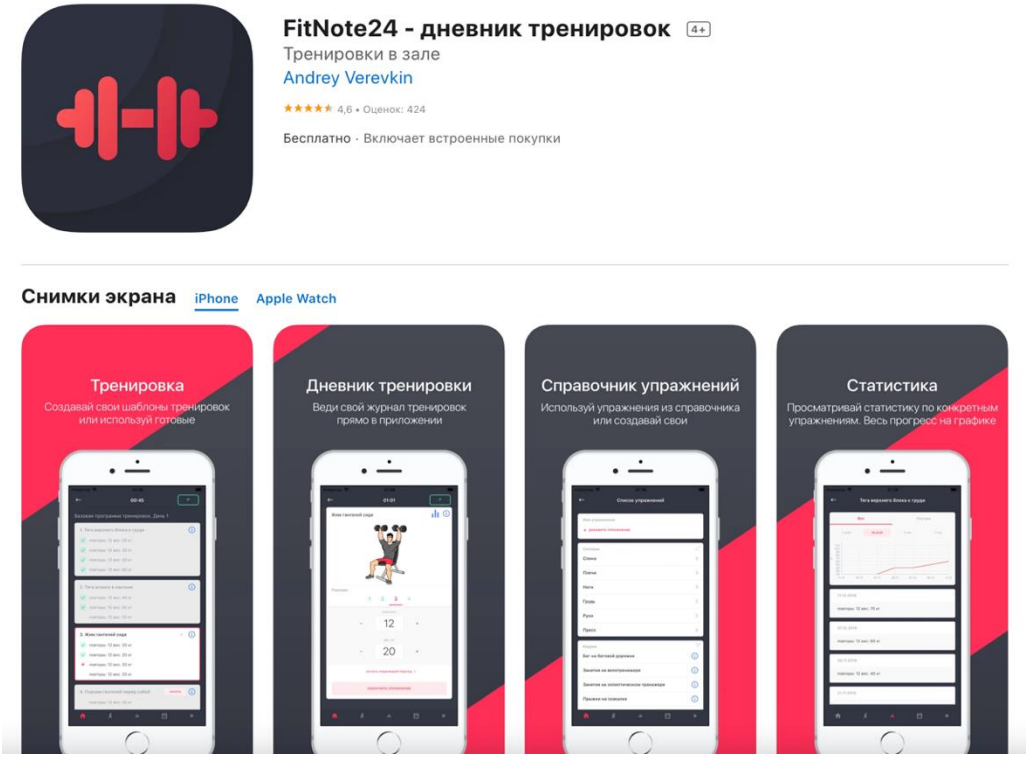

Рисунок 1.1 Зображення з AppStore. Мобільний додаток FitNote24 – дневник тренировок

Fitsession: Workout Journal

Недоліки:

- Відсутність кастомізації кольорів
- Відсутність перемикання ваги з kg→lbs і навпаки
- Відсутність створення тренувань

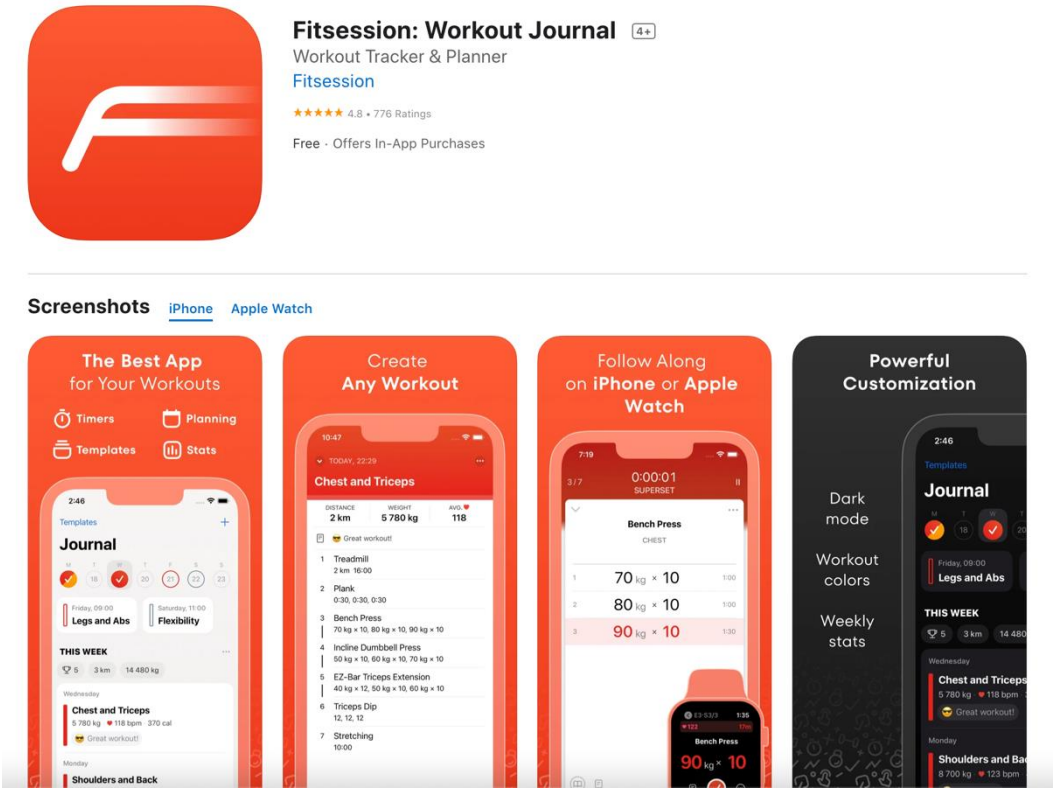

Рисунок 1.2 Зображення з AppStore. Мобільний додаток Fitsession: Workout Journal

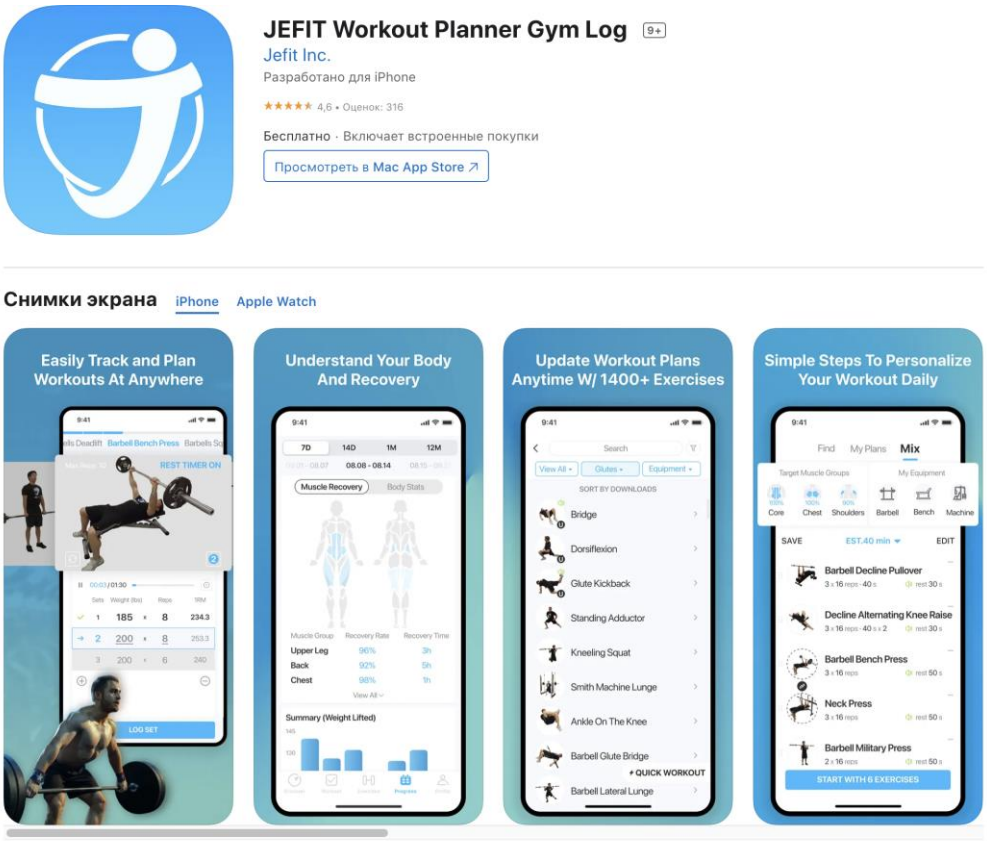

Рисунок 1.3 Зображення з AppStore. Мобільний додаток JEFIT Workout Planner Gym Log

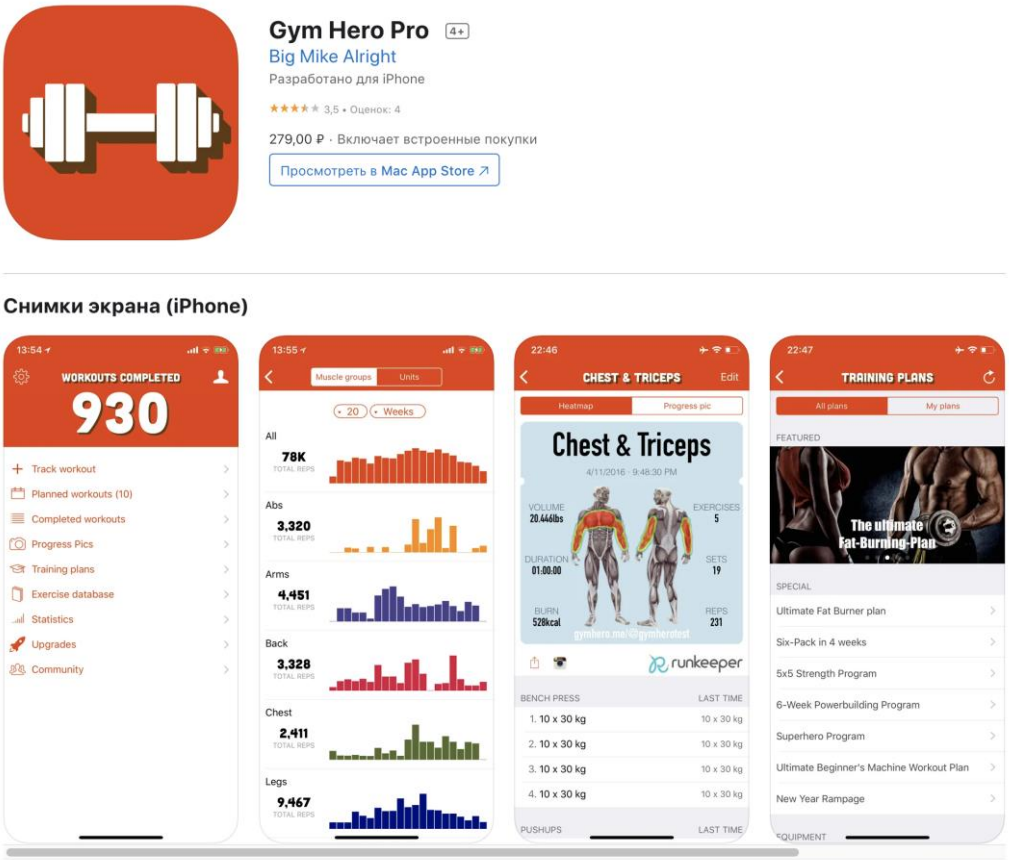

Рисунок 1.4 Зображення з AppStore. Мобільний додаток Gym Hero Pro

Таблиця 1.5 Порівняльна таблиця існуючих програмних продуктів

	FitNote24	Workout Journal	JEFIT Workout Planner Gym Log	Gym Hero Pro
Персональний аккаунт	+	+	+	+
Кастомізація інтерфейсу	+	-	-	-
Перемикання одиниць вимірювання	-	-	-	+
Створення власних вправ	-	+	-	-
Створення тренувань	+	-	+	+
Відсутність платного контенту	-	-	-	-

1.3. Перелік вимог до програмного додатку

Вимоги до програмного додатку – вимоги які ставляться перед додатком, та які він має задовольняти. Програміст має реалізувати програму (додаток) таким чином щоб задовольнити всі вимоги та реалізувати потрібний функціонал.

Для більшої конкретизації вимог до програмного додатку, використаємо порівняльну таблицю з попереднього розділу, потрібно виправити всі проблеми та зберегти необхідний функціонал.

Можна зафіксувати наступні проблеми, та поставити їх вирішення як ціль для додатку:

1. Збереження даних про тренування
2. Фіксування даних в процесі тренування
3. Збереження в певному форматі для послідуючого коректного відображення
4. Фіксування часу всього тренування
5. Запис вправ
6. Створення тренувань на базі вправ
7. Запис кількості підходів та повторів
8. Фіксування моментів відпочинку
9. Відображення прогресу

РОЗДІЛ 2
ПОСТАНОВКА ЗАДАЧІ

2.1 Мета та задачі

Основною метою створення додатку є фіксація результатів тренувань для подальшого їх перегляду та аналізу.

Задачі:

1. Огляд предметної області
2. Вибір середовища розробки
3. Підбір методів розробки
4. Налаштування середовища розробки
5. Початкові налаштування додатку
6. Розробка локального сховища даних
7. Розробка контролерів для підключення до арі
8. Розробка інтерфейсу додатку
9. Підключення логіки

Технічне завдання на розробку IT- продукту наводиться у додатку А

2.2 Програмні засоби реалізації

Для реалізації застосунку було вибрано мову Dart та фреймворк Flutter.

Мову Dart розробляє компанія Google. Дана мова позиціонується як мова структурованого програмування для Веб. Розробники думали, що в майбутньому Dart може стати заміною JavaScript, котрий відчуває значні проблеми і різних аспектах від наявних проблем з розширюваністю, продуктивністю і розробкою складних застосунків. Мова отримала схожий на Java синтаксис..

Flutter — комплект засобів для розробки та фреймворк з відкритим вихідним кодом для створення мобільних додатків під Android та iOS, веб-додатків, а також настільних додатків під Windows, macOS та Linux з

використанням мови програмування Dart, розроблений та розвивається корпорацією Google.

Через обмеження динамічного коду в App Store, під iOS Flutter використовує AOT-компіляцію. Також використовується можливість платформи Dart - «гаряче перезавантаження», коли зміна коду застосовується одразу у працюючому додатку без необхідності його перезавантаження.

2.3 Планування робіт

2.3.1 Ідентифікація мети IT-проекту методом SMART

Проблема: незручність використання та зберігання даних в паперовому вигляді а також відсутність певного структурування та відображення результатів

Ціль: створити додаток для записування результатів тренування спортсмена

Мета проекту: розробити додаток для запису результатів тренувань спортсмена, які будуть вноситися в додаток та переглядатися через деякий час в зручному вигляді та з можливістю їх редагування

Продукт проекту: мобільний додаток для моніторингу тренувань спортсмена

Результат: оптимізація зберігання результатів тренувань. Можливість швидко переглядати та редагувати дані.

2.3.2 Планування змісту структури робіт IT-проекту (WBS)

Матриця відповідальності суттєво полегшує управління проектом. Її будують виходячи з попередньо розробленої WBS-структури та OBS-структури. Вона застосовується для здійснення керівних впливів шляхом введення принципу відповідальності (Principle Responsibility).

Правильно складена матриця відповідальності покаже хто керує виконанням певної ланки зі структури більш низького рівня. За виконання якогось із WBS-елементів може відповідати тільки одна людина. Але хід відразу декількох складових робіт може бути підконтрольний єдиного особі

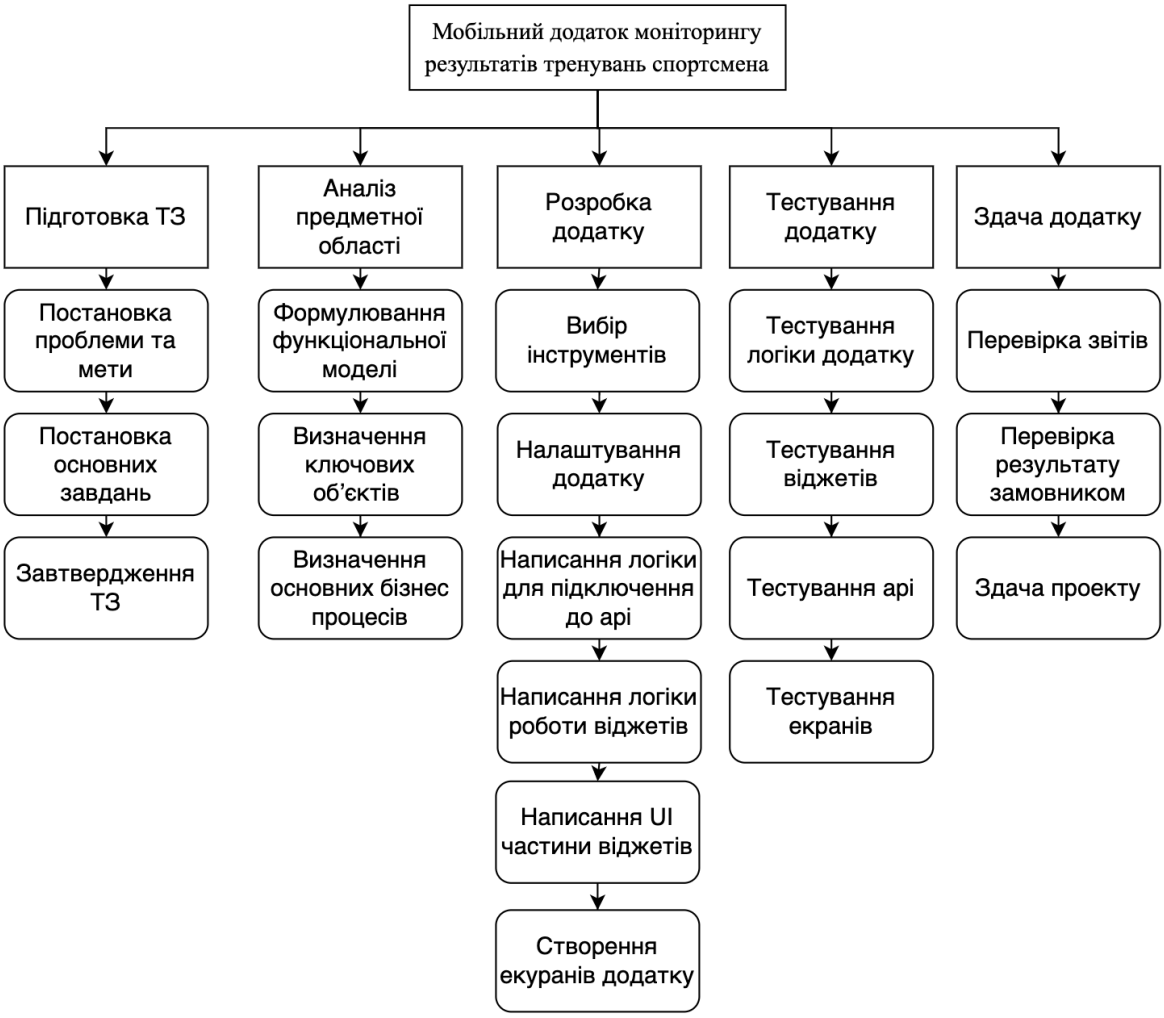

Рисунок 2.1 WBS–діаграма розробки мобільного додатку моніторингу результатів тренування спортсмена

WBS-діаграма відповідає за відображення задач, їх послідовності та вносить ясність в групування завдань. В нашому випадку бачимо що діаграма складається з 5-ти стовпців, тобто показує що є 5 основних етапів розробки застосунку.

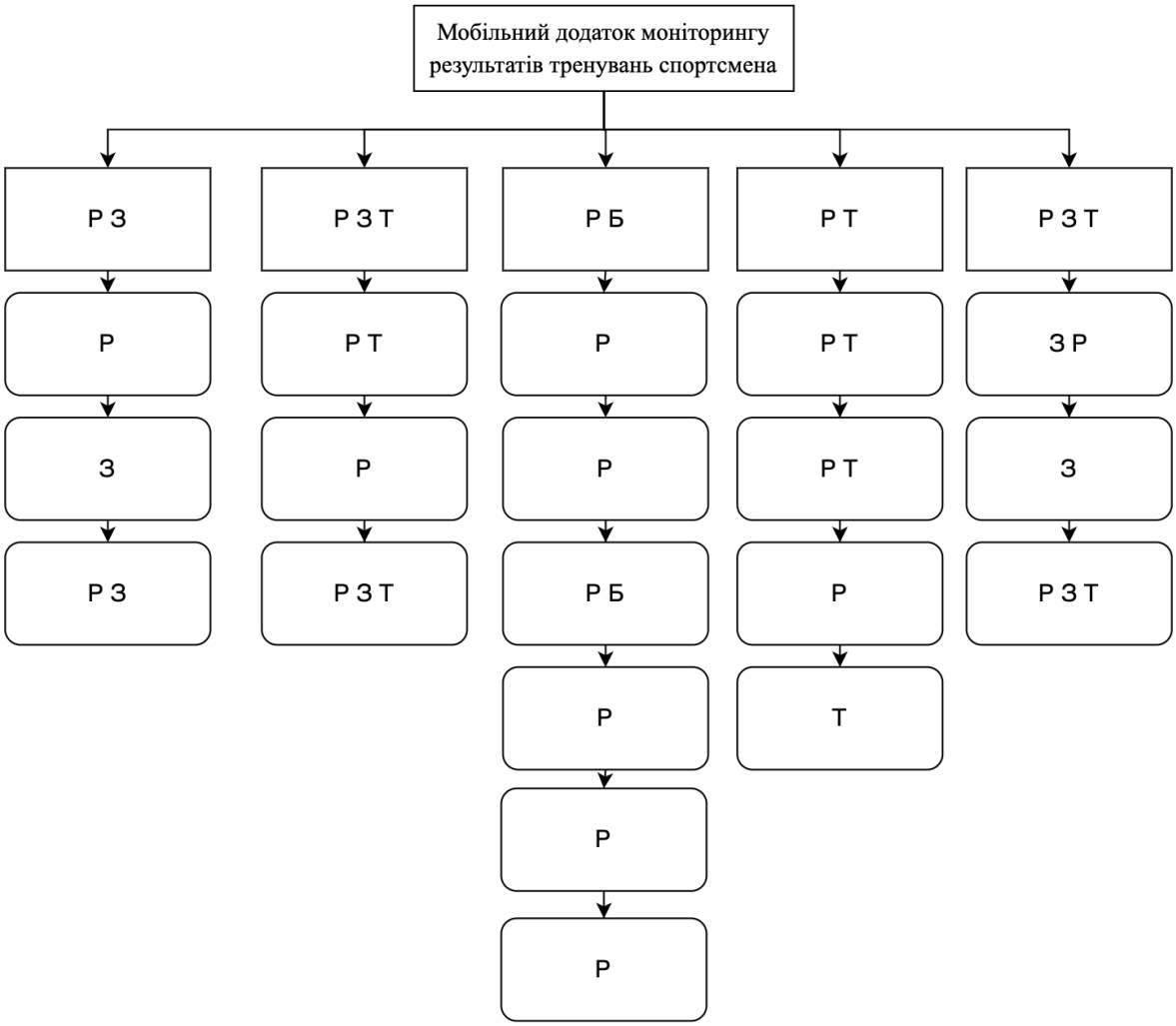

Рисунок 2.2 OBS-діаграма розробки мобільного додатку моніторингу тренувань спортсмена

Дана діаграма показує хто відповідає за який пункт та вносить ясність в те які особи будуть задіяні в який момент розробки проекту. По діаграмі видно що розробник приймає найбільшу участь в створенні додатку, тестування проходить додаток на передостанньому етапі.

2.3.3 Розробка матриці відповідальності

Великої зручності в контроль над розробкою додатку вносить матриця відповідальності. Вона будується а основі двох попередніх діаграм та відображає в більш зручному вигляді

Таблиця 2.3 Матриця відповідальності

	Розробник Стрельченко А. П.	Замовник Чибіряк Я. І.	Тестувальник	Бекендер
1. Підготовка ТЗ	***	***	***	***
1.1. Постановка проблеми та мети				
1.2. Постановка основних завдань				
1.3. Затвердження ТЗ				
2. Аналіз предметнох області	***	***	***	***
2.1. Формулювання функціональної моделі				
2.2. Визначення ключових об’єктів				
2.3. Визначення основних бізнес процесів				
3. Розробка додатку	***	***	***	***
3.1. Вибір інструментів				
3.2. Налаштування додатку				
3.3. Написання логіки для підключення до арі				
3.4. Написання логіки роботи віджетів				
3.5. Написання UI частини віджетів				
3.6. Створення екранів додатку				
4. Тестуваня додатку	***	***	***	***
4.1. Тестування логіки додатку				
4.2. Тестування віджетів				
4.3. Тестування арі				
4.4. Тестування екранів				
5. Здача додатку	***	***	***	***
5.1. Перевірка звітів				
5.2. Перевірка результату замовником				
5.3. Здача проекту				

За даними з матриці відповідальності видно хто і за яку частину продукту відповідає, тож маємо чіткі межі відповідальності та можна більш правтильно розрахувати час та ризики.

2.3.4 Планування ризиків проекту

Під час проведення аналізу було виявлено такі ризики

- Відмова замовника від дизайну
- Втрата даних проекту
- Повільна робота застосунку
- Хвороба розробника

Таблиця 2.4 Ймовірність виникнення ризиків

	ймовірність	R1	R2	R3	R4
>60%	ймовірні	+	+		+
40%-60%	можливі				
<40%	малоймовірні			+	

Таблиця 2.5 Величина втрат

	втрати	R1	R2	R3	R4
> 7 днів	максимальні	+	+		
3 - 7 днів	середні			+	+
< 3 днів	мінімальні				

РОЗДІЛ 3
ПРОЕКТУВАННЯ ЗАСТОСУНКУ

3.1 Моделювання використання застосунку

Для складання діаграми сценарію використання застосунку, потрібно визначити весь функціонал додатку. Виділити його в блоки. Визначити які користувачі мають доступ до цих блоків. Вияснити чи може бути якесь виключення з частинним доступом до певної частини додатку.

Аналіз вимог та задач дозволяє запропонувати наступну діграму використання застосунку (рис.3.1).

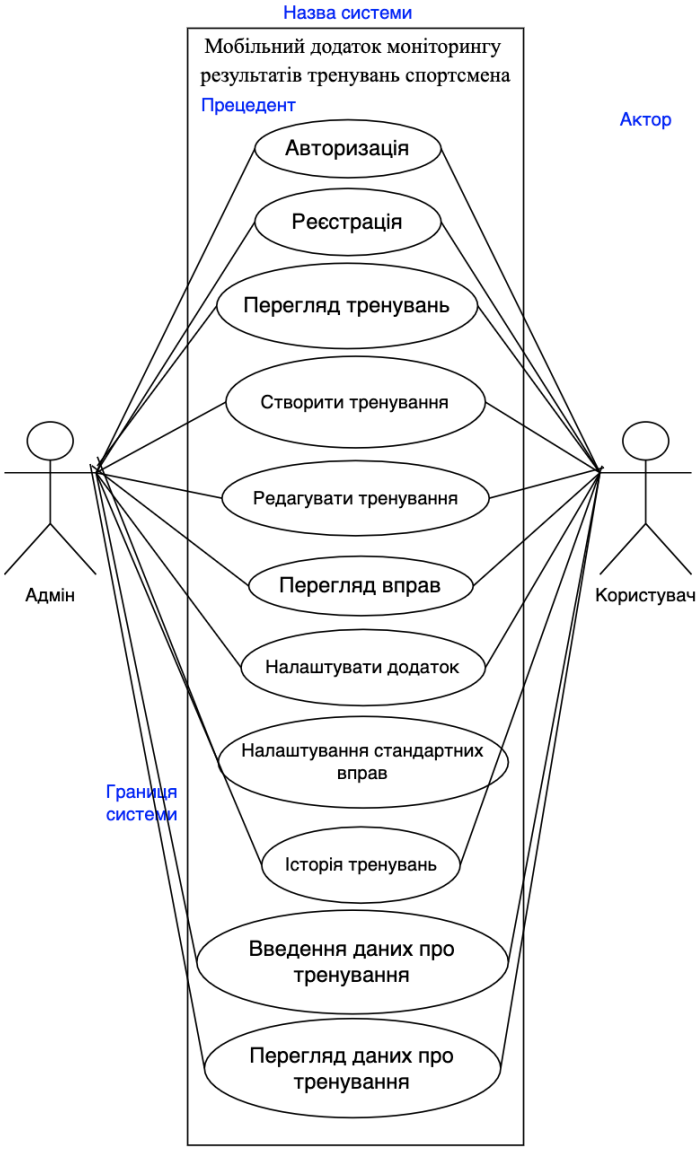

Рисунок 3.1. Діагама сценарію використання застосунку

3.2 Моделювання застосунку «Мобільний додаток моніторингу результатів тренувань спортсмена»

Основною проблемою більшості застосунків є відсутність певного функціоналу або ж проблеми з його реалізацією. До таких проблем можна віднести відсутність створювати власні тренування або ж відсутність їх модифікувати під себе, відсутність таймеру для перерв, відсутність можливості редагувати тренування яку відбулося в минулому. Створення власних тренувань буде додано для зручності використання програми. Таймер буде доданий між підходами щоб користувач міг бачити скільки часу йому залишилося відпочивати та підготовуватися до наступної вправи/підходу

Функції які використовуються в проекті:

Main – головна функція в якій виконується весь код

Hive.init – функція для ініціалізації сховища даних

Hive.initFlutter – функція для ініціалізаї роботи сховища даних з Flutter

Get.put – функція для покладення даних в сховище

Get.lazyPut – асинхронна функція для покладення даних в сховище

createDio – функція для створення клієнту для асинхронних запитів

addInterceptors – функція для додавання «перехватчиків» для асинхронних запитів

getHTTP | postHTTP | putHTTP | deleteHTTP – функції для маніпулювання з бекендом за допомогою ajax запитів

getToken – функція для отримання токену для зв’язку з бекендом

set Token – функція для записування токену в сховище

trainingsService.getList – запит на список тренувань

authServise.login – запит на логін юзера в систему

authServise.logout – запит на вилогінювання юзера з системи

authServise.registration – запит на реєстрацію юзера

build – головний метод віджетів, рендерить UI

locale – створення нових мовних пакетів

getLocaleFromLanguage – отримання назви локалі по назві мови

Get.updateLocale – оновлення локалі

buildTheme – створення теми

appBarTheme – створення теми для appBar

Color – створення кольору

onInit – метод який спрацьовує при ініціалізації класу

createTrainingTemplate – функція для створення шаблонк тренування

createTraining – функція для створення тренування

updateTraining – функція для оновлення тренування

startTraining – функція для початку тренування

endTraining – функція для завершення тренування

startExercise – функція для початку вправи

endExercise – функція для завершення вправи

validate – функція для валідації

Box.add – метод для додавання даних в бокс

Controler.clear – метод для очищення стану контролера

getInitialData – функція для отримання початкового стану контролера

updateList – функція для оновлення списку тренувань

deleteItem – функція для видалення тренування

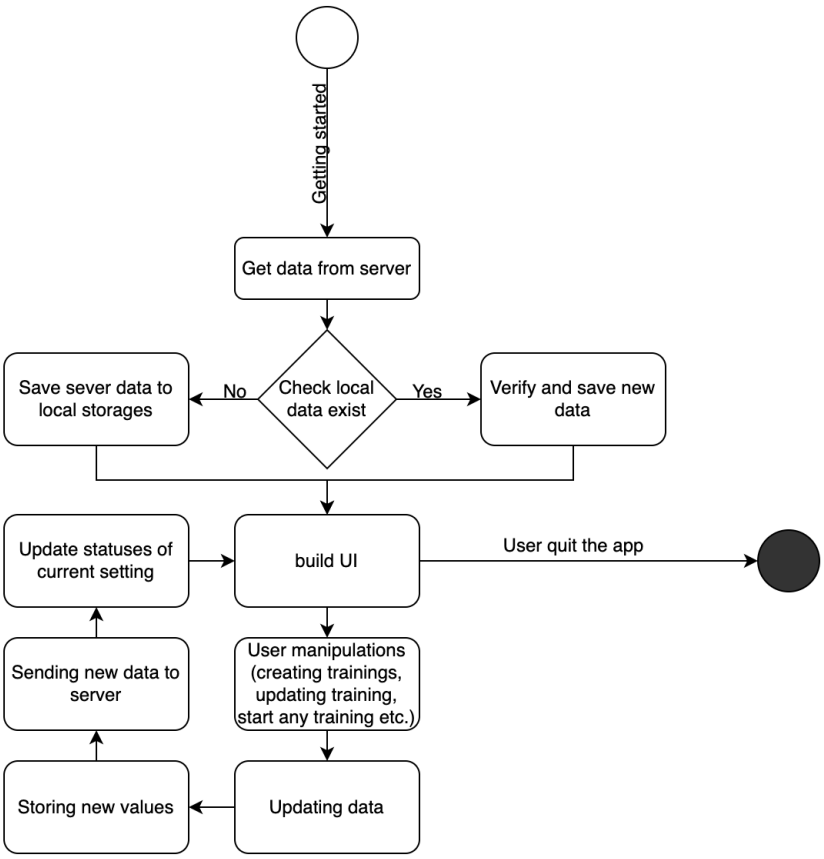

Рисунок 3.2. Діаграма станів мобільного додатку моніторингу результатів тренувань спортсмена

По даним зі схеми (рис 3.2) програма починає свою роботу з отримання даних з серверу. Після отримання даних потрібно їх перевірити з локальними даними і якщо є зміни то використати найбільш актуальну версію. Після того як дані отримані та оновлені, їх можна дістати з сховища даних та відмалювати в візуальній частині сайту. Коли юзер робить будь-які маніпуляції з додатком ми оновлюємо локально дані. Локальне оновлення даних тягне за собою глобальне оновлення даних (якщо користувач зберіг дані). Після глобального оновлення даних вони відправляються на сервер, де обробляються та оновлюються в базі даних. Після чого ми отримуємо результат оновлення даних, знову записуємо їх в глобальне сховище. Оновлюємо візуальну частину та чекаємо нових маніпуляцій.

3.2.1 Функціональне моделювання мобільного додатку в IDEF0.

IDEF0 - це методологіяпризначення для побудування ієрархічних діаграм.

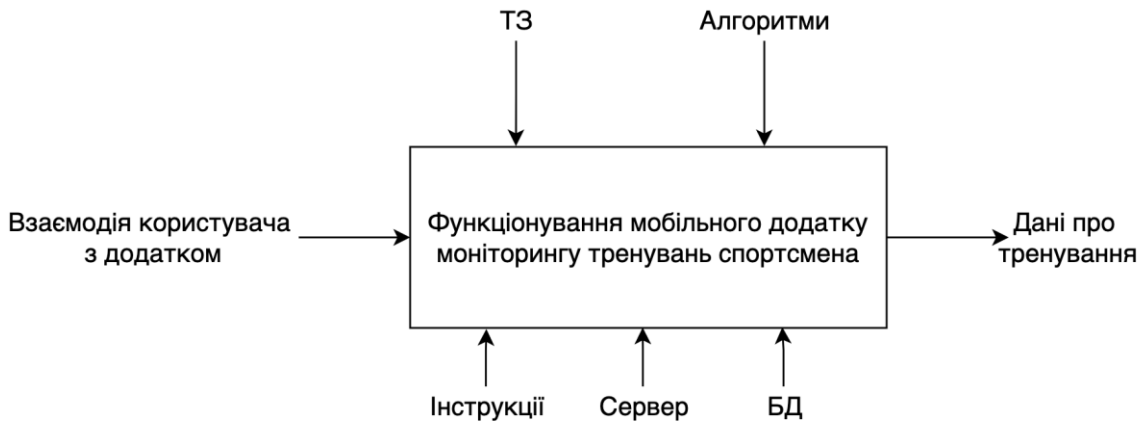

Рисунок 3.3 Діаграма IDEF0

В діашрамі відображено як взаємодія користувача прихзводить до отримання даних про певне тренування. За допомогою алгоритмів та іструкцій написаних програмістом по певному ТЗ, дані отримані сервером з бази даних відправляються на клієнт та відображаються користувачеві.

3.2.2 Діаграма 1-го рівня декомпозиції

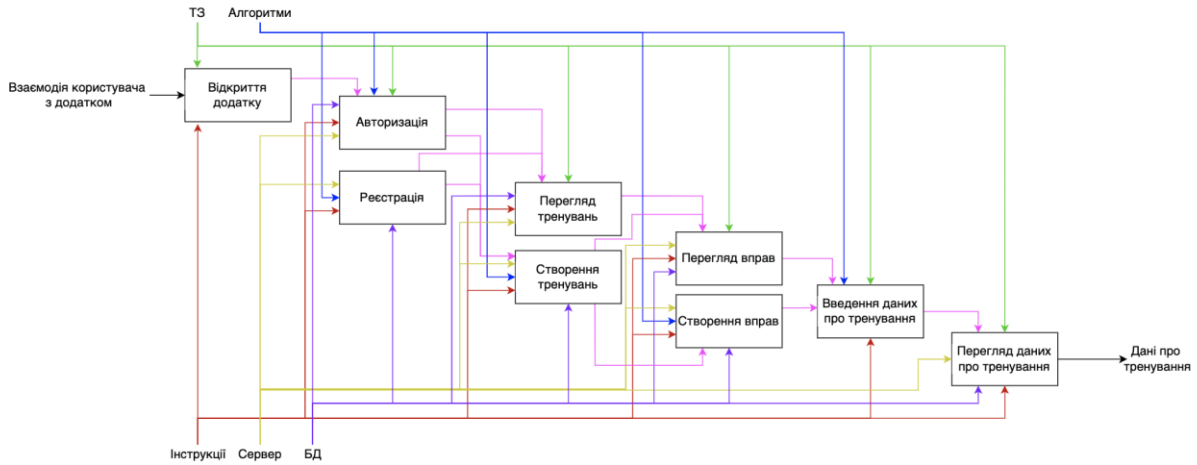

Рисунок 3.4 Діаграма 1-го рівня декомпозиції

В даній діаграмі більш детально розглянуто як працює логіка всередині додатку. Показано що і звідки отримується для відображення чи зміни даних в додатку. Як ми бачимо після відкриття додатку користувачеві потрібно обов’язково авторизуватися або зареєструватися щоб прожовжити роботу з додатком. Якщо ж користувач був до цього авторизований то додаток буде відразу в стані після авторизації.

3.3. Модель проектування.

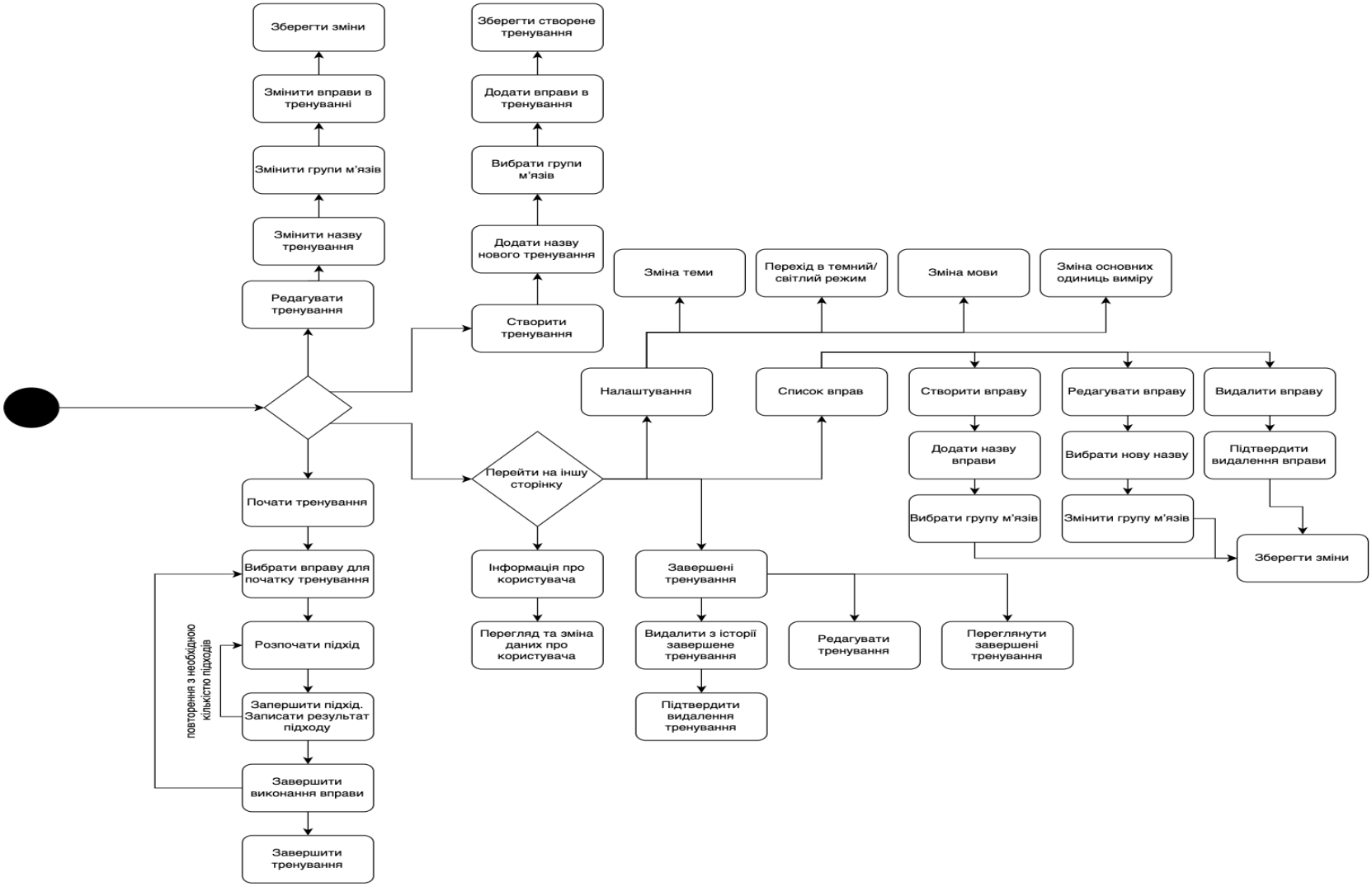

Рисунок 3.5 Діаграми діяльності мобільного додатку моніторингу результатів тренувань спортсмена

В діаграмі діяльності мобільного додатку (рис 3.3) показано яким чином відбувається вся взаємодія в додатку. Прослідкувавши всі ланцюжки можна побачити що є екрани на яких є варіанти куди послідувати далі, а є екрани в яких тільки один шлях.

4. РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

4.1 Розробка візуальної частини

Для створення екрану будуть використовуватися набір стандартних віджетів Flutter та деякі модифікації їх. Для контролю даних в текстових полях будемо використовувати класи-контролери, які будуть контролювати та оновлювати локальний стейт віджету. Для відображення контенту потрібно переписати метод build, для цього використаємо модифікатор @override (рис. 4.1)

```
@override
Widget build(BuildContext context) {
  return Scaffold(
      body: Form(
    key: _formKey,
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
        InputStyled(
          controller: nameController,
          validator: (value) {
            if (value == null || value.isEmpty) {
              return 'Будь ласка введіть ім\'я';
            }
            return null;
          },
          label: 'Ім\'я',
        ), // InputStyled
        InputStyled(
          controller: emailController,
          validator: (value) {
            if (value == null || value.isEmpty) {
              return 'Будь ласка введіть пошту';
            }
            return null;
          },
          label: 'Пошта',
        ), // InputStyled
        InputStyled(
          obscureText: true,
          controller: passwordController,
          validator: (value) {
            if (value == null || value.isEmpty) {
              return 'Будь ласка введіть пароль';
            }
            return null;
          },
          label: 'Пароль',
        ), // InputStyled
        ButtonStyled(label: 'Зареєструватись', onPressed: register)
      ], // <Widget>[]
    ), // Column
  )); // Form // Scaffold
}
```

Рисунок 4.1 Створення екрану Register

Після того як щапустимо приведений на рис. 4.1 код отримаємо сторінку, яка відображена на рис 4.2. На рисунку видно 3 поля для введення тексту. Одне з яких є зашифроване та використовується для введення паролю. Кожне з полів має свою валідпцію на помилки та відображає їх (рис 4.3)

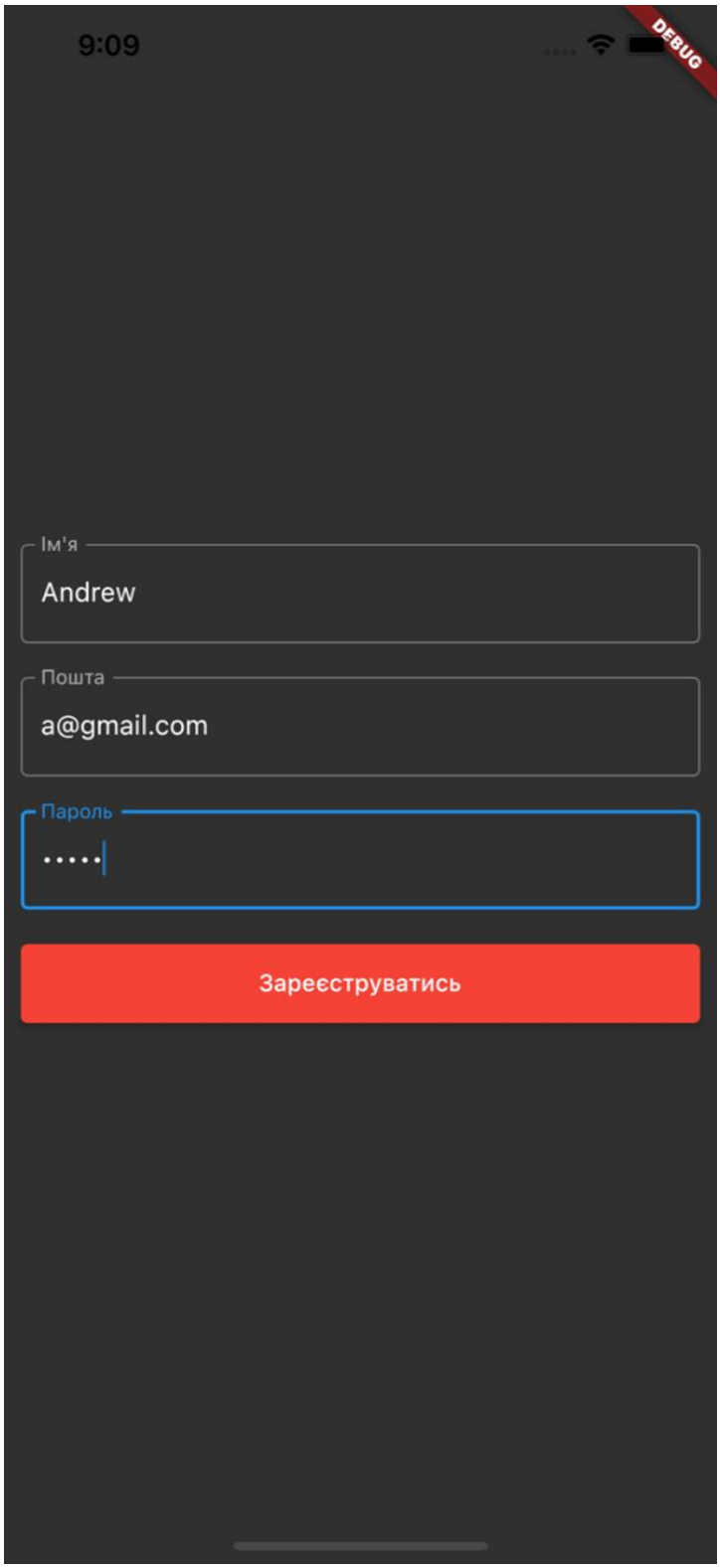

Рисунок 4.2 Вигляд екрана Ragister

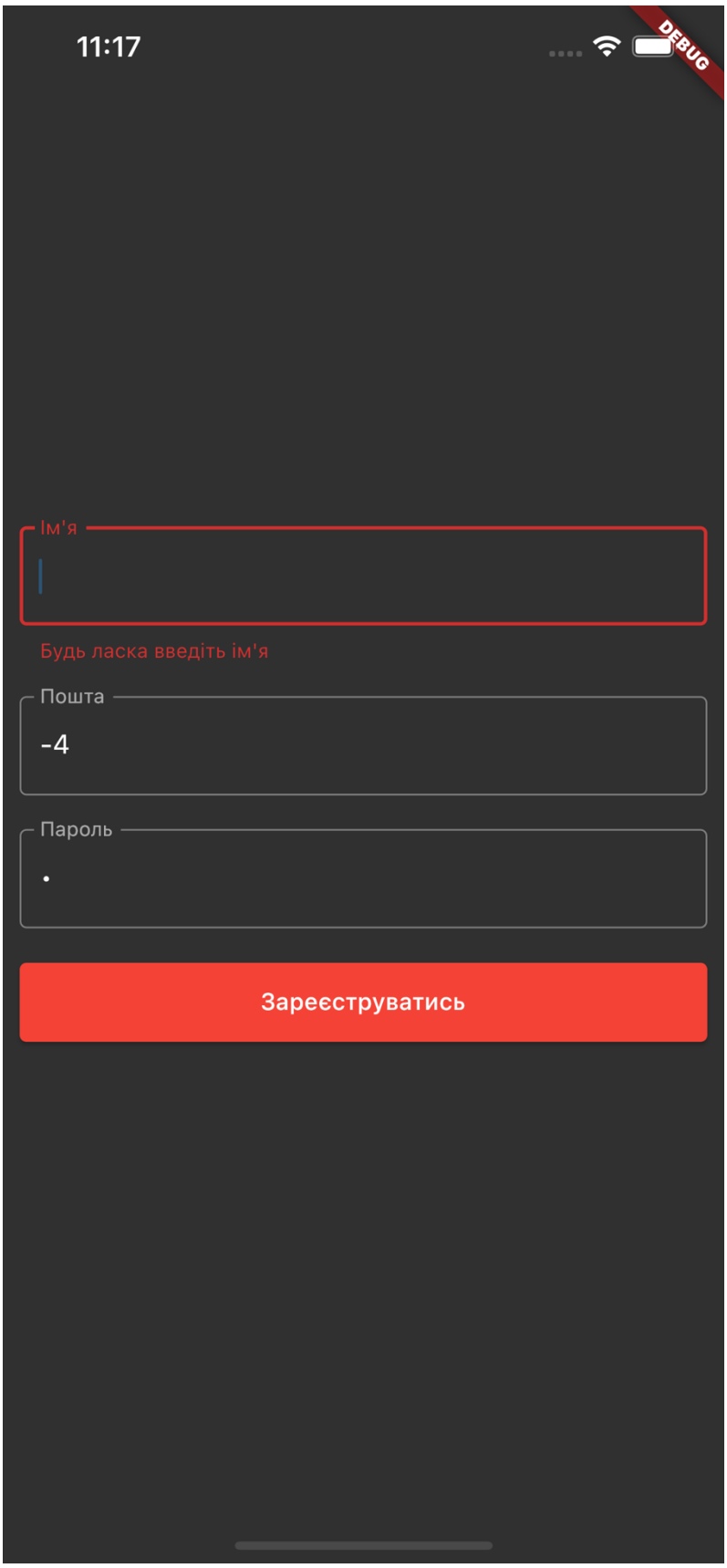

Рисунок 4.3 Відображення помилки вводу

4.2 Розробка логіки для віджетів

Логіка буде двох видів. Перший – логіка відображення візуальої частини (рис. 4.4) та другий – логіка отримання та відправлення даних.

```
14    final passwordController = TextEditingController();
15    final emailController = TextEditingController();
16    final nameController = TextEditingController();
17    TokenController tokenController =
18        Get.find<TokenController>(tag: "tokenController");
19
20    void register() async {
21      if (_formKey.currentState!.validate()) {
22        Future<User> user = registerUser(
23                  name: nameController.text.toString(),
24                  password: passwordController.text.toString(),
25                  email: emailController.text.toString())
26            .catchError((error) {
27          Get.snackbar('Failed Registration', error.toString());
28        });
29
30        user.then((value) async {
31          // await storage.write(key: 'token', value: value.token);
32          tokenController.setToken(value.token);
33          Get.offAllNamed('/home');
34          Get.snackbar('Success Registered', 'Welcome to GYM Manager');
35        });
36      }
37    }
38
39    final _formKey = GlobalKey<FormState>();
```

Рисунок 4.4 Логіка екрану Register

Для редагування полів використовуються контролери (класи які керують даними поля). TextEditingController – клас який керує текстовими полями (рис 4.4). Для доступу до даних форми використовується formKey, це унікальний ключ форми, який не повинен повторюватися, щоб ввід і валідація були правильними. Функція validate валідує дані форми, і якщо дані не валідні то вертає false, якщо ж валідні true (рис 4.4)

4.3 Розробка контролерів для взаємодії з api

Контролери для взаєодії з api (рис 4.5) використовуватимуться для отримання даних та відправлення їх в локальне сховище даних або напряму в віджет.

```
class ExerciseService {
  static var client = http.Client();

  static Future<Exercise> getWorkoutExerciseCompletedItem(
      String completedId) async {
    final box = GetStorage();

    var response = await client.get(
      Uri.parse(
          '$baseUrl/workout/training/exercise/completed?completedId=$completedId'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
    );

    if (response.statusCode == 200) {
      var exercise = jsonDecode(response.body);
      return Exercise.fromJson(exercise);
    } else {
      throw Exception(jsonDecode(response.body));
    }
  }

  static Future<List<Exercise>> getCompletedExerciseList(
      String? exerciseId) async {
    final box = GetStorage();

    var response = await client.get(
      Uri.parse('$baseUrl/workout/training/exercise/completed/list'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
    );

    if (response.statusCode == 200) {
      var exercise = jsonDecode(response.body);
      var parsedData =
          List.from(exercise).map((exercise) => Exercise.fromJson(exercise));
      return exerciseId == null
          ? parsedData.toList()
          : parsedData
              .where((element) => element.exerciseId == exerciseId)
              .toList();
    } else {
      throw Exception(jsonDecode(response.body));
    }
  }
}
```

Рисунок 4.5 Приклад контролера для взаємодії з api

Для взаємодії з арі потрібен клієнт, який створюється за допомогою http.Client. Отриманий клієнт використовується в запитах, в нього додається токен (унікальний ідентифікатор користувача в системі) та необхідні частини запиту. Після отримання результату виконання запиту потрібно перевірити чи запит пройшов успішно (без помилок) чи був помилковим. Для цього перевіряється response.statusCode. 200 статус каже що все пройшло успішно і ми можемо використовувати дані з цього результату (рис 4.5)

4.4 Розробка локального сховища даних

Так як за сховище даниз буде відповідати GetX, нам потрібно буде тільки реалізувати логіку передачі даних в GetX та отримання їх з нього.

```
class UserController extends GetxController {
  var isLoading = true.obs;
  var user =
      User(completedExercises: [], trainings: [], trainingTemplates: []).obs;

  TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");

  @override
  void onInit() {
    getUserData();
    super.onInit();
  }

  void getUserData() async {
    try {
      isLoading(true);
      var user = await UserService.getUser();
      this.user.value = user;
    } catch (err) {
      tokenController.clearToken();
      Get.offAllNamed('/');
      Get.snackbar('Failed to get user data', err.toString());
    } finally {
      isLoading(false);
    }
  }
}

```

Рисунок 4.6 Реалізація UserControler

Прикладом контролера для зберігання даних в локальному середовищі може бути UserControler (рис 4.6). В ньому створюються локальні змінні isLoading та user, які є observable (ті за якими можна слідкувати в процесі їх зміни). Дані для змінної user отримуються з серверу при ініціалізації додатку. Якщо дані не будуть отримані то користувачу буде висвітлена помилка з отримання даних користувача та перенаправлено на сторінку авторизації.

4.5 Підключення логіки до візуальної частини

Підключення логіки до візуальної частини відбувається напряму в створенні візуальних елементів. В них передаються параметри, які мають певну логіку та відповідають за певну частину логіки всього віджета (рис. 4.7)

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Form(
      key: _formKey,
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          InputStyled(
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Будь ласка введіть пошту';
              }
              return null;
            },
            controller: emailController,
            label: 'Пошта',
          ), // InputStyled
          InputStyled(
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Будь ласка введіть пароль';
              }
              return null;
            },
            obscureText: true,
            controller: passwordController,
            label: "Пароль",
          ), // InputStyled
          TextButton(
            style: TextButton.styleFrom(primary: Colors.blue),
            child: Text(
              'Відновити пароль',
            ), // Text
            onPressed: () {
              Get.toNamed('/forgot-password');
            },
          ), // TextButton
          ButtonStyled(label: 'Ввійти', onPressed: login),
          Container(
              child: Row(
            children: <Widget>[
              Text('Не маєш акаунту?'),
              TextButton(
                style: TextButton.styleFrom(primary: Colors.blue),
                child: Text(
                  'Зареєструватись',
                ), // Text
                onPressed: () {
                  Get.toNamed('/register');
```

Рисунок 4.7 Відображення підключення логіки до візуальних елементів додатку

Як ми бачимо з прикладу на рис 4.7, такі елементи логіки як validator, controller передаються параметрами та налаштовують віджет певним чином. Наприклад параметр на стрічці 64 (obscureText: true) відповідає за те чи приховувати введений текст, що корисно для таких полів як пароль.

ВИСНОВКИ

У кваліфікаційному проекті було розроблено мобільний додаток для фіксації досягнень спортсмена.

В процесі аналізу було виявлено основні позитивні та негативні моменти конкурентних застосунків та знайдено способи їх вирішення.

Для розробки було використано такі технології як Flutter, Hive, GetX. Всі представлені технології написані на мові Dart, яка є високорівневою мовою програмування з високим рівнем абстракції коду. Основна причина вибору цієї мови була цікавість до нових мов програмування які розвивають великі компанії. Серед вибору була ще мова Swift, але вона не настільки зручно реалізує модульний (частинний) підхід до створення елементів (віджетів) та їх відображення. До того ж Flutter, як на мою думку, більш перспективна технологія, якщо брати до уваги другу версію і старше.

Основними складностями стали 2 моменти. Перший – створення форми new exercise (нова вправа). Потрібно було зв’язати вправи з типами та тренуваннями. А також додати до бази користувача. Другий – створення екрану підходів та логіка для нього, адже під час тренування потрібно записувати дані в саме тренування, зберегти його початок, відслідковувати який саме це підхід та якої саме вправи. Підходи потрібно було відображати в окремому вікні для комфортного перегляду та не нагромаджувати основний екран, з який і йде майже вся взаємодія під час тренування. Потрібно зробити його максимально зручним для користувача та максимально функціональним водночас. Результат більш ніж задовольнив вимоги до цього екрану та виправдав всі сподівання під час його написання.

Загалом під час написання додатку я поліпшив свої знання про об’єктно-орієнтовані мови програмування. Оновив знання про класи, їх методи та властивості. Познайомився з новими фремворками для розробки.

Написана програма задовольняє поставлені перед нею цілі та відповідає критеріям, які було необхідно виконати.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Документація Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://flutter.dev/
2. Документаці по GetX[Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/packages/get
3. Стаття про переваги GetX[Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/568488/
4. Стаття «The ultimate guide to GetX state management in Flutter»[Електронний ресурс] – Режим доступу до ресурсу: https://blog.logrocket.com/ultimate-guide-getx-state-management-flutter/
5. Стаття «Основы Flutter для начинающих (Часть I)»[Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/560008/
6. Відеоматеріал «Уроки Flutter и Dart с нуля / #1 – Разработка мобильных приложений для начинающих»[Електронний ресурс] – Режим доступу до ресурсу: https://www.youtube.com/watch?v=YPRaA6KhyXc&t=579s
7. Довідник до e-learning системи навчання[Електронний ресурс] – Режим доступу до ресурсу: https://www.td.org/talent-development-glossary-terms/what-is-e-learning
8. «The official package repository for Dart and Flutter apps.» [Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/
9. Документація до платформи Node.Js[Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/
10. Документація по Hive [Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/packages/hive

ДОДАТКИ

Додаток А
Технічне завдання

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Технічне завдання
на створення програмного продукту

«Мобільний додаток моніторингу результатів тренувань спортсмена»

2022

1. Призначення й мета створення веб-застосунку.

1.1 Призначення веб-застосунку

Додаток повинен надавати спортсмену фіксувати результати своїх тренувань та переглядати їх. Давати можливість створювати власні тренування та вправи. Тренування складається з багатьох вправ. Вправа складається з декількох підходів між якими мають бути перерви

1.2 Мета створення веб-застосунку

Метою даного дослідження є створення застосунку для фіксації досягнень та моніторингу досягнень спортсмена.

1.3 Цільова аудиторія

В цільову аудиторію входять всі люди хто займається спортом та хоче систематизувати свої зайняття, покращити результати

1. Вимоги до застосунку.

2.1 Вимоги до структури й функціонування застосунку.

Застосунок повинен бути представлений у вигляді мобільного додатку. Має бути можливість створити вправу або використовувати вправи за-умовчуванням. Вправи повинні відноситися до певної групи м’язів, на які буде йти навантаження. Повинна бути можливість створити тренування, яке

складатиметься з деякої кількості вправ, яка має динамічно змінюватися в залежності від вибору користувача. Кожне тренування / вправа повинне мати назву. Після початку тренування повинна бути можливість вибору певної вправи (вибирає користувач) для її початку. Під час знаходження в одній вправі користувач повинен мати можливість редагувати дані які він ввів та видаляти помилкові підходи. Після завершення кожного підходу, результати мають записуватися користувачем та зберігатися в системі. Після завершення вправи, вона помічається як виконана і не може бути почата знову. Після завершення всіх вправ тренування вважається виконаним та відправляється в історію. В історії має бути можливість перегляду тренування та редагування помилкових показників. Для відокремлення користувачів має використовуватися аккаунти. Аккаунт користувач створює самостійно або за допомогою сторонніх сервісів авторизації. Для створення та редагування початкового набору вправ має бути роль адміністратора, аккаунт якого матиме розширений функціонал.

2.2 Структура застосунку

2.2.1 Екран входу в аакаунт

Має надавати можливість створити аккаунт або ввійти в нього.

2.2.2 Головний екран (екран тренувань)

Має включати в себе дані про шаблони тренувань за якими можна почати нове тренування. Містить кнопки для початку тренування та для створення нового тренування.

2.2.3 Екран вправ

Має відображати всі вправи. Повинен містити кнопки для створення нових вправ, для редагування створених вправ.

2.2.4 Екран завершених тренувань

Має відображати завершені тренування. Повинен мати можливість редагувати тренування та видалити його з історії. Тренування зберігаються з датою та часом їх початку для простішої ідентифікації різних тренувань.

2.2.5 Екран налаштувань

Включає в себе зміну мови, перемикання в light/dark mode, зміну основних одиниць виміру.

2.3 Вимоги до видів забезпечення.

2.3.1 Вимоги до інформаційного забезпечення

Для реалізації додатку використовуватиметься мова Dart.

Реалізація додатку відбувається за допомогою наступних технологій:

- Flutter
- GetX
- Intl
- Hive

Додаток Б
Частина програмного коду

Main.dart

```
import 'package:flutter/material.dart';
import 'package:get_storage/get_storage.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
import 'package:gym_manager/src/page/forgot-passoword.dart';
import 'package:gym_manager/src/page/home.dart';
import 'package:gym_manager/src/page/register.dart';
import 'src/page/login.dart';
import 'package:get/get.dart';

final RouteObserver<ModalRoute<void>> routeObserver =
    RouteObserver<ModalRoute<void>>();
void main() async {
  await GetStorage.init();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    TokenController tokenController =
        Get.put(TokenController(), tag: 'tokenController');

    return GetMaterialApp(
        navigatorObservers: <RouteObserver<ModalRoute<void>>>[routeObserver],
        title: 'GYM Manager',
        theme: ThemeData(
            brightness: Brightness.dark,
            primaryColor: Colors.lightGreen,
```

```
    accentColor: Colors.lightGreen,
    elevatedButtonTheme: ElevatedButtonThemeData(
        style: ButtonStyle(
            backgroundColor:
                MaterialStateProperty.all<Color>(Colors.lightGreen))),
    buttonTheme: ButtonThemeData(
      buttonColor: Colors.lightGreen,
    )),
getPages: [
  GetPage(
      name: '/',
      page: () {
        return tokenController.token.isNotEmpty
            ? MyHomePage()
            : LoginScreen();
      }),
  GetPage(
    name: '/home',
    page: () => MyHomePage(),
  ),
  GetPage(
    name: '/register',
    page: () => RegisterScreen(),
  ),
  GetPage(
    name: '/login',
    page: () => LoginScreen(),
  ),
  GetPage(
    name: '/forgot-password',
```

```
          page: () => ForgotPasswordScreen(),
        ),
      ]);
  }
}
```

Constants.dart

```
// String baseUrl = 'http://192.168.0.137:8000';
String baseUrl = 'https://fitnessmobileapp.herokuapp.com';
```

Api-expertice.dart

```
import 'dart:convert';

import 'package:get_storage/get_storage.dart';
import 'package:gym_manager/src/constants.dart';
import 'package:gym_manager/src/model/exercise.dart';
import 'package:http/http.dart' as http;

class ExerciseService {
  static var client = http.Client();

  static Future<Exercise> getWorkoutExerciseCompletedItem(
      String completedId) async {
    final box = GetStorage();

    var response = await client.get(
      Uri.parse(

'$baseUrl/workout/training/exercise/completed?completedId=$completedId'),
      headers: <String, String>{
```

```
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    },
  );

  if (response.statusCode == 200) {
    var exercise = jsonDecode(response.body);
    return Exercise.fromJson(exercise);
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<List<Exercise>> getCompletedExerciseList(
    String? exerciseId) async {
  final box = GetStorage();

  var response = await client.get(
    Uri.parse('$baseUrl/workout/training/exercise/completed/list'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    },
  );

  if (response.statusCode == 200) {
    var exercise = jsonDecode(response.body);
    var parsedData =
        List.from(exercise).map((exercise) => Exercise.fromJson(exercise));
    return exerciseId == null
```

```
        ? parsedData.toList()
        : parsedData
            .where((element) => element.exerciseId == exerciseId)
            .toList();
    } else {
      throw Exception(jsonDecode(response.body));
    }
  }
}
```

Api-training-type.dart

```
import 'dart:convert';

import 'package:get/instance_manager.dart';
import 'package:get_storage/get_storage.dart';
import 'package:gym_manager/src/constants.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
import 'package:gym_manager/src/model/workout-type.dart';
import 'package:http/http.dart' as http;

class WorkoutTypeService {
  static var client = http.Client();
  TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");

  static Future<List<WorkoutType>> fetchWorkoutTypes() async {
    final box = GetStorage();
    var response = await client
        .get(Uri.parse('$baseUrl/workout/type'), headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
```

```
      'Authorization': box.read('token') ?? ''
    });

    if (response.statusCode == 200) {
      var list = jsonDecode(response.body);
      List<WorkoutType>? trainings = list != null
          ? List.from(list).map((e) => WorkoutType.fromJson(e)).toList()
          : null;
      return trainings ?? List<WorkoutType>.empty();
    } else {
      throw Exception(jsonDecode(response.body));
    }
  }
}
```

Api-user.dart

```
import 'dart:convert';

import 'package:get_storage/get_storage.dart';
import 'package:gym_manager/src/model/user.dart';
import 'package:http/http.dart' as http;
import '../constants.dart';

class UserService {
  static var client = http.Client();

  static Future<User> registerUser(
      {required String name,
      required String email,
      required String password}) async {
```

```
  final response = await http.post(
    Uri.parse('$baseUrl/auth/registration'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode({'password': password, 'name': name, 'email': email}),
  );

  print(jsonDecode(response.body));

  if (response.statusCode == 201) {
    return User.fromJson(jsonDecode(response.body));
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<User> loginUser(
    {required String email, required String password}) async {
  final response = await http.post(Uri.parse('$baseUrl/auth/login'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: jsonEncode({'password': password, 'email': email}));

  print(jsonDecode(response.body));

  if (response.statusCode == 200) {
    return User.fromJson(jsonDecode(response.body));
  } else {
```

```
      throw Exception(jsonDecode(response.body));
    }
  }

  static Future<User> getUser() async {
    final box = GetStorage();

    var response =
        await client.get(Uri.parse('$baseUrl/user'), headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    });

    print(jsonDecode(response.body));

    if (response.statusCode == 200) {
      return User.fromJson(jsonDecode(response.body));
    } else {
      throw Exception(jsonDecode(response.body));
    }
  }
}
```

Api-workout-training.dart

```
import 'dart:convert';

import 'package:get/get.dart';
import 'package:get_storage/get_storage.dart';
import 'package:gym_manager/src/constants.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
```

```
import 'package:gym_manager/src/model/exercise.dart';
import 'package:gym_manager/src/model/workout-training.dart';
import 'package:http/http.dart' as http;

class WorkoutTrainingService {
  static var client = http.Client();
  static TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");

  static Future<WorkoutTraining> createWorkoutTraining(
      String trainingTemplate) async {
    final box = GetStorage();
    var response = await client.post(Uri.parse('$baseUrl/workout/training'),
        headers: <String, String>{
          'Content-Type': 'application/json; charset=UTF-8',
          'Authorization': box.read('token') ?? ''
        },
        body: jsonEncode({"id": trainingTemplate}));

    if (response.statusCode == 201) {
      var training = jsonDecode(response.body);
      return WorkoutTraining.fromJson(training);
    } else {
      throw Exception(jsonDecode(response.body));
    }
  }

  static Future<WorkoutTraining> getWorkoutTraining(String trainingid) async {
    final box = GetStorage();
    var response = await client.get(
```

```
    Uri.parse('$baseUrl/workout/training/$trainingid'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    },
  );

  if (response.statusCode == 200) {
    var training = jsonDecode(response.body);
    return WorkoutTraining.fromJson(training);
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<WorkoutTraining> startWorkoutTraining(String trainingid) async {
  final box = GetStorage();
  var response =
      await client.put(Uri.parse('$baseUrl/workout/training/start'),
          headers: <String, String>{
            'Content-Type': 'application/json; charset=UTF-8',
            'Authorization': box.read('token') ?? ''
          },
          body: jsonEncode({"trainingId": trainingid}));

  if (response.statusCode == 200) {
    var training = jsonDecode(response.body);
    return WorkoutTraining.fromJson(training);
  } else {
    throw Exception(jsonDecode(response.body));
```

```
  }
}

static Future<WorkoutTraining> finishWorkoutTraining(
    String trainingid) async {
  final box = GetStorage();
  var response =
      await client.put(Uri.parse('$baseUrl/workout/training/finish'),
          headers: <String, String>{
            'Content-Type': 'application/json; charset=UTF-8',
            'Authorization': box.read('token') ?? ''
          },
          body: jsonEncode({"trainingId": trainingid}));

  if (response.statusCode == 200) {
    var training = jsonDecode(response.body);
    return WorkoutTraining.fromJson(training);
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<WorkoutTraining> workoutTrainingExerciseStart(
    String trainingid, String exerciseId) async {
  final box = GetStorage();

  var response = await client.put(
      Uri.parse('$baseUrl/workout/training/exercise/start'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
```

```
        'Authorization': box.read('token') ?? ''
      },
      body: jsonEncode({"trainingId": trainingid, "exerciseId": exerciseId}));

  if (response.statusCode == 200) {
    var training = jsonDecode(response.body);
    return WorkoutTraining.fromJson(training);
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<WorkoutTraining> workoutTrainingExerciseFinish(
    Exercise completedExercise) async {
  final box = GetStorage();

  var response =
      await client.put(Uri.parse('$baseUrl/workout/training/exercise/finish'),
          headers: <String, String>{
            'Content-Type': 'application/json; charset=UTF-8',
            'Authorization': box.read('token') ?? ''
          },
          body: jsonEncode(completedExercise.toJson()));

  if (response.statusCode == 200) {
    var training = jsonDecode(response.body);
    return WorkoutTraining.fromJson(training);
  } else {
    throw Exception(jsonDecode(response.body));
  }
```

```
}

static Future<WorkoutTraining> addExercisesToWorkout(
    List<String> exercises, String trainingId) async {
  final box = GetStorage();

  var response = await client.put(Uri.parse('$baseUrl/workout/training'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
      body: jsonEncode({"trainingId": trainingId, "exercises": exercises}));

  if (response.statusCode == 200) {
    var training = jsonDecode(response.body);
    return WorkoutTraining.fromJson(training);
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<WorkoutTraining> deleteExerciseFromWorkout(
    String exerciseId, String trainingId) async {
  final box = GetStorage();

  var response = await client.delete(Uri.parse('$baseUrl/workout/training'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
```

```
      body: jsonEncode({"trainingId": trainingId, "exerciseId": exerciseId}));

  if (response.statusCode == 200) {
    var training = jsonDecode(response.body);
    return WorkoutTraining.fromJson(training);
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<List<WorkoutTraining>> getAllCompletedWorkouts() async {
  final box = GetStorage();

  var response = await client.get(
    Uri.parse('$baseUrl/workout/training'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    },
  );
  if (response.statusCode == 401) {
    if (tokenController.initialized) {
      tokenController.clearToken();
    } else {
      final box = GetStorage();
      box.remove('token');
    }
    Get.offAllNamed('/');
    throw Exception(jsonDecode('Не авторизований запит'));
  } else if (response.statusCode == 200) {
```

```
    var workouts = jsonDecode(response.body);
    return List.from(workouts)
        .map((workout) => WorkoutTraining.fromJson(workout))
        .toList();
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<List<WorkoutTraining>> deleteWorkout(String trainingId) async {
  final box = GetStorage();

  var response = await client.delete(Uri.parse('$baseUrl/workout/coaching'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
      body: jsonEncode({"trainingId": trainingId}));
  if (response.statusCode == 401) {
    if (tokenController.initialized) {
      tokenController.clearToken();
    } else {
      final box = GetStorage();
      box.remove('token');
    }
    Get.offAllNamed('/');
    throw Exception(jsonDecode('Не авторизований запит'));
  } else if (response.statusCode == 200) {
    var workouts = jsonDecode(response.body);
    return List.from(workouts)
```

```
        .map((workout) => WorkoutTraining.fromJson(workout))
        .toList();
    } else {
      throw Exception(jsonDecode(response.body));
    }
  }
}
```

Api-workout.dart

```
import 'dart:convert';

import 'package:get/get.dart';
import 'package:get_storage/get_storage.dart';
import 'package:gym_manager/src/constants.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
import 'package:gym_manager/src/model/workout-exercise.dart';
import 'package:gym_manager/src/model/workout.dart';
import 'package:http/http.dart' as http;

class WorkoutService {
  static var client = http.Client();
  static TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");

  static Future<List<Workout>> fetchWorkoutTemplates() async {
    final box = GetStorage();
    var response = await client
        .get(Uri.parse('$baseUrl/workout'), headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
```

```
  });

  if (response.statusCode == 200) {
    var list = jsonDecode(response.body);
    List<Workout>? trainigs = list != null
        ? List.from(list).map((e) => Workout.fromJson(e)).toList()
        : null;
    return trainigs ?? List<Workout>.empty();
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<List<WorkoutExersiseType>> getWorkoutExercise(
    List<String> exercises) async {
  final box = GetStorage();
  var response = await client.post(Uri.parse('$baseUrl/workout/exercise/all'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
      body: jsonEncode({"exercises": (exercises)}));

  if (response.statusCode == 200) {
    var list = jsonDecode(response.body);
    List<WorkoutExersiseType>? exercisesList = list != null
        ? List.from(list).map((e) => WorkoutExersiseType.fromJson(e)).toList()
        : null;
    return exercisesList ?? List<WorkoutExersiseType>.empty();
  } else {
```

```
    throw Exception(jsonDecode(response.body));
  }
}

static Future<List<WorkoutExersiseType>> getWorkoutExerciseAll() async {
  final box = GetStorage();
  var response = await client.get(
    Uri.parse('$baseUrl/workout/exercise'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    },
  );

  if (response.statusCode == 200) {
    var list = jsonDecode(response.body);
    List<WorkoutExersiseType>? exercisesList = list != null
        ? List.from(list).map((e) => WorkoutExersiseType.fromJson(e)).toList()
        : null;
    return exercisesList ?? List<WorkoutExersiseType>.empty();
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

static Future<List<WorkoutExersiseType>> editWorkoutExercise(
    WorkoutExersiseType newExercise) async {
  final box = GetStorage();
  var response = await client.put(Uri.parse('$baseUrl/workout/exercise'),
      headers: <String, String>{
```

```
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    },
    body: jsonEncode(newExercise.toJson()));

 if (response.statusCode == 200) {
  var list = jsonDecode(response.body);
  List<WorkoutExersiseType>? exercisesList = list != null
      ? List.from(list).map((e) => WorkoutExersiseType.fromJson(e)).toList()
      : null;
  return exercisesList ?? List<WorkoutExersiseType>.empty();
 } else if (response.statusCode == 401) {
  tokenController.clearToken();
  Get.offAllNamed('/');
  throw Exception(jsonDecode(response.body)["message"]);
 } else {
  throw Exception([jsonDecode(response.body)['message']]);
 }
}

static Future<List<WorkoutExersiseType>> createExercise(
  Map<String, String> exercise) async {
 final box = GetStorage();
 var response = await client.post(Uri.parse('$baseUrl/workout/exercise'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
      'Authorization': box.read('token') ?? ''
    },
    body: jsonEncode(exercise));
```

```
  if (response.statusCode == 201) {
    var list = jsonDecode(response.body);
    List<WorkoutExersiseType>? exercisesList = list != null
        ? List.from(list).map((e) => WorkoutExersiseType.fromJson(e)).toList()
        : null;
    return exercisesList ?? List<WorkoutExersiseType>.empty();
  } else if (response.statusCode == 401) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    throw Exception(jsonDecode(response.body)["message"]);
  } else {
    throw Exception([jsonDecode(response.body)['message']]);
  }
}

static Future<List<Workout>> createWorkoutTemplate(
    Map<String, dynamic> workoutTemplate) async {
  final box = GetStorage();
  var response = await client.post(Uri.parse('$baseUrl/workout'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
      body: jsonEncode(workoutTemplate));

  if (response.statusCode == 201) {
    var list = jsonDecode(response.body);
    List<Workout>? exercisesList = list != null
        ? List.from(list).map((e) => Workout.fromJson(e)).toList()
        : null;
```

```
    return exercisesList ?? List<Workout>.empty();
  } else if (response.statusCode == 401) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    throw Exception(jsonDecode(response.body)["message"]);
  } else {
    throw Exception([jsonDecode(response.body)['message']]);
  }
}

static Future<List<Workout>> deleteWorkoutTemplate(
    String workoutTemplateId) async {
  final box = GetStorage();
  var response = await client.delete(Uri.parse('$baseUrl/workout'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
      body: jsonEncode({"id": workoutTemplateId}));

  if (response.statusCode == 200) {
    var list = jsonDecode(response.body);
    List<Workout>? exercisesList = list != null
        ? List.from(list).map((e) => Workout.fromJson(e)).toList()
        : null;
    return exercisesList ?? List<Workout>.empty();
  } else if (response.statusCode == 401) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    throw Exception(jsonDecode(response.body)["message"]);
```

```
  } else {
    throw Exception([jsonDecode(response.body)['message']]);
  }
}

static Future<Workout> addNewExerciseToWorkoutTemplate(
    String workoutTemplateId, List<String> exercises) async {
  final box = GetStorage();
  var response = await client.put(Uri.parse('$baseUrl/workout/exercises'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
        'Authorization': box.read('token') ?? ''
      },
      body: jsonEncode({"id": workoutTemplateId, "exercises": exercises}));

  if (response.statusCode == 200) {
    var workout = jsonDecode(response.body);
    return Workout.fromJson(workout);
  } else if (response.statusCode == 401) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    throw Exception(jsonDecode(response.body)["message"]);
  } else {
    throw Exception([jsonDecode(response.body)['message']]);
  }
}

static Future<Workout> deleteExerciseFromWorkoutTemplate(
    String workoutTemplateId, String exerciseId) async {
  final box = GetStorage();
```

```
    var response = await client.delete(Uri.parse('$baseUrl/workout/exercises'),
        headers: <String, String>{
          'Content-Type': 'application/json; charset=UTF-8',
          'Authorization': box.read('token') ?? ''
        },
        body: jsonEncode({"id": workoutTemplateId, "exerciseId": exerciseId}));

    if (response.statusCode == 200) {
      var workout = jsonDecode(response.body);
      return Workout.fromJson(workout);
    } else if (response.statusCode == 401) {
      tokenController.clearToken();
      Get.offAllNamed('/');
      throw Exception(jsonDecode(response.body)["message"]);
    } else {
      throw Exception([jsonDecode(response.body)['message']]);
    }
  }
}
```

Completed-workouts-controller.dart

```
import 'package:get/get.dart';
import 'package:gym_manager/src/api/api-exercice.dart';
import 'package:gym_manager/src/api/api-workout-training.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
import 'package:gym_manager/src/model/exercise.dart';
import 'package:gym_manager/src/model/workout-training.dart';

class CompletedWorkoutsController extends GetxController {
  TokenController tokenController =
```

```
    Get.find<TokenController>(tag: "tokenController");

  RxBool isLoading = true.obs;
  RxBool isLoadingExercises = true.obs;

  var completedWorkouts = List<WorkoutTraining>.empty().obs;
  var completedExercisesList = List<Exercise>.empty().obs;

  void getAllCompletedWorkouts() async {
    try {
      isLoading(true);

      var workoutsList = await
WorkoutTrainingService.getAllCompletedWorkouts();
      completedWorkouts.assignAll(workoutsList.reversed);
    } catch (err) {
      Get.snackbar('Помилка', err.toString());
    } finally {
      isLoading(false);
    }
  }

  void deleteCompletedWorkout(String trainingId) async {
    try {
      isLoading(true);
      var workoutsList = await WorkoutTrainingService.deleteWorkout(trainingId);
      completedWorkouts.assignAll(workoutsList.reversed);
    } catch (err) {
      Get.snackbar('Помилка', err.toString());
    } finally {
```

```
    isLoading(false);
   }
  }

  void getCompletedExerciseListByExerciseId() async {
   try {
    isLoadingExercises(true);
    var completedExerciseList =
        await ExerciseService.getCompletedExerciseList(null);
    completedExercisesList.assignAll(completedExerciseList);
   } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Помилка', err.toString());
   } finally {
    isLoadingExercises(false);
   }
  }
}
```

Ecercise-controller.dart

```
import 'package:get/get.dart';
import 'package:gym_manager/src/api/api-exercice.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
import 'package:gym_manager/src/model/exercise.dart';
import 'package:gym_manager/src/model/set.dart';
import 'package:gym_manager/src/utils/convertation-weight.dart';

class ExerciseController extends GetxController {
  var count = 0.obs;
```

```
var weight = 0.0.obs;
var isLoading = true.obs;

TokenController tokenController =
    Get.find<TokenController>(tag: "tokenController");

var currentExercise =
    Exercise(exerciseSet: [], startTime: DateTime.now().toString()).obs;

var completedExerciseList = List<Exercise>.empty().obs;

var selectedWeightGroup = 0.obs;
var selectedUnit = 0.obs;

finishExercise() {
  this.currentExercise.update((val) {
    val!.endTime = DateTime.now().toString();
  });
}

setExerciseId(String exerciseId) {
  this.currentExercise.update((val) {
    val!.exerciseId = exerciseId;
  });
}

setTrainingId(String trainingId) {
  this.currentExercise.update((val) {
    val!.trainingId = trainingId;
  });
```

```
}

updateSelectedWeightGroup(int newGroup) {
  this.selectedWeightGroup.value = newGroup;
  weight.value = convertWeight(weight.value, newGroup);
}

updateSelectedUnit(int newUnit) {
  this.selectedUnit.value = newUnit;
}

updateCount(int value) {
  this.count.value = value;
}

updateWeight(double newWeight) {
  this.weight.value = double.parse(newWeight.toStringAsFixed(2));
}

updateSetByIndex(String newWeight, String newCount, String index) {
  List<ExerciseSet> copyList = List.from(currentExercise.value.exerciseSet);
  copyList[int.parse(index)] = ExerciseSet(
      weight: double.parse(newWeight.replaceFirst(',', '.')),
      count: int.parse(newCount));

  this.currentExercise.update((val) {
    val!.exerciseSet = copyList.toList();
  });
}
```

```
addSet() {
  this.currentExercise.update((val) {
    val!.exerciseSet = [
      ExerciseSet(
          count: this.count.value,
          weight: this.selectedWeightGroup.value == 0
              ? this.weight.value
              : convertWeight(this.weight.value, 0)),
      ...this.currentExercise.value.exerciseSet
    ];
  });
}

deleteSet(int index) {
  this.currentExercise.update((val) {
    val!.exerciseSet.removeAt(index);
  });
}

getCompletedExercise(String completedExerciseId) async {
  try {
    isLoading(true);
    var completedExercise =
        await ExerciseService.getWorkoutExerciseCompletedItem(
            completedExerciseId);
    this.currentExercise.value = completedExercise;
  } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Помилка', err.toString());
```

```
    } finally {
      isLoading(false);
    }
  }

  getCompletedExerciseListByExerciseId(String exerciseId) async {
    try {
      isLoading(true);
      var completedExerciseList =
          await ExerciseService.getCompletedExerciseList(exerciseId);
      this.completedExerciseList.assignAll(completedExerciseList);
    } catch (err) {
      tokenController.clearToken();
      Get.offAllNamed('/');
      Get.snackbar('Помилка', err.toString());
    } finally {
      isLoading(false);
    }
  }

  @override
  void onInit() {
    super.onInit();
  }
}
```

Slidable-list-controller.dart

```
import 'package:get/get.dart';

class SlidableListController extends GetxController {
```

```
  var list = List<Map<String, String>>.empty().obs;

  void setList(List<Map<String, String>> newList) {
    list.assignAll(newList);
  }
}
```

Token-controller.dart

```
import 'package:get/get.dart';
import 'package:get_storage/get_storage.dart';

class TokenController extends GetxController {
  final box = GetStorage();
  String get token => box.read('token') ?? '';
  void setToken(String token) => box.write('token', token);
  void clearToken() => box.remove('token');
}
```

User-controller.dart

```
import 'package:get/get.dart';
import 'package:gym_manager/src/api/api-user.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';

import 'package:gym_manager/src/model/user.dart';

class UserController extends GetxController {
  var isLoading = true.obs;
  var user =
      User(completedExercises: [], trainings: [], trainingTemplates: []).obs;
```

```
  TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");

  @override
  void onInit() {
    getUserData();
    super.onInit();
  }

  void getUserData() async {
    try {
      isLoading(true);
      var user = await UserService.getUser();
      this.user.value = user;
    } catch (err) {
      tokenController.clearToken();
      Get.offAllNamed('/');
      Get.snackbar('Failed to get user data', err.toString());
    } finally {
      isLoading(false);
    }
  }
}
```

Workout-controller.dart

```
import 'package:get/get.dart';
import 'package:gym_manager/src/api/api-workout.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
import 'package:gym_manager/src/model/workout.dart';
```

```
class WorkoutController extends GetxController {
  var isLoading = true.obs;
  var workoutList = List<Workout>.empty().obs;

  TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");

  @override
  void onInit() {
    getWorkouTemplates();
    super.onInit();
  }

  void getWorkouTemplates() async {
    try {
      isLoading(true);
      var workouts = await WorkoutService.fetchWorkoutTemplates();

      print('workouts: ${workouts.map((e) => e.toJson())}');
      this.workoutList.assignAll(workouts);
    } catch (err) {
      tokenController.clearToken();

      Get.offAllNamed('/');
      Get.snackbar('Failed to get user data', err.toString());
    } finally {
      isLoading(false);
    }
  }
```

```
void createWorkoutTemplate(Map<String, dynamic> workoutTemplate) async {
  try {
    isLoading(true);
    var workouts =
        await WorkoutService.createWorkoutTemplate(workoutTemplate);

    print('workouts: ${workouts.map((e) => e.toJson())}');
    this.workoutList.assignAll(workouts);
  } catch (err) {
    Get.snackbar('Failed to get user data', err.toString());
  } finally {
    isLoading(false);
  }
}

void deleteWorkoutTemplate(String workoutTemplate) async {
  try {
    isLoading(true);
    var workouts =
        await WorkoutService.deleteWorkoutTemplate(workoutTemplate);

    print('workouts: ${workouts.map((e) => e.toJson())}');
    this.workoutList.assignAll(workouts);
  } catch (err) {
    Get.snackbar('Помилка видалення', err.toString());
  } finally {
    Get.offNamed('/home');
    isLoading(false);
  }
}
```

```
}
```

Workout-exercise-controller.dart

```
import 'package:get/get.dart';
import 'package:gym_manager/src/api/api-workout.dart';
import 'package:gym_manager/src/model/workout-exercise.dart';
import 'package:gym_manager/src/model/workout.dart';

class WorkoutExerciseController extends GetxController {
  var isLoading = true.obs;
  var workoutExersiseList = List<WorkoutExersiseType>.empty().obs;
  var currentSelectedIndex = 0.obs;

  // final List<String> workoutExersiseListInitial;

  // WorkoutExerciseController({required this.workoutExersiseListInitial});

  // @override
  // void onInit() {
  //   if (this.workoutExersiseListInitial.length > 0)
  //     getWorkoutExercises(this.workoutExersiseListInitial);
  //   else
  //     getWorkoutExercisesAll();
  //   super.onInit();
  // }

  void setCurrentSelectedIndex(int index) {
    this.currentSelectedIndex.value = index;
  }
```

```
void getWorkoutExercises(List<String> exericisesIds) async {
  try {
    isLoading(true);
    var exercisesList =
        await WorkoutService.getWorkoutExercise(exericisesIds);

    print('exercisesList exersise: ${exercisesList.map((e) => e.toJson())}');
    this.workoutExersiseList.assignAll(exercisesList);
  } catch (err) {
    Get.snackbar('Failed to get exercises list data', err.toString());
  } finally {
    isLoading(false);
  }
}

void getWorkoutExercisesAll() async {
  try {
    isLoading(true);
    var exercisesList = await WorkoutService.getWorkoutExerciseAll();

    print('exercisesList all: ${exercisesList.map((e) => e.toJson())}');
    this.workoutExersiseList.assignAll(exercisesList);
  } catch (err) {
    Get.snackbar('Failed togetWorkoutExercisesAll', err.toString());
  } finally {
    isLoading(false);
  }
}

void createExercise(Map<String, String> exercise) async {
```

```
  try {
    isLoading(true);
    var newExerciseList = await WorkoutService.createExercise(exercise);

    print('new exercise success created');
    this.workoutExersiseList.assignAll(newExerciseList);
  } catch (err) {
    Get.snackbar('Failed to create list data', err.toString());
  } finally {
    isLoading(false);
  }
}

void updateExercise(
    WorkoutExersiseType newExercise, List<String> exercises) async {
  try {
    isLoading(true);
    await WorkoutService.editWorkoutExercise(newExercise);
    this.getWorkoutExercises(exercises);
    Get.back();
  } catch (err) {
    Get.snackbar('Failed to create list data', err.toString());
  } finally {
    isLoading(false);
  }
}

Future<List<String>?> addExercisesToWorkoutTemplate(
    String workoutTemplateId, List<String> exercisIds) async {
  try {
```

```
    isLoading(true);
    Workout workout = await
WorkoutService.addNewExerciseToWorkoutTemplate(
        workoutTemplateId, exercisIds);
    this.getWorkoutExercises(workout.exercises);
    return workout.exercises;
    // Get.back();
   } catch (err) {
    Get.snackbar('Failed to create list data', err.toString());
   } finally {
    isLoading(false);
   }
  }

  Future<List<String>?> deleteExerciseFromWorkoutTemplate(
    String workoutTemplateId, String exerciseId) async {
   try {
    isLoading(true);
    Workout workout = await
WorkoutService.deleteExerciseFromWorkoutTemplate(
        workoutTemplateId, exerciseId);
    this.getWorkoutExercises(workout.exercises);
    return workout.exercises;
    // Get.back();
   } catch (err) {
    Get.snackbar('Failed to create list data', err.toString());
   } finally {
    isLoading(false);
   }
  }
```

```
}
```

Workout-training-controller.dart

```
import 'package:get/get.dart';
import 'package:gym_manager/src/api/api-workout-training.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';
import 'package:gym_manager/src/model/exercise.dart';
import 'package:gym_manager/src/model/workout-training.dart';
import 'package:gym_manager/src/page/workout-page.dart';

class WorkoutTrainingController extends GetxController {
  var isLoading = true.obs;
  var workoutTraining = WorkoutTraining(
    date: DateTime.now(),
    exercises: [],
  ).obs;

  TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");

  @override
  void onInit() {
    // fetchTrainigTypes();
    super.onInit();
  }

  void createWorkoutTraining(String trainingTemplate) async {
    try {
      isLoading(true);
      var workoutTraining =
```

```
        await WorkoutTrainingService.createWorkoutTraining(trainingTemplate);
    print('workoutTraining:  ${workoutTraining.toJson()}');
    this.workoutTraining.value = workoutTraining;
    this.startWorkoutTraining();
  } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Failed Fetch', err.toString());
  } finally {
    isLoading(false);
  }
}

void getWorkoutTraining(String trainingId) async {
  try {
    isLoading(true);
    var workoutTraining =
        await WorkoutTrainingService.getWorkoutTraining(trainingId);
    print('workoutTraining:  ${workoutTraining.toJson()}');
    this.workoutTraining.value = workoutTraining;
  } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Failed Fetch', err.toString());
  } finally {
    isLoading(false);
  }
}

void workoutExerciseStart(String trainingId, String exerciseId) async {
```

```
  try {
    isLoading(true);
    var workoutTraining =
        await WorkoutTrainingService.workoutTrainingExerciseStart(
            trainingId, exerciseId);
    print('workoutTraining:  ${workoutTraining.toJson()}');
    this.workoutTraining.value = workoutTraining;
  } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Failed Fetch', err.toString());
  } finally {
    isLoading(false);
  }
}

void workoutExerciseFinish(Exercise completedExrecise,
    {bool isBack = false}) async {
  try {
    if (completedExrecise.exerciseSet.length == 0) {
      isBack ? Get.back() : Get.off(WorkoutPage());
    } else {
      isLoading(true);
      var workoutTraining =
          await WorkoutTrainingService.workoutTrainingExerciseFinish(
              completedExrecise);
      print(workoutTraining.toJson());
      print('workoutTraining:  ${workoutTraining.toJson()}');
      this.workoutTraining.value = workoutTraining;
      isBack ? Get.back() : Get.off(WorkoutPage());
```

```
    }
  } catch (err) {
    print(err.toString());
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Failed Fetch', err.toString());
  } finally {
    isLoading(false);
  }
}

void startWorkoutTraining() async {
  try {
    isLoading(true);
    var workoutTraining = await WorkoutTrainingService.startWorkoutTraining(
        this.workoutTraining.value.id);
    this.workoutTraining.value = workoutTraining;
  } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Failed Fetch', err.toString());
  } finally {
    isLoading(false);
  }
}

void finishWorkoutTraining() async {
  try {
    var workoutTraining = await WorkoutTrainingService.finishWorkoutTraining(
        this.workoutTraining.value.id);
```

```
    this.workoutTraining.value = workoutTraining;
    Get.offNamed('/home');
  } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Failed Fetch', err.toString());
  }
 }

 void addExercisesToWorkout(List<String> exercises, String trainingId) async {
  try {
    isLoading(true);
    var workoutTraining = await
WorkoutTrainingService.addExercisesToWorkout(
        exercises, trainingId);
    this.workoutTraining.value = workoutTraining;
  } catch (err) {
    tokenController.clearToken();
    Get.offAllNamed('/');
    Get.snackbar('Failed Fetch', err.toString());
  } finally {
    isLoading(false);
  }
 }

 void deleteExerciseFromWorkout(String exerciseId, String trainingId) async {
  try {
    isLoading(true);
    var workoutTraining =
        await WorkoutTrainingService.deleteExerciseFromWorkout(
```

```
            exerciseId, trainingId);
      this.workoutTraining.value = workoutTraining;
    } catch (err) {
      tokenController.clearToken();
      Get.offAllNamed('/');
      Get.snackbar('Failed Fetch', err.toString());
    } finally {
      isLoading(false);
    }
  }
}
```

Workout-type-controller.dart

```
import 'package:get/get.dart';
import 'package:gym_manager/src/api/api-training-type.dart';
import 'package:gym_manager/src/controllers/token-contoller.dart';

import 'package:gym_manager/src/model/workout-type.dart';

class WorkoutTypeController extends GetxController {
  var isLoading = true.obs;
  var workoutTypeList = List<WorkoutType>.empty().obs;
  TokenController tokenController =
      Get.find<TokenController>(tag: "tokenController");
  @override
  void onInit() {
    fetchTrainigTypes();
    super.onInit();
  }
```

```
  void fetchTrainigTypes() async {
    try {
      isLoading(true);
      var workoutTypeList = await WorkoutTypeService.fetchWorkoutTypes();
      print('workoutTypeList:  ${workoutTypeList.map((e) => e.toJson())}');
      this.workoutTypeList.assignAll(workoutTypeList);
    } catch (err) {
      tokenController.clearToken();
      Get.offAllNamed('/');
      Get.snackbar('Failed Fetch', err.toString());
    } finally {
      isLoading(false);
    }
  }
}
```

Model/exercise.dart

```
import 'dart:convert';

import 'set.dart';

Exercise exerciseFromJson(String str) => Exercise.fromJson(json.decode(str));

String exerciseToJson(Exercise data) => json.encode(data.toJson());

class Exercise {
  Exercise({
    this.id = '',
    this.exerciseId = '',
    this.trainingId = '',
```

```
    this.startTime = '',
    this.endTime = '',
    required this.exerciseSet,
  });

  String id;
  String exerciseId;
  String trainingId;
  String startTime;
  String endTime;
  List<ExerciseSet> exerciseSet;

  factory Exercise.fromJson(Map<String, dynamic> json) => Exercise(
        id: json["id"],
        exerciseId: json["exerciseId"],
        trainingId: json["trainingId"],
        startTime: json["startTime"],
        endTime: json["endTime"],
        exerciseSet: List<ExerciseSet>.from(
            json["sets"].map((x) => ExerciseSet.fromJson(x))),
      );

  Map<String, dynamic> toJson() => {
        "exerciseId": exerciseId,
        "trainingId": trainingId,
        "startTime": startTime,
        "endTime": endTime,
        "sets": List<dynamic>.from(exerciseSet.map((x) => x.toJson())),
      };
}
```

Models/set.dart

```
class ExerciseSet {
  ExerciseSet({
    required this.weight,
    required this.count,
  });

  final double weight;
  final int count;

  factory ExerciseSet.fromJson(Map<String, dynamic> json) => ExerciseSet(
        weight: json["weight"].toDouble(),
        count: json["count"],
      );

  Map<String, dynamic> toJson() => {
        "weight": weight,
        "count": count,
      };
}
```

Models/user.dart

```
import 'dart:convert';
import 'package:gym_manager/src/constants.dart';

import 'package:http/http.dart' as http;

Future<User> registerUser(
    {required String name,
```

```
    required String email,
    required String password}) async {
  final response = await http.post(
    Uri.parse('$baseUrl/auth/registration'),
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode({'password': password, 'name': name, 'email': email}),
  );

  print(jsonDecode(response.body));

  if (response.statusCode == 201) {
    return User.fromJson(jsonDecode(response.body));
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

Future<User> loginUser(
    {required String email, required String password}) async {
  final response = await http.post(Uri.parse('$baseUrl/auth/login'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: jsonEncode({'password': password, 'email': email}));

  print(jsonDecode(response.body));

  if (response.statusCode == 200) {
```

```
    return User.fromJson(jsonDecode(response.body));
  } else {
    throw Exception(jsonDecode(response.body));
  }
}

class User {
  User({
    this.id = '',
    required this.trainingTemplates,
    required this.completedExercises,
    required this.trainings,
    this.email = '',
    this.name = '',
    this.token = '',
  });

  final String id;
  final List<String> trainingTemplates;
  final List<String> completedExercises;
  final List<String> trainings;
  final String email;
  final String name;
  final String token;

  factory User.fromJson(Map<String, dynamic> json) => User(
        id: json["id"],
        trainingTemplates:
            List<String>.from(json["trainingTemplates"].map((x) => x)),
        completedExercises:
```

```
            List<String>.from(json["completedExercises"].map((x) => x)),
        trainings: List<String>.from(json["trainings"].map((x) => x)),
        email: json["email"],
        name: json["name"],
        token: json["token"],
      );

  Map<String, dynamic> toJson() => {
        "id": id,
        "trainingTemplates":
            List<dynamic>.from(trainingTemplates.map((x) => x)),
        "completedExercises":
            List<dynamic>.from(completedExercises.map((x) => x)),
        "trainings": List<dynamic>.from(trainings.map((x) => x)),
        "email": email,
        "name": name,
        "token": token,
      };
}
```

Models/workout-exercise.dart

```
import 'dart:convert';

List<WorkoutExersiseType> workoutExersiseTypeFromJson(String str) =>
    List<WorkoutExersiseType>.from(
        json.decode(str).map((x) => WorkoutExersiseType.fromJson(x)));

String workoutExersiseTypeToJson(List<WorkoutExersiseType> data) =>
    json.encode(List<dynamic>.from(data.map((x) => x.toJson())));
```

```
class WorkoutExersiseType {
  WorkoutExersiseType({
    this.id = '',
    this.name = '',
    this.exerciseType = '',
  });

  final String id;
  final String name;
  final String exerciseType;

  factory WorkoutExersiseType.fromJson(Map<String, dynamic> json) =>
      WorkoutExersiseType(
        id: json["id"],
        name: json["name"],
        exerciseType: json["exerciseType"],
      );

  Map<String, dynamic> toJson() => {
        "id": id,
        "name": name,
        "exerciseType": exerciseType,
      };
}
```

Models/workout-training-exercises.dart

```
class WorkoutTrainingExercise {
  WorkoutTrainingExercise(
      {required this.id,
      required this.name,
```

```
    required this.exerciseType,
    required this.status,
    required this.completedExerciseId});

  final String id;
  final String name;
  final String exerciseType;
  final int status;
  final String completedExerciseId;

  factory WorkoutTrainingExercise.fromJson(Map<String, dynamic> json) =>
    WorkoutTrainingExercise(
      id: json["id"] ?? "error id",
      name: json["name"] ?? "error name",
      exerciseType: json["exerciseType"] ?? "error exercise Type",
      status: json["status"] ?? 0,
      completedExerciseId:
        json["completedExerciseId"] ?? "not completed exerciess");

  Map<String, dynamic> toJson() => {
      "id": id,
      "name": name,
      "exerciseType": exerciseType,
      "status": status,
    };
}
```

Models/workout-training.dart

```
// To parse this JSON data, do
//
```

```
//     final workoutTraining = workoutTrainingFromJson(jsonString);

import 'dart:convert';

import 'package:gym_manager/src/model/workout-training-exercises.dart';

WorkoutTraining workoutTrainingFromJson(String str) =>
    WorkoutTraining.fromJson(json.decode(str));

String workoutTrainingToJson(WorkoutTraining data) =>
    json.encode(data.toJson());

class WorkoutTraining {
  WorkoutTraining(
      {this.id = '',
      this.name = '',
      required this.date,
      this.status = 0,
      required this.exercises});

  final String id;
  final String name;
  final DateTime date;
  final int status;
  final List<WorkoutTrainingExercise> exercises;

  factory WorkoutTraining.fromJson(Map<String, dynamic> json) =>
      WorkoutTraining(
        id: json["id"],
        name: json["name"],
```

```
      date: DateTime.parse(json["date"]),
      status: json["status"] ?? 0,
      exercises: List<WorkoutTrainingExercise>.from(
          json["exercises"].map((x) => WorkoutTrainingExercise.fromJson(x))),
    );

  Map<String, dynamic> toJson() => {
      "id": id,
      "name": name,
      "date": date.toIso8601String(),
      "status": status,
      "exercises": List<dynamic>.from(exercises.map((x) => x.toJson())),
    };
}
```

Models/workout-type.dart

```
class WorkoutType {
  WorkoutType({
    required this.id,
    required this.name,
  });

  final String id;
  final String name;

  factory WorkoutType.fromJson(Map<String, dynamic> json) => WorkoutType(
      id: json["id"],
      name: json["name"],
    );
```

```
  Map<String, dynamic> toJson() => {
        "id": id,
        "name": name,
      };
}
```

Models/workout.dart

```
import 'dart:convert';

List<Workout> workoutFromJson(String str) =>
    List<Workout>.from(json.decode(str).map((x) => Workout.fromJson(x)));

String workoutToJson(List<Workout> data) =>
    json.encode(List<dynamic>.from(data.map((x) => x.toJson())));

class Workout {
  Workout({
    required this.id,
    required this.exercises,
    required this.name,
  });

  final String id;
  final List<String> exercises;
  final String name;

  factory Workout.fromJson(Map<String, dynamic> json) => Workout(
        id: json["id"],
        exercises: List<String>.from(json["exercises"].map((x) => x)),
        name: json["name"] ?? '',
```

```
    );

  Map<String, dynamic> toJson() => {
        "id": id,
        "exercises": List<dynamic>.from(exercises.map((x) => x)),
        "name": name,
      };
}
```

Page/create-workout-template/chouse-workout-exercise-type.dart

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:gym_manager/src/controllers/workout-type-controller.dart';
import  'package:gym_manager/src/page/create-workout-template/choose-workout-exersise.dart';
import 'package:gym_manager/src/widget/elements/button.dart';

class ChooseWorkoutExericiseType extends StatefulWidget {
  final String workoutName;

  ChooseWorkoutExericiseType({
    required this.workoutName,
  });

  @override
  _ChooseWorkoutExericiseTypeState createState() =>
      _ChooseWorkoutExericiseTypeState(
        this.workoutName,
      );
}
```

```
class _ChooseWorkoutExericiseTypeState
    extends State<ChooseWorkoutExericiseType> {
  final String workoutName;

  _ChooseWorkoutExericiseTypeState(this.workoutName);

  List<String> _selectedWorkoutTypes = [];

  final createdController =
      Get.create(() => WorkoutTypeController(), tag: 'workoutTypes');
  final workoutTypesController =
      Get.find<WorkoutTypeController>(tag: "workoutTypes");

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(this.workoutName)),
      body: Container(
        child: Column(
          children: [
            Expanded(
              flex: 1,
              child: createWorkoutTypesList(),
            ),
            ButtonStyled(
                label: 'Далі',
                onPressed: () async {
                  List<String> exercises = await Get.to(ChooseWorkoutExercises(
                    filterExercises: this._selectedWorkoutTypes,
```

```
                workoutName: this.workoutName,
              ));
              Get.back(result: exercises);
            })
        ],
      ),
    ),
  );
}

void _onWorkoutTypesSelect(bool? selected, exerciseId) {
  if (selected == true) {
    setState(() {
      _selectedWorkoutTypes = [..._selectedWorkoutTypes, exerciseId];
    });
  } else {
    setState(() {
      _selectedWorkoutTypes = [
        ..._selectedWorkoutTypes.where((elem) {
          return elem != exerciseId;
        })
      ];
    });
  }
}

createWorkoutTypesList() {
  return Obx(() {
    if (workoutTypesController.isLoading.value) {
      return Center(child: CircularProgressIndicator());
```

```
    }
    return ListView.builder(
      itemCount: workoutTypesController.workoutTypeList.length,
      itemBuilder: (_, index) {
        return CheckboxListTile(
          value: _selectedWorkoutTypes
              .contains(workoutTypesController.workoutTypeList[index].id),
          onChanged: (bool? selected) {
            _onWorkoutTypesSelect(
                selected, workoutTypesController.workoutTypeList[index].id);
          },
          title: Text(workoutTypesController.workoutTypeList[index].name),
        );
      });
  });
 }
}
```

Page/create-workout-template/chouse-workout-exercise.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:get/get_state_manager/get_state_manager.dart';
import 'package:get/instance_manager.dart';
import 'package:get/get.dart';
import 'package:gym_manager/src/controllers/workout-exersise-controller.dart';
import 'package:gym_manager/src/controllers/workout-type-controller.dart';
import 'package:gym_manager/src/widget/elements/button.dart';
import 'package:gym_manager/src/widget/elements/input.dart';
import 'package:modal_bottom_sheet/modal_bottom_sheet.dart';
```

```
class ChooseWorkoutExercises extends StatefulWidget {
  final List<String> filterExercises;
  final String workoutName;

  const ChooseWorkoutExercises({
    required this.filterExercises,
    required this.workoutName,
  });

  @override
  ChooseWorkoutExercisesState createState() {
    return ChooseWorkoutExercisesState(
      this.filterExercises,
      this.workoutName,
    );
  }
}

class ChooseWorkoutExercisesState extends State<ChooseWorkoutExercises> {
  final List<String> filterExercises;
  final String workoutName;

  ChooseWorkoutExercisesState(
    this.filterExercises,
    this.workoutName,
  );

  final _formKeyExercise = GlobalKey<FormState>();
  String _selectedType = '';
  List<String> _selectedExercises = List<String>.empty();
```

```
TextEditingController newExercisesNameController = TextEditingController();

final createController =
    Get.create(() => WorkoutExerciseController(), tag: 'newTrainingPage');
final exercisesController =
    Get.find<WorkoutExerciseController>(tag: 'newTrainingPage');
final workoutTypesController =
    Get.find<WorkoutTypeController>(tag: 'workoutTypes');

void _onExercisesSelected(bool? selected, exerciseId) {
  if (selected == true) {
    setState(() {
      _selectedExercises = [..._selectedExercises, exerciseId];
    });
  } else {
    setState(() {
      _selectedExercises = [
        ..._selectedExercises.where((elem) {
          return elem != exerciseId;
        })
      ];
    });
  }
}

@override
void initState() {
  exercisesController.getWorkoutExercisesAll();
  super.initState();
```

```
}

@override
Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(
        title: Text("Виберіть вправи "),
      ),
      body: Obx(() {
        if (exercisesController.isLoading.value) {
          return Center(child: CircularProgressIndicator());
        }

        return Container(
          child: Column(
            children: [
              Expanded(flex: 2, child: createExerciseList()),
              ButtonStyled(
                onPressed: addNewExerciseModal,
                label: 'Добавити нову вправу',
              ),
              ButtonStyled(label: 'Підвердити', onPressed: submit)
            ],
          ),
        );
      }));
}

void changeSelectedType(String value) {
  setState(() {
```

```
    _selectedType = value;
  });
}

ListView createExerciseList() {
  final filteredExercisesList =
      exercisesController.workoutExersiseList.where((exercise) {
    return this.filterExercises.contains(exercise.exerciseType);
  }).toList();

  return ListView.builder(
      itemCount: filteredExercisesList.length,
      itemBuilder: (_, index) {
        return CheckboxListTile(
          value: _selectedExercises.contains(filteredExercisesList[index].id),
          onChanged: (bool? selected) {
            _onExercisesSelected(selected, filteredExercisesList[index].id);
          },
          title: Text(filteredExercisesList[index].name),
        );
      });
}

void submit() {
  if (_selectedExercises.length > 0) {
    Get.back(result: _selectedExercises);
  } else
    Get.snackbar('Помилка!', "Будь ласка виберіть хоча б одну вправу");
}
```

```
void addNewExerciseModal() {
  showCupertinoModalBottomSheet(
      expand: false,
      context: context,
      backgroundColor: Colors.transparent,
      builder: (context) {
        return Obx(() {
          if (workoutTypesController.isLoading.value) {
            return Center(child: CircularProgressIndicator());
          }
          return Material(
              shadowColor: Colors.red,
              child: SafeArea(
                child: Padding(
                  padding: const EdgeInsets.symmetric(vertical: 20.0),
                  child: Form(
                    key: _formKeyExercise,
                    child: Column(
                      children: <Widget>[
                        Padding(
                          padding: const EdgeInsets.only(bottom: 10),
                          child: Text('Створення вправи',
                              style: TextStyle(
                                  fontSize: 18, color: Colors.blue)),
                        ),
                        InputStyled(
                          controller: newExercisesNameController,
                          label: 'Назва вправи',
                          validator: (value) {
                            if (value == null || value.isEmpty) {
```

```
      return 'Заповніть поле з назвою вправи';
    }
    return null;
  },
),
SizedBox(
  height: 100,
  child: CupertinoPicker.builder(
      itemExtent: 40.0,
      onSelectedItemChanged: (int value) {
        changeSelectedType(workoutTypesController
            .workoutTypeList[value].id);
      },
      childCount: workoutTypesController
          .workoutTypeList.length,
      itemBuilder: (_, index) {
        return Text(
          workoutTypesController
              .workoutTypeList[index].name,
          style: TextStyle(color: Colors.white),
        );
      }),
),
ButtonStyled(
  onPressed: () {
    if (_formKeyExercise.currentState!.validate()) {
      exercisesController.createExercise({
        "name": newExercisesNameController.value.text
            .toString(),
        "exerciseType": _selectedType.length > 0
```

```
                        ? _selectedType
                        : workoutTypesController
                            .workoutTypeList[0].id
                    });
                    Get.back();
                    newExercisesNameController.clear();
                    changeSelectedType('');
                  }
                },
                label: 'Додати вправу',
              ),
            ],
          ),
        ),
      ),
    ));
  });
 });
}
```

Page/create-workout-template/create-new-training.dart

```
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:get/get_state_manager/get_state_manager.dart';
import 'package:get/instance_manager.dart';
import 'package:get/get.dart';
import 'package:gym_manager/src/controllers/workout-controller.dart';
import 'package:gym_manager/src/controllers/workout-exersise-controller.dart';
```

```
import  'package:gym_manager/src/page/create-workout-template/choose-workout-exercise-type.dart';
import 'package:gym_manager/src/widget/elements/button.dart';
import 'package:gym_manager/src/widget/elements/input.dart';

class CreateNewTraining extends StatefulWidget {
  @override
  CreateNewTrainingState createState() {
    return CreateNewTrainingState();
  }
}

class CreateNewTrainingState extends State<CreateNewTraining> {
  final _formKey = GlobalKey<FormState>();

  TextEditingController trainingNameController = TextEditingController();

  final createController =
      Get.create(() => WorkoutExerciseController(), tag: 'newTrainingPage');
  final exercisesController =
      Get.find<WorkoutExerciseController>(tag: 'newTrainingPage');

  final workoutTemplateControllerCreator =
      Get.create(() => WorkoutController(), tag: 'createNewTemplate');
  final workoutTemplateController =
      Get.find<WorkoutController>(tag: 'createNewTemplate');

  @override
  void initState() {
    exercisesController.getWorkoutExercisesAll();
```

```
  super.initState();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(
        title: Text("Створення тренування"),
      ),
      body: Obx(() {
        if (exercisesController.isLoading.value) {
          return Center(child: CircularProgressIndicator());
        }
        return Container(
          padding: EdgeInsets.symmetric(vertical: 5),
          child: Form(
            key: _formKey,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: <Widget>[
                Expanded(
                  child: InputStyled(
                    controller: trainingNameController,
                    label: 'Назва тренування',
                    validator: (value) {
                      if (value == null || value.isEmpty) {
                        return 'Заповніть поле з назвою тренування';
                      }
                      return null;
                    },
```

```
                ),
              ),
              Center(
                child: ButtonStyled(
                  onPressed: () async {
                    if (_formKey.currentState!.validate()) {
                      List<String>? exercises =
                          await Get.to(ChooseWorkoutExericiseType(
                        workoutName: trainingNameController.text,
                      ));
                      Get.back(result: {
                        "exercises": exercises,
                        "workoutName": trainingNameController.text
                      });
                    }
                  },
                  label: 'Далі',
                ),
              ),
            ],
          ),
        ),
      );
    }));
  }
}
```

Page/exercise-page/exercise-history.dart

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
```

```
import 'package:gym_manager/src/controllers/exercise-controller.dart';
import 'package:gym_manager/src/model/set.dart';
import 'package:gym_manager/src/utils/format-date.dart';

class ExerciseHistory extends StatefulWidget {
  const ExerciseHistory({Key? key}) : super(key: key);

  @override
  _ExerciseHistoryState createState() => _ExerciseHistoryState();
}

class _ExerciseHistoryState extends State<ExerciseHistory> {
  final exerciseController =
      Get.find<ExerciseController>(tag: 'exerciseMainPageController');

  String getParsedDate(int index) {
    return formatDate(
        exerciseController.completedExerciseList[index].startTime);
  }

  ExerciseSet foundMaxWeight(int index) {
    List<ExerciseSet> currentSets =
        exerciseController.completedExerciseList[index].exerciseSet;
    ExerciseSet maxWeightSet = currentSets
        .reduce((prev, curr) => prev.weight > curr.weight ? prev : curr);
    return maxWeightSet;
  }

  String parseExerciseSetToString(ExerciseSet exerciseSet) {
    return '${exerciseSet.weight} кг * ${exerciseSet.count}';
```

```
}

ExerciseSet calcMaxWeightInList() {
  var exercises = exerciseController.completedExerciseList.asMap().entries;
  List<ExerciseSet> parsedExercises = exercises.map((e) {
    return foundMaxWeight(e.key);
  }).toList();

  return parsedExercises.reduce(
      (value, element) => value.weight > element.weight ? value : element);
}

@override
Widget build(BuildContext context) {
  return Obx(() {
    if (exerciseController.isLoading.value) {
      return Center(child: CircularProgressIndicator());
    } else if (exerciseController.completedExerciseList.isEmpty) {
      return Center(child: Text('список пустий'));
    }
    return Column(crossAxisAlignment: CrossAxisAlignment.stretch, children: [
      Card(
        color: Colors.lightGreen,
        child: Padding(
          padding: const EdgeInsets.all(20.0),
          child: Text(
              'Рекорд: ${parseExerciseSetToString(calcMaxWeightInList())}'),
        ),
      ),
      Expanded(
```

```
          child: ListView.builder(
              itemCount: exerciseController.completedExerciseList.length,
              itemBuilder: (BuildContext context, int index) {
                return Card(
                  child: Padding(
                    padding: const EdgeInsets.all(20),
                    child: Row(
                      mainAxisAlignment: MainAxisAlignment.spaceBetween,
                      children: [
                        Text('Дата:  ${getParsedDate(index)}'),
                        Text(
                            'Макс: ${parseExerciseSetToString(foundMaxWeight(index))}')
                      ],
                    ),
                  ),
                );
              }),
        ),
      ]);
    });
  }
}
```

Page/exercise-page/exercise-page.dart

```
import 'package:badges/badges.dart';
import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:gym_manager/src/controllers/exercise-controller.dart';
import 'package:gym_manager/src/controllers/workout-training-controller.dart';
```

```
import 'package:gym_manager/src/model/workout-training-exercises.dart';
import 'package:gym_manager/src/page/exercise-page/exercise.dart';
import 'package:gym_manager/src/page/exercise-page/sets.dart';

import 'exercise-history.dart';

class ExerciseMainPage extends StatefulWidget {
  final WorkoutTrainingExercise exercise;

  const ExerciseMainPage({required this.exercise});

  @override
  _ExerciseMainPageState createState() =>
      _ExerciseMainPageState(exercise: this.exercise);
}

class _ExerciseMainPageState extends State<ExerciseMainPage> {
  final WorkoutTrainingExercise exercise;
  _ExerciseMainPageState({required this.exercise});

  final WorkoutTrainingController workoutTraining =
      Get.find<WorkoutTrainingController>(tag: "workoutTraining");
  final exerciseController =
      Get.put(ExerciseController(), tag: 'exerciseMainPageController');

  int _selectedIndex = 0;

  @override
  void initState() {
    super.initState();
```

```
  exerciseController.getCompletedExerciseListByExerciseId(this.exercise.id);
  if (this.exercise.completedExerciseId.isEmpty) {
    exerciseController
        .setTrainingId(workoutTraining.workoutTraining.value.id);
    exerciseController.setExerciseId(this.exercise.id);
  } else {
    this
        .exerciseController
        .getCompletedExercise(this.exercise.completedExerciseId);
  }
}

void _onItemTapped(int index) {
  setState(() {
    _selectedIndex = index;
  });
}

Widget getListScreens() {
  int currentvalue = Get.arguments?['preview'] ?? false
      ? _selectedIndex + 1
      : _selectedIndex;
  switch (currentvalue) {
    case 0:
      return ExercisePage(
        exerciseName: this.exercise.name,
      );
    case 1:
      return Obx(() {
        if (exerciseController.currentExercise.value.exerciseSet.length == 0)
```

```
          return Center(
            child: Text('Підходів немає'),
          );
        return SetsPage();
      });
    case 2:
      return ExerciseHistory();
    case 3:
      return Container(
        child: Text('timer'),
      );
    default:
      return ExercisePage(
        exerciseName: this.exercise.name,
      );
  }
}

_getBottomNavigationTabs() {
  List<BottomNavigationBarItem> items = [
    BottomNavigationBarItem(
        icon: Badge(
            shape: BadgeShape.circle,
            badgeColor: Colors.lightGreen,
            borderRadius: BorderRadius.circular(100),
            child: Icon(Icons.list),
            badgeContent: Obx(() {
              return Text(
                exerciseController.currentExercise.value.exerciseSet.length
                    .toString(),
```

```
                style: TextStyle(color: Colors.white),
              );
            })),
        label: 'Підходи',
        backgroundColor: Colors.lightGreen),
    BottomNavigationBarItem(
        icon: Icon(Icons.lock_clock),
        label: 'Історія',
        backgroundColor: Colors.lightGreen),
  ];
  if ((Get.arguments?['preview'] ?? false) == false) {
    items = [
      BottomNavigationBarItem(
          icon: Icon(
            Icons.work,
          ),
          label: 'Тренування',
          backgroundColor: Colors.lightGreen),
      ...items,
      BottomNavigationBarItem(
          icon: Icon(Icons.watch_later),
          label: 'Таймер',
          backgroundColor: Colors.lightGreen)
    ];
  }
  return items;
}

@override
Widget build(BuildContext context) {
```

```
    return GestureDetector(
      onTap: () => FocusManager.instance.primaryFocus?.unfocus(),
      onPanDown: (_) {
        FocusScope.of(context).requestFocus(FocusNode());
      },
      behavior: HitTestBehavior.opaque,
      child: Scaffold(
        body: SafeArea(child: getListScreens()),
        bottomNavigationBar: BottomNavigationBar(
          backgroundColor: Colors.lightGreen,
          items: _getBottomNavigationTabs(),
          currentIndex: _selectedIndex,
          selectedItemColor: Colors.white,
          onTap: _onItemTapped,
          showUnselectedLabels: false,
          showSelectedLabels: true,
        ),
      ),
    );
  }
}
```