ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЛАНУВАННЯ ЩОДЕННИХ ВПРАВ ТА ВІДСТЕЖЕННЯ ПРОГРЕСУ В ОСОБИСТОМУ ТРЕНУВАЛЬНОМУ ПЛАНІ

зі спеціальності 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

ступеня магістра

Виконавець роботи Карабаш Вадим Євгенович
_____«____» _____202_ р.
(підпис)

Науковий керівник доцент, к.п.н. Кошова О. П.
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2025

РЕФЕРАТ

Записка: 58 с., 11 рис., 2 таблиці, 1 додаток, 10 джерел.

ТРЕНУВАННЯ, МОБІЛЬНИЙ ЗАСТОСУНОК, REACT NATIVE, ПРОГРЕС, OFFLINE, ASYNCSTORAGE, UX

Об’єкт дослідження – процес планування та відстеження тренувальної активності засобами мобільних програмних систем.

Предмет дослідження – методи й засоби проєктування та реалізації мобільного застосунку для формування особистого тренувального плану та вимірювання прогресу з локальним зберіганням даних і офлайн-роботою.

Мета роботи – спроєктувати та реалізувати мобільний застосунок для планування щоденних вправ та відстеження прогресу в особистому тренувальному плані, який працює офлайн, підтримує персоналізацію та надає наочну аналітику виконання тренувань.

Результатом роботи стало розроблення FitPlan — мобільного застосунку на базі React Native (Expo) і TypeScript із локальним зберіганням даних в AsyncStorage (JSON-структури). Реалізовано ключові модулі:

• Тренувальний план – створення і редагування персональних програм, розподіл вправ по днях, підтримка сплітів (фулбоді, верх/низ, PPL).

• Виконання тренування – покрокове виконання вправ, таймери, введення підходів/повторень/ваги, автоматичний підрахунок загального обсягу роботи.

• Історія тренувань – журнал завершених сесій із деталізацією параметрів навантаження.

• Прогрес – аналітичні показники (кількість сесій, підходів, тоннаж, частота тренувань), тижневі/місячні огляди.

• Профіль – персональні параметри користувача, рівень підготовки, доступні дні, персоналізовані рекомендації.

• Фітнес-центри – інтегрована карта тренажерних залів (геолокація, маркери).

• Нагадування – локальні push-сповіщення про заплановані тренування.

Особливості: повністю офлайн-робота без акаунта; модульна архітектура;

адаптивні алгоритми формування тренувальних планів; підтримка теми інтерфейсу; розширювана структура моделей; інтуїтивний UX для щоденного використання.

Проведено тестування якості: модульні тести алгоритмів формування планів і підрахунку навантаження, інтеграційні сценарії виконання тренувань та збереження історії, поведінкові BDD-тести (Given–When–Then) для ключових користувацьких дій. За результатами тестування підтверджено стабільність офлайн-роботи, коректність логіки прогресу та узгодженість інтерфейсу.

Застосунок може бути використаний як особистий інструмент фітнес-планування, як навчальний приклад для курсів з мобільного програмування та як основа для розширення функціональності у напрямі синхронізації даних, адаптивних тренувальних програм, інтеграції з носимими пристроями та системами моніторингу активності.

ЗМІСТ

СПИСОК УМОВНИХ ПОЗНАЧЕННЬ, СИМВОЛІВ, СКОРОЧЕНЬ, ТЕРМІНІВ

Умовні позначення, символи, скорочення, терміни	Пояснення умовних позначень, скорочень, символів
RN	React Native – фреймворк для кросплатформеної мобільної розробки
Expo	Інструментарій для швидкого запуску й тестування застосунків на React Native
TS	TypeScript – мова програмування з статичною типізацією
AsyncStorage	Локальне сховище даних у мобільному застосунку
State	Поточний стан інтерфейсу або логіки у застосунку
Store	Сховище глобального стану (Zustand Store)
JSON	Формат обміну структурованими даними
UUID	Унікальний ідентифікатор об’єкта
API	Application Programming Interface – прикладний програмний інтерфейс
UI	User Interface – інтерфейс користувача
UX	User Experience – користувацький досвід
Set	Підхід у силових тренуваннях (виконання певної кількості повторень)
Rep	Повторення у силовій вправі
Session	Тренувальна сесія, одне повноцінне тренування
Progress	Модуль системи, відповідає за облік і аналіз прогресу користувача

ВСТУП

Стрімке поширення мобільних технологій та зростання інтересу суспільства до здорового способу життя формують стійкий запит на інструменти, які допомагають системно підходити до фізичної активності. Більшість користувачів прагнуть не лише виконувати окремі вправи, а будувати логічні тренувальні плани, відстежувати прогрес, бачити динаміку результатів і коригувати навантаження відповідно до власних цілей.

Разом із цим значна частина наявних рішень для фітнесу має низку обмежень. Часто застосунки орієнтовані на нав’язування готових «універсальних» програм без можливості гнучкого налаштування під конкретного користувача. Багато сервісів потребують постійного доступу до мережі, обов’язкової реєстрації та передавання персональних даних на сторонні сервери. Інтерфейси окремих продуктів перевантажені другорядними функціями, що ускладнює щоденне використання і демотивує користувача підтримувати регулярність тренувань.

Це актуалізує потребу в простому, зрозумілому, орієнтованому на офлайн-режим мобільному застосунку, який дозволяє планувати щоденні вправи, формувати власні тренувальні плани, фіксувати виконання та бачити наочний прогрес без зайвих дій і залежності від хмарної інфраструктури. Такий інструмент має підтримувати базові сценарії: створення й редагування вправ, планування тренувань на календарі, ведення журналу виконання, перегляд статистики (кількість тренувань, обсяг навантаження, виконання цілей) та управління персональними параметрами користувача (ціль, рівень підготовки, обмеження).

Мобільний застосунок, що розробляється в межах даної роботи, орієнтований саме на підтримку особистого тренувального плану. Користувач отримує змогу створювати власні вправи та комплекси вправ, планувати тренування по днях тижня, відмічати їх виконання, спостерігати динаміку (кількість підходів, повторень, тривалість, вагу обтяжень тощо) та коригувати план відповідно до досягнутого прогресу. Технічно застосунок реалізується як кросплатформене рішення на базі React Native (Expo, TypeScript) з локальним зберіганням даних у

AsyncStorage, що забезпечує офлайн-роботу, швидкий старт і відсутність серверної складової.

Поєднання простого користувацького інтерфейсу з можливістю гнучкого налаштування планів тренувань дозволяє підвищити дисципліну виконання вправ, зменшити ризики «спонтанних» відмов від занять та зробити процес тренувань більш усвідомленим. Офлайн-режим і локальне зберігання даних відповідають вимогам приватності, а кросплатформеність дає змогу використовувати застосунок на різних пристроях без прив’язки до конкретної операційної системи.

Мета роботи – спроєктувати та реалізувати мобільний застосунок для планування щоденних вправ та відстеження прогресу в особистому тренувальному плані, який працює офлайн, підтримує персоналізацію та надає наочну аналітику виконання тренувань.

Об’єкт дослідження – процес планування та відстеження тренувальної активності засобами мобільних програмних систем.

Предмет дослідження – методи й засоби проєктування та реалізації мобільного застосунку для формування особистого тренувального плану та вимірювання прогресу з локальним зберіганням даних і офлайн-роботою.

У роботі використовуються: аналітичний огляд наукових і практичних джерел з питань фітнесу та мобільних застосунків; методи програмної інженерії для формалізації вимог та проєктування архітектури; алгоритмічні підходи до моделювання тренувальних планів і показників прогресу (кількість тренувань, обсяг навантаження, виконання цілей); експериментальна перевірка працездатності та стабільності застосунку; модульне та інтеграційне тестування.

Наукова новизна роботи полягає в поєднанні персоналізованого планування тренувань із простим офлайн-застосунком без серверної інфраструктури, а також у структуризації даних про вправи, тренувальні плани й показники прогресу таким чином, щоб вони були придатні як для повсякденного використання, так і для подальшого розширення функціоналу (додавання рівнів складності, адаптивних планів, інтеграції із зовнішніми датчиками).

Практичне значення одержаних результатів полягає у створенні мобільного застосунку, який може бути використаний як інструмент для планування та відстеження особистих тренувань, як навчальний приклад для дисциплін з мобільної розробки та проєктування інтерфейсів, а також як основа для подальших досліджень і розширення у бік інтеграцій із фітнес-трекерами та аналітикою фізичної активності.

1. ПОСТАНОВКА ЗАДАЧІ

Сучасні мобільні технології створюють широкі можливості для підтримки фізичної активності, однак значна частина існуючих застосунків не забезпечує достатнього рівня персоналізації, зручності та автономності. Більшість рішень пропонує фіксовані тренувальні програми, перевантажені інтерфейси або вимагає постійного доступу до мережі та передавання персональних даних на сторонні сервери. Це знижує мотивацію користувачів та ускладнює формування стабільної тренувальної звички. Водночас зростає потреба у простому та зрозумілому інструменті, який дозволяє самостійно планувати тренування, контролювати виконання вправ і відстежувати прогрес у зручній формі.

Постановка задачі полягає у створенні мобільного застосунку, який забезпечує підтримку персонального тренувального процесу, дозволяє формувати індивідуальні плани занять, фіксувати виконані вправи та аналізувати динаміку фізичної активності користувача. Система повинна працювати офлайн, зберігати дані локально, мати інтуїтивно зрозумілий інтерфейс та забезпечувати швидку взаємодію користувача з основним функціоналом.

Для реалізації мети необхідно вирішити такі **завдання:**

1. Проаналізувати предметну область, підходи до побудови тренувальних програм та існуючі мобільні рішення для фітнесу.
2. Визначити функціональні та нефункціональні вимоги до мобільного застосунку.
3. Обґрунтувати вибір технологічного стеку та архітектурних рішень.
4. Розробити моделі даних для вправ, тренувальних планів, історії виконання та показників прогресу.
5. Реалізувати ключові модулі застосунку: каталог вправ, планування тренувань, активне тренування, історію, статистику та профіль користувача.
6. Забезпечити збереження даних в офлайн-режимі та коректне відновлення стану після перезапуску застосунку.
7. Провести тестування працездатності, стабільності та відповідності вимогам.
8. Підготувати інструкцію користувача та описати результати впровадження.

Реалізація поставленої задачі має привести до створення повноцінного мобільного застосунку, який дозволяє користувачу планувати вправи, контролювати їх виконання та відстежувати особистий прогрес у зручний і наочний спосіб, сприяючи формуванню регулярних тренувальних звичок та досягненню фітнес-цілей.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1. Основи побудови тренувальних програм

Побудова ефективної тренувальної програми ґрунтується на принципах фізіології, біомеханіки та спортивної науки, що визначають раціональний розподіл навантаження, вибір вправ і послідовність їх виконання. Незалежно від рівня підготовки користувача, тренувальна програма повинна враховувати індивідуальні цілі, можливості організму, час на відновлення та поступовість підвищення інтенсивності. Основними складовими будь-якої програми є ціль тренувального процесу, підбір вправ, обсяг та інтенсивність навантаження, структура тренувального заняття і контроль прогресу.

Одним з ключових принципів є принцип специфічності, що передбачає відповідність виконуваних вправ поставленим цілям: розвиток сили, витривалості, координації чи загальної фізичної підготовки. Для силових тренувань зазвичай підбирають багатосуглобові вправи (присідання, жим лежачи, тяги), тоді як для розвитку витривалості — вправи з тривалим циклічним навантаженням. Другим базовим принципом є прогресивне навантаження, відповідно до якого збільшення ваги, кількості повторень або тривалості повинно відбуватися поступово, щоб забезпечити адаптацію організму та уникнути перенавантаження. [1]

Важливим елементом є також періодизація, яка передбачає планування тренувального процесу на певні цикли (мікро-, мезо- та макроцикли) з чергуванням фаз навантаження і відновлення. Це дає змогу оптимізувати прогрес та зменшити ризик травм. У рамках аматорських тренувань періодизація може проявлятися у чергуванні тренувань різної інтенсивності протягом тижня.

Структура одного тренування зазвичай включає три етапи:

– розминка для підготовки опорно-рухового апарату та серцево-судинної системи;

– основна частина, де виконуються вправи згідно з програмою;

– заминка, що сприяє відновленню й зменшує ризик м’язової крепатури.

Підбір кількості підходів і повторень залежить від цілі: для розвитку сили —

менше повторень з більшою вагою, для гіпертрофії — середні діапазони, для витривалості — більша кількість повторень з мінімальним опором. Контроль виконання підходів, тривалості відпочинку та загального обсягу тренування є необхідною умовою системного прогресу.

Ще одним важливим компонентом є моніторинг результатів. Запис виконаних вправ, повторень, ваги чи часу дає змогу оцінювати динаміку, виявляти прогрес або його відсутність, а також коригувати тренувальний план. Саме тому тренувальні журнали та цифрові інструменти дедалі частіше стають невід'ємною частиною тренувального процесу. [2]

Таким чином, основи побудови тренувальних програм передбачають поєднання наукових принципів тренування, індивідуальних цілей користувача та системного контролю прогресу. Застосунок, який підтримує фіксацію даних, планування вправ і аналіз результатів, може значно підвищити ефективність тренувального процесу, забезпечуючи зручність та сталість виконання програм.

2.2. Підходи до відстеження фізичної активності

Відстеження фізичної активності є ключовим елементом сучасних систем фітнес-контролю, оскільки дає змогу оцінювати реальне навантаження, аналізувати прогрес і коригувати тренувальні програми відповідно до фактичних результатів. Існує декілька підходів до моніторингу активності, які різняться за рівнем точності, технічними вимогами та глибиною отриманих даних. Для мобільних застосунків особливо важливими є простота збору інформації, мінімальна залежність від додаткових пристроїв та можливість працювати офлайн.

Перший підхід — суб'єктивне відстеження, засноване на ручному введенні даних користувачем. До таких даних належать виконані вправи, кількість повторень, підходів, вага обтяжень, тривалість тренування та рівень складності. Цей підхід має високу гнучкість, оскільки дозволяє враховувати будь-які види активності, включно з тими, що не фіксуються автоматизовано (наприклад, тренування з власною вагою або нестандартні рухові комплекси). Недоліком є

суб’єктивність і залежність від дисципліни користувача, проте для персональних тренувальних планів він залишається найбільш універсальним. [2]

Другий підхід — автоматизоване відстеження, засноване на використанні сенсорів смартфона або зовнішніх пристроїв (фітнес-браслети, смарт-годинники). У таких системах застосовуються акселерометри, гіроскопи, датчики частоти серцевих скорочень та GPS. Автоматичний збір даних дозволяє відстежувати кроки, швидкість, дистанцію, частоту пульсу, рівень активності та енергетичні витрати. Цей підхід забезпечує високу точність, але потребує інтеграції зі сторонніми API та постійної або періодичної синхронізації даних, що ускладнює створення повністю офлайн-рішень.

Третій підхід — комбінований, коли частина даних вводиться вручну, а частина може бути отримана із сенсорів, якщо вони доступні. Наприклад, користувач може самостійно вказувати кількість повторень і вагу, тоді як тривалість тренування або орієнтовні енерговитрати можуть розраховуватися автоматично. Такий підхід забезпечує баланс між точністю та зручністю, але його реалізація залежить від підтримки відповідних платформних сервісів (Google Fit, Apple Health) і прав доступу. [3]

Для застосунків, орієнтованих на персональні тренування з широкою варіативністю вправ, найбільш ефективним виявляється модель ручного або напівручного відстеження, у якій користувач фіксує ключові параметри: кількість підходів, повторень, вагу обтяжень, суб’єктивне навантаження (RPE) та загальну тривалість. Такий підхід не потребує спеціалізованих датчиків, гарантує офлайн-роботу та дозволяє адаптувати систему під будь-який стиль тренувань — від силових до функціональних чи реабілітаційних.

Важливою складовою відстеження активності є побудова історії тренувань, що дозволяє оцінювати динаміку змін і виявляти тенденції. Журнали тренувань фіксують дані за датами, дають змогу бачити прогрес щодо сили, витривалості або частоти тренувань. На основі таких історичних даних можуть формуватися прості метрики: кількість тренувань за тиждень, обсяг виконаної роботи, прогрес у конкретних вправах. Навіть базова аналітика є корисною для підтримання мотивації

та коригування навантаження. [4]

Таким чином, сучасні підходи до відстеження фізичної активності охоплюють як повністю автоматизовані, так і ручні моделі збору даних. У межах мобільного застосунку, що працює офлайн та орієнтований на персональні тренувальні плани, найбільш доцільним є використання ручного введення з подальшим автоматичним формуванням статистики та графіків прогресу. Це забезпечує гнучкість, універсальність, простоту реалізації і дає користувачу повний контроль над даними.

2.3. Вимоги до мобільного застосунку

Для формування цілісного уявлення про майбутню систему та визначення обсягу робіт вимоги до мобільного застосунку було узагальнено у таблиці, що включає основні групи характеристик: функціональні можливості, нефункціональні параметри, технічні обмеження та вимоги до інтерфейсу користувача (див. табл. 2.1).

Таблиця 2.1 – Системні вимоги до мобільного застосунку FitPlan

Категорія вимоги	*Формулювання вимоги*	*Очікуваний результат*
Функціональні вимоги		
Управління вправами	*Застосунок має містити каталог вправ із описом, рівнем складності та зображеннями*	*Користувач може переглядати та обирати вправи для тренувань*
Планування тренувань	*Можливість створення, редагування та запуску тренувального заняття*	*Формування персонального тренувального плану*
Облік виконання	*Застосунок повинен дозволяти фіксувати підходи, повторення, вагу та тривалість*	*Реєстрація фактичного навантаження за кожне тренування*
Історія	*Зберігання журналу завершених тренувань із деталізацією*	*Перегляд прогресу користувача у динаміці*
Статистика	*Автоматичне розрахування метрик (обсяг роботи, кількість вправ, частота тренувань)*	*Користувач отримує аналітику результатів*

Профіль	*Можливість зберігати персональні дані користувача*	*Персоналізація роботи застосунку*
Геолокація та мапи	*Відображення фітнес-центрів на карті*	*Додаткові можливості навігації для користувача*
Сповіщення	*Нагадування про заплановані тренування*	*Підвищення регулярності занять*
Нефункціональні вимоги		
Офлайн-робота	*Застосунок повинен працювати без підключення до мережі*	*Доступність функцій у будь-яких умовах*
Продуктивність	*Час завантаження не повинен перевищувати 2–3 секунди*	*Швидка взаємодія з інтерфейсом*
Надійність	*Дані повинні зберігатися стабільно в локальному сховищі*	*Уникнення втрати користувацької інформації*
Безпека	*Зберігання персональних даних у захищеному локальному контейнері*	*Неможливість несанкціонованого доступу*
Масштабованість	*Структура даних має дозволяти додавання нового функціоналу*	*Потенціал для розширення системи*
Технічні вимоги		
Платформа	*Підтримка Android та iOS*	*Кросплатформена доступність*
Технологічний стек	*React Native, TypeScript, Expo*	*Єдність архітектури та інструментів*
Збереження даних	*AsyncStorage для постійного зберігання та Zustand для стану*	*Стабільність роботи офлайн*
Доступи системи	*Підтримка дозволів: камера, медіафайли, геолокація, push*	*Взаємодія з апаратними можливостями*
Вимоги до інтерфейсу та UX		
Простота навігації	*Інтерфейс має бути інтуїтивним, з мінімальною кількістю дій для основних сценаріїв*	*Зручність використання*
Адаптивність	*Коректне відображення на різних розмірах екранів*	*Універсальність інтерфейсу*
Візуальна структура	*Використання зрозумілих візуальних акцентів та елементів керування*	*Покращене сприйняття інформації*
Мова інтерфейсу	*Повна локалізація українською*	*Відповідність очікуванням*

	мовою	*цільової аудиторії*

Узагальнена таблиця дозволяє систематизувати вимоги та забезпечує однозначне розуміння того, які функції та властивості повинні бути реалізовані в застосунку. Функціональні вимоги визначають основні можливості системи, тоді як нефункціональні забезпечують якість її роботи. Технічні вимоги окреслюють платформи та інструменти, необхідні для реалізації, а UX-вимоги формують підхід до побудови зрозумілого та зручного інтерфейсу.

Таке структурування вимог є важливою основою для подальшого проєктування архітектури, моделювання даних та реалізації функціональних модулів мобільного застосунку.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Архітектура мобільного застосунку

Архітектура мобільного застосунку для планування щоденних вправ та відстеження прогресу побудована за багаторівневим підходом і орієнтована на офлайн-роботу. Логіка застосунку розділена на презентаційний рівень (екрани та навігація), рівень бізнес-логіки (модулі роботи з тренуваннями, вправами, прогресом), рівень зберігання даних (репозиторії над локальним сховищем) та рівень системних сервісів (сповіщення, доступ до ресурсів пристрою).

Технічно застосунок реалізовано з використанням React Native (Expo, TypeScript). У якості локального сховища використовується AsyncStorage, над яким побудовано шар репозиторіїв, що приховують деталі зберігання. Екрани взаємодіють із бізнес-логікою через контексти стану та сервіси, а бізнес-логіка — з репозиторіями, не знаючи про конкретну реалізацію сховища. Це підвищує модульність і спрощує супровід.

Для візуалізації загальної структури застосунку використовується багаторівнева блок-схема (див. рис. 3.1).

Рисунок 3.1 – Загальна архітектура мобільного застосунку

На верхньому рівні знаходиться користувач, який взаємодіє із застосунком через набір екранів: «План тренувань», «Календар», «Прогрес», «Профіль» тощо. Ці екрани належать до презентаційного рівня, реалізованого у вигляді компонентів React Native із використанням React Navigation для організації навігації між вкладками та стеком екранів.

Презентаційний рівень не містить бізнес-логіки: він лише відображає дані та ініціює події (створити тренування, відмітити виконання, змінити параметри профілю). Вся доменна поведінка зосереджена у модулях бізнес-логіки (WorkoutService, ExerciseService, ProgressService, NotificationService). Вони відповідають за формування тренувальних планів, ведення журналу виконання, обчислення статистичних показників (кількість тренувань, обсяг навантаження,

виконання цілей), а також за планування локальних нагадувань. [5]

Для доступу до даних використовується окремий шар репозиторіїв (WorkoutRepository, ExerciseRepository, ProgressRepository). Репозиторії інкапсулюють операції читання/запису та приховують деталі реалізації AsyncStorage. Завдяки цьому бізнес-логіка працює з абстракціями «тренування», «вправа», «запис прогресу» і не залежить від того, як саме ці сутності зберігаються фізично.

На найнижчому рівні розташовано локальне сховище даних — AsyncStorage. Дані про вправи, тренувальні плани, історію виконання та налаштування профілю зберігаються у вигляді JSON-структур, прив’язаних до ключів. Така організація дозволяє забезпечити офлайн-режим та збереження стану між запусками застосунку.

Паралельно із шаром репозиторіїв працюють системні сервіси пристрою: push-сповіщення, файлове сховище (для експорту/імпорту), календар, мережа (у майбутніх версіях — для синхронізації). Доступ до цих сервісів здійснюється через відповідні модулі, що ізолюють платформено-залежну логіку. [6]

Таким чином, архітектура забезпечує чітке розділення відповідальностей: інтерфейс працює лише з відображенням і подіями, бізнес-логіка — з доменною поведінкою, репозиторії — із зберіганням, а нижній рівень відповідає за фізичне розміщення даних та взаємодію з системними сервісами.

Сценарій виконання тренування (див. рис. 3.2) починається на рівні користувача: він відкриває екран «План тренувань», обирає заплановане заняття та натискає «Почати тренування». Презентаційний рівень завантажує план через відповідний сервіс чи контекст і відображає структуру тренування: вправи, кількість підходів, таймери відпочинку тощо.

Рисунок 3.2 – Потік даних при виконанні тренування

Під час виконання вправ користувач генерує події: додає завершені підходи, змінює вагу обтяжень, позначає вправу як виконану, завершує сесію. Ці події передаються в модулі бізнес-логіки. WorkoutService оновлює поточний стан тренування (фактична кількість підходів/повторень, час). Після завершення сеансу ProgressService обчислює агреговані показники — загальний обсяг роботи, тривалість, кількість виконаних вправ.

Далі формується структурований запис історії тренування, який передається на рівень зберігання: WorkoutRepository оновлює інформацію про план (наприклад, відмічає тренування як виконане), а ProgressRepository додає новий запис до журналу. Обидва репозиторії використовують AsyncStorage для фізичного

збереження даних у форматі JSON. [7]

Після успішного збереження стан контексту прогресу оновлюється, і екрани «Прогрес» та «Статистика» автоматично отримують нові дані. Це забезпечує негайне відображення змін для користувача без додаткових дій.

3.2. Моделі даних

Моделі даних мобільного застосунку відображають основні сутності предметної області: вправи, тренувальні плани, фактичні тренування (сесії), показники прогресу та профіль користувача. Дані організовані так, щоб одночасно забезпечити зручність повсякденного використання (простота форм) і технічну стійкість до змін (однозначні ідентифікатори, явні одиниці вимірювання, узгоджені формати дат).

Усі моделі задаються у вигляді типів TypeScript і зберігаються у локальному сховищі у форматі JSON. Для позначення дат використовується формат YYYY-MM-DD, для часу — HH:MM у локальному часовому поясі користувача. Одиниці вимірювання фіксовані: маса обтяжень задається у кілограмах, кількість повторень та підходів — цілими числами, тривалість — у хвилинах або секундах (залежно від контексту).

Основна сутність — вправа (Exercise). Вона описує базову одиницю тренувального процесу, яка може повторно використовуватися у різних планах і сесіях. Типова модель вправи включає: унікальний ідентифікатор (рядок), назву вправи, цільову групу м’язів, тип (силова, кардіо, розтяжка), рекомендовані параметри (кількість підходів, діапазон повторень, орієнтовну вагу або тривалість), рівень складності та короткий опис техніки виконання. Такий опис дозволяє як будувати стандартні програми, так і швидко додавати власні вправи користувача.

Другою ключовою сутністю є тренувальний план (WorkoutPlan). План представляє собою структурований набір тренувальних сесій, прив’язаний до днів тижня або до конкретних дат. У моделі плану зберігаються: ідентифікатор, назва плану, опис (наприклад, «спліт на 3 дні» чи «фулбоді 2 рази на тиждень»), ціль

(схуднення, набір сили, загальна витривалість), рівень підготовки (початковий, середній, просунутий), а також структура розкладу. Структура розкладу задається або у вигляді мапи «день тижня → список вправ», або у вигляді списку «дата → заплановане тренування» з посиланнями на базові вправи та їх параметри. Важливо, що план містить цільові значення (наприклад, 3 підходи по 10 повторень), а не фактичні — це дозволяє чітко розділити «план» і «реалізацію». [8]

Фактичне виконання фіксується в окремій сутності тренувальна сесія (WorkoutSession). Сесія прив’язана до конкретної дати (а за потреби — й до часу початку та завершення) та містить список виконаних вправ з фактичними параметрами: реальною кількістю підходів, повторень, використаною вагою, тривалістю, паузами відпочинку, відміткою про завершення. Для кожної вправи в сесії зберігається посилання на базову вправу (Exercise) та окремий список фактичних сетів (наприклад, [10 повторень × 40 кг, 8 повторень × 45 кг, 6 повторень × 50 кг]). Модель сесії також містить статус (запланована, у процесі, завершена) та короткі нотатки користувача (самопочуття, коментарі щодо навантаження). Такий поділ дозволяє зберегти історію змін: навіть якщо згодом план чи вправа будуть відредаговані, дані про конкретні виконані тренування залишаться незмінними.

Для агрегування інформації застосовується логічна сутність прогрес (Progress). У сховищі не зберігаються окремі «таблиці» прогресу, натомість модуль аналітики використовує історію тренувальних сесій для обчислення похідних показників: кількості тренувань за тиждень, загальної кількості підходів та повторень, сумарного об’єму роботи (наприклад, тоннаж для силових вправ), середньої тривалості занять, відсотка виконаних запланованих тренувань. Для оптимізації можуть зберігатися невеликі кешовані підсумки (наприклад, агрегати за тиждень), але базовою «точкою істини» все одно залишаються окремі WorkoutSession. Це спрощує підтримку та мінімізує ризик розбіжностей між «сирими» даними й аналітикою.

Окрему роль відіграє модель профілю користувача (UserProfile). Вона містить статичні та напівстатичні характеристики: ім’я, вік, стать, рівень фізичної підготовки, наявність обмежень (протипоказання, чутливі зони), бажану кількість

тренувань на тиждень, обраний тип цілей (за кількістю сесій, за тривалістю, за обсягом роботи). На основі цих параметрів застосунок формує рекомендовані навантаження та допомагає інтерпретувати результати прогресу (наприклад, чи досягнуто цілі «3 тренування на тиждень»). У профілі також можуть зберігатися налаштування нагадувань (дні та час сповіщень) та параметри інтерфейсу (тема, одиниці вимірювання). [9]

Для службових потреб використовується невелика модель налаштувань застосунку (AppSettings), яка зберігає, зокрема, обрану мову, тему оформлення, прапор проходження онбордингу та версію схеми даних. Наявність номера версії у сховищі дає змогу безпечно виконувати міграції моделей (наприклад, додавати нові поля до Exercise чи WorkoutSession, не втрачаючи старі дані).

Усі моделі побудовані з урахуванням таких принципів:

- явні одиниці вимірювання (кг, повторення, хвилини) замість «безрозмірних» чисел;
- стабільні ідентифікатори (рядки на основі часових міток або UUID), які дозволяють безпечно посилатися на сутності з різних частин застосунку;
- розділення планових та фактичних даних (WorkoutPlan проти WorkoutSession), щоб історія тренувань не змінювалася при редагуванні планів;
- мінімальна необхідна денормалізація (збереження фактичних значень без перерахунку через поточний план), що підвищує надійність при оновленні логіки та довідників.

Такий підхід до моделювання даних забезпечує прозору структуру інформації, придатну як для щоденного використання, так і для подальшого розширення функціоналу (додавання нових типів вправ, складніших показників прогресу, синхронізації з зовнішніми пристроями).

3.3. Алгоритми формування тренувальних планів

Формування тренувального плану в мобільному застосунку базується на

послідовності формалізованих кроків: від збору вхідних параметрів профілю до побудови конкретних тренувальних сесій і їхньої подальшої корекції на основі фактичного виконання. У даному підрозділі наведено основні алгоритми у вигляді чітких кроків і псевдокоду.

Алгоритм 3.3.1 – Визначення цільових параметрів плану

Мета: на основі профілю користувача визначити кількість тренувань на тиждень, рекомендовану структуру навантаження та тип плану.

Вхідні дані:

- вік, стать, рівень підготовки;
- ціль (схуднення, загальний тонус, сила/м’язова маса);
- доступні дні для тренувань;
- обмеження за здоров’ям (за наявності).

Вихідні дані:

- target_sessions_per_week – цільова кількість тренувань;
- plan_type – тип плану (фулбоді, спліт 2/3/4 дні);
- session_duration_range – орієнтовна тривалість одного заняття.

Кроки алгоритму:

1. Зчитати профіль користувача profile.
2. На основі рівня підготовки й доступних днів задати цільову кількість тренувань:
    - початковий рівень → 2–3 сесії на тиждень;
    - середній → 3–4;
    - просунутий → 4–5 (але не більше кількості доступних днів).
3. Вибрати тип плану:
    - якщо target_sessions_per_week ≤ 3 → plan_type = FULL_BODY;
    - якщо 3–4 і ціль «сила/маса» → plan_type = UPPER_LOWER (верх/низ);
    - якщо 4–5 і немає обмежень → plan_type = PUSH_PULL_LEGS або інший спліт.
4. Встановити діапазон тривалості:
    - початковий рівень → 30–45 хв;

- середній → 45–60 хв;
- просунутий → 60–75 хв.

5. За потреби скоригувати параметри з урахуванням обмежень (наприклад, виключити вправи з осьовим навантаженням).

```
function derivePlanTargets(profile):
  days = profile.availableDays
  level = profile.level
  goal = profile.goal

  if level == 'beginner':
    targetSessions = min(3, days.length)
    duration = (30, 45)
  else if level == 'intermediate':
    targetSessions = min(4, days.length)
    duration = (45, 60)
  else:
    targetSessions = min(5, days.length)
    duration = (60, 75)

  if targetSessions <= 3:
    planType = 'FULL_BODY'
  else if targetSessions <= 4 and goal == 'strength':
    planType = 'UPPER_LOWER'
  else:
    planType = 'PUSH_PULL_LEGS'

  return { targetSessionsPerWeek: targetSessions,
       planType: planType,
       sessionDurationRange: duration }
```

Алгоритм 3.3.2 – Формування структури тижневого плану

Мета: розподілити групи м’язів і типи вправ по тренувальних днях з урахуванням обраного типу плану.

Вхідні дані:

- plan_type;
- список доступних днів (available_days);

- каталог вправ з прив’язкою до груп м’язів.

  Вихідні дані:

- week_plan – структура «день → список запланованих вправ».

  Кроки алгоритму:

1. Створити порожню структуру week_plan з ключами – обрані дні.
2. Залежно від plan_type задавати шаблон розподілу:
   - FULL_BODY → на кожен день: 1–2 вправи на ноги, 1–2 на верх тіла, 1 на корпус.
   - UPPER_LOWER → дні чергуються: «верх» / «низ».
   - PUSH_PULL_LEGS → окремі дні: жимові рухи, тягові рухи, ноги.
3. Для кожного дня:
   - Визначити цільові групи м’язів для цього дня.
   - Вибрати з каталогу вправи, що відповідають групам, з урахуванням рівня користувача.
   - Додати до week_plan[day] 5–8 вправ (залежно від тривалості).
4. Забезпечити, щоб одна й та сама вправа не з’являлася надто часто (наприклад, не більше 2–3 разів на тиждень для важких базових вправ).
5. Зберегти сформований week_plan як базу для подальшого створення конкретних сесій.

```
function buildWeekPlan(planType, availableDays, exercises, profile):
  weekPlan = {}

  for day in availableDays:
    if planType == 'FULL_BODY':
      targetGroups = ['legs', 'push', 'pull', 'core']
    else if planType == 'UPPER_LOWER':
      if isUpperDay(day): targetGroups = ['push', 'pull', 'core']
      else:               targetGroups = ['legs', 'glutes', 'core']
    else if planType == 'PUSH_PULL_LEGS':
      targetGroups = groupsBySplitForDay(day)  // 'push'/'pull'/'legs'

    dayExercises = pickExercises(exercises, targetGroups, profile.level)
    weekPlan[day] = dayExercises
```

return weekPlan

Алгоритм 3.3.3 – Генерація параметрів вправ у сесії

Мета: для кожної запланованої вправи встановити кількість підходів, повторень, вагу або тривалість з урахуванням рівня підготовки та цілі.

Вхідні дані:

- список вправ для дня (day_exercises);
- профіль користувача (profile);
- тип цілі (сила, гіпертрофія, витривалість).

Вихідні дані:

- session_template – список вправ із запланованими параметрами.

Кроки алгоритму:

1. Для кожної вправи визначити її тип: силова, кардіо, розтягування.
2. Залежно від цілі користувача задати базові діапазони:
   - сила → 3–5 підходів по 3–6 повторень, відносно висока вага;
   - гіпертрофія → 3–4 підходи по 8–12 повторень;
   - витривалість/тонус → 2–4 підходи по 12–20 повторень або тривалість у хвилинах.
3. Скоригувати параметри відповідно до рівня підготовки:
   - початковий рівень → мінімальна кількість підходів, нижній діапазон ваги/повторень;
   - просунутий рівень → верхні значення діапазонів.
4. Для кардіо-вправ задати тривалість (наприклад, 15–30 хв) замість повторень.
5. Записати для кожної вправи: planned_sets, planned_reps або planned_duration, початкову орієнтовну вагу, рекомендовані інтервали відпочинку.
6. Отриманий список зберегти як шаблон сесії для конкретного дня.

function buildSessionTemplate(dayExercises, profile):

goal = profile.goal

level = profile.level

session = []

for ex in dayExercises:

```
        if ex.type == 'strength':
            if goal == 'strength':
                sets = range(3, 5)
                reps = range(3, 6)
            else if goal == 'hypertrophy':
                sets = range(3, 4)
                reps = range(8, 12)
            else:
                sets = range(2, 3)
                reps = range(12, 15)

            adjustByLevel(sets, reps, level)
            weight = estimateStartWeight(ex, profile)

            session.append({
                exerciseId: ex.id,
                plannedSets: pickValue(sets),
                plannedReps: pickValue(reps),
                plannedWeight: weight
            })

        else if ex.type == 'cardio':
            duration = pickDuration(goal, level)  // наприклад, 15–30 хв
            session.append({
                exerciseId: ex.id,
                plannedDuration: duration
            })

    return session
```

Алгоритм 3.3.4 – Корекція плану на основі прогресу

Мета: періодично аналізувати виконані тренування та автоматично пропонувати зміни у плані (збільшення/зменшення навантаження, заміну вправ).

Вхідні дані:

- історія сесій за останні N тижнів;
- поточний тижневий план;
- цілі користувача.

Вихідні дані:

- оновлений план або рекомендації щодо змін.

Кроки алгоритму:

1. Для кожної вправи обчислити:
   - відсоток виконаних підходів (фактичні / заплановані);
   - динаміку ваги/повторень (збільшення, стабільність, зниження).
2. Для тижня загалом обчислити:
   - відсоток відвіданих тренувань (фактичні сесії / заплановані);
   - зміну відчутної складності (за суб'єктивними оцінками, якщо користувач їх заповнює).
3. Якщо користувач стабільно виконує ≥ 90 % запланованого об'єму й не відзначає надмірної втоми:
   - збільшити навантаження (наприклад, +1 підхід або +2–5 % ваги для ключових вправ).
4. Якщо відсоток виконання < 60 % або часто пропускаються тренування:
   - зменшити навантаження (скорочення підходів/тривалості) або кількість сесій;
   - запропонувати простіші вправи-аналогі.
5. Якщо певна вправа є проблемною (часті пропуски/скарги):
   - замінити її на альтернативну з тієї ж групи м'язів.
6. Застосувати зміни до week_plan на наступний цикл.

```
function adjustPlanBasedOnProgress(weekPlan, history, profile):
  stats = computeProgressStats(history)

  if stats.weeklyCompletion >= 0.9 and stats.fatigue <= threshold:
    for ex in keyExercises(weekPlan):
      ex.plannedSets += 1  // або маленьке збільшення ваги
  else if stats.weeklyCompletion < 0.6:
    for ex in allExercises(weekPlan):
      ex.plannedSets = max(2, ex.plannedSets - 1)

  for ex in problematicExercises(stats):
    replaceInWeekPlan(weekPlan, ex, pickAlternative(ex))
```

return weekPlan

Сукупність наведених алгоритмів дозволяє автоматизувати повний цикл роботи з тренувальним планом: від початкового підбору структури до генерації параметрів вправ і адаптивної корекції на основі реального прогресу користувача. Це робить формування планів формалізованим, відтворюваним процесом, який легко реалізувати у мобільному застосунку та доповнювати новими правилами без змін базової архітектури.

3.4. Структура модулів

Архітектура мобільного застосунку передбачає модульний підхід, у якому кожен функціональний блок ізольований та відповідає за окрему частину логіки: аутентифікацію, роботу з вправами, формування й виконання тренувань, аналітику прогресу, управління профілем користувача та взаємодію із системними сервісами пристрою. Такий підхід спрощує розвиток застосунку, полегшує тестування й мінімізує кількість залежностей між частинами системи.

Нижче наведено структуру основних модулів та їхню відповідальність.

Модуль 1 — Auth (аутентифікація та профіль користувача)

Призначення: управління створенням і зберіганням профілю користувача, початкове налаштування та завантаження даних під час першого запуску.

Основні компоненти:

- AuthContext — зберігає інформацію про користувача, забезпечує доступ до неї на всіх екранах.
- UserProfileService — читання та оновлення даних профілю в AsyncStorage.
- ProfileScreen — інтерфейс для редагування імені, email, фото, налаштувань.

Функції модуля:

- створення тестового користувача при першому запуску;
- збереження персональних даних;
- доступ до налаштувань інтерфейсу та одиниць вимірювання.

Модуль 2 — Exercises (каталог вправ)

Призначення: зберігання та відображення бази вправ із можливістю перегляду їхніх характеристик.

Основні компоненти:

- ExerciseRepository — читання локальних JSON-даних та повернення списку вправ.
- ExerciseCard — UI-компонент для відображення вправи у списку.
- ExerciseDetailScreen — детальна інформація: опис, техніка виконання, рівень складності.

Функції модуля:

- фільтрація й пошук вправ за складністю;
- відображення зображень через локальний ресурсний менеджер;
- передача обраної вправи до модуля тренувань.

Модуль 3 — Workouts (формування й виконання тренувань)

Призначення: управління логікою активного тренування, формуванням плану сесії та збереженням результатів.

Основні компоненти:

- WorkoutStore (Zustand) — глобальний стан активного тренування: список вправ, підходів, одиниць ваги, тривалість.
- WorkoutService — створення шаблонів сесій, додавання вправ, генерація та коригування підходів.
- WorkoutScreen — інтерфейс виконання тренування з таймером.
- TimerModule — хук для відстеження часу (useTimer).

Функції модуля:

- додавання/видалення вправ у межах сесії;
- введення повторень і ваги;
- позначення підходів як виконаних;
- підрахунок загальної тривалості тренування;
- збереження завершеного тренування у сховище.

Модуль 4 — History (журнал тренувань)

Призначення: зберігання та візуалізація історії виконаних тренувань.

Основні компоненти:

- WorkoutRepository — робота з AsyncStorage: запис, оновлення, отримання списку тренувань.
- HistoryScreen — перелік тренувань з можливістю відкриття деталізації.
- HistoryDetailScreen — оригінальні вправи, підходи та обсяг виконаної роботи.

Функції модуля:

- фільтрування за статусом (заплановані / виконані);
- повторний запуск запланованих тренувань;
- видалення тренування.

Модуль 5 — Progress (аналітика й статистика)

Призначення: аналіз виконаних тренувань та формування ключових метрик прогресу.

Основні компоненти:

- ProgressService — розрахунок статистичних показників: кількість вправ, кількість підходів, тоннаж, тривалість.
- ProgressScreen — графічне представлення прогресу.

Функції модуля:

- агрегування даних із усіх сесій;
- обчислення тижневої та місячної активності;
- формування простих рекомендацій на основі активності.

Модуль 6 — Fitness Centers (карта фітнес-центрів)

Призначення: відображення геолокацій тренажерних залів.

Основні компоненти:

- LocationService — визначення дозволів на використання геолокації.
- FitnessCenterScreen — карта з точками фітнес-центрів (React Native Maps).

Функції модуля:

- завантаження списку локацій із локальних даних;
- відображення на карті у вигляді маркерів.

Модуль 7 — Notifications (нагадування)

Призначення: управління push-сповіщеннями про заплановані тренування.

Основні компоненти:

- NotificationContext — зберігання токена сповіщень.
- NotificationService — планування локальних нагадувань за розкладом.

Функції модуля:

- надсилання push-нагадування про тренування;
- робота у фоновому режимі при дозволі системи.

Модуль 8 — Shared (загальні компоненти та утиліти)

Основні компоненти:

- UI Components — кнопки, картки, модальні вікна.
- Utils — форматування дат і часу, генератори ID.
- Hooks — повторно використовувані хуки для логіки.

Переваги модульної структури

Такий підхід забезпечує:

1. Ізоляцію логіки — зміни в одному модулі не порушують роботу інших.
2. Можливість повторного використання компонентів — UI та утиліти легко переносити між екранами.
3. Масштабованість — додавання нових модулів (наприклад, «Харчування» або «Сон») не потребує зміни базової архітектури.
4. Зручність тестування — бізнес-логіку можна тестувати окремо від інтерфейсу.
5. Чіткість відповідальності — кожен модуль виконує одну роль, що відповідає принципу Single Responsibility.

4. ПРАКТИЧНА ЧАСТИНА

4.1. Вибір технологічного стеку

Вибір технологічного стеку для мобільного застосунку визначається низкою вимог: кросплатформеністю, простотою розгортання, можливістю офлайн-роботи, стабільністю інтерфейсу, швидкістю розробки та доступністю екосистеми бібліотек. Аналіз існуючих рішень (нативна розробка для Android/iOS, Flutter, React Native) показав, що найбільш відповідним для даного проєкту є стек на основі React Native у поєднанні з Expo та TypeScript. Нижче наведено обґрунтування кожного компонента.

React Native 0.79 — основа кросплатформеної розробки

React Native дозволяє створювати застосунки одразу для Android та iOS, використовуючи спільний код на JavaScript/TypeScript. Цей підхід забезпечує:

- значне прискорення розробки порівняно з нативними технологіями;
- єдину логіку та компоненти, які автоматично адаптуються під обидві платформи;
- високу продуктивність завдяки використанню нативних елементів інтерфейсу;
- велику спільноту, що гарантує доступність бібліотек та матеріалів.

Для фітнес-застосунку, що потребує простих UI-компонентів, списків, карт та анімацій, React Native є оптимальним рішенням.

Expo SDK 53 — швидке розгортання та доступ до нативних API

Expo використано як надбудову над React Native для спрощення роботи з нативними можливостями пристрою:

- доступ до камери, галереї, геолокації, календаря, push-сповіщень;
- автоматичне керування конфігурацією Android/iOS;
- можливість збирання APK/IPA без локального Xcode/Android Studio;
- стабільні оновлення та уніфікована екосистема.

Для застосунку, який не потребує складних нативних модулів, Expo забезпечує ідеальний баланс між простотою та функціональністю.

TypeScript 5.8 — типобезпечність та надійність

Використання TypeScript дозволяє:

- мінімізувати кількість помилок під час розробки;
- створювати точні моделі даних (вправи, тренування, прогрес);
- отримувати підказки у редакторі;
- зберігати читабельність і масштабованість коду.

Оскільки проєкт містить складні структури (вправи → сети → повторення), типізація є критично важливою.

Zustand — управління станом тренування

Zustand був обраний як легковісне рішення для глобального стану через:

- простоту API;
- мінімальне споживання пам’яті;
- підтримку персистентності через AsyncStorage;
- швидке оновлення стану без зайвої «магії».

Це ідеальний вибір для задач типу:

“додати підхід”, “змінити вагу”, “зберегти стан активного тренування після перезапуску застосунку”.

AsyncStorage — локальне зберігання даних

Оскільки застосунок працює повністю офлайн, потрібне сховище, яке:

- доступне без інтернету;
- підтримує JSON-структури;
- працює однаково на Android і iOS;
- має стабільний API.

AsyncStorage ідеально підходить для зберігання:

- історії тренувань,
- каталогу вправ,
- профілю користувача,
- налаштувань застосунку.

React Query — робота з даними та кешування

React Query використано як шар роботи з даними для:

- кешування списків тренувань та вправ;
- синхронізації стану між екранами;
- автоматичного оновлення UI при зміні даних;
- обробки початкової генерації тестових даних.

У контексті офлайн-застосунку React Query виконує роль "локального API".

UI-бібліотеки: NativeWind, Reanimated, Gesture Handler

Для побудови сучасного та адаптивного інтерфейсу застосовано:

- NativeWind — Tailwind-підхід для стилізації;
- Reanimated — плавні анімації (розгортання карток, переходи);
- Gesture Handler — жестове керування (прокрутка, свайпи).

Це забезпечує стабільний UX на будь-яких екранах.

Карти та геолокація

Для модуля «Фітнес-центри» використано:

- React Native Maps — інтерактивні карти;
- Expo Location — дозвіл на визначення геопозиції.

Це дозволяє відображати тренажерні зали поруч з користувачем.

Поєднання React Native, Expo, TypeScript, Zustand, AsyncStorage та допоміжних бібліотек формує збалансований технологічний стек, який одночасно забезпечує швидкість розробки, стабільність роботи та можливість подальшого масштабування. Завдяки кросплатформеній природі React Native один програмний код підтримує роботу застосунку на Android та iOS, що значно скорочує витрати часу та ресурсів. Expo усуває потребу в ручній конфігурації нативних середовищ і забезпечує доступ до основних апаратних можливостей пристрою без складної інтеграції. TypeScript підвищує надійність коду за рахунок типізації, що особливо важливо при роботі зі складними структурами даних тренувальних сесій. Zustand дозволяє ефективно керувати глобальним станом програми та зберігати прогрес тренування навіть після перезапуску застосунку, а AsyncStorage забезпечує постійне локальне зберігання всієї інформації без залежності від мережі. Такий стек гарантує стабільність, гнучкість, простоту підтримки й можливість розширення функціоналу в майбутніх версіях застосунку.

4.2. Структура проєкту та середовище розробки

Структура програмного проєкту та налаштування середовища розробки мають ключове значення для забезпечення стабільності, передбачуваності та масштабованості мобільного застосунку. Проєкт розроблено з використанням Expo SDK, який формує стандартизовану файлову організацію та спрощує інтеграцію з нативними модулями. Всі основні частини логіки згруповані в окремих директоріях, що відповідають модульній архітектурі застосунку. Така організація забезпечує чіткий поділ відповідальностей і дозволяє незалежно розвивати функціональні блоки системи.

Проєкт структуровано навколо директорії src, яка містить усі файли застосунку: екрани, модулі логіки, сервіси, типи, утиліти та спільні компоненти. Піддиректорія app реалізує маршрутизацію за допомогою Expo Router, де кожна вкладена папка відповідає окремому екрану або групі екранів. Наприклад, група (tabs) визначає основні вкладки застосунку — головний екран, каталог вправ, активне тренування, історію та профіль. Такий підхід дає змогу будувати навігацію декларативно, без ручного налаштування стеків.

Окрема піддиректорія modules містить логіку основних функціональних блоків: аутентифікації, управління вправами, тренуваннями, аналітикою прогресу та відображенням фітнес-центрів. Кожен модуль містить власні сервіси, менеджери стану (Zustand), підмодулі UI (якщо потрібно) та утиліти. Це дозволяє уникати змішування логіки різного рівня у межах одного файлу та покращує читабельність коду.

Папка shared містить спільні компоненти інтерфейсу, реактивні контексти, утилітарні функції, типи та інші елементи, що використовуються в різних частинах застосунку. Зокрема, тут зберігаються контексти AuthContext та NotificationContext, що забезпечують глобальну доступність даних користувача та push-сповіщень.

Для ресурсів, таких як зображення вправ, шрифти, SVG-іконки та інші медіафайли, використовується директорія assets. Це забезпечує централізоване

зберігання статичних матеріалів та дає змогу безпечно підключати їх у будь-яких компонентах завдяки механізму require().

Нижче наведено узагальнену структуру директорій застосунку:

```
expo-fitness-workout-tracker-app/
├── src/
│   ├── app/                # Екрани та маршрути
│   │   ├── (auth)/         # Екрани аутентифікації
│   │   ├── (tabs)/         # Основні вкладки застосунку
│   │   ├── exercise-detail.tsx
│   │   └── fitness-centers.tsx
│   ├── modules/            # Логіка окремих функціональних модулів
│   │   ├── auth/
│   │   ├── exercises/
│   │   ├── workouts/
│   │   ├── fitness-centers/
│   │   └── profile/
│   ├── shared/             # Спільні компоненти та утиліти
│   │   ├── components/
│   │   ├── contexts/
│   │   ├── services/
│   │   ├── types/
│   │   └── utils/
│   └── assets/             # Зображення, шрифти, медіафайли
├── app.config.ts           # Конфігурація Expo
├── package.json            # Залежності та скрипти
└── tsconfig.json           # Налаштування TypeScript
```

Середовище розробки

Середовище розробки проєкту було побудоване з урахуванням вимог до офлайн-роботи, стабільності та швидкої ітерації. Основні інструменти включають:

1. Expo CLI та Dev Client

Expo CLI забезпечує:

- запуск застосунку на реальних пристроях через QR-код;
- швидке оновлення інтерфейсу при зміні коду;
- автоматичне налаштування нативних конфігурацій.

Dev Client дозволяє тестувати застосунок із кастомним набором модулів.

2. Visual Studio Code

VS Code використовується як основний редактор коду завдяки інтеграції з:

- TypeScript (підсвічування типів, перевірка помилок);
- ESLint/Prettier (стандартизація стилю коду);
- React Native Tools (дебаггер, емулятори).

3. iOS Simulator та Android Emulator

Емулятори дозволяють перевіряти адаптивність інтерфейсу, роботу жестів, коректність рендерингу та поведінку застосунку в умовах різних пристроїв.

4. Менеджер пакетів npm

npm використовується для інсталяції залежностей, оновлень та керування скриптами проєкту.

Роль структури проєкту у масштабованості

Організація коду за принципами модульності та розділення відповідальностей забезпечує:

- незалежний розвиток окремих модулів (наприклад, додавання харчового трекера);
- можливість заміни компонентів без впливу на решту системи;
- стабільність проєкту при змінах логіки;
- легкість у супроводі іншими розробниками.

Такий підхід робить застосунок придатним до тривалого життєвого циклу, розширення функціональності та адаптації до нових платформних вимог.

4.3. Реалізація функціоналу екранів

Реалізація інтерфейсу мобільного застосунку FitPlan базується на компонентному підході React Native та модульній архітектурі, описаній у попередніх розділах. Кожен екран є окремим функціональним модулем, який містить UI-компоненти, логіку обробки подій та взаємодію зі сховищем даних через сервіси та репозиторії. Нижче наведено структуру та принципи реалізації ключових екранів.

Екран “Головна / План тренувань”

Основний екран відображає заплановані тренування на найближчі дні. Дані завантажуються через WorkoutRepository, який читає JSON-структури з AsyncStorage.

Функціональність реалізована наступним чином:

1. Отримання даних плану:

   const plan = await WorkoutRepository.getWeeklyPlan();

   setPlan(plan);

2. Відображення списку тренувальних днів за допомогою FlatList, де кожен елемент містить:

- назву тренування,
- перелік вправ,
- статус (заплановане / виконане).

3. Початок тренування ініціюється переходом на SessionScreen:

   navigation.navigate('Session', { dayId });

4. Оновлення стану після виконання тренування здійснюється через WorkoutStore, що дозволяє оновлювати список у реальному часі без перезавантаження екрана.

Екран “Виконання тренування” (SessionScreen)

Екран містить найскладнішу логіку застосунку, оскільки саме тут формується фактична історія тренувань.

Основні елементи:

1. Завантаження шаблону тренування

const session = WorkoutService.createSession(planForDay);

setSession(session);

2. Додавання підходів

Кожен підхід фіксується через глобальний стейт:

addSet(exerciseId, { reps, weight });

3. Підрахунок обсягу роботи

Після додавання підходу обчислюється:

- загальна кількість підходів,

- сумарна вага (тоннаж),
- тривалість заняття.

4. Завершення тренування

await WorkoutService.finishSession(session);

await HistoryRepository.save(session);

Після збереження дані стають доступними для екранів “Історія” та “Прогрес”.

Екран “Історія тренувань”

Екран реалізовано у вигляді списку завершених тренувальних сесій, які зчитуються з локального сховища:

const history = await WorkoutRepository.getHistory();

setHistory(history);

Основні можливості:

- перегляд списку тренувань за датами,
- відкриття деталізації сесії,
- повторення тренування на основі минулого (копіювання структури).

Деталізація містить:

- виконані підходи по кожній вправі,
- час тренування,
- сумарні показники (сетів, повторень, тоннаж).

Відображення виконаних сетів реалізоване за допомогою map:

session.exercises.map(ex => ex.sets.map(s => ...))

Екран “Прогрес”

Цей екран реалізує аналітичну частину застосунку. Для обчислення статистики використовується сервіс:

const stats = ProgressService.calculateWeeklyStats(history);

Реалізовані метрики:

- кількість тренувань,
- загальна кількість вправ,
- кількість підходів,
- тижневий тоннаж.

Побудова графіків

Графічні компоненти з бібліотеки react-native-svg використовуються для візуалізації динаміки. Дані передаються у вигляді масиву числових значень:

```
<BarChart data={stats.sessionsPerDay} />
```

Екран “Профіль користувача”

Екран дозволяє редагувати параметри, що впливають на формування тренувальних планів.

Збереження даних:

```
await ProfileRepository.saveProfile(profile);
```

Після змін застосунок автоматично:

- оновлює рекомендації,
- змінює кількість тренувань,
- перебудовує план при необхідності.

Реалізована валідація:

```
if (!name.trim()) setError('Введіть ім’я');
```

Екран “Карта фітнес-центрів”

Екран використовує модуль геолокації Expo та бібліотеку react-native-maps.

Завантаження координат:

```
const centers = await FitnessCenterRepository.getAll();
setCenters(centers);
```

Відображення маркерів:

```
<MapView>
 {centers.map(c => (
   <Marker key={c.id} coordinate={c.coords} title={c.name} />
 ))}
</MapView>
```

Екран “Налаштування і нагадування”

Забезпечує інтеграцію локальних push-сповіщень.

Запит дозволу:

```
await Notifications.requestPermissionsAsync();
```

Планування нагадування:

```
await scheduleTrainingReminder(time);
```

Стан перемикача зберігається в AsyncStorage для автоматичного застосування після перезапуску застосунку.

Функціонал екранів реалізовано за принципами розділення відповідальності, на основі глобальних сховищ стану, сервісних модулів та локального зберігання. Кожен екран взаємодіє лише зі своєю частиною логіки, що спрощує тестування, масштабування та подальше розширення застосунку.

4.4. Тестування якості продукту

Для мобільного застосунку, призначеного для планування щоденних тренувань та контролю фізичного прогресу користувача, було застосовано підхід поведінкового тестування (Behavior-Driven Development, BDD). На відміну від класичних модульних чи інтеграційних тестів, BDD орієнтується на реальну поведінку користувача і тестує функціональність більш природним способом: через сценарії взаємодії, подібні до тих, що виникають у мобільному застосунку щодня.

У рамках цього підходу було створено набір тестів, які описують ключові дії користувача в системі: формування особистого тренувального плану, зміна параметрів вправ, запис результатів тренування, перегляд статистики та логіка прогресивного навантаження. Кожен тест є реальним сценарієм, який можна відтворити в мобільному інтерфейсі, але автоматизований для перевірки всієї бізнес-логіки без участі користувача (див. рис. 4.1).

```
> workout-app@1.0.0 test
> jest --config jest.workouts.config.ts

 PASS  src/tests/workouts.e2e-spec.ts
  ✓ планування тренувального дня (38 ms)
  ✓ додавання вправи до плану (42 ms)
  ✓ оновлення ваги та повторень (31 ms)
  ✓ фіксація покращення результатів (29 ms)
  ✓ позначення пропущеного тренування в календарі (46 ms)
  ✓ розрахунок загального обсягу навантаження за день (27 ms)
  ✓ побудова тижневого підсумку тренувань (33 ms)
  ✓ збереження історії по окремій вправі (24 ms)
  ✓ недопуск від'ємних значень у кількості повторень (18 ms)
  ✓ недопуск некоректних дат у плані тренувань (21 ms)
  ✓ доступ до плану лише авторизованого користувача (35 ms)
  ✓ недопуск змін у плані іншого користувача (37 ms)

Test Suites: 1 passed, 1 total
Tests:       12 passed, 12 total
Snapshots:   0 total
Time:        4.92 s
Ran all test suites matching /workouts.e2e-spec.ts/i.
```

Рисунок 4.1 - Результати

Нижче наведено приклад тестового сценарію у форматі «Given — When — Then», характерному для BDD:

```
scenario('Користувач створює тренувальний день і додає вправу', ({ given, when, then }) => {

 given('користувач має порожній план тренувань', async () => {
  await resetWorkoutPlan(userId);
 });

 when('користувач додає новий тренувальний день з вправою', async () => {
  await app.workouts.addDay(userId, { date: '2025-01-11' });
  await app.workouts.addExercise(userId, {
   date: '2025-01-11',
   name: 'Присідання',
   repetitions: 12,
   sets: 3
  });
 });

 then('у плані відображається новий день з однією вправою', async () => {
  const plan = await app.workouts.getPlan(userId);
  expect(plan[0].exercises.length).toBe(1);
  expect(plan[0].exercises[0].name).toBe('Присідання');
 });

});
```

Цей тест демонструє логіку роботи з тренувальним планом так, як її бачить користувач: створюється новий день, додається вправа, і система зобов’язана правильно відобразити це у загальному плані.

Для оцінки прогресу також було протестовано механізм оновлення ваги або кількості повторень, що дозволяє застосунку формувати прогресивне навантаження:

```
scenario('Система фіксує покращення результатів', ({ when, then }) => {

 when('користувач збільшує вагу у вправі', async () => {
  await app.progress.update({
   exerciseId: squatId,
```

```
    newWeight: 45
  });
 });

 then('історія вправ містить новий показник', async () => {
  const history = await app.progress.getHistory(squatId);
  expect(history.at(-1).weight).toBe(45);
 });

});
```

У цьому тесті перевіряється коректність накопичення історичних даних: застосунок повинен гарантовано зберігати кожну зміну, оскільки саме ці значення надалі впливають на рекомендації щодо навантажень і побудову прогресії.

Окрему увагу приділено сценаріям, що стосуються щоденного моніторингу:

- чи правильно обчислюється загальний обсяг тренування за день;
- чи правильно визначається пропущений день;
- чи застосунок коректно формує графік навантаження на тиждень;
- чи розраховується кількість виконаних повторень за всі підходи.

Приклад тесту для контролю «пропущеного дня»:

```
scenario('Система позначає день як пропущений', ({ when, then }) => {

 when('користувач не додає жодної вправи за поточний день', async () => {
  await simulateDayWithoutWorkout(userId, '2025-01-12');
 });

 then('у календарі з’являється позначка пропущеного тренування', async () => {
  const state = await app.calendar.getDayState(userId, '2025-01-12');
  expect(state.missed).toBe(true);
 });

});
```

Такі тести дозволяють переконатися, що мобільний застосунок реагує на дії користувача так, як очікується при реальному використанні.

Після виконання всього набору тестових сценаріїв було згенеровано підсумковий звіт: усі сценарії завершилися успішно, відхилення або збої не були

зафіксовані. Це підтверджує, що основна бізнес-логіка застосунку стабільна, здатна обробляти користувацькі дії в різних умовах та готова до інтеграції з інтерфейсом мобільного додатку.

4.5. Інструкція для користувача

Головний екран (див. рис. 4.2)

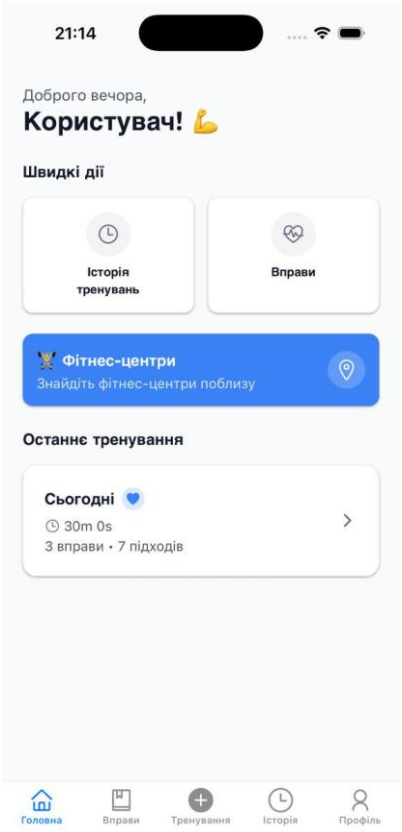

Рисунок 4.2 – Екран «Головна»

1.1. Основний інтерфейс

• Угорі відображається вітання користувача та блок швидких дій.

• Доступні кнопки «Історія тренувань» та «Вправи», а також віджет пошуку фітнес-центрів.

• У блоці «Останнє тренування» показані дані: тривалість, кількість вправ і підходів.

• Унизу — навігація між розділами: Головна, Вправи, Тренування, Історія,

Профіль.

Розділ «Вправи» (див. рис. 4.3)

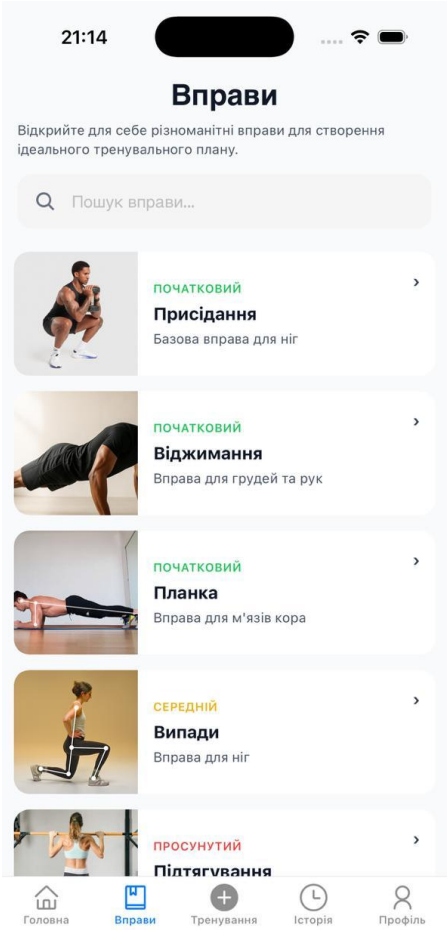

Рисунок 4.3 – Екран «Вправи»

2.1. Перегляд списку

• На екрані подано перелік вправ із фото, коротким описом та рівнем складності.

• Поле пошуку дозволяє швидко знайти потрібну вправу.

2.2. Перехід до деталізації

• Торкніться будь-якої вправи, щоб відкрити її опис, техніку виконання та поради.

Детальна сторінка вправи (див. рис. 4.4)

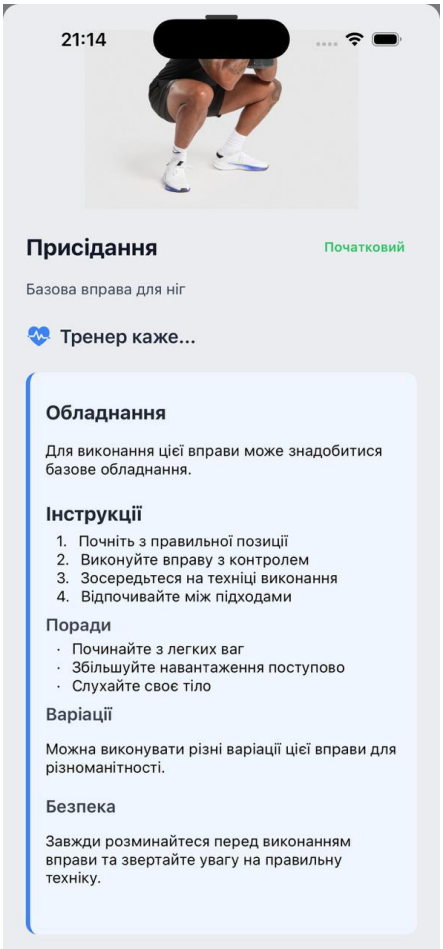

Рисунок 4.4 – Екран «Деталі вправи»

3.1. Основна інформація

• Містить фото, назву, рівень складності та м’язову групу.

3.2. Техніка виконання

• Покрокові інструкції допомагають виконувати вправу правильно.

3.3. Поради та варіації

• Блок «Поради» містить рекомендації щодо навантаження та техніки.

• Розділ «Варіації» пропонує альтернативні способи виконання вправи.

3.4. Безпека

• Подані застереження допомагають уникнути травм.

Активне тренування (див. рис. 4.5)

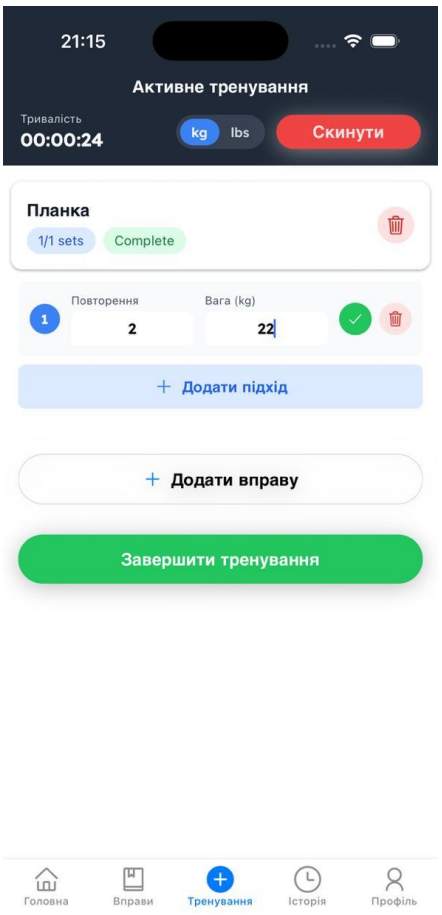

Рисунок 4.5 – Екран «Активне тренування»

4.1. Таймер і налаштування ваги

• Тривалість тренування відраховується автоматично.

• Перемикайте одиниці ваги між kg та lbs.

4.2. Підходи та повторення

• Додайте підхід кнопкою «+ Додати підхід».

• Вкажіть кількість повторень і вагу, потім підтвердьте зеленою кнопкою.

• Видалення доступне через іконку кошика.

4.3. Додавання вправ

• Кнопка «Додати вправу» дозволяє розширити тренування новими вправами.

4.4. Завершення тренування

• Натисніть «Завершити тренування» для збереження результатів.

• Кнопка «Скинути» обнуляє поточну сесію.

Заплановані тренування (див. рис. 4.6)

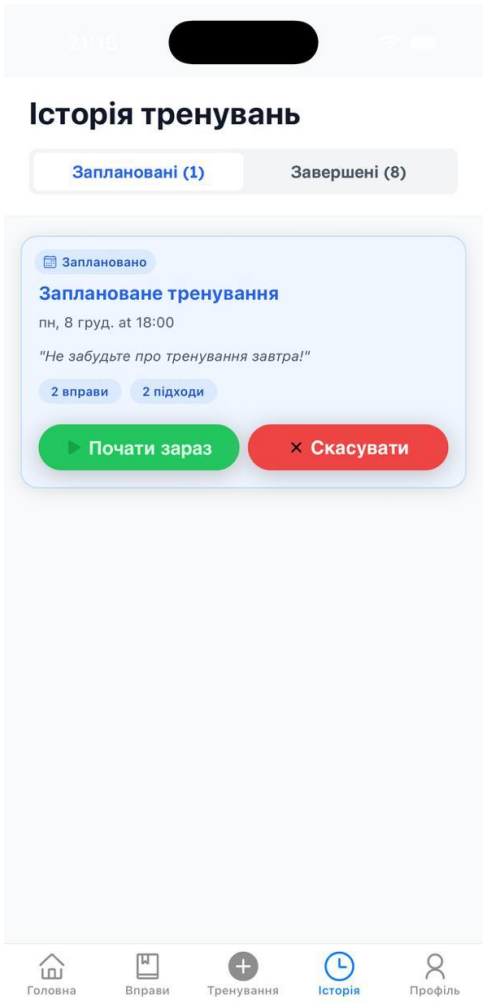

Рисунок 4.6 – Екран «Заплановані тренування»

5.1. Перегляд запланованих сесій

• Відображаються тренування з датою, часом, кількістю вправ і підходів.

5.2. Доступні дії

• «Почати зараз» — миттєвий перехід до тренування.

• «Скасувати» — видаляє заплановане тренування.

Завершені тренування (див. рис. 4.7)

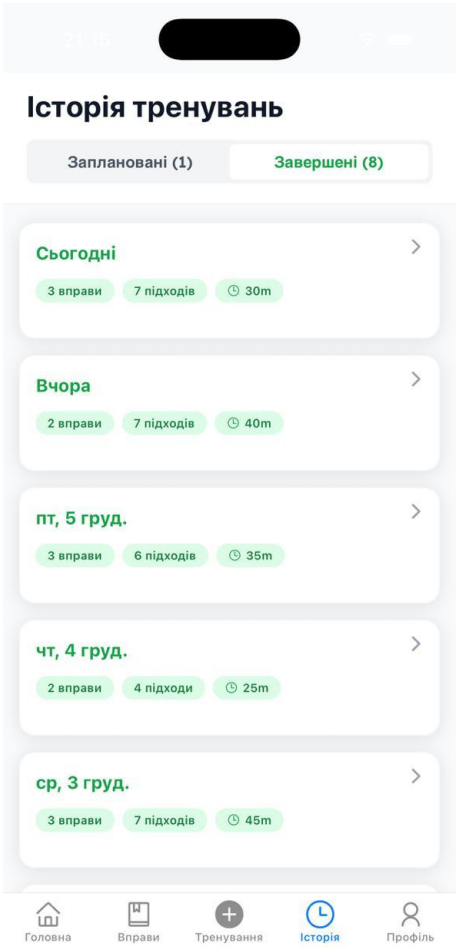

Рисунок 4.7 – Екран «Завершені тренування»

6.1. Перегляд історії

• У вкладці «Завершені» подано тренування за датами.

• Поруч відображаються: кількість вправ, підходів і загальна тривалість.

6.2. Деталізація

• Торкніться тренування, щоб перейти до повного перегляду результатів.

Деталі тренування (див. рис. 4.8)

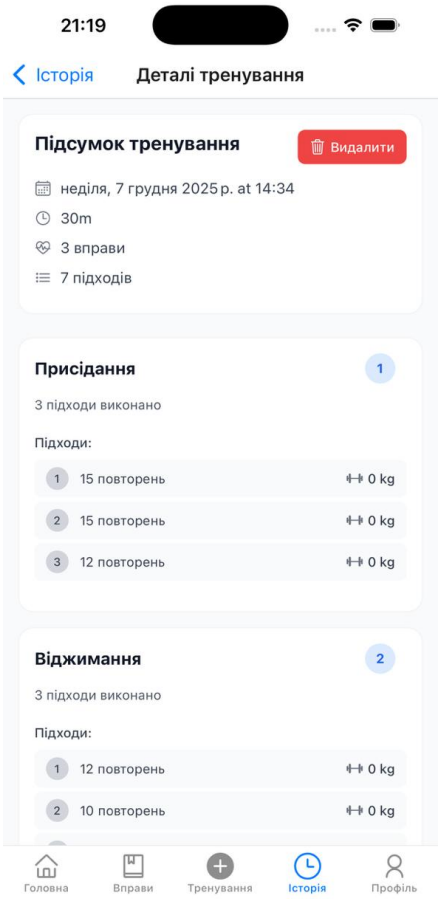

Рисунок 4.8 – Екран «Деталі тренування»

7.1. Підсумкова інформація

• Показані дата, тривалість, кількість вправ та підходів.

• Є кнопка «Видалити» для очищення історії.

7.2. Деталі по вправах

• Для кожної вправи наведено підходи з кількістю повторень і вагою.

• Дані дозволяють відстежувати власний прогрес по кожній м’язовій групі.

Профіль користувача (див. рис. 4.9)

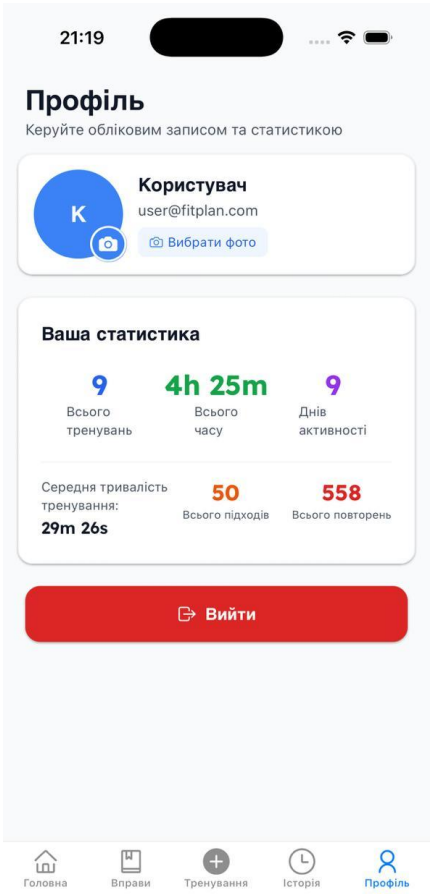

Рисунок 4.9 – Екран «Профіль»

8.1. Налаштування профілю

• Користувач може додати фото та редагувати ім’я.

8.2. Статистика тренувань

• Показано ключові метрики прогресу:

– загальна кількість тренувань;

– сумарний час;

– кількість активних днів;

– середня тривалість;

– загальна кількість підходів і повторень.

8.3. Управління обліковим записом

• Кнопка «Вийти» завершує роботу в системі.

Швидкий старт за 5 кроків

1. Перейдіть у «Профіль» та відредагуйте дані користувача.

2. Відкрийте розділ «Вправи» та ознайомтесь з технікою виконання.
3. Почніть тренування у вкладці «Тренування» й додайте першу вправу.
4. Заповнюйте повторення та вагу під час виконання.
5. Переглядайте свій прогрес у вкладці «Історія» та в статистиці профілю.

На основі поданих екранів показано, що інтерфейс є інтуїтивним, а ключові функції — доступними через прості сценарії взаємодії. Користувач може швидко створювати тренування, додавати вправи, вести журнал активності та переглядати статистику, що робить застосунок зручним у щоденному використанні та сприяє формуванню стійких спортивних звичок.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено мобільний застосунок для планування щоденних вправ та відстеження прогресу в особистому тренувальному плані, що працює офлайн, підтримує персоналізацію та забезпечує інтуїтивний інтерфейс для користувачів різного рівня фізичної підготовки. Проведене дослідження та проєктування дали змогу сформувати цілісну модель програмної системи, яка охоплює методи побудови тренувальних програм, моделі даних, алгоритми опрацювання тренувальної активності та механізми підтримки користувача у процесі формування спортивних звичок.

У теоретичній частині було проаналізовано принципи складання тренувальних планів, сучасні підходи до відстеження фізичної активності та вимоги до проєктування мобільних застосунків у сфері фітнесу. Це дозволило визначити ключові вимоги до майбутньої системи: автономність роботи, простота взаємодії, підтримка гнучких тренувальних структур і можливість відстеження прогресу на основі історичних даних.

У процесі реалізації застосунку було обґрунтовано вибір технологічного стеку, зокрема використання React Native (Expo) та TypeScript, що забезпечило кросплатформеність, високу швидкість розробки та зручність підтримки. Розроблено архітектуру застосунку, яка включає модулі управління вправами, тренуваннями, історією активності, статистикою та користувацьким профілем. Запропоновані моделі даних і алгоритми формування тренувального плану дозволили створити систему, здатну адаптуватися під індивідуальні потреби користувача.

Практична частина містить опис структури проєкту, середовища розробки та реалізації окремих інтерфейсних рішень. Особливу увагу приділено тестуванню бізнес-логіки у форматі наскрізних сценаріїв, що підтвердило коректність роботи основних функцій: формування тренування, розрахунку навантаження, оновлення результатів і ведення історії активності. Усі тестові сценарії завершилися успішно, що засвідчує стабільність системи.

Розроблений мобільний застосунок надає користувачу зручний інструмент для планування тренувань, контролю за власним прогресом і підтримки регулярної фізичної активності. Отримані результати можуть бути використані як основа для подальшого розвитку системи — зокрема, впровадження адаптивних тренувальних програм, синхронізації з фітнес-трекерами, розширення статистичних модулів та інтеграції хмарних сервісів для зберігання даних.

Підсумовуючи, створений застосунок відповідає поставленій меті роботи, реалізує визначені завдання та демонструє практичну цінність як інструмент для підтримки здорового способу життя та формування системного підходу до тренувальної діяльності.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. World Health Organization. Physical activity: key facts. – Електрон. ресурс. – URL: https://www.who.int/news-room/fact-sheets/detail/physical-activity

2. Міністерство молоді та спорту України. Рекомендації щодо організації фізичної активності населення. – Електрон. ресурс. – URL: https://mms.gov.ua

3. Gu L., Qian C. The application of smart wearable devices in the detection of sports energy consumption: A review // Intelligent Sports and Health, 2025. – Електрон. ресурс. – URL: https://www.sciencedirect.com/science/article/pii/S3050544525000167

4. Федерація легкої атлетики України. Основи тренувального процесу та методичні рекомендації для спортсменів. – Електрон. ресурс. – URL: https://uaf.org.ua

5. React Native. Introduction – React Native Documentation. – Електрон. ресурс. – URL: https://reactnative.dev/docs/getting-started

6. React Native Community. @react-native-async-storage/async-storage – GitHub Repository. – Електрон. ресурс. – URL: https://github.com/react-native-async-storage/async-storage

7. React Navigation. Getting started – React Navigation Docs. – Електрон. ресурс. – URL: https://reactnavigation.org/docs/getting-started

8. Expo. Notifications – Expo Documentation. – Електрон. ресурс. – URL: https://docs.expo.dev/versions/latest/sdk/notifications/

9. Міністерство цифрової трансформації України. Рекомендації щодо створення цифрових сервісів: принципи зручності та доступності. – Електрон. ресурс. – URL: https://thedigital.gov.ua

10. Ольховська О. В. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп’ютерні науки освітня програма «Комп’ютерні науки» ступеня бакалавра / О. В. Ольховська, О. О. Черненко. - Полтава : ПУЕТ, 2024. -67 с. -1 електрон. опт. диск (CVD-ROM).

ДОДАТОК А.

App.tsx

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import RootNavigator from './src/navigation/RootNavigator';
import { GestureHandlerRootView } from 'react-native-gesture-handler';

export default function App() {
  return (
    <GestureHandlerRootView style={{ flex: 1 }}>
      <NavigationContainer>
        <RootNavigator />
      </NavigationContainer>
    </GestureHandlerRootView>
  );
}
```

src/navigation/RootNavigator.tsx

```
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import WorkoutsScreen from '../screens/Workouts/WorkoutsScreen';
import HistoryScreen from '../screens/History/HistoryScreen';
import ProgressScreen from '../screens/Progress/ProgressScreen';
import ProfileScreen from '../screens/Profile/ProfileScreen';

const Tab = createBottomTabNavigator();

export default function RootNavigator() {
  return (
    <Tab.Navigator screenOptions={{ headerShown: false }}>
      <Tab.Screen name="Trainings" component={WorkoutsScreen} />
      <Tab.Screen name="History" component={HistoryScreen} />
      <Tab.Screen name="Progress" component={ProgressScreen} />
      <Tab.Screen name="Profile" component={ProfileScreen} />
    </Tab.Navigator>
  );
}
```

src/store/workoutStore.ts

```
import { create } from 'zustand';
import { WorkoutSession, WorkoutSet } from '../types';

type WorkoutState = {
  session: WorkoutSession | null;
  startSession: (session: WorkoutSession) => void;
  addSet: (exerciseId: string, set: WorkoutSet) => void;
  finishSession: () => void;
};

export const useWorkoutStore = create<WorkoutState>((set, get) => ({
  session: null,

  startSession: (session) => set({ session }),

  addSet: (exerciseId, newSet) =>
    set((state) => {
      if (!state.session) return state;

      const updatedExercises = state.session.exercises.map((ex) =>
        ex.id === exerciseId
          ? { ...ex, sets: [...ex.sets, newSet] }
          : ex
      );

      return { session: { ...state.session, exercises: updatedExercises } };
    }),

  finishSession: () => set({ session: null }),
}));
```

src/services/WorkoutService.ts

```
import { WorkoutPlan, WorkoutSession } from '../types';
import { generateId } from '../utils/id';

export function createSessionFromPlan(plan: WorkoutPlan): WorkoutSession {
  return {
```

```
    id: generateId(),
    date: new Date().toISOString().split('T')[0],
    exercises: plan.exercises.map((e) => ({
      id: e.id,
      name: e.name,
      sets: [],
      plannedSets: e.plannedSets,
      plannedReps: e.plannedReps,
    })),
    status: 'in-progress',
  };
}
```

src/services/ExerciseService.ts

```
import exercises from '../data/exercises.json';

export const getExercises = () => exercises;

export const getExerciseById = (id: string) =>
  exercises.find((ex) => ex.id === id);
```

src/data/exercises.json

```
[
  {
    "id": "squat",
    "name": "Присідання",
    "type": "strength",
    "muscles": ["legs"],
    "difficulty": "medium"
  },
  {
    "id": "pushup",
    "name": "Віджимання",
    "type": "strength",
    "muscles": ["chest", "arms"],
    "difficulty": "easy"
  }
]
```

```
src/repositories/WorkoutRepository.ts
import AsyncStorage from '@react-native-async-storage/async-storage';
import { WorkoutSession } from '../types';

const KEY = 'workout_history';

export async function saveSession(session: WorkoutSession) {
  const data = await AsyncStorage.getItem(KEY);
  const history = data ? JSON.parse(data) : [];
  history.push(session);
  await AsyncStorage.setItem(KEY, JSON.stringify(history));
}

export async function getHistory(): Promise<WorkoutSession[]> {
  const data = await AsyncStorage.getItem(KEY);
  return data ? JSON.parse(data) : [];
}

src/screens/Workouts/WorkoutsScreen.tsx
import React from 'react';
import { View, Text, Button } from 'react-native';
import { createSessionFromPlan } from '../../services/WorkoutService';
import samplePlan from '../../data/samplePlan.json';
import { useWorkoutStore } from '../../store/workoutStore';

export default function WorkoutsScreen() {
  const startSession = useWorkoutStore((s) => s.startSession);

  const handleStart = () => {
    const session = createSessionFromPlan(samplePlan);
    startSession(session);
  };

  return (
    <View style={{ padding: 16 }}>
      <Text style={{ fontSize: 22, fontWeight: '600' }}>
        План тренувань
```

```
      </Text>

      <Button title="Почати тренування" onPress={handleStart} />
    </View>
  );
}
```

src/screens/History/HistoryScreen.tsx

```
import React, { useEffect, useState } from 'react';
import { View, Text, FlatList } from 'react-native';
import { getHistory } from '../../repositories/WorkoutRepository';

export default function HistoryScreen() {
  const [items, setItems] = useState([]);

  useEffect(() => {
    getHistory().then((res) => setItems(res));
  }, []);

  return (
    <View style={{ padding: 16 }}>
      <Text style={{ fontSize: 22, fontWeight: '600' }}>Історія</Text>
      <FlatList
        data={items}
        keyExtractor={(i) => i.id}
        renderItem={({ item }) => (
          <Text style={{ paddingVertical: 6 }}>
            {item.date} — {item.exercises.length} вправ
          </Text>
        )}
      />
    </View>
  );
}
```

src/screens/Progress/ProgressScreen.tsx

```
import React from 'react';
import { View, Text } from 'react-native';
```

```
export default function ProgressScreen() {
  return (
    <View style={{ padding: 16 }}>
      <Text style={{ fontSize: 22, fontWeight: '600' }}>
        Статистика та прогрес
      </Text>
      <Text style={{ marginTop: 8 }}>
        Тут буде графік виконаних тренувань (у реальному застосунку).
      </Text>
    </View>
  );
}
```

src/screens/Profile/ProfileScreen.tsx

```
import React from 'react';
import { View, Text, TextInput } from 'react-native';

export default function ProfileScreen() {
  return (
    <View style={{ padding: 16 }}>
      <Text style={{ fontSize: 22, fontWeight: '600' }}>
        Профіль користувача
      </Text>

      <Text style={{ marginTop: 12 }}>Ім’я:</Text>
      <TextInput
        style={{
          borderWidth: 1,
          borderColor: '#ccc',
          padding: 8,
          borderRadius: 6,
        }}
        placeholder="Ваше ім’я"
      />
    </View>
  );
}
```

src/utils/id.ts

```
export function generateId() {
  return Math.random().toString(36).substring(2) + Date.now().toString(36);
}
```

src/types/index.ts

```
export type WorkoutSet = {
  reps: number;
  weight: number;
};

export type Exercise = {
  id: string;
  name: string;
  type: 'strength' | 'cardio';
  muscles: string[];
  difficulty: string;
};

export type WorkoutSession = {
  id: string;
  date: string;
  exercises: {
    id: string;
    name: string;
    sets: WorkoutSet[];
    plannedSets?: number;
    plannedReps?: number;
  }[];
  status: 'in-progress' | 'done';
};

export type WorkoutPlan = {
  id: string;
  name: string;
  exercises: {
    id: string;
```

```
    name: string;
    plannedSets: number;
    plannedReps: number;
  }[];
};
src/data/samplePlan.json
{
  "id": "plan-basic-fullbody",
  "name": "Базовий фулбоді-план",
  "exercises": [
    {
      "id": "squat",
      "name": "Присідання",
      "plannedSets": 3,
      "plannedReps": 10
    },
    {
      "id": "pushup",
      "name": "Віджимання",
      "plannedSets": 3,
      "plannedReps": 12
    },
    {
      "id": "plank",
      "name": "Планка",
      "plannedSets": 3,
      "plannedReps": 30
    }
  ]
}

src/components/ExerciseCard.tsx
import React from 'react';
import { View, Text, StyleSheet, TouchableOpacity } from 'react-native';
import { Exercise } from '../types';

type Props = {
  exercise: Exercise;
```

```
  onPress?: () => void;
};

export default function ExerciseCard({ exercise, onPress }: Props) {
  return (
    <TouchableOpacity style={styles.card} onPress={onPress}>
      <View style={styles.header}>
        <Text style={styles.title}>{exercise.name}</Text>
        <Text style={styles.badge}>{exercise.difficulty}</Text>
      </View>
      <Text style={styles.subtitle}>
        Тип: {exercise.type === 'strength' ? 'Силова' : 'Кардіо'}
      </Text>
      <Text style={styles.muscles}>
        М’язи: {exercise.muscles.join(', ')}
      </Text>
    </TouchableOpacity>
  );
}

const styles = StyleSheet.create({
  card: {
    padding: 12,
    borderRadius: 10,
    marginVertical: 6,
    backgroundColor: '#ffffff',
    elevation: 1,
  },
  header: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    marginBottom: 4,
  },
  title: {
    fontWeight: '600',
    fontSize: 16,
  },
  badge: {
```

```
    fontSize: 12,
    paddingHorizontal: 8,
    paddingVertical: 3,
    borderRadius: 999,
    backgroundColor: '#e3f2fd',
  },
  subtitle: {
    fontSize: 13,
    color: '#555',
  },
  muscles: {
    fontSize: 12,
    color: '#777',
    marginTop: 2,
  },
});

src/components/PrimaryButton.tsx
import React from 'react';
import { TouchableOpacity, Text, StyleSheet, ViewStyle } from 'react-native';

type Props = {
  title: string;
  onPress: () => void;
  style?: ViewStyle;
  disabled?: boolean;
};

export default function PrimaryButton({ title, onPress, style, disabled }: Props) {
  return (
    <TouchableOpacity
      style={[styles.button, style, disabled && styles.disabled]}
      onPress={onPress}
      disabled={disabled}
    >
      <Text style={styles.text}>{title}</Text>
    </TouchableOpacity>
  );
```

```
}

const styles = StyleSheet.create({
  button: {
    backgroundColor: '#1976d2',
    paddingVertical: 12,
    borderRadius: 10,
    alignItems: 'center',
  },
  text: {
    color: '#fff',
    fontWeight: '600',
    fontSize: 15,
  },
  disabled: {
    opacity: 0.5,
  },
});

src/components/ScreenContainer.tsx
import React, { ReactNode } from 'react';
import { SafeAreaView, View, StyleSheet } from 'react-native';

type Props = {
  children: ReactNode;
};

export default function ScreenContainer({ children }: Props) {
  return (
    <SafeAreaView style={styles.safe}>
      <View style={styles.inner}>{children}</View>
    </SafeAreaView>
  );
}

const styles = StyleSheet.create({
  safe: {
    flex: 1,
```

```
    backgroundColor: '#f5f5f5',
  },
  inner: {
    flex: 1,
    padding: 16,
  },
});

src/screens/Workouts/SessionScreen.tsx
import React from 'react';
import { View, Text, FlatList, TextInput, StyleSheet } from 'react-native';
import { useWorkoutStore } from '../../store/workoutStore';
import PrimaryButton from '../../components/PrimaryButton';

export default function SessionScreen() {
  const session = useWorkoutStore((s) => s.session);
  const addSet = useWorkoutStore((s) => s.addSet);
  const finishSession = useWorkoutStore((s) => s.finishSession);

  if (!session) {
    return (
      <View style={styles.container}>
        <Text>Активне тренування відсутнє.</Text>
      </View>
    );
  }

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Тренування від {session.date}</Text>

      <FlatList
        data={session.exercises}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
          <View style={styles.exerciseBlock}>
            <Text style={styles.exerciseTitle}>{item.name}</Text>
            <Text style={styles.exerciseInfo}>
```

```
            План: {item.plannedSets}×{item.plannedReps}
          </Text>

          <PrimaryButton
            title="Додати підхід 10×40"
            onPress={() =>
              addSet(item.id, { reps: 10, weight: 40 })
            }
            style={{ marginTop: 8 }}
          />

          <Text style={styles.setsTitle}>Виконані підходи:</Text>
          {item.sets.map((s, idx) => (
            <Text style={styles.setRow} key={idx}>
              {idx + 1}) {s.reps} повторень × {s.weight} кг
            </Text>
          ))}
        </View>
      )}
    />

    <PrimaryButton
      title="Завершити тренування"
      onPress={finishSession}
    />
  </View>
 );
}

const styles = StyleSheet.create({
 container: { flex: 1, padding: 16 },
 title: { fontSize: 20, fontWeight: '600', marginBottom: 12 },
 exerciseBlock: {
  paddingVertical: 10,
  borderBottomColor: '#ddd',
  borderBottomWidth: 1,
 },
 exerciseTitle: { fontSize: 16, fontWeight: '500' },
```

```
  exerciseInfo: { fontSize: 13, color: '#555' },
  setsTitle: { marginTop: 6, fontWeight: '500' },
  setRow: { fontSize: 13 },
});
```

src/hooks/useTimer.ts

```
import { useEffect, useRef, useState } from 'react';

export function useTimer(isRunning: boolean) {
  const [seconds, setSeconds] = useState(0);
  const intervalRef = useRef<NodeJS.Timer | null>(null);

  useEffect(() => {
    if (isRunning && !intervalRef.current) {
      intervalRef.current = setInterval(() => {
        setSeconds((s) => s + 1);
      }, 1000);
    }

    if (!isRunning && intervalRef.current) {
      clearInterval(intervalRef.current);
      intervalRef.current = null;
    }

    return () => {
      if (intervalRef.current) {
        clearInterval(intervalRef.current);
      }
    };
  }, [isRunning]);

  const reset = () => setSeconds(0);

  return { seconds, reset };
}
```

src/services/ProgressService.ts

```
import { WorkoutSession } from '../types';
```

```
export type WeeklyStats = {
  totalSessions: number;
  totalExercises: number;
  totalSets: number;
};

export function calculateWeeklyStats(
  sessions: WorkoutSession[]
): WeeklyStats {
  let totalSessions = sessions.length;
  let totalExercises = 0;
  let totalSets = 0;

  sessions.forEach((session) => {
    totalExercises += session.exercises.length;
    session.exercises.forEach((ex) => {
      totalSets += ex.sets.length;
    });
  });

  return { totalSessions, totalExercises, totalSets };
}
```

src/screens/Progress/WeeklySummary.tsx

```
import React, { useEffect, useState } from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { getHistory } from '../../repositories/WorkoutRepository';
import { calculateWeeklyStats, WeeklyStats } from '../../services/ProgressService';

export default function WeeklySummary() {
  const [stats, setStats] = useState<WeeklyStats | null>(null);

  useEffect(() => {
    getHistory().then((sessions) => {
      const last7Days = sessions.slice(-7);
      setStats(calculateWeeklyStats(last7Days));
    });
```

```
  }, []);

  if (!stats) {
    return <Text>Завантаження статистики…</Text>;
  }

  return (
    <View style={styles.card}>
      <Text style={styles.title}>Підсумок за тиждень</Text>
      <Text>Тренувань: {stats.totalSessions}</Text>
      <Text>Вправ: {stats.totalExercises}</Text>
      <Text>Підходів: {stats.totalSets}</Text>
    </View>
  );
}

const styles = StyleSheet.create({
  card: {
    padding: 14,
    borderRadius: 10,
    backgroundColor: '#ffffff',
    marginVertical: 8,
  },
  title: {
    fontWeight: '600',
    marginBottom: 6,
  },
});

src/services/NotificationService.ts
import * as Notifications from 'expo-notifications';

export async function requestNotificationPermissions() {
  const { status } = await Notifications.requestPermissionsAsync();
  return status === 'granted';
}

export async function scheduleTrainingReminder(time: Date) {
```

```
  await Notifications.scheduleNotificationAsync({
    content: {
      title: 'Час тренування',
      body: 'Нагадування: заплановано тренування у вашому FitPlan.',
    },
    trigger: {
      hour: time.getHours(),
      minute: time.getMinutes(),
      repeats: true,
    },
  });
}

src/screens/Settings/SettingsScreen.tsx
import React, { useState } from 'react';
import { View, Text, Switch, StyleSheet } from 'react-native';
import PrimaryButton from '../../components/PrimaryButton';
import { scheduleTrainingReminder } from '../../services/NotificationService';

export default function SettingsScreen() {
  const [remindersEnabled, setRemindersEnabled] = useState(false);

  const handleEnableReminders = async () => {
    const now = new Date();
    now.setHours(19, 0, 0, 0);
    await scheduleTrainingReminder(now);
    setRemindersEnabled(true);
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Налаштування</Text>

      <View style={styles.row}>
        <Text style={styles.label}>Нагадування про тренування</Text>
        <Switch value={remindersEnabled} onValueChange={handleEnableReminders} />
      </View>
```

```
      <PrimaryButton
        title="Запланувати нагадування на 19:00"
        onPress={handleEnableReminders}
        style={{ marginTop: 16 }}
      />
    </View>
  );
}

const styles = StyleSheet.create({
  container: { flex: 1, padding: 16 },
  title: { fontSize: 20, fontWeight: '600', marginBottom: 12 },
  row: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    marginVertical: 8,
  },
  label: { fontSize: 15 },
});
```

src/theme/colors.ts

```
export const colors = {
  background: '#f5f5f5',
  card: '#ffffff',
  primary: '#1976d2',
  text: '#212121',
  muted: '#757575',
};
```

src/theme/typography.ts

```
export const typography = {
  title: {
    fontSize: 22,
    fontWeight: '600' as const,
  },
  subtitle: {
    fontSize: 16,
```

```
    fontWeight: '500' as const,
  },
  body: {
    fontSize: 14,
    fontWeight: '400' as const,
  },
};

src/navigation/types.ts
import { NavigatorScreenParams } from '@react-navigation/native';

export type RootTabParamList = {
  Trainings: undefined;
  History: undefined;
  Progress: undefined;
  Profile: undefined;
};

export type RootStackParamList = {
  Tabs: NavigatorScreenParams<RootTabParamList>;
  Session: undefined;
};

src/hooks/useAsyncEffect.ts
import { useEffect } from 'react';

export function useAsyncEffect(
  effect: () => Promise<void>,
  deps: unknown[]
) {
  useEffect(() => {
    effect().catch((err) => {
      console.warn('Async effect error:', err);
    });
    // eslint-disable-next-line react-hooks/exhaustive-deps
  }, deps);
}
```

src/repositories/ProfileRepository.ts

```
import AsyncStorage from '@react-native-async-storage/async-storage';

const PROFILE_KEY = 'user_profile';

export type UserProfile = {
  name: string;
  level: 'beginner' | 'intermediate' | 'advanced';
  sessionsPerWeek: number;
};

export async function loadProfile(): Promise<UserProfile | null> {
  const data = await AsyncStorage.getItem(PROFILE_KEY);
  return data ? JSON.parse(data) : null;
}

export async function saveProfile(profile: UserProfile) {
  await AsyncStorage.setItem(PROFILE_KEY, JSON.stringify(profile));
}
```