Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ДЛЯ ТРЕКІНГУ ТРЕНУВАНЬ З РЕКОМЕНДАЦІЙНОЮ СИСТЕМОЮ НА БАЗІ ШТУЧНОГО ІНТЕЛЕКТУ

DEVELOPMENT AND RESEARCH OF A CROSS-PLATFORM APPLICATION FOR TRACKING TRAINING WITH A RECOMMENDATION SYSTEM BASED ON ARTIFICIAL INTELLIGENCE

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Алієв П. В.

Керівник:
Бойко Л. С.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Алієву Павлу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: *«Розробка та дослідження кросплатформного застосунку для трекінгу тренувань з рекомендаційною системою на базі штучного інтелекту».*
Керівник роботи: *к. т. н., доцент Бойко Лев Степанович.*
затверджені наказом закладу вищої освіти від *«29» березня 2025 року № 190/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: *04 грудня 2025 р.*
3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ*
4. Зміст розрахунково-пояснювальної записки: *аналіз вже існуючих застосунків для трекінгу тренувань та рекомендаційних систем тренувань; формування вимог для мобільного застосунку; проектування архітектури та основних модулів; реалізація компонентів кросплатформного застосунку засобами SDK Flutter/Dart; тестування застосунку та експеремантельне дослідження результативності програмного забезпечення, аналіз отриманих результатів*
5. Перелік графічного матеріалу *8 рисунків, 11 лістингів коду*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	Бойко Л. С.		
Теоретичне дослідження та практична реалізація	Бойко Л. С.		
Експериментальне дослідження системи	Бойко Л. С.		
Нормоконтроль	Повстяна Ю. С.		
Гарант ОП	Андрущак І. Є.		
Показник запозичень тексту	___%		
Академічна доброчесність	Бойко Л. С.		

7. Дата видачі завдання *«02 квітня 2025 р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну модель та архітектуру системи	01.11.2025	
4	Описати засоби розробки об'єкта проектування	19.11.2025	
5	Практична реалізація об'єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Алієв П. В.

Керівник кваліфікаційної роботи _____ Бойко Л. С.

АНОТАЦІЯ

Алієв П. В. Розробка та дослідження кросплаформного застосунку для трекінгу тренувань з рекомендаційною системою на базі штучного інтелекту. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота магістра складається з вступу, 3 розділів, висновків, списку використаних джерел, додатків.

У роботі досліджено методи та алгоритми побудови гібридної RAG-архітектури, що забезпечує безпечну, фізіологічно обґрунтовану та контекстно-залежну генерацію тренувальних планів.

Робота присвячена створенню гібридної інтелектуальної системи, що використовує LLM-технології для генерації тренувальних планів, які динамічно адаптуються до фізіологічного стану користувача (втома, травми, прогрес) та забезпечують безпеку рекомендацій.

Ключові слова: кросплатформа, трекінг тренувань, архітектура RAG (Retrieval-Augmented Generation), SDK Flutter, Golang, LLM (Large Language Models), ШІ (Штучний Інтелект), UX (User Experience).

ABSTRACT

Aliiev P. V. Development and Research of a Cross-Platform Application for Tracking Training With a Recommendation System Based on Artificial Intelligence. Manuscript.

Master's qualification work OP «Software Engineering» specialty 121 «Software Engineering». Lutsk National Technical University. Lutsk, 2025.

Master's qualification work consists of an introduction, 3 chapters, conclusions, a list of used sources, appendices.

The work investigates methods and algorithms for building a hybrid RAG architecture that provides safe, physiologically based and context-dependent generation of training plans.

The work is devoted to the creation of a hybrid intelligent system that uses LLM technologies to generate personalized training plans that dynamically adapt to the user's physiological state (fatigue, injuries, progress) and ensure the safety of recommendations.

Keywords: cross-platform, training tracking, RAG (Retrieval-Augmented Generation) architecture, Flutter SDK, Golang, LLM (Large Language Models), AI (Artificial Intelligence), UX (User Experience).

ЗМІСТ

ВСТУП

Актуальність теми. Стрімкий розвиток індустрії цифрового здоров'я (mHealth) та фітнесу виявив значні технологічні прогалини у наявних програмних продуктах. Більшість фітнес-застосунків надають користувачам або статичні, шаблонні плани, або використовують застарілі механізми колаборативної фільтрації, які не здатні забезпечити адекватну контекстуальну адаптацію. Ця проблема набуває особливої гостроти, коли йдеться про користувачів з індивідуальними фізичними обмеженнями, наприклад, хронічними травмами або певним рівнем втоми. Жорсткі алгоритми не можуть гнучко реагувати на ці фактори, що підвищує ризик травматизму та призводить до низької ефективності тренувального процесу.

Використання великих мовних моделей (LLM) пропонує рішення завдяки їхній здатності до складного міркування та генерації унікального контенту. Однак пряме застосування LLM у сфері здоров'я є високоризиковим через їхню схильність до «галюцинацій» – генерації неіснуючих або потенційно небезпечних рекомендацій. Таким чином, існує нагальна потреба у розробці гібридної інтелектуальної системи, яка б поєднала гнучкість генеративного ШІ з абсолютною надійністю та фізіологічною верифікованістю доменної бази знань. Це забезпечить користувачеві безпечне, високоперсоналізоване та ефективне керування його тренувальним процесом.

Мета дослідження – розробка та верифікація інтелектуальної системи трекінгу фізичних тренувань із застосуванням гібридної RAG-архітектури, що гарантує безпечну, фізіологічно обґрунтовану та високоперсоналізовану генерацію навантажень.

Для досягнення поставленої мети визначено такі завдання:

– провести системний аналіз існуючих рішень та методів рекомендаційних систем, обґрунтувати необхідність переходу до генеративних моделей;

– розробити архітектуру програмного комплексу, що включає клієнтську, серверну частини та інтелектуальний модуль RAG, з урахуванням вимог високої доступності та безпеки;

– здійснити програмну реалізацію всіх компонентів: асинхронного API, кросплатформного клієнта та механізму керування LLM за допомогою інженерії промптів;

– провести тестування застосунку та зробити дослідження в порівнянні з іншими програмними засобами, зібрати отримані дані та отримати фідбек (feedback) від учасників дослідження.

Оцінити ергономіку та зручність користувацького інтерфейсу (Usability) для підтвердження готовності системи до експлуатації.

Об'єкт дослідження – процес створення адаптивних рекомендаційних систем для персоналізованого планування фізичних тренувань.

Предмет дослідження – методи та алгоритми побудови гібридної RAG-архітектури, що забезпечує безпечну, фізіологічно обґрунтовану та контекстно-залежну генерацію тренувальних планів.

Практична цінність роботи полягає у створенні технологічно зрілого, функціонально повного мінімально життєздатного продукту (MVP), який може бути впроваджений у дослідну експлуатацію та має високий потенціал для комерціалізації. Основні практичні результати: впроваджена система пост-валідації та фільтрації на базі RAG забезпечує виключення потенційно небезпечних вправ при наявності травм, що значно мінімізує ризик для здоров'я користувача. Висока персоналізація та ефективність: система динамічно адаптує тренувальний об'єм та інтенсивність (зокрема, з урахуванням RPE), що підвищує ефективність тренувального процесу порівняно зі статичними шаблонами. Розроблені серверне рішення на мові програмування Golang та кросплатформний застосунок на Flutter забезпечують високу стійкість, швидкість та здатність системи до масштабування. Створення науково-технічної бази: розроблена RAG-архітектура може бути використана

як шаблон для побудови інших інтелектуальних систем у критично важливих доменних сферах

Апробація роботи. Результати кваліфікаційної роботи заслуховувались на засіданні кафедри інженерії програмного забезпечення та відображені у публікації в науковому збірнику «Студентський науковий вісник» (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Стрімкий розвиток інформаційних технологій та глобальна цифровізація суспільства докорінно змінили підходи до піклування про здоров'я та фізичну активність. Сектор мобільної охорони здоров'я (mHealth) на сьогодні є одним із найбільш динамічних сегментів ринку програмного забезпечення, що обумовлено зростанням попиту на інструменти для самостійного моніторингу фізіологічних показників та управління тренувальним процесом [1]. За даними міжнародних аналітичних агентств, обсяг світового ринку фітнес-застосунків демонструє стійку тенденцію до зростання, що актуалізує необхідність розробки нових, більш досконалих програмних продуктів [2].

Варто зазначити, що пандемія COVID-19 та подальші глобальні зміни стали каталізатором трансформації споживацької поведінки користувачів. Закриття фітнес-центрів та обмеження соціальних контактів призвели до масового переходу спортсменів-аматорів до домашніх тренувань або автономних занять у залах без залучення персональних тренерів. Якщо на етапі зародження ринку (2010-2015 рр.) застосунки виконували переважно роль пасивних реєстраторів активності (крокоміри, прості трекери калорій), то сучасний користувач висуває значно вищі вимоги. Сьогодні існує запит на комплексні екосистеми, здатні замінити живого інструктора, забезпечуючи при цьому персоналізований підхід, адаптивність навантажень та психологічну підтримку [3].

Згідно з дослідженнями поведінкових факторів, ключовою проблемою існуючих рішень залишається низький рівень утримання користувачів (Retention Rate) у довгостроковій перспективі. Статистика свідчить, що понад 60 % користувачів припиняють використання фітнес-застосунків протягом

першого місяця після завантаження [4]. Основними причинами відтоку (Churn Rate) є відсутність мотивації, одноманітність запропонованих програм, складність інтерфейсу або, що найважливіше, невідповідність алгоритмічних рекомендацій реальному фізичному стану користувача [5]. Це вказує на те, що ринок перенасичений простими рішеннями («цифровими блокнотами»), проте відчуває гострий дефіцит інтелектуальних систем, побудованих на базі алгоритмів машинного навчання (ML) та великих мовних моделей (LLM), які здатні аналізувати великі масиви даних (Big Data) про прогрес користувача та динамічно коригувати тренувальний план [6].

Для формування вичерпного переліку вимог до проектованої системи та уникнення типових архітектурних помилок, у рамках кваліфікаційної роботи проведено детальне дослідження провідних гравців ринку. Для аналізу було обрано чотири застосунки, що представляють різні концептуальні підходи до організації тренувального процесу:

– Strava (соціальна модель);

– Nike Training Club (контент-орієнтована модель);

– Strong (інструментальна модель);

– Fitbod (алгоритмічна модель).

Застосунок Strava є беззаперечним лідером у сегменті трекінгу циклічних видів спорту (біг, велоспорт, тріатлон) та яскравим прикладом успішної імплементації соціальної складової у фітнес-індустрію [7]. Основна ціннісна пропозиція (Value Proposition) продукту полягає не стільки в глибокій аналітиці біомеханічних показників, скільки в побудові глобальної спільноти атлетів.

Strava використовує агресивну модель гейміфікації через систему «сегментів» – віртуально розмічених ділянок маршруту, де користувачі змагаються за лідерство в реальному часі [7]. З точки зору проектування інтерфейсу (UI/UX), застосунок фокусується на стрічці активності (Feed), що стимулює користувача повертатися до застосунку для отримання соціального схвалення («кудосів»). Проте, при спробі екстраполювати успіх Strava на сферу

силових тренувань (Gym & Bodybuilding), виявляються суттєві функціональні та архітектурні обмеження.

Система дозволяє записати факт тренування у тренажерному залі, але інтерфейс введення даних не адаптований для детальної фіксації специфічних параметрів: робочої ваги, кількості повторень (Reps), інтенсивності за шкалою RPE (Rate of Perceived Exertion) та часу відпочинку між підходами [7].

Графіки та метрики Strava оптимізовані для відображення кардіо-показників (темп, пульсові зони, набір висоти). Для бодібілдингу критично важливими є зовсім інші метрики: об'єм тонажу (Volume Load), розрахунковий одноповторний максимум (1RM), частота навантаження на м'язову групу. Strava не надає інструментів для візуалізації цих даних, що робить її непридатною для серйозного силового тренінгу [8].

Застосунок працює виключно як реєстратор (Tracker). Він не містить алгоритмів, які б аналізували попередні силові сесії користувача для надання порад щодо відновлення або зміни інтенсивності.

Досвід Strava демонструє важливість соціального фактору, але підтверджує, що універсальні інтерфейси є неефективними для специфічних задач силового прогресу. Проектована система повинна фокусуватися на глибокій спеціалізації.

Nike Training Club представляє сегмент застосунків, що базуються на професійному медіа-контенті. Це бібліотека відео-тренувань, розроблена сертифікованими тренерами компанії Nike. Модель утримання користувача тут базується на високій якості візуального матеріалу (High Production Value) та авторитеті бренду [9].

NTC використовує педагогічний підхід «слідуй за мною» (Follow-along). Користувач обирає програму (наприклад, «Гіпертрофія за 4 тижні») і отримує статичний розклад занять. Головною перевагою такого підходу є низький поріг входу для новачків: користувачу не потрібно володіти знаннями з фізіології чи методики складання тренувань – достатньо повторювати рухи з екрана. Однак аналіз архітектури NTC виявив критичний недолік – відсутність адаптивності

(Lack of Adaptability) [10]. Програми у NTC є жорстко зафіксованими шаблонами (JSON-сценаріями), які не змінюються під впливом зовнішніх факторів.

Ігнорування принципу прогресивного перенавантаження: основою росту м'язів є постійне збільшення стресу (ваги або повторень). NTC пропонує виконувати вправи з заданою інтенсивністю, але не відстежує, чи стала ця вага залегкою для користувача. Це призводить до плато результатів.

Відсутність механізму зворотного зв'язку (Feedback Loop). Якщо користувач не може виконати певну вправу через біль у суглобі або відсутність специфічного обладнання, NTC не пропонує автоматичної альтернативи (Exercise Substitution). Користувач змушений або пропускати вправу, або переривати тренування, що руйнує цілісність процесу [12].

Проектована система повинна вирішити проблему «статичності» NTC шляхом динамічної генерації тренувань у реальному часі на основі поточного стану користувача.

Застосунок Strong є еталоном у категорії «цифрових щоденників» тренувань. На відміну від Strava чи NTC, він орієнтований на досвідчених атлетів, які самостійно планують свій тренувальний процес і потребують лише зручного інструменту для фіксації результатів [13].

UX/UI Strong вирізняється мінімалістичним інтерфейсом та високою швидкістю взаємодії. Ключовою особливістю є оптимізація введення даних: система запам'ятовує ваги з попереднього тренування, автоматично розраховує вагу розминочних підходів та має вбудований таймер відпочинку [14]. Цей застосунок ефективно вирішує проблему «незручності» паперових щоденників або Excel-таблиць на смартфоні.

Головним недоліком Strong є повна відсутність «інтелекту». Це пасивна база даних. Застосунок не аналізує ефективність програми. Якщо користувач робить помилку у плануванні (наприклад, тренує одну й ту ж групу м'язів щодня), Strong не попередить про ризик перетренованості [15]. Відсутність прогнозування. Система не використовує накопичені історичні дані для

прогнозування результатів (Predictive Analytics). Користувач залишається сам на сам зі статистикою, яку він має інтерпретувати самостійно.

Strong доводить, що зручність інтерфейсу (Usability) є критичною вимогою. Проте для масового користувача інструментального підходу недостатньо – потрібна експертна система, яка візьме на себе функцію планування.

Fitbod є найближчим аналогом проектованої системи та лідером у сегменті алгоритмічних фітнес-застосунків. Його основна функція – автоматична генерація тренувань на основі алгоритмів машинного навчання, які аналізують історію тренувань та доступне обладнання [16]. Fitbod використовує алгоритм, який відстежує ступінь відновлення м'язових груп (Muscle Recovery State). Наприклад, якщо користувач вчора тренував груди, сьогодні застосунок запропонує тренування ніг. Також алгоритм намагається реалізувати принцип прогресивного перенавантаження, поступово збільшуючи вагу снарядів [1].

Проте детальний аналіз роботи Fitbod виявляє суттєві обмеження, властиві традиційним ML-алгоритмам без використання генеративного ШІ (LLM):

- проблема «Чорної скриньки» (Black Box Problem). Алгоритм Fitbod є непрозорим. Користувач отримує рекомендацію («Жим 80 кг на 5 разів»), але не отримує пояснення, чому обрано саме ці параметри. Це знижує довіру до системи, особливо у випадках, коли рекомендація здається нелогічною [17];
- відсутність контекстуального розуміння. Fitbod оперує лише цифрами. Він не розуміє текстових запитів користувача, таких як: «Я відчуваю слабкість у попереку, заміни всі осьові навантаження на безпечні аналоги». Традиційний алгоритм не може гнучко обробити такий запит, тоді як LLM здатна зрозуміти семантику та адаптувати програму [17];
- алгоритми Fitbod часто намагаються лінійно збільшувати навантаження, що не завжди відповідає фізіології людини, яка має нелінійний характер адаптації.

Проведений компаративний аналіз дозволяє класифікувати існуючі рішення та виявити вільну нішу для розробки. Більшість продуктів на ринку зосереджені на крайнощах: або повна свобода дій без підказок (Strong), або жорсткі рамки без права на зміну (NTC).

На основі аналізу можна стверджувати, що існує незадоволений попит на системи класу «Explainable AI Coach» (Пояснюваний ШІ-тренер). Ключовою відмінністю проектованого застосунку стане інтеграція Великої Мовної Моделі (LLM), яка дозволить поєднати зручність логування (як у Strong) з глибокою персоналізацією, недоступною для звичайних алгоритмів (як у Fitbod). Це дозволить реалізувати концепцію «розумного партнера», який не просто диктує умови, а веде діалог з користувачем, пояснює логіку тренувального процесу та адаптується до найменших змін у контексті життя користувача [17]. Саме розробка такої гібридної системи, що поєднує інженерні методи мобільної розробки та новітні підходи у сфері штучного інтелекту, є метою даної кваліфікаційної роботи.

1.2 Огляд і аналіз методів та засобів розробки кросплатформного застосунку для трекінгу тренувань з рекомендаційною системою на базі штучного інтелекту для вирішення проблеми дослідження

Ефективність сучасного програмного продукту у сфері HealthTech визначається не стільки візуальною привабливістю інтерфейсу, скільки якістю та релевантністю роботи його аналітичного ядра – рекомендаційної системи. Задача автоматизованої генерації тренувального плану є класичною проблемою пошуку оптимального рішення в умовах високої невизначеності та неповноти вхідних даних [18]. На відміну від рекомендаційних систем у сфері електронної комерції (наприклад, Amazon чи Netflix), де помилка алгоритму коштує користувачеві лише втраченого часу на перегляд нецікавого контенту, у сфері фізичного виховання ціна алгоритмічної помилки є значно вищою. Некоректно підібрана вага або вправа може призвести до відсутності спортивного прогресу

(ефект плато), демотивації або навіть до фізичного травмування опорно-рухового апарату [19].

В рамках дослідження було проведено системний аналіз існуючих підходів до побудови рекомендаційних систем, оцінено їхню алгоритмічну складність, специфіку обробки даних та застосовність до задач силового тренінгу.

Історично першим та найбільш зрозумілим підходом до персоналізації є контент-орієнтована фільтрація. Головна гіпотеза цього методу базується на припущенні стабільності інтересів користувача: якщо користувачеві сподобався певний об'єкт у минулому, то йому з високою ймовірністю сподобається інший об'єкт, який має схожі характеристики або атрибути [20].

У контексті предметної області даної кваліфікаційної роботи «об'єктами» виступають фізичні вправи. Для роботи алгоритму кожна вправа повинна бути описана набором формальних ознак (так званим вектором ознак). До таких атрибутів зазвичай відносять: цільову групу м'язів (наприклад, грудні м'язи, квадріцепс), тип необхідного обладнання (штанга, гантелі, блочний тренажер), тип біомеханічного руху (штовхальний, тяговий) та рівень технічної складності.

Сутність роботи алгоритму полягає у створенні «Профілю користувача», який формується на основі його попередніх дій. Якщо спортсмен часто виконує жим штанги лежачи та відтискання на брусах, система формує профіль, у якому домінують атрибути «грудні м'язи» та «штовхальні рухи». При пошуку рекомендацій алгоритм порівнює вектори доступних у базі вправ із вектором профілю користувача.

Для визначення ступеня схожості між профілем та потенційною вправою найчастіше використовується математичний метод косинусної подібності. Говорячи простою мовою, система будує два багатовимірні вектори в просторі ознак і вимірює кут між ними. Чим менший кут, тим більше вправа відповідає інтересам користувача. Якщо косинус кута наближається до одиниці, вправа вважається ідеальною рекомендацією [21].

Головною перевагою контентного підходу є його незалежність від інших користувачів. Це означає, що система може рекомендувати нову вправу, щойно вона була додана в базу даних, базуючись лише на її описі. Також цей метод забезпечує високу прозорість (Interpretability): система завжди може «пояснити» свій вибір, наприклад: «Ми рекомендуємо вам жим гантелей, тому що це базова вправа на груди, схожа на ті, що ви виконували раніше» [22]. Критичним недоліком для фітнесу є так звана проблема «надмірної спеціалізації». Алгоритм схильний замикати користувача у «інформаційній бульбашці», пропонуючи лише те, що він вже робив. Якщо користувач ігнорує тренування ніг, система, базуючись на історії, перестане пропонувати вправи на ноги, що суперечить методиці гармонійного фізичного розвитку.

Альтернативним підходом, що дозволяє подолати обмеження контентної фільтрації, є колаборативна фільтрація. Цей метод базується на аналізі колективної поведінки спільноти. Основна ідея методу формулюється так: «користувачам, які мали схожі вподобання в минулому, сподобаються схожі речі і в майбутньому». Цей підхід не вимагає глибокого розуміння природи вправ (біомеханіки), а оперує лише матрицею взаємодій, де рядками є користувачі, а стовпцями – вправи [23].

Існує два основні різновиди цього методу:

– фільтрація на основі користувачів (User-based). Алгоритм шукає у базі даних «сусідів» – людей, чия історія тренувань максимально корелює з історією поточного користувача. Якщо знайдений «сусід» успішно прогресував, виконуючи певну програму, ця ж програма буде рекомендована поточному користувачу;

– фільтрація на основі об’єктів (Item-based). Алгоритм шукає закономірності у спільному використанні вправ. Наприклад, якщо статистика показує, що 90 % атлетів, які роблять присідання, у тому ж тренуванні роблять розгинання ніг, система встановить сильний асоціативний зв’язок між цими вправами. Для реалізації цього методу часто використовують техніки зменшення розмірності даних, такі як матрична факторизація. Цей

математичний прийом дозволяє виявити приховані (латентні) фактори, що впливають на вибір користувачів, навіть якщо ці фактори неможливо описати словами явно. Попри високу точність у таких сферах як кіноіндустрія, у фітнесі колаборативна фільтрація стикається з серйозними викликами.

Проблема «Холодного старту» (Cold Start Problem) – новий користувач, який тільки встановив застосунок, не має історії тренувань. Відповідно, алгоритм не може знайти йому «сусідів» і не може надати жодної рекомендації. Для комерційного продукту це критично, оскільки саме на етапі першого запуску вирішується, чи залишиться користувач у застосунку [24].

У базі може бути тисячі вправ, а середньостатистичний користувач використовує лише 20-30 з них. Матриця взаємодій виявляється майже порожньою, що ускладнює статистичний аналіз. Колаборативний метод може порадити новачкові технічно складну вправу (наприклад, ривок штанги) лише тому, що її часто виконують досвідчені користувачі зі схожими антропометричними даними.

Фізичні тренування – це процес, що розгортається у часі. Результат сьогоднішнього тренування безпосередньо залежить від того, що спортсмен робив вчора і позавчора (накопичена втома, відновлення). Класичні методи, описані вище, часто ігнорують цю часову залежність, розглядаючи кожне тренування як ізольовану подію.

Для вирішення цієї проблеми застосовуються рекурентні нейронні мережі (RNN), а зокрема їх вдосконалена архітектура – мережі з довгою короткостроковою пам'яттю (LSTM). Головна особливість LSTM полягає в наявності «механізму пам'яті», який дозволяє моделі запам'ятовувати послідовність подій. Мережа аналізує ланцюжок попередніх тренувань (Time Series Data) і намагається спрогнозувати наступний крок.

У контексті кваліфікаційної роботи це дозволяє реалізувати прогнозування навантаження. На вхід нейромережі подається послідовність ваг, піднятих користувачем за останній місяць. На виході модель прогнозує оптимальну вагу для наступного заняття. Це дозволяє реалізувати принцип

нелінійної періодизації навантажень, що є значно ефективнішим за просте лінійне додавання ваги [25]. Найбільш перспективним напрямком, обраним як основний для реалізації у даній роботі, є використання великих мовних моделей (Large Language Models), таких як GPT-4 або Claude. Цей підхід фундаментально відрізняється від усіх попередніх, оскільки переходить від числового аналізу до семантичного розуміння контексту [26]. Традиційні ML-моделі працюють з жорсткими даними (числа, категорії). Вони не здатні зрозуміти нюанси людської мови. Натомість LLM дозволяє реалізувати концепцію «ШІ-тренера», з яким можна спілкуватися природною мовою.

Архітектура RAG (Retrieval-Augmented Generation). Для усунення головного недоліку LLM – схильності до вигадування фактів (галюцинацій) – у роботі пропонується використати архітектурний патерн RAG (Генерація з доповненим пошуком). Логіка роботи RAG у проектованій системі виглядає наступним чином: Користувач робить запит (наприклад, «У мене болить коліно, склади тренування на ноги без осьового навантаження»).

Система спочатку виконує пошук у власній базі знань (Knowledge Base) перевірених вправ, відфільтровуючи ті, що навантажують колінний суглоб. Знайдений контекст (список безпечних вправ та історія ваг користувача) передається у промпт (інструкцію) для LLM. LLM генерує фінальну відповідь, формуючи збалансовану програму та надаючи текстове пояснення, чому обрано саме ці вправи.

Такий підхід поєднує точність бази даних з гнучкістю генеративного інтелекту. Це дозволяє вирішити проблему «чорної скриньки» (Black Box), оскільки модель може аргументувати кожну свою рекомендацію, що підвищує довіру користувача до системи [27].

Перевагами LLM-підходу є мультимодальність – можливість врахування неструктурованих даних (коментарі користувача про самопочуття, текстовий опис травм), гіпер-персоналізація – модель може адаптувати стиль спілкування (суворий тренер або підтримуючий ментор) залежно від психотипу користувача.

Вирішення проблеми холодного старту: навіть без історії тренувань, LLM може провести детальне інтерв'ю (анкетування) нового користувача і на основі його текстових відповідей згенерувати релевантну стартову програму. Проведений у першому розділі аналіз дозволив сформулювати концепцію майбутньої розробки. Ринок фітнес-застосунків потребує переходу від простих трекерів до інтелектуальних систем адаптивного тренінгу. Встановлено, що класичні методи фільтрації (контентна та колаборативна) мають суттєві обмеження для сфери спорту, зокрема проблему «холодного старту» та нездатність враховувати складний фізіологічний контекст. Використання нейронних мереж LSTM дозволяє працювати з часовими рядами, але є складним у реалізації та вимогливим до обчислювальних ресурсів. Оптимальним рішенням визначено використання гібридного підходу на базі LLM (Large Language Models) з архітектурою RAG. Це дозволить створити кросплатформний застосунок, який забезпечує не лише підбір навантажень, але й зрозумілу комунікацію з користувачем, імітуючи роботу реального персонального тренера.

Обраний технологічний стек (Flutter для клієнтської частини, Golang для бекенду та OpenAI API для інтелектуального модуля) повністю відповідає поставленим задачам та сучасним вимогам індустрії розробки програмного забезпечення [28].

1.3 Постановка завдання на кваліфікаційну роботу магістра

Метою кваліфікаційної роботи магістра є розробка кросплатформного застосунку для трекінгу тренувань з рекомендаційною системою на базі штучного інтелекту, його та перевірка шляхом створення баз даних користувачів.

Для досягнення цілей кваліфікаційної роботи магістра потрібно знайти шляхи реалізації таких завдань:

– провести системний аналіз існуючих рішень та методів рекомендаційних систем, обґрунтувати необхідність переходу до генеративних моделей;

– розробити архітектуру програмного комплексу, що включає клієнтську, серверну частини та інтелектуальний модуль RAG, з урахуванням вимог високої доступності та безпеки;

– здійснити програмну реалізацію всіх компонентів: асинхронного API, кросплатформного клієнта та механізму керування LLM за допомогою інженерії промптів;

– провести тестування застосунку та зробити дослідження в порівнянні з іншими програмними засобами, зібрати отримані дані та отримати фідбек (feedback) від учасників дослідження.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ДЛЯ ТРЕКІНГУ ТРЕНУВАНЬ З РЕКОМЕНДАЦІЙНОЮ СИСТЕМОЮ НА БАЗІ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Вибір технологічного стеку для реалізації кваліфікаційної роботи є критично важливим етапом, що визначає не лише швидкість розробки (Time-to-Market), але й життєвий цикл програмного продукту, його масштабованість, продуктивність та вартість підтримки. У контексті розробки інтелектуальної системи трекінгу тренувань вибір інструментів базувався на багатокритеріальному аналізі, що враховує специфічні вимоги до мобільних застосунків класу mHealth та необхідність інтеграції з алгоритмами штучного інтелекту [29].

Згідно з міжнародним стандартом якості програмного забезпечення ISO/IEC 25010, основними критеріями вибору засобів розробки були визначені:

– здатність системи забезпечувати плавний рендеринг інтерфейсу (60 fps) та швидку обробку запитів;

– можливість роботи на двох домінуючих мобільних платформах (iOS та Android) з єдиною кодовою базою;

– читабельність коду, наявність типізації та розвиненої екосистеми бібліотек;

– наявність нативних засобів для роботи з ML-моделями та великими даними.

Першим етапом проектування став вибір стратегії розробки клієнтської частини. Сучасна індустрія пропонує два діаметрально протилежних підходи: нативна розробка (Native) та кросплатформна (Cross-platform). Нативний підхід передбачає створення двох окремих застосунків: одного для iOS (Swift або

Objective-C) та іншого для Android (Kotlin або Java). Перевагами є максимальна продуктивність, прямий доступ до апаратних засобів (Bluetooth, акселерометр), повна відповідність гайдлайнам (HIG для iOS та Material Design для Android). Недоліки включають необхідність подвійного бюджету та часу на розробку, що робить цей підхід нераціональним для одного розробника.

Кросплатформний підхід дозволяє писати код один раз і запускати його на обох платформах. Переваги: скорочення часу розробки на 40-50 %, єдина логіка роботи, спрощене тестування. Раніше кросплатформні фреймворки (Ionic, PhoneGap) використовували WebView, що призводило до низької продуктивності. Сучасні рішення (Flutter, React Native) усунули цей недолік завдяки компіляції в нативний код або ефективним мостам. Для реалізації проекту обрано кросплатформний підхід, що дозволяє зосередитися на розробці інтелектуальної складової (AI), а не на підтримці двох кодових баз.

На ринку кросплатформної розробки домінують два фреймворки: React Native (від Meta) та Flutter (від Google). React Native використовує мову JavaScript і працює за принципом «моста» (JS Bridge), що забезпечує високу продуктивність, але може призводити до «вузьких місць» (UI jank) при складних графічних елементах. Мова JavaScript є динамічно типізованою, що підвищує ризик помилок під час виконання.

Flutter використовує мову Dart і власний графічний рушій Skia, що забезпечує нативну продуктивність і стабільні 60-120 FPS навіть при складних анімаціях. Dart є суворо типізованою мовою, що дозволяє уникнути помилок на етапі компіляції, а бібліотека віджетів Flutter забезпечує швидке створення адаптивного інтерфейсу. У порівнянні з React Native, Flutter дає кращу продуктивність і стабільність інтерфейсу, тому для реалізації обрано Flutter.

Для реалізації серверної частини було обрано Golang. Ключовою перевагою Go є вбудована підтримка конкурентності через горутини та канали. Горутини являють собою «легкі потоки» виконання, які споживають лише кілька кілобайт оперативної пам'яті. Це дозволяє одночасно запускати тисячі або навіть мільйони горутин на одному сервері, що критично важливо для

обробки великої кількості одночасних запитів. Канали забезпечують безпечний механізм комунікації між горутинами, дозволяючи у доволі простий спосіб реалізовувати складні паттерни паралельної обробки без типових проблем таких як, наприклад, гонка даних.

Продуктивність Go наближається до рівня компільованих мов як C++ завдяки компіляції безпосередньо в машинний код, але при цьому мова залишається значно простішою у використанні. Статична типізація та потужна система типів виявляють більшість помилок на етапі компіляції, що суттєво зменшує кількість runtime-помило. Вбудований збирач сміття (garbage collector) ефективно керує пам'яттю без необхідності ручного контролю, як у C++, але з мінімальними паузами.

Щодо бази даних, для зберігання тренувальних даних було обрано PostgreSQL. Дана система управління базами даних (СУБД) гарантує атомарність, консистентність, ізольованість та довговічність транзакцій навіть за високих навантажень або збоїв системи. Це означає, що дані ніколи не втрачаються та не пошкоджуються у результаті незавершених операцій, що критично важливо для будь-якого рішення, а для сфери healthcare – зокрема.

PostgreSQL підтримує надзвичайно широкий спектр типів даних, що виходить далеко за межі стандартних реляційних СУБД. Окрім звичайних числових, текстових та часових типів, дана СУБД нативно працює з JSON та JSONB, що дозволяє ефективно зберігати та індексувати документо-орієнтовані дані без необхідності використовувати окреме NoSQL рішення. Підтримка масивів, геометричних типів, мережевих адрес, UUID та можливість створювати власні типи даних робить PostgreSQL універсальним інструментом для найрізноманітніших сценаріїв використання.

Щодо вибору моделі для рекомендаційної системи, було вирішено використовувати хмарне API для LLM (OpenAI GPT-4), оскільки воно забезпечує високу якість генерації тексту і підтримку великого вікна контексту. Для оркестрації компонентів системи обрано LangChain, що забезпечує зручну інтеграцію з векторним пошуком і управління пам'яттю діалогу.

Функціональні вимоги до системи включають генерацію тренувальних програм на основі аналізу історичних даних користувача. Система повинна автоматично коригувати інтенсивність тренувань, дотримуючись принципу прогресивного навантаження (Progressive Overload) та аналізуючи щонайменше 10-15 попередніх сесій.

Система також повинна бути адаптивною залежно від рівня досвіду користувача: для новачків вона має брати на себе управління вибором вправ, для досвідчених атлетів – пропонувати розширену аналітику та можливість коригування тренувань. Важливою вимогою є також збір даних про стан користувача, зокрема антропометричних показників та історії травм, що буде використовуватись для налаштування тренувальних планів.

Нефункціональні вимоги включають високу продуктивність і швидкість відповіді системи. Для забезпечення роботи в умовах обмеженого інтернет-з'єднання передбачено офлайн-режим, а також застосування механізмів асинхронної обробки запитів для забезпечення чутливості інтерфейсу.

2.2 Практична реалізація об’єкта проектування

Мобільний застосунок реалізовано на SDK Flutter з використанням архітектурного шаблону Business Logic Component (BLoC), що передбачає структурований підхід до організації коду, за якого бізнес-логіка чітко відокремлена від інтерфейсу користувача. Цей процес включає кілька послідовних етапів, кожен із яких має свої особливості та вимоги до реалізації.

Початковим етапом є підготовка проєкту та встановлення необхідних залежностей. Після створення нового Flutter-проєкту у файлі pubspec.yaml вказується залежність flutter_bloc, що надає зручні засоби для реалізації патерну BloC.

Лістинг 2.1 – Файл залежностей Flutter-проєкту pubspec.yaml

```
.................
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  flutter_bloc: ^7.0.1
.................
```

Кінець лістингу 2.1

Наступним кроком є проєктування архітектури мобільного застосунку з урахуванням принципів розділення відповідальностей. Типова структура проєкту передбачає створення окремих каталогів для різних рівнів абстракції. Директорія для моделей даних містить класи, що описують сутності предметної області з їхніми атрибутами та методами. Директорія для блоків включає логіку обробки подій та керування станом для кожної функціональної області застосунку. Репозиторії інкапсулюють деталі отримання та збереження даних, приховуючи від блоків конкретні джерела інформації. Віджети інтерфейсу організовані окремо і відповідають виключно за візуальне представлення даних та реагування на взаємодію користувача.

Розробка конкретного функціоналу розпочинається зі створення класів подій, які представляють можливі дії користувача або системні події в межах певної функціональної області. Кожна подія є незмінним об'єктом, що містить усю необхідну інформацію для її обробки. Паралельно визначаються класи станів, які описують різні можливі стани інтерфейсу користувача. Наприклад, типовими станами можуть бути початковий стан, стан завантаження даних, стан успішного отримання даних або стан помилки. Використання окремих класів для кожного стану дозволяє точно контролювати поведінку інтерфейсу в різних ситуаціях та спрощує тестування логіки застосунку.

На рисунку 2.1 зображено екрани мобільного застосунку, які бачить користувач під час першого запуску застосунку.

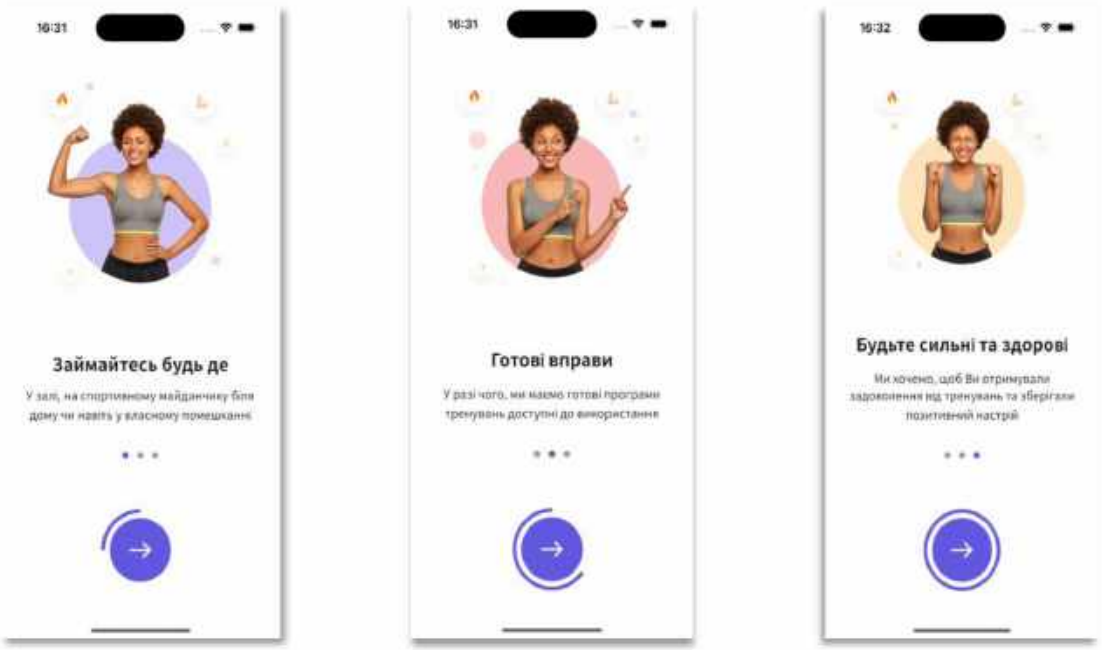


Рисунок 2.1 – Початкові екрани мобільного застосунку

Центральним компонентом архітектури є власне Bloc-клас, який координує взаємодію між подіями та станами. У конструкторі блоку реєструються обробники для кожного типу події за допомогою методу on, який приймає два параметри: тип події та функцію-обробник. Обробник отримує конкретну подію та емітер станів, через який він може послідовно випромінювати нові стани у відповідь на подію. Всередині обробників виконуються необхідні операції, такі як валідація даних, звернення до репозиторіїв для отримання або збереження інформації, обробка помилок та формування результуючих станів. Обробка асинхронних операцій виконується за допомогою конструкцій async/await мови Dart.

При використанні патерну BLoC, важливим є організація такої структури директорій та файлів, які дозволять швидко та легко орієнтуватися у коді для реалізації нових функцій або виправленні знайдених помилок. Так, на рисунку 2.2 наведено типову організацію функціонального модуля у Flutter-застосунку з використанням патерну BloC.

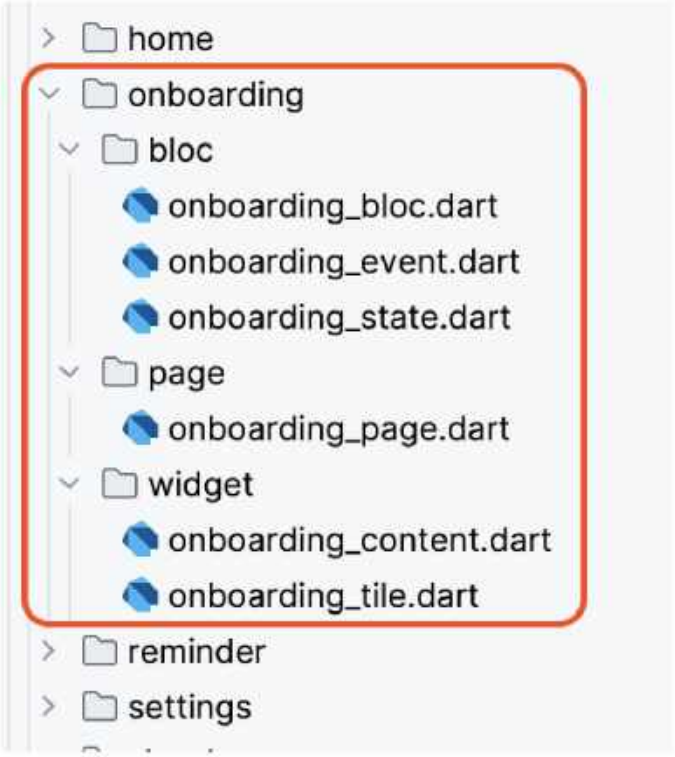


Рисунок 2.2 – Структура файлів при використанні архітектурного шаблону BLoC для екранів онбордингу користувача

Файл onboarding_bloc.dart є центральним елементом архітектури BLoC і містить класс OnboardingBloc, що відповідає за обробку дій користувача (OnboardingEvent) та менеджмент стану процесу онбордингу (OnboardingState), що включає в себе координацію переходів між слайдами/екранами та збереження прогресу проходження онбордингу (ліст. 2.2).

Лістинг 2.2 – Клас OnboardingBloc у файлі onboarding_bloc.dart

```
class OnboardingBloc extends Bloc<OnboardingEvent, OnboardingState> {
  OnboardingBloc() : super(OnboardingInitial());

  int pageIndex = 0;

  final pageController = PageController(initialPage: 0);

  @override
  Stream<OnboardingState> mapEventToState(
    OnboardingEvent event,
  ) async* {
```

```
    if (event is PageChangedEvent) {
      if (pageIndex == 2) {
        yield NextScreenState();
        return;
      }
      pageIndex += 1;

      pageController.animateToPage(
        pageIndex,
        duration: Duration(milliseconds: 500),
        curve: Curves.ease,
      );

      yield PageChangedState(counter: pageIndex);
    } else if (event is PageSwipedEvent) {
      pageIndex = event.index;
      yield PageChangedState(counter: pageIndex);
    }
  }
}
```

Кінець лістингу 2.2

Файл onboarding_event.dart використовується для опису усіх можливих подій, які можуть відбутися в процесі онбордингу. У нашому випадку це дві події: PageChangedEvent та PageSwipedEvent які відбуваються при натиску на кнопку «далі» або при виконанні жесту «свайп» користувачем відповідно (ліст. 2.3).

Лістинг 2.3 – Файл onboarding_event.dart

```
part of 'onboarding_bloc.dart';

@immutable
abstract class OnboardingEvent {}

class PageChangedEvent extends OnboardingEvent {}

class PageSwipedEvent extends OnboardingEvent {
  final int index;

  PageSwipedEvent({required this.index});
}
```

Кінець лістингу 2.3

У файлі onboarding_state.dart описуються усі можливі стани, в яких може перебувати процес онбордингу (ліст. 2.4):

- початковий стан (клас OnboardingInitial)
- стан зміни поточної сторінки (клас PageChangedState)
- перехід на наступний екран (клас NextScreenState)

Лістинг 2.4 – Файл onboarding_state.dart

```
part of 'onboarding_bloc.dart';

@immutable
abstract class OnboardingState {}

class OnboardingInitial extends OnboardingState {}

class PageChangedState extends OnboardingState {
  final int counter;

  PageChangedState({
    required this.counter,
  });
}

class NextScreenState extends OnboardingState {}
```

Кінець лістингу 2.4

Інтеграція «блоку» з візуальною (UI) частиною застосунку здійснюється через спеціалізовані віджети пакету flutter_bloc. BlocProvider використовується для створення екземпляру відповідного класу та надання доступу до нього всім дочірнім віджетам через контекст. Цей віджет автоматично керує життєвим циклом блоку, гарантуючи коректне вивільнення ресурсів при видаленні з дерева віджетів. Для випадків, коли один віджет потребує доступу до декількох блоків одночасно, застосовується MultiBlocProvider, який дозволяє організувати ієрархічну структуру провайдерів.

Відображення інтерфейсу на основі поточного стану відбувається за допомогою віджету BlocBuilder, який автоматично перебудовує своє дочірнє дерево віджетів при зміні стану, викликаючи функцію builder з актуальним

станом. У функції builder аналізується тип поточного стану та повертається відповідна структура віджетів. Для ситуацій, коли потрібно виконати одноразову дію у відповідь на зміну стану без перебудови інтерфейсу, використовується BlocListener. Це може бути показ діалогового вікна, перехід на інший екран або відображення сповіщення. Коли необхідна комбінація обох підходів, застосовується BlocConsumer, який об'єднує функціональність builder та listener в одному віджеті. У лістингу наведено вихідний код класу OnboardingPage з файлу onboarding_page.dart, який відповідає за формування та відображення інтерфейсу користувача для екранів онбордингу (ліст. 2.5).

Лістинг 2.5 – Клас OnboardingPage з файлу onboarding_page.dart

```
class OnboardingPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _buildBody(context),
    );
  }
  BlocProvider<OnboardingBloc> _buildBody(BuildContext context) {
    return BlocProvider<OnboardingBloc>(
      create: (BuildContext context) => OnboardingBloc(),
      child: BlocConsumer<OnboardingBloc, OnboardingState>(
        listenWhen: (_, currState) => currState is NextScreenState,
        listener: (context, state) {
          Navigator.of(context).pushReplacement(
            MaterialPageRoute(
              builder: (_) {
                return SignUpPage();
              },
            ),
          );
        },
        buildWhen: (_, currState) => currState is OnboardingInitial,
        builder: (context, state) {
          return OnboardingContent();
        },
      ),
    );
  }
}
```

Кінець лістингу 2.5

Решта екранів мобільного застосунку реалізовані аналогічним чином – з використанням патерну BLoC, що забезпечує єдиний архітектурний підхід до розробки та підтримки проєкту. Така уніфікована структура дозволяє швидко орієнтуватися в кодовій базі, оскільки кожен екран слідує однаковим принципам організації бізнес-логіки та управління станом. Це особливо важливо при масштабуванні застосунку, коли команда розробників може змінюватися та розширюватися.

Застосування BLoC на всіх екранах забезпечує чітке розділення відповідальності між компонентами застосунку. Так, бізнес-логіка повністю відокремлена від презентаційного шару, що означає, що UI-компоненти відповідають виключно за відображення даних та реагування на дії користувача, тоді як BLoC-и обробляють всі обчислення, валідацію даних (рис. 2.3) та взаємодію з зовнішніми сервісами (рис. 2.4). Така архітектура значно спрощує тестування, адже бізнес-логіку можна перевіряти окремо від інтерфейсу користувача, без необхідності рендерингу віджетів або симуляції взаємодії з екраном.

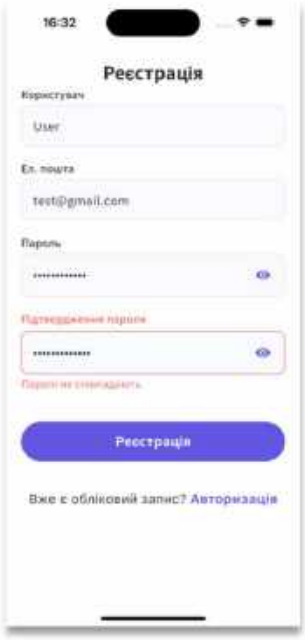


Рисунок 2.3 – Валідація форми введення даних

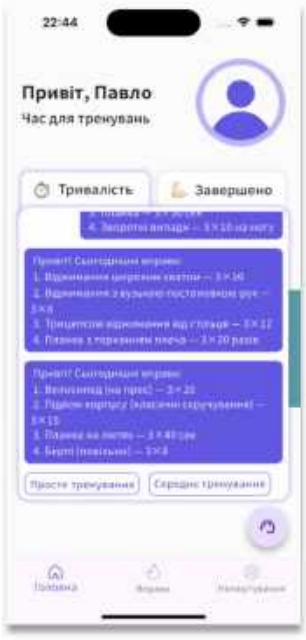


Рисунок 2.4 – Взаємодія з бекендом для отримання програми тренувань

Для розробки бекенд частини було використано мікросервісну архітектуру, яка передбачає розділення функціональності застосунку на незалежні сервіси. Даний підхід забезпечує низку переваг, серед яких можна виділити такі як можливість незалежного масштабування окремих компонентів системи (сервісів), спрощення процесу розробки та тестування, а також підвищення загальної надійності рішення завдяки ізоляції потенційних збоїв у межах окремих сервісів.

Проте слід зазначити, що повноцінна підтримка мікросервісної архітектури на початкових етапах розробки проєкту є складним завданням, яке вимагає значних ресурсів та часу. Необхідність налаштування інфраструктури для міжсервісної комунікації, впровадження механізмів виявлення сервісів, організації централізованого логування та моніторингу, а також забезпечення узгодженості даних між сервісами створює додаткову складність у процесі розробки яка є зайвою на етапі створення MVP. Саме тому було прийнято рішення про поетапне впровадження мікросервісної архітектури і на поточній стадії розробки бекенд рішення складається з двох основних сервісів, кожен з яких відповідає за чітко визначену область функціональності. Такий розподіл дозволяє зберегти основні переваги мікросервісного підходу, водночас уникаючи надмірної складності, характерної для систем з великою кількістю сервісів.

Перший сервіс, називається core nf є центральним компонентом системи і містить основний бізнес-логіку застосунку. Цей сервіс відповідає за обробку користувацьких запитів, взаємодію з базою даних, реалізацію бізнес-правил та забезпечення основної функціональності застосунку. Сервіс core реалізує усі критичні операції, включаючи аутентифікацію та авторизацію користувачів, управління даними та роботу з LLM. Розміщення всієї основної функціональності (бізнес-логіки) в межах одного сервісу на початковому етапі дозволяє швидше реалізовувати нові можливості та тестувати їх, оскільки немає необхідності координувати зміни між різними сервісами.

Наступний сервіс – notification. Його винесено в окремий компонент системи через його специфічну функціональність та потенційно високі вимоги до ресурсів. Цей сервіс спеціалізується виключно на відправці електронних листів користувачам системи. Відокремлення функціональності повідомлень від основного сервісу має декілька важливих переваг. По-перше, це дозволяє незалежно масштабувати сервіс відправки листів у разі збільшення навантаження, не впливаючи на роботу основного функціоналу. По-друге, ізоляція сервісу повідомлень забезпечує більшу надійність системи в цілому, оскільки потенційні проблеми з відправкою електронної пошти не призведуть до збоїв в роботі основних функцій за стосунку (рис. 2.5).

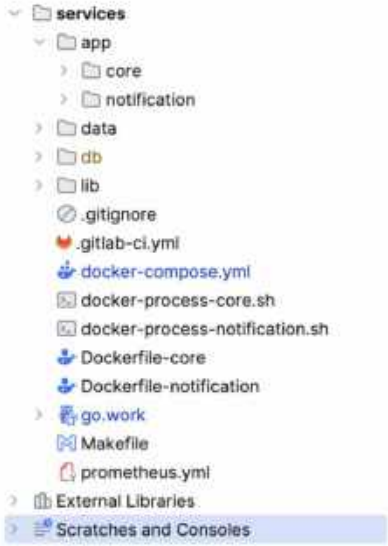
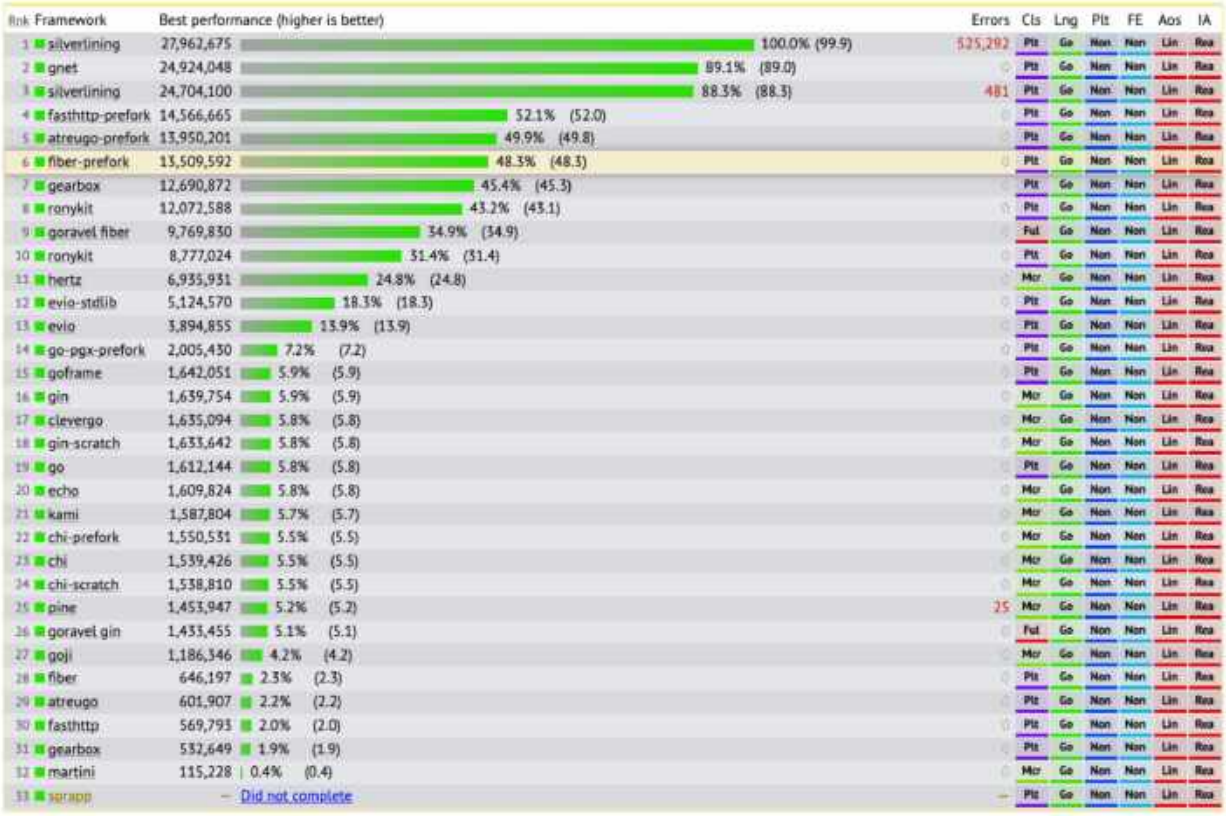


Рисунок 2.5 – Структура бекенд застосунку

У основному сервісі – core – реалізовано HTTP API, за допомогою fiber.

Fiber – це сучасний фреймворк для створення веб-застосунків на мові Golang, що поєднує високу продуктивність зі зручністю використання. Основною архітектурною особливістю Fiber є використання бібліотеки fasthttp замість стандартного пакету net/http, що забезпечує суттєве підвищення швидкості обробки HTTP-запитів (рис. 2.6). Результати бенчмарків демонструють, що Fiber здатний обробляти значно більшу кількість запитів за

секунду порівняно з іншими популярними Go-фреймворками, що робить його оптимальним вибором для високонавантажених систем.



Rnk	Framework	Best performance (higher is better)		Errors	Cls	Lng	Plt	FE	Aos	IA
1	silverlining	27,962,675	100.0% (99.9)	525,292	Plt	Go	Non	Non	Lin	Rea
2	gnet	24,924,048	89.1% (89.0)		Plt	Go	Non	Non	Lin	Rea
3	silverlining	24,704,100	88.3% (88.3)	481	Plt	Go	Non	Non	Lin	Rea
4	fasthttp-prefork	14,566,665	52.1% (52.0)		Plt	Go	Non	Non	Lin	Rea
5	atreugo-prefork	13,950,201	49.9% (49.8)		Plt	Go	Non	Non	Lin	Rea
6	fiber-prefork	13,509,592	48.3% (48.3)		Plt	Go	Non	Non	Lin	Rea
7	gearbox	12,690,872	45.4% (45.3)		Plt	Go	Non	Non	Lin	Rea
8	ronykit	12,072,588	43.2% (43.1)		Plt	Go	Non	Non	Lin	Rea
9	goravel fiber	9,769,830	34.9% (34.9)		Ful	Go	Non	Non	Lin	Rea
10	ronykit	8,777,024	31.4% (31.4)		Plt	Go	Non	Non	Lin	Rea
11	hertz	6,935,931	24.8% (24.8)		Mcr	Go	Non	Non	Lin	Rea
12	evio-stdlib	5,124,570	18.3% (18.3)		Plt	Go	Non	Non	Lin	Rea
13	evio	3,894,855	13.9% (13.9)		Plt	Go	Non	Non	Lin	Rea
14	go-pgx-prefork	2,005,430	7.2% (7.2)		Plt	Go	Non	Non	Lin	Rea
15	goframe	1,642,051	5.9% (5.9)		Plt	Go	Non	Non	Lin	Rea
16	gin	1,639,754	5.9% (5.9)		Mcr	Go	Non	Non	Lin	Rea
17	clevergo	1,635,094	5.8% (5.8)		Mcr	Go	Non	Non	Lin	Rea
18	gin-scratch	1,633,642	5.8% (5.8)		Mcr	Go	Non	Non	Lin	Rea
19	go	1,612,144	5.8% (5.8)		Plt	Go	Non	Non	Lin	Rea
20	echo	1,609,824	5.8% (5.8)		Mcr	Go	Non	Non	Lin	Rea
21	kami	1,587,804	5.7% (5.7)		Mcr	Go	Non	Non	Lin	Rea
22	chi-prefork	1,550,531	5.5% (5.5)		Mcr	Go	Non	Non	Lin	Rea
23	chi	1,539,426	5.5% (5.5)		Mcr	Go	Non	Non	Lin	Rea
24	chi-scratch	1,538,810	5.5% (5.5)		Mcr	Go	Non	Non	Lin	Rea
25	pine	1,453,947	5.2% (5.2)	25	Mcr	Go	Non	Non	Lin	Rea
26	goravel gin	1,433,455	5.1% (5.1)		Ful	Go	Non	Non	Lin	Rea
27	goji	1,186,346	4.2% (4.2)		Mcr	Go	Non	Non	Lin	Rea
28	fiber	646,197	2.3% (2.3)		Plt	Go	Non	Non	Lin	Rea
29	atreugo	601,907	2.2% (2.2)		Plt	Go	Non	Non	Lin	Rea
30	fasthttp	569,793	2.0% (2.0)		Plt	Go	Non	Non	Lin	Rea
31	gearbox	532,649	1.9% (1.9)		Plt	Go	Non	Non	Lin	Rea
32	martini	115,228	0.4% (0.4)		Mcr	Go	Non	Non	Lin	Rea
33	sprapp	–	Did not complete	–	Plt	Go	Non	Non	Lin	Rea

Рисунок 2.6 – Результати виконання Plaintext тесту. Фреймворк Fiber на 6-ій позиції

Також важливою перевагою Fiber є його інтуїтивний та лаконічний синтаксис, який свідомо розроблявся схожим до Express.js. Це рішення значно знижує поріг входу для розробників, які мають досвід роботи з JavaScript екосистемою, дозволяючи їм швидко адаптуватися до Golang. При цьому фреймворк зберігає усі переваги go як компільованої мови зі строгою типізацією, включаючи ефективне керування пам'яттю, вбудовану підтримку конкурентності через горутини та високу безпеку типів на етапі компіляції.

Fiber надає комплексну екосистему вбудованих middleware компонентів, що охоплюють типові потреби веб-розробки. Серед них можна виділити механізми логування запитів, обробку CORS політик, стиснення відповідей за допомогою gzip або brotli, централізовану обробку помилок, валідацію вхідних

даних та управління сесіями. Наявність цих готових рішень дозволяє розробникам зосередитися на бізнес-логіці застосунку, а не на створенні базової інфраструктури, що прискорює розробку та зменшує кількість потенційних помилок. Проте, не зважаючи на значний набір існуючих middleware, можливість розробки та використання власних рішень. Так, у лістингу 2.6 наведено реалізацію middleware для перевірки токену доступу користувача під час виконання HTTP-запиту.

Лістинг 2.6 – Middleware для перевірки валідності токену доступу

```
func CheckAuthMiddleware(
	authService *services.AuthService,
) fiber.Handler {
	return func(c *fiber.Ctx) error {
		headers := c.GetReqHeaders()
		ctx := c.Context()

		token, isOk := headers["Authorization"]
		if !isOk || len(token) != 1 {
			return c.Status(http.StatusUnauthorized).
				JSON(api.NewUnauthorizedErrorResponse())
		}

		isValid, user, userDataContract, err := authService.
			IsValidAccessToken(ctx, token[0], true)
		if err != nil || !isValid {
			return c.Status(http.StatusUnauthorized).
				JSON(api.NewUnauthorizedErrorResponse())
		}

		c.Locals(config.User, user)
		c.Locals(config.UserData, userDataContract)
		return c.Next()
	}
}
```

Кінець лістингу 2.6

Приклад підключення (використання) middleware CheckAuthMiddleware наведено у лістингу 2.7.

Лістинг 2.7 – Приклад використання middleware CheckAuthMiddleware

```
...
api := app.Group("/api", logger.New())
...
protectedHandler := api.Use(
	middlewares.CheckAuthMiddleware(s.Auth),
	middlewares.CheckPermissionMiddleware(s.AccessControl),
)
...
```

Кінець лістингу 2.7

Як можна було поміти з лістингу 2.6, API повертає відповідь у форматі JSON. Основною перевагою даного формату є його універсальність та широка підтримка практично усіма сучасними мовами програмування та платформами. Цей формат став де-факто стандартом для веб-API завдяки своїй простоті, читабельності та ефективності, що робить його природним вибором для сучасних розподілених систем.

Структура JSON наочно відображає ієрархічні та вкладені дані через об'єкти та масиви, що дозволяє представляти складні моделі даних у зрозумілій формі. Можливість вкладення об'єктів один в одного на довільну глибину надає гнучкість у моделюванні предметної області. При цьому простота базових типів даних JSON, які включають рядки, числа, булеві значення, null, об'єкти та масиви, робить формат легким для розуміння та використання навіть для початківців.

Проте при роботі навіть з таким простим форматом як JSON, варто пам'ятати, що програми працюють з даними у вигляді структур, об'єктів та змінних у пам'яті, які мають специфічне представлення для конкретної мови програмування, а мережеві протоколи оперують послідовностями байтів або текстом. Для того щоб перетворити внутрішні структури даних, наприклад, у формат JSON, необхідно виконати операцію серіалізації даних (і десеріалізації для зворотного процесу).

Для серіалізації та десеріалізації структур даних Golang у формат JSON було обрано пакет easyjson, забезпечує значне пришвидшення процесу роботи з даними у порівнянні із стандартною бібліотекою encoding/json мови go.

Причиною такої продуктивності є те, що даний пакет генерує спеціалізований код для кожної структури даних заздалегідь, замість використання рефлексії в процесі виконання програми. Так, наприклад, для типу даних SignInRequest, який використовується під час авторизації користувача (лістинг 2.8) код методу для серіалізації виглядатиме як у лістингу 2.9

Лістинг 2.8 – Тип даних (структура) SignInRequest

```
type SignInRequest struct {
	Email    string `json:"email"`
	Password string `json:"password"`
}
```

Кінець лістингу 2.8

Лістинг 2.9 – Алгоритм серіалізації типу SignInRequest у формат JSON

```
func easyjsonB1a2c735EncodeFitnessApplicationHandlersAuth(
	out *jwriter.Writer,
	in SignInRequest,
) {
	out.RawByte('{')
	first := true
	_ = first
	{
		const prefix string = ",\"email\":"
		out.RawString(prefix[1:])
		out.String(string(in.Email))
	}
	{
		const prefix string = ",\"password\":"
		out.RawString(prefix)
		out.String(string(in.Password))
	}
	out.RawByte('}')
}
```

Кінець лістингу 2.9

Такий підхід дозволяє досягти швидкодії, близької до роботи з бінарними форматами, зберігаючи при цьому читабельність та простоту JSON. Пакет автоматично створює оптимізовані функції MarshalJSON та UnmarshalJSON, що дозволяє використовувати цей пакет без значних змін у існуючому коді ліст. 2.10).

Лістинг 2.10 – Імплементація MarshalJSON та UnmarshalJSON

```
// MarshalJSON supports json.Marshaler interface
func (v SignInRequest) MarshalJSON() ([]byte, error) {
	w := jwriter.Writer{}
	easyjsonB1a2c735EncodeFitnessApplicationHandlersAuth(&w, v)
	return w.Buffer.BuildBytes(), w.Error
}

// UnmarshalJSON supports json.Unmarshaler interface
func (v *SignInRequest) UnmarshalJSON(data []byte) error {
	r := jlexer.Lexer{Data: data}
	easyjsonB1a2c735DecodeFitnessApplicationHandlersAuth(&r, v)
	return r.Error()
}
```

Кінець лістингу 2.10

Для забезпечення ефективної взаємодії між сервісами core та notification було обрано брокер повідомлень NATS, який являє собою високопродуктивну систему обміну повідомленнями, спеціально розроблену для сучасних розподілених систем і мікросервісних архітектур. Основною перевагою NATS є простота використання в поєднанні з надзвичайно високою швидкістю обробки повідомлень та мінімальними вимогами до системних ресурсів. На відміну від більш складних брокерів повідомлень, NATS забезпечує легкий (з точки зору потрібних для роботи ресурсів) та ефективний механізм комунікації, що особливо важливо для систем, де критичною є низька затримка при передачі повідомлень.

NATS підтримує декілька моделей обміну повідомленнями: publish-subscribe, request-reply та work queuing. Така гнучкість дозволяє обирати потрібний шаблон комунікації між сервісами без необхідності адаптації сервісів

під можливості брокера повідомлень. Варто зазначити, що архітектура NATS базується на концепції тем (subjects), які використовуються для маршрутизації повідомлень між відправниками та отримувачами. Сервіси можуть публікувати повідомлення в певні теми (subject'и), а інші сервіси, підписані на ці теми, автоматично отримуватиму відповідні повідомлення. Така модель забезпечує слабку зв'язаність між компонентами системи, оскільки відправник не повинен нічого знати про потенційних отримувачів повідомлень.

Для серіалізації та десеріалізації структур даних Golang у формат, придатний для передачі через NATS також було використано комбінацію JSON/easyjson. У даному випадку вибір JSON як формату серіалізації обумовлений декількома факторами. По-перше, JSON є широко розповсюдженим та добре зрозумілим форматом обміну даними, що спрощує процес розробки та налагодження системи. По-друге, текстовий характер JSON робить процес моніторингу та налагодження міжсервісної комунікації більш зручним, оскільки розробники можуть безпосередньо читати та аналізувати вміст повідомлень без використання спеціалізованих інструментів.

Незважаючи на переваги такого підходу, варто зазначити, що для систем з високими вимогами до продуктивності та ефективного використання пропускної здатності мережі більш доцільним був би вибір Protocol Buffers (ProtoBuf) як формату серіалізації та десеріалізації даних.

ProtoBuf, розроблений компанією Google, являє собою бінарний формат серіалізації структурованих даних, що забезпечує значно кращі характеристики як за швидкістю обробки, так і за розміром повідомлень порівняно з текстовими форматами на кшталт JSON. Проте попри очевидні технічні переваги ProtoBuf, рішення використовувати JSON з easyjson на поточному етапі розробки було виправданим з точки зору простоти та зручності роботи.

Обидва бекенд-сервіси для збереження або отримання потрібних даних мають доступ до СУБД PostgreSQL. Дана СУБД характеризується здатністю підтримувати складні запити та великі обсяги даних, що робить її ідеальним вибором для високонавантажених систем та рішень з мікросервісною

архітектурою. PostgreSQL гарантує повну відповідність стандарту ACID, що означає забезпечення атомарності, консистентності, ізоляції та довговічності транзакцій – критично важливих показників для збереження цілісності даних у розподілених системах.

Також у випадку мікросервісної архітектури, PostgreSQL забезпечує ефективну конкурентну роботу завдяки механізму Multi-Version Concurrency Control (MVCC), який мінімізує блокування при одночасному доступі різних сервісів до спільних даних. СУБД підтримує різні рівні ізоляції транзакцій, що дозволяє налаштувати оптимальний баланс між консистентністю даних та продуктивністю залежно від потреб кожного сервісу. Вбудовані механізми реплікації та розподілу навантаження забезпечують високу доступність та масштабованість системи при зростанні обсягів даних та кількості запитів.

Для взаємодії з СУБД було використано Bun - високопродуктивним ORM для go. Bun побудований на основі sql.DB із стандартної бібліотеки Golang, що забезпечує мінімальні накладні витрати та максимальну сумісність зі стандартними інструментами мови. Також дана ORM відзначається своєю продуктивністю завдяки мінімальному використанню рефлексії та ефективному керуванню пам'яттю. Бібліотека генерує оптимізовані SQL запити, які максимально використовують можливості PostgreSQL, включаючи підтримку складних з'єднань, підзапитів, віконних функцій та специфічних для PostgreSQL конструкцій. Важливою перевагою є підтримка пакетних операцій, які дозволяють ефективно виконувати масові вставки та оновлення з мінімальною кількістю звернень до бази даних. Окрім згаданого вище, суттєвою перевагою bun є механізм міграцій «з коробки».

Система міграцій Bun інтегрована безпосередньо в бібліотеку та підтримує версіонування схеми бази даних з можливістю автоматичного відстеження змін. Міграції пишуться на SQL або go, з використанням того ж API, що й основний код застосунку, що забезпечує консистентність та зручність підтримки. Bun автоматично відстежує застосовані міграції та забезпечує

можливість відкату змін у разі необхідності, що важливо для безпечного розгортання оновлень як у тестовому так і робочому середовищі (ліст.2.11).

Лістинг 2.11 – Функція виконання міграції БД

```
func Migrate(ctx context.Context, db *bun.DB) error {
	migrator := migrate.NewMigrator(db, Migrations)

	group, err := migrator.Migrate(ctx)
	if err != nil {
		return err
	}

	if group.ID == 0 {
		fmt.Printf("there are no new migrations to run\n")
		return nil
	}

	fmt.Printf("migrated to %s\n", group)
	return nil
}
```

Кінець лістингу 2.11

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ РОЗРОБКИ КРОСПЛАТФОРМНОГО ЗАСТОСУНКУ ДЛЯ ТРЕКІНГУ ТРЕНУВАНЬ З РЕКОМЕНДАЦІЙНОЮ СИСТЕМОЮ НА БАЗІ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Методика проведення дослідження

Метою експериментального дослідження є комплексна верифікація розробленого кросплатформного застосунку для трекінгу тренувань з рекомендаційною системою на базі штучного інтелекту, оцінка його ефективності та зручності використання у порівнянні з існуючими на ринку рішеннями.

У дослідженні взяли участь 15 респондентів, які відповідали наступним критеріям:

- вік 20 - 35 років;
- можливість виконувати фізичні вправи не менше 3х разів на тиждень;
- наявний досвід тренувань;
- досвід у використанні апаратних та/або програмних трекерів фізичної активності.

Учасників дослідження було розділено на три групи по 5 у кожній. Респонденти першої групи займалися по заздалегідь складеній програмі тренувань, яка була адаптована під потреби, можливості та запити кожного із учасників індивідуально професійним фітнес-тренером.

Друга група користувачів у своїх тренуваннях використовувала розроблений мобільний застосунок. Учасники цієї групи могли формувати індивідуальні комплекси вправ за допомогою ШІ або ж обирати заздалегідь підготовлені програми тренувань (рис. 3.1).

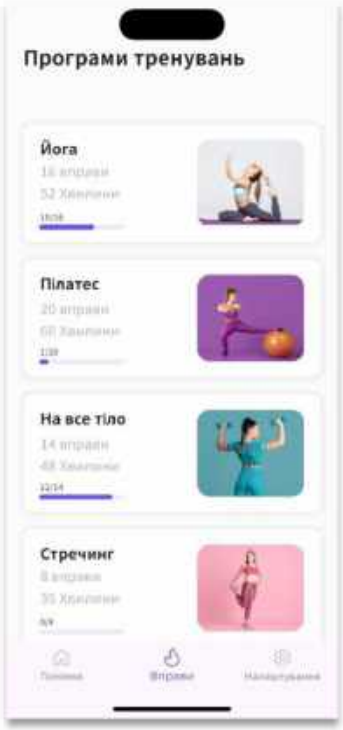


Рисунок 3.1 – Вибір програми тренувань

Учасники третьої групи користувалися довільними програмними рішеннями для контролю періодичності та тривалості тренувань.

За результатами дослідження, учасників попросили оцінити зручність та «дружність» інтерфейсу застосунку яким вони користувалися (окрім тих хто був у групі №1), самопочуття, мотивацію до занять, програму тренувань та ймовірність продовження занять у відповідному режимі після завершення періоду дослідження. Кожен із зазначених вище критеріїв оцінювався за 10-бальною шкалою від 1 до 10.

Тривалість дослідження становила 14 календарних днів.

3.2 Обробка та аналіз отриманих результатів

Проведене експериментальне дослідження мало на меті порівняти ефективність трьох різних підходів до організації фізичних тренувань. У дослідженні прийняло участь три групи респондентів, кожна з яких

використовувала свій метод до планування та контролю тренувального процесу. Оцінювання здійснювалося за п'ятьма параметрами:

– оцінка самопочуття респондентів після завершення періоду досліджень;

– оцінка мотивації до занять під час періоду досліджень («Показник 2»);

– оцінка інтерфейсу використовуваного програмного рішення («Показник 3»), який ігнорується учасниками з групи 1;

– оцінка тренувальної програми («Показник 4»);

– оцінка ймовірності продовження занять у такому ж вигляді («Показник 5»).

Графік медіани оцінок кожної групи респондентів по кожному із показників наведено нижче:

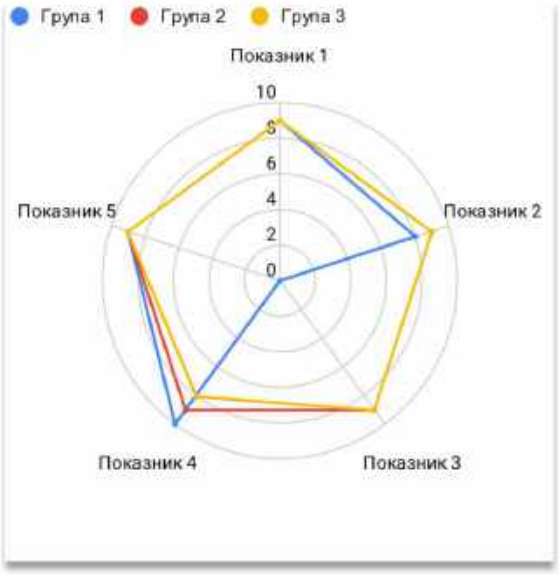


Рисунок 3.2 – Медіанна оцінка досліджуваних показників.

Перша група, яка займалася за програмою, складеною професійним фітнес-тренером, показала найвищі результати у показниках тренувальної програми та ймовірності продовження тренувань, отримавши максимальні оцінки 10 та 9 балів відповідно. Водночас ця група продемонструвала високі показники самопочуття та мотивації (по 9 та 8 балів).

Друга група, яка використовувала розроблений мобільний застосунок, продемонструвала найбільш збалансовані показники. Респонденти цієї групи оцінили усі п'ять параметрів на рівні 9 балів, що свідчить про високу якість та ефективність застосунку. Особливо важливим є той факт, що показники мотивації, тренувальної програми та ймовірності продовження тренувань у цій групі виявилися на одному рівні з результатами першої групи, яка мала підтримку професійного тренера.

Третя група користувачів, які застосовували довільні програмні рішення для контролю тренувань, показала дещо нижчі результати порівняно з другою групою. Хоча самопочуття, мотивація, зручність інтерфейсу та ймовірність продовження занять отримали високі оцінки на рівні 9 балів, якість тренувальної програми була оцінена нижче – лише на 8 балів. Це може свідчити про те, що загальнодоступні програмні рішення не завжди забезпечують достатній рівень персоналізації тренувального процесу.

Загалом результати дослідження демонструють, що розроблений мобільний застосунок з можливостями штучного інтелекту здатний забезпечити рівень ефективності, порівнянний з роботою професійного тренера, при цьому надаючи зручний користувацький інтерфейс. Це підтверджує доцільність використання сучасних технологій для створення персоналізованих програм тренувань, які можуть стати доступною альтернативою дорогим послугам персонального тренера.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено та досліджено кросплатформний застосунок для трекінгу тренувань з рекомендаційною системою на базі штучного інтелекту, який доступний на мобільних платформах iOS та Android.

Практична цінність виконаної роботи полягає у створенні MVP, який може бути впроваджений у дослідну експлуатацію та має значний потенціал для комерціалізації. Висока персоналізація та ефективність досягаються завдяки динамічній адаптації тренувального процесу завдяки врахуванню показника RPE, що підвищує ефективність тренувань порівняно зі статичними шаблонами.

Розроблені серверне рішення на Golang та кросплатформний застосунок на Flutter забезпечують високу стійкість, швидкість та здатність системи до масштабування. Створена RAG-архітектура може бути використана як шаблон для побудови інших інтелектуальних систем у критично важливих доменних сферах, де поєднання генеративних можливостей LLM з верифікованою базою знань є критично необхідним.

Вибір SDK Flutter забезпечив високу продуктивність розробки мобільного застосунку. Завдяки власному рушію рендерингу Skia, рішення на Flutter можуть досягати частоти оновлення екрану 60-120 кадрів за секунду, що мало позитивний вплив на UX, а за рахунок кросплатформності даного SDK суттєво скоротився час для створення застосунку під кожну з цільових платформ. Використання шаблону проєктування BLoC та відповідних dart-пакетів, дозволило втілити принцип розділення відповідальності шляхом ізоляції бізнес-логіки від презентаційного рівня, використовуючи реактивне програмування на основі потоків даних (streams) для автоматичного оновлення стану інтерфейсу користувача. Така архітектура мобільного застосунку забезпечила високий рівень тестованості коду, підвищила його модульність і масштабованість, а

також у перспективі спрощує підтримку проєкту в процесі його розвитку та розширення функціональності.

Для розробки бекенд частини об'єкту дослідження було застосовано мову програмування Golang у поєднанні з веб-фреймворком Fiber та мікросервісною архітектурою. Це рішення виявилося оптимальним з огляду на сучасні вимоги до розподілених систем та необхідність забезпечення високої продуктивності додатку. Go демонструє виняткову продуктивність завдяки компіляції безпосередньо в машинний код, а вбудована підтримка конкурентності через горутини дозволяє ефективно обробляти тисячі одночасних запитів з мінімальним споживанням ресурсів. Fiber пропонує інтуїтивний API та надзвичайно високу швидкість обробки HTTP-запитів завдяки використанню бібліотеки fasthttp. Фреймворк надає широкий набір middleware для роботи з роутингом, валідацією та обробкою помилок, що дозволяє зосередитися на бізнес-логіці замість реалізації базового функціоналу. Використання мікросервісної архітектури дозволило реалізувати бекенд рішення у вигляді окремих незалежних сервісів, кожен з яких відповідає за конкретну бізнес-функцію, що забезпечує високий рівень модульності та можливість паралельної розробки різних компонентів системи. Кожен мікросервіс може бути розгорнутий та масштабований незалежно, що значно підвищує гнучкість системи та забезпечує відмовостійкість.

Результати дослідження підтверджують гіпотезу про те, що застосування гібридних інтелектуальних систем на базі LLM у поєднанні з RAG-архітектурою дозволяє створювати високоперсоналізовані рекомендації в сфері фізичного здоров'я без компромісів щодо безпеки користувача. Розроблена система демонструє, що сучасні технології штучного інтелекту можуть стати доступною альтернативою дорогим послугам персонального тренера, забезпечуючи при цьому не менший рівень ефективності та безпеки тренувального процесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дорошенко А. Ю. Розподілені системи та паралельні обчислення: підручник. Київ : НаУКМА, 2018. 326 с.

2. Заковоротний О. Ю., Орлова Т.О., Гриньов Д.В. Обчислювальний інтелект. Навчальний посібник. Харків: НТУ «ХПІ», 2024. 264 с.

3. Недашківська Н. І. Прийняття рішень в ієрархічних системах. URL: https://ai.kpi.ua/ua/masters/2019nedashkivska-n-i-pryiniattia-rishen-v-iierarkhichny khsys temakh.pdf (дата звернення: 01.09.2025.)

4. Любченко В. В. Методи та алгоритми рекомендаційних систем: конспект лекцій. Одеса : ОНПУ, 2020. 84 с.

5. Васильєв О. М. Розробка мобільних застосунків : практикум. Київ : КПІ ім. Ігоря Сікорського, 2021. 145 с.

6. Ashok Kumar Shaw, Biswadip Basu Mallik, Dac-Nhuong Le, Gunjan Mukherjee, Rahul Kar. Fuzzy Logic Applications in Computer Science and Mathematics. Wiley, 2023. 304 p.

7. Mathematics-Driven Computational Linguistics and Discourse Analysis for Political Insight and Pedagogical Innovation. URL: https://surl.li/mwlepn (дата звернення: 11.09.2025.).

8. Бабич М. В. Архітектура програмного забезпечення : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2019. 156 с.

9. New Information Technologies, Simulation and Automation. URL: https://arxiv.org/abs/2301.01028?utm_source=chatgpt.com

10. Жилін А. В., Шаповал О. М., Успенський О. А. Технології захисту інформації в інформаційно-телекомунікаційних системах : навч. посіб. ІСЗЗІ КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2021. 213 с.

11. The Go Programming Language. URL: https://go.dev/ (дата звернення: 12.09.2025.).

12. ДСТУ ISO 9241-210:2014. Ергономіка взаємодії людина-система. Частина 210. Людино-орієнтоване проєктування інтерактивних систем (ISO 9241-210:2010, IDT).

13. Карпенко С. Г. Організація баз даних та знань у високонавантажених системах : конспект лекцій. Харків : ХНУРЕ, 2019. 156 с.

14. Ковальчук С. С. Методи забезпечення відмовостійкості програмних систем : монографія. Львів : Видавництво Львівської політехніки, 2019. 210 с.

15. Купер А. Про інтерфейс. Основи проєктування взаємодії : навч. посіб. Харків : Фабула, 2019. 720 с.

16. Дворецький М. Л., Нездолій Ю. О., Дворецька С. В., Кандиба І. О. Розробка мобільних застосунків для OS Android : навч. посіб. Миколаїв : Вид-во ЧНУ ім. Петра Могили, 2021. 140 с.

17. Лисенко О. І., Тачиніна О. М. Методи оптимізації : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 385 с.

18. Станков, І. С. Аналіз та порівняння рекомендаційних систем колаборативної фільтрації : дипломна робота ... бакалавра : 124 Системний аналіз. Київ, 2020. 100 с.

19. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol : O'Reilly Media, URL: https://surli.cc/vatigj (дата звернення: 14.09.2025).

20. Comprehensive Evaluation of Matrix Factorization Models for Collaborative Filtering Recommender Systems. URL: https://arxiv.org/abs/2410.17644?utm_source (дата звернення: 01.10.2025).

21. Lewis P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems (NeurIPS). 2020. Vol. 33. P. 9459-9474.

22. Петренко А. І. Основи побудови сучасних веб-сервісів : навч. посіб. Харків : ХНУРЕ, 2020. 184 с.

23. Software Architecture: The Hard Parts. URL: https://surl.li/qldehw (дата звернення: 21.10.2025).

24. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. 3rd ed. Sebastopol : O'Reilly Media, 2022. 550 p.

25. Docker: Practical Guide for Developers and DevOps Teams. https://surl.li/wawcge

26. Napoli M. Beginning Flutter: A Hands On Guide to App Development. Indianapolis : Wrox, 2019. 250 p.

27. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 614 p.

28. Research on the application of ergonomics in UI interface design - Zhiyong Chen, 2023 URL: https://www.researchgate.net/publication/374966311_Research_on_the_application_of_ergonomics_in_UI_interface_design (дата звернення: 22.10.2025).

29. Schoenfeld B. J. Science and Development of Muscle Hypertrophy. 2nd ed. Champaign : Human Kinetics, 2020. 312 p.