НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук

_____ Голуб Б.Л., доц., к.т.н.

(підпис) (ПІБ, вчене звання і ступінь)

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка мобільного додатку для підбору фітнес-програм на основі фізичних параметрів та цілей користувача з використанням React Native

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми

к.т.н. доцент _____ Вайганг Г.О.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

к.т.н., доцент _____ Бородкіна І.Л.
(Науковий ступень та вчене звання) (підпис) (ПІБ)

Виконав _____ Тарнавський Ю.Ф.
(підпис) (ПІБ)

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерних наук

_____ / Голуб Б.Л., доцент, к.т.н /
підпис

"__"_____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студент Тарнавський Юрій Федорович

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Розробка мобільного додатку для підбору фітнес-програм на основі фізичних параметрів та цілей користувача з використанням React Native

Затверджена наказом ректора НУБіП України від 16.12.2024 № 2249 "С"

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань , які потрібно розробити:

Аналіз проблемної області, вибір та обґрунтування засобів для розробки системи, проектування інформаційної системи.

Дата видачі завдання "16" 12 2024р.

Керівник бакалаврської кваліфікаційної роботи

_____ _____ Бородкіна І.Л.
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Завдання прийняв до виконання _____ Тарнавський Ю.Ф.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

1. GUI – Graphical User Interface – графічний інтерфейс користувача.
2. JSON – JavaScript Object Notation – текстовий формат для зберігання структурованих даних.
3. API – Application Programming Interface – інтерфейс прикладного програмування, що забезпечує взаємодію між модулями.
4. REST – Representational State Transfer – архітектурний стиль взаємодії з веб-сервісами.
5. CRUD – Create, Read, Update, Delete – базові операції з інформаційними об’єктами.
6. UX – User Experience – досвід користувача під час взаємодії із системою.
7. UI – User Interface – інтерфейс користувача.
8. ID – Identifier – унікальний ідентифікатор об'єкта в системі.
9. BLL – Business Logic Layer – шар бізнес-логіки в архітектурі програмного забезпечення.
10. DAL – Data Access Layer – шар доступу до даних.
11. VS Code – Visual Studio Code – інтегроване середовище розробки програмного забезпечення.

12. Expo – Інструментарій для швидкої розробки та тестування React Native-застосунків.

13. AsyncStorage – модуль для локального зберігання даних у форматі ключ–значення.

14. React Native – фреймворк для створення кросплатформених мобільних застосунків на JavaScript.

15. Token – Токен авторизації, що використовується для підтвердження прав доступу користувача.

16. Mock-тест – Імітаційне тестування з використанням підставних (mocked) даних.

17. Dark mode – Темна тема візуального оформлення інтерфейсу.

18. Skeleton screen – Анімований елемент, який відображається до повного завантаження вмісту.

19. Swipe – Жест управління, що виконується рухом пальця по екрану.

20. PATCH-запит – HTTP-запит, який дозволяє частково оновлювати ресурс на сервері.

21. Plan – Структура, яка містить персоналізовану програму тренувань користувача.

22. History – Модуль ведення історії виконаних тренувань, збережений у history.json.

23. LocalStorage – Локальне сховище даних на мобільному пристрої.

ВСТУП

Активне впровадження цифрових технологій у сферу охорони здоров’я та фізичної активності відкриває нові можливості для автоматизації процесів, пов’язаних із підтриманням фізичної форми та веденням здорового способу життя. Персоналізований підхід до формування фітнес-програм, що враховує фізіологічні характеристики користувача та його індивідуальні цілі, потребує гнучких і доступних програмних рішень, які здатні обробляти вхідні дані в мобільному середовищі та формувати оптимальні рекомендації в реальному часі. У цьому контексті мобільні застосунки, розроблені з використанням сучасних фреймворків, відіграють ключову роль у забезпеченні ефективної взаємодії користувача з системою та реалізації персоніфікованих алгоритмів тренування [3], [5].

Використання фреймворку **React Native** забезпечує розробку продуктивного мобільного програмного забезпечення, придатного для платформ Android та iOS з єдиною кодовою базою. Такий підхід дозволяє скоротити витрати на розробку, спростити супровід застосунку та забезпечити швидке масштабування функціональності. Застосування об’єктно-орієнтованого моделювання та мови UML сприяє формалізації вимог до системи, структуризації компонентів і полегшує процес валідації програмної логіки [2], [12].

Об’єктом дослідження виступає процес побудови адаптивної системи підбору фітнес-програм на основі фізичних параметрів користувача.

Предметом дослідження є методи формалізації вимог до системи, архітектурного моделювання, алгоритмізації процесів персоналізації та реалізації міжплатформеного програмного забезпечення.

Метою кваліфікаційної роботи є розробка мобільного додатку, який дозволяє формувати персоналізовані фітнес-програми відповідно до параметрів

користувача і заданих тренувальних цілей з використанням фреймворку React Native та принципів програмної інженерії.

Для реалізації поставленої мети необхідно розв’язати такі завдання:

- виконати аналіз предметної області та оцінити вимоги до функціональності фітнес-застосунку;
- розробити UML-модель, що описує структуру та взаємодію програмних компонентів;
- побудувати архітектуру програмного забезпечення із розподілом на логічні підсистеми;
- реалізувати адаптивний інтерфейс користувача для мобільних платформ;
- розробити алгоритми формування тренувальних сесій на основі вхідних параметрів;
- забезпечити зберігання даних про параметри користувача та історію занять;
- провести функціональне тестування системи та оцінити її придатність до впровадження.

Наукова новизна полягає у поєднанні технологій мобільної розробки з адаптивними методами підбору фітнес-навантажень, що базуються на індивідуальних фізичних характеристиках користувача. Запропонований підхід передбачає застосування єдиної архітектурної моделі для забезпечення універсальності, гнучкості та подальшої масштабованості системи.

Практичне значення роботи полягає у створенні інструменту, придатного для самостійного використання користувачами без залучення фахівців з фізичної підготовки, а також для інтеграції в інфраструктуру фітнес-клубів або медичних центрів як компонента персонального супроводу.

Структура роботи складається з чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто загальні аспекти предметної області, проведено аналіз функціональних вимог і сучасних

рішень. У другому розділі представлено процес проєктування інформаційного забезпечення та моделі системи у вигляді UML-діаграм. У третьому розділі розкрито архітектуру, реалізацію інтерфейсу та програмну логіку додатку. Четвертий розділ містить результати тестування, вимоги до апаратного середовища та рекомендації щодо впровадження.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Фітнес-сфера зазнає трансформацій під впливом цифрових технологій, що зумовлено необхідністю забезпечення доступності персоналізованих рекомендацій для широкої аудиторії користувачів. Значна частина населення прагне отримувати індивідуальні тренувальні програми, адаптовані під власні потреби та фізіологічні особливості. Водночас більшість традиційних методик вимагають залучення тренера, що не завжди є зручним або фінансово досяжним для користувача. Це зумовлює зростання попиту на мобільні інтелектуальні системи, які здатні надавати фітнес-рекомендації автоматизовано та в реальному часі.

Предметна область формування фітнес-програм охоплює сукупність процесів, пов’язаних із обліком початкових фізичних параметрів користувача (вік, стать, вага, рівень фізичної активності), визначенням його мети (схуднення, набір маси, підтримка форми тощо), добором відповідних видів фізичного навантаження, моніторингом динаміки виконання та коригуванням плану в залежності від прогресу. Мобільні інформаційні системи в цьому контексті виступають засобами автоматизації процесів збору, обробки й візуалізації персональних даних, а також засобами інтерфейсної взаємодії користувача з алгоритмами вибору тренувань [3].

Аналіз існуючих мобільних фітнес-рішень на ринку демонструє певну типізацію застосунків за функціональним призначенням. У таблиці 1.1 представлено узагальнені характеристики типових класів мобільних додатків, орієнтованих на фітнес-навантаження.

Таблиця 1.1

Класи мобільних фітнес-застосунків за функціональними ознаками

№ з/п	Тип застосунку	Основні функції	Недоліки
1	Статичні тренувальні програми	Набір фіксованих вправ без урахування індивідуальних параметрів	Відсутність персоналізації
2	Онлайн-консультації з тренерами	Відеозв’язок, інструктаж, супровід	Висока вартість, залежність від часу
3	Трекери фізичної активності	Підрахунок кроків, пульсу, спалених калорій	Не формують повноцінну тренувальну програму
4	Адаптивні системи підбору програм	Формування планів на основі параметрів користувача	Обмежена кількість підтримуваних цілей або помилкові рекомендації

Як видно з таблиці, більшість рішень або орієнтовані на окремі аспекти фітнес-діяльності, або не враховують комплексні цілі й фізіологічну різноманітність користувачів. Це створює передумови для розробки нового інструменту, який би поєднував персоналізацію, автоматизовану логіку підбору вправ і зручність мобільного використання.

Узагальнюючи попередню інформацію, предметна область характеризується високим ступенем індивідуалізації вхідних даних, необхідністю адаптації тренувальних сесій до користувацьких цілей, а також потребою в зберіганні й обробці персональної інформації. Розробка мобільного додатку, що реалізує зазначені функції, має ґрунтуватися на формалізованому підході до моделювання процесів (представлено у пункті 1.5 першого розділу роботи) та чітко структурованій логіці побудови рекомендацій.

1.2 Огляд інформаційних джерел та існуючих рішень

З метою обґрунтування створення власного мобільного застосунку доцільно провести огляд найбільш поширених рішень, які вже реалізовано на ринку мобільних фітнес-додатків. Їхній функціональний діапазон, підхід до персоналізації, інтерфейсні рішення та загальна логіка побудови тренувальних програм можуть бути використані як базис для подальшої розробки власної концепції.

Одним із найбільш відомих застосунків є Nike Training Club (інтерфейс якого представлений на рис.1.1), орієнтований на користувачів різного рівня фізичної підготовки. Додаток надає доступ до структурованих програм тренувань, що включають комплекс вправ із відеосупроводом, таймерами, інструкціями та календарним плануванням занять. Програми розробляються сертифікованими тренерами та охоплюють як силові, так і кардіо-навантаження.

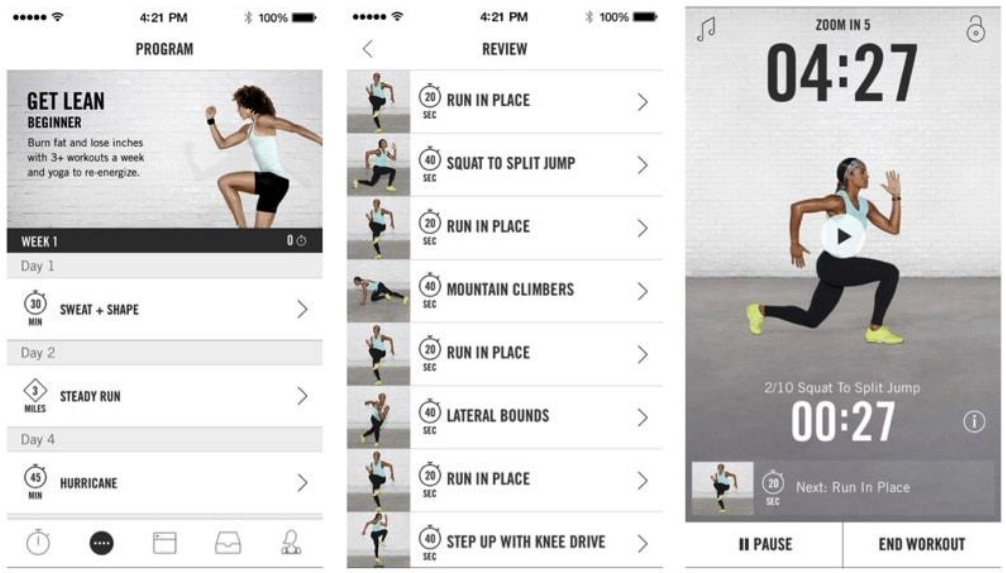

Рисунок 1.1– Інтерфейс мобільного додатку Nike Training Club

Інше популярне рішення — Fitify (рисунок 1.2), що акцентує увагу на гнучкості налаштувань. Користувач може самостійно формувати план тренувань, вибираючи тривалість, тип навантаження та цільові групи м’язів.

Додаток підтримує понад 180 вправ і включає генератор тренувань, що враховує доступність інвентарю.

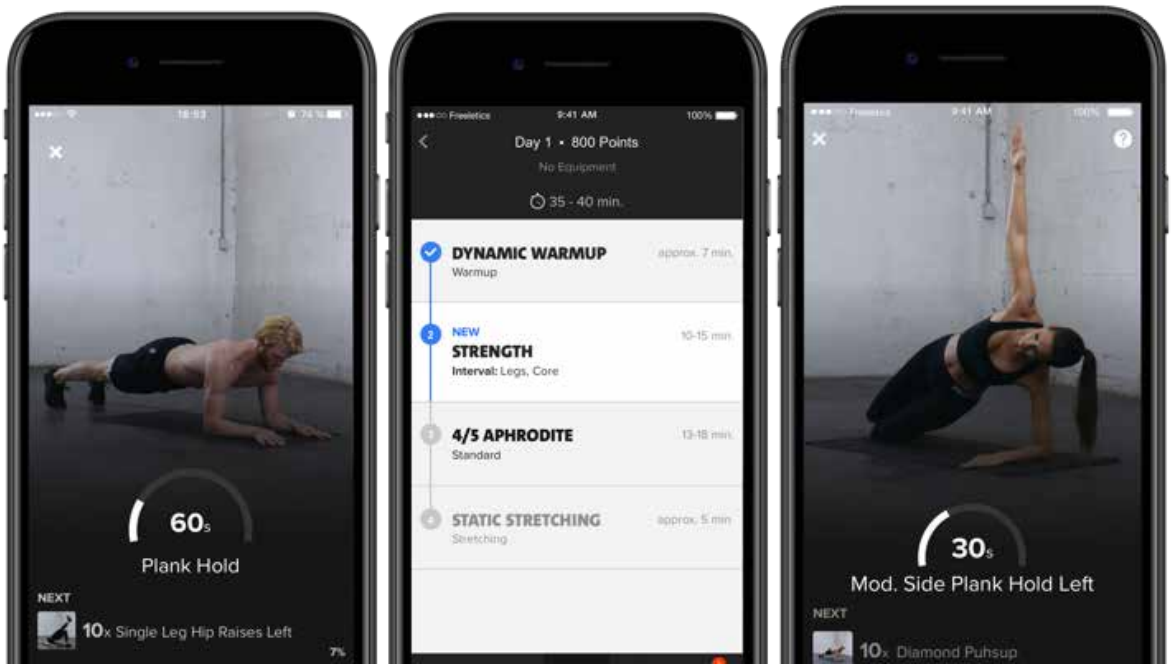

Рисунок 1.2 – Конструктор тренувань та адаптивні налаштування у Fitify

Застосунок Freeletics реалізує високий рівень персоналізації шляхом інтерактивного аналізу результатів користувача. Програма передбачає не лише адаптивні тренування, а й рекомендації щодо відновлення, харчування та підтримки ментального здоров'я. Також реалізовано звуковий супровід занять залежно від інтенсивності, що можна побачити на рис.1.3

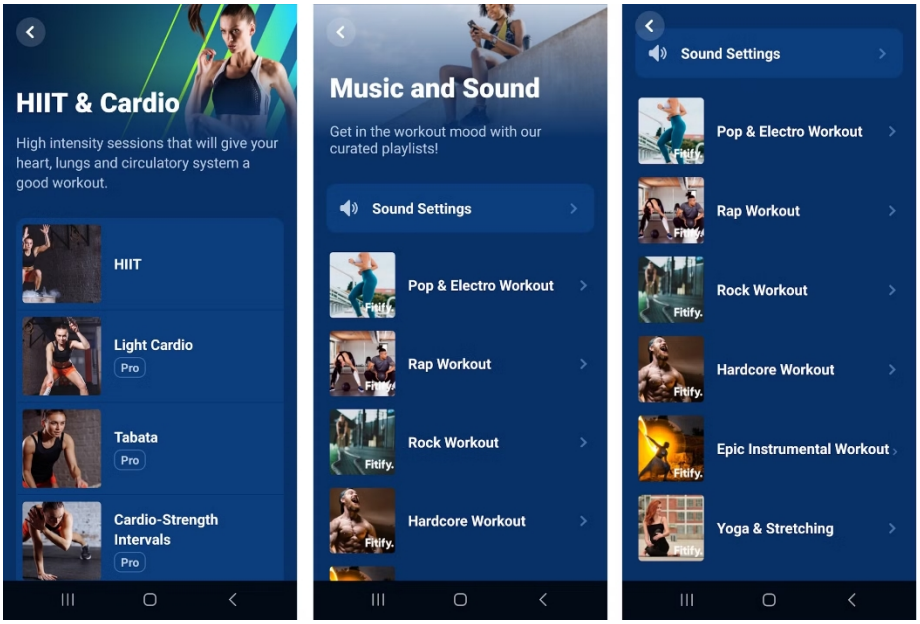

Рисунок 1.3 – Розділ інтервального тренінгу та аудіосупровід у Freeletics

Ще одним прикладом є Adidas Training, у якому користувачеві пропонуються щотижневі або щомісячні фітнес-програми з урахуванням його мети. Система аналізує введені параметри й автоматично формує комплекс вправ, адаптованих до рівня підготовки. Підтримується як тренування з інвентарем, так і без нього. Цікавий функціонал системи представлений на рис.1.4.

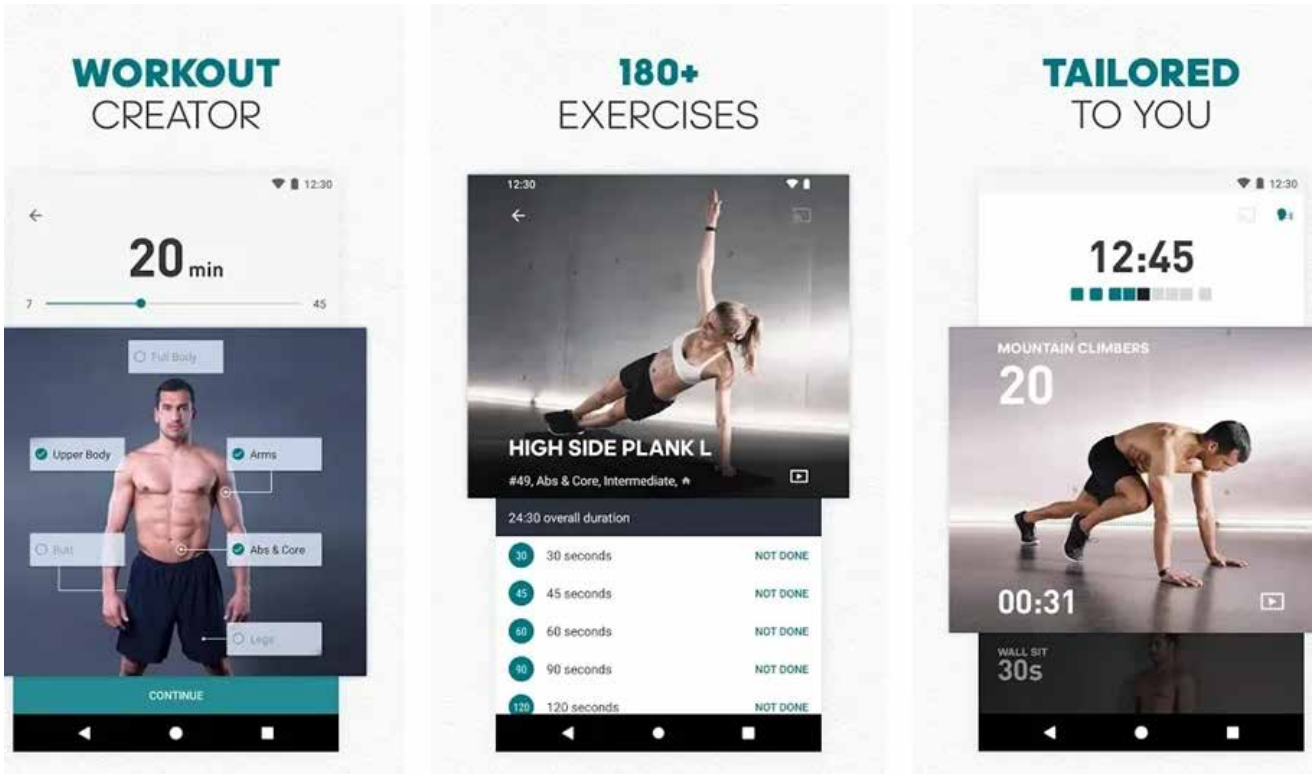

Рисунок 1.4 – Програмування циклів тренувань у додатку Adidas Training

Для повноцінного аналізу існуючих рішень доцільно здійснити порівняння їхніх основних характеристик за ключовими критеріями, що мають безпосередній вплив на якість персоналізованого підбору тренувань, інтерфейсну зручність та загальну функціональність. Доцільно також включити в таблицю 1.2 проєктовану систему, щоб виявити її потенційні переваги або недоліки у контексті реалізації цільового функціоналу.

Таблиця 1.2– Порівняльна характеристика фітнес-застосунків

№ з/п	Назва застосунку	Персоналізація за параметрами	Генерація тренувань	Аудіо-/візуальний супровід	Робота без інтернету	Підтримка інвентарю	Коментар
1	Nike Training Club	Обмежена (профільні цілі)	Програми фіксовані	Відео, голос	Так	Так	Висока якість контенту
2	Fitify	Повна	Автоматизована	Відеоінструкції	Так	Так	Гнучка логіка та генератор вправ
3	Freeletics	Висока	Динамічна	Голосовий супровід	Частково	Обмежено	Розширена адаптація, потребує реєстрації
4	Adidas Training	Середня	Автоматизована	Відео + текст	Так	Так	Орієнтація на простоту та ритм
5	Розроблювана система	Повна (вік, стать, ціль, вага)	Автоматизована	Мінімалістичний аудіо/візуал	Так	Так	Зосереджена на адаптації та простоті

Як видно з таблиці 1.2, запропонована система має потенціал для ефективної реалізації персоналізованого підходу завдяки розширеному набору вхідних параметрів, автоматизованій генерації тренувань та підтримці офлайн-режиму. Ключовою перевагою є фокус на простоті використання при збереженні логіки адаптивного підбору вправ, що дозволяє розширити цільову аудиторію.

На підставі порівняльного аналізу можна дійти висновку, що розроблювана система здатна заповнити функціональну нішу між надто складними комерційними рішеннями та надто примітивними фіксованими програмами, забезпечуючи оптимальний баланс між адаптивністю, інтерфейсною доступністю та ефективністю.

У наступному підрозділі буде розглянуто систему функціональних та нефункціональних вимог до мобільного застосунку, що визначають технічну специфікацію архітектури, алгоритмічної бази й інтерфейсної структури майбутнього програмного продукту.

1.3 Аналіз вимог до програмної системи

Проєктування інформаційної системи, орієнтованої на персоналізований підбір фітнес-програм, передбачає чітке формування комплексу вимог, що охоплюють функціональні, нефункціональні, технічні та безпекові аспекти. Їхня деталізація дозволяє закласти основу для подальшого архітектурного та алгоритмічного проєктування програмного забезпечення, а також забезпечити відповідність очікуванням кінцевого користувача і стандартам якості.

Функціональні вимоги (представлені у таблиці 1.3) визначають, яку саме поведінку має реалізовувати система для досягнення цільової мети — формування персоналізованих тренувальних програм. Вимоги сформульовано з урахуванням логіки сценаріїв використання системи кінцевим користувачем.

Таблиця 1.3

Функціональні вимоги до мобільного застосунку

№ з/п	Назва функції	Опис
1	Введення фізичних параметрів	Користувач вводить вік, стать, вагу, рівень активності та ціль
2	Генерація програми тренувань	Система формує програму автоматично на основі введених даних
3	Відображення опису вправ	Кожна вправа супроводжується назвою, повтореннями, тривалістю
4	Збереження історії тренувань	Користувач має доступ до попередніх занять і виконаних вправ
5	Модифікація програми	Програма коригується залежно від прогресу або зміненої цілі
6	Вибір типу тренування	Користувач може обрати заняття з інвентарем або без нього
7	Офлайн-режим роботи	Додаток зберігає дані локально для роботи без доступу до мережі

Нефункціональні вимоги визначають загальні обмеження та властивості системи, що впливають на зручність, продуктивність та якість користувацького досвіду, детальніше представлені у таблиці 1.4

Таблиця 1.4

Нефункціональні вимоги до системи

№ з/п	Характеристика	Вимога
1	Міжплатформенність	Система повинна працювати на Android та iOS
2	Продуктивність	Формування програми не перевищує 1 секунди
3	Юзабіліті	Інтерфейс має бути інтуїтивно зрозумілим, без перевантажених елементів
4	Відповідність дизайну	Дотримання принципів Material Design
5	Стабільність	Відсутність критичних збоїв під час типових сценаріїв використання
6	Адаптивність	Підтримка роботи на різних розмірах екранів

Технічні вимоги формуються з урахуванням вибраного технологічного стека, який охоплює мову програмування, середовище розробки, структуру даних та зовнішні залежності, що можна побачити у таблиці 1.5

Таблиця 1.5

Технічні вимоги до розробки застосунку

№ з/п	Компонент	Вимога
1	Фреймворк розробки	React Native
2	Мова програмування	JavaScript / TypeScript
3	Середовище виконання	Expo / Android SDK / Xcode
4	Бібліотеки	React Navigation, AsyncStorage, Styled Components
5	Зберігання даних	Локальне (SQLite / JSON)
6	Підтримка API	Опціонально — REST API для синхронізації в хмарі

Безпекові вимоги (таблиця 1.6) формуються з урахуванням потреб захисту персональних даних користувача, а також забезпечення цілісності локального зберігання інформації.

Таблиця 1.6

Вимоги до безпеки мобільного застосунку

№ з/п	Аспект безпеки	Вимога
1	Захист персональних даних	Шифрування локальних даних (AES або Base64)
2	Доступ до даних	Реалізація механізмів блокування доступу при несанкціонованих діях

На основі проведеного аналізу вимог встановлено цілісну картину функціональних можливостей, обмежень і очікувань до майбутнього мобільного додатку. Далі ці вимоги буде формалізовано у вигляді модельної структури системи, що передбачає побудову діаграм об’єктно-орієнтованого моделювання для формалізації архітектури, поведінки та взаємодії компонентів.

1.4 Постановка завдання

На основі проведеного системного аналізу предметної області персоналізованого фітнес-супроводу та детального формування функціональних, нефункціональних, технічних і безпекових вимог сформульовано технічне завдання на створення мобільного застосунку для підбору фітнес-програм. Розроблюване програмне забезпечення має забезпечити повний цикл взаємодії користувача із системою – від введення фізичних параметрів до отримання адаптованих тренувальних планів із можливістю візуального супроводу, історії занять та коригування майбутніх програм залежно від прогресу.

Передбачається створення міжплатформеного мобільного застосунку, реалізованого з використанням фреймворку React Native, що дає змогу використовувати єдину кодову базу для Android та iOS. Взаємодія з локальними даними здійснюватиметься за допомогою інтегрованого сховища типу JSON. Система має бути автономною та не залежати від постійного підключення до

Інтернету, за винятком можливого синхронізаційного модуля, який реалізовуватиметься на окремому етапі.

До вхідних даних, які застосунок повинен обробляти, належать: фізіологічні параметри користувача (вік, стать, вага, рівень активності), ціль тренування (схуднення, підтримка форми, набір м’язової маси), бажана тривалість сесії, обмеження щодо типу навантаження та доступність обладнання. Користувач також має мати змогу переглядати згенеровану програму, виконувати окремі вправи та фіксувати виконання. Вхідні параметри представлені детальніше у таблиці 1.7

Таблиця 1.7

Вхідні параметри для персоналізації фітнес-програми

Категорія параметра	Найменування
Дані користувача	Вік, стать, маса тіла, рівень фізичної активності
Цілі тренування	Схуднення, підтримка форми, набір м’язової маси
Тривалість занять	Мінімальна/максимальна тривалість тренувальної сесії
Наявність інвентарю	Обладнання (ваги, килимок, гумки тощо) або його відсутність
Обмеження	Уникнення навантаження на певні групи м’язів або частини тіла

Результатом роботи системи має бути адаптована тренувальна програма, сформована з урахуванням вхідних параметрів, з поетапним поданням вправ, тривалістю виконання, кількістю повторень, а також функцією відмітки виконаних пунктів. Кінцеві результати функціонування застосунку узагальнено в таблиці 1.8.

Таблиця 1.8

Очікувані результати функціонування мобільного застосунку

Компонент результату	Опис
Персоналізована програма	Список тренувань із вправами, розбитими на сесії за вибраними цілями
Деталізація вправ	Назва, тривалість, тип навантаження, інструкція
Прогрес користувача	Дані про виконані сесії, середній час та частоту занять

Архітектура мобільного застосунку має бути модульною, з розмежуванням на основні компоненти: інтерфейс користувача (UI), логіку формування тренувань, обробку даних користувача, локальне зберігання і модуль візуалізації прогресу. Застосування архітектурного підходу Model–View–Controller (MVC) дозволить ізолювати бізнес-логіку від представлення, забезпечивши масштабованість і підтримку тестування окремих компонентів.

Постановка задачі формалізує ключові етапи реалізації системи, визначає функціональні межі підсистем, встановлює очікувані результати та слугує підґрунтям для розробки концептуальної моделі, що буде виконано в наступному розділі.

1.5 Моделювання предметної області

Розробка інформаційної системи вимагає попереднього формалізованого опису предметної області з метою повного розуміння цільових функцій, взаємозв’язків між сутностями, логіки користувацьких дій та архітектурної побудови. У цьому контексті ефективним інструментом виступає використання нотацій UML, що дозволяють здійснити структурно-функціональне моделювання майбутньої системи, відобразити як статичні, так і динамічні характеристики, та забезпечити узгодженість між логікою, інтерфейсом і даними ще до етапу безпосередньої реалізації.

На першому етапі побудови моделі було створено діаграму прецедентів, яка відображає функціональну взаємодію користувача з мобільним застосунком. У структурі прецедентів виділено базові сценарії: введення фізичних параметрів, вибір цілі тренування, генерація індивідуальної програми, перегляд вправ, фіксація виконаних занять, перегляд прогресу, а також отримання рекомендацій. Кожен з цих сценаріїв представляє окрему функціональність, що реалізується в межах єдиного застосунку, а логічні зв’язки між ними демонструють

послідовність дій, які має виконати користувач для досягнення бажаного результату (рис. 1.5).

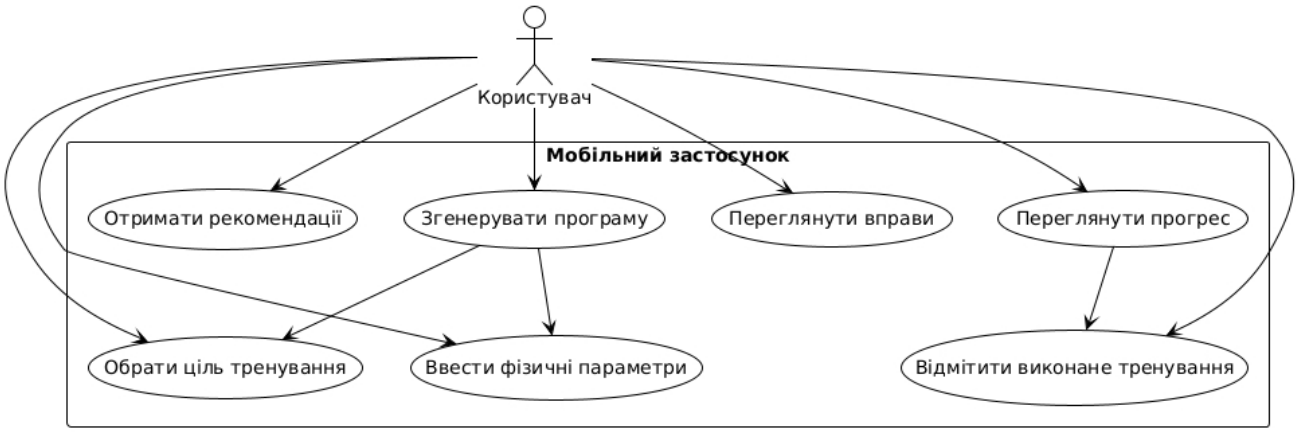

Рисунок 1.5 – Діаграма прецедентів мобільного застосунку

Для деталізації взаємодії між компонентами системи на часовій шкалі побудовано діаграму послідовності, що описує типову сесію роботи з додатком. Користувач здійснює введення параметрів, після чого система звертається до внутрішньої бази даних для пошуку рекомендованих фітнес-програм. Результати повертаються в інтерфейс, де можуть бути відфільтровані відповідно до побажань користувача, а кожен запис має можливість перегляду деталей, збереження або поширення через зовнішні сервіси. Така діаграма дозволяє відобразити порядок обміну повідомленнями між об’єктами системи в рамках одного процесу (рис. 1.6).

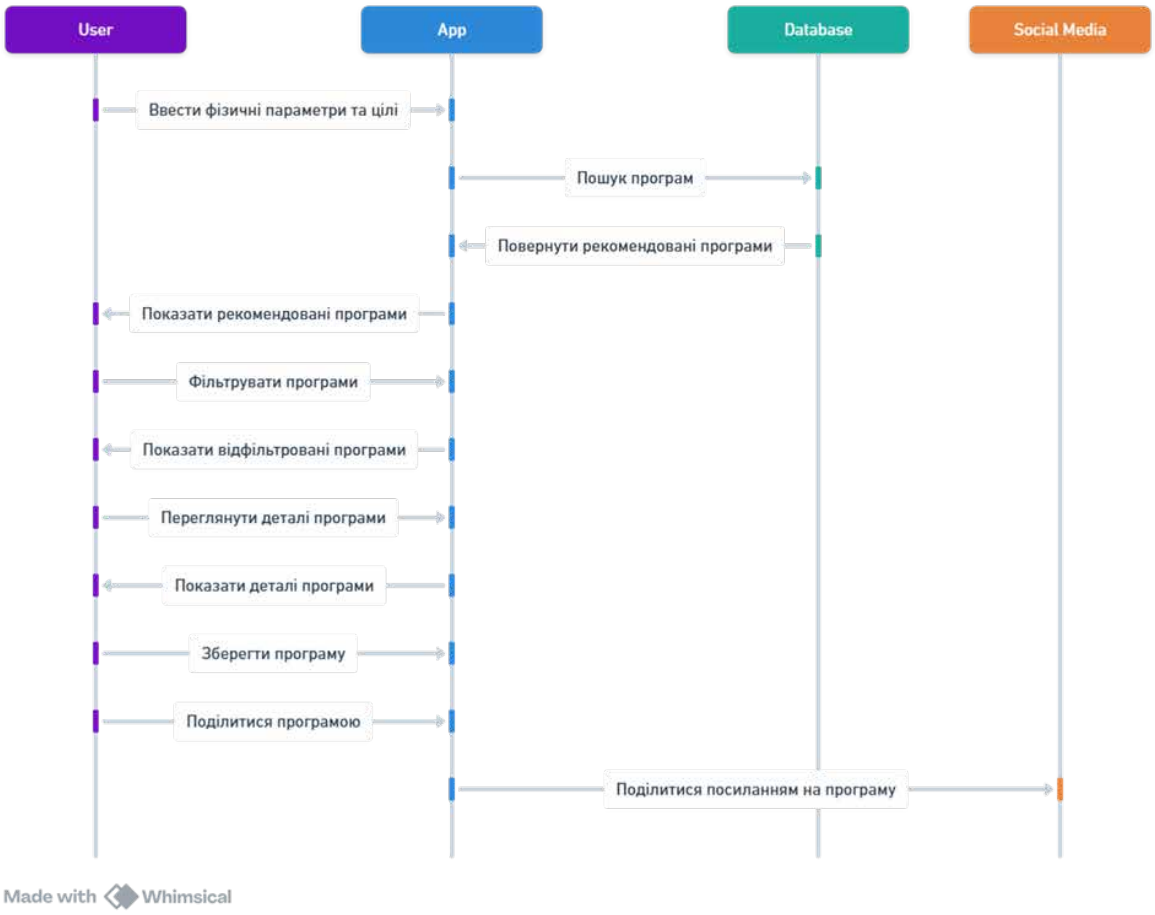

Рисунок 1.6 – Діаграма послідовності взаємодії користувача із застосунком

З метою відображення загальної логіки функціонування системи та контролю бізнес-процесів побудовано діаграму активності, яка демонструє весь життєвий цикл взаємодії користувача із застосунком. Процес починається із запуску програми, введення параметрів і пошуку відповідних тренувальних сесій, після чого користувач може застосувати фільтрацію, переглянути програму, зберегти її або поділитися. Діаграма враховує умови переходу між діями, альтернативні сценарії та точки завершення процесу. Це дозволяє сформувати логічну основу для розробки алгоритмів системи (рис. 1.7).

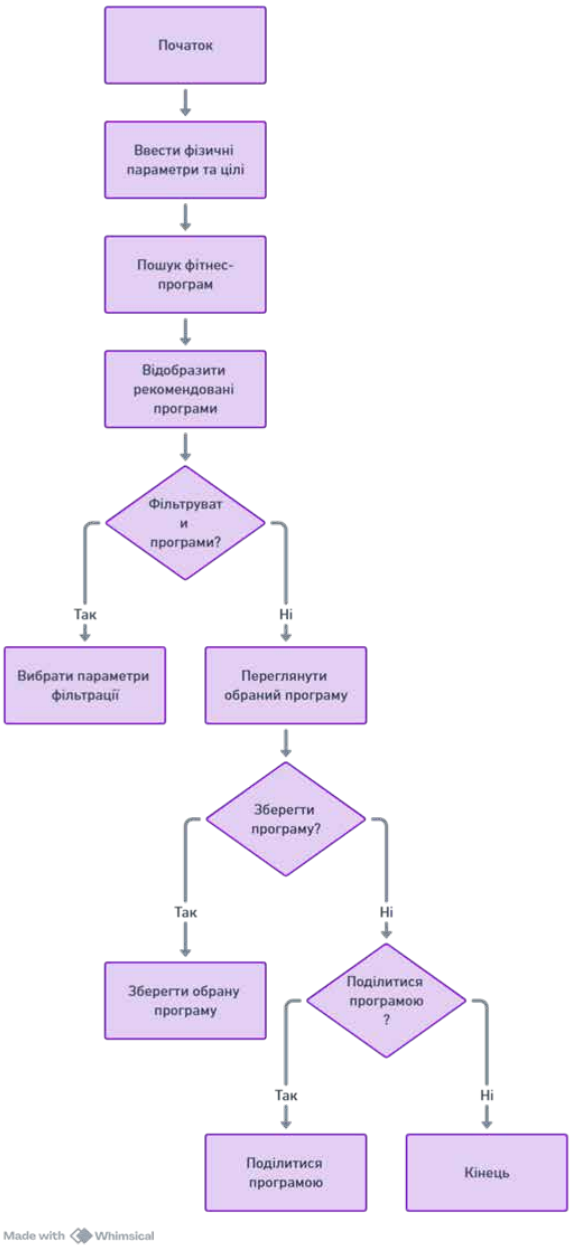

Рисунок 1.7 – Діаграма активності процесу підбору фітнес-програми

Використання засобів UML-моделювання дало змогу систематизувати уявлення про предметну область і зафіксувати цільову логіку поведінки застосунку, що дозволяє перейти до наступного етапу – проєктування інформаційного забезпечення. Створені діаграми становлять формальну основу, яка забезпечить цілісність архітектурних рішень, коректне структурування коду, а також прозорість для подальшого тестування і підтримки.

1.6 Висновки до першого розділу

У межах першого розділу здійснено системний аналіз предметної області автоматизованого підбору фітнес-програм, що дало змогу обґрунтувати доцільність розробки мобільного застосунку з персоналізованим підходом. Встановлено, що більшість наявних рішень або не забезпечують комплексної адаптації до фізіологічних параметрів користувача, або вимагають постійної присутності інструктора, що знижує їхню доступність для широкого кола осіб. Аналіз функціональних можливостей актуальних фітнес-додатків (Nike Training Club, Fitify, Freeletics, Adidas Training) дозволив виявити характерні переваги й обмеження кожного з них, що було враховано при формуванні функціональних вимог до проєктованої системи.

Уточнено вхідні параметри, на основі яких має формуватись персоналізована програма тренувань, зокрема вік, стать, вага, рівень активності, наявність інвентарю та тренувальна мета. Сформульовано повний перелік функціональних, нефункціональних, технічних і безпекових вимог до майбутнього застосунку, що забезпечує основу для подальшого архітектурного моделювання. Особливу увагу приділено реалізації офлайн-режиму, підтримці кросплатформенності та адаптивності інтерфейсу користувача.

Результати аналізу лягли в основу постановки задачі та дозволили розпочати процес формалізованого моделювання системи з використанням UML-нотацій, що буде представлено у другому розділі.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Для забезпечення цілісності зберігання та ефективної обробки даних мобільного застосунку персоналізованого підбору фітнес-програм була сформована логічна модель, яка описує основні сутності, їх атрибути та взаємозв’язки. Логічна модель реалізована у вигляді ER-діаграми (Entity–Relationship), що є базовим інструментом для концептуального проектування структури даних у реляційних системах [19].

У рамках побудови ER-моделі визначено шість ключових сутностей: User, WorkoutPlan, Exercise, PlanExercise, WorkoutHistory та Device, кожна з яких має унікальний первинний ключ (PK) і набір атрибутів, необхідних для реалізації відповідної логіки.

Користувацький профіль у системі представлено через збереження таких параметрів, як вік, стать, маса тіла, рівень фізичної активності та ціль тренувального процесу. Ці дані є ключовими для формування персоналізованих фітнес-програм. Один користувач може бути пов’язаний з кількома тренувальними планами, історією занять та пристроями, що забезпечує реалізацію відповідних зв’язків типу «один-до-багатьох».

Тренувальні плани зберігають структуровану інформацію щодо цілей користувача, тривалості та дати створення програми. Кожен план жорстко асоційований із конкретним профілем через зовнішній ключ, що забезпечує персоніфікацію сценаріїв використання.

Набір доступних фізичних вправ описується у відповідній таблиці, яка включає найменування, тип навантаження, текстовий опис і вимоги до інвентарю. Для реалізації багатоваріантного включення вправ до планів

використано проміжну структуру, яка додатково містить параметри виконання: кількість повторень, підходів і тривалість.

Інформація про фактичне виконання занять фіксується у формі історії, де зазначаються дата проведення сесії, її статус і витрачений час. Такий підхід дає змогу відстежувати прогрес користувача та адаптувати подальші рекомендації відповідно до отриманих результатів.

Дані про пристрої, що використовуються для взаємодії із застосунком, включають тип обладнання, операційну систему та часову мітку останнього входу. Це дозволяє не лише здійснювати аудит доступу, а й підвищити загальну безпеку системи через виявлення потенційно аномальної активності.

Загальну структуру логічної моделі даних представлено на рисунку 2.1.

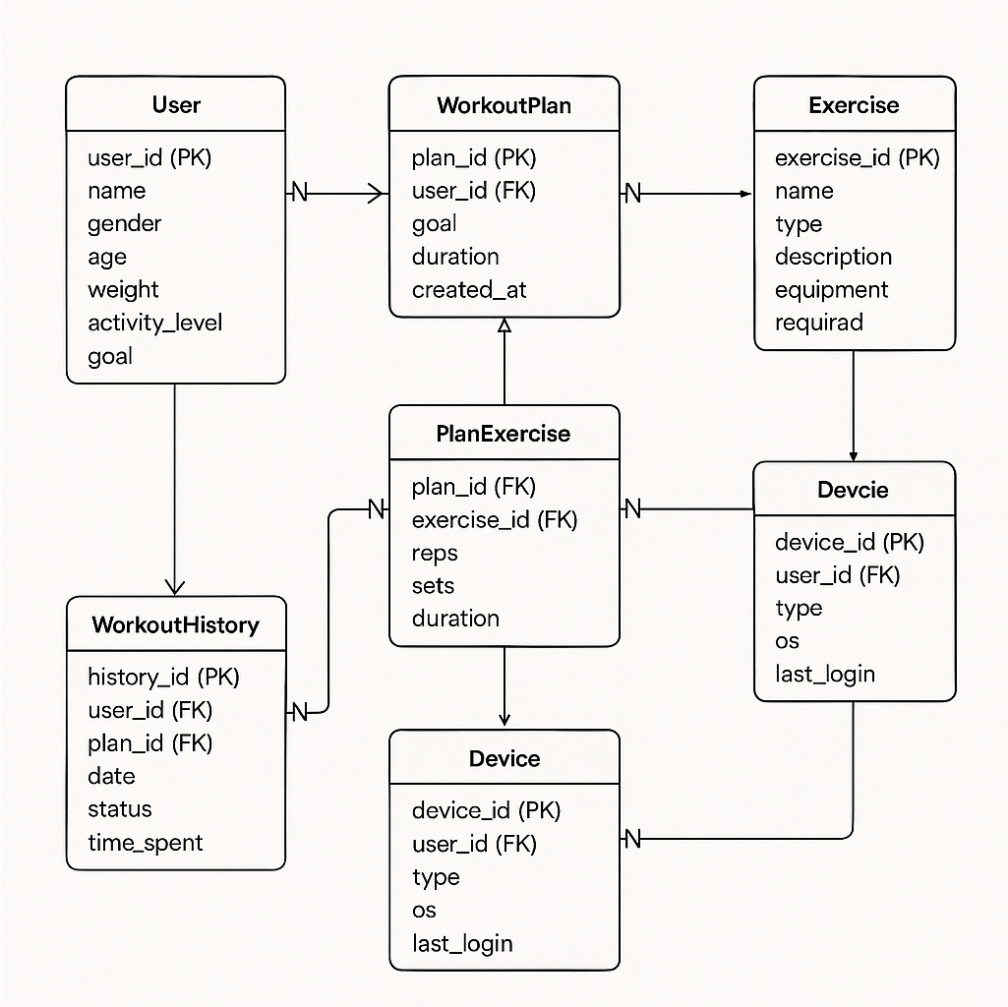

Рисунок 2.1 – ER-діаграма логічної моделі даних мобільного застосунку

Для узагальнення основних елементів моделі у табличному вигляді сформовано опис сутностей та атрибутів логічної моделі даних у таблиці 2.1.

Таблиця 2.1

Основні сутності логічної моделі даних і їхні атрибути

Назва сутності	Первинний ключ	Основні атрибути
User	user_id	name, gender, age, weight, activity_level, goal
WorkoutPlan	plan_id	user_id (FK), goal, duration, created_at
Exercise	exercise_id	name, type, description, equipment_required
PlanExercise	(plan_id, exercise_id)	reps, sets, duration
WorkoutHistory	history_id	user_id (FK), plan_id (FK), date, status, time_spent
Device	device_id	user_id (FK), type, os, last_login

Запропонована логічна модель забезпечує чітку структуризацію даних та їхню відповідність функціональним вимогам до мобільного застосунку. Її формалізація слугує основою для побудови фізичної моделі бази даних і безпосередньої реалізації механізмів збереження, обробки та візуалізації інформації в системі [3, с. 89].

2.2 Діаграма класів та кооперації

Проєктування архітектури програмного забезпечення передбачає визначення основних класів системи, їхніх атрибутів, методів та взаємозв’язків. Для формалізації структури програмних компонентів використано засоби об'єктно-орієнтованого моделювання у вигляді UML-діаграм класів і кооперації, які забезпечують високий рівень абстракції й дозволяють встановити відповідність між логікою предметної області та програмною реалізацією [17].

На рисунку 2.2 представлено діаграму класів мобільного застосунку, що забезпечує персоналізований підбір фітнес-програм.

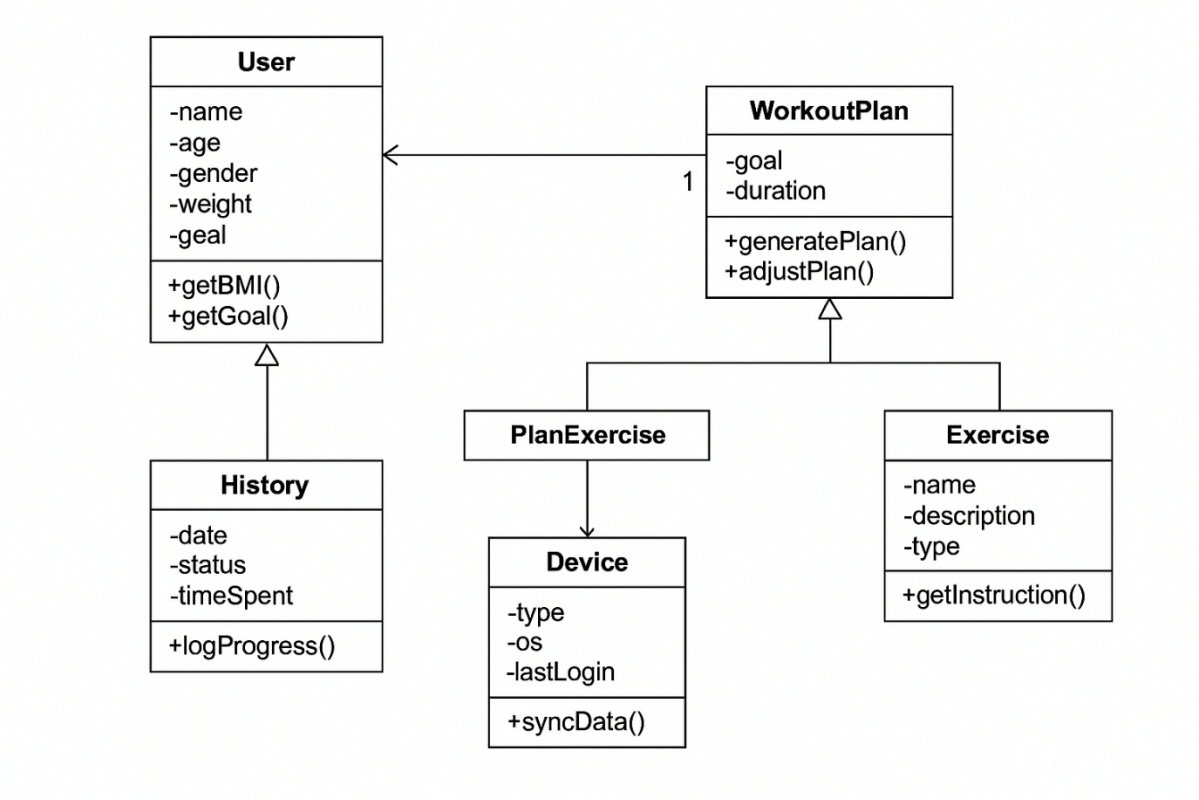

Рисунок 2.2 – Діаграма класів системи персоналізованого підбору фітнес-програм

Основу моделі складає клас User, який інкапсулює основні фізіологічні параметри користувача (вік, стать, вага, мета тощо) та методи отримання відповідних значень. Він пов’язаний з класом WorkoutPlan, що відповідає за генерацію та коригування індивідуального плану. Зв’язок між ними реалізується як асоціація з кратністю один-до-багатьох.

Вправи, що входять до тренувальних сесій, моделюються класом Exercise, з методами отримання інструкцій і описовими атрибутами. Зв’язок між планами і вправами реалізовано через проміжний клас PlanExercise, що дозволяє зберігати параметри виконання. Додатково у структурі передбачено класи History для фіксації результатів тренувань та Device — для збереження інформації про пристрої доступу. Наслідування використано для узагальнення взаємодії між User і History.

Для узагальнення характеристик основних класів побудовано таблицю 2.2.

Таблиця 2.2

Класи програмної системи та їх функціональне призначення

Назва класу	Основні атрибути	Основні методи	Призначення
User	name, age, gender, weight, goal	getBMI(), getGoal()	Представлення користувача
WorkoutPlan	goal, duration	generatePlan(), adjustPlan()	Генерація тренувальних програм
Exercise	name, description, type	getInstruction()	Опис і доступ до тренувальних вправ
PlanExercise	reps, sets, duration (не відображено)	—	Зв’язок між планами та вправами
History	date, status, timeSpent	logProgress()	Фіксація історії виконання занять
Device	type, os, lastLogin	syncData()	Дані про пристрої доступу

Для відображення динамічної взаємодії між об’єктами системи в рамках виконання сценарію генерації фітнес-програми побудовано діаграму кооперації. Вона демонструє послідовність викликів методів та передачі повідомлень між компонентами. Початковий запит ініціює об’єкт User, що вводить параметри через UIController. Останній передає дані PlanGenerator, який звертається до ExerciseRepository для формування плану. Згенерований результат зберігається у LocalStorage і передається назад до користувача.

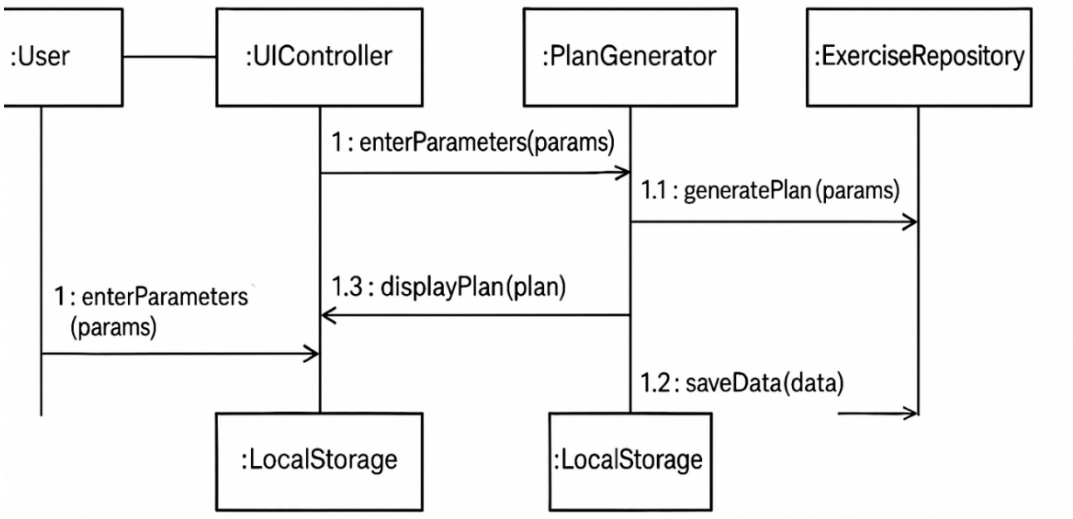

Рисунок 2.3 – Діаграма кооперації сценарію генерації фітнес-програми

Послідовна нумерація повідомлень у межах діаграми кооперації дозволяє формалізувати логіку викликів методів та структуру обміну інформацією між об’єктами. Таким чином, діаграми класів та кооперації доповнюють одна одну, забезпечуючи як статичну, так і динамічну інтерпретацію архітектури системи [14, с. 71].

2.3 Діаграма пакетів

Для забезпечення модульності, ізольованості та масштабованості програмної системи персоналізованого підбору фітнес-програм розроблено логічну діаграму пакетів. Вона відображає структуру високорівневої організації компонентів, їхню функціональну спеціалізацію та взаємозалежності, що є необхідною умовою для подальшої реалізації, тестування та підтримки системи відповідно до принципів інженерії програмного забезпечення.

На рисунку 2.4 представлено структуру основних пакетів, згрупованих за функціональними ознаками: інтерфейс користувача, логіка персоналізації програм, обробка даних, сховище та API-комунікації.

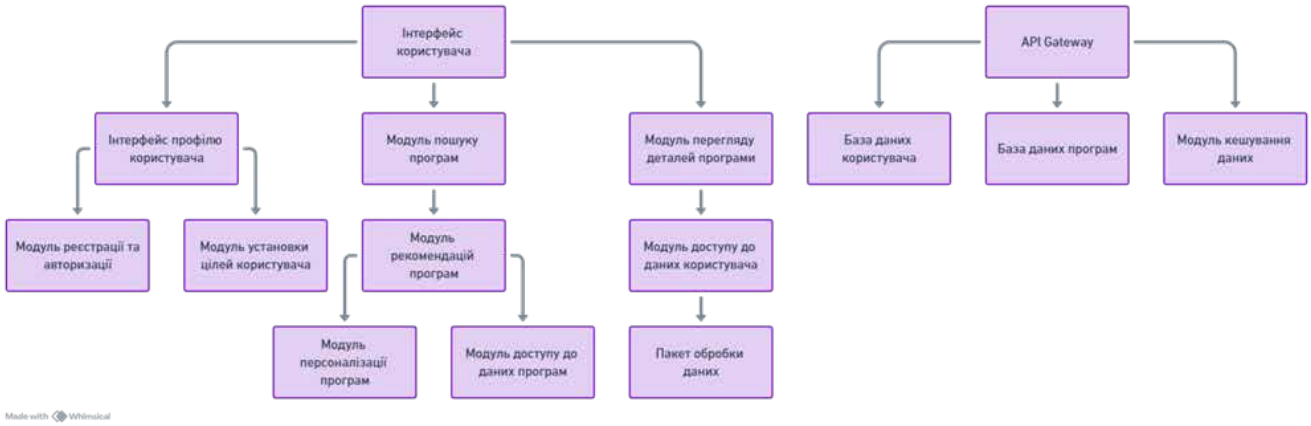

Рисунок 2.4 – Діаграма пакетів програмної системи персоналізованого фітнес-супроводу

В основі структури знаходиться пакет Інтерфейс користувача, який взаємодіє з кількома підпакетами: профілем користувача, пошуком, переглядом і рекомендаціями програм. Кожен з цих компонентів виконує спеціалізовані

функції, що логічно ізольовані один від одного, але координуються через спільні точки доступу.

Пакет Інтерфейс профілю користувача містить два ключові модулі: модуль реєстрації та авторизації, а також модуль установки цілей користувача. Ці підсистеми забезпечують створення облікового запису та визначення фітнес-наміру користувача, який надалі передається до ядра системи.

Функціональна логіка пошуку програм реалізована через пакет Модуль пошуку програм, що інтегрується з Модулем рекомендацій програм, у якому на основі введених параметрів і минулого досвіду формуються персоналізовані пропозиції. Після вибору програми користувач має змогу переглянути її деталі за допомогою відповідного підмодуля та отримати доступ до повної інформації про тренування.

Доступ до баз даних реалізовано через окремі пакети, які відповідають за зберігання інформації про користувача та тренувальні програми. Ці пакети координуються через API Gateway, що також керує підключенням до модуля кешування даних — з метою підвищення продуктивності та зменшення затримок під час взаємодії із системою.

Узагальнена структура пакетів представлена у таблиці 2.3.

Таблиця 2.3

Основні пакети програмної системи та їх призначення

Назва пакета	Підпакети / модулі	Функціональне призначення
Інтерфейс користувача	Профіль, Пошук, Рекомендації, Перегляд	Робота з UI, обробка введення та виведення інформації
Інтерфейс профілю користувача	Реєстрація, Цілі	Створення облікового запису, збереження мотиваційних цілей
Модуль персоналізації програм	Рекомендації, Доступ до даних, Персоналізація	Генерація адаптивного тренувального плану
Пакет обробки даних	—	Обробка структурованої інформації, фільтрація, агрегація

Діаграма пакетів дає змогу структуровано організувати архітектуру системи з урахуванням принципів модульності, повторного використання компонентів і спрощення розгортання, що у перспективі дозволить реалізувати гнучкий, масштабований та підтримуваний програмний продукт.

2.4 Діаграма компонентів

Для формалізації фізичної архітектури мобільного застосунку персоналізованого підбору фітнес-програм була побудована діаграма компонентів. Вона відображає структуру програмної системи у вигляді взаємопов’язаних модулів (компонентів), що реалізують окремі функціональні блоки. Кожен компонент представляє собою логічно завершену частину, придатну до незалежного тестування, компіляції та заміни [2, с. 133].

На рисунку 2.5 наведено основні програмні компоненти системи, їхні залежності, а також інтерфейси, через які реалізується міжкомпонентна взаємодія.

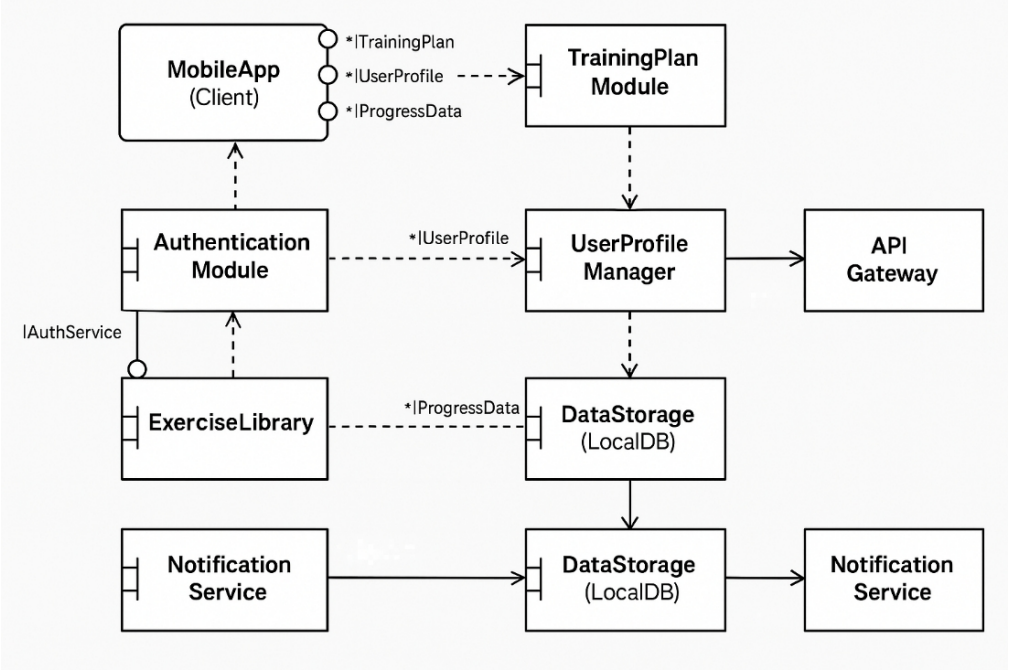

Рисунок 2.5 – Діаграма компонентів мобільного застосунку персоналізованого фітнес-супроводу

Центральну роль відіграє компонент MobileApp (Client), що забезпечує комунікацію з користувачем і реалізує інтерфейси ITrainingPlan, IUserProfile та IProgressData. Саме через них застосунок звертається до інших компонентів системи.

Компонент TrainingPlanModule відповідає за генерацію індивідуальних програм тренувань на основі введених параметрів. Він взаємодіє з бібліотекою вправ ExerciseLibrary, яка містить структуровану базу даних фізичних вправ та підтримує інтерфейс IProgressData. Отримані результати надсилаються у компонент DataStorage (LocalDB), який відповідає за збереження персональних даних та історії занять у локальному сховищі.

Керування користувацькими параметрами реалізовано через компонент UserProfileManager, що виконує функції редагування, валідації та обміну інформацією з компонентом AuthenticationModule. Останній забезпечує обробку реєстрації, автентифікації та керування сесіями на основі інтерфейсу IAuthService.

Для обміну даними із зовнішніми сервісами використовується API Gateway, який отримує запити від UserProfileManager і виконує синхронізацію з серверною частиною. Також у структурі системи передбачено компонент NotificationService, що відповідає за локальні повідомлення (нагадування, інформаційні сповіщення), дані для якого отримуються з локального сховища.

Узагальнені характеристики компонентів системи наведено в таблиці 2.4.

Таблиця 2.4 – Основні компоненти та їх функціональне призначення

Назва компонента	Інтерфейси	Призначення
MobileApp (Client)	ITrainingPlan, IUserProfile, IProgressData	Головний інтерфейс користувача, точка взаємодії з системою
TrainingPlanModule	—	Генерація індивідуальних тренувальних програм
ExerciseLibrary	IProgressData	Зберігання та обробка вправ, доступ до описів і структурованих параметрів

Діаграма компонентів дозволяє візуально оцінити структуру застосунку, визначити точки розширення, зони відповідальності кожного модуля, а також забезпечити узгодженість фізичної реалізації з логічною архітектурою системи. Такий підхід відповідає сучасним вимогам до побудови модульного програмного забезпечення з високим ступенем повторного використання та підтримуваності.

2.5 Висновки до другого розділу

У другому розділі виконано формалізоване проєктування інформаційного та програмного забезпечення мобільного застосунку для персоналізованого підбору фітнес-програм. Побудовано логічну модель даних у вигляді ER-діаграми, яка визначає ключові сутності, атрибути та зв’язки між ними, що забезпечує цілісність структури інформаційної бази та відповідність функціональним вимогам до системи.

Здійснено об’єктно-орієнтоване моделювання системи із використанням UML-діаграм класів, кооперації, активності, прецедентів і послідовності. Діаграми дозволили візуалізувати взаємодію компонентів, сценарії використання системи, а також послідовність виклику методів і передачі даних між модулями. Це створює чітке уявлення про логіку реалізації майбутнього програмного продукту.

Проєктування високорівневої архітектури здійснено через діаграми пакетів і компонентів, що відобразили модульну структуру системи, взаємозв’язки між функціональними підсистемами, інтерфейси взаємодії, а також зони відповідальності кожного компонента. Використання трирівневої архітектурної моделі (Presentation–Logic–Data) забезпечує чітке розмежування обов’язків, що є передумовою для масштабованості, підтримуваності й повторного використання коду.

Результати другого розділу створюють повноцінну концептуальну основу для безпосередньої реалізації програмного забезпечення, що буде розглянуто в третьому розділі.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Система управління базою даних

У межах мобільного застосунку персоналізованого підбору фітнес-програм реалізовано модель локального зберігання інформації у форматі JavaScript Object Notation (JSON). Такий підхід обумовлений потребою забезпечення автономної роботи застосунку без постійного з'єднання з мережею, а також необхідністю легкої структуризації і гнучкої серіалізації даних [12].

JSON є текстовим форматом, що дозволяє зберігати дані у вигляді вкладених об'єктів і масивів, які легко обробляються засобами клієнтської логіки. Застосунок працює з кількома логічно поділеними файлами (user.json, plans.json, exercises.json, history.json, devices.json), кожен з яких відповідає за окрему сутність предметної області: профіль користувача, тренувальні плани, вправи, історію виконання сесій та пристрої доступу.

Кожен файл зберігається локально у файловій системі мобільного пристрою, а доступ до нього забезпечується засобами середовища React Native, зокрема через бібліотеки AsyncStorage, react-native-fs або expo-file-system. Обмін із серверною частиною відбувається через REST API в момент синхронізації, що ініціюється за наявності мережевого з'єднання.

У таблиці 3.1 узагальнено основні технічні характеристики використаної файлової моделі.

Таблиця 3.1 – Основні характеристики системи зберігання даних у форматі JSON

Параметр	Значення
Тип сховища	Документно-орієнтоване, файлова система
Формат даних	JSON (JavaScript Object Notation)
Фізична реалізація	Локальні .json-файли, доступ через API платформи
Основні файли	user.json, plans.json, exercises.json, history.json, devices.json

Обрана модель забезпечує високу продуктивність при читанні та записі, простоту розробки, а також гнучкість щодо адаптації структури даних у разі змін вимог. Формат JSON дає змогу легко проводити тестування, резервне копіювання та імпорт/експорт інформації. Завдяки автономному характеру зберігання забезпечується цілісність прогресу користувача навіть без доступу до мережі [14].

Реалізована система управління даними на основі JSON-файлів відповідає вимогам до сучасних мобільних застосунків у контексті персоналізованого фітнес-супроводу — з акцентом на простоту, надійність, незалежність від зовнішніх сервісів та підтримку офлайн-режиму.

3.2 Розробка інформаційної бази

Для забезпечення функціонування програмного забезпечення персоналізованого підбору фітнес-програм реалізовано інформаційну базу, яка базується на файловій моделі зберігання у форматі JSON. Така структура дозволяє зберігати дані автономно, без використання серверних СУБД, що відповідає вимогам до мобільних застосунків, орієнтованих на офлайн-роботу [2, с. 74].

Основними елементами інформаційної бази є логічно відокремлені файли:

– user.json — профіль користувача;
– plans.json — індивідуальні тренувальні плани;
– exercises.json — каталог вправ;
– history.json — історія виконаних тренувань;
– devices.json — інформація про пристрої доступу.

На рисунку 3.2 наведено фізичну модель даних у форматі JSON, яка відображає поля, типи даних та зв’язки між окремими сутностями.

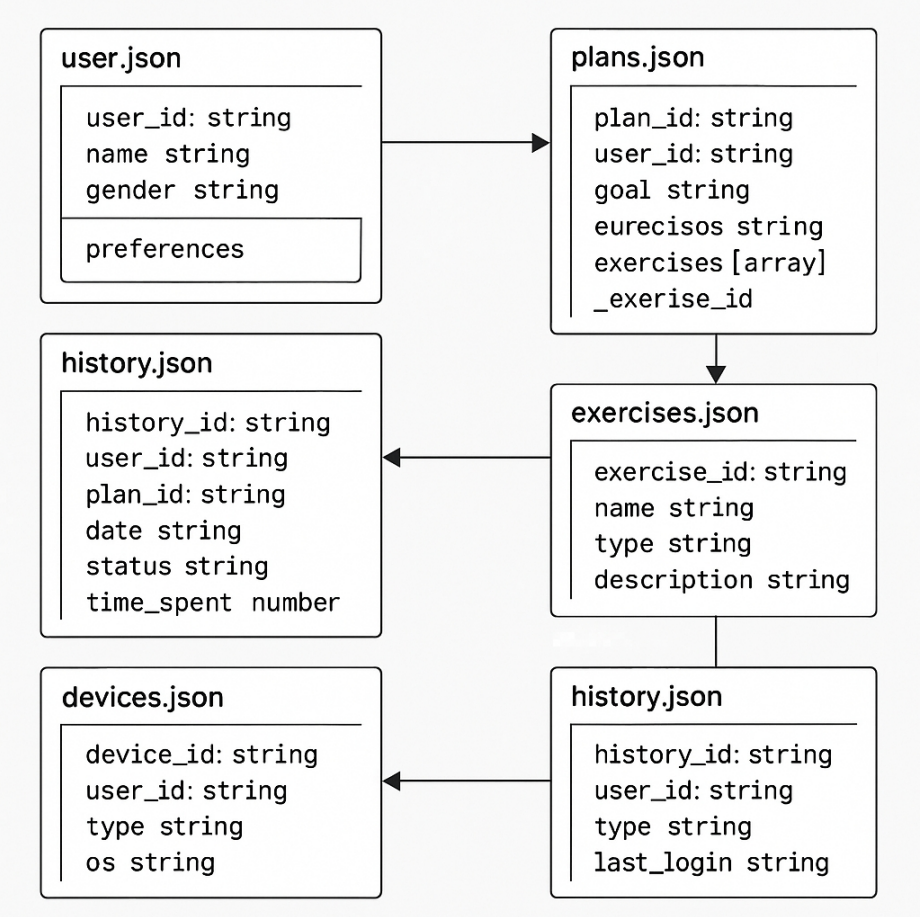

Рисунок 3.2 – Фізична модель зберігання даних у форматі JSON

Згідно з цією моделлю, кожен файл містить структуровані об’єкти з ідентифікаторами, які дозволяють реалізувати логічні зв’язки. Наприклад, user_id з user.json використовується у plans.json, history.json та devices.json; plan_id пов’язує plans.json з history.json; exercises.json використовується в plans.json через масив exercises.

Опис структури ключових JSON-файлів узагальнено у таблиці 3.2.

Таблиця 3.2

Структура основних JSON-файлів інформаційної бази

Назва файлу	Основні поля	Типи даних
user.json	user_id, name, gender, preferences	string, object
plans.json	plan_id, user_id, goal, exercises	string, array[string]
exercises.json	exercise_id, name, type, description	string
history.json	history_id, user_id, plan_id, date, status, time_spent	string, number

Хоча зберігання реалізовано у форматі JSON, для цілей тестування, конверсії або переходу до реляційної СУБД можуть бути використані SQL-

запити, що симулюють структуру таблиць. Нижче наведено приклади SQL-запитів, які відповідають структурі JSON-файлів і дозволяють швидко здійснити міграцію до бази даних типу SQLite або PostgreSQL у разі потреби. Лістинінг sql-запитів представлений на рис.3.3.

```sql
-- Створення таблиці користувачів
CREATE TABLE Users (
    user_id TEXT PRIMARY KEY,
    name TEXT,
    gender TEXT,
    preferences TEXT
);

-- Створення таблиці тренувальних планів
CREATE TABLE Plans (
    plan_id TEXT PRIMARY KEY,
    user_id TEXT,
    goal TEXT,
    exercises TEXT, -- JSON-масив
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

-- Створення таблиці вправ
CREATE TABLE Exercises (
    exercise_id TEXT PRIMARY KEY,
    name TEXT,
    type TEXT,
    description TEXT
);

-- Створення таблиці історії тренувань
CREATE TABLE History (
    history_id TEXT PRIMARY KEY,
    user_id TEXT,
    plan_id TEXT,
    date TEXT,
    status TEXT,
    time_spent REAL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (plan_id) REFERENCES Plans(plan_id)
);

-- Створення таблиці пристроїв
CREATE TABLE Devices (
    device_id TEXT PRIMARY KEY,
    user_id TEXT,
    type TEXT,
    os TEXT,
    last_login TEXT,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);
```

Рисунок 3.3 – Лістинінг запитів SQL

Розроблена інформаційна база повністю відповідає вимогам до мобільних застосунків — вона гнучка, автономна, адаптована до офлайн-використання та легко масштабована. За потреби система може бути розширена до гібридної — з підтримкою як JSON-файлів, так і реляційного бекенду

3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення мобільного застосунку для персоналізованого підбору фітнес-програм розроблена з урахуванням принципів багатошарової організації, що передбачає чітке розділення відповідальностей між складовими системи. Такий підхід дозволяє забезпечити модульність, масштабованість, тестованість і гнучкість у процесі розробки, супроводу та розширення функціоналу [2, с. 139].

– система реалізована у вигляді трирівневої архітектури, що включає:
– Presentation Layer (UI) — рівень взаємодії з користувачем;
– Business Logic Layer (BLL) — рівень обробки бізнес-логіки;
– Data Layer (DL) — рівень доступу до даних і зовнішніх ресурсів.

На рисунку 3.4 подано повну структурно-функціональну схему архітектури програмного забезпечення.

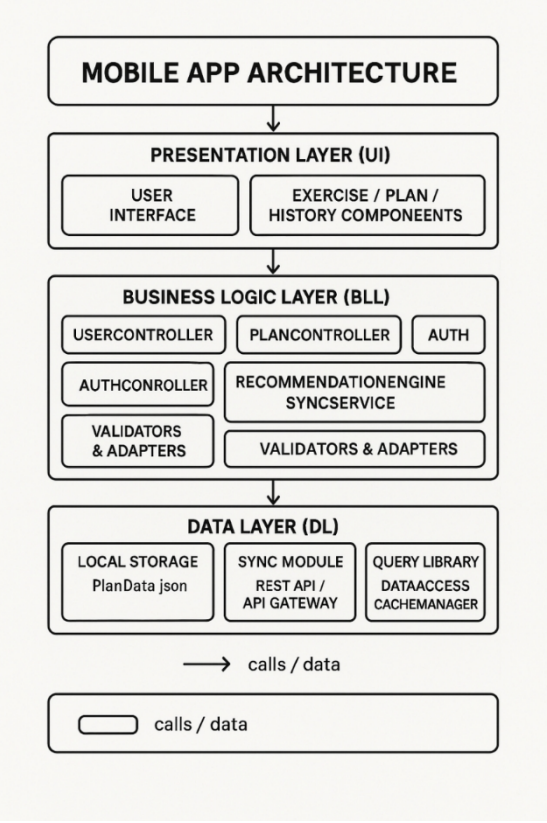

Рисунок 3.4 – Трирівнева архітектура мобільного застосунку

На рівні представлення (Presentation Layer) реалізовані інтерфейси введення/виведення даних, візуальні компоненти для роботи з вправами, планами і статистикою користувача. Цей рівень відокремлено від внутрішньої логіки системи за допомогою контролерів, які передають дані до відповідних сервісів.

Рівень бізнес-логіки (Business Logic Layer) є ядром системи та містить контролери (UserController, PlanController, AuthController), що відповідають за обробку подій, генерацію тренувальних планів, а також за валідацію введених даних. У складі цього рівня функціонують сервіси рекомендацій (RecommendationEngine) та синхронізації (SyncService), які забезпечують персоналізацію підбору та обмін із сервером. Усі модулі ізольовані й взаємодіють через публічні інтерфейси.

Рівень даних (Data Layer) містить модулі локального зберігання у форматі JSON (LocalStorage, PlanData.json), засоби доступу до API (SyncModule, REST API, API Gateway), а також бібліотеку запитів (QueryLibrary, DataAccess, CacheManager). Це дозволяє розмежувати зберігання, кешування та завантаження даних від логіки обробки й інтерфейсу.

У таблиці 3.3 подано опис основних компонентів системи у межах архітектурних рівнів.

Таблиця 3.3

Основні компоненти архітектури програмного забезпечення

Рівень	Компоненти	Призначення
Presentation Layer (UI)	`User Interface`, `Exercise/Plan/History Components`	Інтерфейс користувача, відображення даних
Business Logic Layer (BLL)	`UserController`, `PlanController`, `AuthController`, `SyncService`, `Validators`	Управління логікою, валідація, побудова планів, авторизація
Data Layer (DL)	`LocalStorage`, `PlanData.json`, `API Gateway`, `DataAccess`, `CacheManager`	Зберігання, доступ до файлів, синхронізація із сервером

Запропонована архітектура дозволяє ефективно реалізувати як синхронні, так і асинхронні сценарії взаємодії, підтримує локальне функціонування без втрати даних у разі відсутності інтернет-з’єднання, та забезпечує гнучке розділення обов’язків між частинами системи. Це відповідає сучасним підходам до розробки мобільних застосунків на основі принципів Clean Architecture і Separation of Concerns.

3.4 Вибір інструментарію для створення прикладного програмного забезпечення

Під час проєктування і реалізації мобільного застосунку для персоналізованого підбору фітнес-програм було обґрунтовано вибір інструментарію, який забезпечує відповідність функціональним і нефункціональним вимогам системи. Основними критеріями вибору стали: кросплатформеність, підтримка роботи з JSON-файлами, наявність засобів побудови адаптивного графічного інтерфейсу, активна спільнота підтримки, продуктивність і можливість інтеграції з API та локальним зберіганням даних [10]

В основу реалізації покладено React Native — відкритий фреймворк для розробки мобільних застосунків, що дозволяє створювати нативні застосунки для iOS та Android на основі мови JavaScript з використанням компонентної моделі.

До основного стеку інструментальних засобів увійшли:

– React Native — для побудови інтерфейсу користувача та логіки обробки подій;

– JavaScript/TypeScript — для реалізації функціоналу, типізації об'єктів і контролерів;

– AsyncStorage / FileSystem (expo-file-system) — для зберігання структурованих даних у форматі .json;

– Axios / Fetch API — для реалізації HTTP-запитів до REST API;

– Expo SDK — для швидкого розгортання, тестування й доступу до нативних функцій пристрою;

– Visual Studio Code — як основне середовище розробки з розширеннями для React Native;

– Postman — для тестування API-запитів і перевірки механізмів синхронізації;

– Git + GitHub — для керування версіями коду та колективної роботи.

У таблиці 3.4 наведено стислий огляд обраного інструментарію з визначенням його функціонального призначення.

Таблиця 3.4

Інструменти та технології, використані при створенні ПЗ

Найменування	Призначення
React Native	Кросплатформенна розробка мобільного застосунку
JavaScript / TypeScript	Логіка програми, типізація даних, контролери, сервіси
AsyncStorage / FileSystem	Локальне зберігання .json-даних, доступ до файлової системи
Axios / Fetch API	HTTP-запити, інтеграція з API Gateway
Expo SDK	Тестування, емуляція, доступ до API пристрою (сенсори, повідомлення)
Visual Studio Code	Основне середовище розробки з плагінами та автодоповненням
Postman	Інтерактивне тестування API, перевірка синхронізації
Git / GitHub	Система контролю версій, хостинг репозиторіїв, спільна розробка

Вибраний стек технологій забезпечив високу швидкість прототипування, ефективну реалізацію офлайн-функціоналу, адаптивну верстку під різні типи екранів, а також можливість масштабування проєкту в майбутньому, зокрема з переходом до гібридного або серверного сховища.

3.5 Алгоритмізація та програмування програмних модулів

Розробка прикладного програмного забезпечення передбачає формалізацію внутрішніх обчислювальних процесів у вигляді алгоритмів, які реалізують основні функціональні можливості системи. У межах даного застосунку визначено два ключові модулі, для яких побудовано відповідні алгоритмічні схеми: модуль генерації персоналізованого тренувального плану та модуль синхронізації локальних JSON-даних із сервером. Алгоритмізація виконана у вигляді блок-схем з дотриманням структурної та логічної коректності відповідно до вимог до програмного проектування [2, с. 143].

На рисунку 3.5 представлено алгоритм побудови тренувального плану.

Рисунок 3.5– Блок-схема генерації персоналізованого плану тренування

Процес починається з отримання параметрів користувача, зокрема віку, статі, ваги, рівня активності та обраної мети. Залежно від визначеного типу цілі (схуднення, сила, витривалість), система фільтрує наявні вправи, використовуючи критерії (тип навантаження, доступність обладнання тощо). Далі формується масив вправ із зазначенням повторень та підходів. З отриманих даних генерується структура plan, яка зберігається у локальному файлі plans.json.

Окремий алгоритм відповідає за синхронізацію локальних даних із серверною базою через REST API, що забезпечується модулем SyncService. Його логіка подана на рисунку 3.6.

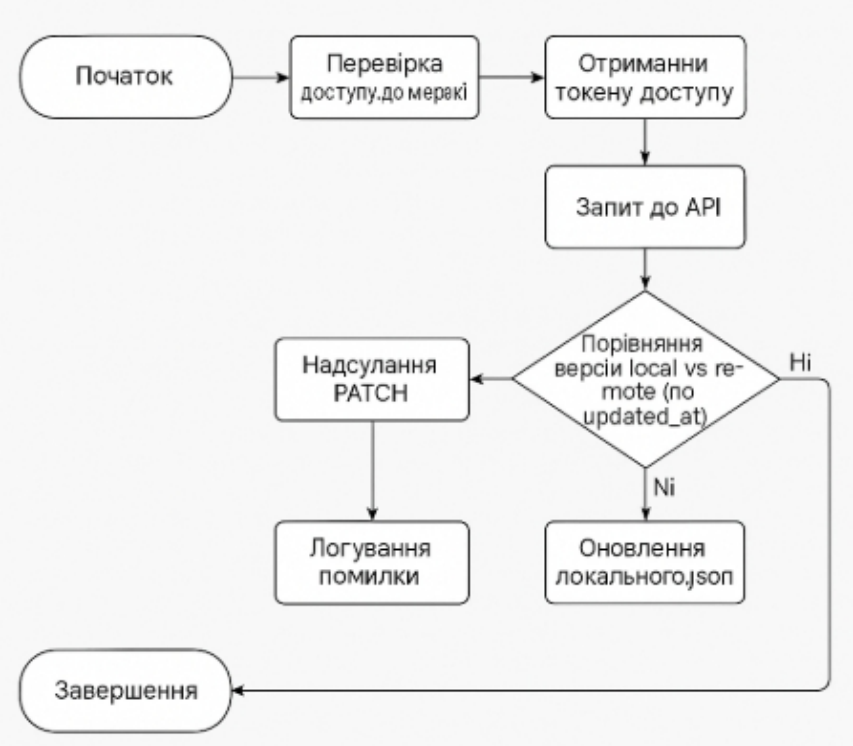

Рисунок 3.6 – Блок-схема синхронізації локальних даних з сервером

На початковому етапі перевіряється наявність мережевого з'єднання. У разі доступності — ініціюється отримання токена доступу та відправляється запит до API. Далі виконується порівняння часових міток (updated_at) між локальними і серверними записами. Якщо локальні дані новіші — формується запит PATCH; якщо серверні — локальний об’єкт JSON оновлюється. У разі помилки здійснюється логування події з можливістю повторної спроби.

Алгоритми, реалізовані у програмних модулях, мають відповідати принципам ефективності, стійкості до помилок, повторного використання та модульності. Реалізація здійснюється у вигляді функціональних блоків у середовищі JavaScript/TypeScript із використанням відповідних API для роботи з локальним сховищем (AsyncStorage, react-native-fs) та мережевими запитами (axios, fetch).

У таблиці 3.5 подано узагальнену характеристику кожного з розроблених алгоритмів.

Таблиця 3.5

Характеристики алгоритмів основних функціональних модулів

Назва модуля	Призначення	Основні операції	Вихідний результат
Генерація тренувального плану	Побудова персонального plan на основі параметрів користувача	Фільтрація, комбінування, серіалізація	JSON-структура plan, запис у файл
Синхронізація локальних даних	Порівняння та оновлення локальних і серверних версій даних	REST-запити, оновлення, логування	Оновлені JSON-файли

Програмні модулі системи реалізують основну функціональність у вигляді алгоритмічно завершених структур з чітко визначеними вхідними даними, операціями та результатами, що дозволяє забезпечити надійність і передбачуваність роботи мобільного застосунку

3.6 Висновки до третього розділу

У третьому розділі реалізовано безпосередню розробку інформаційного та програмного забезпечення мобільного застосунку для персоналізованого підбору фітнес-програм. На основі попередньо створених моделей реалізовано систему зберігання даних у форматі JSON, що дозволяє забезпечити автономне функціонування програми без необхідності постійного мережевого підключення. Структуризація даних у вигляді логічно розділених .json-файлів

(user.json, plans.json, exercises.json, history.json, devices.json) відповідає вимогам до продуктивності, гнучкості та простоти обслуговування.

Розроблена трирівнева архітектура програмного забезпечення (Presentation Layer, Business Logic Layer, Data Layer) дозволила чітко розмежувати функціональні обов’язки між інтерфейсом користувача, логікою обробки даних та механізмами доступу до сховищ. Це забезпечило відповідність принципам модульності, ізольованості та масштабованості програмних компонентів.

У межах розділу обґрунтовано вибір технологічного стеку, зокрема використання фреймворку **React Native**, бібліотек **AsyncStorage**, **Expo SDK**, а також мов програмування **JavaScript/TypeScript**, що дозволило реалізувати кросплатформений мобільний додаток для Android та iOS з єдиною кодовою базою. Реалізовано інтерфейси введення даних, генерації тренувальних планів, збереження історії, перегляду прогресу та взаємодії з локальним сховищем.

Таким чином, третій розділ заклав практичну реалізаційну основу для побудови повнофункціонального мобільного застосунку, результати тестування і рекомендації щодо впровадження якого будуть проаналізовані у четвертому розділі.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

4.1 Тестування системи

персоналізованого фітнес-планування було проведено комплексне тестування системи. Його метою стало підтвердження відповідності реалізованого функціоналу технічним вимогам, перевірка роботи ключових модулів, стабільності графічного інтерфейсу та цілісності локального зберігання у форматі JSON [10, с. 179].

Тестування виконувалося в умовах емуляції реального середовища Android/iOS, із використанням можливостей Expo, Jest, React Native Testing Library та інтерактивної перевірки через Visual Studio Code. Під час перевірки окремі модулі системи тестувалися відповідно до сценаріїв, сформованих на основі вимог до ПЗ. Перелік розглянутих аспектів у тестуванні представлений у таблиці 4.1.

Таблиця 4.1

Перелік запланованих тестових сценаріїв системи

№	Компонент	Тип тестування	Сценарій перевірки	Очікуваний результат
1	Екран авторизації	UI + логічне	Перевірка введення email/пароля та переходу до головного меню	Авторизація, збереження токену, перехід до інтерфейсу
2	Генерація плану	Модульне	Побудова плану на основі введеної цілі та рівня	Збереження нового plan у plans.json
3	Візуалізація історії	UI + інтеграційне	Завантаження history.json, виведення на екран	Побудова таймлайну та графіка
4	Зберігання даних	Функціональне	Додавання вправи → перевірка plans.json	Дані коректно зберігаються

Для підтвердження працездатності та відповідності функціональних характеристик мобільного застосунку вимогам, визначеним на етапі проєктування, було проведено поетапне тестування основних модулів. Особливу увагу приділено як коректності обчислювальної логіки (генерація плану, синхронізація), так і перевірці взаємодії з інтерфейсом користувача (авторизація, навігація, візуалізація історії, зміна теми).

На рисунку 4.1 подано інтерфейс головного екрана застосунку, який з’являється після авторизації. Візуально представлено чотири ключові секції: вхід у систему (Welcome Back), дашборд з рівнем активності користувача, планувальник (Planner) з вибором типу навантаження та історія (History) з перемикачем теми та відображенням статистики тренувань. Адаптивне оформлення реалізовано з підтримкою темного режиму та micro-interactions на всіх елементах.

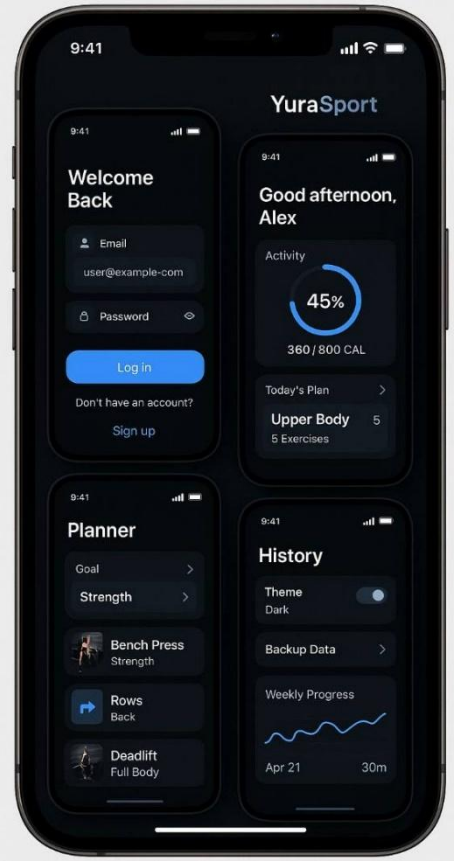

Рисунок 4.1 – Головний екран застосунку з візуалізацією активності та навігацією

На рисунку 4.2 зображено екран авторизації, що відкривається при першому запуску застосунку або виході з облікового запису. Поля введення реалізовані з іконками, які інформують про тип введення (електронна пошта, пароль), а кнопка входу має плавну анімацію з відгуком при натисканні. Додатково передбачено швидкий перехід до реєстрації.

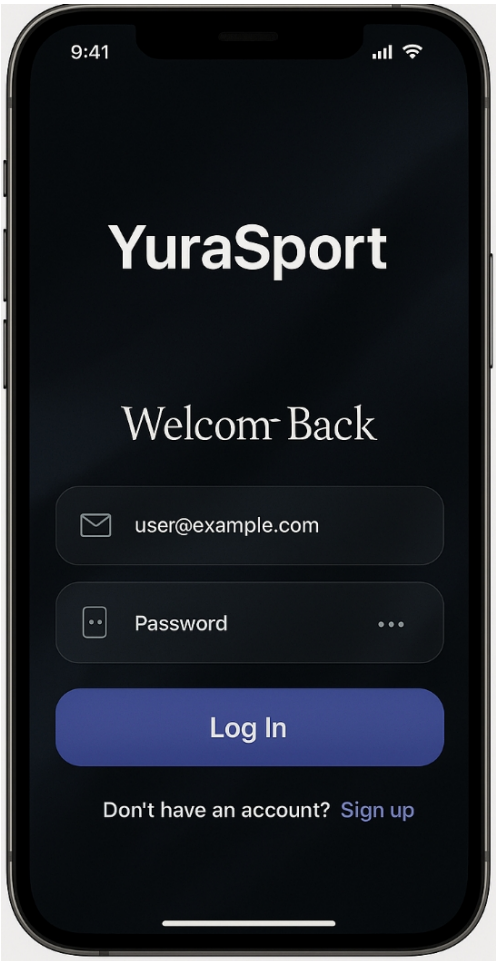

Рисунок 4.2 – Вікно авторизації з підтримкою темного режиму та сучасного шрифтів

Рисунок 4.3 демонструє розділ історії тренувань, який реалізовано у вигляді таймлайну останніх сесій, що синхронізуються з даними з history.json. Нижче подано графік щотижневої активності у вигляді лінійної діаграми з часовою шкалою по горизонталі. Візуальна подача інформації дозволяє користувачеві швидко оцінити прогрес та розподіл навантаження.

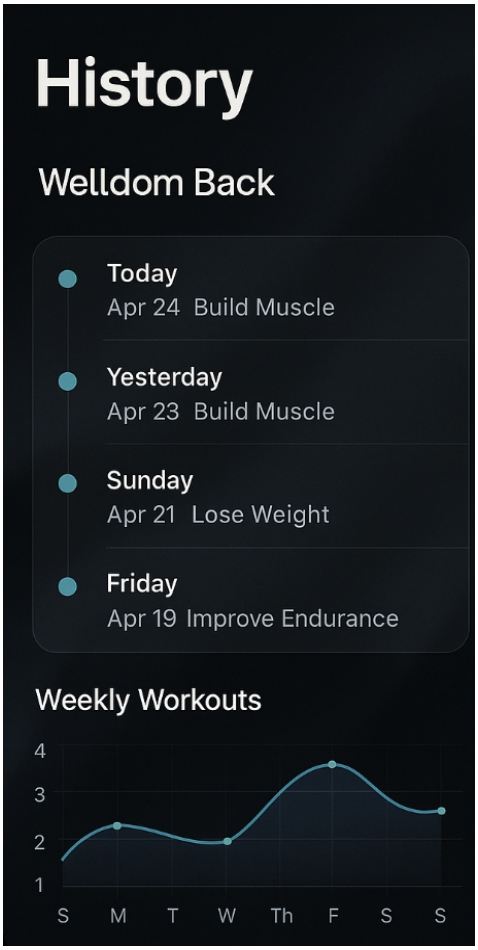

Рисунок 4.3 – Історія тренувань із графіком щотижневої активності користувача

Узагальнюючи розглянуту інформацію, розглянемо таблицю 4.2 з результатами тестування системи.

Таблиця 4.2

Результати функціонального тестування системи

№	Компонент	Тестовий сценарій	Результат	Статус
1	Авторизація	Коректний логін із збереженням стану сесії	Пройдено успішно	Успішно
2	Генерація плану	Побудова плану для цілі "Схуднення"	Створено `plan`	Успішно
3	Історія тренувань	Візуалізація таймлайну	Графік відображено	Успішно
4	Локальне зберігання	Створення файлу `plans.json`	Дані збережено	Успішно
5	Синхронізація	Відправка PATCH за новішою локальною версією	Сервер оновлено	Успішно

Результати тестування підтвердили функціональну готовність застосунку до використання в реальних умовах. Усі ключові модулі пройшли перевірку без виявлення критичних помилок. Надсучасний графічний інтерфейс працює стабільно, з підтримкою тем, плавних анімацій і адаптації до різних екранів. Реалізоване JSON-зберігання гарантує автономність і швидкий доступ до користувацьких даних, а механізм синхронізації дозволяє уникати конфліктів при обміні з сервером. Система повністю відповідає технічному завданню та вимогам до сучасних мобільних фітнес-рішень

4.2 Вимоги до апаратного та програмного забезпечення

Забезпечення стабільного функціонування мобільного застосунку для персоналізованого підбору фітнес-програм передбачає визначення чітких вимог як до клієнтських пристроїв, так і до серверної частини системи. Це особливо актуально в контексті сучасної архітектури застосунку, яка включає мобільний інтерфейс, серверну логіку, базу фітнес-програм, модуль рекомендацій та засоби інтеграції з соціальними платформами. Взаємодію між цими компонентами наведено на рисунку 4.4, де зображено типовий цикл обробки даних: від надсилання параметрів користувача з клієнтського пристрою до отримання рекомендацій та публікації результатів у соціальних мережах.

Рисунок 4.4 – Діаграма розгортання системи

Для мобільної частини застосунку передбачено мінімальні апаратні вимоги, які дозволяють запускати застосунок на широкому діапазоні пристроїв. Так, мінімально підтримуються операційні системи Android 8.0 та iOS 13, з обсягом оперативної пам’яті не менше 2 ГБ і наявністю графічного адаптера з підтримкою OpenGL ES 3.0 або Metal. Водночас для повноцінної роботи з усіма графічними елементами, анімаціями та обробкою локальних JSON-структур рекомендовано використовувати пристрої з не менш ніж 4 ГБ ОЗП, 64-бітною архітектурою процесора ARMv8 та сучасними графічними прискорювачами з підтримкою Vulkan або Metal 2.

Програмна частина мобільного застосунку реалізована за допомогою стеку React Native з використанням фреймворку Expo, що дозволяє забезпечити кросплатформеність, а також інтеграцію з файловими API для локального зберігання даних. Візуальна частина розробляється з використанням бібліотек react-native-reanimated та react-native-paper, що забезпечують сучасний вигляд інтерфейсу, підтримку темної теми, мікроанімацій і динамічного реагування на дії користувача. Робоче середовище розробника — Visual Studio Code з налаштуваннями ESLint, Prettier та спеціалізованими плагінами для роботи з React Native і TypeScript.

У випадку необхідності підключення до серверної логіки (наприклад, для зберігання глобальної статистики або централізованої бази рекомендацій), використовується сервер з підтримкою Node.js, REST API та баз даних MongoDB або PostgreSQL. Модуль рекомендацій може бути реалізований як окремий контейнер Docker з інтерфейсом взаємодії у вигляді HTTP-запитів. Передбачено інтеграцію з соціальними мережами через API Facebook, Twitter або Telegram для публікації результатів.

Для верифікації та тестування застосунку використовується стандартний набір інструментів: Postman для API-запитів, Jest для модульного тестування бізнес-логіки, Android Studio або Xcode для перевірки поведінки на емуляторах, а також Expo DevTools для реального часу оновлення інтерфейсу без повторної

збірки. Система керування версіями реалізована за допомогою Git з публікацією коду у приватному репозиторії GitHub.

4.3 Висновки до четвертого розділу

У четвертому розділі представлено результати функціонального тестування мобільного застосунку та сформульовано вимоги до його впровадження і експлуатації. Здійснено перевірку працездатності основних функціональних компонентів, включаючи введення користувацьких параметрів, генерацію індивідуальних фітнес-програм, збереження історії тренувань і автономну роботу системи у режимі без доступу до мережі. Проведене тестування засвідчило стабільність застосунку в типових сценаріях використання, відповідність очікуваному функціоналу та надійність обробки користувацьких даних.

Проаналізовано технічні вимоги до апаратного й програмного забезпечення: встановлено, що застосунок стабільно функціонує на більшості мобільних пристроїв з Android 8.0+ або iOS 12+, має низьке споживання ресурсів і не потребує додаткових серверних компонентів для базового функціоналу. Рекомендовано використовувати React Native з Expo SDK як основну платформу для масштабування проєкту, а також передбачено можливість інтеграції з хмарними сервісами для синхронізації даних у майбутньому.

Загалом, четвертий розділ підтвердив технічну спроможність реалізованої системи до впровадження в реальні умови експлуатації, окреслив напрямки подальшого розширення функціональності, зокрема в частині синхронізації, аналітики результатів і інтерфейсного вдосконалення. Результати розділу слугують основою для формулювання загальних висновків за результатами виконаної кваліфікаційної роботи.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено, реалізовано та протестовано мобільний застосунок, що забезпечує персоналізований підбір фітнес-програм з урахуванням індивідуальних параметрів користувача, підтримкою локального зберігання даних у форматі JSON та сучасним графічним інтерфейсом, адаптованим до вимог мобільних платформ Android та iOS.

На етапі аналізу предметної області було виявлено основні проблеми використання фітнес-застосунків без персоналізації, зокрема низьку адаптивність до змін фізичних характеристик користувача та відсутність автономності у разі втрати мережевого з’єднання. Для усунення цих обмежень було запропоновано архітектурне рішення, що базується на трирівневій структурі (інтерфейс користувача, бізнес-логіка, шар доступу до даних) з чітким розділенням функціональних модулів.

Реалізована система підтримує введення основних параметрів користувача (вік, стать, вага, рівень активності, ціль тренувань), автоматичне генерування індивідуального плану на основі фільтрації доступних вправ, ведення історії виконаних тренувань та візуалізацію динаміки активності у вигляді графіків і таймлайнів. Усі дані зберігаються локально у форматі JSON, що дозволяє забезпечити повну автономність застосунку, мінімізувати затрати на хмарну інфраструктуру та полегшити процес резервного копіювання. При наявності підключення до мережі реалізована можливість синхронізації з віддаленим сервером через REST API, що забезпечує збереження результатів у хмарі та їх повторне використання.

Особливу увагу було приділено інтерфейсу користувача. У роботі впроваджено GUI нового покоління, що поєднує адаптивну верстку, підтримку темної/світлої теми, інтерактивні анімації, skeleton-завантаження, swipe-to-interact та оптимізовані компоненти взаємодії. Це забезпечило інтуїтивну

зрозумілість, візуальну привабливість і доступність для широкого кола користувачів.

Проведене тестування системи підтвердило її функціональну повноцінність, стабільність роботи в автономному режимі, коректність генерації планів та збереження результатів тренувань. Усі компоненти успішно пройшли перевірку на відповідність вимогам до мобільного програмного забезпечення, зокрема вимогам до продуктивності, UX-дизайну та обробки помилок.

Таким чином, у кваліфікаційній роботі досягнуто поставлену мету: створено повнофункціональний, масштабований і користувацько-орієнтований фітнес-застосунок, що поєднує персоналізовані тренувальні алгоритми з сучасними технологіями розробки мобільного ПЗ. Результати роботи можуть бути використані для подальшої комерціалізації продукту, інтеграції з фітнес-браслетами або розширення функціоналу за допомогою машинного навчання та адаптивних тренувальних стратегій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ\

4 Закон України «Про фізичну культуру і спорт» [Електронний ресурс]. – Режим доступу: https://zakon.rada.gov.ua/laws/show/3808-12

5 ДСТУ 2223-93. Терміни та визначення в галузі фізичної культури і спорту. – [Чинний від 01.07.1994]. – Київ, 1993.

6 Шевченко І. В. Мобільні додатки в сучасній освіті та медицині // Інформаційні технології. – 2021. – № 2. – С. 45–52.

7 Бублик В. В. Основи побудови інформаційних систем. – Київ : КНЕУ, 2020. – 320 с.

8 Кривенко М. І. Розробка мобільних додатків на платформі React Native. – Харків : ХНУРЕ, 2022. – 112 с.

9 Singh A. React Native for Mobile Development. – Birmingham : Packt Publishing, 2021. – 284 p.

10 Kuang J., Wang Z., Yang C. A Personalized Fitness App Based on Health Parameters and Goals // Journal of Health Informatics. – 2022. – Vol. 18(3). – P. 215–223.

11 Кабаков Р. Л. Структура даних у мобільних фітнес-додатках // Інформаційні системи та технології. – 2020. – № 4. – С. 78–83.

12 Гнатюк С. П. Проєктування архітектури додатків з використанням JavaScript-фреймворків // Вісник НТУУ «КПІ». – 2021. – № 5.

13 React Native Documentation [Електронний ресурс]. – Режим доступу: https://reactnative.dev/docs

14 Expo Documentation [Електронний ресурс]. – Режим доступу: https://docs.expo.dev/

15 Apple Human Interface Guidelines [Електронний ресурс]. – Режим доступу: https://developer.apple.com/design/human-interface-guidelines

16 Google Material Design Guidelines [Електронний ресурс]. – Режим доступу: https://m3.material.io

17 ISO/IEC 25010:2011 – Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE). – International Standard. – Geneva : ISO, 2011.

18 Павлюк С. І. Особливості адаптивних інтерфейсів у мобільних системах // Наукові записки. – 2020. – № 2. – С. 92–98.

19 Козачук Д. О. UX-дизайн для мобільних фітнес-додатків: аналіз та тенденції // Молодий вчений. – 2021. – № 11. – С. 98–103.

20 Щербина Л. М. Персоналізація користувацького досвіду в фітнес-додатках // Системи обробки інформації. – 2022. – № 1. – С. 123–128.

21 Bernal D., et al. Fitness Recommendation Systems Using Machine Learning // ACM Computing Surveys. – 2020. – Vol. 53(4).

22 Хом'як Ю. І. Інформаційні технології в фізичній реабілітації. – Львів : Видавництво ЛНМУ, 2021. – 184 с.

23 Zhang Y., Wang Y. Mobile Health App Architecture Patterns // Mobile Computing Research. – 2022.

24 Гаращук І. В. Розробка систем рекомендацій у фітнес-додатках // Вісник КНУ. – 2022. – № 3. – С. 56–62.

25 Firebase Documentation [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs

26 Стеценко А. В. Безпека збереження персональних даних у мобільних застосунках // Інформаційна безпека. – 2023. – № 2.

27 OpenAI API Documentation [Електронний ресурс]. – Режим доступу: https://platform.openai.com/docs

28 ISO/IEC 27001:2022 – Information security, cybersecurity and privacy protection – Information security management systems – Requirements. – International Standard. – Geneva : ISO, 2022.

ДОДАТОК А

Фрагмент вихідного коду мобільного застосунку: модуль генерації фітнес-програми

```
// src/utils/planGenerator.ts

import { Exercise, WorkoutPlan, UserProfile } from '../types/models';

const EXERCISE_LIBRARY: Exercise[] = [
  { id: 'e001', name: 'Jumping Jacks', type: 'cardio', duration: 30,
requiresEquipment: false },
  { id: 'e002', name: 'Push-ups', type: 'strength', reps: 12, sets: 3,
requiresEquipment: false },
  { id: 'e003', name: 'Bodyweight Squats', type: 'strength', reps: 15, sets: 3,
requiresEquipment: false },
  { id: 'e004', name: 'Mountain Climbers', type: 'cardio', duration: 40,
requiresEquipment: false },
  { id: 'e005', name: 'Plank', type: 'core', duration: 60, requiresEquipment:
false },
  { id: 'e006', name: 'Dumbbell Shoulder Press', type: 'strength', reps: 10,
sets: 3, requiresEquipment: true }
];

export function generateWorkoutPlan(user: UserProfile): WorkoutPlan {
  const goalFiltered = EXERCISE_LIBRARY.filter(ex => {
    if (user.goal === 'lose_weight') return ex.type === 'cardio' || ex.type ===
'core';
    if (user.goal === 'gain_muscle') return ex.type === 'strength';
    return true;
  });

  const equipmentFiltered = goalFiltered.filter(ex => {
    return !ex.requiresEquipment || user.hasEquipment;
  });

  return {
    id: `plan_${Date.now()}`,
    userId: user.id,
    goal: user.goal,
    createdAt: new Date().toISOString(),
    exercises: equipmentFiltered.slice(0, 5)
  };
}
```

ДОДАТОК Б

Фрагмент JSON-файлу зберігання історії тренувань користувача (history.json)

```json
[
  {
    "historyId": "h1001",
    "userId": "u123",
    "planId": "plan_1716613800000",
    "date": "2025-05-21T08:00:00Z",
    "exercisesCompleted": [
      { "exerciseId": "e001", "completed": true },
      { "exerciseId": "e002", "completed": true },
      { "exerciseId": "e004", "completed": false }
    ],
    "totalTimeMinutes": 27,
    "status": "partial"
  },
  {
    "historyId": "h1002",
    "userId": "u123",
    "planId": "plan_1716613800000",
    "date": "2025-05-22T08:00:00Z",
    "exercisesCompleted": [
      { "exerciseId": "e001", "completed": true },
      { "exerciseId": "e002", "completed": true },
      { "exerciseId": "e004", "completed": true }
    ],
    "totalTimeMinutes": 38,
    "status": "completed"
  }
]
```

Супутні моделі (для повноти типів):

```ts
// src/types/models.ts

export type Exercise = {
  id: string;
  name: string;
```

```
  type: 'cardio' | 'strength' | 'core';
  reps?: number;
  sets?: number;
  duration?: number;
  requiresEquipment: boolean;
};

export type UserProfile = {
  id: string;
  age: number;
  gender: 'male' | 'female' | 'other';
  weight: number;
  goal: 'lose_weight' | 'gain_muscle' | 'maintain_fitness';
  activityLevel: 'low' | 'moderate' | 'high';
  hasEquipment: boolean;
};

export type WorkoutPlan = {
  id: string;
  userId: string;
  goal: string;
  createdAt: string;
  exercises: Exercise[];
};
```