

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ВІДСТЕЖЕННЯ ТА
АНАЛІТИКИ ТРЕНУВАНЬ
WORKOUT TRACKER AND ANALYTICS MOBILE APP

Виконав(ла): студент(ка) 4 курсу, групи КНТ-
121сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТИЩЕНКО С.С.

(ПРІЗВИЩЕ та ініціали)

Керівник СТЕПАНЕНКО О.О.

(ПРІЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ТИЩЕНКА Сергія Сергійовича
 (ПРІЗВИЩЕ, ім'я, по батькові)

- Тема проєкту (роботи) Мобільний застосунок для відстеження та аналітики тренувань.
Workout Tracker and Analytics Mobile App.
 керівник проєкту (роботи) к.т.н., доцент, СТЕПАНЕНКО Олександр
Олексійович
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
- затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168
- Строк подання студентом проєкту (роботи) 01 червня 2024 року
- Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СТЕПАНЕНКО О.О., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 04 » квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

Підпис Сергій ТИЩЕНКО
 (підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

Олександр СТЕПАНЕНКО
 (підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 147 с., 3 табл., 23 рис., 3 дод., 23 джерела.

МОБІЛЬНИЙ ЗАСТОСУНОК, ANDROID, IOS, REACT NATIVE, ТРЕНУВАННЯ.

Об'єкт дослідження – програмні засоби для відстеження та аналітики тренувань.

Предмет дослідження – мобільні застосунки для відстеження тренувань.

Мета роботи – створення мобільного застосунку для відстеження та аналітики тренувань.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування TypeScript, фреймворк React Native, мобільний пристрій з операційною системою Android, мобільний пристрій з операційною системою iOS.

Результати. Створено мобільний застосунок для відстеження та аналітики тренувань.

Висновки. Розроблено мобільний застосунок для відстеження та аналітики тренувань.

Галузь використання – особисте використання при будь-яких тренуваннях.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 147 pages, 3 tables, 23 figures, 3 appendixes, 23 sources.

MOBILE APPLICATION, ANDROID, IOS, REACT NATIVE, WORKOUT.

Object of study is a mobile application for workouts tracking and analytics.

Subject of research are mobile applications for workouts tracking.

The purpose of the work is to create a software system for workouts tracking and analysis.

Materials, methods, and tools: object-oriented programming, TypeScript programming language, React Native framework, mobile device with Android operating system, mobile device with iOS operating system.

Results. Mobile application for workout tracking and analytics has been created.

Conclusions Mobile application for workout tracking and analytics has been developed.

The field of use – personal use for any training.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Аналіз потреб користувача	10
1.2 Аналіз існуючих аналогів для відстеження тренувань	10
1.3 Висновки	14
2 Матеріали і методи.....	16
2.1 Обґрунтування використання фреймворку	16
2.2 Вибір фреймворку.....	18
2.2.1 React Native	18
2.2.2 Flutter	19
2.2.3 Ionic.....	21
2.2.4 Підсумок	22
2.3 Вибір мови програмування	23
2.4 Вибір архітектури	25
3 Розробка програмного забезпечення.....	28
3.1 Основні відомості про розробку.....	28
3.2 Реалізація інтерфейсу	30
3.2.1 Інтерфейс розділу «Вправи»	33
3.2.2 Інтерфейс сторінки календаря	39
3.2.3 Інтерфейс профілю.....	40
3.3 Дані та сховище.....	44
3.4 Висновки за розділом	45
4 Експлуатація, тестування та експериментальне дослідження програми	46
4.1 Призначення й умови застосування програми.....	46
4.2 Розгортання на локальному комп'ютері.....	46
4.3 Інсталяція застосунку	48
Висновки	50

Перелік джерел посилання	51
Додаток А Код програми.....	53
Додаток Б Технічне завдання.....	140
Додаток В Слайди презентації.....	136

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- API – Application Programming Interface;
IDE – Integrated Development Environment;
UI – User Interface;
ОС – Операційна система;
ПЗ – Програмне забезпечення.

ВСТУП

Сучасний стан розвитку інформаційних технологій та збільшення популярності здорового способу життя призводять до зростання інтересу до засобів моніторингу фізичної активності. Мобільні застосунки для відстеження та аналітики тренувань є важливою складовою цього тренду, оскільки вони надають користувачам можливість контролювати свої тренування, відстежувати прогрес і аналізувати результати.

Багато існуючих рішень орієнтовані на онлайн-інтеграції та залежні від зовнішніх серверів, що може створювати проблеми з безпекою даних і вимагає постійного доступу до інтернету. Це не завжди зручно для користувачів, які хочуть мати повний контроль над своїми даними. Також велика кількість подібних застосунків перевантажені функціями та намагаються нав'язати безліч способів допомогти користувачу (персоналізовані тренувальні плани, вибір правильних вправ тощо). Це незручно для користувачів, яким застосунок потрібен в якості асистента, а не консультанта. Таким чином, є необхідність у створенні мобільного застосунку, який дозволяє відстежувати тренування та аналізувати дані автономно.

Актуальність даної роботи зумовлена необхідністю створення автономного мобільного застосунку для трекінгу тренувань та аналізу даних, який би задовольняв потреби користувачів у зручності, точності та безпеці. Підстави для розробки цієї теми включають аналіз існуючих рішень, виявлення їхніх недоліків і визначення вимог користувачів до автономних трекерів фізичної активності.

Метою даної роботи є розробка мобільного застосунку, який забезпечить користувачам можливість ефективно відстежувати свої тренування та аналізувати дані локально, без інтеграції з зовнішніми сервісами. Використання цього застосунку є повністю індивідуальним для кожного користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз потреб користувача

Підтримка здорового способу життя та досягнення фізичної форми часто вимагають систематичного та цілеспрямованого підходу до тренувань.

Однією з ключових причин відстеження тренувань є бажання досягнення максимальних результатів у тренуваннях. Люди, які займаються спортом або просто прагнуть покращити своє здоров'я, мають великий інтерес до моніторингу своїх досягнень та прогресу. Відстеження тренувань дозволяє виявити сильні та слабкі сторони атлета, розробити ефективну стратегію навантажень та постійно покращувати результати, а також вчасно помічати та виправляти проблеми у своєму тренувальному плані.

Для успішного відстеження прогресу фізичних тренувань необхідно мати на увазі ключові метрики, які відображають різні аспекти фізичної активності та здоров'я. Серед таких метрик можуть бути кількість повторень, максимальна вага за один підхід та інші. Інтеграція цих метрик у мобільний застосунок дозволить користувачам зручно та ефективно відстежувати свій прогрес та досягати поставлених цілей у фізичній активності.

1.2 Аналіз існуючих аналогів для відстеження тренувань

Ринок мобільних застосунків для трекінгу тренувань розвивається швидкими темпами. Більшість сучасних програмних рішень пропонують широкий спектр функцій для відстеження фізичної активності: підрахунок кроків, вимірювання серцевого ритму, підрахунок калорій, аналіз тренувань та інтеграції з іншими фітнес-платформами. Подібна реалізація розрахована на людей з невеликими знаннями в області фітнесу, тренувань та здоров'я.

Нині більш досвідченому атлету стає важко знайти застосунок, який би виступав лише в якості «журналу» для внесення тренувань та відображення їх

статистики за певний період, а не нав'язував своє бачення оптимального тренування.

Для порівняння було обрано декілька різних додатків для відстеження тренувань.

1.2.1 Застосунок «GymKeeper»

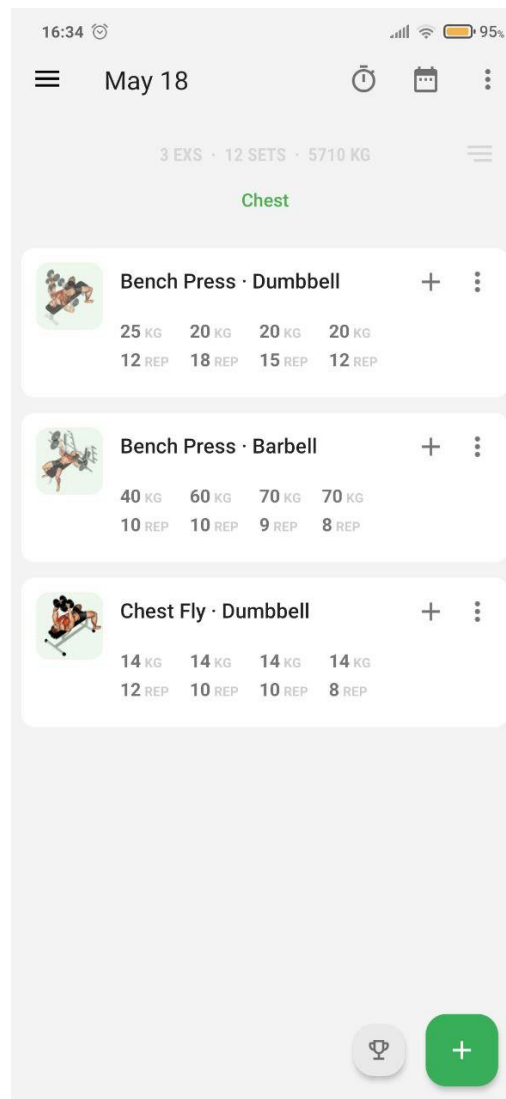


Рисунок 1.1 – Застосунок GymKeeper [1]

Даний застосунок (рис. 1.1) є популярним рішенням саме на роль «журналу» для ведення тренувань.

Переваги:

- великий вибір вбудованих вправ;

- сучасний та зручний дизайн;
- наявність статистики окремо за кожною вправою;
- відсутність необхідності створювати акаунт.

Недоліки:

- обмежена кількість доступних для запису тренувань (всього 12);
- для зняття обмеження потрібно придбати платну версію;
- рвані анімації, застосунок періодично працює доволі повільно;
- відсутність оновлень протягом довгого часу.

1.2.2 Застосунок «FitNotes»

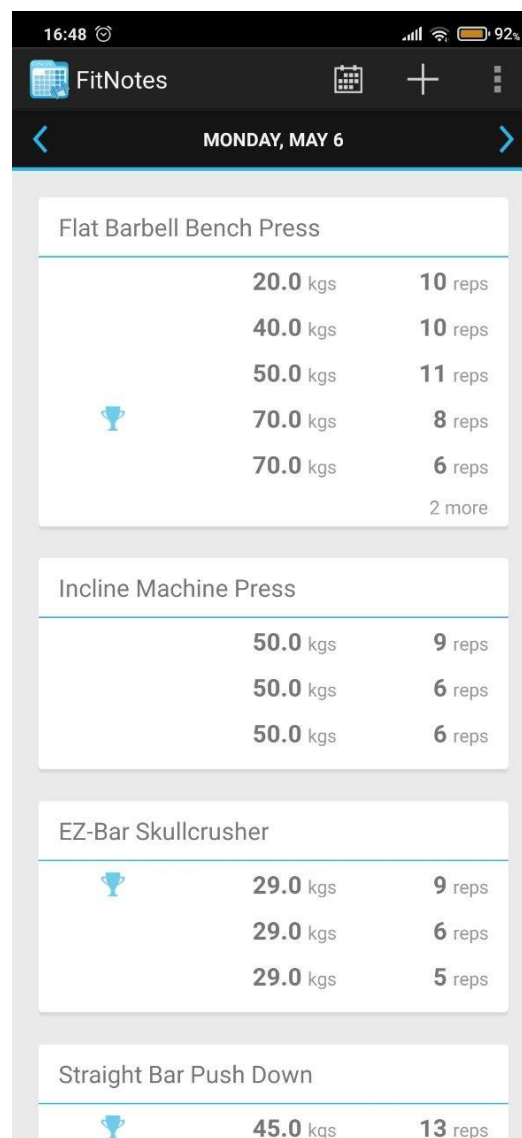


Рисунок 1.2 – Застосунок FitNotes [2]

Являє собою надійний журнал для запису тренувань (рис.1.2).

Переваги:

- мала вага додатку;
- весь основний функціонал є безкоштовним, без нав'язливих підписок чи реклами;

- не потребує створення акаунта;

- простий у використанні.

Недоліки:

- застарілий інтерфейс;
- відсутня версія для iOS;
- відсутність оновлень протягом довгого часу.

1.2.3 Застосунок «Home workouts BeStronger»

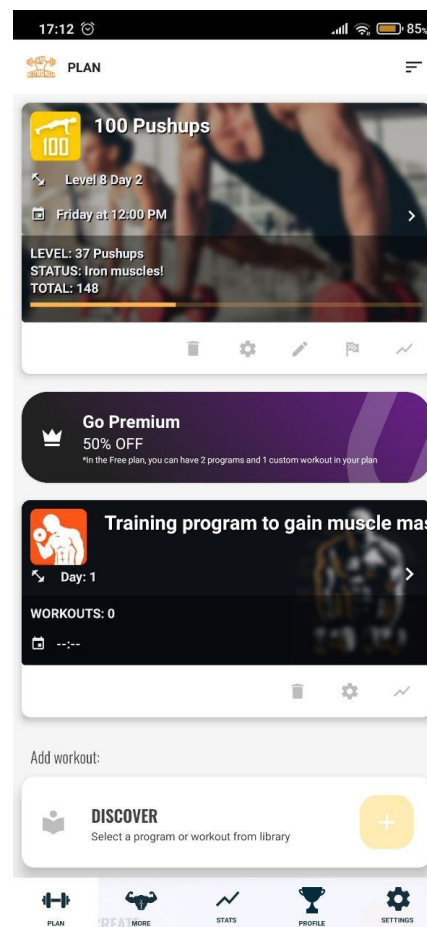


Рисунок 1.3 – Застосунок «Home workouts BeStronger» [3]

Даний застосунок (рис. 1.3) намагається поєднати в собі як функціонал журналу тренувань, так і асистента, пропонує готові програми, рецепти страв.

Переваги:

- великий набір готових програм тренувань для різних цілей;
- наявність версії для iOS;
- наявність довідника вправ;
- підтримка інтеграції з GoogleFit.

Недоліки:

- величезна кількість реклами у безкоштовній версії;
- значна частина функціоналу доступна лише після оформлення платної підписки;
- незручний інтерфейс;
- необхідне постійне інтернет-з'єднання;
- присутні помітні візуальні баги.

1.3 Висновки

Цей розділ розкриває ключові фактори, що спонукають людей використовувати мобільні додатки для відстеження тренувань. Такі програми слугують корисним інструментом для підтримки мотивації та самодисципліни, адже вони візуалізують прогрес, надають детальну статистику та пропонують додаткові функції, що роблять фізичні вправи більш цікавими та результативними.

Для визначення функціональних вимог майбутнього продукту було проведено аналіз предметної області та проведено порівняння з існуючими аналогами.

Результати порівняння за основними критеріями із розроблюваним застосунком (назва – «Trackfit») наведено у табл. 1.1.

Таблиця 1.1 – Порівняння програмних продуктів

Критерій порівняння	GymKeeper	FitNotes	Home Workouts BeStronger	Trackfit
Версія для Android	+	+	+	+
Версія для iOS	-	-	-	+
Необхідність створення акаунта	-	-	+	-
Наявність платних функцій	+	-	+	-
Наявність реклами	-	-	+	-
Необхідність інтернет-з'єднання	-	-	+	-

З наведеної таблиці видно, що всі аналоги не мають версії для ОС iOS, деякі закривають частину функціоналу до придбання платної підписки. Тому було визначено функціональні вимоги до розроблюваного застосунку, а саме:

- відсутність необхідності в авторизації;
- відсутність необхідності в інтернет з'єднанні;
- необмежений основний функціонал без необхідності оформлювати підписки чи робити будь-які інші покупки;
- відсутність реклами в застосунку;
- сучасний, інтуїтивний та простий інтерфейс;
- наявність версії для ОС Android;
- наявність версії для ОС iOS.

2 МАТЕРІАЛИ І МЕТОДИ

Мобільні застосунки стали неодмінною частиною нашого повсякденного життя завдяки їхній зручності та доступності. В будь-який момент вони дозволяють нам мати доступ до необхідної інформації та сервісів просто використовуючи смартфон. Для створення додатку планується використовувати фреймворк.

2.1 Обґрунтування використання фреймворку

Для того щоб зрозуміти причини використання фреймворку, варто розібрати відмінності та переваги в розробці застосунків при їх використанні.

Що таке фреймворк? Це набір інструментів, бібліотек та правил, який використовується для створення програм. Він зазвичай являє собою структуру, яка визначає, як компоненти програми повинні взаємодіяти між собою, які шаблони використовувати для створення інтерфейсів і які методи використовувати для роботи з базами даних та іншими зовнішніми ресурсами [4].

При розробці мобільних застосунків існують деякі типові проблеми. Одна з них - забезпечення сумісності з двома найпопулярнішими мобільними платформами (iOS, Android) та різними версіями операційних систем. Це може стати складним завданням, особливо при розробці мультиплатформених застосунків.

Традиційним, або нативним, методом для розробки мобільного застосунку буде використання інструментів, які розроблені для конкретної операційної системи. Наприклад, для Android використовується мова програмування Java чи Kotlin, а для iOS - Swift. Ці застосунки мають повний доступ до всіх функцій та можливостей пристрою і такі програми часто працюють швидше та ефективніше, оскільки вони оптимізовані для конкретної платформи.

Але головним недоліком даного підходу є велика вартість кінцевого продукту, оскільки потрібно створити два окремих застосунки, які ніяк між собою не пов'язані. Вони написані на різних мовах, використовують різні інструменти для збірки і найголовніше – вимагають окрему команду розробки для кожного застосунку, оскільки вони фактично є двома незалежними проєктами.

На щастя для вирішення цієї проблеми було створено багато фреймворків для кросплатформеної розробки мобільних застосунків.

Однією з головних переваг при використанні фреймворків є можливість створення кросплатформених додатків. Це означає, що один код може використовуватися для обох основних мобільних платформ - Android та iOS. Замість розробки окремих додатків для кожної платформи, можна зекономити час та ресурси, використовуючи єдиний код для обох. Це значно скорочує час розробки і суттєво знижує витрати на підтримку та оновлення застосунків, оскільки потрібна лише одна команда розробки.

Фреймворки підходять як для розробки простих застосунків, так і для створення багатофункціональних корпоративних продуктів зі складними алгоритмами та різноманітними можливостями.

До недоліків можна віднести гіршу оптимізацію в порівнянні з нативними застосунками, але сучасні фреймворки працюють дуже швидко у більшості випадків, тому кінцевий користувач не помітить жодних проблем при використанні.

При розробці продукту слід використовувати найкращі практики для конкретного фреймворку, обирати правильну архітектуру та в цілому слідкувати за якістю коду, що актуально і для розробки застосунків використовуючи специфічні для конкретної платформи мови програмування та інші інструменти.

2.2 Вибір фреймворку

Існує багато відомих фреймворків, що дозволяють створювати мобільні застосунки. У цьому розділі буде представлено порівняння кількох найпопулярніших фреймворків для мобільної розробки.

2.2.1 React Native

React Native — це фреймворк для розробки мобільних додатків, створений компанією Meta (колишній Facebook) [5]. Він дозволяє розробникам створювати додатки для iOS та Android за допомогою JavaScript і React. Основна ідея React Native полягає в тому, щоб надати розробникам можливість використовувати єдину кодову базу для створення кросплатформених додатків, що значно скорочує час і витрати на розробку.

Його створення стало результатом прагнення компанії поліпшити досвід розробки мобільних додатків, спираючись на успіх бібліотеки React, яка використовувалася для розробки веб-додатків. Сьогодні React Native активно розвивається і підтримується спільнотою розробників та компанією Meta (раніше Facebook).

Основними концепціями є:

- компонентний підхід: React Native використовує компоненти як основні будівельні блоки додатків. Компоненти можуть бути як простими (кнопка, текстове поле), так і складними (форми, списки). Це дозволяє створювати модульний код, який можна використовувати багато разів;
- JSX: React Native використовує JSX (JavaScript XML) для опису інтерфейсів користувача. JSX дозволяє писати HTML-подібний код всередині JavaScript, що робить код більш читабельним і зрозумілим;
- Мости і нативні модулі: React Native дозволяє використовувати нативний код (Java, Swift, Objective-C) для реалізації специфічних функцій, які важко або неможливо реалізувати використовуючи JavaScript. Це

забезпечується через так званий "міст" (Bridge), який дозволяє обмінюватися даними між JavaScript та нативним кодом.

Переваги React Native:

- кросплатформеність;
- React Native використовує нативні компоненти для рендерингу інтерфейсу, що забезпечує продуктивність, близьку до нативних додатків у більшості випадків;
- швидкий цикл розробки: завдяки функції "гарячого перезавантаження" (hot reloading) [6], розробники можуть миттєво бачити зміни в коді без необхідності перезавантажувати додаток. Це значно пришвидшує процес розробки та тестування;
- велика спільнота та екосистема: React Native має велику і активну спільноту, яка створює і підтримує численні бібліотеки та інструменти, що розширюють можливості фреймворку. Це дозволяє легко знаходити існуючі рішення для різних задач та проблем.

Недоліки:

- швидкодія при роботі із 3D графікою: React Native не підходить для створення складних і візуально навантажених ігор. Він не має вбудованих інструментів та бібліотек для роботи з 3D, а існуючі бібліотеки значно поступаються спеціалізованим ігровим движкам;
- реалізація деяких функцій може вимагати використання нативного коду, що збільшує складність розробки і потребує знань Java або, Swift.

2.2.2 Flutter

Flutter — це сучасний фреймворк для розробки кросплатформених мобільних, веб- та десктопних додатків, створений компанією Google [7]. Він дозволяє розробникам використовувати єдиний кодовий базис для створення додатків під різні платформи, зберігаючи при цьому високу продуктивність та нативний вигляд додатків.

Основні концепції:

- Dart: використовує мову програмування Dart, також створену Google. Dart відома своєю простотою та ефективністю, що робить її чудовим вибором для розробки додатків;
- компонентний підхід: подібно до React, Flutter використовує компоненти (або віджети) як основні будівельні блоки інтерфейсу користувача. У Flutter все є віджетом — від макету до анімацій. Віджети є незмінними, і кожен стан інтерфейсу представлений новим набором віджетів. Це дозволяє легко керувати станом додатка;
- швидкий цикл розробки: однією з ключових особливостей Flutter є функція "гарячого перезавантаження" (hot reload), яка дозволяє розробникам миттєво бачити зміни в коді без перезавантаження додатка.

Переваги Flutter:

- кросплатформеність;
- висока продуктивність: Flutter використовує власний графічний движок Skia, який забезпечує високу продуктивність та нативний вигляд додатків. Dart компілюється у нативний код, що додатково покращує продуктивність;
- активна спільнота та підтримка Google: Flutter має активну спільноту розробників та підтримку від Google, що забезпечує регулярні оновлення, нові функції та велику кількість бібліотек та плагінів.

Недоліки:

- необхідність вивчати нову мову програмування. Використання Dart може бути перешкодою для розробників, які не знайомі з цією мовою;
- молодість фреймворку: Хоча Flutter активно розвивається, він все ще є відносно молодим фреймворком. Це означає, що можуть виникати проблеми з підтримкою деяких нових функцій або інтеграцією зі старими системами.

2.2.3 Ionic

Ionic — це популярний фреймворк для розробки кросплатформених мобільних додатків, який дозволяє використовувати звичні веб-технології, такі як HTML, CSS і JavaScript, для створення додатків для iOS, Android та веб. Розроблений компанією Ionic, цей фреймворк надає потужні інструменти та бібліотеки, що дозволяють розробникам створювати додатки з нативним виглядом і відчуттям, використовуючи той самий код.

Основні концепції:

- дозволяє створювати мобільні додатки, використовуючи знайомі веб-технології, такі як HTML, CSS і JavaScript. Це робить його доступним для широкого кола веб-розробників;
- гібридний підхід: Ionic використовує веб-інтерфейс, який працює всередині веб-переглядача (WebView) у мобільному додатку. Це дозволяє створювати програми, які мають нативний вигляд і відчуття, але при цьому використовують веб-технології;
- Capacitor: це платформа від Ionic, яка дозволяє використовувати нативні API для доступу до функцій пристрою, таких як камера, GPS, файлові системи тощо.

Переваги:

- кросплатформеність;
- гнучкість у виборі фреймворків: Ionic підтримує різні фреймворки для побудови додатків, включаючи Angular, React і Vue. Це дозволяє розробникам вибирати інструменти, які найкраще відповідають їхнім потребам.

Недоліки:

- продуктивність: написаний код не трансформується у нативні компоненти, натомість він являє собою веб-сторінку, «загорнуту» у оболонку для відповідної платформи;

- обмежений доступ до нативних API: хоча Capacitor дозволяє використовувати багато нативних функцій, деякі специфічні або нові функції можуть бути недоступними або вимагати додаткових зусиль для інтеграції;
- розмір додатку: гібридні додатки можуть бути більшими за розміром у порівнянні з нативними, що може вплинути на швидкість завантаження та споживання пам'яті.

2.2.4 Підсумок

При виборі фреймворку слід враховувати особливості конкретного проєкту, навички команди розробників та можливість майбутнього покращення та підтримки продукту. В таблиці 2.1 наведено порівняння описаних фреймворків.

Таблиця 2.1 – Порівняння популярних мобільних фреймворків

Критерій порівняння	React Native	Flutter	Ionic
Мова програмування	JavaScript	Dart	HTML, CSS, JavaScript
Активність і розмір спільноти	Велика та активна спільнота, з багатою документацією та ресурсами	Швидко зростаюча спільнота, з активною підтримкою Google	Зріла спільнота, з фокусом на веб-розробку
Гнучкість	Висока гнучкість, дозволяє використовувати власні компоненти та бібліотеки	Висока гнучкість, завдяки власному двигуну рендерингу	Висока гнучкість, за рахунок використання веб-технологій
Продуктивність	Висока, близька до нативної	Дуже висока, близька до нативної	Помірна, може бути нижча порівняно з нативними додатками

Після детального огляду трьох популярних фреймворків для розробки мобільних додатків, для розробки було обрано фреймворк React Native.

Причини:

- React Native використовує JavaScript — одну з найпопулярніших мов програмування у світі. Це робить його доступним для широкого кола розробників;
- має велику і активну спільноту, яка створила численні бібліотеки та готові компоненти для різних задач. Це значно пришвидшує процес розробки, дозволяючи використовувати готові рішення для реалізації складних функціональних можливостей;
- гнучкість і масштабованість: фреймворк не диктує жорстких рамок стосовно структури проєкту, дозволяє зручно інтегрувати існуючі компоненти поряд із власними.

2.3 Вибір мови програмування

Оскільки було обрано фреймворк React Native, то найбільш очевидним варіантом є використовувати JavaScript. Але для розробки було обрано TypeScript. Мета цього підрозділу пояснити причини такого вибору.

JavaScript є однією з найвідоміших мов програмування, широко застосовуваною в розробці фронтенду та бекенду. TypeScript, створений компанією Microsoft, є надбудовою JavaScript, яка додає статичну типізацію та інші можливості. Це означає, що будь-який код JavaScript є валідним TypeScript кодом. Натомість TypeScript привносить нові функції та розширює синтаксис JavaScript, що робить код більш надійними, зручним в обслуговуванні та масштабуванні [8].

Однією з особливостей TypeScript є те, що це компільована мова програмування, на відміну від інтерпретованої мови JavaScript. Це означає, що перед запуском програми, код TypeScript буде перетворений у чистий JavaScript код, який вже буде виконуватися у цільовому середовищі. Хоча це

може здатися зайвим, але це допомагає у виявленні помилок на етапі компіляції, що в свою чергу дозволяє збільшити швидкість розробки, особливо при роботі над великими проектами великою командою. Крім того, компіляція дозволяє TypeScript розширити функціонал, який потім перетворюється в чистий JavaScript.

Ще однією відмінністю TypeScript від JavaScript є строга типізація. Це означає, що змінні, функції, аргументи та інші конструкції повинні мати визначений конкретний тип даних.

Також TypeScript підтримує об'єктно-орієнтовані принципи, такі як класи, інтерфейси, перерахування та узагальнена типізація, що дозволяє розкрити повний потенціал цієї мови програмування.

Основні переваги TypeScript [9]:

- безпечність типів, яка досягається за рахунок статичної типізації, інтерфейсів, перерахувань, тощо та допомагає уникати помилок завдяки чітко визначеним типам для змінних, функцій та об'єктів;
- краща читабельність коду, зручність підтримки та масштабування;
- інтеграція із різними IDE, що в результаті дає зручні інструменти для розробника, такі як навігація кодом, автодоповнення, позначення помилок типізації;
- сумісність з JavaScript: оскільки TypeScript є розширенням JavaScript, це дозволяє поступово впроваджувати його у вже існуючі проекти написані на JavaScript.

Недоліки:

- розробникам, які звикли до JavaScript, треба певний час для вивчення TypeScript і його особливостей;
- використання TypeScript вимагає додаткової конфігурації і налаштувань у проекті.

Враховуючи всі переваги, було вирішено використовувати саме TypeScript. Це дозволить підвищити надійність, безпеку коду, а також покращити можливість подальшого масштабування.

2.4 Вибір архітектури

У сучасній розробці вибір архітектури грає ключову роль у забезпеченні масштабованості, підтримуваності та ефективності роботи додатка. Однією з популярних архітектур для управління станом у фронтенд-розробці є FLUX. Ця архітектура була запропонована компанією Facebook і широко використовується у поєднанні з фреймворком React. У цьому розділі буде розглянуто що таке FLUX, основні концепції, переваги та недоліки.

FLUX — це архітектурний підхід для управління даними у фронтенд-додатках, який передбачає односторонній потік даних. Основні компоненти FLUX-архітектури [10][11]:

- дії (Actions): дії є об'єктами, які несуть дані від користувача або інших джерел до диспетчера. Вони описують подію, яка відбулася в додатку (наприклад, користувач натиснув кнопку);
- диспетчер (Dispatcher): диспетчер є центральним хабом, який приймає всі дії та передає їх відповідним обробникам. Він відповідає за координацію дій і забезпечує їх послідовне оброблення;
- сховища (Stores): сховища зберігають стан додатка і логіку управління станом. Вони підписуються на дії від диспетчера, оновлюють свій стан і повідомляють про зміни своїх підписників;
- вигляд (View): вигляд є презентаційним рівнем додатка, який відображає стан, збережений у сховищах. Компоненти React є типовими виглядами у FLUX-архітектурі.

Цей підхід забезпечує чітку структуру і потік даних, що полегшує управління станом додатка.

Узагальнену схему взаємодії описаних компонентів наведено нижче (рис. 2.1).

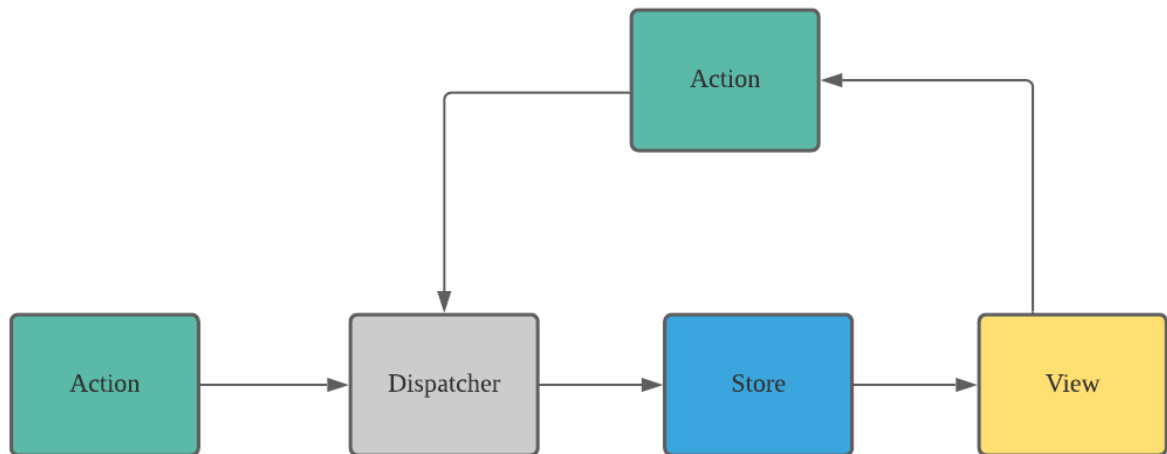


Рисунок 2.1 – Узагальнена схема взаємодії у FLUX [12]

Переваги FLUX:

- односторонній потік даних: робить архітектуру передбачуваною і зрозумілою (рис 2.2);

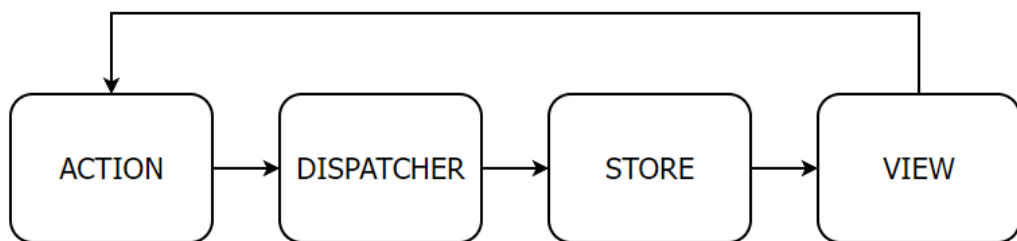


Рисунок 2.2 – Візуалізація одностороннього потоку даних [13]

- чітка структура: поділ додатка на дії, диспетчер, сховища та вигляди забезпечує чітку організацію коду, що полегшує підтримку і масштабування проекту;
- проста логіка управління станом: FLUX забезпечує просту і передбачувану логіку управління станом, що знижує ймовірність виникнення складних багів і покращує надійність додатка;
- легкість тестування: компоненти FLUX легко піддаються модульному тестуванню, що покращує якість коду і знижує ймовірність помилок.

Недоліки:

- крива навчання: для розробників, які не знайомі з концепцією FLUX, може знадобитися час на вивчення і розуміння цієї архітектури;
- більша кількість коду: використання FLUX може призвести до збільшення кількості коду через необхідність створення окремих файлів для дій, диспетчера і сховищ.

Вибір FLUX для даного проекту базується на його здатності забезпечити передбачуване, надійне і масштабоване управління станом додатка. Незважаючи на певні недоліки, такі як крива навчання і збільшення кількості коду, переваги FLUX значно перевищують ці мінуси в контексті нашого проекту.

Також він ідеально інтегрується з React, який вже використовується. Це дозволяє використовувати всі переваги обох технологій.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Основні відомості про розробку

Зручність і якість розробки проекту напряду залежить від обраної архітектури. Це фундамент для підтримки та розширення програми в майбутньому.

Важливим аспектом читаємості та зручності у навігації є форматування коду. Для цього було використано Prettier та ESLint.

Prettier - це інструмент форматування коду, який автоматично приводить код до визначеного стилю, включаючи розташування дужок, відступи, розмір табуляції тощо. Використання Prettier допомагає забезпечити послідовність у стилі коду між членами команди та зменшує час, який витрачається на ручне форматування. Він спрощує процес роботи з кодом, забезпечуючи чистоту та уніфікований стиль усього проекту [14].

ESLint - це інструмент статичного аналізу коду для JavaScript, призначений для виявлення потенційних проблем у вашому коді та виконання консольних правил стилю. Він допомагає забезпечити однорідність, чистоту та надійність вашого коду, допомагаючи виявляти помилки, які можуть виникнути при виконанні або масштабуванні програмного забезпечення. Використання ESLint дозволяє команді розробників працювати більш ефективно, дотримуючись спільних стандартів коду та зменшуючи кількість помилок в коді [15].

Узагальнена файлова система застосунку виглядає наступним чином:

- файли конфігурацій;
- файли компонентів;
- файли навігаторів;
- файли екранів;
- файли допоміжних функцій;
- файли типів;

– файли стану додатку.

Детальну файлову структуру із вказанням імен наведено у таблиці 3.1.

Таблиця 3.1 – Файлова структура проєкту

Тип файлу	Назва файлу
1	2
Файли конфігурацій	.gitignore
	.prettierrc
	babel-config.js
	tsconfig.json
	.eslintrc.json
Файли компонентів	base.button.tsx
	atoms.tsx
	timerModal.tsx
	WorkoutByDate.tsx
	GraphTab.tsx
	WeightTab.tsx
	exerciseCard.tsx
	graphPeriodSelector.tsx
	headerTimer.tsx
	setDetails.tsx
Файли навігаторів	login.navigator.tsx
	profile.navigator.tsx
	root.navigator.tsx
	tab.navigator.tsx
	workout.navigator.tsx
Файли екранів	calendar.screen.tsx
	about.screen.tsx
	analysis.screen.tsx
	bodyTracker.screen.tsx
	stats.screen.tsx
	profile.screen.tsx
	categories.screen.tsx
	exercises.screen.tsx
	home.tsx
	workout.screen.tsx
	workoutLog.screen.tsx
Файли допоміжних функцій	date.utils.tsx
	utils.tsx
Файли типів	types.ts

Продовження таблиці 3.1

1	2
Файли стану додатку	selectors.ts
	slice.ts
	types.ts
	index.ts
	root.store.ts

Файли у форматі `tsx` – містять у собі TypeScript код, який відповідає за рендер компонентів чи сторінок на екрані телефону. Стилiзація всіх компонентів і сторінок включені у цей файл. Файли стану додатку використовуються для керування глобальним станом додатку. Також вони містять код для взаємодії із внутрішнім сховищем смартфона.

3.2 Реалізація інтерфейсу

Інтерфейс створений із компонентів. Компонент являє собою невелику незалежну частину інтерфейсу. Компонентний підхід має багато переваг:

- модульність: компоненти дозволяють розділити інтерфейс на логічні блоки, спрощуючи навігацію та підтримку коду;
- перевикористання: компоненти можна легко використовувати в різних частинах додатку або навіть у різних проєктах, що спрощує розробку та зменшує дублювання коду;
- швидкість розробки: компоненти спрощують процес розробки, оскільки можна швидко виправляти помилки та додавати новий функціонал, не впливаючи на інші частини додатку.

Інтерфейс застосунку написаний із застосуванням власних компонентів, а також компонентів із бібліотеки `React Native Paper`, яка надає зручні та стильні UI-елементи, щоб швидко будувати додатки для мобільних пристроїв. Вона має в собі широкий набір готових компонентів, таких як кнопки, тексти, картки, меню та інші, які можна легко налаштовувати та

розширювати за потреби. React Native Paper пропонує чистий дизайн та використовує Material Design від Google, що робить додатки стильними та сучасними.

У React Native кожен компонент має життєвий цикл. Життєвий цикл компонентів у React визначає різні етапи, через які проходить компонент від його створення до видалення з DOM. Цей цикл можна розділити на кілька основних фаз: монтування, оновлення та демонтування. Для кожної фази є відповідні методи життєвого циклу.

Фаза створення (Mounting) включає в себе методи `constructor()`, `render()`, `componentDidMount()`.

Фаза оновлення (Updating) включає в себе методи `componentDidUpdate()`, `shouldComponentUpdate()`.

Фаза розмонтування (Unmounting) викликається перед тим, як компонент буде видалений. Включає в себе метод `componentWillUnmount()`.

Сучасний React використовує підхід функціональних компонентів та хуків замість класів. Вони використовують функції для створення компонентів. Замість методів життєвого циклу використовують хуки. Хуки дають можливість компоненту взаємодіяти із існуючими інструментами React такі як внутрішній стан (`useState`), підписка на події (`useEffect`), контекст (`useContext`), виконувати побічні ефекти.

Життєвий цикл компонентів у React зображений на рис. 3.1.



React Hooks Lifecycle

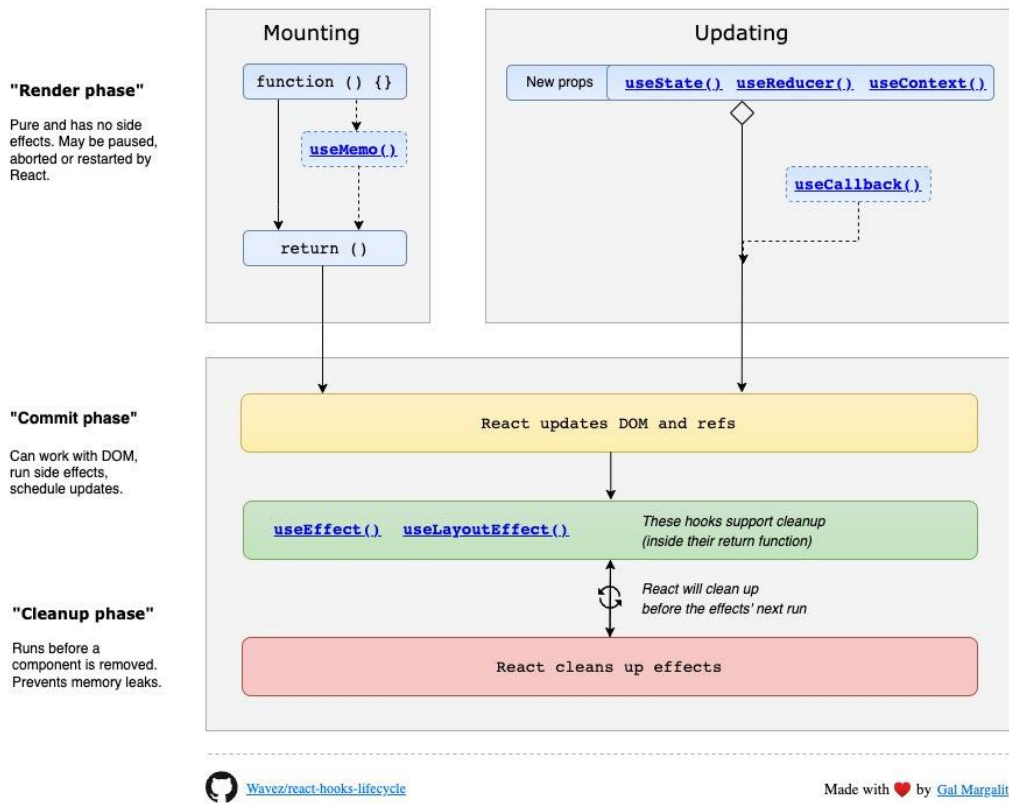


Рисунок 3.1 – Життєвий цикл компонента в React [16]

Навігація додатку побудована навколо трьох основних розділів: «вправи», «календар» та «профіль». Навігатор виконано у вигляді таб-навігатора у нижній частині екрану (рис. 3.2).

Кожен розділ містить один чи декілька екранів, які відповідають за певну частину функціоналу програми.

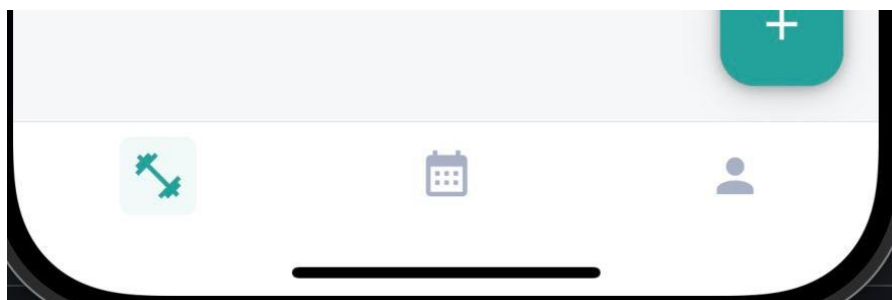


Рисунок 3.2 – Таб-навігатор застосунку

3.2.1 Інтерфейс розділу «Вправи»

Домашня сторінка даного розділу являє собою список із вправ, які користувач зробив за сьогодні. Список може бути порожнім (рис. 3.3), або мати контент (рис. 3.4). На екрані присутня кнопка для додавання вправ, кнопка для старту таймеру (рис. 3.5) у верхній частині екрану.

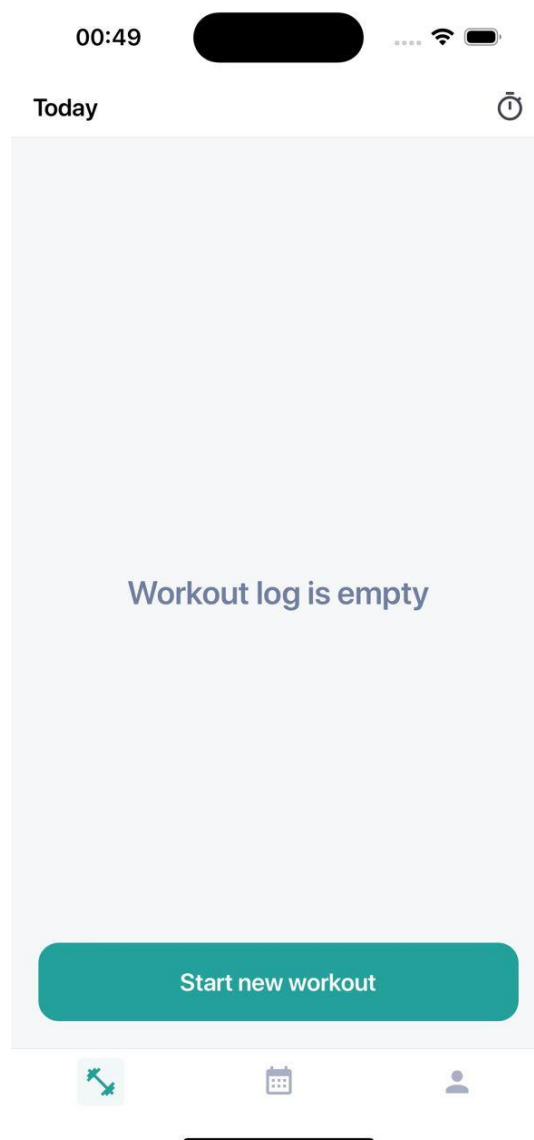


Рисунок 3.3 – Порожня домашня сторінка

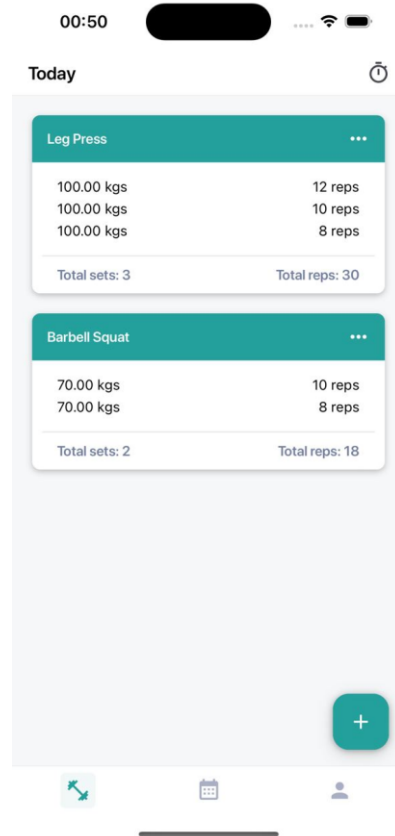


Рисунок 3.4 – Домашня сторінка із вправами

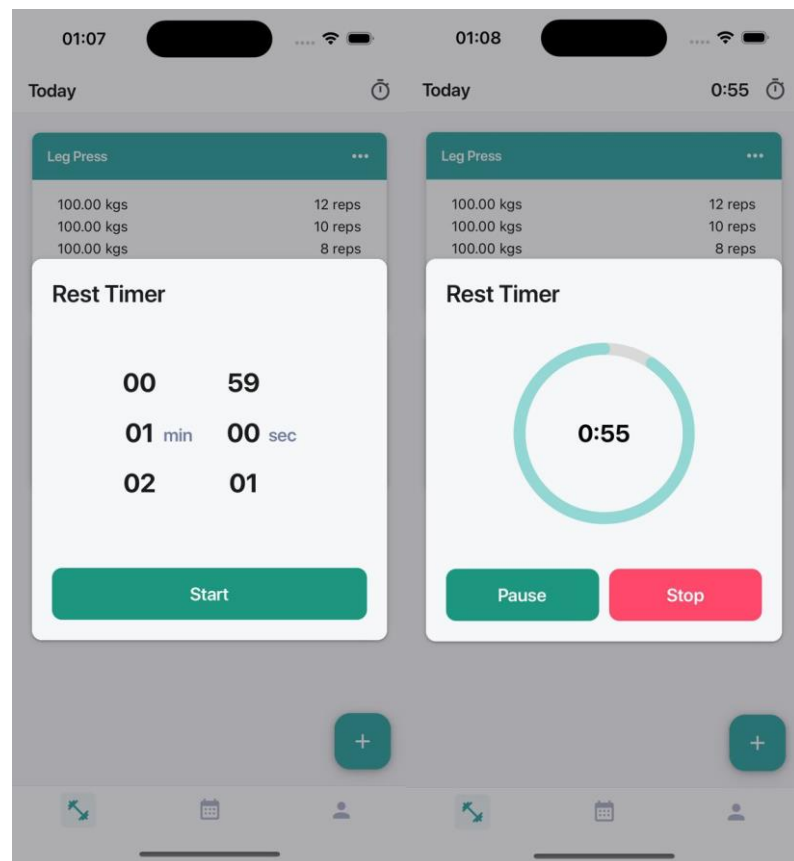


Рисунок 3.5 – Інтерфейс таймеру

Після натискання кнопки «Додати тренування» або кнопки «+» відкривається екран вибору категорії вправи (рис. 3.6).

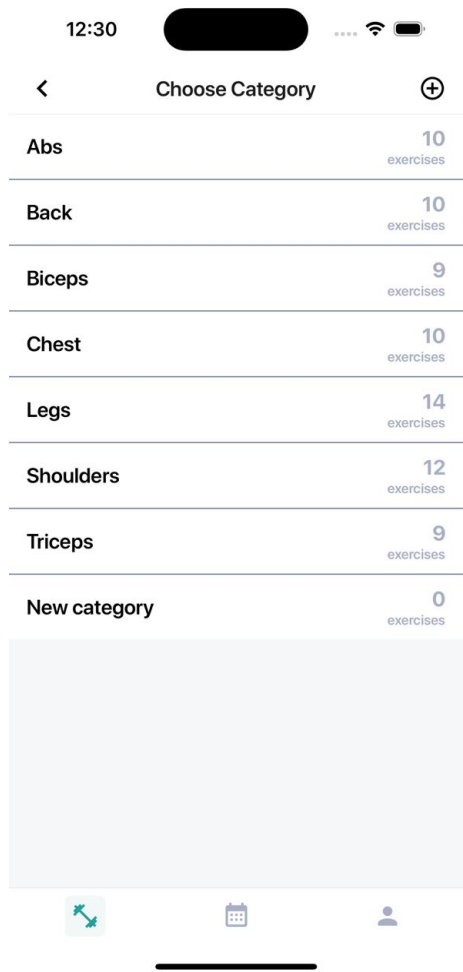


Рисунок 3.6 – Екран вибору категорії

Після вибору категорії відкривається екран вибору вправи за обраною категорією (рис. 3.7).

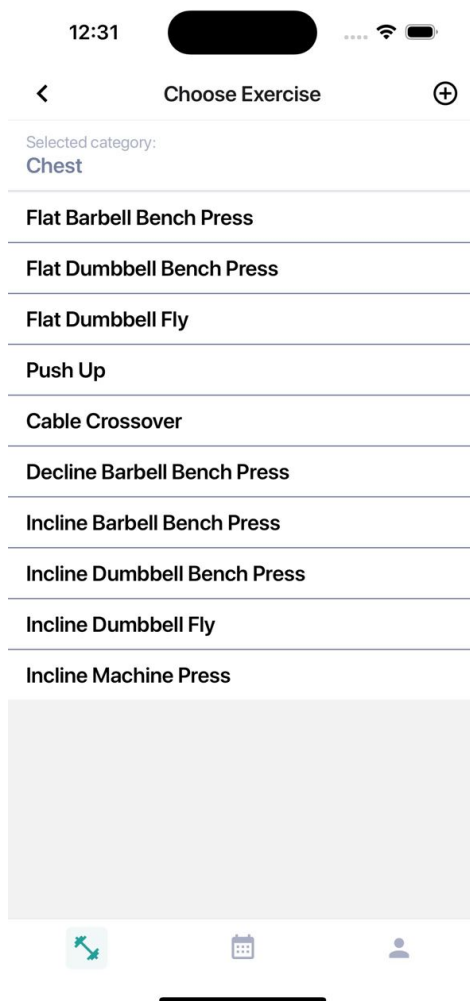


Рисунок 3.7 – Екран вибору вправи

Вибір вправи відкриває екран для відстеження конкретної вправи (рис. 3.8). Він складається із форми, де можна вказати вагу та кількість повторень, а також списку підходів за конкретний день для цієї вправи. Також у хедері сторінки присутній доступ до таймеру, щоби після виконання підходу користувач мав змогу засікти час відпочинку.

Кожен підхід можна видалити, або редагувати, якщо раніше було введено помилкові дані.

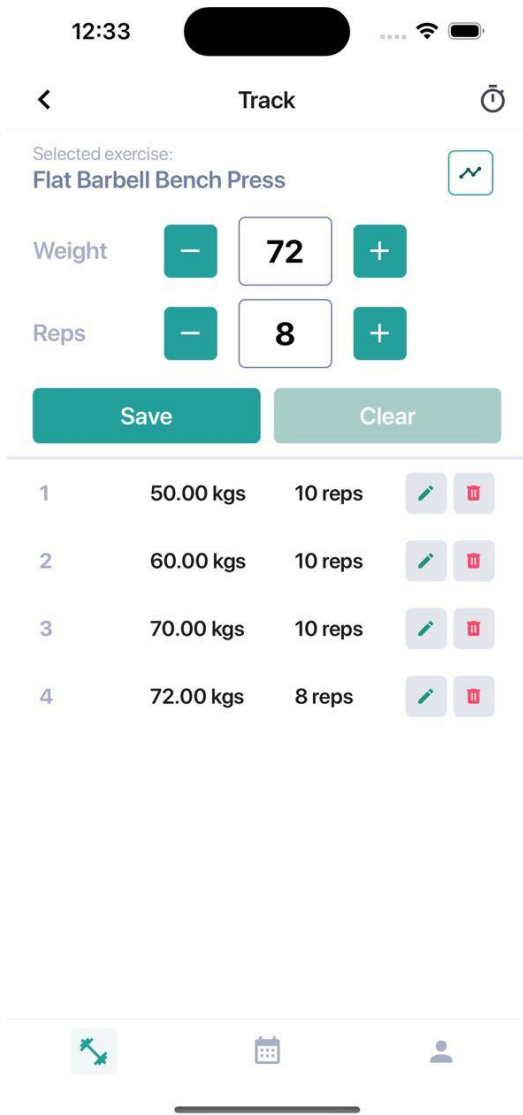


Рисунок 3.8 – Екран відстеження вправи

Для перегляду історії та статистики за конкретною вправою потрібно натиснути на кнопку із графіком.

Екран статистики виконаний у вигляді двох вкладок: історія підходів (рис. 3.9) та графік (рис. 3.10).

Історія підходів показує інформацію відсортовану за датою. Таким чином можна легко згадати яку вагу та яку кількість повторень користувач робив минулого разу для конкретної вправи.

Графік відображає статистику за певний період (1,3,6,12 місяців або за весь час) та обраним критерієм.



Рисунок 3.9 – Історія підходів



Рисунок 3.10 – Графік виконаних вправ

3.2.2 Інтерфейс сторінки календаря

Екран календаря складається із власне календаря та компонента, що відображає історію виконаних вправ за обраний день (рис. 3.11). Також він дозволяє перейти до редагування вправ за обраний день.

Усі дні, коли було записане будь-яке тренування відмічені крапкою під цифрою дати.

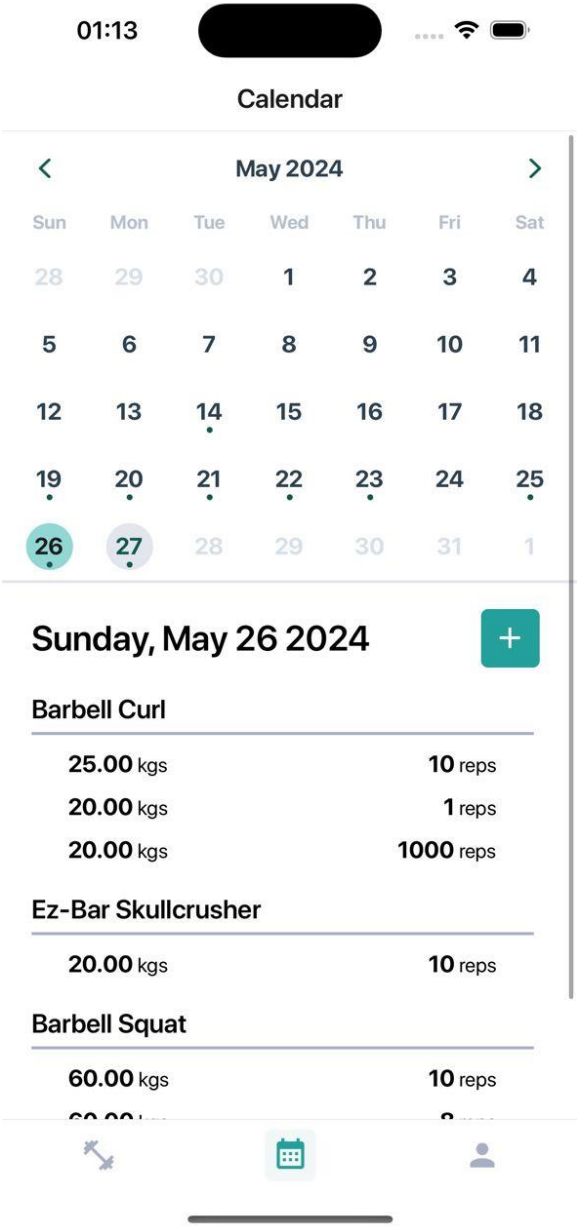


Рисунок 3.11 – Інтерфейс календаря із обраним днем

3.2.3 Інтерфейс профілю

Розділ профілю складається із 4 екранів: «Трекер тіла», «Статистика», «Аналітика» та «Про застосунок».

Основна сторінка профілю містить кнопки для переходу на інші екрани (рис. 3.12).

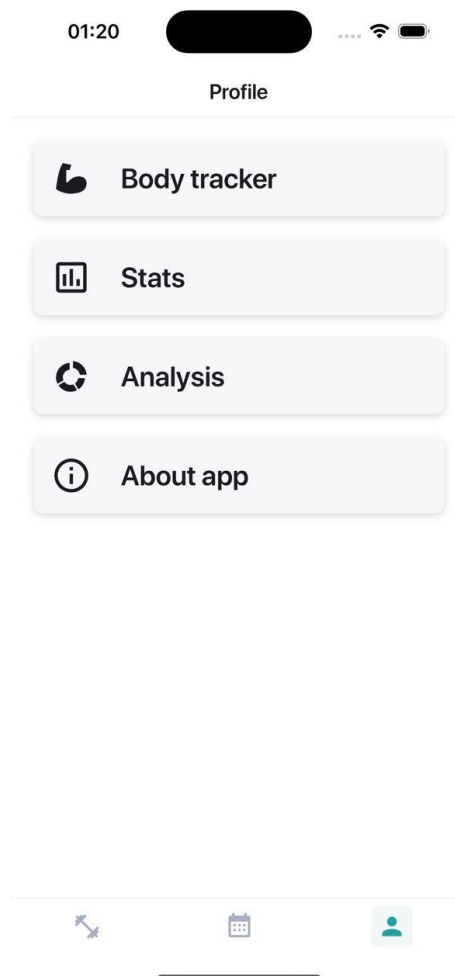


Рисунок 3.12 – Інтерфейс екрану профілю

Екран «Трекер тіла» дозволяє вести історію ваги власного тіла, яка відображається у вигляді списку (рис. 3.13). Якщо перейти на вкладку графіку - можна подивитись динаміку ваги за обраний період (рис. 3.14).

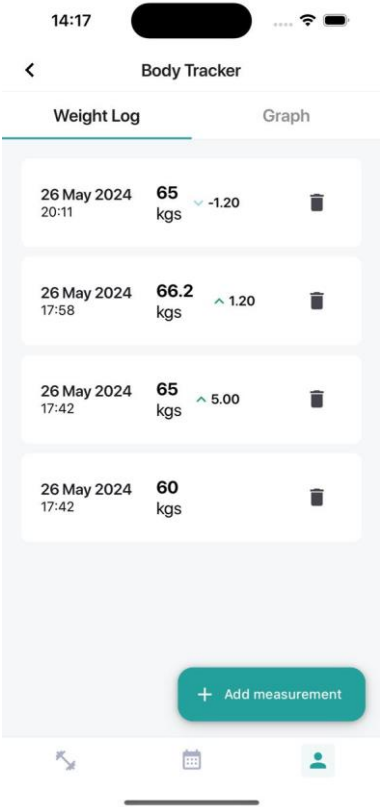


Рисунок 3.13 – Історія ваги

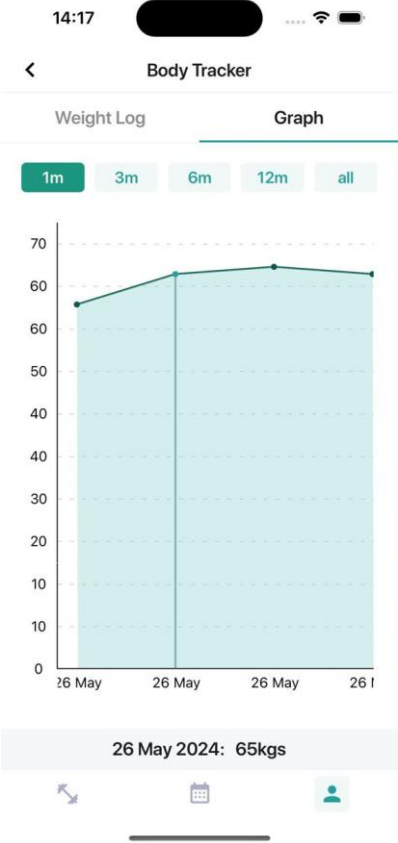


Рисунок 3.14 – Графік ваги

Екран «Статистика» містить узагальнену статистику користувача за весь період користування застосунком (рис. 3.15). Включає в себе загальну кількість тренувань, загальну кількість підходів, загальну кількість повторень та сумарну підняту вагу за всі тренування.

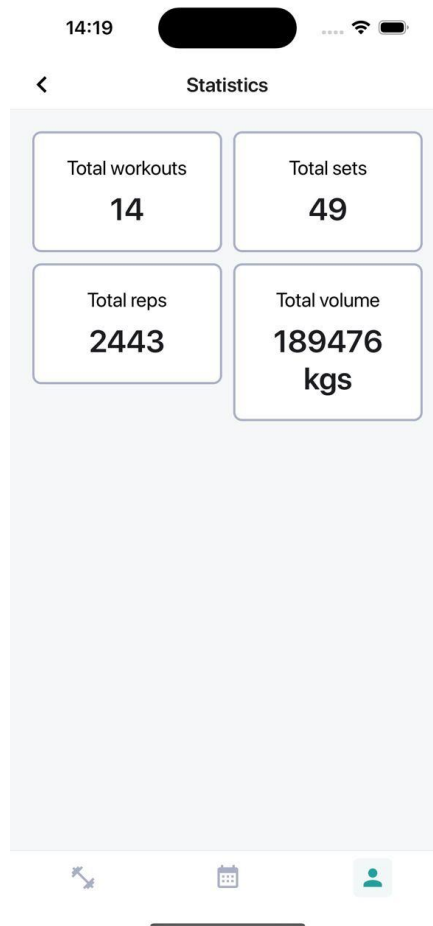


Рисунок 3.15 – Екран загальної статистики

Екран «Аналітика» містить результати аналізу тренувань, що зображений у вигляді секторної діаграми (pie chart) (рис. 3.16).. Він дозволяє побачити статистику за обраний період часу та критерієм. Доступні критерії: кількість підходів, кількість повторень, кількість тренувань.

Це корисно для виявлення дизбалансу при тренуваннях протягом великого проміжку часу. Можна побачити які м'язи тренуються недостатньо чи навпаки занадто багато. Під графіком наведено точну статистику за обраним критерієм.

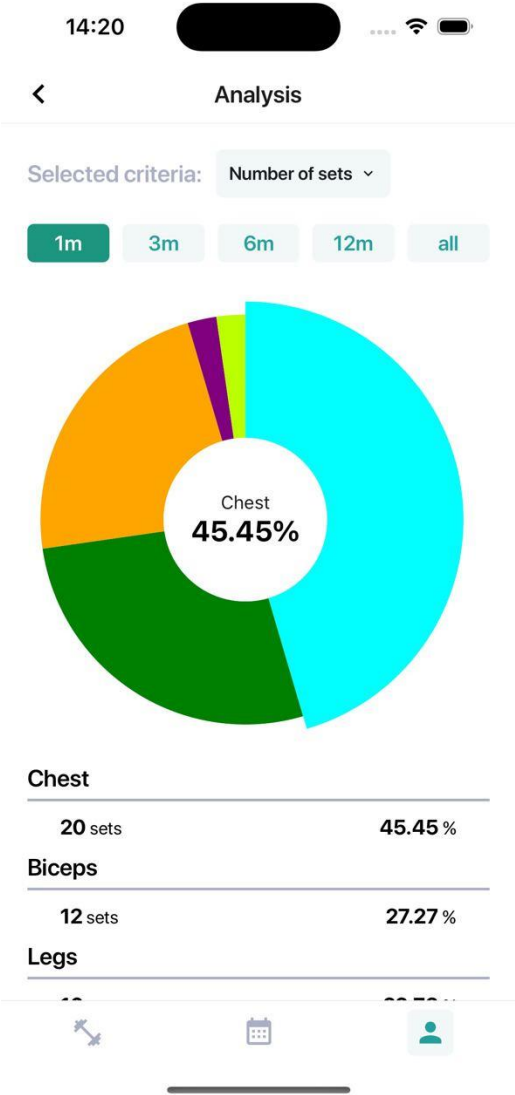


Рисунок 3.16 – Екран аналітики

Екран «Про застосунок» зазвичай містить загальну інформацію про сам додаток, таку як версія програми, історія змін, розробник, контактна інформація та можливо посилання на політику конфіденційності чи умови використання. Це дозволяє користувачам отримати більше інформації про додаток, перевірити оновлення, зв'язатися з розробниками або дізнатися про інші важливі аспекти використання застосунку.

3.3 Дані та сховище

В якості сховища використовується внутрішня пам'ять телефону. Для взаємодії із нею було використано бібліотеку `react-native-async-storage` [17].

Усі дані та стан додатку можна поділити на наступні частини (сховища):

- `timer` – містить поточний стан таймеру та методи для взаємодії із ним. Він не зберігає свій стан після закриття застосунку;
- `exercises` – містить дані про категорії (ім'я, опис, `id`) та вправи (ім'я, `id` категорії, власний `id`);
- `workouts` – містить дані про всі тренування які користувач додавав за весь час користування. Кожне тренування містить в собі `id` вправи, `id` категорії, масив підходів та дату запису;
- `bodyTracker` – містить інформацію про всі виміри ваги за весь час користування застосунком. Являє собою масив із записів, кожний з яких включає вказану вагу та дату.

Кожна із цих частин складається із наступних файлів:

- `slice.ts` – основний файл, де прописана логіка взаємодії із даними за допомогою редюсерів (`reducer`);
- `types.ts` – пов'язані типи, які описують схему даних кожної сутності, відповідні моделі;
- `selectors.ts` – містить селектори для вибірки даних із відповідної частини. Перевагою селекторів є те, що їх можна перевикористовувати у різних частинах додатку, оскільки доволі часто існує необхідність отримати одну і ту саму інформацію на різних екранах. Це дозволяє зменшити кількість коду та підвищити його читаємість;
- `index.ts` – «загальний» файл, який збирає в собі всі методи та сховища та експортує їх. Створений для більш зручного відображення імпортів у файлах, що використовують ті чи інші сховища.

Також модулі можуть містити деякі інші допоміжні файли. Узагальнена структура файлів зображена на рисунку 3.17.

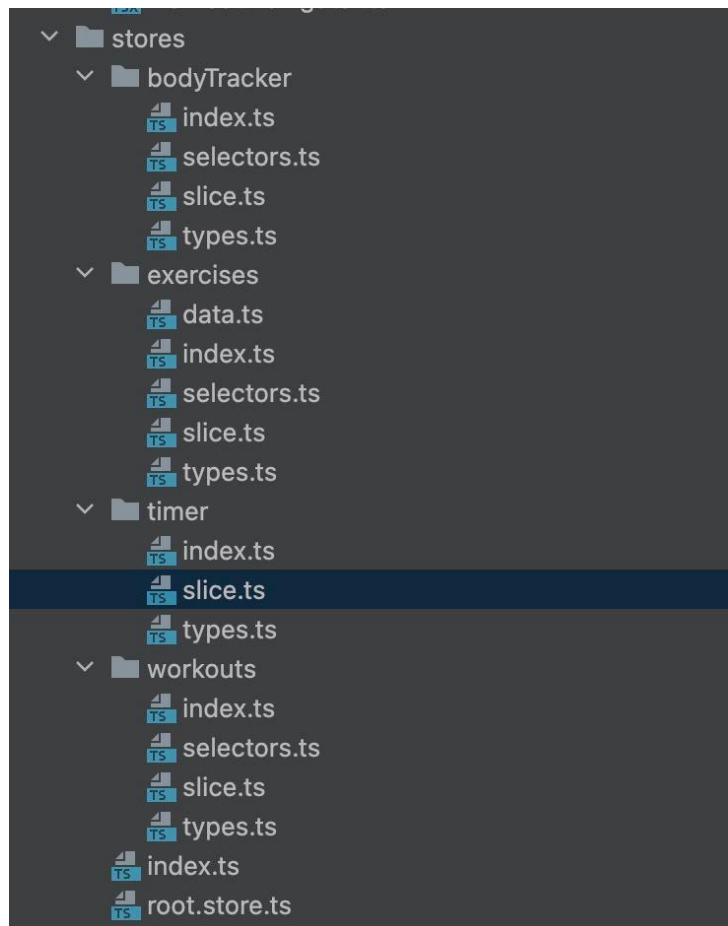


Рисунок 3.17 – Файлове дерево модулів сховища

3.4 Висновки за розділом

В даному розділі було наведено основну інформацію про розробку застосунку. Детально описано інтерфейс, структуру сховища та взаємодії із ним. Наведено опис деяких компонентів та їх функціоналу.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Розроблюваний застосунок призначений для зручного та швидкого відстеження тренувань. Програмне забезпечення виконує наступні функції:

- дає можливість обирати вправи за існуючими категоріями;
- дає можливість створювати власні категорії;
- дає можливість створювати власні вправи;
- дає можливість відстежувати прогрес за конкретною вправою;
- надає аналітику по всіх тренуваннях;
- пропонує сучасний, інтуїтивно зрозумілий інтерфейс.

Для запуску програми необхідно мати пристрій із ОС Android версії 6.0 або новіше, або пристрій із ОС iOS версії 12.4 або новіше. Також на пристрої повинно бути 40 МБ вільного місця у пам'яті.

4.2 Розгортання на локальному комп'ютері

Для подальшої роботи над застосунком, новому розробнику потрібно правильно налаштувати локальне оточення. Розробка навіть кросплатформених застосунків вимагає встановлення інструментів відповідної платформи.

Мінімальний список необхідних інструментів :

- оточення NodeJS;
- пакетний менеджер NPM;
- службу перегляду файлів Watchman;
- React Native command line interface.

Необхідні інструменти для запуску на ОС Android [18]:

- Java Development Kit;

- Android Studio.

Необхідні інструменти для запуску на ОС iOS [19]:

- обов'язково мати комп'ютер із встановленою ОС MacOS;
- Xcode версії 14.1;
- CocoaPods.

Після налаштування середовища потрібно виконати наступні дії у терміналі, в корінній папці проєкту:

- `npm install` – завантажує всі необхідні пакети, які використані в проєкті;
- `npm expo start` – запускає версію додатку для розробки.

В процесі розробки виникає потреба в тестуванні застосунку. Для цього можна використовувати емулятори мобільних пристроїв, чи реальний мобільний пристрій. Детальну інструкцію про те як це зробити можна знайти в офіційній документації React Native [20].

Якщо все середовище було налаштовано правильно, розробник повинен зібрати програму та встановити її на емулятор чи реальний пристрій.

Команди для запуску застосунку:

- ОС Android - `npm expo run:android`;
- ОС iOS - `npm expo run:ios`.

При модифікації конфігурації проєкту, чи встановленні нових бібліотек, що включають в себе нативний код, проєкт треба зібрати заново.

При правильному виконанні всіх кроків та успішній збірці – розробник побачить екран із індикатором завантаження додатку (рис. 4.1). Коли процес завантаження завершиться – відкриється домашня сторінка із вправами.

У випадку виникнення будь-яких помилок вони будуть відображені у консолі.

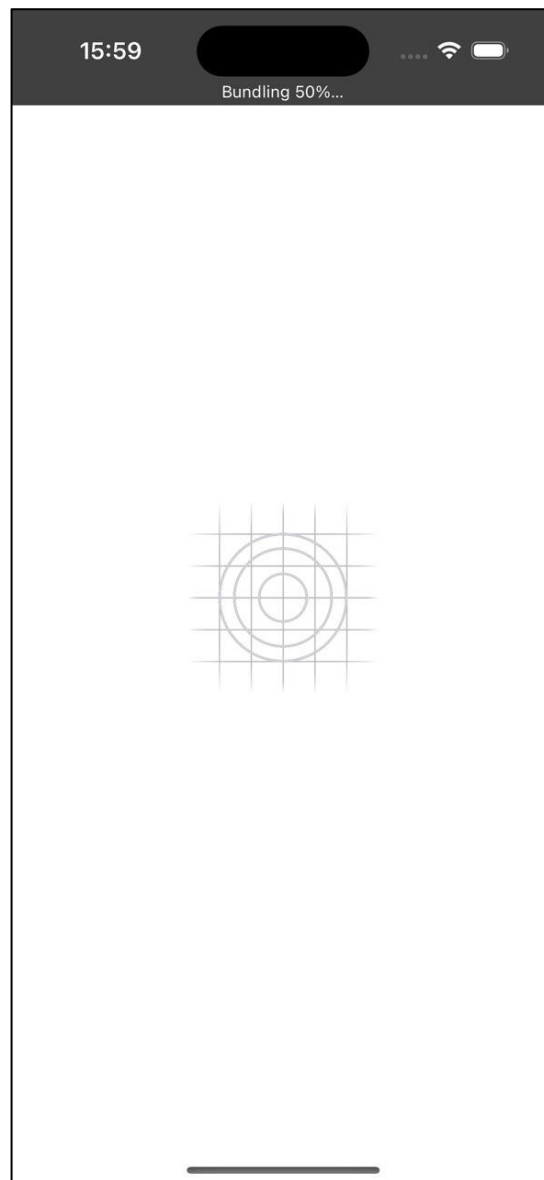


Рисунок 4.1 – Завантаження застосунку у версії для розробки

4.3 Інсталяція застосунку

Додавання застосунку у магазини відповідної платформи потребує коштів, тому на даному етапі було вирішено не робити цього. У документації React Native є детальна інструкція для публікації в Google Play Store [21] та Apple App Store [22].

Для встановлення застосунку на реальний телефон потрібно встановити .apk файл для Android, та .ipa файл для iOS.

Для створення .apk файлу розробнику потрібно перейти до директорії /android та запустити команду ./gradlew assembleRelease.

Для iOS встановити додаток напряму із файлу неможливо через обмеження безпеки самої ОС. Даний процес доволі складний і неочевидний, його опис не має сенсу в контексті даної записки. Детальний посібник з інструкціями для розповсюдження вказаний за посиланням [23].

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи бакалавра було розроблено повноцінний мобільний застосунок для відстеження та аналітики тренувань.

Розробку було виконано використовуючи фреймворк React Native та мову програмування TypeScript.

Обрані технології та архітектурні підходи забезпечують комплексне рішення для розробки нашого додатка, поєднуючи надійність, продуктивність і масштабованість. Вибір TypeScript підвищує якість коду і полегшує його підтримку, React Native забезпечує ефективну кросплатформенну розробку, а архітектура FLUX надає чітку і передбачувану структуру управління станом. Всі ці рішення разом створюють міцну основу для успішної реалізації проекту, дозволяючи зосередитися на створенні високоякісного, надійного і масштабованого продукту, який відповідає сучасним вимогам.

Тестування розробленого застосунку не виявило критичних помилок.

В результаті виконання завдання було отримано систему, яка відповідає зазначеним вимогам. Система є зручною у використанні та виконує усі необхідні функції.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. GymKeeper. Про цей додаток [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.kg.app.sportdiary>.
2. FitNotes. Про цей додаток [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.github.jamesgay.fitnotes>.
3. Home Workouts by BeStronger. Про цей додаток [Електронний ресурс]. – Режим доступу: <https://play.google.com/store/apps/details?id=com.shvagerfm.bestronge>.
4. What is a framework [Electronic resource]. – Access mode: <https://www.codecademy.com/resources/blog/what-is-a-framework/>.
5. What is React Native [Electronic resource]. – Access mode: [https://www.netguru.com/glossary/react-native#:~:text=React%20Native%20\(also%20known%20as,by%20using%20the%20same%20codebase](https://www.netguru.com/glossary/react-native#:~:text=React%20Native%20(also%20known%20as,by%20using%20the%20same%20codebase).
6. React Native hot reloading [Electronic resource]. – Access mode: <https://reactnative.dev/blog/2016/03/24/introducing-hot-reloading>.
7. About Flutter [Electronic resource]. – Access mode: <https://flutter.dev/>.
8. TypeScript vs JavaScript [Electronic resource]. – Access mode: <https://www.sencha.com/blog/typescript-vs-javascript-choosing-the-right-language-for-your-project/#:~:text=JavaScript%20and%20TypeScript%20are%20two,%2C%20and%20advanced%20type%2Dchecking>.
9. Переваги TypeScript над JavaScript [Електронний ресурс]. – Режим доступу: <https://drukarnia.com.ua/articles/typescript-ce-javascript-na-steroyidakh-uAxV2>.
10. FLUX introduction [Electronic resource]. – Access mode: <https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/>.
11. Flux in-depth overview [Electronic resource]. – Access mode:

<https://facebookarchive.github.io/flux/docs/in-depth-overview/>.

12. What is the Flux pattern? [Electronic resource]. – Access mode: https://miro.medium.com/v2/resize:fit:720/format:webp/1*6gzKauaLur48CJh-iwEFAQ.png.

13. FLUX у зображеннях [Електронний ресурс]. – Режим доступу: <https://habrastorage.org/r/w1560/files/87a/24c/3c1/87a24c3c10b944969d5ce58c9f7da406.png>.

14. What is Prettier [Electronic resource]. – Access mode: <https://prettier.io/docs/en/>.

15. ESLint core concepts [Electronic resource]. – Access mode: <https://eslint.org/docs/latest/use/core-concepts/>.

16. React hooks lifecycle [Electronic resource]. – Access mode: <https://github.com/Wavez/react-hooks-lifecycle>.

17. React Native Async Storage documentation [Electronic resource]. – Access mode: <https://github.com/react-native-async-storage/async-storage>.

18. React Native Android env setup [Electronic resource]. – Access mode: <https://reactnative.dev/docs/set-up-your-environment?platform=android>.

19. React Native iOS env setup [Electronic resource]. – Access mode: <https://reactnative.dev/docs/set-up-your-environment?platform=ios>.

20. Run React Native App on physical device [Electronic resource]. – Access mode: <https://reactnative.dev/docs/running-on-device>.

21. Generate signed Android APK [Electronic resource]. – Access mode: <https://reactnative.dev/docs/signed-apk-android>.

22. iOS App store publishing overview [Electronic resource]. – Access mode: <https://reactnative.dev/docs/publishing-to-app-store>.

23. Detailed iOS App distribution guide [Electronic resource]. – Access mode: <https://medium.com/@fadilah.arifki29/deployment-ios-in-app-store-react-native-d845e99e8329>.

ДОДАТОК А

Код програми

A.1.1 App.tsx

```
import { GestureHandlerRootView } from 'react-native-gesture-handler';
import { NavigationContainer } from '@react-navigation/native';
import {
  SafeAreaProvider,
  initialWindowMetrics,
} from 'react-native-safe-area-context';
import RootNavigator from './src/navigators/root.navigator';
import { AppRegistry, StyleSheet } from 'react-native';
import { Provider } from 'react-redux';
import { PersistGate } from 'redux-persist/integration/react';
import { store, persistor } from './src/stores';
import { PaperProvider } from 'react-native-paper';
import { useFonts } from 'expo-font';

AppRegistry.registerComponent('trackfit', () => App);

export default function App() {
  const [fontsLoaded] = useFonts({
    'SFProDisplay-Black': require('./assets/fonts/SFProDisplay-Black.otf'),
    'SFProDisplay-Light': require('./assets/fonts/SFProDisplay-Light.otf'),
    'SFProDisplay-Semibold': require('./assets/fonts/SFProDisplay-Semibold.otf'),
    'SFProDisplay-BlackItalic': require('./assets/fonts/SFProDisplay-BlackItalic.otf'),
    'SFProDisplay-LightItalic': require('./assets/fonts/SFProDisplay-LightItalic.otf'),
    'SFProDisplay-SemiboldItalic':
require('./assets/fonts/SFProDisplay-SemiboldItalic.otf'),
    'SFProDisplay-Bold': require('./assets/fonts/SFProDisplay-Bold.otf'),
    'SFProDisplay-Medium': require('./assets/fonts/SFProDisplay-Medium.otf'),
    'SFProDisplay-Thin': require('./assets/fonts/SFProDisplay-Thin.otf'),
    'SFProDisplay-BoldItalic': require('./assets/fonts/SFProDisplay-BoldItalic.otf'),
    'SFProDisplay-MediumItalic': require('./assets/fonts/SFProDisplay-MediumItalic.otf'),
    'SFProDisplay-ThinItalic': require('./assets/fonts/SFProDisplay-ThinItalic.otf'),
    'SFProDisplay-Heavy': require('./assets/fonts/SFProDisplay-Heavy.otf'),
    'SFProDisplay-Regular': require('./assets/fonts/SFProDisplay-Regular.otf'),
  });
```

```

    'SFProDisplay-Ultralight': require('./assets/fonts/SFProDisplay-
Ultralight.otf'),
    'SFProDisplay-HeavyItalic': require('./assets/fonts/SFProDisplay-
HeavyItalic.otf'),
    'SFProDisplay-RegularItalic':
require('./assets/fonts/SFProDisplay-RegularItalic.otf'),
    'SFProDisplay-UltralightItalic':
require('./assets/fonts/SFProDisplay-UltralightItalic.otf'),
  });
  if (!fontsLoaded) return null;
  return (
    <GestureHandlerRootView style={StyleSheet.absoluteFill}>
      <Provider store={store}>
        <PersistGate persistor={persistor}>
          <NavigationContainer>
            <PaperProvider>
              <SafeAreaProvider
                style={StyleSheet.absoluteFill}
                initialMetrics={initialWindowMetrics}>
                <RootNavigator />
              </SafeAreaProvider>
            </PaperProvider>
          </NavigationContainer>
        </PersistGate>
      </Provider>
    </GestureHandlerRootView>
  );
}

```

A.1.2 root.store.ts

```

import { combineReducers, configureStore } from '@reduxjs/toolkit';
import { setupListeners } from '@reduxjs/toolkit/query';
import { exercisesSlice } from './exercises';
import { TypedUseSelectorHook, useSelector, useDispatch } from 'react-
redux';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { persistReducer, persistStore } from 'redux-persist';
import {
  FLUSH,
  PAUSE,
  PERSIST,
  PURGE,
  REGISTER,
  REHYDRATE,
} from 'redux-persist/es/constants';
import { workoutsSlice } from './workouts';
import { timerSlice } from './timer';

```

```

import { bodyTrackerSlice } from './bodyTracker';

const rootPersistConfig = {
  key: 'root',
  storage: AsyncStorage,
  whitelist: ['exercises', 'workouts', 'bodyTracker'],
};

const rootReducer = combineReducers({
  exercises: exercisesSlice.reducer,
  workouts: workoutsSlice.reducer,
  timer: timerSlice.reducer,
  bodyTracker: bodyTrackerSlice.reducer,
});

const persistedReducer = persistReducer(rootPersistConfig, rootReducer);

export const store = configureStore({
  reducer: persistedReducer,
  middleware: getDefaultMiddleware =>
    getDefaultMiddleware({
      serializableCheck: {
        ignoredActions: [FLUSH, REHYDRATE, PAUSE, PERSIST, PURGE,
REGISTER],
      },
    }),
});

export const persistor = persistStore(store);

setupListeners(store.dispatch);

export type RootState = ReturnType<typeof store.getState>;
export type AppDispatch = typeof store.dispatch;
export type AppThunkConfig = {
  state: RootState;
  dispatch: AppDispatch;
};

export const useAppDispatch = () => useDispatch<AppDispatch>();
export const useAppSelector: TypedUseSelectorHook<RootState> =
useSelector;

```

A.1.3 bodyTracker.slice.ts

```

import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { BodyTrackerState, WeightWithTimestamp } from './types';

```

```

const initialState: BodyTrackerState = {
  weight: [],
};

export const bodyTrackerSlice = createSlice({
  name: 'bodyTracker',
  initialState: initialState,
  reducers: {
    logWeight: (
      state: BodyTrackerState,
      action: PayloadAction<WeightWithTimestamp>,
    ): BodyTrackerState => ({
      ...state,
      weight: [...state.weight, action.payload],
    }),
    updateWeight: (
      state: BodyTrackerState,
      action: PayloadAction<WeightWithTimestamp>,
    ): BodyTrackerState => {
      const index = state.weight.findIndex(
        entry => entry.date === action.payload.date,
      );
      if (index !== -1) {
        const updatedWeight = [...state.weight];
        updatedWeight[index] = action.payload;
        return { ...state, weight: updatedWeight };
      }
      return state;
    },
    deleteWeight: (
      state: BodyTrackerState,
      action: PayloadAction<string>,
    ): BodyTrackerState => {
      const filteredWeight = state.weight.filter(
        entry => entry.date !== action.payload,
      );
      return {
        ...state,
        weight: filteredWeight,
      };
    },
  },
});

export const { logWeight, updateWeight, deleteWeight } =
  bodyTrackerSlice.actions;

export type BodyTrackerState = {

```

```

    weight: WeightWithTimestamp[];
  };

export type WeightWithTimestamp = {
  weight: number;
  date: string;
};

```

A.1.4 exercises.slice.ts

```

import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import { CreateCategoryPayload, ExercisesSlice } from './types';
import {
  ChestExercises,
  BackExercises,
  BicepsExercises,
  LegsExercises,
  ShouldersExercises,
  TricepsExercises,
  AbsExercises,
} from './data';
import { getTodayDate } from '../../utils';

const defaultState: ExercisesSlice = {
  categories: [
    {
      id: 'abs',
      name: 'categories.abs',
      exercises: AbsExercises,
    },
    {
      id: 'back',
      name: 'categories.back',
      exercises: BackExercises,
    },
    {
      id: 'biceps',
      name: 'categories.biceps',
      exercises: BicepsExercises,
    },
    {
      id: 'chest',
      name: 'categories.chest',
      exercises: ChestExercises,
    },
    {
      id: 'legs',
      name: 'categories.legs',

```

```

    exercises: LegsExercises,
  },
  {
    id: 'shoulders',
    name: 'categories.shoulders',
    exercises: ShouldersExercises,
  },
  {
    id: 'triceps',
    name: 'categories.triceps',
    exercises: TricepsExercises,
  },
],
};

export const exercisesSlice = createSlice({
  name: 'categories',
  initialState: defaultState,
  reducers: {
    createCategory: (
      state: ExercisesSlice,
      action: PayloadAction<CreateCategoryPayload>,
    ) => {
      const updatedCategories = [
        ...state.categories,
        {
          id: `${action.payload.name}${getTodayDate()}`,
          exercises: [],
          name: action.payload.name,
          isCustom: true,
        },
      ].sort((a, b) => a.name.localeCompare(b.name));
      return {
        ...state,
        categories: updatedCategories,
      };
    },
  },
});

export const { createCategory } = exercisesSlice.actions;

export type ExercisesSlice = {
  categories: ExerciseCategory[];
};

export type ExerciseCategory = {
  id: string;

```

```

    name: string;
    exercises: DefaultExercise[];
    isCustom?: boolean;
  };

export type DefaultExercise = {
  id: string;
  name: string;
  categoryId: string;
  type: ExerciseType;
};

export type ExerciseType =
  | 'weight_and_reps'
  | 'distance_and_time'
  | 'weight_and_time';

export type CreateCategoryPayload = {
  name: string
}

import { createSelector } from '@reduxjs/toolkit';
import { RootState } from '../root.store';

export const getCategoryById = createSelector(
  [
    (state: RootState) => state.exercises.categories,
    (_, categoryId: string) => categoryId,
  ],
  (categories, categoryId) => {
    return categories.find(category => category.id === categoryId);
  },
);

```

A.1.5 timer.slice.ts

```

import { createSlice } from '@reduxjs/toolkit';
import { TimerState } from './types';

const defaultState: TimerState = {
  isTimerVisible: false,
  isRunning: false,
  duration: 0, // Duration in seconds
  remainingTime: 0,
};

export const timerSlice = createSlice({
  name: 'timer',

```

```

    initialState: defaultState,
    reducers: {
      showTimer: (state): TimerState => ({ ...state, isTimerVisible: true
    })),
      hideTimer: (state): TimerState => ({ ...state, isTimerVisible: false
    })),
      setDuration: (state: TimerState, action) => ({
        ...state,
        duration: action.payload,
        remainingTime: action.payload,
      }),
      startTimer: (state: TimerState): TimerState => ({
        ...state,
        isRunning: true,
      }),
      stopTimer: (state: TimerState): TimerState => ({
        ...state,
        isRunning: false,
      }),
      pauseTimer: (state: TimerState): TimerState => ({
        ...state,
        isRunning: false,
      }),
      resetTimer: (state: TimerState): TimerState => ({
        ...state,
        isRunning: false,
        remainingTime: state.duration,
      }),
      decrementTime: (state: TimerState): TimerState => {
        if (state.remainingTime > 0) {
          return { ...state, remainingTime: state.remainingTime - 1 };
        } else {
          return { ...state, isRunning: false };
        }
      },
    },
  }));

export const {
  showTimer,
  hideTimer,
  setDuration,
  stopTimer,
  startTimer,
  resetTimer,
  decrementTime,
  pauseTimer,
} = timerSlice.actions;

```

```
export type TimerState = {
  isTimerVisible: boolean;
  isRunning: boolean;
  duration: number;
  remainingTime: number;
};
```

A.1.6 workouts.slice.ts

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit';
import {
  CreateSetPayload,
  DeleteSetFromExercisePayload,
  UpdateSetPayload,
  WorkoutsSlice,
} from './types';
import dayjs from 'dayjs';

const defaultState: WorkoutsSlice = {};

export const workoutsSlice = createSlice({
  name: 'workouts',
  initialState: defaultState,
  reducers: {
    logNewSet: (
      state: WorkoutsSlice,
      action: PayloadAction<CreateSetPayload>,
    ) => {
      const { exerciseId, set: newSet, categoryId, date } =
        action.payload;

      const workoutByDate = state[date];

      // if first exercise
      if (!workoutByDate) {
        return {
          ...state,
          [date]: {
            [exerciseId]: {
              exerciseId,
              categoryId,
              sets: [newSet],
              createdAt: dayjs().toISOString(),
            },
          },
        };
      }
    },
  },
});
```

```

    if (workoutByDate) {
      const targetExercise = workoutByDate[exerciseId];
      if (!targetExercise) {
        // log new exercise
        return {
          ...state,
          [date]: {
            ...state[date],
            [exerciseId]: {
              exerciseId,
              categoryId,
              sets: [newSet],
              createdAt: dayjs().toISOString(),
            },
          },
        };
      } else {
        // update existing
        return {
          ...state,
          [date]: {
            ...state[date],
            [exerciseId]: {
              ...targetExercise,
              sets: [...targetExercise.sets, newSet],
            },
          },
        };
      }
    }

    return state;
  },
  updateSet: (
    state: WorkoutsSlice,
    action: PayloadAction<UpdateSetPayload>,
  ) => {
    const { exerciseId, set, date, setIndex } = action.payload;
    const workoutByDate = state[date];
    if (workoutByDate) {
      const exercise = workoutByDate[exerciseId];
      const updatedSets = [...exercise.sets];
      updatedSets[setIndex] = { ...set, createdAt: exercise.createdAt
};

    return {
      ...state,

```

```

        [date]: {
            ...state[date],
            [exerciseId]: { ...exercise, sets: updatedSets },
        },
    };
}
},
deleteSetFromExercise: (
    state: WorkoutsSlice,
    action: PayloadAction<DeleteSetFromExercisePayload>,
) => {
    const { date, exerciseId, setIndex } = action.payload;
    const workoutByDate = state[date];
    if (workoutByDate) {
        const exercise = workoutByDate[exerciseId];
        const updatedSets = exercise.sets.filter(
            (_, index) => index !== setIndex,
        );

        return {
            ...state,
            [date]: {
                ...state[date],
                [exerciseId]: {
                    ...exercise,
                    sets: updatedSets,
                },
            },
        };
    }
},
});

export const { logNewSet, deleteSetFromExercise, updateSet } =
    workoutsSlice.actions;

export type WorkoutsSlice = {
    [dateKey: string]: { [exerciseId: string]: Workout };
};

export type Workout = {
    exerciseId: string;
    categoryId: string;
    sets: Set[];
    createdAt: string;
};

```

```

export type Set = {
  weight: number;
  reps: number;
  createdAt: string;
};

export type DeleteSetFromExercisePayload = {
  date: string;
  exerciseId: string;
  setIndex: number;
};

export type CreateSetPayload = {
  exerciseId: string;
  categoryId: string;
  set: Set;
  date: string;
};

export type UpdateSetPayload = {
  exerciseId: string;
  date: string;
  setIndex: number;
  set: Set;
};

export enum GraphPeriod {
  '1m' = '1m',
  '3m' = '3m',
  '6m' = '6m',
  '12m' = '12m',
  'all' = 'all',
}

export enum GraphCriteria {
  'MAX_WEIGHT' = 'MAX_WEIGHT',
  'MAX_REPS' = 'MAX_REPS',
  'WORKOUT_REPS' = 'WORKOUT_REPS',
}

export enum AnalysisCriteria {
  NUMBER_OF_SETS = 'number_of_sets',
  NUMBER_OF_REPS = 'number_of_reps',
  NUMBER_OF_WORKOUTS = 'number_of_workouts',
}

import { createSelector } from '@reduxjs/toolkit';
import { RootState } from '../root.store';

```

```

import { getTodayDate } from '../utils';
import {
  AnalysisCriteria,
  GraphCriteria,
  GraphPeriod,
  Set,
  WorkoutsSlice,
} from './types';
import dayjs from 'dayjs';

var isBetween = require('dayjs/plugin/isBetween');
dayjs.extend(isBetween);

export const getTodayWorkout = createSelector(
  [(state: RootState) => state.workouts],
  workouts => workouts[getTodayDate()],
);

export const getTodayWorkoutByExerciseId = createSelector(
  [(state: RootState) => state.workouts, (_, exerciseId) => exerciseId],
  (workouts, exerciseId) => {
    const todayWorkout = workouts[getTodayDate()];
    if (todayWorkout) {
      return todayWorkout[exerciseId];
    }
  },
);

export const getWorkoutByDate = createSelector(
  [(state: RootState) => state.workouts, (_, date: string) => date],
  (workouts, date) => workouts[date],
);

export const getWorkoutByExerciseIdForDate = createSelector(
  [
    (state: RootState) => state.workouts,
    (_, date: string) => date,
    (_, _d, exerciseId) => exerciseId,
  ],
  (workouts, date, exerciseId) => {
    const todayWorkout = workouts[date];
    if (todayWorkout) {
      return todayWorkout[exerciseId];
    }
  },
);

export const getExerciseHistory = createSelector(

```

```

[(state: RootState) => state.workouts, (_, exerciseId) => exerciseId],
(workouts, exerciseId) => {
  const exerciseSetsByDate: Record<string, Set[]> = {};
  Object.entries(workouts).forEach(([date, exercises]) => {
    const sets = Object.values(exercises)
      .filter(workout => workout.exerciseId === exerciseId)
      .map(workout => workout.sets)
      .flat(); // Flatten arrays of sets
    if (sets.length > 0) {
      exerciseSetsByDate[date] = sets;
    }
  });

  // Convert keys (dates) to Date objects and sort them in descending
order
  const sortedDates = Object.keys(exerciseSetsByDate)
    .map(date => new Date(date))
    .sort((a, b) => b.getTime() - a.getTime())
    .map(date => date.toISOString().split('T')[0]); // Convert back to
ISO string

  // Create a new object with sorted dates
  const sortedExerciseSetsByDate: Record<string, Set[]> = {};
  sortedDates.forEach(date => {
    sortedExerciseSetsByDate[date] = exerciseSetsByDate[date];
  });

  return sortedExerciseSetsByDate;
},
);

export const getExerciseHistoryForPeriod = createSelector(
[
  (state: RootState) => state.workouts,
  (_, exerciseId: string) => exerciseId,
  (_, __, dateRange: GraphPeriod) => dateRange, // Additional parameter
for date range
  (_, __, ___, selectionCriteria: GraphCriteria) => selectionCriteria,
// Additional parameter for selection criteria
],
(workouts, exerciseId, dateRange, selectionCriteria) => {
  const exerciseData: Record<string, number> = {};

  Object.entries(workouts).forEach(([date, exercises]) => {
    const currentDate = new Date();
    const workoutDate = new Date(date);
    const timeDiff = currentDate.getTime() - workoutDate.getTime();
    const diffMonths = Math.floor(timeDiff / (1000 * 3600 * 24 * 30));

```

```

    if (
      (dateRange === GraphPeriod['1m'] && diffMonths <= 1) ||
      (dateRange === GraphPeriod['3m'] && diffMonths <= 3) ||
      (dateRange === GraphPeriod['6m'] && diffMonths <= 6) ||
      (dateRange === GraphPeriod['12m'] && diffMonths <= 12) ||
      dateRange === GraphPeriod.all
    ) {
      Object.values(exercises)
        .filter(workout => workout.exerciseId === exerciseId)
        .forEach(workout => {
          const sets = workout.sets;
          let value = 0;

          if (selectionCriteria === GraphCriteria.MAX_REPS) {
            value = Math.max(...sets.map(set => set.reps));
          } else if (selectionCriteria === GraphCriteria.MAX_WEIGHT) {
            value = Math.max(...sets.map(set => set.weight));
          } else if (selectionCriteria === GraphCriteria.WORKOUT_REPS)
{
            value = sets.reduce((acc, set) => acc + set.reps, 0);
          }

          exerciseData[date] = value;
        });
    }
  });

  // Convert keys (dates) to Date objects and sort them in descending
  order
  const sortedDates = Object.keys(exerciseData)
    .map(date => new Date(date))
    .sort((a, b) => b.getTime() - a.getTime())
    .map(date => date.toISOString().split('T')[0]); // Convert back to
  ISO string

  // Create a new object with sorted dates and corresponding values
  const sortedExerciseData: Record<string, number> = {};
  sortedDates.forEach(date => {
    sortedExerciseData[date] = exerciseData[date];
  });

  return sortedExerciseData;
},
);

export const getTotalWorkouts = createSelector(
  [(state: RootState) => state.workouts],

```

```

    workouts => {
      return Object.keys(workouts).length;
    },
  );

export const getTotalSets = createSelector(
  [(state: RootState) => state.workouts],
  workouts => {
    let totalSets = 0;
    Object.values(workouts).forEach(workoutByDate => {
      Object.values(workoutByDate).forEach(exercise => {
        totalSets += exercise.sets.length;
      });
    });
    return totalSets;
  },
);

export const getTotalReps = createSelector(
  (state: RootState) => state.workouts,
  workouts => {
    let totalReps = 0;
    Object.values(workouts).forEach(workoutByDate => {
      Object.values(workoutByDate).forEach(exercise => {
        exercise.sets.forEach(set => {
          totalReps += set.reps;
        });
      });
    });
    return totalReps;
  },
);

export const getTotalVolume = createSelector(
  (state: RootState) => state.workouts,
  workouts => {
    let totalVolume = 0;
    Object.values(workouts).forEach(workoutByDate => {
      Object.values(workoutByDate).forEach(exercise => {
        exercise.sets.forEach(set => {
          totalVolume += set.weight * set.reps;
        });
      });
    });
    return totalVolume;
  },
);

```

```

// Utility function to filter workouts by period
const filterWorkoutsByPeriod = (
  workouts: WorkoutsSlice,
  period: GraphPeriod,
) => {
  const endDate = dayjs();
  let startDate;

  switch (period) {
    case GraphPeriod['1m']:
      startDate = endDate.subtract(1, 'month');
      break;
    case GraphPeriod['3m']:
      startDate = endDate.subtract(3, 'month');
      break;
    case GraphPeriod['6m']:
      startDate = endDate.subtract(6, 'month');
      break;
    case GraphPeriod['12m']:
      startDate = endDate.subtract(12, 'month');
      break;
    case GraphPeriod['all']:
    default:
      startDate = dayjs('1970-01-01'); // Start from epoch time
      break;
  }

  const filteredWorkouts: WorkoutsSlice = {};

  Object.keys(workouts).forEach(dateKey => {
    const workoutDate = dayjs(dateKey);
    if (workoutDate.isBetween(startDate, endDate, null, '[')) {
      filteredWorkouts[dateKey] = workouts[dateKey];
    }
  });

  return filteredWorkouts;
};

export const getSetsPercentageByCategory = createSelector(
  (state: RootState) => state.workouts,
  (state: RootState, period: GraphPeriod) => period,
  (state: RootState, period: GraphPeriod, criteria: AnalysisCriteria) =>
    criteria,
  (workouts, period, criteria) => {
    const filteredWorkouts = filterWorkoutsByPeriod(workouts, period);
    const categoryMetrics: { [categoryId: string]: number } = {};
    let totalValue = 0;

```

```

Object.values(filteredWorkouts).forEach(workoutByDate => {
  Object.values(workoutByDate).forEach(exercise => {
    const categoryId = exercise.categoryId;
    let valueToAdd = 0;

    switch (criteria) {
      case AnalysisCriteria.NUMBER_OF_SETS:
        valueToAdd = exercise.sets.length;
        break;
      case AnalysisCriteria.NUMBER_OF_REPS:
        valueToAdd = exercise.sets.reduce((sum, set) => sum +
set.reps, 0);
        break;
      case AnalysisCriteria.NUMBER_OF_WORKOUTS:
        valueToAdd = 1; // each exercise counts as one workout
        break;
    }

    if (!categoryMetrics[categoryId]) {
      categoryMetrics[categoryId] = 0;
    }

    categoryMetrics[categoryId] += valueToAdd;
    totalValue += valueToAdd;
  });
});

return Object.entries(categoryMetrics).map(([categoryId, value]) =>
({
  categoryId,
  value,
  percentage: totalValue > 0 ? (value / totalValue) * 100 : 0,
}));
},
);

```

A.1.7 profile.navigator.tsx

```

import { createStackNavigator } from '@react-navigation/stack';
import { ProfileNavigatorStackParamList } from '../types';
import { ProfileScreens } from '../types';
import ProfileScreen from '../modules/profile/profile.screen';
import AboutScreen from '../modules/profile/screens/about.screen';
import StatsScreen from '../modules/profile/screens/stats.screen';
import AnalysisScreen from '../modules/profile/screens/analysis.screen';
import BodyTrackerScreen from
'../modules/profile/screens/bodyTracker.screen';

```

```

import { BackNavigationButton } from '../components';
import { Fonts } from '../consts';

const Stack = createStackNavigator<ProfileNavigatorStackParamList>();

const ProfileNavigator = () => {
  return (
    <Stack.Navigator
      initialRouteName={ProfileScreens.PROFILE_HOME}
      screenOptions={{
        headerBackTitleVisible: false,
        headerBackImage: BackNavigationButton,
        headerTitleStyle: {
          ...Fonts.semibold_18,
          textTransform: 'capitalize',
        },
      }}>
      <Stack.Screen
        name={ProfileScreens.PROFILE_HOME}
        component={ProfileScreen}
        options={{
          headerTitle: 'Profile',
        }}
      />
      <Stack.Screen
        name={ProfileScreens.ABOUT_APP}
        component={AboutScreen}
        options={{ headerTitle: 'About app' }}
      />
      <Stack.Screen
        name={ProfileScreens.STATS}
        component={StatsScreen}
        options={{ headerTitle: 'Statistics' }}
      />
      <Stack.Screen
        name={ProfileScreens.ANALYSIS}
        component={AnalysisScreen}
        options={{ headerTitle: 'Analysis' }}
      />
      <Stack.Screen
        name={ProfileScreens.BODY_TRACKER}
        component={BodyTrackerScreen}
        options={{ headerTitle: 'Body tracker' }}
      />
    </Stack.Navigator>
  );
};

```

```
export default ProfileNavigator;
```

A.1.8 tab.navigator.tsx

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { TabScreens } from '../types';
import CalendarScreen from '../modules/calendar/calendar.screen';
import WorkoutNavigator from './workout.navigator';
import { TabNavigatorParamList } from './types';
import { Icon } from 'react-native-paper';
import { Colors, Fonts } from '../consts';
import { View } from 'react-native';
import { TimerModal } from '../components';
import ProfileNavigator from './profile.navigator';

const Tab = createBottomTabNavigator<TabNavigatorParamList>();

const TabBarIcon = (props: {
  focused: boolean;
  color: string;
  size: number;
  source: string;
}) => (
  <View
    style={{
      backgroundColor: props.focused ? Colors.brand.background :
'transparent',
      padding: 5,
      borderRadius: 5,
    }}>
    <Icon size={props.size} color={props.color} source={props.source} />
  </View>
);

const TabNavigator = () => {
  return (
    <>
      <Tab.Navigator
        screenOptions={{
          tabBarShowLabel: false,
          tabBarActiveTintColor: Colors.brand.default,
          tabBarInactiveTintColor: Colors.neutral.ghost,
          headerTitleStyle: {
            ...Fonts.semibold_18,
            textTransform: 'capitalize',
          },
        }}>
        <Tab.Screen
```

```

        name={TabScreens.WORKOUT}
        component={WorkoutNavigator}
        options={{
          headerShown: false,
          tabBarIcon: props => <TabBarIcon {...props}
source={'dumbbell'} />,
        }}
      />
    <Tab.Screen
      name={TabScreens.CALENDAR}
      component={CalendarScreen}
      options={{
        tabBarIcon: props => (
          <TabBarIcon {...props} source={'calendar-month'} />
        ),
      }}
    />
    <Tab.Screen
      name={TabScreens.PROFILE}
      component={ProfileNavigator}
      options={{
        headerShown: false,
        tabBarIcon: props => <TabBarIcon {...props}
source={'account'} />,
      }}
    />
  </Tab.Navigator>
  <TimerModal />
</>
);
};

export default TabNavigator;

```

A.1.9 workout.navigator.tsx

```

import { createStackNavigator } from '@react-navigation/stack';
import { WorkoutNavigatorStackParamList } from '../types';
import { WorkoutScreens } from '../types';
import {
  Home,
  CategoriesScreen,
  ExercisesScreen,
  WorkoutScreen,
  WorkoutLog, HistoryNavigator,
} from '../modules/workout';
import { i18n } from '../i18n';
import { BackNavigationButton } from '../components';

```

```

import { Fonts } from '../consts';
import { getTodayDate } from '../utils';

const Stack = createStackNavigator<WorkoutNavigatorStackParamList>();

const WorkoutNavigator = () => {
  return (
    <Stack.Navigator
      initialRouteName={WorkoutScreens.HOME}
      screenOptions={{
        headerBackTitleVisible: false,
        headerBackImage: BackNavigationButton,
        headerTitleStyle: {
          ...Fonts.semibold_18,
          textTransform: 'capitalize',
        },
      }}>
      <Stack.Screen
        name={WorkoutScreens.HOME}
        component={Home}
        initialParams={{
          date: getTodayDate(),
        }}
        options={{
          headerTitle: 'Today',
          headerTitleAlign: 'left',
        }}
      />
      <Stack.Screen
        name={WorkoutScreens.CATEGORIES}
        component={CategoriesScreen}
        options={{
          headerTitle: i18n.t('choose_category'),
        }}
      />
      <Stack.Screen
        name={WorkoutScreens.EXERCISES}
        component={ExercisesScreen}
        options={{
          headerTitle: i18n.t('choose_exercise'),
        }}
      />
      <Stack.Screen
        name={WorkoutScreens.LOG_WORKOUT}
        component={WorkoutScreen}
        options={{
          headerTitle: i18n.t('track'),
        }}
      />
    </Stack.Navigator>
  );
};

```

```

    />
    <Stack.Screen
      name={WorkoutScreens.HISTORY}
      component={HistoryNavigator}
      options={{
        headerTitle: i18n.t('history'),
      }}
    />
  </Stack.Navigator>
);
};

export default WorkoutNavigator;

```

A.1.10 navigators.types.tsx

```

import {
  BodyTrackerScreens,
  HistoryScreens,
  LoginScreens,
  ProfileScreens,
  RootRoutes,
  TabScreens,
  WorkoutScreens,
} from '../types';

export type LoginNavigatorStackParamList = {
  [LoginScreens.WELCOME]: undefined;
  [LoginScreens.EMAIL_LOGIN]: undefined;
  [LoginScreens.REGISTER]: undefined;
};

export type RootNavigatorStackParamList = {
  [RootRoutes.LOGIN]: undefined;
  [RootRoutes.TAB]: undefined;
};

export type TabNavigatorParamList = {
  [TabScreens.WORKOUT]: undefined;
  [TabScreens.CALENDAR]: undefined;
  [TabScreens.PROFILE]: undefined;
};

export type WorkoutNavigatorStackParamList = {
  [WorkoutScreens.HOME]: {
    date: string;
  };
  [WorkoutScreens.CATEGORIES]: {

```

```

    date: string;
  };
  [WorkoutScreens.EXERCISES]: {
    date: string;
    categoryId: string;
  };
  [WorkoutScreens.LOG_WORKOUT]: {
    categoryId: string;
    exerciseId: string;
    date: string;
  };
  [WorkoutScreens.HISTORY]: {
    exerciseId: string;
    categoryId: string;
  };
};

export type HistoryNavigatorParamList = {
  [HistoryScreens.LOG]: {
    exerciseId: string;
    categoryId: string;
  };
  [HistoryScreens.GRAPH]: {
    exerciseId: string;
    categoryId: string;
  };
};

export type ProfileNavigatorStackParamList = {
  [ProfileScreens.PROFILE_HOME]: undefined;
  [ProfileScreens.BODY_TRACKER]: undefined;
  [ProfileScreens.STATS]: undefined;
  [ProfileScreens.ANALYSIS]: undefined;
  [ProfileScreens.ABOUT_APP]: undefined;
};

export type BodyTrackerNavigatorParamList = {
  [BodyTrackerScreens.LOG]: undefined;
  [BodyTrackerScreens.GRAPH]: undefined;
};

```

A.1.11 calendar.screen.tsx

```

import React, { useCallback, useMemo, useState } from 'react';
import { ScrollView, StyleSheet, Text } from 'react-native';
import { PageWrapper } from '../../components';
import { DateData, Calendar } from 'react-native-calendars';

```

```

import { useAppSelector } from '../..//stores';
import { getTodayDate } from '../..//utils';
import { MarkedDates } from 'react-native-calendars/src/types';
import { Colors, FontFamily, Fonts } from '../..//consts';
import { WorkoutByDate } from '../components/WorkoutByDate';

const CalendarScreen = () => {
  const [selectedDay, setSelectedDay] = useState<string>(getTodayDate());
  const workouts = useAppSelector(state => state.workouts);
  const markedDates = useMemo(() => {
    const baseObject = Object.keys(workouts).reduce(
      (acc: MarkedDates, curr) => ({
        ...acc,
        [curr]: {
          marked: true,
        },
      }),
      {}
    );
    baseObject[selectedDay] = {
      ...baseObject[selectedDay],
      selected: true,
      selectedColor: Colors.brand.outline,
      selectedTextColor: Colors.neutral.primary,
    };
    return baseObject;
  }, [workouts, selectedDay]);

  const onDayPress = useCallback((date: DateData) => {
    setSelectedDay(date.dateString);
  }, []);

  return (
    <PageWrapper>
      <ScrollView style={styles.root}>
        <Calendar
          markedDates={markedDates}
          onDayPress={onDayPress}
          enableSwipeMonths={true}
          allowSelectionOutOfRange={false}
          current={selectedDay}
          initialDate={selectedDay}
          maxDate={getTodayDate()}
          selectedDay={selectedDay}
          theme={{
            backgroundColor: Colors.neutral.background,
            textDayFontFamily: FontFamily.SFProDisplay_Bold,
            textMonthFontFamily: FontFamily.SFProDisplay_Bold,
          }}
        />
      </ScrollView>
    </PageWrapper>
  );
};

```

```

        textDayHeaderFontFamily: FontFamily.SFProDisplay_Bold,
        todayTextColor: Colors.brand.dark,
        todayBackgroundColor: Colors.neutral.line,
        dotColor: Colors.brand.dark,
        selectedDotColor: Colors.brand.dark,
        arrowColor: Colors.brand.dark,
      }}
    />
    {selectedDay ? (
      <WorkoutByDate date={selectedDay} />
    ) : (
      <Text style={styles.selectDay}>Select day first</Text>
    )}
  </ScrollView>
</PageWrapper>
);
};

const styles = StyleSheet.create({
  root: { flex: 1 },
  selectDay: {
    ...Fonts.semibold_24,
    color: Colors.neutral.secondary,
    alignSelf: 'center',
    marginBottom: 'auto',
    marginTop: 'auto',
    marginHorizontal: 20,
    textAlign: 'center',
  },
});

export default CalendarScreen;

```

A.1.12 WorkoutByDate.tsx

```

import React, { useCallback, useMemo } from 'react';
import { StyleSheet, Text, TouchableOpacity, View } from 'react-native';
import { Colors, Fonts } from '../../const';
import { useAppSelector } from '../../stores';
import { getWorkoutByDate, Workout } from '../../stores/workouts';
import { getExerciseDetails } from '../../workout/utills';
import { i18n } from '../../i18n';
import { EmptyList } from '../../workout/components';
import dayjs from 'dayjs';
import { useNavigation } from '@react-navigation/native';
import { TabScreens, WorkoutScreens } from '../../types';
import { BottomTabNavigationProp } from '@react-navigation/bottom-tabs';
import { TabNavigatorParamList } from '../../navigators/types';

```

```

import { formatWeight } from '../../utils';
import { IconButton } from 'react-native-paper';

type WorkoutByDateProps = {
  date: string;
};

const ExerciseInfo = ({ workout }: { workout: Workout }) => {
  const exerciseDetails = useAppSelector(state =>
    getExerciseDetails(state, workout.categoryId, workout.exerciseId),
  );

  return (
    <View>
      <View style={styles.header}>
        <Text style={styles.exerciseName}>
          {exerciseDetails?.name ? i18n.t(exerciseDetails.name) : 'Not
found'}
        </Text>
      </View>
      <View style={styles.setsContainer}>
        {workout.sets.map(set => (
          <View key={set.createdAt} style={styles.setRow}>
            <Text style={Fonts.bold_16}>
              {formatWeight(set.weight)}
            <Text style={Fonts.regular_14}> kgs</Text>
          </Text>
            <Text style={Fonts.bold_16}>
              {set.reps} <Text style={Fonts.regular_14}>reps</Text>
            </Text>
          </View>
        ))}
      </View>
    </View>
  );
};

export const WorkoutByDate = ({ date }: WorkoutByDateProps) => {
  const navigation =
    useNavigation<BottomTabNavigationProp<TabNavigatorParamList>>();
  const workout = useAppSelector(state => getWorkoutByDate(state, date));
  const data = useMemo(
    () => (workout ? Object.values(workout) : []),
    [workout],
  );
  const openWorkoutForDate = useCallback(() => {
    navigation.navigate(TabScreens.WORKOUT, {
      screen: WorkoutScreens.HOME,
    });
  });
};

```

```

    params: {
      date: date,
    },
  });
}, [navigation, date]);

const hasData = useMemo(() => data && data.length, [data]);

const renderInfo = () => {
  if (hasData) {
    return data.map(workout => (
      <ExerciseInfo key={workout.createdAt} workout={workout} />
    ));
  }
  return (
    <View
      style={{
        alignSelf: 'center',
        marginTop: 'auto',
        marginBottom: 'auto',
        gap: 20,
      }}>
      <EmptyList message={i18n.t('empty_workout_log')} />
      <TouchableOpacity style={styles.actionBtn}
        onPress={openWorkoutForDate}>
        <Text style={styles.openBtnText}>Add workout</Text>
      </TouchableOpacity>
    </View>
  );
};

return (
  <View style={styles.modalContainer}>
    <View style={styles.infoHeader}>
      <Text style={Fonts.semibold_24}>
        {dayjs(date).format('dddd, MMMM D YYYY')}
      </Text>
      {!!hasData && (
        <IconButton
          onPress={openWorkoutForDate}
          icon={'plus'}
          iconColor={Colors.white}
          mode={'contained'}
          style={{ borderRadius: 5, backgroundColor:
Colors.brand.default }}
        />
      )}
    </View>
    <View

```

```

        style={{
          paddingVertical: 20,
          gap: 10,
          paddingRight: 10,
          flex: 1,
        }}>
        {renderInfo()}
      </View>
    </View>
  );
};

const styles = StyleSheet.create({
  modalContainer: {
    backgroundColor: Colors.white,
    height: '80%',
    padding: 20,
    borderTopWidth: 2,
    borderColor: Colors.neutral.line,
  },
  header: {
    borderBottomWidth: 2,
    borderColor: Colors.neutral.ghost,
    paddingBottom: 5,
  },
  exerciseName: {
    ...Fonts.semibold_18,
  },
  setsContainer: {
    gap: 10,
    padding: 10,
    paddingHorizontal: 25,
  },
  setRow: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  btnContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    gap: 10,
    padding: 10,
  },
  closeBtn: {
    flex: 1,
    backgroundColor: Colors.neutral.background,

```

```

      alignItems: 'center',
      justifyContent: 'center',
      paddingVertical: 10,
    },
    actionBtn: {
      backgroundColor: Colors.brand.default,
      alignItems: 'center',
      justifyContent: 'center',
      paddingVertical: 10,
      borderRadius: 6,
    },
    openBtnText: {
      ...Fonts.semibold_16,
      color: Colors.white,
    },
    infoHeader: {
      flexDirection: 'row',
      justifyContent: 'space-between',
      alignItems: 'center',
      height: 35,
    },
  },
});

```

A.1.13 profile.screen.tsx

```

import React, { useCallback } from 'react';
import { StyleSheet, ScrollView } from 'react-native';
import auth from '@react-native-firebase/auth';
import { PageWrapper } from '../../components';
import { ProfileItem } from './components';
import { useNavigation } from '@react-navigation/native';
import { Colors } from '../../consts';
import { StackNavigationProp } from '@react-navigation/stack';
import { ProfileScreens } from '../../types';
import { ProfileNavigatorStackParamList } from '../../navigators/types';

const ProfileScreen = () => {
  const navigation =
    useNavigation<StackNavigationProp<ProfileNavigatorStackParamList>>>();
  const onSignOut = async () => {
    return auth().signOut();
  };

  const openBodyTracker = useCallback(() => {
    navigation.navigate(ProfileScreens.BODY_TRACKER);
  }, [navigation]);
  const openStats = useCallback(() => {
    navigation.navigate(ProfileScreens.STATS);
  }, [navigation]);

```

```

    }, [navigation]);
    const openAnalysis = useCallback(() => {
      navigation.navigate(ProfileScreens.ANALYSIS);
    }, [navigation]);
    const openAboutApp = useCallback(() => {
      navigation.navigate(ProfileScreens.ABOUT_APP);
    }, [navigation]);

    return (
      <PageWrapper>
        <ScrollView
          style={styles.root}
          contentContainerStyle={styles.rootContent}>
          <ProfileItem
            icon={'arm-flex'}
            title={'Body tracker'}
            onPress={openBodyTracker}
          />
          <ProfileItem
            icon={'chart-box-outline'}
            title={'Stats'}
            onPress={openStats}
          />
          <ProfileItem
            icon={'chart-arc'}
            title={'Analysis'}
            onPress={openAnalysis}
          />
          <ProfileItem
            icon={'information-outline'}
            title={'About app'}
            onPress={openAboutApp}
          />
        </ScrollView>
      </PageWrapper>
    );
  };

  const styles = StyleSheet.create({
    root: {
      flex: 1,
      backgroundColor: Colors.white,
    },
    rootContent: {
      padding: 20,
      gap: 20,
    },
  });

```

```
export default ProfileScreen;
```

A.1.14 analysis.screen.tsx

```
import React, { useMemo, useState } from 'react';
import { PageWrapper } from '../../components';
import {
  FlatList,
  ScrollView,
  StyleSheet,
  Text,
  TouchableOpacity,
  View,
} from 'react-native';
import { useAppSelector } from '../../stores';
import {
  AnalysisCriteria,
  getSetsPercentageByCategory,
  GraphPeriod,
} from '../../stores/workouts';
import { PieChart } from 'react-native-gifted-charts';
import { i18n } from '../../i18n';
import { Colors, Fonts } from '../../consts';
import { GraphPeriodSelector } from
  '../../workout/components/graphPeriodSelector';
import { Dialog, Icon, Portal } from 'react-native-paper';

const getValueLabel = (criteria: AnalysisCriteria) => {
  if (criteria === AnalysisCriteria.NUMBER_OF_WORKOUTS)
    return i18n.t('workouts');
  if (criteria === AnalysisCriteria.NUMBER_OF_SETS) return
i18n.t('sets');
  if (criteria === AnalysisCriteria.NUMBER_OF_REPS) return
i18n.t('reps');
};

const CategoryWithData = (props: {
  category: string;
  value: number;
  percentage: string;
  selectedCriteria: AnalysisCriteria;
}) => {
  return (
    <View>
      <View style={styles.header}>
        <Text style={styles.exerciseName}>{props.category}</Text>
      </View>
    </View>
  );
}
```

```

    <View style={styles.setsContainer}>
      <View style={styles.setRow}>
        <Text style={Fonts.bold_16}>
          {props.value}
        <Text style={Fonts.regular_14}>
          {' '}
          {getValueLabel(props.selectedCriteria)?.toLowerCase()}
        </Text>
      </Text>
      <Text style={Fonts.bold_16}>
        {props.percentage} <Text style={Fonts.regular_14}>>%</Text>
      </Text>
    </View>
  </View>
</View>
);
};

const AnalysisScreen = () => {
  const [width, setWidth] = useState(0);

  const [selectedPeriod, setSelectedPeriod] = useState<GraphPeriod>(
    GraphPeriod['1m'],
  );
  const [selectedCriteria, setSelectedCriteria] =
    useState<AnalysisCriteria>(
      AnalysisCriteria.NUMBER_OF_SETS,
    );
  const [criteriaSelectorVisible, setCriteriaSelectorVisible] =
    useState<boolean>(false);

  const result = useAppSelector(state =>
    getSetsPercentageByCategory(state, selectedPeriod, selectedCriteria),
  );

  const pieData = useMemo(
    () =>
      result.map(r => {
        return {
          value: r.percentage,
          label: r.categoryId,
          sets: r.value,
        };
      }),
    [result],
  );

  const [selected, setSelected] = useState(pieData[0]);

```

```

return (
  <PageWrapper style={{ backgroundColor: Colors.white }}>
    <ScrollView
      style={{ flex: 1 }}
      contentContainerStyle={{ paddingBottom: 20 }}>
      <AnalysisCriteriaSelector
        selectedCriteria={selectedCriteria}
        onPress={() => setCriteriaSelectorVisible(true)}
      />
      <GraphPeriodSelector
        selectedPeriod={selectedPeriod}
        onChange={period => {
          setSelectedPeriod(period);
        }}
      />
      <View
        style={styles.chartContainer}
        onLayout={e => {
          setWidth(e.nativeEvent.layout.width - 80);
        }}>
        <PieChart
          donut
          showText
          textColor="black"
          textSize={20}
          focusOnPress
          radius={width / 2}
          textBackgroundRadius={26}
          data={pieData}
          onPress={(props: any) => {
            setSelected(props);
          }}
          extraRadiusForFocused={10}
          centerLabelComponent={() => {
            return (
              <View
                style={{ justifyContent: 'center', alignItems: 'center'

```

```

        fontSize: 22,
        color: 'black',
        fontWeight: 'bold',
      }}>
      {selected.value.toFixed(2)}%
    </Text>
  ) : (
    <Text
      style={{
        ...Fonts.semibold_14,
        color: Colors.neutral.ghost,
      }}>
      Press to select
    </Text>
  )}
</View>
);
}}
/>
</View>
<View style={styles.infoContainer}>
  <FlatList
    data={pieData}
    renderItem={({ item, index }) => (
      <CategoryWithData
        key={index}
        category={i18n.t(`categories.${item.label}`)}
        value={item.sets}
        selectedCriteria={selectedCriteria}
        percentage={item.value.toFixed(2)}
      />
    )}
    scrollEnabled={false}
    ItemSeparatorComponent={() => <View />}
  />
</View>
<SelectCriteriaModal
  visible={criteriaSelectorVisible}
  selectedCriteria={selectedCriteria}
  onDismiss={() => setCriteriaSelectorVisible(false)}
  onSubmit={criteria => {
    setSelectedCriteria(criteria);
  }}
/>
</ScrollView>
</PageWrapper>
);
};

```

```

const styles = StyleSheet.create({
  chartContainer: {
    width: '100%',
    padding: 20,
  },
  infoContainer: {
    backgroundColor: Colors.white,
    borderRadius: 10,
    paddingHorizontal: 20,
  },
  categoryName: {
    ...Fonts.semibold_18,
    flex: 1,
  },
  sets: {
    ...Fonts.semibold_18,
    flex: 0.5,
    textAlign: 'left',
  },
  percentage: {
    ...Fonts.semibold_18,
    flex: 0.5,
    textAlign: 'right',
  },
  infoItem: {
    flexDirection: 'row',
    paddingVertical: 10,
    paddingHorizontal: 10,
    alignItems: 'center',
    justifyContent: 'space-between',
    gap: 15,
  },
  header: {
    borderBottomWidth: 2,
    borderColor: Colors.neutral.ghost,
    paddingBottom: 5,
  },
  exerciseName: {
    ...Fonts.semibold_18,
  },
  setsContainer: {
    gap: 10,
    padding: 10,
    paddingHorizontal: 25,
  },
  setRow: {
    flexDirection: 'row',

```

```

        alignItems: 'center',
        justifyContent: 'space-between',
      },
    ));

const AnalysisCriteriaSelector = ({
  selectedCriteria,
  onPress,
}: {
  selectedCriteria: AnalysisCriteria;
  onPress: () => void;
}) => {
  return (
    <View style={selectorStyles.container}>
      <Text style={selectorStyles.selectedCriteria}>
        {i18n.t('selected_criteria')}:
      </Text>
      <TouchableOpacity
        activeOpacity={0.8}
        style={selectorStyles.btn}
        onPress={onPress}>
        <Text
style={selectorStyles.btnText}>{i18n.t(selectedCriteria)}</Text>
          <Icon
            size={14}
            source={'chevron-down'}
            color={Colors.neutral.primary}
          />
        </TouchableOpacity>
      </View>
    );
  };

const selectorStyles = StyleSheet.create({
  container: {
    flexDirection: 'row',
    alignItems: 'center',
    gap: 10,
    paddingHorizontal: 20,
    paddingTop: 20,
    marginBottom: 10,
  },
  selectedCriteria: {
    ...Fonts.semibold_18,
    color: Colors.neutral.ghost,
  },
  btn: {
    flexDirection: 'row',

```

```

        alignItems: 'center',
        justifyContent: 'space-between',
        backgroundColor: Colors.brand.background,
        borderRadius: 6,
        padding: 10,
        gap: 5,
      },
      btnText: {
        ...Fonts.semibold_14,
        color: Colors.neutral.primary,
      },
    }));

const Criteria = ({
  selected,
  criteria,
  description,
  onPress,
}: {
  selected: boolean;
  criteria: AnalysisCriteria;
  description?: string;
  onPress: (value: AnalysisCriteria) => void;
}) => {
  return (
    <TouchableOpacity
      style={[
        dialogStyles.criteriaContainer,
        selected && dialogStyles.selectedCriteriaContainer,
      ]}
      activeOpacity={0.8}
      onPress={() => onPress(criteria)}>
      <View style={dialogStyles.textContainer}>
        <Text style={dialogStyles.name}>{i18n.t(criteria)}</Text>
        {description ? (
          <Text
            style={[
              dialogStyles.description,
              selected && dialogStyles.selectedDescription,
            ]}>
            {description}
          </Text>
        ) : null}
      </View>
      {selected ? (
        <Icon size={20} source={'check'} color={Colors.brand.dark} />
      ) : null}
    </TouchableOpacity>
  );
};

```

```

    );
};

type SelectCriteriaModalProps = {
  visible: boolean;
  selectedCriteria: AnalysisCriteria;
  onDismiss: () => void;
  onSubmit: (value: AnalysisCriteria) => void;
};

const SelectCriteriaModal = ({
  visible,
  onDismiss,
  selectedCriteria,
  onSubmit,
}: SelectCriteriaModalProps) => {
  const dismiss = () => {
    onDismiss();
  };
  const onSelect = (criteria: AnalysisCriteria) => {
    onSubmit(criteria);
    onDismiss();
  };
  return (
    <Portal>
      <Dialog
        style={dialogStyles.createModalContainer}
        onDismiss={dismiss}
        visible={visible}>
        <Dialog.Title style={Fonts.semibold_24}>
          {i18n.t('select_criteria')}
        </Dialog.Title>
        <Dialog.Content style={{ gap: 15 }}>
          <Criteria
            selected={selectedCriteria ===
AnalysisCriteria.NUMBER_OF_SETS}
            criteria={AnalysisCriteria.NUMBER_OF_SETS}
            onPress={onSelect}
          />
          <Criteria
            selected={selectedCriteria ===
AnalysisCriteria.NUMBER_OF_REPS}
            criteria={AnalysisCriteria.NUMBER_OF_REPS}
            onPress={onSelect}
          />
          <Criteria
            selected={selectedCriteria ===
AnalysisCriteria.NUMBER_OF_WORKOUTS}

```

```

        criteria={AnalysisCriteria.NUMBER_OF_WORKOUTS}
        onPress={onSelect}
      />
    </Dialog.Content>
  </Dialog>
</Portal>
);
};

const dialogStyles = StyleSheet.create({
  createModalContainer: {
    backgroundColor: Colors.white,
    borderRadius: 10,
  },
  modalBtn: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 5,
    paddingVertical: 10,
  },
  modalBtnText: {
    ...Fonts.bold_16,
  },
  criteriaContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    backgroundColor: Colors.neutral.background,
    paddingVertical: 15,
    paddingHorizontal: 20,
    borderRadius: 10,
  },
  selectedCriteriaContainer: {
    backgroundColor: Colors.brand.outline,
  },
  textContainer: {},
  name: {
    ...Fonts.semibold_18,
  },
  description: {
    ...Fonts.regular_14,
    color: Colors.neutral.ghost,
  },
  selectedDescription: {
    color: Colors.neutral.secondary,
  },
});

```

```
export default AnalysisScreen;
```

A.1.15 bodyTracker.screen.tsx

```
import React from 'react';
import { BodyTrackerNavigatorParamList } from
'../../../../../navigators/types';
import { BodyTrackerScreens } from '../../../../../types';
import { Colors, Fonts } from '../../../../../consts';
import { createMaterialTopTabNavigator } from '@react-
navigation/material-top-tabs';
import { GraphTab, WeightTab } from '../components';

const Tab =
createMaterialTopTabNavigator<BodyTrackerNavigatorParamList>();

const BodyTrackerScreen = () => {
  return (
    <Tab.Navigator
      initialRouteName={BodyTrackerScreens.LOG}
      screenOptions={{
        tabBarIndicatorStyle: { backgroundColor: Colors.brand.default },
        tabBarBounces: false,
        tabBarLabelStyle: {
          ...Fonts.semibold_18,
          textTransform: 'capitalize',
        },
      }}>
      <Tab.Screen
        name={BodyTrackerScreens.LOG}
        component={WeightTab}
        options={{
          title: 'Weight Log',
        }}
      />
      <Tab.Screen
        name={BodyTrackerScreens.GRAPH}
        component={GraphTab}
        options={{
          title: 'Graph',
        }}
      />
    </Tab.Navigator>
  );
};

export default BodyTrackerScreen;
```

A.1.16 stats.screen.tsx

```
import React from 'react';
import { PageWrapper } from '../../components';
import { Text, View, StyleSheet } from 'react-native';
import { Colors, Fonts, SCREEN_WIDTH } from '../../consts';
import { useAppSelector } from '../../stores';
import {
  getTotalReps,
  getTotalSets,
  getTotalVolume,
  getTotalWorkouts,
} from '../../stores/workouts';
import { i18n } from '../../i18n';

type StatItemProps = {
  label: string;
  value: any;
};

const StatItem = ({ label, value }: StatItemProps) => {
  return (
    <View style={styles.statItem}>
      <Text style={styles.statLabel}>{label}</Text>
      <Text style={styles.statValue} adjustsFontSizeToFit>
        {value}
      </Text>
    </View>
  );
};

const StatsScreen = () => {
  const totalWorkouts = useAppSelector(state => getTotalWorkouts(state));
  const totalSets = useAppSelector(state => getTotalSets(state));
  const totalReps = useAppSelector(state => getTotalReps(state));
  const totalVolume = useAppSelector(state => getTotalVolume(state));
  return (
    <PageWrapper style={{ paddingVertical: 20 }}>
      <View style={styles.statsContainer}>
        <StatItem value={totalWorkouts} label={i18n.t('total_workouts')} />
        <StatItem value={totalSets} label={i18n.t('total_sets')} />
        <StatItem value={totalReps} label={i18n.t('total_reps')} />
        <StatItem
          value={` ${totalVolume} ${i18n.t('kgs')} `}
          label={i18n.t('total_volume')}
        />
      </View>
    </PageWrapper>
  );
};
```

```

    </PageWrapper>
  );
};

const styles = StyleSheet.create({
  statItem: {
    padding: 20,
    backgroundColor: Colors.white,
    width: SCREEN_WIDTH / 2 - 25,
    borderWidth: 2,
    borderRadius: 8,
    borderColor: Colors.neutral.ghost,
    gap: 10,
  },
  statLabel: {
    ...Fonts.regular_18,
    textAlign: 'center',
  },
  statValue: {
    ...Fonts.semibold_28,
    color: Colors.neutral.primary,
    textAlign: 'center',
    width: '100%',
  },
  statsContainer: {
    flexDirection: 'row',
    flexWrap: 'wrap',
    flex: 1,
    alignItems: 'flex-start',
    paddingHorizontal: 20,
    gap: 10,
  },
});

```

```
export default StatsScreen;
```

A.1.17 GraphTab.tsx

```

import { PageWrapper } from '../../components';
import { Text, StyleSheet, View } from 'react-native';
import React, { useMemo, useState } from 'react';
import { GraphPeriodSelector } from
  '../../workout/components/graphPeriodSelector';
import { GraphPeriod } from '../../stores/workouts';
import { Colors, Fonts, SCREEN_HEIGHT, SCREEN_WIDTH } from
  '../../consts';
import { LineChart } from 'react-native-gifted-charts';
import { useAppSelector } from '../../stores';

```

```

import dayjs from 'dayjs';
import { i18n } from '../i18n';

export const GraphTab = () => {
  const [selectedPeriod, setSelectedPeriod] = useState<GraphPeriod>(
    GraphPeriod['1m'],
  );
  const [graphHeight, setGraphHeight] = useState<number>(SCREEN_HEIGHT *
0.6);
  const [selectedPoint, setSelectedPoint] = useState<{
    value: any;
    date: string;
  } | null>(null);

  const weight = useAppSelector(state => state.bodyTracker.weight);

  const data = useMemo(() => {
    return weight.map(wt => ({
      value: wt.weight,
      date: wt.date,
      label: dayjs(wt.date).format('DD MMM'),
    }));
  }, [weight]);

  return (
    <PageWrapper style={{ backgroundColor: Colors.white, paddingTop: 10
}}>
      <GraphPeriodSelector
        selectedPeriod={selectedPeriod}
        onChange={period => {
          setSelectedPoint(null);
          setSelectedPeriod(period);
        }}
      />
      <View
        style={styles.chartContainer}
        onLayout={layout => {
          setGraphHeight(layout.nativeEvent.layout.height - 40 * 2);
        }}
      >
        <LineChart
          height={graphHeight}
          width={SCREEN_WIDTH - 80}
          areaChart
          color={Colors.brand.dark}
          dataPointsColor={Colors.brand.dark}
          startFillColor={Colors.brand.outline}
          endFillColor={Colors.brand.outline}
          startOpacity={0.4}

```

```

        endOpacity={0.4}
        data={data}
        adjustToWidth
        formatYLabel={label => `${Math.round(Number(label) / 10) *
10}`}

        focusEnabled={true}
        showTextOnFocus={true}
        showStripOnFocus={true}
        focusedDataPointColor={Colors.brand.default}
        unFocusOnPressOut={false}
        onFocus={({props: any}) => {
            setSelectedPoint({ value: props.value, date: props.date });
        }}
        pointerConfig={{
            resetPointerOnDataChange: true,
            persistPointer: true,
        }}
    />
</View>
<View style={styles.selectedContainer}>
    {selectedPoint ? (
        <>
            <Text style={styles.selectedValueText}>
                {dayjs(selectedPoint.date).format('DD MMM YYYY')}:
            </Text>
            <Text style={styles.selectedValueText}>
                {selectedPoint.value}
                {i18n.t('kgs')}
            </Text>
        </>
    ) : (
        <Text
style={styles.placeholder}>{i18n.t('graph_details')}</Text>
    )}
</View>
</PageWrapper>
);
};

const styles = StyleSheet.create({
    chartContainer: {
        paddingHorizontal: 20,
        flex: 1,
        marginTop: 20,
    },
    selectedContainer: {
        backgroundColor: Colors.neutral.background,
        flexDirection: 'row',
    },
});

```

```

    alignItems: 'center',
    justifyContent: 'center',
    gap: 10,
    height: 40,
  },
  selectedValueText: {
    ...Fonts.semibold_18,
    color: Colors.neutral.primary,
  },
  placeholder: {
    ...Fonts.semibold_14,
    color: Colors.neutral.ghost,
  },
});

```

A.1.18 WeightTab.tsx

```

import { useAppDispatch, useAppSelector } from '../../../stores';
import React, { useCallback, useMemo, useState } from 'react';
import {
  deleteWeight,
  logWeight,
  WeightWithTimestamp,
} from '../../../stores/bodyTracker';
import {
  FlatList,
  StyleSheet,
  Text,
  TouchableOpacity,
  View,
} from 'react-native';
import {
  Dialog,
  FAB,
  Icon,
  IconButton,
  Portal,
  TextInput,
} from 'react-native-paper';
import { Colors, Fonts } from '../../../consts';
import dayjs from 'dayjs';
import { i18n } from '../../../i18n';
import { PageWrapper } from '../../../components';
import { EmptyList } from '../../workout/components';
import _ from 'lodash';

const getDiffIcon = (diff: number): { icon: string; color: string } => {
  if (diff === 0) return { icon: 'equal', color: Colors.button.warning };

```

```

    if (diff > 0) return { icon: 'chevron-up', color: Colors.button.success
  };
  return { icon: 'chevron-down', color: Colors.brand.outline };
};

export const WeightTab = () => {
  const dispatch = useAppDispatch();
  const weight = useAppSelector(state => state.bodyTracker.weight);

  const [weightModalVisible, setWeightModalVisible] =
    useState<boolean>(false);

  const onAddWeightPress = useCallback(() => {
    setWeightModalVisible(true);
  }, []);

  const renderWeight = (
    weightWithTimestamp: WeightWithTimestamp,
    prevItem: WeightWithTimestamp | undefined,
  ) => {
    const diffNumber = prevItem
      ? weightWithTimestamp.weight - prevItem.weight
      : null;
    const renderDiff = Number.isFinite(diffNumber) ? (
      <View
        style={{
          width: 50,
          flexDirection: 'row',
          alignItems: 'center',
          justifyContent: 'flex-end',
          gap: 5,
        }}>
        <View style={{ width: 15 }}>
          <Icon
            size={20}
            source={getDiffIcon(diffNumber!).icon}
            color={getDiffIcon(diffNumber!).color}
          />
        </View>
        <Text
          style={{ minWidth: 15, textAlign: 'right', ...Fonts.semibold_14
        }}>
          {diffNumber?.toFixed(2)}
        </Text>
      </View>
    ) : null;

    const onDelete = () => {

```

```

    dispatch(deleteWeight(weightWithTimestamp.date));
  };

  return (
    <View style={styles.weightItemContainer}>
      <View>
        <Text style={styles.weightDate}>
          {dayjs(weightWithTimestamp.date).format('DD MMM YYYY')}
        </Text>
        <Text>{dayjs(weightWithTimestamp.date).format('HH:mm')}</Text>
      </View>
      <View style={{ flex: 1, flexDirection: 'row', gap: 10 }}>
        <View>
          <Text style={styles.weight}>{weightWithTimestamp.weight}
        </Text>
        <Text style={styles.weightPostfix}>{i18n.t('kgs')}</Text>
      </View>
      {renderDiff}
    </View>
    <IconButton icon={'delete'} onPress={onDelete} />
  </View>
);
};

const reversedWeight = useMemo(() => [...weight].reverse(), [weight]);

return (
  <PageWrapper>
    <FlatList
      style={styles.weightList}
      contentContainerStyle={styles.weightListContent}
      data={reversedWeight}
      renderItem={({ item, index }) =>
        renderWeight(item, reversedWeight[index + 1])
      }
      ListEmptyComponent={() => (
        <EmptyList message={i18n.t('no_weight_added')} />
      )}
    />
    <FAB
      icon={'plus'}
      color={Colors.white}
      style={styles.fab}
      label={i18n.t('add_measurement')}
      onPress={onAddWeightPress}
    />
    <WeightModal
      visible={weightModalVisible}

```

```

        onClose={() => setWeightModalVisible(false)}
      />
    </PageWrapper>
  );
};

const WeightModal = ({
  visible,
  onClose,
}: {
  visible: boolean;
  onClose: VoidFunction;
}) => {
  const dispatch = useAppDispatch();
  const [weight, setWeight] = useState<string>('');
  const [error, setError] = useState<string | null>(null);

  const onChangeWeight = (value: string) => {
    setError(null);
    setWeight(value);
  };

  const dismiss = useCallback(() => {
    onClose();
    setWeight('');
    setError(null);
  }, [onClose]);

  const onCreate = useCallback(() => {
    if (!_.isNumber(Number(weight))) {
      setError(i18n.t('enter_valid_number'));
      return;
    }
    const date = dayjs().toISOString();
    dispatch(logWeight({ weight: Number(weight), date }));
    dismiss();
  }, [weight, dismiss, dispatch]);

  return (
    <Portal>
      <Dialog
        style={styles.createModalContainer}
        onDismiss={dismiss}
        visible={visible}>
        <Dialog.Title style={Fonts.semibold_24}>
          {i18n.t('add_measurement')}
        </Dialog.Title>
        <Dialog.Content>

```

```

<TextInput
  value={weight}
  onChangeText={onChangeWeight}
  style={{
    height: 40,
    backgroundColor: Colors.neutral.background,
    borderColor: Colors.neutral.background,
  }}
  placeholderTextColor={Colors.neutral.ghost}
  outlineColor={Colors.neutral.line}
  activeOutlineColor={Colors.neutral.line}
  contentStyle={{
    ...Fonts.regular_16,
    color: Colors.neutral.ghost,
    paddingHorizontal: 20,
  }}
  keyboardType={'numeric'}
  placeholder={i18n.t('enter_weight')}
  mode={'outlined'}
  outlineStyle={{
    borderWidth: 1,
    borderRadius: 6,
  }}
/>
{error && <Text style={styles.inputError}>{error}</Text>}
</Dialog.Content>
<Dialog.Actions style={{ flexDirection: 'row', gap: 5 }}>
  <TouchableOpacity
    onPress={dismiss}
    activeOpacity={0.8}
    style={[styles.modalBtn, { backgroundColor:
Colors.neutral.line }]}>
    <Text style={[styles.modalBtnText]}>{i18n.t('cancel')}</Text>
  </TouchableOpacity>
  <TouchableOpacity
    activeOpacity={0.8}
    onPress={onCreate}
    style={[
      styles.modalBtn,
      { backgroundColor: Colors.brand.default },
    ]}>
    <Text style={[styles.modalBtnText, { color: Colors.white }]}>
      {i18n.t('add')}
    </Text>
  </TouchableOpacity>
</Dialog.Actions>
</Dialog>
</Portal>

```

```

    );
  };

const styles = StyleSheet.create({
  weightList: {
    flex: 1,
  },
  weightListContent: {
    flex: 1,
    padding: 20,
    gap: 10,
  },
  weightContainer: {
    flexDirection: 'row',
  },
  fab: {
    position: 'absolute',
    right: 16,
    bottom: 16,
    backgroundColor: Colors.brand.default,
  },
  createModalContainer: {
    backgroundColor: Colors.white,
    borderRadius: 10,
  },
  modalBtn: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 5,
    paddingVertical: 10,
  },
  modalBtnText: {
    ...Fonts.bold_16,
  },
  inputError: {
    ...Fonts.semibold_12,
    color: Colors.button.error,
    marginTop: 5,
  },
  weightItemContainer: {
    flexDirection: 'row',
    padding: 20,
    gap: 25,
    backgroundColor: Colors.white,
    borderRadius: 8,
    alignItems: 'center',
    justifyContent: 'space-between',
  },
});

```

```

    },
    weightDate: {
      ...Fonts.semibold_16,
      color: Colors.neutral.primary,
    },
    weight: {
      ...Fonts.bold_18,
    },
    weightPostfix: {
      ...Fonts.regular_18,
    },
  }));

```

A.1.19 profileItem.tsx

```

import React from 'react';
import { StyleSheet, Text, TouchableOpacity, View } from 'react-native';
import { Colors, Fonts } from '../../const';
import { Icon } from 'react-native-paper';

type ProfileItemProps = {
  icon: any;
  title: string;
  onPress: () => void;
};

export const ProfileItem = ({ onPress, title, icon }: ProfileItemProps) => {
  return (
    <TouchableOpacity style={styles.root} activeOpacity={0.8}
    onPress={onPress}>
      <View>
        <Icon size={35} source={icon} />
      </View>
      <Text style={styles.title}>{title}</Text>
    </TouchableOpacity>
  );
};

const styles = StyleSheet.create({
  root: {
    backgroundColor: Colors.neutral.background,
    padding: 15,
    flexDirection: 'row',
    alignItems: 'center',
    // justifyContent: 'space-between',
    gap: 25,
    borderRadius: 8,
  },

```

```

    shadowColor: Colors.black,
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.15,
    shadowRadius: 3,

    elevation: 5,
  },
  title: {
    ...Fonts.semibold_24,
    color: Colors.neutral.primary,
  },
});

```

A.1.20 home.tsx

```

import React, { useCallback, useEffect } from 'react';
import { StyleSheet, FlatList, Text, TouchableOpacity } from 'react-native';
import BaseButton from '../components/buttons/base.button';
import { RouteProp, useNavigation, useRoute } from '@react-navigation/native';
import { WorkoutScreens } from '../types';
import { WorkoutNavigatorStackParamList } from '../navigators/types';
import { StackNavigationProp } from '@react-navigation/stack';
import { useAppSelector } from '../stores';
import ExerciseCard from './components/exerciseCard';
import { getWorkoutByDate } from '../stores/workouts';
import { getTodayDate } from '../utils';
import { PageWrapper } from '../components';
import { Colors } from '../consts';
import { i18n } from '../i18n';
import { FAB, Icon } from 'react-native-paper';
import { HeaderTimer } from './components';
import { EmptyList } from './components';
import dayjs from 'dayjs';

const HeaderDate = ({
  selectedDate,
  style,
  onPress,
}: {
  selectedDate: string;
  style: any;
  onPress: () => void;
}) => {

```

```

const isToday = selectedDate === getTodayDate();
const displayDate = isToday
  ? i18n.t('today')
  : dayjs(selectedDate).format('dddd, MMMM D YYYY');

return (
  <TouchableOpacity
    style={styles.dateBtn}
    activeOpacity={0.8}
    onPress={onPress}>
    <Text style={style}>{displayDate}</Text>
    {!isToday ? <Icon size={15} source={'refresh'} /> : null}
  </TouchableOpacity>
);
};

export const Home = () => {
  const navigation =
    useNavigation<StackNavigationProp<WorkoutNavigatorStackParamList>>>();
  const route =
    useRoute<RouteProp<WorkoutNavigatorStackParamList,
WorkoutScreens.HOME>>>();

  const selectedDate = route.params.date;

  const onTitlePress = useCallback(() => {
    navigation.setParams({
      date: getTodayDate(),
    });
  }, [navigation]);

  useEffect(() => {
    navigation.setOptions({
      headerRight: () => <HeaderTimer />,
      headerTitle: props => (
        <HeaderDate
          onPress={onTitlePress}
          selectedDate={selectedDate}
          style={props.style}
        />
      ),
    });
  }, [navigation, selectedDate, onTitlePress]);

  const currentWorkout = useAppSelector(state =>
    getWorkoutByDate(state, selectedDate),
  );

```

```

const isWorkoutEmpty = !currentWorkout;

const navigateToCategories = useCallback(() => {
  navigation.navigate(WorkoutScreens.CATEGORIES, {
    date: selectedDate,
  });
}, [navigation, selectedDate]);

return (
  <PageWrapper style={{ paddingHorizontal: 0 }}>
    <FlatList
      style={styles.list}
      contentContainerStyle={styles.listContent}
      data={currentWorkout ? Object.values(currentWorkout) : []}
      keyExtractor={item => item.createdAt}
      renderItem={({ item }) => (
        <ExerciseCard
          workout={item}
          onCardPress={() => {
            navigation.navigate(WorkoutScreens.LOG_WORKOUT, {
              exerciseId: item.exerciseId,
              categoryId: item.categoryId,
              date: selectedDate,
            });
          }}
        />
      )}
      ListEmptyComponent={() => <EmptyList
message={i18n.t('log_empty')} />}
    />
    {isWorkoutEmpty ? (
      <BaseButton
        text={'Start new workout'}
        onPress={navigateToCategories}
        containerStyle={styles.startBtn}
      />
    ) : (
      <FAB
        icon={'plus'}
        color={Colors.white}
        style={styles.fab}
        onPress={navigateToCategories}
      />
    )}
  </PageWrapper>
);
};

```

```

const styles = StyleSheet.create({
  list: {
    flex: 1,
    width: '100%',
    paddingTop: 20,
  },
  listContent: {
    gap: 20,
    minHeight: '100%',
    paddingHorizontal: 20,
    paddingBottom: 40,
  },
  startBtn: {
    marginBottom: 20,
    marginHorizontal: 20,
  },
  fab: {
    position: 'absolute',
    right: 16,
    bottom: 16,
    backgroundColor: Colors.brand.default,
  },
  dateBtn: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    gap: 5,
  },
});

```

A.1.21 categories.screen.tsx

```

import React, { useCallback, useEffect, useState } from 'react';
import {
  View,
  StyleSheet,
  Text,
  FlatList,
  TouchableOpacity,
  Keyboard,
} from 'react-native';
import {
  createCategory,
  ExerciseCategory,
  useAppDispatch,
  useAppSelector,
} from '../stores';
import { Colors, Fonts } from '../consts';

```

```

import { RouteProp, useNavigation, useRoute } from '@react-
navigation/native';
import { StackNavigationProp } from '@react-navigation/stack';
import { WorkoutNavigatorStackParamList } from '../../navigators/types';
import { WorkoutScreens } from '../../types';
import { i18n } from '../../i18n';
import { HeaderAddButton, Separator } from './components';
import { PageWrapper } from '../../components';
import { Dialog, Portal, TextInput } from 'react-native-paper';

type CreateCategoryDialogProps = {
  visible: boolean;
  onClose: () => void;
};

const CreateCategoryDialog = (props: CreateCategoryDialogProps) => {
  const dispatch = useAppDispatch();
  const [name, setName] = useState<string>('');
  const [error, setError] = useState<string | null>(null);

  const onCreate = useCallback(() => {
    setError(null);
    if (name) {
      dispatch(createCategory({ name }));
      Keyboard.dismiss();
      props.onClose();
    }
    if (!name) {
      setError(i18n.t('errors.empty_name'));
    }
  }, [name, dispatch, props]);

  const dismiss = useCallback(() => {
    props.onClose();
    setName('');
    setError(null);
  }, [props]);

  const onChangeText = (value: string) => {
    setError(null);
    setName(value);
  };

  return (
    <Portal>
      <Dialog
        style={styles.createModalContainer}
        onDismiss={dismiss}

```

```

visible={props.visible}>
<Dialog.Title style={Fonts.semibold_24}>
  {i18n.t('create_category')}
</Dialog.Title>
<Dialog.Content>
  <TextInput
    value={name}
    onChangeText={onChangeText}
    style={{
      height: 40,
      backgroundColor: Colors.neutral.background,
      borderColor: Colors.neutral.background,
    }}
    placeholderTextColor={Colors.neutral.ghost}
    outlineColor={Colors.neutral.line}
    activeOutlineColor={Colors.neutral.line}
    contentStyle={{
      ...Fonts.regular_16,
      color: Colors.neutral.ghost,
      paddingHorizontal: 20,
    }}
    placeholder='Category Name'
    mode='outlined'
    outlineStyle={{
      borderWidth: 1,
      borderRadius: 6,
    }}
  />
  {error && <Text style={styles.inputError}>{error}</Text>}
</Dialog.Content>
<Dialog.Actions style={{ flexDirection: 'row', gap: 5 }}>
  <TouchableOpacity
    onPress={dismiss}
    activeOpacity={0.8}
    style={[styles.modalBtn, { backgroundColor:
Colors.neutral.line }]}>
    <Text style={[styles.modalBtnText]}>{i18n.t('cancel')}</Text>
  </TouchableOpacity>
  <TouchableOpacity
    activeOpacity={0.8}
    onPress={onCreate}
    style={[
      styles.modalBtn,
      { backgroundColor: Colors.brand.default },
    ]}>
    <Text style={[styles.modalBtnText, { color: Colors.white }]}>
      {i18n.t('create')}
    </Text>
  </TouchableOpacity>
</Dialog.Actions>

```

```

        </TouchableOpacity>
      </Dialog.Actions>
    </Dialog>
  </Portal>
);
};

export const CategoriesScreen = () => {
  const navigation =
    useNavigation<StackNavigationProp<WorkoutNavigatorStackParamList>>();
  const route =
    useRoute<
      RouteProp<WorkoutNavigatorStackParamList,
WorkoutScreens.CATEGORIES>
    >();
  const { categories } = useAppSelector(state => state.exercises);
  const [createCategoryDialogVisible, setCreateCategoryDialogVisible] =
    useState(false);

  useEffect(() => {
    navigation.setOptions({
      headerRight: () => (
        <HeaderAddButton onPress={() =>
setCreateCategoryDialogVisible(true)} />
      ),
    });
  }, [navigation]);

  const renderCategory = useCallback(
    (category: ExerciseCategory) => {
      return (
        <TouchableOpacity
          onPress={() =>
            navigation.navigate(WorkoutScreens.EXERCISES, {
              categoryId: category.id,
              date: route.params.date,
            })
          }
          activeOpacity={0.8}
          style={styles.category}>
          <Text
style={styles.categoryName}>{i18n.t(category.name)}</Text>
          <View style={styles.amountContainer}>
            <Text style={styles.exercisesAmount}>
              {category.exercises.length}
            </Text>
            <Text style={styles.exercisesAmountLabel}>
              {i18n.t('exercises.exercises')}
            </Text>
          </View>
        </TouchableOpacity>
      );
    },
  );

  return (
    <View style={styles.container}>
      {categories.map(renderCategory)}
    </View>
  );
};

```

```

        </Text>
      </View>
    </TouchableOpacity>
  );
},
[navigation, route.params.date],
);
return (
  <PageWrapper style={{ paddingHorizontal: 0 }}>
    <FlatList
      style={styles.list}
      data={categories}
      keyExtractor={category => category.id}
      renderItem={({ item }) => renderCategory(item)}
      ItemSeparatorComponent={Separator}
    />
    <CreateCategoryDialog
      visible={createCategoryDialogVisible}
      onClose={() => setCreateCategoryDialogVisible(false)}
    />
  </PageWrapper>
);
};

const styles = StyleSheet.create({
  list: {
    flex: 1,
  },
  category: {
    paddingVertical: 10,
    paddingHorizontal: 15,
    backgroundColor: Colors.white,
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  categoryName: {
    ...Fonts.semibold_18,
    color: Colors.black,
  },
  exercisesAmount: {
    ...Fonts.bold_18,
    color: Colors.neutral.ghost,
    textAlign: 'right',
  },
  exercisesAmountLabel: {
    ...Fonts.semibold_12,
    color: Colors.neutral.ghost,
  },
});

```

```

      textTransform: 'lowercase',
    },
    amountContainer: {},
    createModalContainer: {
      backgroundColor: Colors.white,
      borderRadius: 10,
    },
    modalBtn: {
      flex: 1,
      alignItems: 'center',
      justifyContent: 'center',
      borderRadius: 5,
      paddingVertical: 10,
    },
    modalBtnText: {
      ...Fonts.bold_16,
    },
    inputError: {
      ...Fonts.semibold_12,
      color: Colors.button.error,
      marginTop: 5,
    },
  },
});

```

A.1.22 exercises.screen.tsx

```

import React, { useCallback, useEffect } from 'react';
import {
  FlatList,
  StyleSheet,
  Text,
  TouchableOpacity,
  View,
} from 'react-native';
import { RouteProp, useNavigation, useRoute } from '@react-navigation/native';
import { WorkoutNavigatorStackParamList } from '../../navigators/types';
import { WorkoutScreens } from '../../types';
import { DefaultExercise, getCategoryById, useAppSelector } from '../../stores';
import { Colors, Fonts } from '../../consts';
import { i18n } from '../../i18n';
import { StackNavigationProp } from '@react-navigation/stack';
import { getExercisesByCategoryId } from '../utils';
import { EmptyList, HeaderAddButton, Separator } from '../components';

export const ExercisesScreen = () => {
  const route =

```

```

    useRoute<
      RouteProp<WorkoutNavigatorStackParamList, WorkoutScreens.EXERCISES>
    >();
    const navigation =
      useNavigation<StackNavigationProp<WorkoutNavigatorStackParamList>>();

    useEffect(() => {
      navigation.setOptions({
        headerRight: () => <HeaderAddButton onPress={() => {}} />,
      });
    }, []);

    const selectedCategory = useAppSelector(state =>
      getCategoryById(state, route.params.categoryId),
    );

    const exercises = useAppSelector(state =>
      getExercisesByCategoryId(state, route.params.categoryId),
    );

    const renderExercise = useCallback(
      ({ item }: { item: DefaultExercise }) => {
        return (
          <TouchableOpacity
            onPress={() =>
              navigation.navigate(WorkoutScreens.LOG_WORKOUT, {
                categoryId: item.categoryId,
                exerciseId: item.id,
                date: route.params.date,
              })
            }
            activeOpacity={0.8}
            style={styles.exercise}>
            <Text style={styles.exerciseName}>{i18n.t(item.name)}</Text>
          </TouchableOpacity>
        );
      },
      [navigation, route.params.date],
    );

    const ListHeader = () => (
      <View>
        <Text style={styles.selectedCategory}>
          {i18n.t('selected_category')}:
        </Text>
        <Text style={styles.selectedCategoryName}>
          {i18n.t(`${selectedCategory?.name}`)}
        </Text>
      </View>
    );

```

```

    </View>
  );

  return (
    <View style={StyleSheet.absoluteFillObject}>
      <FlatList
        style={styles.list}
        contentContainerStyle={styles.listContent}
        data={exercises}
        keyExtractor={exercise => exercise.id}
        renderItem={renderExercise}
        ItemSeparatorComponent={Separator}
        ListHeaderComponent={ListHeader}
        ListHeaderComponentStyle={styles.headerContainer}
        ListEmptyComponent={() => (
          <EmptyList message={i18n.t('empty_category')} />
        )}
      />
    </View>
  );
};

const styles = StyleSheet.create({
  list: {
    flex: 1,
  },
  listContent: {
    flex: 1,
  },
  exercise: {
    paddingVertical: 10,
    paddingHorizontal: 15,
    backgroundColor: Colors.white,
  },
  exerciseName: {
    ...Fonts.semibold_18,
    color: Colors.black,
  },
  selectedCategory: {
    ...Fonts.regular_14,
    color: Colors.neutral.ghost,
  },
  selectedCategoryName: {
    ...Fonts.semibold_18,
    color: Colors.neutral.secondary,
  },
  headerContainer: {
    backgroundColor: Colors.white,

```

```

paddingHorizontal: 15,
paddingVertical: 10,
borderBottomWidth: 2,
borderBlockColor: Colors.neutral.line,
},
});

```

A.1.23 workout.screen.tsx

```

import React, { useCallback, useEffect, useState } from 'react';
import {
  View,
  StyleSheet,
  Text,
  Keyboard,
  FlatList,
  TouchableOpacity,
} from 'react-native';
import { RouteProp, useNavigation, useRoute } from '@react-
navigation/native';
import { WorkoutNavigatorStackParamList } from '../../navigators/types';
import { WorkoutScreens } from '../../types';
import { useAppDispatch, useAppSelector } from '../../stores';
import { i18n } from '../../i18n';
import { HeaderTimer, WeightForm } from './components';
import {
  deleteSetFromExercise,
  getWorkoutByExerciseIdForDate,
  logNewSet,
  Set,
  updateSet,
} from '../../stores/workouts';
import dayjs from 'dayjs';
import { Colors, Fonts } from '../../consts';
import { getExerciseDetails } from './utils';
import { SetDetails } from './components/setDetails';
import { ListRenderItemInfo } from '@react-native/virtualized-
lists/Lists/VirtualizedList';
import { IconButton, Portal, Dialog, Snackbar } from 'react-native-
paper';
import { useBottomTabBarHeight } from '@react-navigation/bottom-tabs';
import { StackNavigationProp } from '@react-navigation/stack';

type DeleteConfirmationProps = {
  visible: boolean;
  onSubmit: () => void;
  onDismiss: () => void;
};

```

```

const DeleteConfirmation = (props: DeleteConfirmationProps) => {
  return (
    <Portal>
      <Dialog
        style={styles.deleteModalContainer}
        visible={props.visible}
        onDismiss={props.onDismiss}>
        <Dialog.Title style={styles.deleteTitle}>
          {i18n.t('delete_set')}
        </Dialog.Title>
        <Dialog.Content>
          <Text style={styles.deleteDescription}>
            {i18n.t('delete_set_description')}
          </Text>
        </Dialog.Content>
        <Dialog.Actions style={{ flexDirection: 'row', gap: 5 }}>
          <TouchableOpacity
            activeOpacity={0.8}
            onPress={props.onDismiss}
            style={[styles.alertBtn, { backgroundColor:
Colors.neutral.line }]}>
            <Text
              style={[styles.alertBtnText, { color:
Colors.neutral.primary }]}>
              {i18n.t('delete_set_no')}
            </Text>
          </TouchableOpacity>
          <TouchableOpacity
            activeOpacity={0.8}
            onPress={props.onSubmit}
            style={[styles.alertBtn, { backgroundColor:
Colors.button.error }]}>
            <Text style={[styles.alertBtnText, { color: Colors.white }]}>
              {i18n.t('delete_set_yes')}
            </Text>
          </TouchableOpacity>
        </Dialog.Actions>
      </Dialog>
    </Portal>
  );
};

export const WorkoutScreen = () => {
  const dispatch = useAppDispatch();
  const bottomTabBarHeight = useBottomTabBarHeight();
  const navigation =
    useNavigation<StackNavigationProp<WorkoutNavigatorStackParamList>>();

```

```

const route =
  useRoute<
    RouteProp<WorkoutNavigatorStackParamList,
    WorkoutScreens.LOG_WORKOUT>
  >();

useEffect(() => {
  navigation.setOptions({
    headerRight: () => <HeaderTimer />,
  });
}, []);

const { exerciseId, categoryId, date } = route.params;

const exercise = useAppSelector(state =>
  getExerciseDetails(state, categoryId, exerciseId),
);

const workout = useAppSelector(state =>
  getWorkoutByExerciseIdForDate(state, date, exerciseId),
);
const initialWeight = workout?.sets[workout?.sets?.length - 1]?.weight;
const initialReps = workout?.sets[workout?.sets?.length - 1]?.reps;

const [weight, setWeight] = useState<number>(initialWeight ?? 0);
const [reps, setReps] = useState<number>(initialReps ?? 0);

const [deleteModalVisible, setDeleteModalVisible] =
useState<boolean>(false);
const [deleteModalInfo, setDeleteModalInfo] = useState<{
  id: string;
  index: number;
} | null>(null);

const [updateState, setUpdateState] = useState<{
  index: number;
  exerciseId: string;
} | null>(null);

const [successSnackbarVisible, setSuccessSnackbarVisible] =
  useState<boolean>(false);
const [deleteSnackbarVisible, setDeleteSnackbarVisible] =
  useState<boolean>(false);

const onSave = useCallback(() => {
  if (!updateState) {
    dispatch(
      logNewSet({

```

```

        categoryId,
        exerciseId,
        set: { weight, reps, createdAt: dayjs().toISOString() },
        date,
      )),
    );
  }
  if (updateState) {
    dispatch(
      updateSet({
        exerciseId,
        set: { weight, reps, createdAt: dayjs().toISOString() },
        setIndex: updateState.index,
        date,
      )),
    );
    setUpdateState(null);
  }
  Keyboard.dismiss();
  setSuccessSnackBarVisible(true);
}, [weight, reps, dispatch, updateState, date, exerciseId]);

const onDelete = useCallback(
  (exerciseId: string, index: number) => {
    setDeleteModalVisible(true);
    setDeleteModalInfo({ id: exerciseId, index });
  },
  [dispatch],
);

const onDeleteDismiss = useCallback(() => {
  setDeleteModalInfo(null);
  setDeleteModalVisible(false);
}, []);

const onClear = useCallback(() => {
  setWeight(0);
  setReps(0);
}, []);

const onDeleteSubmit = useCallback(() => {
  if (deleteModalInfo) {
    dispatch(
      deleteSetFromExercise({
        date,
        exerciseId: deleteModalInfo.id,
        setIndex: deleteModalInfo.index,
      )),
    );
  }

```

```

    );
    onDeleteDismiss();
    setDeleteSnackBarVisible(true);
  }
}, [deleteModalInfo, onDeleteDismiss, date]);

const onEdit = useCallback(
  (exerciseId: string, index: number) => {
    const selectedSet = workout?.sets[index];
    if (selectedSet) {
      setWeight(selectedSet.weight);
      setReps(selectedSet.reps);
    }
    setUpdateState({ exerciseId, index });
  },
  [workout],
);

const renderSet = (props: ListRenderItemInfo<Set>) => {
  const { item, index } = props;
  return (
    <SetDetails
      set={item}
      number={index + 1}
      isEditing={updateState?.index === index}
      onDeletePress={() => onDelete(exerciseId, index)}
      onEditPress={() => onEdit(exerciseId, index)}
    />
  );
};

const openExerciseHistory = useCallback(() => {
  navigation.navigate(WorkoutScreens.HISTORY, {
    exerciseId,
    categoryId,
  });
}, [navigation, exerciseId, categoryId]);

return (
  <View style={styles.root}>
    <View style={styles.header}>
      <View>
        <Text style={styles.selectedCategory}>
          {i18n.t('selected_exercise')}:
        </Text>
        <Text style={styles.selectedCategoryName}>
          {i18n.t(`${exercise?.name}`)}
        </Text>
      </View>
    </View>
  </View>
);

```

```

</View>
<IconButton
  size={18}
  mode={'outlined'}
  iconColor={Colors.brand.dark}
  style={styles.statsBtn}
  icon={'chart-timeline-variant'}
  onPress={openExerciseHistory}
/>
</View>
<WeightForm
  weight={weight}
  reps={reps}
  onWeightChange={setWeight}
  onRepsChange={setReps}
  onClearPress={onClear}
  onSave={onSave}
/>
<View style={styles.divider} />
<FlatList
  style={styles.setsList}
  contentContainerStyle={styles.setsContainer}
  data={workout?.sets ?? []}
  renderItem={renderSet}
/>
<DeleteConfirmation
  visible={deleteModalVisible}
  onDismiss={onDeleteDismiss}
  onSubmit={onDeleteSubmit}
/>
<Portal>
  <Snackbar
    visible={successSnackbarVisible}
    duration={3000}
    style={styles.snackBar}
    wrapperStyle={{ bottom: bottomTabBarHeight / 2 + 10 }}
    icon={'window-close'}
    onIconPress={() => {
      setSuccessSnackbarVisible(false);
    }}
    onDismiss={() => setSuccessSnackbarVisible(false)}>
    <Text
      style={styles.successText}>{i18n.t('training_saved')}</Text>
    </Snackbar>

    <Snackbar
      visible={deleteSnackbarVisible}
      duration={3000}

```

```

        style={styles.snackBar}
        wrapperStyle={{ bottom: bottomAppBarHeight / 2 + 10 }}
        icon={'window-close'}
        onIconPress={() => {
            setDeleteSnackBarVisible(false);
        }}
        onDismiss={() => setDeleteSnackBarVisible(false)}>
        <Text style={styles.successText}>{i18n.t('set_deleted')}</Text>
    </SnackBar>
</Portal>
</View>
);
};

const styles = StyleSheet.create({
  root: {
    ...StyleSheet.absoluteFillObject,
    backgroundColor: Colors.white,
  },
  setsContainer: {
    backgroundColor: Colors.white,
  },
  selectedCategory: {
    ...Fonts.regular_14,
    color: Colors.neutral.ghost,
  },
  selectedCategoryName: {
    ...Fonts.semibold_18,
    color: Colors.neutral.secondary,
  },
  setsList: {},
  header: {
    paddingHorizontal: 20,
    paddingVertical: 10,
    flexDirection: 'row',
    justifyContent: 'space-between',
    gap: 15,
  },
  divider: {
    height: 2,
    backgroundColor: Colors.neutral.line,
    marginTop: 10,
  },
  statsBtn: {
    borderRadius: 5,
    borderColor: Colors.brand.default,
  },
  deleteModalContainer: {

```

```

        backgroundColor: Colors.neutral.background,
        borderRadius: 10,
      },
      deleteTitle: {
        ...Fonts.semibold_24,
      },
      deleteDescription: {
        ...Fonts.regular_18,
      },
      alertBtn: {
        flex: 1,
        alignItems: 'center',
        justifyContent: 'center',
        borderRadius: 5,
        paddingVertical: 10,
      },
      alertBtnText: {
        ...Fonts.bold_16,
      },
      snackBar: {
        backgroundColor: Colors.button.success,
      },
      successText: {
        ...Fonts.bold_14,
        color: Colors.white,
      },
    },
  ));

```

A.1.24 workoutLog.tsx

```

import React, { useMemo, useState } from 'react';
import { PageWrapper } from '../components';
import {
  FlatList,
  StyleSheet,
  Text,
  TouchableOpacity,
  View,
} from 'react-native';
import { Colors, Fonts, SCREEN_HEIGHT, SCREEN_WIDTH } from '../consts';
import { createMaterialTopTabNavigator } from '@react-navigation/material-top-tabs';
import {
  HistoryNavigatorParamList,
  WorkoutNavigatorStackParamList,
} from '../navigators/types';

```

```

import { HistoryScreens, WorkoutScreens } from '../../types';
import { i18n } from '../../i18n';
import { useAppSelector } from '../../stores';
import {
  getExerciseHistory,
  getExerciseHistoryForPeriod,
  GraphCriteria,
  GraphPeriod,
  Set,
} from '../../stores/workouts';
import { RouteProp, useRoute } from '@react-navigation/native';
import { formatWeight } from '../../utils';
import dayjs from 'dayjs';
import { LineChart } from 'react-native-gifted-charts';
import { GraphPeriodSelector } from './components/graphPeriodSelector';
import { Dialog, Icon, Portal } from 'react-native-paper';

type DateWithSetsProps = {
  date: string;
  sets: Set[];
  categoryId: string;
  exerciseId: string;
};

const DateWithSets = (props: DateWithSetsProps) => {
  return (
    <View>
      <View style={styles.header}>
        <Text style={styles.exerciseName}>
          {dayjs(props.date).format('dddd, MMMM D YYYY')}
        </Text>
      </View>
      <View style={styles.setsContainer}>
        {props.sets.map(set => (
          <View key={set.createdAt} style={styles.setRow}>
            <Text style={Fonts.bold_16}>
              {formatWeight(set.weight)}
            <Text style={Fonts.regular_14}> {i18n.t('kgs')}</Text>
            </Text>
            <Text style={Fonts.bold_16}>
              {set.reps}
            <Text style={Fonts.regular_14}>
              { ' ' }
              {i18n.t('reps').toLowerCase()}
            </Text>
          </Text>
        </View>
      </View>
    </View>
  )
)}

```

```

        </View>
      </View>
    );
  };

export const WorkoutLog = () => {
  const route =
    useRoute<RouteProp<HistoryNavigatorParamList, HistoryScreens.LOG>>();
  const exerciseHistoryByDates = useAppSelector(state =>
    getExerciseHistory(state, route.params.exerciseId),
  );

  return (
    <PageWrapper style={{ backgroundColor: Colors.white }}>
      <FlatList
        contentContainerStyle={{ padding: 20, gap: 20 }}
        data={Object.keys(exerciseHistoryByDates)}
        renderItem={({ item }) => (
          <DateWithSets
            date={item}
            sets={exerciseHistoryByDates[item]}
            exerciseId={route.params.exerciseId}
            categoryId={route.params.categoryId}
          />
        )}
      />
    </PageWrapper>
  );
};

export const Graph = () => {
  const route =
    useRoute<RouteProp<HistoryNavigatorParamList, HistoryScreens.LOG>>();
  const [criteriaSelectorVisible, setCriteriaSelectorVisible] =
    useState<boolean>(false);
  const [graphHeight, setGraphHeight] = useState<number>(SCREEN_HEIGHT *
0.6);
  const [selectedPoint, setSelectedPoint] = useState<{
    value: any;
    date: string;
  } | null>(null);
  const [selectedPeriod, setSelectedPeriod] = useState<GraphPeriod>(
    GraphPeriod['1m'],
  );
  const [selectedCriteria, setSelectedCriteria] =
    useState<GraphCriteria>(
      GraphCriteria.MAX_WEIGHT,
    );
};

```

```

const exerciseHistoryByDates = useAppSelector(state =>
  getExerciseHistoryForPeriod(
    state,
    route.params.exerciseId,
    selectedPeriod,
    selectedCriteria,
  ),
);

const data = useMemo(() => {
  return Object.keys(exerciseHistoryByDates)
    .reverse()
    .map(date => {
      const value = exerciseHistoryByDates[date];
      return {
        value: value,
        date: date,
        label: dayjs(date).format('DD MMM'),
      };
    });
}, [exerciseHistoryByDates]);

return (
  <PageWrapper style={{ backgroundColor: Colors.white }}>
    <GraphCriteriaSelector
      selectedCriteria={selectedCriteria}
      onPress={() => setCriteriaSelectorVisible(true)}
    />
    <GraphPeriodSelector
      selectedPeriod={selectedPeriod}
      onChange={period => {
        setSelectedPoint(null);
        setSelectedPeriod(period);
      }}
    />
    <View
      style={styles.chartContainer}
      onLayout={layout => {
        setGraphHeight(layout.nativeEvent.layout.height - 40 * 2);
      }}
    >
      <LineChart
        height={graphHeight}
        width={SCREEN_WIDTH - 80}
        areaChart
        color={Colors.brand.dark}
        dataPointsColor={Colors.brand.dark}
        startFillColor={Colors.brand.outline}
      />
    </View>
  </PageWrapper>
);

```

```

        endFillColor={Colors.brand.outline}
        startOpacity={0.4}
        endOpacity={0.4}
        data={data}
        adjustToWidth
        formatYLabel={label => `${Math.round(Number(label) / 10) *
10}`}
        focusEnabled={true}
        showTextOnFocus={true}
        showStripOnFocus={true}
        focusedDataPointColor={Colors.brand.default}
        unFocusOnPressOut={false}
        onFocus={({props: any}) => {
            setSelectedPoint({ value: props.value, date: props.date });
        }}
        pointerConfig={{
            resetPointerOnDataChange: true,
            persistPointer: true,
        }}
    />
</View>
<View style={styles.selectedContainer}>
    {selectedPoint ? (
        <>
            <Text style={styles.selectedValueText}>
                {dayjs(selectedPoint.date).format('DD MMM YYYY')}:
            </Text>
            <Text style={styles.selectedValueText}>
                {selectedPoint.value}{' '}
                {selectedCriteria === GraphCriteria.MAX_WEIGHT
                    ? i18n.t('kgs')
                    : i18n.t('reps').toLowerCase()}
            </Text>
        </>
    ) : (
        <Text
style={styles.placeholder}>{i18n.t('graph_details')}</Text>
    )}
</View>
<SelectCriteriaModal
    visible={criteriaSelectorVisible}
    selectedCriteria={selectedCriteria}
    onDismiss={() => setCriteriaSelectorVisible(false)}
    onSubmit={criteria => {
        setSelectedPoint(null);
        setSelectedCriteria(criteria);
    }}
/>

```

```

    </PageWrapper>
  );
};

const styles = StyleSheet.create({
  btnContainer: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    gap: 10,
    paddingTop: 20,
    paddingHorizontal: '10%',
  },
  tabBtn: {
    backgroundColor: Colors.brand.focused,
    paddingVertical: 10,
    borderRadius: 5,
    flex: 1,
  },
  tabBtnActive: {
    backgroundColor: Colors.brand.default,
  },
  tabBtnText: {
    ...Fonts.semibold_18,
    color: Colors.white,
    textAlign: 'center',
  },
  header: {
    borderBottomWidth: 2,
    borderColor: Colors.neutral.ghost,
    paddingBottom: 5,
  },
  exerciseName: {
    ...Fonts.semibold_18,
  },
  setsContainer: {
    gap: 10,
    padding: 10,
    paddingHorizontal: 25,
  },
  setRow: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
  },
  chartContainer: {
    paddingHorizontal: 20,
    flex: 1,
  },
});

```

```

    marginTop: 20,
  },
  selectedContainer: {
    backgroundColor: Colors.neutral.background,
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'center',
    gap: 10,
    height: 40,
  },
  selectedValueText: {
    ...Fonts.semibold_18,
    color: Colors.neutral.primary,
  },
  placeholder: {
    ...Fonts.semibold_14,
    color: Colors.neutral.ghost,
  },
});

const Tab = createMaterialTopTabNavigator<HistoryNavigatorParamList>();

export const HistoryNavigator = () => {
  const route =
    useRoute<
      RouteProp<WorkoutNavigatorStackParamList, WorkoutScreens.HISTORY>
    >();
  const { exerciseId, categoryId } = route.params;
  return (
    <Tab.Navigator
      initialRouteName={HistoryScreens.LOG}
      screenOptions={{
        tabBarIndicatorStyle: { backgroundColor: Colors.brand.default },
        tabBarBounces: false,
        tabBarLabelStyle: {
          ...Fonts.semibold_18,
          textTransform: 'capitalize',
        },
      }}
    >
      <Tab.Screen
        name={HistoryScreens.LOG}
        component={WorkoutLog}
        initialParams={{ exerciseId, categoryId }}
        options={{ title: i18n.t('workout_log') }}
      />
      <Tab.Screen
        name={HistoryScreens.GRAPH}
        component={Graph}

```

```

        initialParams={{ exerciseId, categoryId }}
        options={{ title: i18n.t('graph') }}
      />
    </Tab.Navigator>
  );
};

const CriteriaToTextMap = {
  [GraphCriteria.MAX_REPS]: 'Max reps',
  [GraphCriteria.MAX_WEIGHT]: 'Max weight',
  [GraphCriteria.WORKOUT_REPS]: 'Reps per workout',
};

const GraphCriteriaSelector = ({
  selectedCriteria,
  onPress,
}: {
  selectedCriteria: GraphCriteria;
  onPress: () => void;
}) => {
  return (
    <View style={selectorStyles.container}>
      <Text style={selectorStyles.selectedCriteria}>Selected
criteria:</Text>
      <TouchableOpacity
        activeOpacity={0.8}
        style={selectorStyles.btn}
        onPress={onPress}>
        <Text style={selectorStyles.btnText}>
          {CriteriaToTextMap[selectedCriteria]}
        </Text>
        <Icon
          size={14}
          source={'chevron-down'}
          color={Colors.neutral.primary}
        />
      </TouchableOpacity>
    </View>
  );
};

type SelectCriteriaModalProps = {
  visible: boolean;
  selectedCriteria: GraphCriteria;
  onDismiss: () => void;
  onSubmit: (value: GraphCriteria) => void;
};

```

```

const Criteria = ({
  selected,
  criteria,
  description,
  onPress,
}: {
  selected: boolean;
  criteria: GraphCriteria;
  description: string;
  onPress: (value: GraphCriteria) => void;
}) => {
  return (
    <TouchableOpacity
      style={[
        dialogStyles.criteriaContainer,
        selected && dialogStyles.selectedCriteriaContainer,
      ]}
      activeOpacity={0.8}
      onPress={() => onPress(criteria)}>
      <View style={dialogStyles.textContainer}>
        <Text
          style={dialogStyles.name}>{CriteriaToTextMap[criteria]}</Text>
        <Text
          style={[
            dialogStyles.description,
            selected && dialogStyles.selectedDescription,
          ]}>
          {description}
        </Text>
      </View>
      {selected ? (
        <Icon size={20} source={'check'} color={Colors.brand.dark} />
      ) : null}
    </TouchableOpacity>
  );
};

const SelectCriteriaModal = ({
  visible,
  onDismiss,
  selectedCriteria,
  onSubmit,
}: SelectCriteriaModalProps) => {
  const dismiss = () => {
    onDismiss();
  };
  const onSelect = (criteria: GraphCriteria) => {
    onSubmit(criteria);
  };

```

```

    onDismiss();
  };
  return (
    <Portal>
      <Dialog
        style={dialogStyles.createModalContainer}
        onDismiss={dismiss}
        visible={visible}>
        <Dialog.Title style={Fonts.semibold_24}>Select
criteria</Dialog.Title>
        <Dialog.Content style={{ gap: 15 }}>
          <Criteria
            selected={selectedCriteria === GraphCriteria.MAX_WEIGHT}
            criteria={GraphCriteria.MAX_WEIGHT}
            description={'Maximum weight lifted per set'}
            onPress={onSelect}
          />
          <Criteria
            selected={selectedCriteria === GraphCriteria.MAX_REPS}
            criteria={GraphCriteria.MAX_REPS}
            description={'Maximum reps per set'}
            onPress={onSelect}
          />
          <Criteria
            selected={selectedCriteria === GraphCriteria.WORKOUT_REPS}
            criteria={GraphCriteria.WORKOUT_REPS}
            description={'Maximum reps per workout'}
            onPress={onSelect}
          />
        </Dialog.Content>
      </Dialog>
    </Portal>
  );
};

const dialogStyles = StyleSheet.create({
  createModalContainer: {
    backgroundColor: Colors.white,
    borderRadius: 10,
  },
  modalBtn: {
    flex: 1,
    alignItems: 'center',
    justifyContent: 'center',
    borderRadius: 5,
    paddingVertical: 10,
  },
  modalBtnText: {

```

```

    ...Fonts.bold_16,
  },
  criteriaContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    backgroundColor: Colors.neutral.background,
    paddingVertical: 15,
    paddingHorizontal: 20,
    borderRadius: 10,
  },
  selectedCriteriaContainer: {
    backgroundColor: Colors.brand.outline,
  },
  textContainer: {},
  name: {
    ...Fonts.semibold_18,
  },
  description: {
    ...Fonts.regular_14,
    color: Colors.neutral.ghost,
  },
  selectedDescription: {
    color: Colors.neutral.secondary,
  },
});

```

```

const selectorStyles = StyleSheet.create({
  container: {
    flexDirection: 'row',
    alignItems: 'center',
    gap: 10,
    paddingHorizontal: 20,
    paddingTop: 20,
    marginBottom: 10,
  },
  selectedCriteria: {
    ...Fonts.semibold_18,
    color: Colors.neutral.ghost,
  },
  btn: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    backgroundColor: Colors.brand.background,
    borderRadius: 6,
    padding: 10,
    gap: 5,
  },
});

```

```
    },  
    btnText: {  
      ...Fonts.semibold_14,  
      color: Colors.neutral.primary,  
    },  
  });
```

ДОДАТОК Б
Технічне завдання

Вступ

Найменування продукту – «Trackfit». Застосунок призначений для відстеження та перегляду аналітики тренувань.

Б.1 Підстави для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Мобільний застосунок для відстеження та аналітики тренувань», затверджене наказом Національного університету «Запорізька політехніка» № 168 від 24 квітня 2024 р.

Б.2 Призначення розробки

Метою розробки є створення застосунку для швидкого та зручного відстеження тренувань в будь-який момент.

Б.3 Вимоги до програми

Б.3.1 Вимоги до функціональних характеристик

Вимоги до функціональних характеристик застосунку:

- відсутність необхідності в авторизації;
- відсутність необхідності в інтернет з'єднанні;
- необмежений основний функціонал без необхідності оформлювати підписки чи робити будь-які інші покупки;
- відсутність реклами в застосунку;
- сучасний, інтуїтивний та простий інтерфейс;
- наявність версії для ОС Android;
- наявність версії для ОС iOS.

Б.3.2 Вимоги до інтерфейсу програми

Інтерфейс є надзвичайно важливим елементом для забезпечення зручності роботи користувачів з мобільним застосунком. Було визначено наступні вимоги до інтерфейсу програми:

- навігація та структура екранів повинні бути логічними, послідовними;
- інтерфейс повинен бути зручним та інтуїтивно зрозумілим у використанні, не потребувати додаткових зусиль від користувача для розуміння;
- графічний дизайн повинен бути сучасним, лаконічним та привабливим. Кольорова палітра повинна складатися із сумісних кольорів, шрифти та іконки підходити один до одного.

Б.3.3 Умови експлуатації

Умови експлуатації збігаються із умовами експлуатації мобільного пристрою, на якому встановлено застосунок.

Б.3.4 Вимоги до складу та параметрів технічних засобів

Технічні вимоги до обладнання:

- смартфон на базі ОС Android версії 6.0 або новіше;
- смартфон на базі ОС iOS версії 12.4 або новіше;
- 40 МБ вільного місця у сховищі.

Б.4 Вимоги до програмної документації

Програмна документація повинна включати пояснювальну записку на розроблений продукт та вихідний код програми.

Б.5 Порядок контролю й приймання

Передбачено захист дипломного проєкту у встановлені керівником проєкту строки.

ДОДАТОК В
Слайди презентації

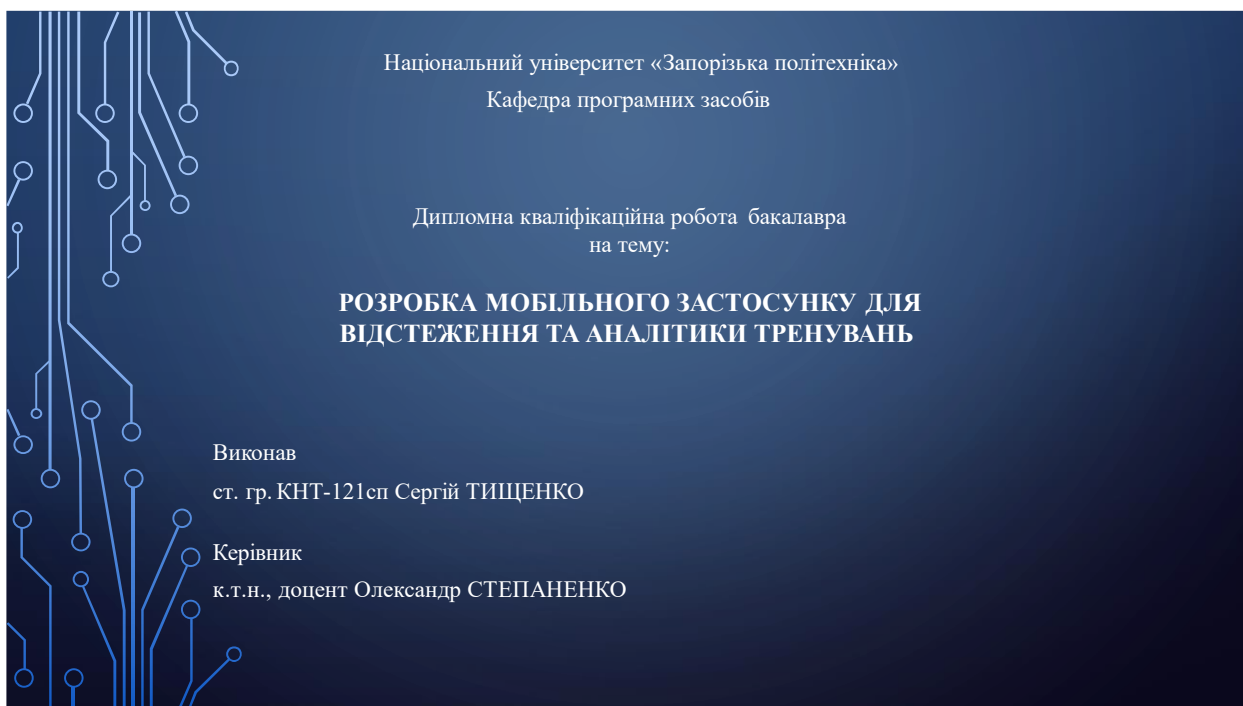


Рисунок В.1 – Слайд 1

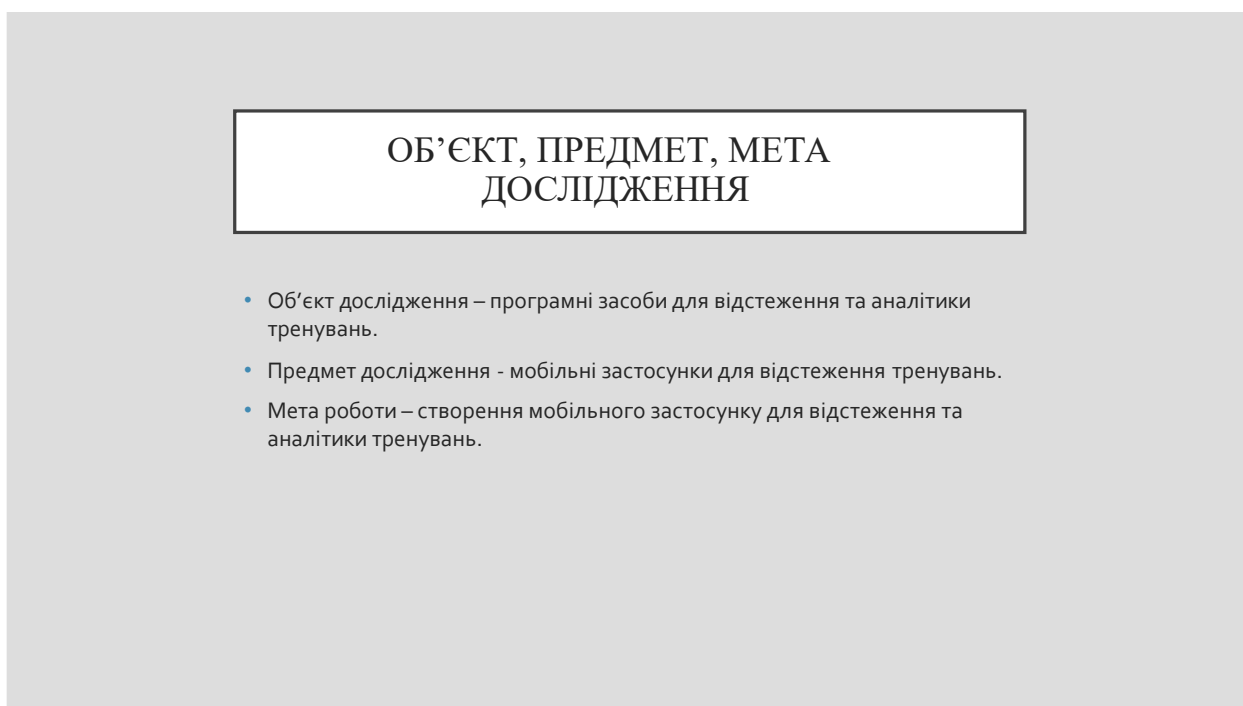


Рисунок В.2 – Слайд 2

ПОСТАНОВКА ЗАВДАННЯ

- Метою даної роботи є розробка мобільного застосунку, який забезпечить користувачам можливість ефективно відстежувати свої тренування та аналізувати дані локально, без інтеграції з зовнішніми сервісами.
- Вхідні дані для системи: дані про тренування, які вніс користувач.
- Вихідні дані: історія тренувань за датами, історія тренувань для кожної вправи, аналітика записаних тренувань за різними критеріями.

Рисунок В.3 – Слайд 3

ОБҐРУНТУВАННЯ ВИБОРУ ФРЕЙМВОРКУ

Критерій порівняння	React Native	Flutter	Ionic
Мова програмування	JavaScript	Dart	HTML, CSS, JavaScript
Активність і розмір спільноти	Велика та активна спільнота, з багатою документацією та ресурсами	Швидко зростаюча спільнота, з активною підтримкою Google	Зріла спільнота, з фокусом на веб-розробку
Гнучкість	Висока гнучкість, дозволяє використовувати власні компоненти та бібліотеки	Висока гнучкість, завдяки власному двигуну рендерингу	Висока гнучкість, за рахунок використання веб-технологій
Продуктивність	Висока, близька до нативної	Дуже висока, близька до нативної	Помірна, може бути нижча порівняно з нативними додатками

Рисунок В.4 – Слайд 4

ВИБІР МОВИ ПРОГРАМУВАННЯ

JavaScript	TypeScript
Інтерпретована мова програмування	Компільована мова програмування
Динамічна типізація	Безпечність типів
Потребує менше коду	Потребує більше коду
Помилки при виконанні	Більше помилок помітні при написанні коду
Легкий у вивченні	Потребує певного досвіду для використання
Більша швидкість розробки	Краща якість готового продукту

Рисунок В.5 – Слайд 5

ВІЗУАЛІЗАЦІЯ АРХІТЕКТУРИ

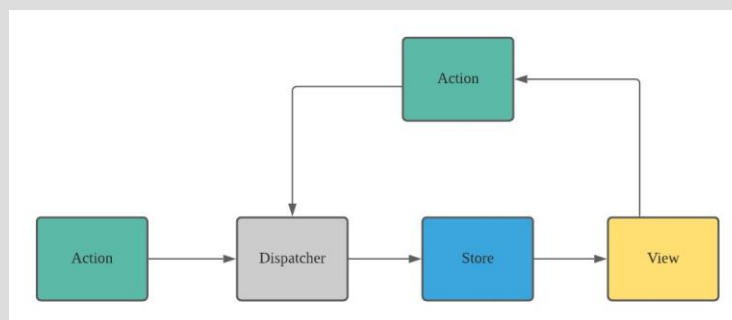


Рисунок В.6 – Слайд 6

ОСНОВНІ СУТНОСТІ

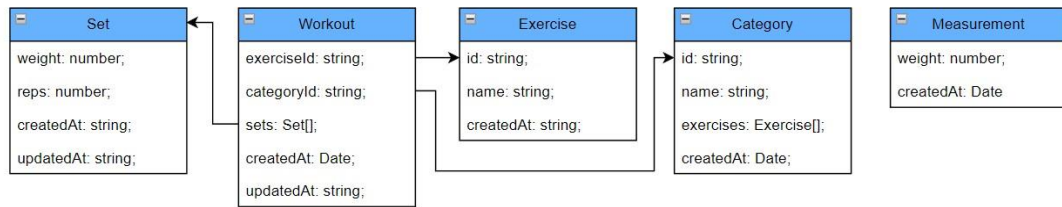


Рисунок В.7 – Слайд 7

СХЕМА СХОВИЩ

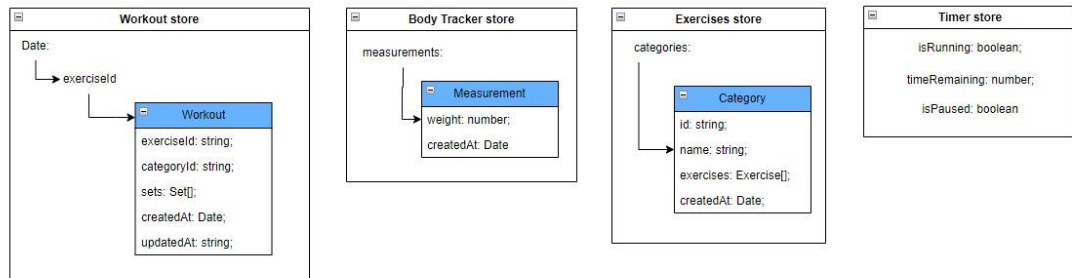


Рисунок В.8 – Слайд 8

ВИКОРИСТАНІ БІБЛІОТЕКИ

- Було використано наступний стек технологій:
 - React Native
 - Redux toolkit
 - Redux persist
 - React-native-async-storage
 - lodash

Рисунок В.9 – Слайд 9

ІНТЕРФЕЙС РОЗДІЛУ ТРЕНУВАНЬ

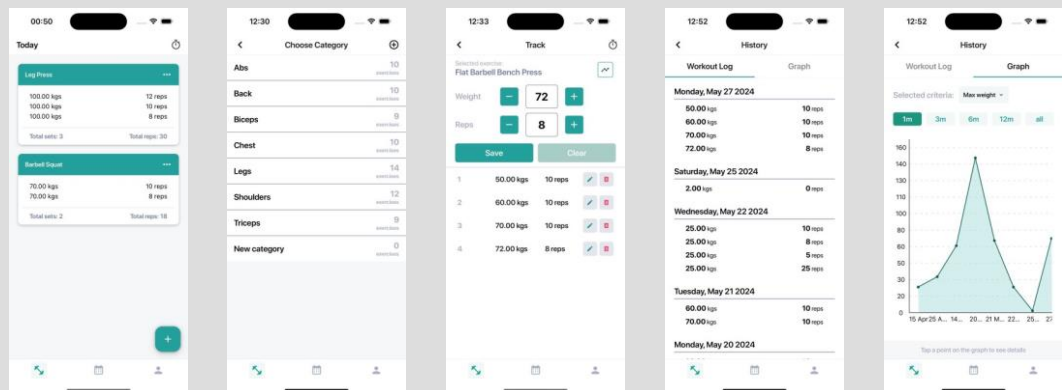


Рисунок В.10 – Слайд 10

ІНТЕРФЕЙС РОЗДІЛУ «КАЛЕНДАР»

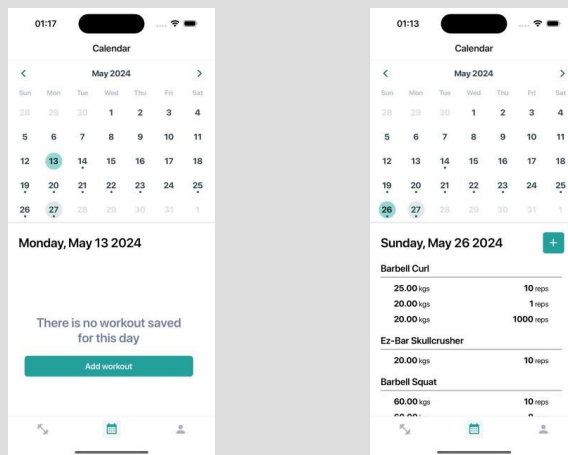


Рисунок В.11 – Слайд 11

ІНТЕРФЕЙС РОЗДІЛУ «ПРОФІЛЬ»

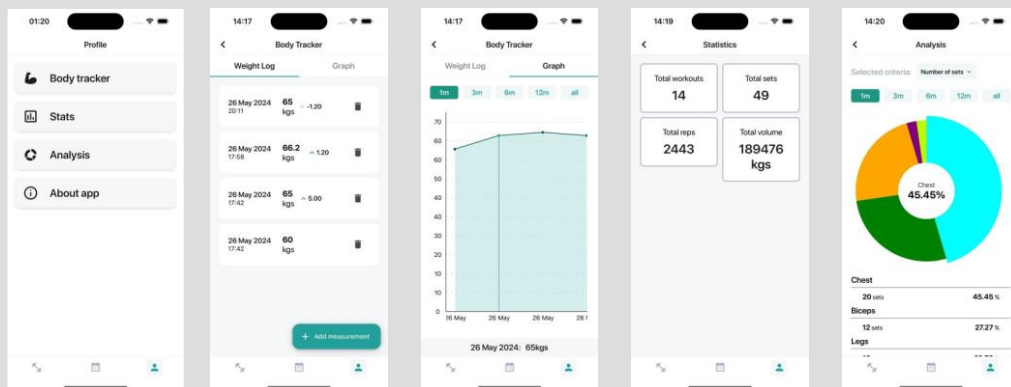


Рисунок В.12 – Слайд 12

ВИСНОВОК

- В результаті виконання кваліфікованої дипломної роботи бакалавра було розроблено мобільний застосунок для відстеження та аналітики тренувань
Застосунок призначений для людей, що підтримують здоровий спосіб життя та слідкують за своєю фізичною формою
- Він призначений для індивідуального використання при тренуваннях, дозволяє вести історію тренувань, слідкувати за прогресією та аналізувати статистику на дистанції.

Рисунок В.13 – Слайд 13