Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Ваховської Віри Миколаївни
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Мобільний застосунок «GymRat» –
Назва теми
віртуальний фітнес-тренер

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.200245.01.04.ПЗ

Виконав студент IV курсу, група ІПЗ-20-1 _____ Віра ВАХОВСЬКА
Підпис Ім'я, ПРІЗВИЩЕ

Керівник д-р фіз.-мат. наук, професор _____ Леонід БЕДРАТЮК
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. техн. наук, доцент _____ Ганна БЕДРАТЮК
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

6 червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ваховській Вірі Миколаївній
Прізвище, ім'я, по батькові студента

1. Тема кваліфікаційної роботи Мобільний застосунок «GymRat» – віртуальний фітнес-тренер

Керівник кваліфікаційної роботи Бедратюк Леонід Петрович,
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання
д-р фіз.-мат. наук, професор
Затверджена наказом ректора університету від 08.01.2024 р. № 6
2. Строк подання студентом роботи на кафедру 06.06.2024 р.
3. Вихідні дані до роботи Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проєктування мобільного застосунку, програмна реалізація та тестування
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди, 19 шт.), діаграма варіантів використання, діаграма активності процесу авторизації та діаграма активності процесу відображення категорій, діаграма архітектури системи, логічна модель бази даних та схема екранів застосунку

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Бедратюк Г. І., старший викладач	4.06	6.06
Антиплагіат	Форкун Ю. В., доцент	30.05.24	06.05.24

7. Дата видачі завдання «_02_» __січня_ 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
2 Проєктування програмного забезпечення	21.02 – 20.03 2024	
3 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
4 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2024	
5 Попередній захист КвР	травень 2024	Згідно графіка
6 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.05.2024	
7 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Заховська В.М.
Підпис Ініціали, прізвище

Керівник роботи _____ _____
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мобільний застосунок «GymRat» – віртуальний фітнес-тренер».

Автор роботи: Ваховська Віра Миколаївна.

Керівник роботи: Бедратюк Леонід Петрович.

Пояснювальна записка: 69 с., 32 рис., 4 табл., 5 дод., 43 джерела.

Графічна частина: 3 креслення.

МОБІЛЬНИЙ ЗАСТОСУНОК, ПЕРСОНАЛІЗОВАНІ ПРОГРАМИ ТРЕНУВАНЬ, ФІТНЕС, ШТУЧНИЙ ІНТЕЛЕКТ, FINE-TUNING, REACT NATIVE.

Метою роботи є розробка мобільного фітнес-застосунку, який дозволятиме користувачам легко знаходити вправи, проходити тренування та створювати власні персоналізовані тренувальні програми, використовуючи штучний інтелект.

У кваліфікаційній роботі було проведено дослідження фітнес-індустрії, здійснено аналіз існуючих мобільних фітнес-застосунків, визначено функціональні вимоги, розроблено технічне завдання, спроєктовано архітектуру системи та застосунку, обрано технології для реалізації, реалізовано застосунок «GymRat», який містить потрібний функціонал, та проведено тестування готового рішення.

Для розробки гібридного мобільного застосунку використовувалися такі інструменти, як React Native, Expo, Redux Toolkit. Зовнішні сервіси: Firebase (Authentication, Firestore, Realtime Database, Storage), Hugging Face (Hub, Meta Llama 2 (fine-tuned), Inference Endpoints) та RapidAPI (ExerciseDB).

Впровадження розробленого застосунку «GymRat» дозволить користувачам отримувати персоналізовані тренування, завдяки використанню тонко налаштованої моделі, та відстежувати свій прогрес у календарю активності, що сприятиме підвищенню мотивації та покращенню результатів.

Застосунок надає користувачам зручний інструмент для здійснення фітнес-тренувань у будь-який час та в будь-якому місці, що сприяє покращенню фізичної активності та здоров'я.

Застосунок «GymRat» може бути використаний на будь-якому гаджеті з ОС Android версії вище 5, та не потребує додаткових налаштувань для початку роботи.

Можливі напрямки подальшої роботи можуть включати розширення функціональності застосунку, вибір інших постачальників послуг або перехід на інші платформи для збільшення доступності.

6.06.2024
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.200245.01.04.ПЗ	Пояснювальна записка	130		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ.200245.01.04.Е8	Архітектура	1		
			системи			
5	А3	КвРІПЗ.200245.01.04.Е8	Логічна модель бази	1		
			даних			
6	А3	КвРІПЗ.200245.01.04.Е8	Схема екранів	1		
			застосунку			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.200245.01.04.ВД		
Виконав		Ваховська В.М.		6.06	Мобільний застосунок «GymRat» – віртуальний фітнес-тренер Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Л.П.		6.06			
Рецензент		Говорущенко Т.О.		6.06		ХНУ, ІПЗ-20-1	
Н. контр.		Бедратюк Г.І		6.06			
Зав. каф.		Бедратюк Л.П.		6.06			

ЗМІСТ

ВСТУП

Сучасне суспільство дедалі більше орієнтується на підтримку здорового способу життя та фізичної активності. Зростання популярності фітнесу та спорту є невід'ємною частиною цього процесу [1]. У зв'язку з цим мобільні технології набули важливої ролі в житті людей, оскільки вони надають можливість займатися спортом у будь-який час та в будь-якому місці. За прогнозами 2024 року, світовий ринок застосунків для фітнесу очікує досягнення доходу в розмірі 6,86 млрд доларів США [2]. Цей прогнозований рівень розвитку свідчить про надзвичайно високий темп зростання, з CAGR (Сукупний середньорічний темп росту) на період 2024-2028 років, що становить 9,99%, і прогнозований обсяг ринку в 10,04 млрд доларів США до кінця 2028 року. У січні 2024 року, відзначено значний підйом популярності – майже 14 мільйонів завантажень по всьому світу [3]. А у січні 2020 року було зафіксовано понад 16,3 мільйона завантажень провідних програм для фітнесу та тренувань, що свідчить про зростання на 80% порівняно з попереднім роком [4].

Актуальність теми є те, що, незважаючи на наявність різноманітних мобільних фітнес-застосунків з вправами та готовими тренуваннями, багато з них не забезпечують зручного та персоналізованого підходу до індивідуальних потреб користувачів. Зазвичай такі застосунки обмежені стандартними програмами та шаблонами тренувань, не враховуючи особливостей кожного користувача. Бажання користувачів мати персоналізовані тренування та можливість самостійного створення програм стає важливим елементом вибору фітнес-застосунків [5].

Тому, на фоні існуючих обмежень, виникає потреба в розробці мобільного застосунку, який виступатиме в ролі віртуального фітнес-тренера та забезпечуватиме можливість користувачам створювати персоналізовані тренування самостійно, так і з використанням штучного інтелекту [6].

Основною метою проєкту є розробка мобільного фітнес-застосунку, який дозволятиме користувачам легко знаходити вправи, проходити тренування, слідкувати за власною активністю та створювати персоналізовані тренувальні програми, використовуючи штучний інтелект.

Завдання, які треба вирішити для досягнення мети:

– дослідити предметну область, визначити особливості та специфіку фітнес-індустрії;

– здійснити аналіз існуючих мобільних фітнес-застосунків, їх функціональності та обмежень для визначення оптимального підходу;

– визначити функціональні вимоги та завдання, які повинен виконувати застосунок;

– підсумувати дану інформацію у технічному завданні;

– виконати проєктування архітектури застосунку за визначеною предметною областю;

– обрати технології для реалізації програмного продукту;

– реалізувати застосунок, який містить потрібний функціонал;

– провести тестування готового рішення.

Практична значимість полягає у створенні нового продукту, який може залучити широку аудиторію користувачів та стати комерційно успішним. Мобільний застосунок «GymRat», розроблений у рамках даного проєкту, забезпечить користувачам можливість займатися фітнесом у будь-який час та в будь-якому місці, що сприятиме покращенню їх фізичної активності та здоров’я. Можливість створення персоналізованих програм та використання штучного інтелекту для рекомендацій зробить його унікальним на ринку мобільних застосунків.

Подальший розвиток проєкту може включати інтеграцію з іншими мобільними застосунками, носимими пристроями та сервісами для забезпечення більш комплексного підходу до підтримки здорового способу життя.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Мобільний застосунок [7] – це програмне забезпечення, розроблене для використання на мобільних пристроях, таких як планшети та смартфони. Вони зазвичай доступні для завантаження у спеціальних магазинах застосунків, таких як Google Play Store для пристроїв з операційною системою Android та App Store для iOS. Типи мобільних застосунків [8] відображені в таблиці 1.

Таблиця 1 – Типи мобільних застосунків

Тип	Опис
Нативні застосунки	Застосунки, що написані рідною мовою програмування для конкретної платформи, а саме Java або Kotlin для Android, і Objective-C або Swift для iOS.
Мобільні веб-застосунки	Мобільні версії веб-сайтів з розширеним інтерактивом. Розроблюються використовуючи HTML, CSS та JavaScript.
Гібридні застосунки	Поєднання нативних та веб-застосунків. Розроблені за допомогою технологій, таких як Xamarin, Apache Cordova, Flatter, React Native, Ionic.

Для охоплення ширшої аудиторії користувачів, що користується різними операційними системами, було обрано гібридний тип застосунку. За допомогою використання однієї кодової бази для різних платформ, вийде ефективно використовувати ресурси і зменшити час розробки, у порівнянні зі створенням окремих нативних застосунків для кожної платформи.

Сьогодні більшість людей користуються мобільними застосунками та системами відстеження, щоб спростити процес тренувань.

Підсумовуючи, можна виділити основні переваги використання мобільних застосунків для занять фітнесом:

– зручність та доступність: можливість займатися фітнесом без необхідності відвідування спортзалу, тобто у будь-який час, в будь-якому місці;

– різноманітність програм та наборів вправ: надання великого розмаїття вправ та відповідних рекомендацій щодо їх виконання;

– персоналізація тренувальних планів: можливість самостійно створювати індивідуальні плани тренувань, з врахуванням унікальних потреб та можливостей кожної особи;

– збереження та аналіз даних: можливість відстежувати власний прогрес тренувань;

При розробці функціональної моделі мобільного застосунку було використано програмне забезпечення AllFusion BPwin (Business Processing) Data Modeler 7.1. та стандарт IDEF0 [9].

На рисунку 1.1 представлено контекстну діаграму, на якій відображено основну функцію модельованого застосунку, що є об’єктом моделювання, у вигляді блоку з назвою. Кожна сторона цього блоку виконує певну роль: ліва сторона відображає об’єкти, які надходять на вхід, верхня сторона містить інформацію для управління процесом, права сторона відображає об’єкти, які виходять у результаті, а нижня сторона показує об’єкти, які здійснюють роботу.

Контекстна діаграма – це високорівнева діаграма, що демонструє взаємодію системи із зовнішнім середовищем системи. Вхідними даними у застосунку є дані від користувача, його цілі, інформація про вправи та тренування. Управління застосунком здійснюється шляхом надання допомоги та порад користувачеві через довідку користувача. На виході отримуємо списки вправ, категорій та тренувань, інформацію про обліковий запис користувача, пройдені тренування (активність) та створені власні тренування. У ролі механізму управління виступає сервіс для зберігання інформації, сервіс для отримання різноманітних вправ, система та натренована модель.

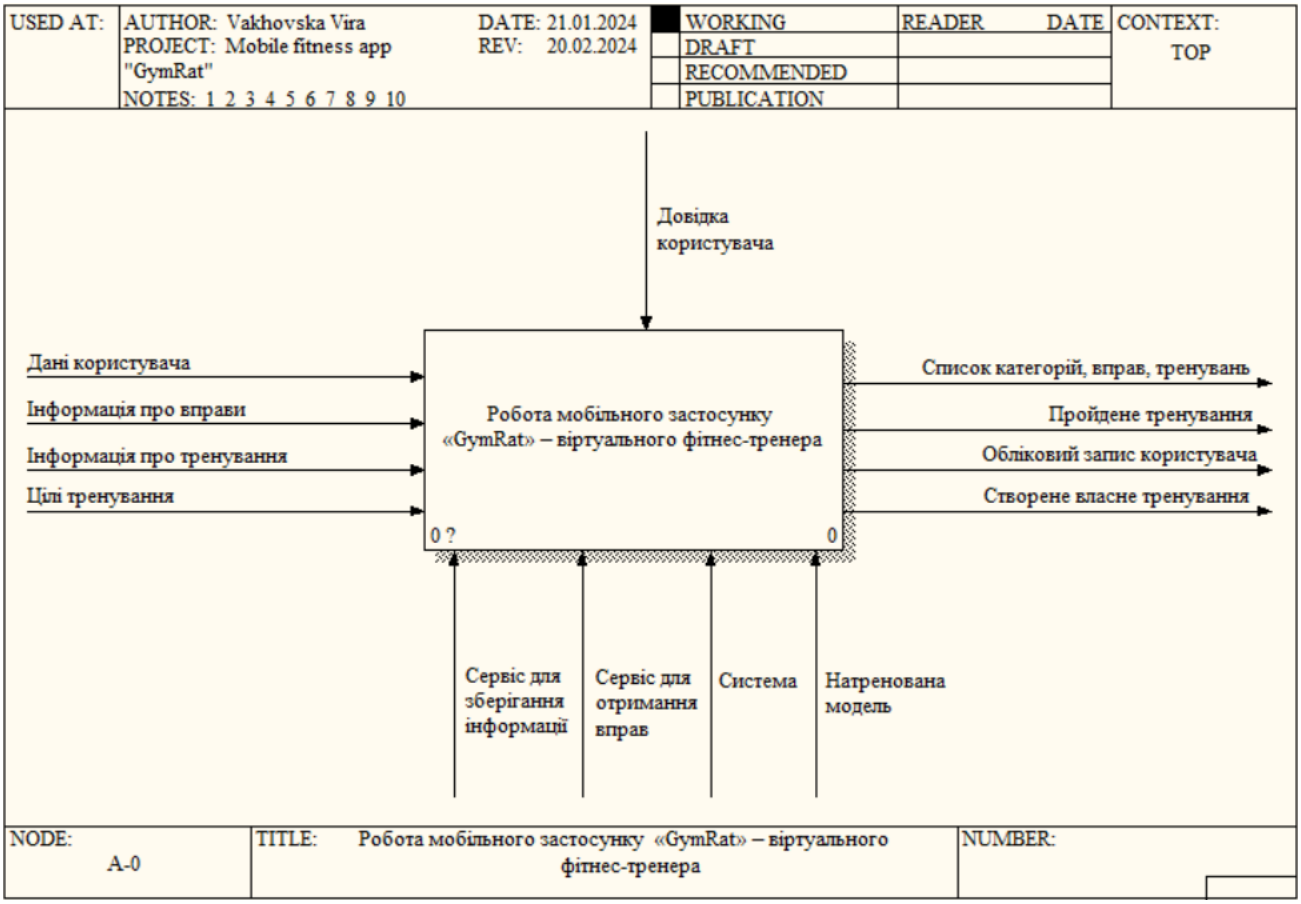


Рисунок 1.1 – Контекстна діаграма роботи мобільного фітнес застосунку

Декомпозиція головної функції системи представлена на рисунку 1.2. Ця декомпозиція дозволяє краще деталізувати головну функцію та складається з чотирьох наступних блоків:

– «Реєстрація/авторизація користувачів» включає в собі процес авторизації та реєстрації використовуючи сервіс для зберігання інформації;

– «Перегляд списків категорій, вправ, тренувань» передбачає відображення списків доступних категорій, вправ та тренувань;

– «Вибір/створення тренування» надає можливість користувачам обрати готове тренування зі списку або створити власне, за бажанням можна отримати згенерований опис тренування від натренованої моделі;

– «Проходження тренування» дозволяє користувачам проходити обране тренування з можливістю механічного, а не лише автоматичного, переключення між вправами та відліком часу за допомогою таймера.

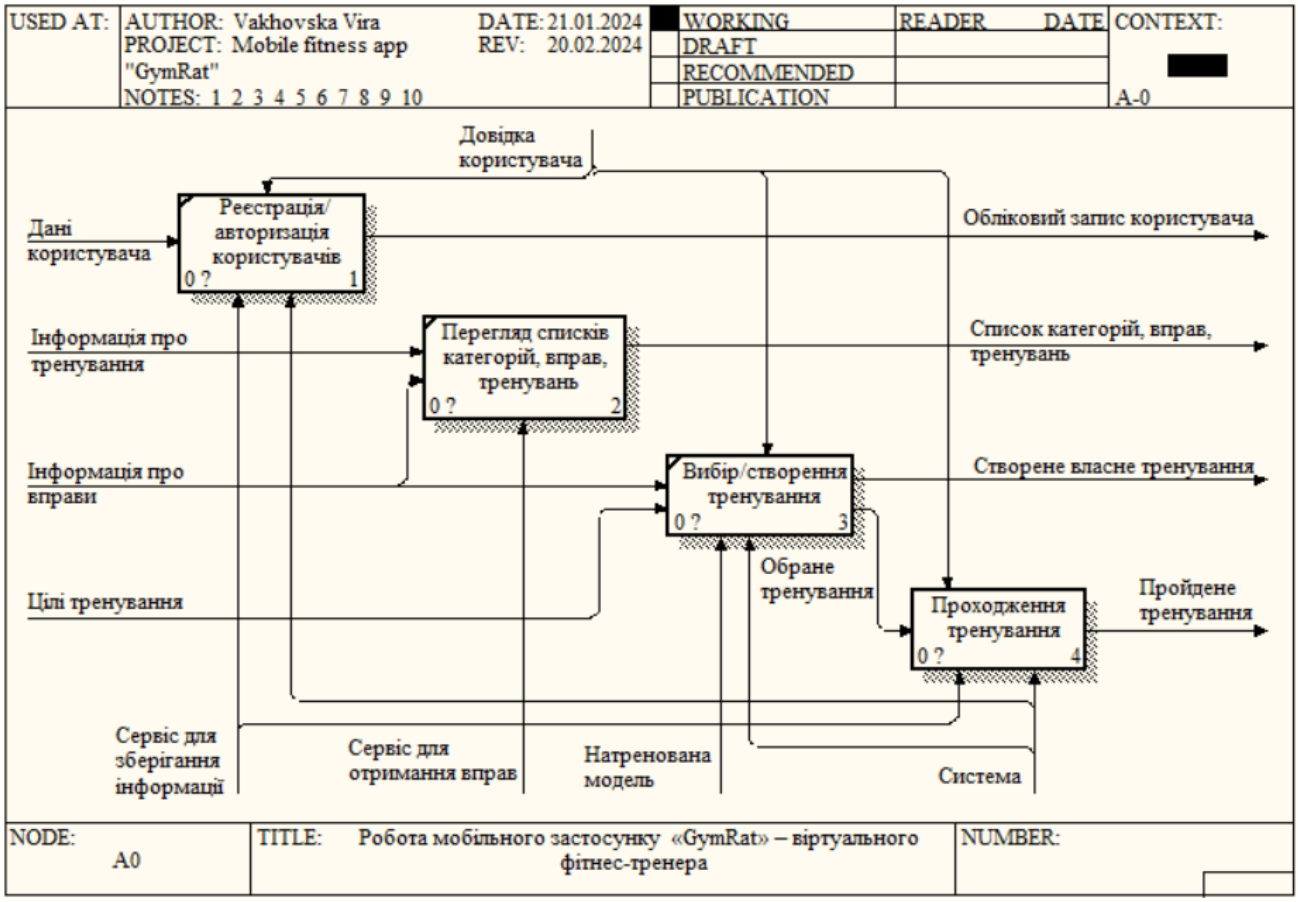


Рисунок 1.2 – Діаграма декомпозиції першого рівня

Декомпозицію функції «Проходження тренування» подано на рисунку 1.3. На вхід цієї функції надходить обране тренування, а на виході користувач отримує відмітку про пройдене тренування. Ця функція розкладається на чотири підфункції:

– «Переглянути інформацію про тренування» забезпечує можливість користувачам ознайомитися з деталями обраного тренування, включаючи його назву, опис, перелік вправ (назва та група цільових м’язів) та кругова діаграма для демонстрації головного фокусу даного тренування;

– «Розпочати тренування» ініціює початок проходження обраного тренування, запускаючи виконання першої вправи;

– «Виконання вправ» дозволяє користувачеві бачити анімацію правильного виконання вправи, таймер, з відліком часу до кінця виконання вправи, та можливість переключити вправу до завершення часу;

– «Завершення тренування» користувач бачить екран привітання та робиться відмітка в сервісі для зберігання інформації.

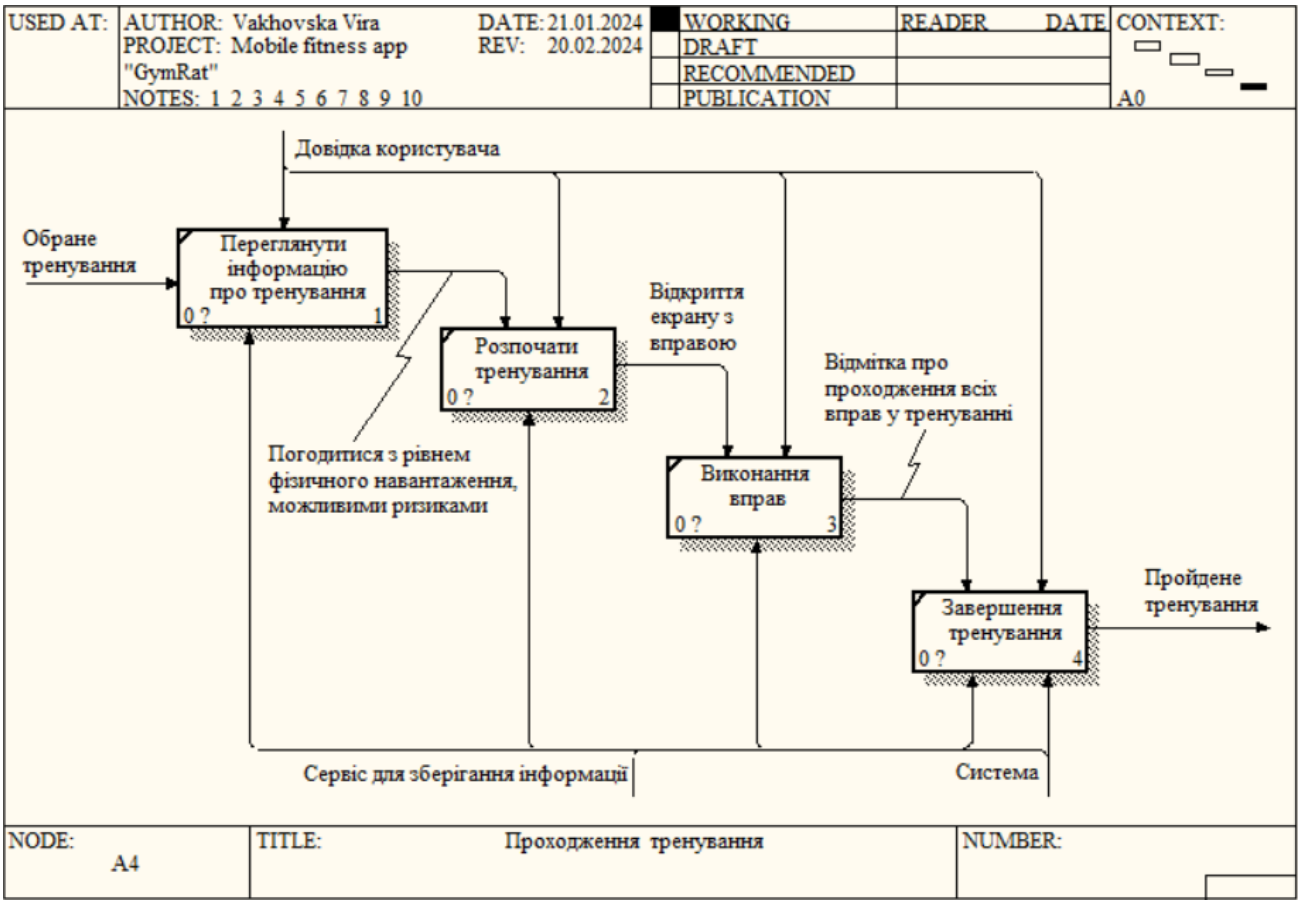


Рисунок 1.3 – Діаграма декомпозиції блоку «Проходження тренування»

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Зараз існує велика кількість мобільних застосунків у категорії «Здоров’я та фітнес» [10], що відображає потенційну широку аудиторію та активний інтерес споживачів [11]. З метою визначення вимог, які потрібно реалізувати у застосунку та ідентифікації недоліків, яких потрібно виправити та уникнути, було проведено порівняльний аналіз двох відомих застосунків у цій категорії – Nike Training Club (NTC) та Jefit.

NTC [12] –мобільний застосунок, спрямований на підтримку активного способу життя через медитації, тренування та здорове харчування.

На сторінці застосунку NTC в Play Market [13] описано наступні функціональні особливості:

– домашні тренування: NTC надає можливість займатися фітнесом в будь-який час та в будь-якому місці, пропонуючи тренування для всіх рівнів підготовки, від початківців до професіоналів;

– тренувальні програми: наявні розроблені тренерами, інструкторами та експертами різноманітні тренувальні програми, для різних фітнес-цілей;

– різноманітність тренувань: користувачі можуть обирати будь-який тип тренувань, включаючи кардіо, силові вправи, вправи на витривалість, медитацію, йогу, тощо;

– спеціалізовані програми: наявні спеціальні програми тренувань для різних частин тіла;

– здорове харчування та відпочинок: NTC не обмежується лише тренуваннями, але також надає користувачам поради щодо активного відпочинку, здорового харчування та відновлення сил;

– спільнота та мотивація: наявна дружня спільнота, де користувачі можуть мотивувати одне одного, обмінюватися досвідом та слідкувати за власними досягненнями;

– синхронізація з іншими фітнес-платформами: для зручного ведення звітності та аналізу даних є синхронізація з Apple Health або Google Fit.

Інтерфейс NTC виконаний в привабливому та сучасному стилі, як це видно на рисунку 1.4. Зручне меню дозволяє користувачам швидко знаходити необхідні тренування та вправи, інформацію про прогрес, а також інші необхідні функції. Однак для тих, хто не має досвіду користування подібними застосунками, інтерфейс може здатися дещо заплутаним і складним через перевантаженість деяких екранів.

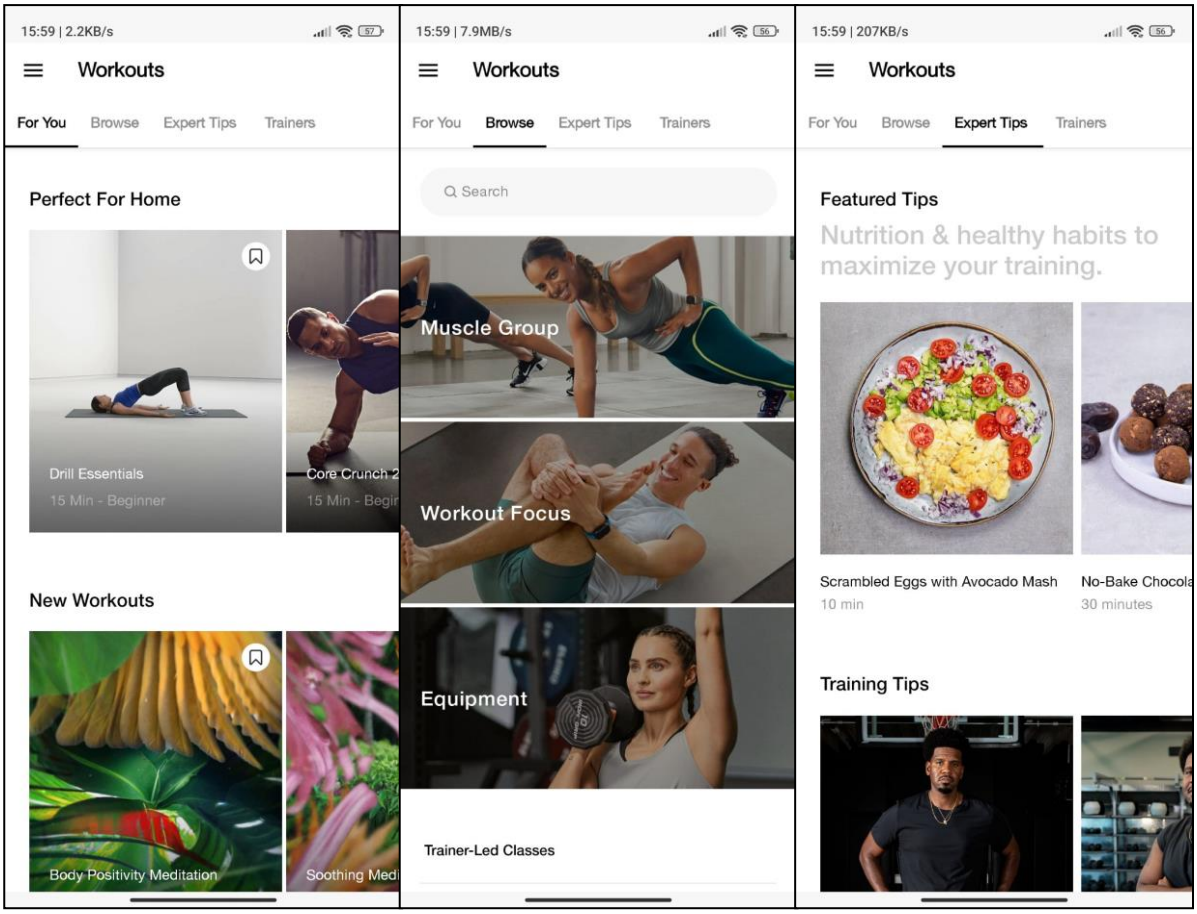


Рисунок 1.4 – Дизайн мобільного застосунку NTC

Застосунок NTC працює на операційних системах Android від версії 8.0 або iOS від версії 16.0, включно. Спочатку він був доступний безкоштовно, проте мав преміум-контент, який під час пандемії COVID став доступним безкоштовно і залишився таким назавжди. Недоліками NTC є відсутність можливості користуватися без авторизації, окремо переглядати вправи та створювати власні тренування. Проходження тренувань у форматі перегляду відео, на мою думку, має більше недоліків, ніж переваг. Оскільки, відсутня можливість легко переключити вправу або повернутися до попередньої. Відповідно, налаштувати індивідуальні інтервали відпочинку між вправами теж не можливо.

Застосунок Jefit, відповідно до інформації у Play Market [14]: Jefit – це інструмент відстеження тренувань, спрямований на фітнес ентузіастів та відвідувачів тренажерного залу. На його офіційній сторінці [15], можна знайти більш детальну інформацію.

Основні функції мобільного застосунку Jefit:

– відстеження тренувань: користувач може записувати вправи, підходи, повторення, ваги, для відстежування прогресу в журналі тренувань;

– планувальник тренувань: можливість створення індивідуальних планів тренувань або вибір готових на основі своїх цілей;

– бібліотека вправ: велика кількість різноманітних вправ з понад 1400 HD-відео та інструкціями виконання;

– спеціальні вправи: можливість додавати вправи для будь-яких програм тренувань;

– відстеження розвитку тіла та сили м’язів: функції для відстеження прогресування сили м’язів;

– наявність фітнес-спільноти: можливість ділитися своїми досягненнями та отримувати підтримку від друзів та тренерів;

– оцінка фізичної підготовки: легке оцінювання витривалості та рівня фізичної підготовки за допомогою спеціальних тренувань;

– інтеграція з Wear OS: можливість відстежувати тренування через модуль Wear OS.

Jefit (рис. 1.5) відрізняється від NTC простим та зрозумілим інтерфейсом, що робить його зручним для будь-якого користувача. Інформація подається в зручній формі, що дозволяє швидко орієнтуватися в застосунку.

Jefit доступний вже для смартфонів з Android 5.0 або iOS 14.0 та новіших версіях. Проте у застосунку також спостерігається відсутність можливості користуватися застосунком без авторизації, проте на відміну від NTC, Jefit пропонує більший вибір вправ, які розділені на категорії відповідно по групах м’язів. Хоча користувачам не надається можливість створення власних тренувань з нуля; є можливість редагувати наявних тренувань за допомогою обмежених опцій. Важливим недоліком є поява нав’язливої реклами, а також значна кількість функцій та опцій про, які заявляє розробник у описі застосунку, є платними.

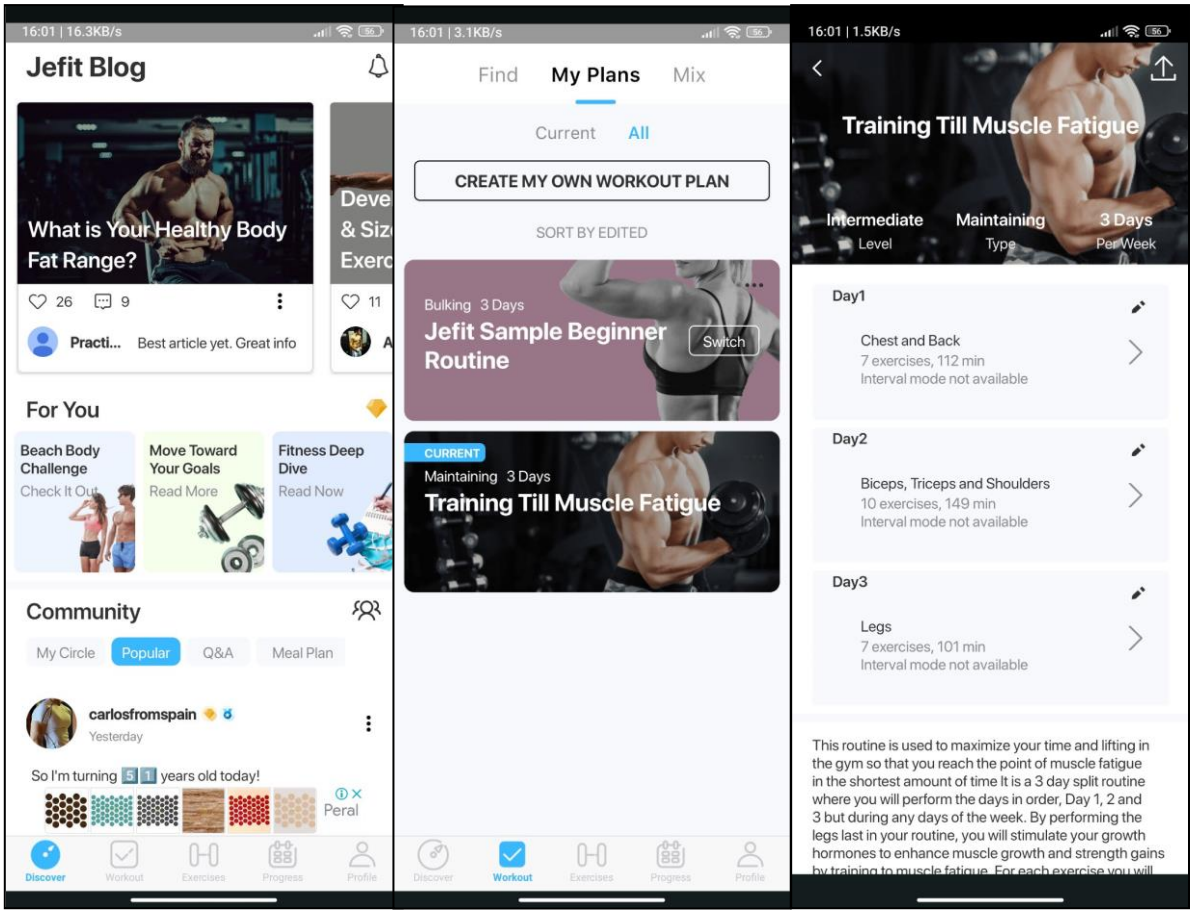


Рисунок 1.5 – Дизайн мобільного застосунку Jefit

Підсумовуючи вищесказане, можна виділити кілька недоліків, яких слід уникати в розробці фітнес-застосунку:

– обов’язкова авторизація: неможливість користування застосунком без авторизації може призвести до відчуття непотрібної складності для користувачів, що може знизити їхнє бажання користуватися застосунком та збільшить ймовірність видалення застосунку одразу після завантаження;

– обмежений та недостатній функціонал безкоштовної версії: важливим аспектом є безкоштовний доступ функціоналу застосунку, оскільки саме через нього користувач знайомиться із застосунком та визначає, чи він відповідає його потребам;

– недостатня персоналізація: користувачам потрібна можливість створити власне тренування та додати у нього вправи, що відповідають їхнім можливостям та потребам;

– надмірна кількість реклами: надмірна кількість рекламних повідомлень може заважати користуватись застосунком та псувати загальне враження користування.

Крім того, застосунок повинен забезпечувати:

– інтуїтивний та зручний інтерфейс: спрощений та легкий доступ до функцій застосунку для покращення користувацького досвіду;

– адаптованість до різних версій операційних систем: розробка застосунку повинна бути виконана спочатку для Android з можливістю швидкої адаптації до роботи на iOS;

– систему відстеження прогресу: надання засобів для фіксації та інструментів аналізу результатів тренувань для підтримки користувачів у досягненні їхніх цілей;

– велику базу даних вправ: розширену бібліотеку вправ для різних типів тренувань та рівнів фізичної підготовки;

– легку навігацію між вправами: можливість пошуку вправи через знаряддя чи назву, та поділити вправи на категорії, що забезпечить можливість легкого сортування вправ.

1.3 Визначення вимог до застосунку та розробка технічного завдання

Для аналізу вимог та моделювання системи з точки зору користувачів використовується діаграма варіантів використання [16]. У застосунку «GymRat» акторами виступають користувач та клієнт.

Актор «Користувач» отримує обмежену версію застосунку, де він може переглядати різні вправи, поділені за категоріями, та проходити доступні тренування. Для отримання повного доступу до застосунку, користувач повинен створити обліковий запис та авторизуватись. Діаграма варіантів використання для актора «Користувач» наведена на рисунку 1.6.

Коли поведінка одного прецедента залежить від іншого, між ними виникає відношення включення, позначене стереотипом «include» [17]. Наприклад, при авторизації відбувається перевірка даних на коректність.

Коли один варіант використання розширює інший, додаючи або змінюючи його поведінку за певних умов застосовується зв’язок «розширення», який позначений стереотипом «extend» [18]. Наприклад, під час перегляду вправ користувач може відфільтрувати вправи за категоріями або виконати пошук за назвою, знаряддям та групою м’язів.

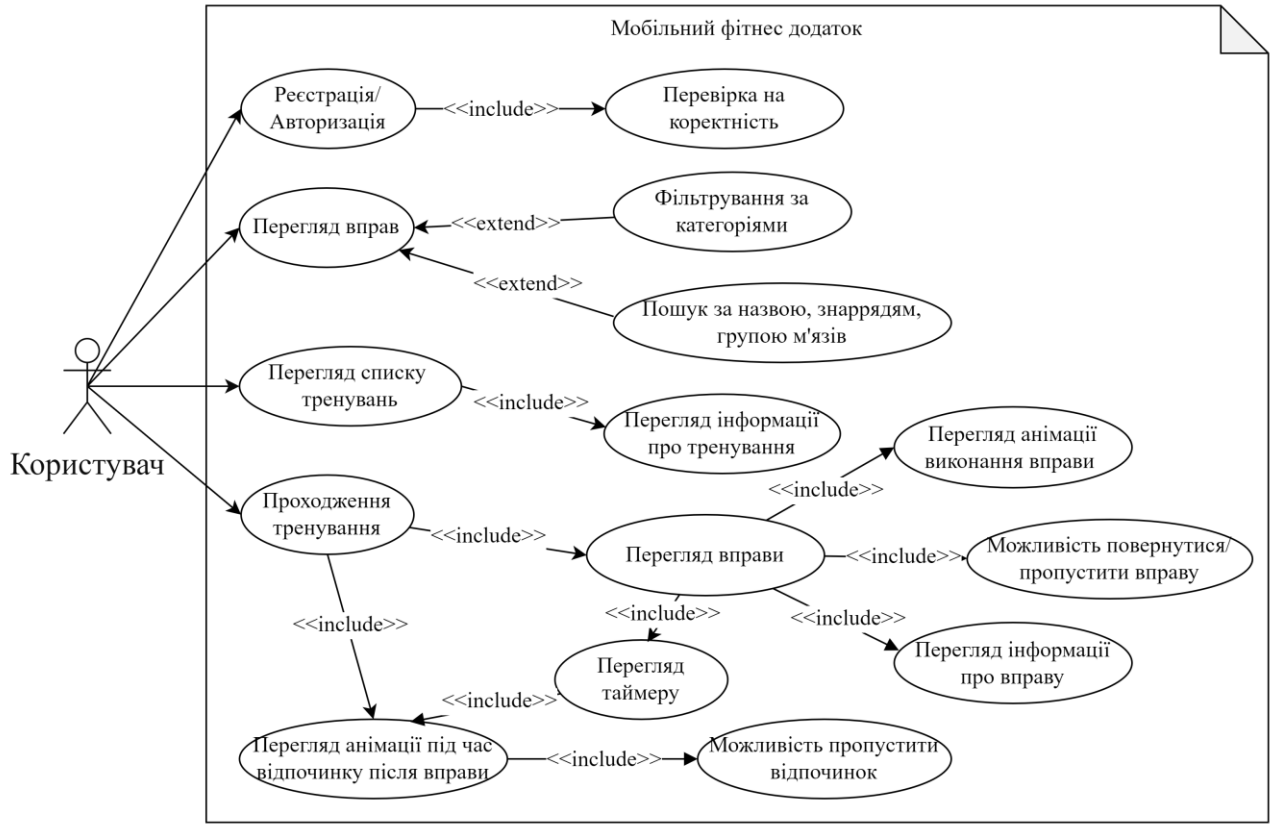


Рисунок 1.6 – Діаграма ВВ Користувача

Під час перегляду списку доступних тренувань, користувач має можливість перейти до екрану з більш детальною інформацією про кожне з них. Тому відношення між варіантами використання «Перегляд списку доступних тренувань» та «Перегляд інформації про тренування» позначено спеціальним стереотипом «include».

Якщо програма обраного тренування влаштовує користувача, він може запустити її виконання, це відображує прецедент «Проходження тренування». Під час тренування користувач бачить анімацію виконання вправи та може спостерігати за таймером, що показує час до завершення вправи. Крім того, за необхідності, користувач може переключатися між вправами. Після кожної вправи користувач має час на відпочинок, це відображено на спеціальному екрані з релаксуючою анімацією та таймером, який також має кнопку пропустити відпочинок. Після успішного завершення тренування з'являється екран привітання.

Актор «Клієнт» – це користувач, який вже пройшов процедуру реєстрації та авторизувався у застосунку під своїм ім'ям. Тобто клієнт вже має обліковий запис для збереження персоналізованої інформації: власних тренувань, улюблених вправ та активності (історії тренувань). Діаграма варіантів використання актора «Клієнт» подана на рисунку 1.7.

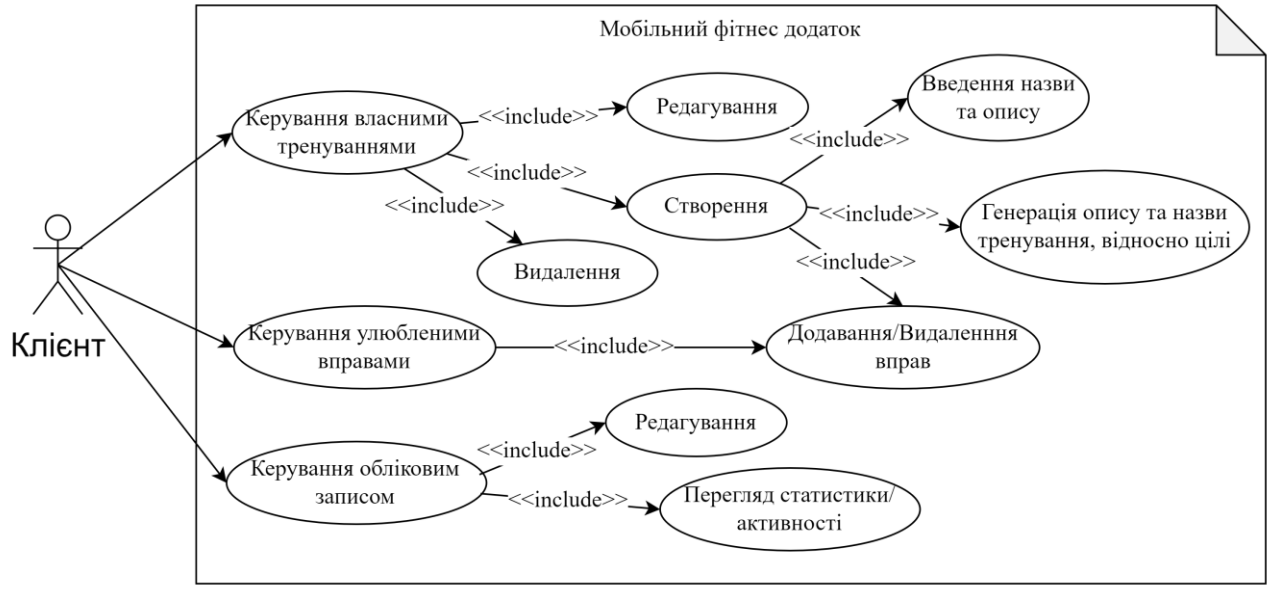


Рисунок 1.7 – Діаграма ВВ клієнта

Оскільки актор «Клієнт» може робити все, що робить актор «Користувач», між ними показано відношення узагальнення.

Відповідно, загальна готова модель варіантів використання мобільного фітнес застосунку «GymRat» подана у додатку А.

Отже, функціональне призначення мобільного фітнес застосунку «GymRat» включає в себе наступні можливості для користувачів:

– перегляд різноманітних вправ та стандартних тренувань;

– проведення тренувань за вибором користувача;

– збереження улюблених вправ для швидкого доступу;

– створення власних тренувань, використовуючи доступні вправи;

– генерація опису (порад) та назви тренування, під потреби користувача, використовуючи штучний інтелект;

– моніторинг та статистика проходження тренувань для відстеження прогресу авторизованого користувача.

На основі проведеного аналізу вимог до мобільного застосунку було розроблене технічне завдання, яке подано у додатку Б.

У даному розділі кваліфікаційної роботи було проведено аналіз завдань, які вимагають автоматизації з використанням інформаційних технологій, а також описано проблему, яка буде вирішена майбутнім програмним продуктом. Було обрано гібридний тип для майбутнього застосунку та визначено ключові переваги використання мобільних застосунків для занять фітнесом. Були розроблені функціональної моделі застосунку «GymRat» використовуючи стандарт IDEF0. Далі був проведений порівняльний аналіз двох відомих застосунків – Nike Training Club (NTC) та Jefit. Описано їх функціональні особливості, переваги та недоліки. На основі цього аналізу було сформульовано вимоги до програмного забезпечення. Було виділено акторів модельованої системи, їх варіанти використання та зв’язки між собою. Результатом цього розділу є розроблене технічне завдання.

2 ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСТУНКУ

2.1 Проєктування архітектури та структури системи

При проєктуванні мобільного застосунку, та системи в цілому, важливо визначитися з типом архітектури, яка найкраще відповідає вимогам та потребам проєкту [19].

В мобільних застосунках стає все більш популярною безсерверна архітектура через свою здатність зосереджувати увагу розробників на функціональності застосунку, не турбуючись про управління серверами та інфраструктурою [20]. У такій архітектурі всі серверні обчислення та обробка даних відбуваються в хмарних сервісах, а не на власних серверах розробника. Це означає, що розробники можуть спрямувати свій час і зусилля на розробку функціоналу застосунку, а не на вирішення питань інфраструктури та масштабування. Крім того, безсерверна архітектура дозволяє ефективно використовувати ресурси, оскільки витрати виникають лише під час виконання програми, а завдання масштабування виконуються хмарним провайдером.

Основні переваги безсерверної архітектури полягають у забезпеченні оптимального середовища для ефективного функціонування [21]:

– ефективне використання ресурсів: генерація опису тренувань штучним інтелектом під потреби користувача є витратним процесом, який повинен відбуватись на стороні сервера та працювати лише після запуску тригера, для економії коштів;

– швидкість розробки та надійність: використання готового сервісу для авторизації та аутентифікації користувачів дозволить значно збільшити швидкість розробки і зменшити ймовірність помилок, що підвищить надійність застосунку;

– гнучкість та розширюваність: використання інших спеціалізованих сервісів інших спеціалізованих сервісів, таких як база даних з вправами та їх

фільтрація, дозволить швидко і просто це інтегрувати без необхідності витрачати час на створення та підтримку власної бази даних;

– централізоване зберігання та доступність даних: можливість зберігання власних даних у хмарному сервісі забезпечить доступ до них з будь-якого пристрою та забезпечить їх безпеку та надійність.

Недоліки безсерверної архітектури [21]:

– потенційні проблеми з безпекою даних: при використанні не надійного хмарного постачальника, зберігання даних в хмарних сервісах може створювати ризик порушення конфіденційності та безпеки даних;

– складність управління версіями: при відсутності прямого контролю над інфраструктурою можуть виникнути проблеми з сумісністю версій інтерфейсів та API;

– потенційні проблеми з продуктивністю: неправильне налаштування сервісу, де зберігатиметься натренована модель генерації опису тренувань штучним інтелектом, може призвести до затримки відповідей або навіть відмову у виконанні завдання через перевищення ліміту;

– складність відлагодження та тестування: безсерверна архітектура може ускладнити процес відлагодження та тестування, особливо коли виникають проблеми з інтеграцією між різними безсерверними сервісами.

Отже, вибір безсерверної архітектури був зумовлений перевагами цього підходу, а недоліки виявилися незначними для даного проєкту, що підтверджує такий вибір.

На рисунку 2.1 зображений загальний варіант безсерверної архітектури для мобільного застосунку «GymRat». Застосунок має взаємодіяти з різними сервісами, що розміщені у хмарі. Сервіс аутентифікації відповідає за перевірку ідентичності користувача та надання доступу до застосунку. Сервіс для отримання вправ забезпечує доступ до різноманітних вправ для тренувань, які зберігаються в базі даних. Цей сервіс також повинен надавати можливість фільтрації цих вправ за різними критеріями, наприклад за групою м’язів та

наявність знаряддя. База даних для особистої інформації користувачів, подана у вигляді сервісу, так як зберігає і дані після реєстрації (ім'я, пошту, пароль), і додані улюблені вправи та створені власні тренування. Також у хмарі розміщена натренована модель для генерації опису тренувань на основі запиту користувача.

У разі потреби API Gateway [22] може бути використаний як посередник між мобільним застосунком та сервісами у хмарі, проте, залежності від обраних постачальників сервісів, цей елемент може бути необов'язковим.

Деякі компоненти, такі як база даних для статичних файлів, можуть бути розміщені локально для заощадження коштів.

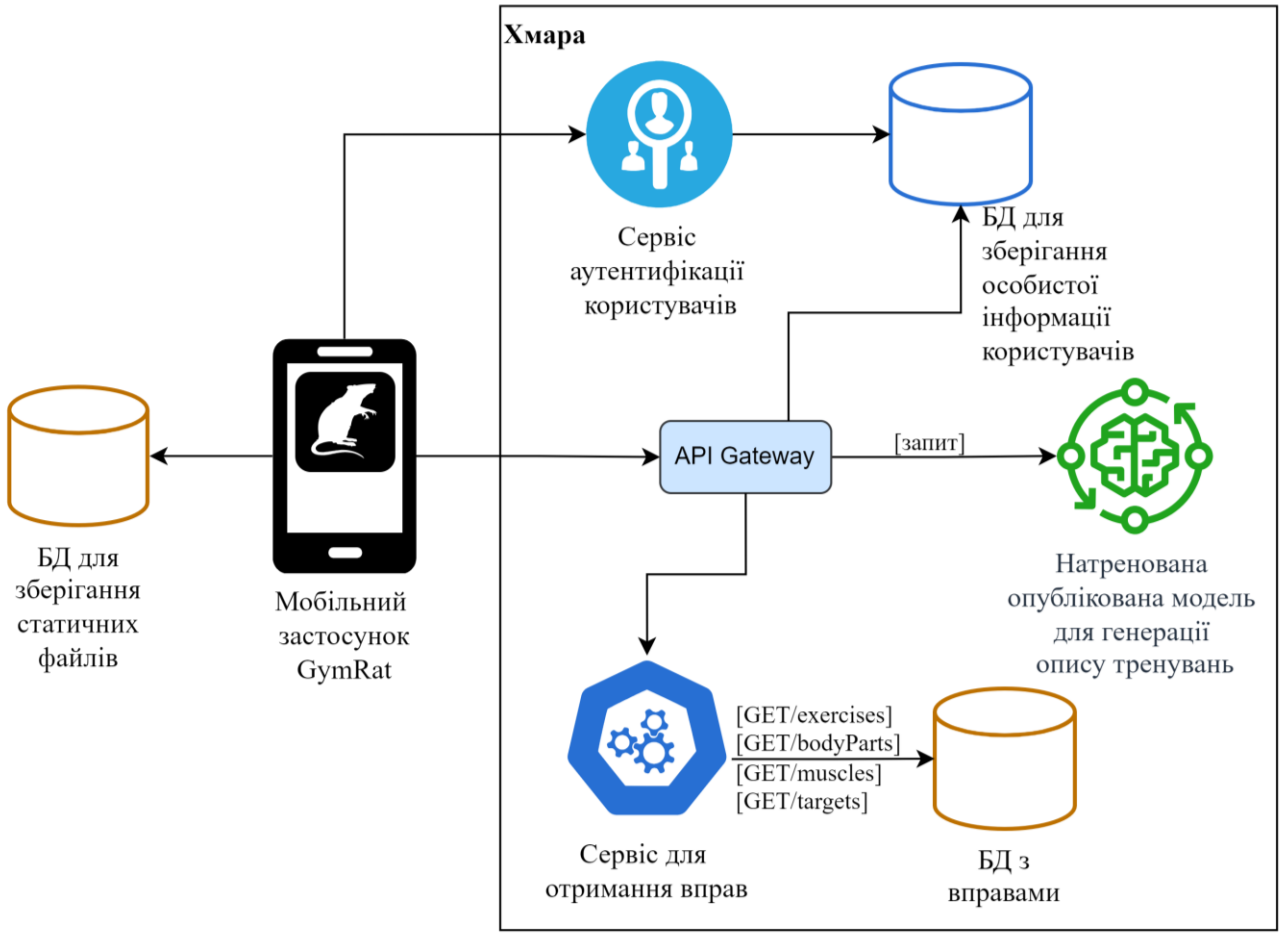


Рисунок 2.1 – Загальна діаграма архітектури системи

Тепер можна перейти до проєктування архітектури самого мобільного застосунку. У розробці мобільних застосунків на React Native [23] зазвичай

використовується компонентна архітектура. Це означає, що застосунок будується з невеликих, незалежних компонентів, які можна повторно використовувати та комбінувати для створення складних інтерфейсів. Кожен компонент відповідає за свою власну частину інтерфейсу або за певну функціональність. Це спрощує розробку, тестування та розширення застосунку, оскільки код розділяється на логічні блоки.

Управління станом також є важливим аспектом при проєктуванні архітектури для застосунку на React Native або React, два з основних підходів [24] – Flux та Redux.

Flux (рис. 2.2), побудований на основі концепції CQRS (Command and Query Responsibility Segregation), складається з компонентів: Store (сховища), Action (дії), View (подання) та Dispatcher (диспатчера). Діяльність в Flux відбувається у вигляді однонаправленого потоку даних, де дія користувача створює Action, який змінює стан у Store через Dispatcher.

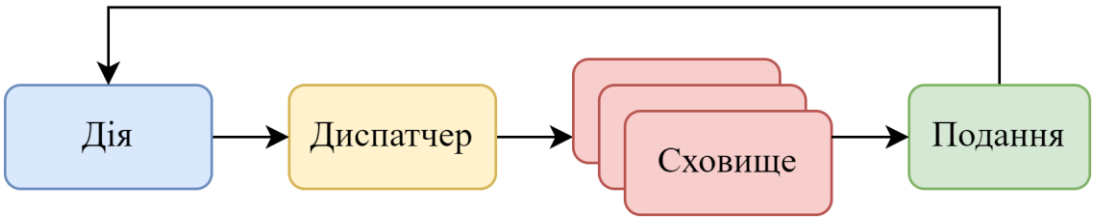


Рисунок 2.2 – Діаграма потоку даних з використанням Flux

Redux (рис. 2.3), спрощена версія Flux, використовує поняття Reducer і централізованого Store. Дія користувача створює Action, який передається в Reducer, що змінює стан у Store, а зміни у Store автоматично оновлюють View.

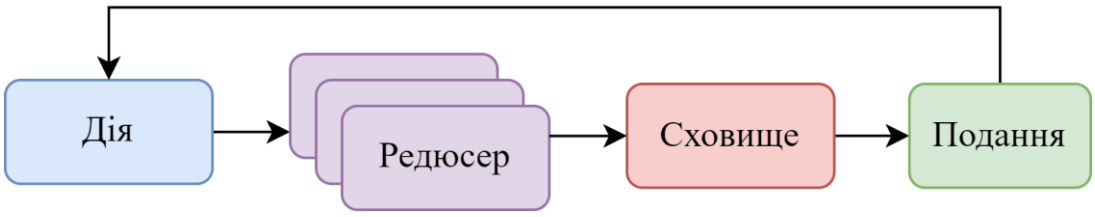


Рисунок 2.3 – Діаграма потоку даних з використанням Redux

Основні відмінності між Flux і Redux полягають у роботі зі Store [25]: Flux використовує багато Store, тоді як Redux має лише один централізований Store. Це впливає на простоту управління даними: у Flux потрібно враховувати декілька Store, тоді як у Redux зміни відбуваються через Reducer (редюсер), що працює з централізованим Store. Під час відлагодження Flux використовує Dispatcher для обробки всіх дій, що спрощує цей процес, тоді як у Redux відлагодження відбувається за допомогою централізованого Store, що спрощує виявлення проблем.

Отже, для управління станом застосунку «GymRat» було обрано Redux через його простоту та ефективність, він надає централізоване сховище, що спрощує роботу з даними та дозволяє з легкістю відстежувати їх зміни.

2.2 Проєктування логічної моделі бази даних

Наступним етапом є формування логічної моделі бази даних [26]. На основі досліджень, проведених у попередньому підрозділі був проведений аналіз потенційних сутностей та їх атрибутів:

– User (користувач) – явний кандидат на сутність;
– UserWorkouts (власні тренування) – явний кандидат на сутність;
– FavoriteExercises (улюблені вправи) – явний кандидат на сутність.

Зв’язок між цими сутностями (рис. 2.4): користувачі можуть створювати власні тренування та зберігати улюблені вправи.

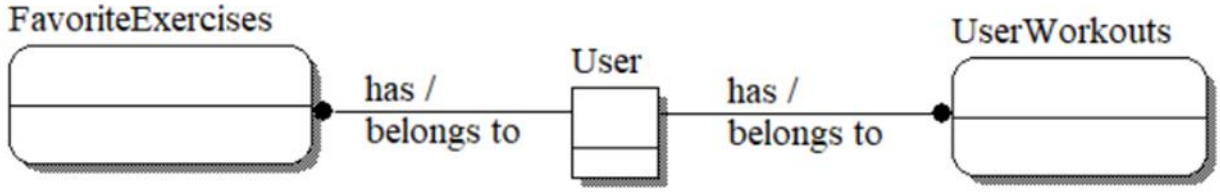


Рисунок 2.4 – Зв’язок між сутностями «User», «UserWorkouts» та «FavoriteExercises»

Зв’язок один-до-багатьох між сутностями User та UserWorkouts, так як кожен користувач може мати будь-яку кількість власних тренувань, та між User та FavoriteExercises, аналогічно, кожен користувач може мати будь-яку кількість улюблених вправ. Зв’язок багато-до-багатьох між UserWorkouts і Exercises, оскільки кожне тренування може містити будь-яку кількість вправ і кожна вправа може відноситися до декількох тренувань одночасно. FavoriteExercises складаються зі звичайних вправ, тому зв’язок буде один-до-багатьох.

Таким чином, після уточнення, вигляд діаграми буде наступним, уточнений варіант діаграми можна побачити на рисунку 2.5.

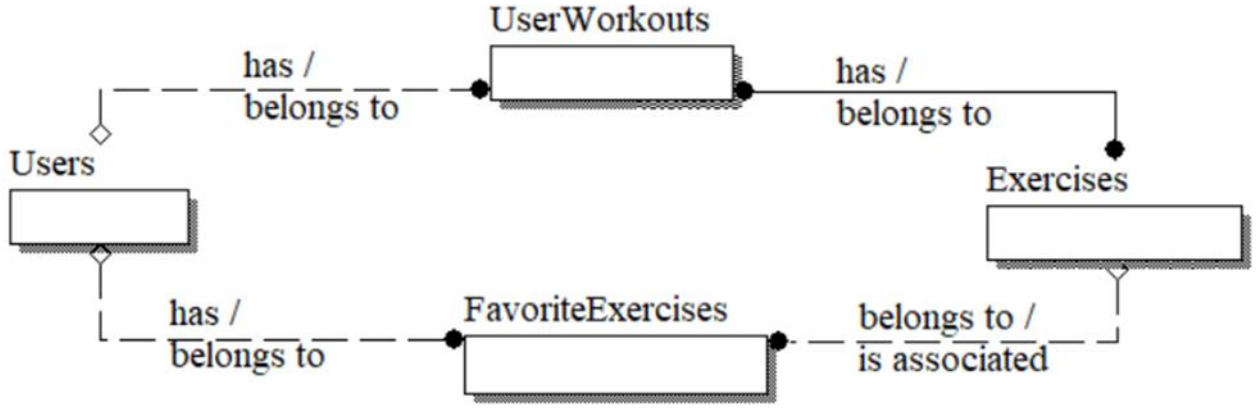


Рисунок 2.5 – Уточнена ER-діаграма

Атрибути вправи залежатимуть від постачальника (сервісу з вправами), але мінімальний бажаний набір є наступним: унікальний ідентифікатор вправи (exercise_id), назва вправи (name), частина тіла, для якої призначена вправа (bodyPart), цільова група м’язів (target), обладнання необхідне для виконання вправи (equipment), URL-адреса GIF-зображення, що демонструє правильне виконання вправи (gifUrl).

Також, було додано ще одну сутність для моніторингу UserActivities прогресу користувача. В ній будуть зберігатись цільові м’язи (targets), час виконання (timestamp) та назва виконаного тренування (workoutName).

При створенні власного тренування, користувач вказує назву тренування (name), додає його опис (description) та фотографію (image) для асоціації.

При додаванні вправи до тренування, користувач обирає вправу зі списку доступних (exercise_id), вказує час на виконання вправи (time) та час на відпочинок (rest).

При додаванні вправи (exercise_id) до списку улюблених автоматично додається дата додавання вправи до улюблених (starredAt) та порядок вправи у тренуванні (order).

Унікальні ідентифікатори мають створюватись автоматично, без втручання користувача, для забезпечення цілісності.

Для побудови моделі «сутність-зв'язок» використано нотацію IDEF1X. IDEF1X є стандартом, який включає мову для опису структури і семантики інформаційних моделей, а також пов'язані з мовою правила і методи розробки логічних моделей даних. Найчастіше така методологія використовується для опису даних в цілях подальшої автоматизації їх обробки за допомогою систем управління базами даних.

Готову модель «сутність-зв'язок» зображено на рисунку 2.6.

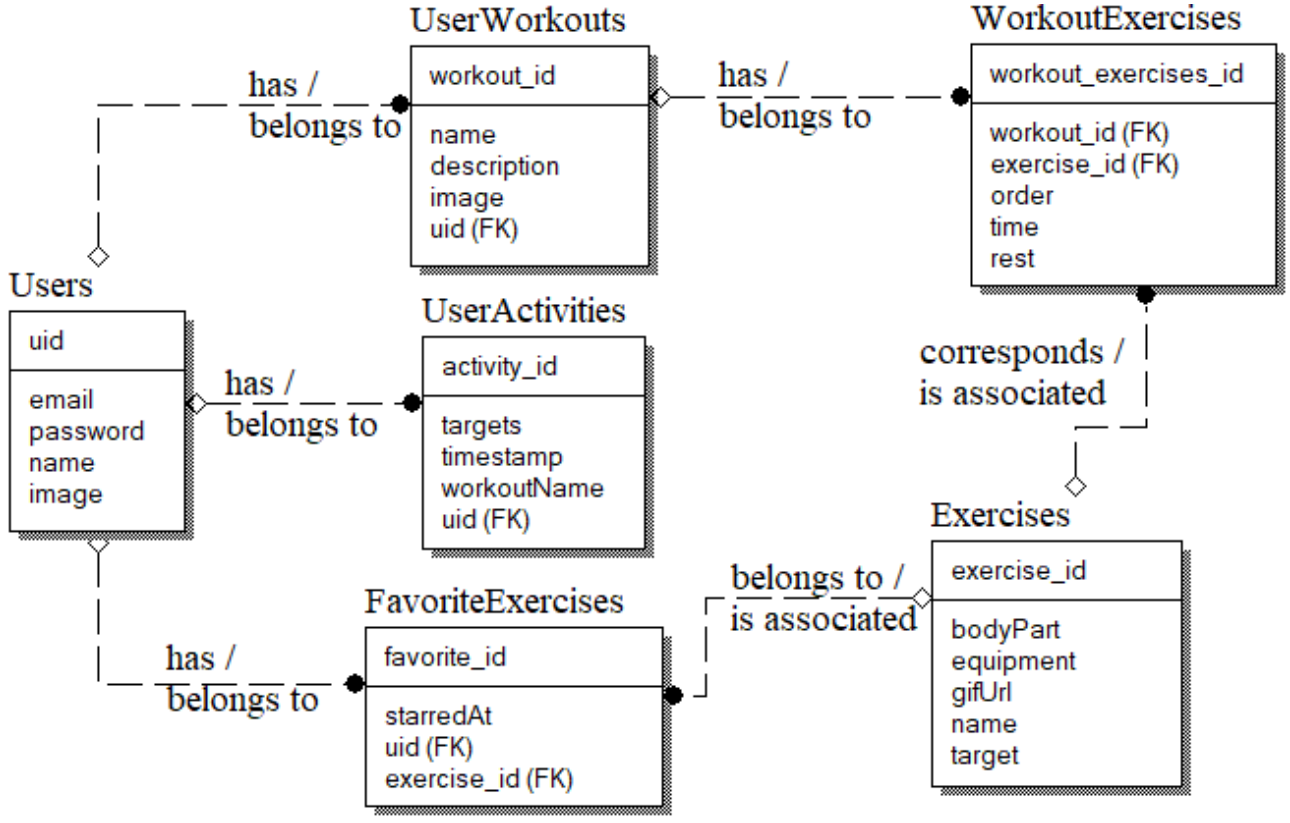


Рисунок 2.6 – ER-діаграма

Розроблена ER-діаграма є прикладом концептуальної діаграми, тобто вона не враховує особливості конкретної системи керування базою даних (СКБД). Тому в наступному розділі з цієї концептуальної діаграмі буде побудована фізична діаграма з урахуванням особливостей обраної СКБД.

2.3 Проєктування інтерфейсу користувача

Для оформлення дизайну мобільного застосунку для тренувань було обрано три основних кольори [27] – чорний, сірий та жовтий, які відображають мінімалістичний та стильний підхід до дизайну.

Використання такого спектру кольорів сприяє збалансованому та лаконічному візуальному враженню, а також надає застосунку сучасний вигляд. Чорний та сірий кольори створюють основу для інтерфейсу, підкреслюючи його стриманий характер, а жовтий додає яскравий акцент, який привертає увагу до важливих елементів інтерфейсу.

Матеріальний дизайн інтерфейсу [28] відзначається своєю легкістю та відсутністю перевантаження деталями. Цей підхід сприяє простоті та зрозумілості взаємодії користувача із застосунком, роблячи його використання приємним та ефективним.

Навігаційне меню складається з чотирьох пунктів:

– головний екран «Home»;

– список вправ «Execises: (category: all)», тобто за замовчуванням буде обрана категорія з усіма вправами;

– список тренувань «Workouts»;

– обліковий запис користувача «Profile».

Перше враження про продукт у користувача формує «Екран-заставка» (Splash Screen). Тому при відкриті застосунку з’являється (рис. 2.7):

– логотип: Gym<RatIcon>Rat;

– коротка інформація про застосунок: «Keep track of your workouts and achieve your fitness goals!»;

– анімація: зверху виїжджає логотип, знизу – текст;

Екран авторизації (рис. 2.7) – відповідає за аутентифікацію користувачів. На цьому екрані користувач може авторизуватися у застосунку. Передбачена можливість користування застосунком без авторизації, тобто анонімно, за посиланням унизу, але це не рекомендовано, так як багато функцій є недоступними, про це користувач дізнається з повідомлення, коли пробує активувати будь яку з них. Склад екрану авторизації:

– логотип: Gym<RatIcon>Rat;

– мотиваційний текст «Transform your body, transform your life. Let this fitness app be your guide on the journey to a healthier and happier you.»;

– поля для введення: email, password (містить іконку для відображення прихованого тексту);

– кнопки та посилання: «Login» для входження у застосунок під своїм обліковим записом, «Don’t have an account? Create one here» для переходу на екран реєстрації, «Use app without authorization» для входження під анонімним обліковим записом:

Екран реєстрації (рис. 2.7) – містить форму для створення облікового запису. Склад екрану реєстрації:

– логотип: Gym<RatIcon>Rat;

– мотиваційний текст: «Exercise is a celebration of what your body can do. Not a punishment for what you ate.»;

– поля для введення: name, email, password, confirm password (password, confirm password повині містити іконку для відображення прихованого тексту);

– кнопки: «Sign up»: для створення облікового запису, Registered succssesfully? Go back to Login» для повернення на екран авторизації;

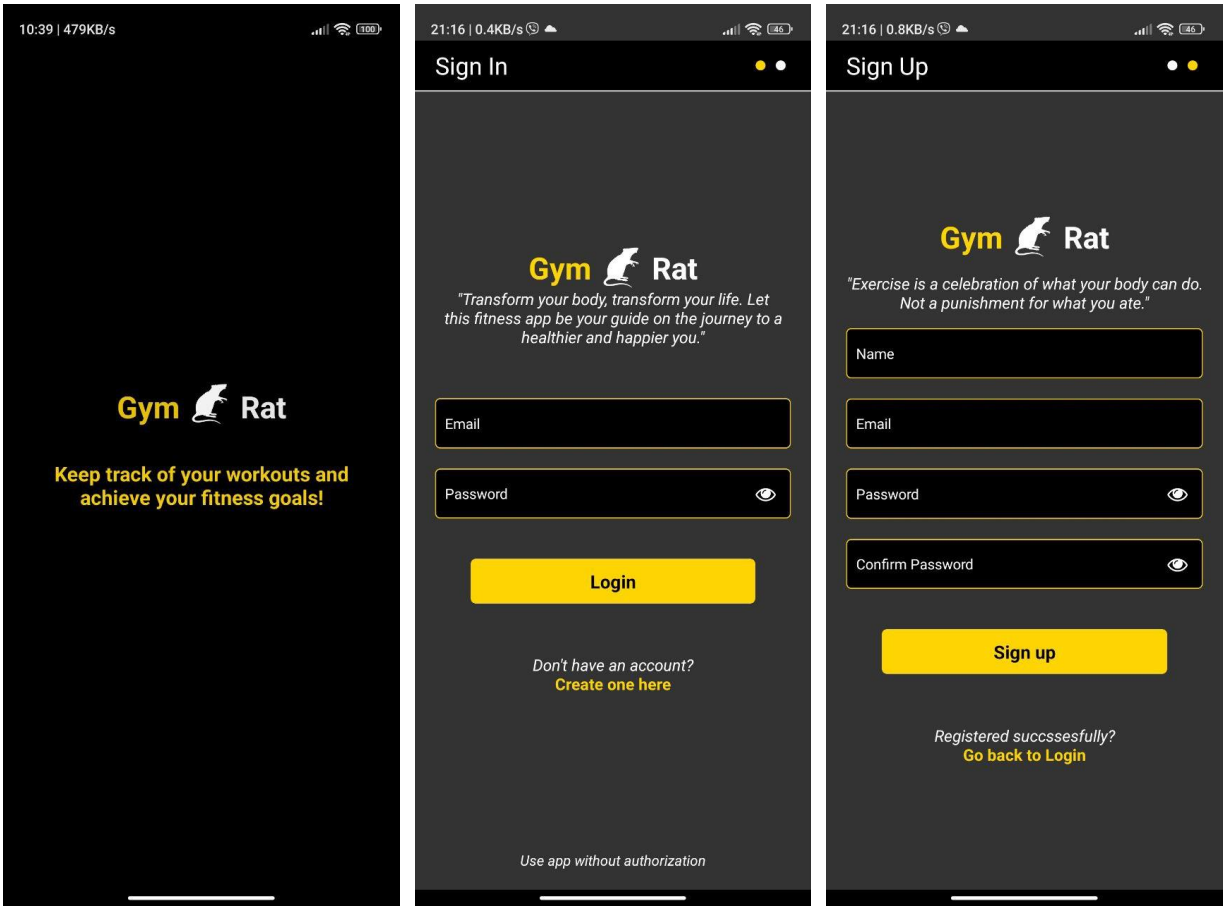


Рисунок 2.7 – «Екран-заставка», екран авторизації та екран реєстрації

Головний екран (рис. 2.8) – відображає список категорій вправ, поділених за цільовою частиною тіла. Склад головного екрану:

– шапка: назва – «Choose Category», активний розділ екрану 1);

– список категорій, кожна категорія складається з назви та картинки категорії;

– підвал: навігаційне меню.

Екран вправ (рис. 2.8) – відображає список доступних вправ. Якщо користувач є авторизованим, то на цьому екрані також має можливість додавати вправи до списку улюблених.

Склад екрану з доступними вправами:

– шапка: назва – кнопка повернення, «Exercise: <category_name>», dot activeDot;

– пагінація: кнопки переходу на різні сторінки списку;

– п’ять вправ, кожна вправа складається з анімації виконання, назви вправи та кнопки додавання вправи до улюблених (при авторизації);

– підвал: навігаційне меню.

Екран з улюбленими вправами – відображає список вправ, які були додані користувачем до списку улюблених. Відрізняється від екрану з вправами лише назвою в шапці екрану – «Favorite exercises».

Екран списку тренувань – відображає список тренувань.

Склад екрану (рис. 2.8) з доступними тренуваннями:

– шапка: назва – «Choose Workout», активний розділ екрану 1;

– тренування: картинка, назва та опис тренування;

– підвал: навігаційне меню.

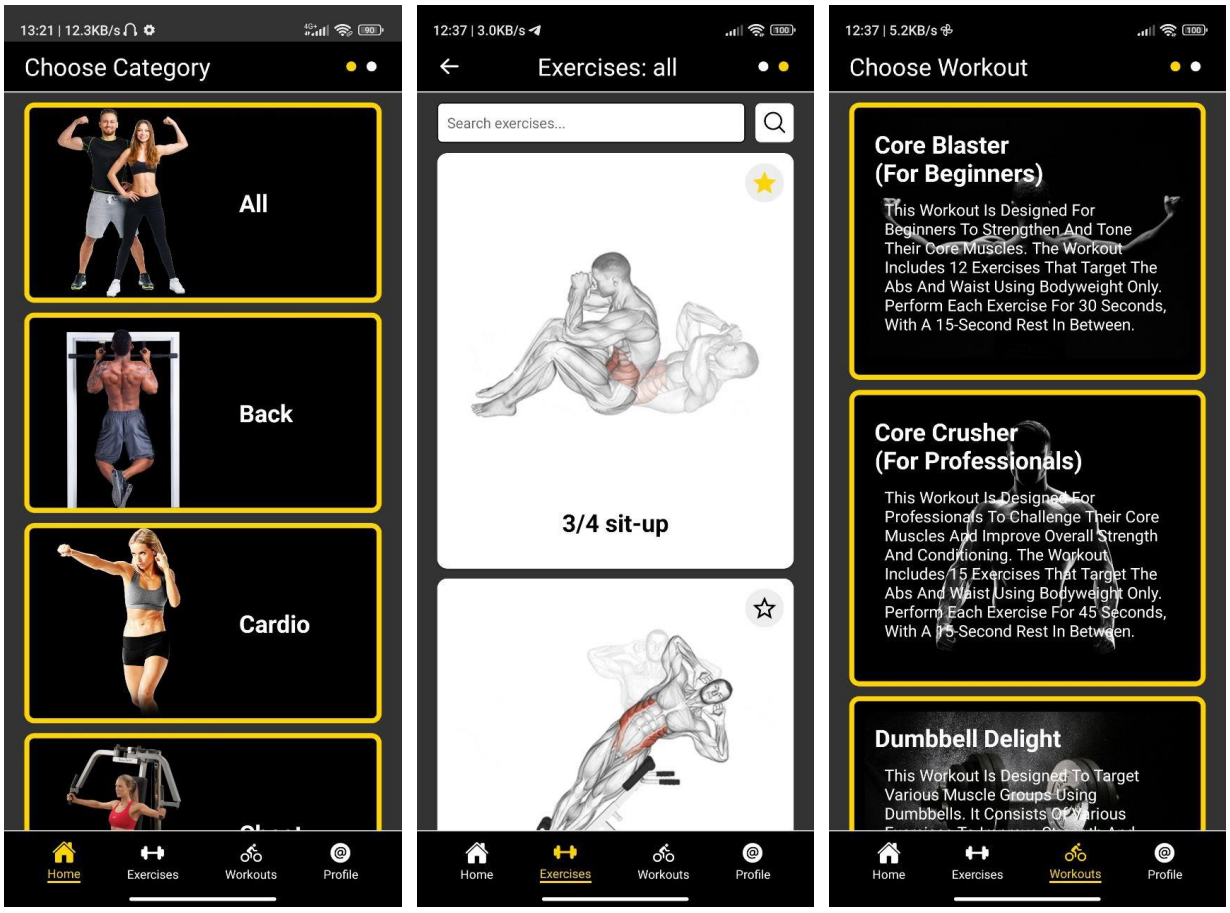


Рисунок 2.8 – Головний екран, екран з вправами та екран з тренуваннями

Екран тренування (рис. 2.9)– відображає інформацію про обране тренування. Склад екрану з інформацією про тренування:

– шапка: кнопка повернення, назва – «Workout #<id>», активний розділ екрану 2;

– інформація про тренування: назва, опис;

– кнопка почати тренування – «Play Workout <icon>»;

– список вправ у тренування: номер, назва, цільова категорія м’язів;

– кругова діаграма для кращого розуміння цільової категорії м’язів;

– підвал: навігаційне меню.

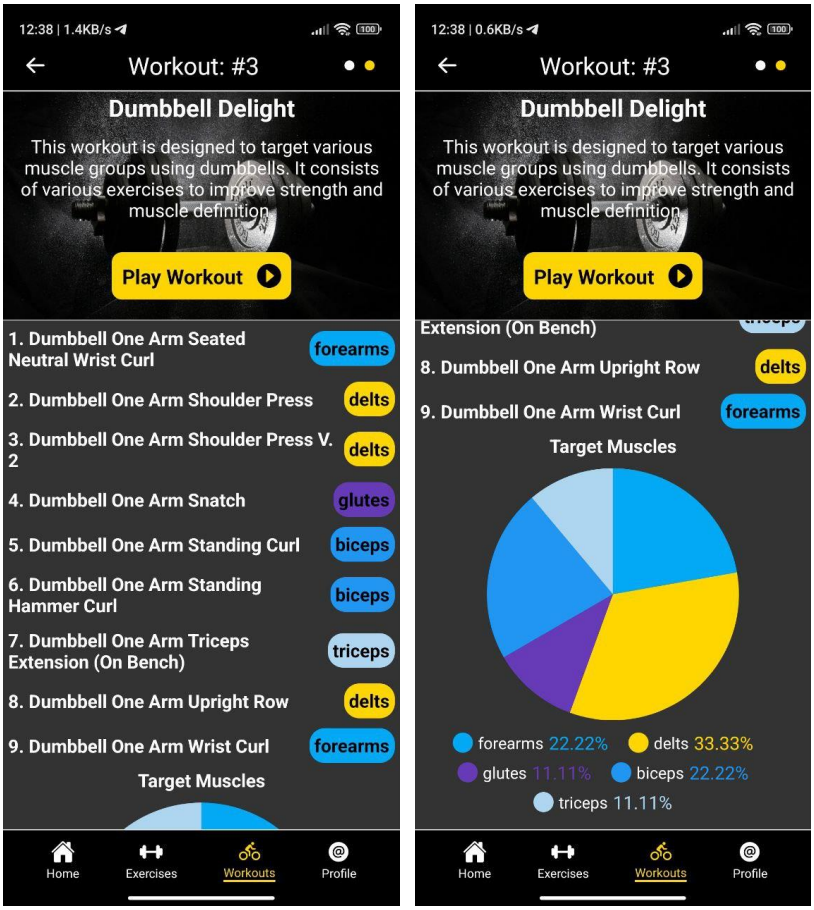


Рисунок 2.9 – Екран про тренування

Екран вправи під час тренування (рис. 2.10) – відображає по черзі вправи з тренування та містить:

– про вправу: анімацію виконання вправи, назву вправи, опис виконання, номер вправи в тренуванні;

– таймер з анімацією.

– кнопки: повернення до попередньої вправи – «<icon> Prev» та перехід до наступної вправи – «Next <icon>»;

Екран відпочинку (рис. 2.10) – дозволяє користувачам відпочити декілька секунд перед наступною вправою. Відкривається автоматично після кожної вправи під час тренування. Склад екрану:

– анімація: текст «Break! Break! Break!», картинка з анімацією ходьби;

– назва екрану: «Rest Time»;

– таймер;

– кнопка «Skip» знехтувати відпочинком.

Екран привітання (рис. 2.10)– піднімає настрій після завершенням тренування. Склад екрану:

– привітальний текст: «Congratulations! You have completed your workout.»;

– кнопка повернення: «Return!».

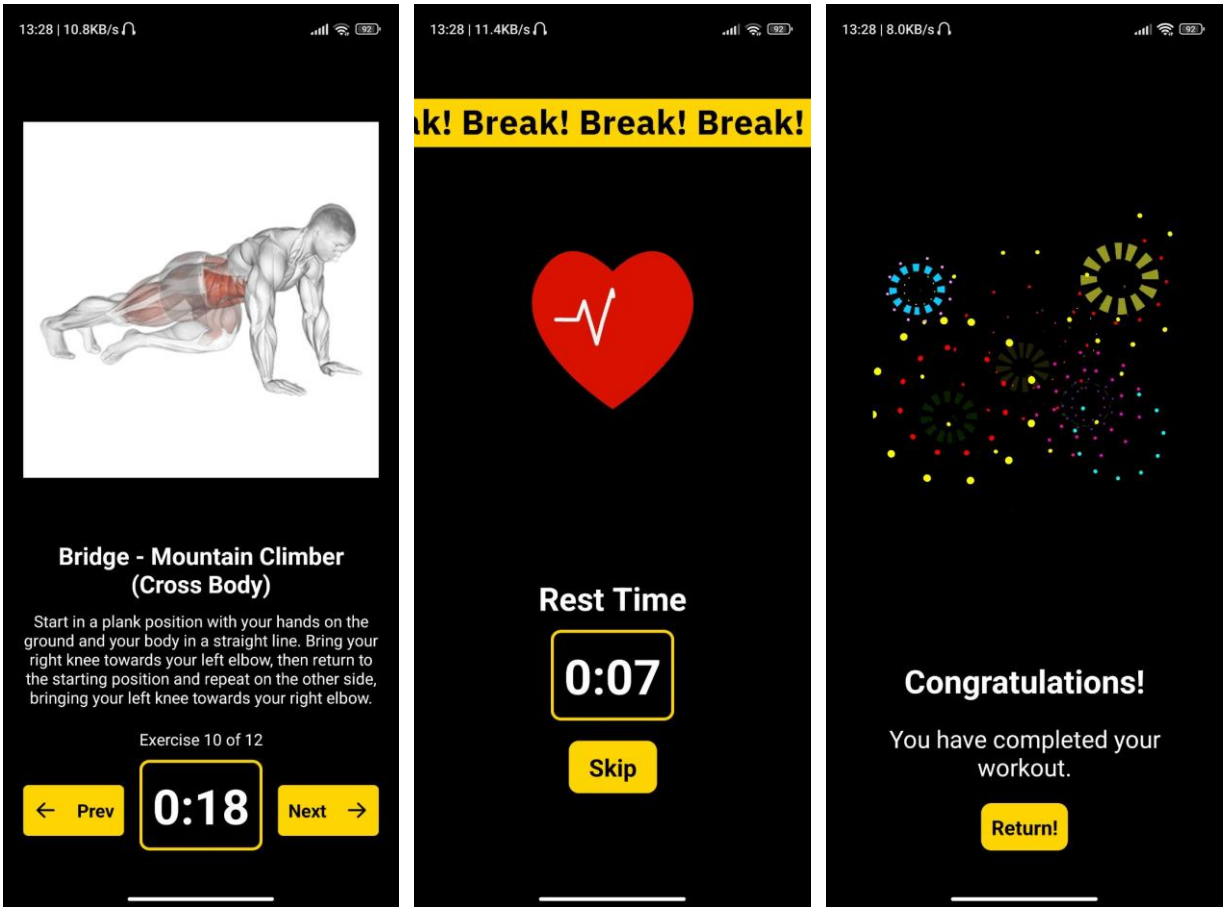


Рисунок 2.10 – Екрани для проходження тренування

Екран облікового запису користувача (рис. 2.11) – відображає інформацію про профіль користувача. Склад екрану облікового запису користувача:

– шапка: назва – «Profile», активний розділ екрану 1);

– інформація: фото, електронна адреса, ім’я, кількість завершених тренувань, кількість улюблених вправ, кількість створених тренувань;

– навігаційне меню: список улюблених вправ «Favorites», список власних тренувань «My Workouts», календар виконання тренувань «Activity Calendar», про застосунок «About App»;

– кнопки: редагування імені користувача, вихід «Logout»;

– підвал: навігаційне меню.

Екран календар активності (рис. 2.11) містить інформацію щодо кількості та складу тренувань користувача, тобто:

– шапка: назва – «Activity Calendar», активний розділ екрану 2);

– календар: сьогоднішній день виділений кольором, дні з тренуваннями позначені крапками;

– навігаційне меню: список улюблених вправ «Favorites», список власних тренувань «My Workouts», календар виконання тренувань «Activity Calendar», про застосунок «About App»;

– інформація про обраний день: заголовок «Workouts for #day», час виконання тренування, назва цього тренування, кругова діаграма цільової категорії м’язів тренування;

– підвал: навігаційне меню.

Екран власних тренувань (рис. 2.11) ідентичний до екрану зі списком тренувань, за вийнятком того, що до кожного тренування зі списку додаються кнопки для управління:

– створити тренування «Create Workout»;

– відредагувати тренування «Pen Icon»;

– та видалити тренування «Trash Icon».

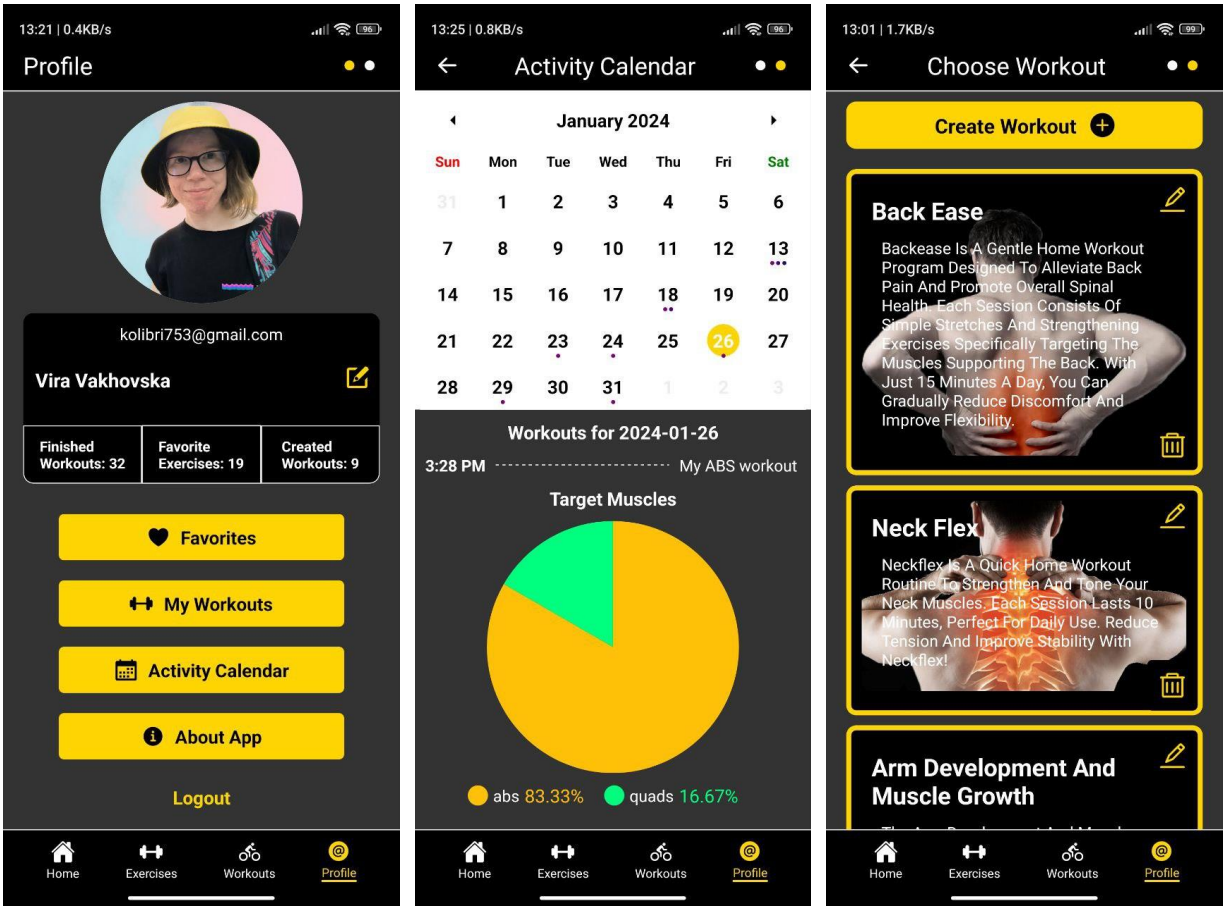


Рисунок 2.11 – Екран облікового запису, екран активності та власних тренувань

При натисканні кнопки створити тренування «Create Workout» користувач переходить на форму з такою ж назвою (рис. 2.12). Форма містить:

– заголовок «Create Workout»;

– необов’язкові поля: поле вибору зображення «Select an Image»;

– обов’язкові поля: назва тренування «Workout Name», опис тренування «Workout Description»;

– кнопка «Generate using AI» для генерації опису та назви тренування використовуючи штучний інтелект;

– підвал: навігаційне меню.

При натисканні на «Generate using AI» відкривається форма (рис. 2.12), де потрібно ввести цілі тренування та натиснути «Get AI-Designed Workout». Після завершення генерації «віртуальний тренер» автоматично введе згенерований текст у відповідні поля. Форма містить:

– заголовок «Generate using AI»;

– поле введіть цілі тренування (наведений приклад);

– кнопка «Get AI-Designed Workout» для отримання підтвердження та надіслання запиту до моделі.

Після створення тренування, до нього додаються вправи, використовуючи відповідну форму (рис. 2.12). Форма виглядає наступним чином:

– заголовок «Add |#Exercise| to your workout», назва вправи підставляється автоматично;

– поля: опис вправи «Workout Description», час тренування: «Time for Exercise», час відпочинку: «Rest Time».

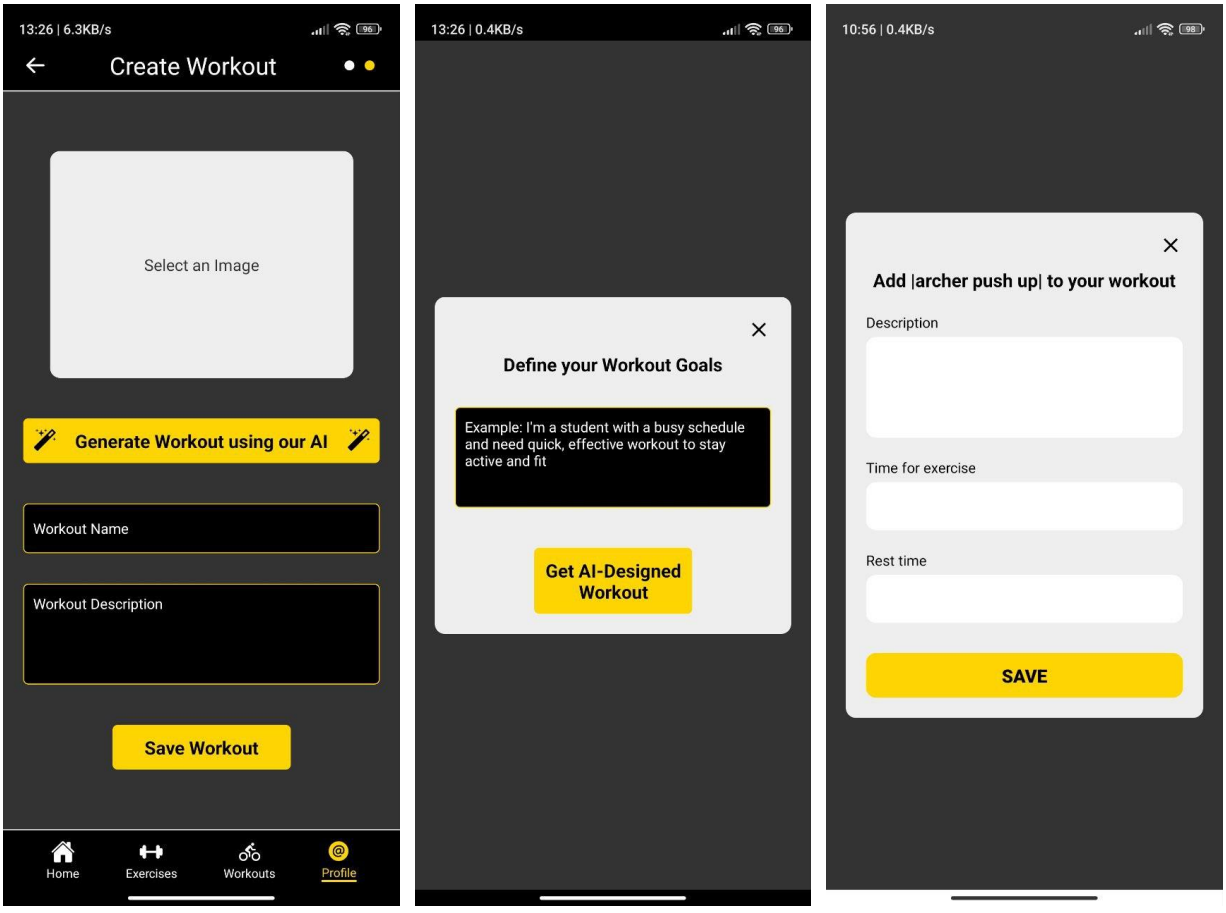


Рисунок 2.12 – Екран створення тренування, форма визначення цілей для генерації штучним інтелектом та додавання вправи до тренування

На рисунку 2.13 подано повну схему переходів між екранами застосунку. Для кращої візуалізації було вирішено зафарбувати сумісні екрани одним

кольором, виділити пунктирною лінією навігаційні елементи, всередині яких можна переходити на будь-який екран у групі, та позначити синім кольором кроки, які доступні лише авторизованому користувачу. Стартовою точкою є «Екран заставка».

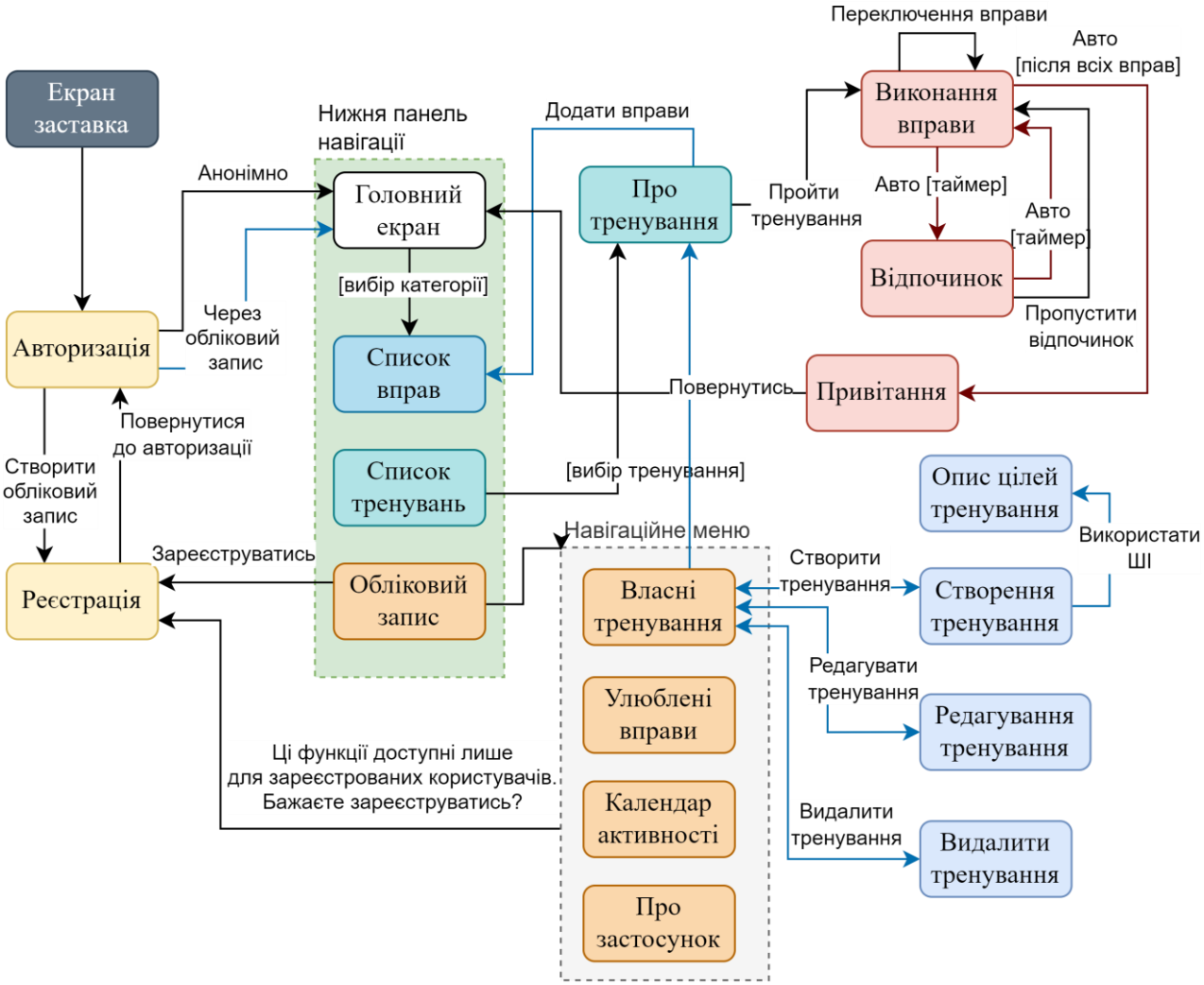


Рисунок 2.13 – Схема переходів між екранами застосунку

2.4 Аналіз та вибір технологій і методів реалізації застосунку

У розділі 1.1 вже було обрано тип мобільного застосунку – гібридний, для охоплення більш ширшої аудиторії. Для реалізації цього було обрано технологію React Native [29], що є однією з передових платформ для створення

гібридних мобільних застосунків. Вона дозволяє писати мобільний код на JavaScript [30], використовуючи React-подібні компоненти, які потім перетворюються в нативний код під конкретну мобільну платформу. Це дозволяє зберегти час і зусилля під час розробки, а також забезпечити високу швидкість роботи самих застосунків.

Для подальшого спрощення розробки та розгортання застосунку було вирішено скористатися платформою Expo. Expo [31] – це набір інструментів і сервісів, який дозволяє розробникам створювати, тестувати та розгортати мобільні застосунки на базі React Native без необхідності налаштування складних середовищ розробки. Використання Expo дозволяє отримати доступ до різноманітних корисних інструментів, таких як швидка збірка, миттєве оновлення додатків та можливість тестування на реальних пристроях, навіть без власного обладнання.

У розділі 2.1 було прийнято рішення використовувати Redux для управління станом застосунку. Для забезпечення більшої ефективності та спрощення роботи з Redux, було обрано Redux Toolkit [32] – це офіційний інструментарій, розроблений командою Redux, який надає ряд корисних утиліт для зручного та ефективного використання його в застосунках. Цей пакет обгортає базовий пакет Redux і містить методи API та загальні залежності, які є необхідними для побудови такого типу застосунку.

Для забезпечення ефективного та надійного зберігання даних у хмарі було вирішено скористатися платформою від Googe – Firebase. Firebase [33] – це платформа від Google, яка надає різноманітні інструменти для розробки мобільних та веб-застосунків. Буде використаний такий набір інструментів:

– Authentication, дозволить легко налаштувати та використовувати систему автентифікації користувачів;

– Firebase Database, основна база даних, використовуватиметься для зберігання всієї персоналізованої інформації, про яку йшлося в розділі 2.2;

– Realtime Database буде використовуватися для зберігання і доступу до стандартних тренувань, які будуть доступні як авторизованим, так і анонімним користувачам, цей сервіс дозволить легкий експорт і імпорт тренувань у форматі JSON;

– Storage буде використовуватися для зберігання статичних файлів застосунку, наприклад зображень категорій м’язів.

Для генерації описів тренувальних програм використовуватиметься натренована модель LLama 2. LLama 2 [34] – це модель, яка використовує передові технології обробки природної мови (NLP) та здатна генерувати текст високої якості на запит користувача. Модель буде натренована на даних, представлених у форматі .csv, використовуючи середовище Google Colab, яке забезпечує можливість виконання Python коду у хмарі. Після тренування модель буде розміщено на Hugging Face Hub. Hugging Face [35] – це платформа для спільного використання моделей машинного навчання та NLP. Далі вона буде опублікована через Inference Endpoints, що забезпечить доступ до неї через API та дозволить легко взаємодіяти через застосунок.

Для отримання вправ обрано ExerciseDB з маркетплейс RapidAPI [36]. RapidAPI є найбільшим постачальником API у світі, завдяки йорму розробники можуть швидко знайти і використовувати різноманітні API від різних провайдерів без необхідності вручну шукати та інтегрувати їх з різних джерел. ExerciseDB надає доступ до понад 1300 вправ з індивідуальними даними про кожну вправу та анімованими демонстраціями. Цей API дозволяє отримувати інформацію про вправи з анімацією правильного виконання поділеними на категорії за частиною тіла, цільовими м’язами та необхідним обладнанням.

Отже, конкретизована діаграма архітектури системи подана на рисунку 2.14. В прямокутнику «Застосунок» представлені фреймворки та бібліотеки, що використовуються для створення інтерфейсу користувача та управління станом застосунку. У прямокутнику «Хмара» зображені хмарні сервіси, що надають доступ до додаткових функцій та даних для застосунку.

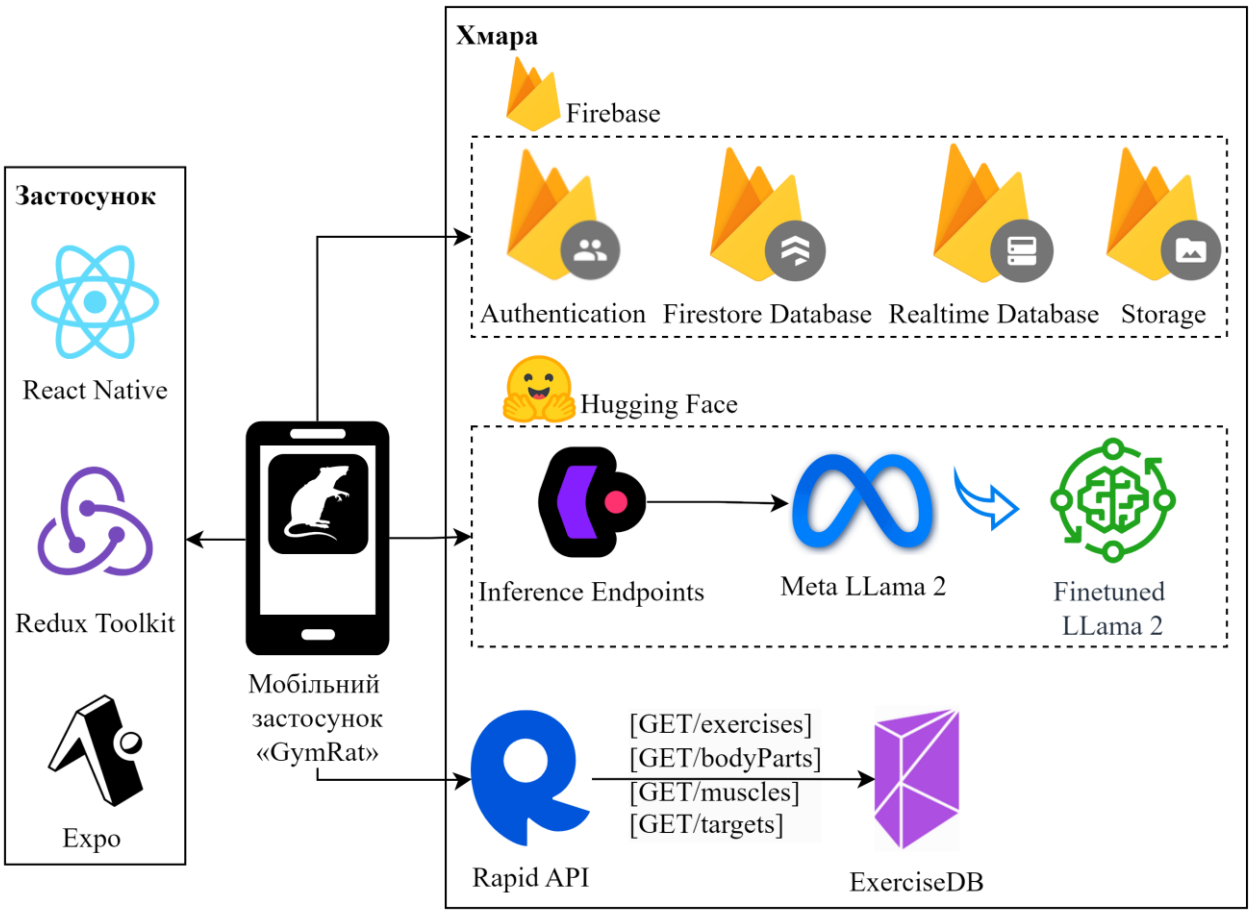


Рисунок 2.14 – Конкретизована діаграма архітектури системи

У цьому розділі кваліфікаційної роботи було вирішено застосувати безсерверну архітектуру для системи та обрано компонентну архітектуру з використанням Redux для управління станом застосунку. Крім того, були використана нотація IDEF1X для проєктування логічної моделі бази даних. Інтерфейс користувача був розроблений у стилі матеріального дизайну з використанням трьох основних кольорів: чорного, сірого та жовтого. Навігаційне меню та опис складу екранів та форм також були включені у цей розділ. У виборі технологій та методів реалізації застосунку було приділено увагу конкретним вимогам та потребам проєкту, що відображено у виборі сервісів на конкретизованій діаграмі архітектури системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація мобільного застосунку

При розробці мобільних застосунків на платформі React Native, одним із ключових понять є компоненти. Вони є основними структурними блоками кожного екрану застосунку, допомагаючи розділити його на менші, логічно зв'язані частини для полегшення процесів розробки, тестування та керування. Компоненти можуть бути організовані в ієрархічній структурі, де батьківський компонент містить дочірні компоненти і передає їм властивості (props) як аргументи, що робить компоненти застосунку більш гнучкими.

У React Native компоненти можуть бути класами, які успадковуються від базового класу React.Component, або функціональними компонентами. Основним файлом у проєкті на React Native є App.js, який виступає в якості точки входу в застосунок та відповідає за його ініціалізацію, структура та конфігурацію (налаштування).

У файлі App.js реалізована основна функціональність, включаючи створення компонента App (рис. 3.1). Зазвичай цей файл також містить налаштування та ініціалізацію різних бібліотек та модулів, таких як маршрутизація, керування станом застосунку (наприклад, Redux або MobX), налаштування теми, локалізація та інші налаштування.

Структура файлу App.js в даному застосунку:

– імпорти бібліотек та компонентів: React – JavaScript-бібліотека для створення інтерфейсів користувача, хуки useState та useEffect для зберігання стану та виконання побічних ефектів у функціональних компонентах, бібліотека навігації react-navigation, офіційний пакет react-redux для інтеграції Redux для керування станом застосунку та обміну даними між компонентами, @reduxjs/toolkit – набір інструментів для роботи з Redux, який спрощує процес створення та управління станом застосунку, useScreenLock – власний хук для блокування повороту екрану під час завантаження застосунку та бібліотека

react-native-root-siblings відображення модальних вікон поверх основного інтерфейсу застосунку;

– постачальник redux-сховища: на кожному рівні навігації (Stack.Navigator) встановлюється постачальник redux-сховища (Provider), щоб надати доступ до сховища всім дочірнім компонентам;

– конфігурація redux-сховище: після імпорту необхідних бібліотек і компонентів, налаштовується redux- сховище за допомогою функції configureStore (rootReducer);

– компонент App: функціональний компонент, у тілі якого реалізовано логіку відображення екранів застосунку та управління їх станом.

– ініціалізація стану завантаження: використовуючи хук useState, створено стан loading, який визначає, чи завантажений застосунок;

– відображення екрану завантаження: якщо застосунок ще завантажується (loading === true), то відображається екран завантаження (SplashScreen);

– головний вміст застосунку: після завантаження застосунку (loading === false), відображається головний вміст, який включає в себе навігаційний контейнер (NavigationContainer), в якому розташовані різні екрани (Stack.Navigator);

– реєстрація компонентів навігації: у Stack.Navigator додаються різні екрани застосунку, наприклад, ті, що не містять навігаційного меню знизу екрану: AuthorizationScreen, RegistrationScreen, WorkoutExerciseScreen та WorkoutCompleteScreen, і ті, що містять навігацію – Main(BottomTabNavigator): HomeScreen, ExercisesScreen, WorkoutStackNavigator (WorkoutsScreen, WorkoutExercisesScreen) та ProfileStackNavigator (ProfileScreen, FavoriteScreen, MyWorkoutsScreen, CreateWorkoutScreen, MyWorkoutExercisesScreen, UpdateWorkoutScreen, ActivityCalendarScreen).

– контейнер RootSiblingParent: забезпечує можливість відображення модальних вікон та повідомлень поверх основного інтерфейсу застосунку.

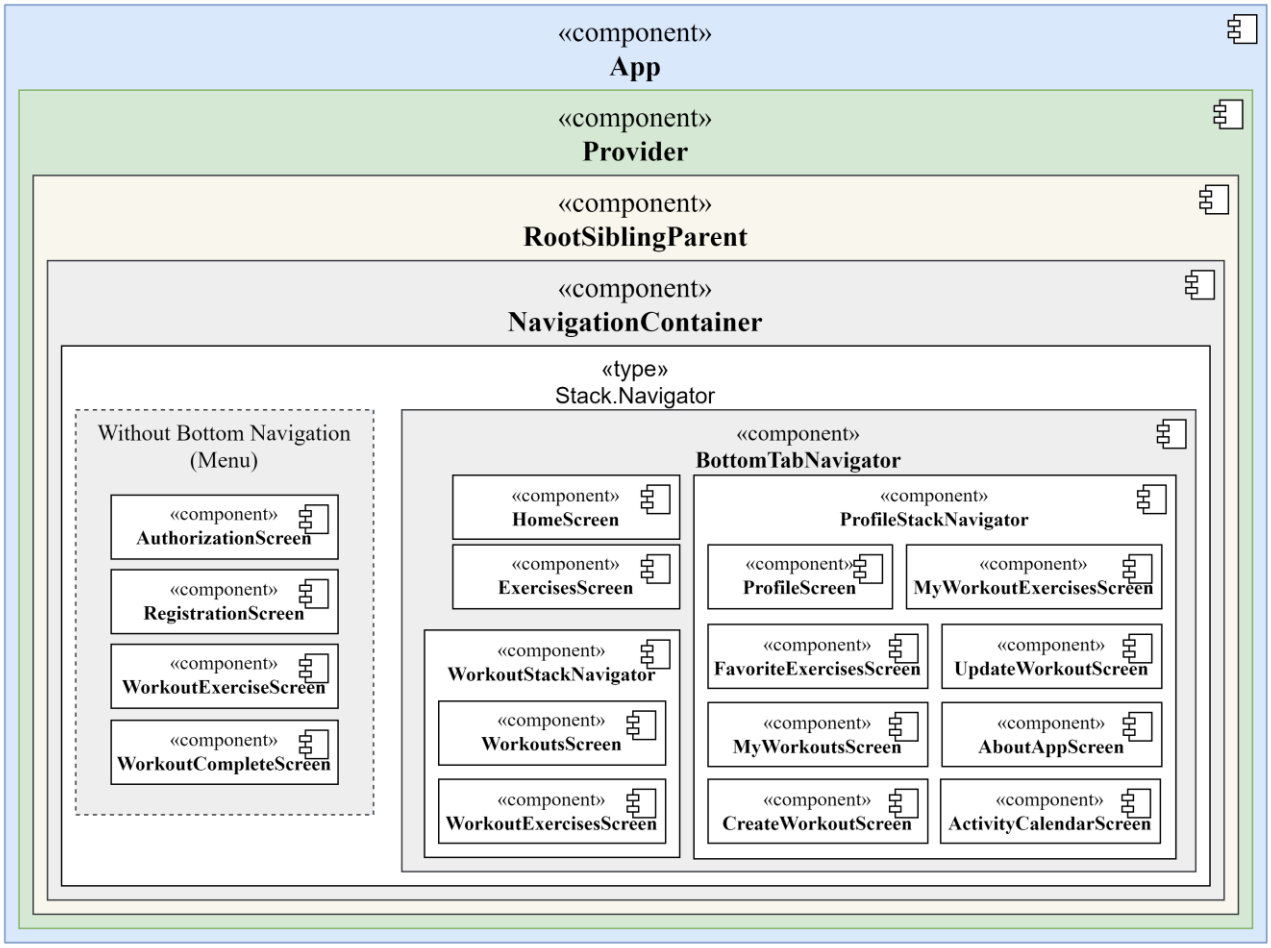


Рисунок 3.1 – Головний функціональний компонент App

При запуску застосунку першим екраном, що відображається є Splash екран. Щоб досягти анімації на цьому екрані, використовується компонент SplashComponent з використанням бібліотеки Animated у React Native. Використовуються компоненти Animated.View для обгортання логотипу та тексту, а також Animated.Text для анімації зміни властивостей тексту. Стан animation зберігається за допомогою useState та ініціалізується зі значенням 0. Під час виконання ефекту useEffect, використовується Animated.timing для анімації зміни значення animation від 0 до 1 протягом 3 секунд. Значення animation використовується для встановлення анімаційних властивостей, таких як прозорість та переміщення, для логотипу, тексту та блоку з інформацією.

Екран авторизації відображається через компонент AuthorizationScreen. Основна функціональність цього компонента полягає в перевірці автентифікації користувача, обробці введених даних та навігації на інші екрани. Redux-слайс

authorizationSlice визначає початковий стан для авторизації, а також редуктори для зміни полів email, password, errors та authenticated. Крім того, він містить дві функції handleLogin і checkAuthenticated, які використовуються для обробки логіки входу користувача та перевірки статусу авторизації.

Компонент RegistrationScreen відповідає за відображення екрану реєстрації користувачів. Для керування станом та виклику дій, які впливають на стан, використовується redux-слайс RegistrationSlice. Цей слайс містить початковий стан initialState з полями, такими як name, email, password, confirmPassword та errors. Редуктори визначені за допомогою функцій-обробників, таких як setName, setEmail, setPassword, setConfirmPassword та setErrors, які змінюють відповідні поля в стані. Крім того, присутня функція handleRegister, яка викликається для обробки процесу реєстрації користувача.

Компонент HomeScreen є основним екраном застосунку, на якому відображаються категорії вправ для вибору. Він використовується для відображення списку категорій та реагує на дії користувача, коли він натискає на певну категорію. Для забезпечення відображення вмісту на екрані та прокрутки списку категорій вправ, компонент використовує компоненти SafeAreaView та ScrollView. У ефекті useEffect застосовується хук useDispatch для отримання об’єкта диспетчера з Redux, і виклик функції fetchCategories. Функція fetchCategories є функцією-екшном, створеною за допомогою createAsyncThunk з Redux Toolkit, і виконує асинхронне завантаження категорій з сервера. У цьому ефекті також відбувається диспетчеризація дії fetchCategories, що спричиняє завантаження категорій при монтуванні компонента HomeScreen.

Компонент ExercisesScreen призначений для відображення екрану з вправами. Він включає у себе ExerciseComponent, що відповідає за відображення окремої вправи на екрані. ExercisesScreen використовує useState та useEffect для зберігання та оновлення стану компонента, таких як searchQuery, fetchedExercises та exercises. Також використовується

useExercisesPagination для роботи з пагінацією вправ. У свою чергу, компонент ExerciseComponent має функціонал для рендерингу елементів, які відображають інформацію про вправу та обробники подій для збереження вправи.

Компонент WorkoutsScreen відповідає за відображення екрану зі списком тренувань. Він використовує useState для зберігання стану, який містить інформацію про доступні тренування. Через useEffect відбувається асинхронне завантаження тренувань з бази даних, і отримані дані зберігаються у стані компонента. При натисканні на конкретне тренування відбувається навігація на екран WorkoutExercisesScreen, де відображається детальна інформація про обране тренування разом з можливістю розпочати його.

Компонент WorkoutExercisesScreen призначений для відображення інформації про конкретне тренування. Він приймає дані про обране тренування через параметр route. Компонент містить зображення тренування, його опис, список вправ, кругову діаграму з цільовими м’язами задіяними під час даного тренування та можливість розпочати саме тренування. При натисканні на кнопку «Play Workout» відбувається навігація на екран WorkoutExerciseScreen для початку проходження вправ.

Компонент WorkoutExerciseScreen відображає конкретну вправу під час тренування та управляє процесом проходження вправ. Цей компонент використовується для відображення окремих етапів тренування із забезпеченням інтерактивної взаємодії користувача.

FavoriteExercisesScreen має аналогічну структуру до ExercisesScreen, але вправи завантажуються з Firebase Firestore.

Аналогічно, MyWorkoutsScreen має такий самий дизайн і функціонал, як і WorkoutsScreen, але з додатковою кнопкою для створення власного тренування, яка веде до спеціального екрану CreateWorkoutScreen.

Компонент ProfileScreen відображає особистий профіль користувача. Інформація про електронну адресу, ім'я та фото отримується з бази даних користувачів. Користувач може змінити своє ім'я, використовуючи функцію

updateUserName, яка оновлює дані в базі даних. Зміна фото виконується за допомогою функції updatePhotoURL, яка також взаємодіє з базою даних. Дані про активність користувача, що відображаються у статистиці, також отримуються з бази даних. Кнопка Logout запускає функцію handleLogout, яка видаляє сесію користувача та повертає його на екран авторизації.

Компонент ActivityCalendarScreen відображає календар активності користувача. Дані про активність завантажуються з бази даних, використовуючи функцію fetchUserActivity. Під час завантаження даних перевіряється, чи користувач увійшов у систему, за допомогою функції checkLoggedInAndAlert. Якщо користувач не увійшов, відображається повідомлення про вхід.

Компонент CreateWorkoutScreen призначений для створення нового тренування користувачем. Користувач вводить назву та опис тренування за допомогою компонентів InputField. Функція handleCreate виконує перевірку наявності обов'язкових полів (назва та опис) і викликає функцію createUserWorkout для збереження тренування в базу даних. Кнопка «Generate Workout using our AI» відкриває модальне вікно для генерації назви та опису тренування за допомогою штучного інтелекту. Функція handleGenerateWorkout викликає функцію generateWorkout для отримання інформації про тренування від штучного інтелекту. Після успішної генерації тренування, дані передаються в onGenerate, яка оновлює стан компонента CreateWorkoutScreen.

Компонент GenerateWorkoutModal відповідає за модальне вікно для генерації тренування за допомогою штучного інтелекту. Користувач вводить відомості про свої цілі тренування в полі вводу за допомогою компонента InputField. Після введення інформації, користувач може натиснути кнопку «Get AI-Designed Workout», щоб запустити процес генерації тренування. Під час генерації тренування, відображається індикатор завантаження. Після успішної генерації тренування, інформація передається в onGenerate, яка оновлює стан батьківського компонента та закриває модальне вікно.

Отже, структура мобільного застосунку показана на рисунку 3.2.

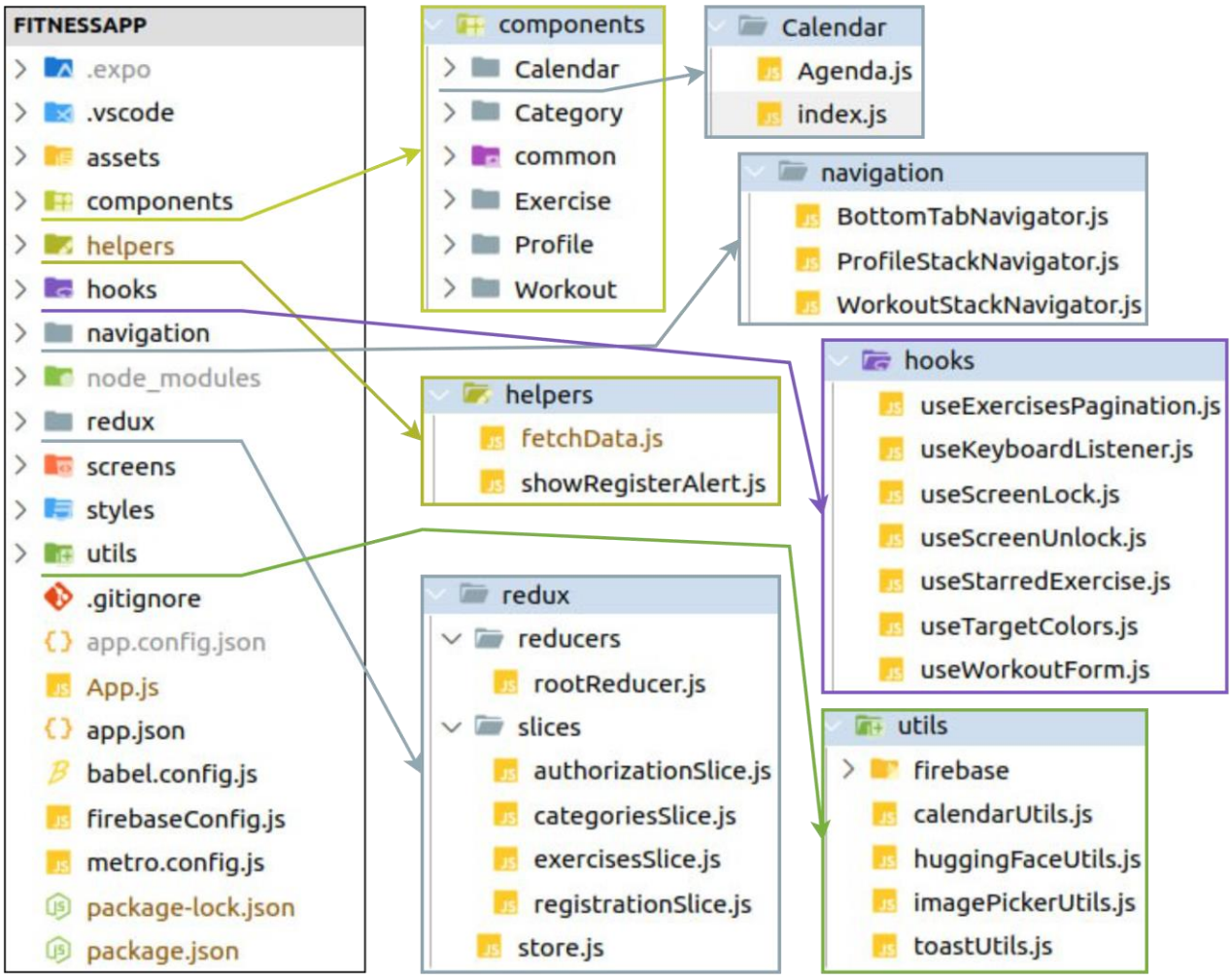


Рисунок 3.2 – Структура проєкту

3.2 Тренування моделі штучного інтелекту на власних даних

Великі моделі природної мови (LLMs, Large Language Models) [37], завдяки трансформерній архітектурі та величезним обсягам тренувальних даних, володіють універсальними функціональностями та вражаючими показниками продуктивності. Однак, це моделі загального призначення, які не сильно підходять для специфічних, конкретних завдань. Для досягнення таких цілей у застосунках використовують спеціалізовані моделі. Однак, замість тренування моделі з нуля, можна скористатися можливістю до налаштувати

існуючу LLM-модель на тренувальних даних. Цей процес має назву тонке налаштування (fine tuning). Коли необхідно дати відповідь на запитання, модель може також використати знання, які отримала під час початкового навчання, так як під час тонкого налаштуванні використовуються ваги моделі, які більше нагадують довгострокову пам’ять (long-term memory), а те що передається на вхід моделі через контекстне вікно, більше схоже на короткострокову пам’ять.

Для початку тонкого налаштування необхідно було встановити наступні обов’язкові бібліотеки:

– transformers: для завантаження великої мовної моделі LLM та її тонкого налаштування;

– bitsandbytes: для завантаження моделі з точністю 4 біти;

– accelerate: для тренування моделей та виконання інференсу в масштабі, тобто один і той самий код на PyTorch може бути виконаний у різних розподілених конфігураціях;

– peft: для тонкого налаштування невеликої кількості параметрів;

– trl: для тренування моделей мови на основі підсиленого навчання.

У попередньому розділі було обрано модель Llama 2 для подальшого тонкого налаштування. Після отримання дозволу від Hugging Face та Meta, було створено токен з правами WRITE на платформі Hugging Face.

Для налаштування квантизації, або квантування моделі була створена функція create_bnb_config. Квантизація – це техніка стиснення моделей глибокого навчання шляхом зменшення кількості бітів, які використовуються для представлення їх ваг та активацій. Використання моделей з точністю 4 біти для трансформаторів дозволяє досягнути вражаючих результатів при значному зменшенні вимог до пам'яті та обчислювальних потреб.

Для завантаження моделі та токенізатора з Hugging Face, використовуючи функцію load_model, було виконано наступні кроки:

– визначено кількість доступних GPU та встановлено максимальну пам’ять GPU;

– завантажено попередньо навчену модель з Hugging Face в точності 4 біти за допомогою методу from_pretrained з класу AutoModelForCausalLM;

– встановлено пристрій для відправлення моделі за допомогою device_map;

– встановлено max_memory до максимальної доступної пам'яті для кожного GPU та доступної оперативної пам'яті CPU, та завантажено токенізатор моделі.

Після цього було ініційовано параметри для модулів transformers та bitsandbytes. Для модуля transformers вказано назву попередньо навченої моделі (meta-llama/Llama-2-7b-hf), яку буде завантажено та налаштовано. Щодо bitsandbytes, активовано 4-бітну точність для завантаження базової моделі, налаштовано квантизацію та обчислювальний тип для 4-бітних базових моделей. Нарешті, викликано функції для отримання об’єктів моделі та токенізатора з відповідними конфігураціями.

Наступним кроком було налаштування набору даних. Спочатку був ініціалізований шлях до набору даних, який знаходися на гугл-диску. Сам набір містить опис тренувань у json-форматі, у відповідь на запит користувача. Функцію load_dataset було використано для завантаження набору даних, передаючи шлях до файлу. Усі записи за замовчуванням були призначені для набору даних train, який використовувався за допомогою параметра split. У результаті чого набір даних було перетворено з файлу із розширенням CSV в словник з інструкціями.

Далі потрібно було обробити дані для подальшого використання. Була визначена функція get_max_length, яка витягує максимальну довжину токена з конфігурації моделі. Якщо максимальна довжина не знайдена, використовується значення за замовчуванням 1024. Функція preprocess_batch, яка токенізує партію даних вхідного набору за допомогою токенізатора моделі. Максимальна довжина послідовності встановлюється за допомогою параметра max_length,

який контролює максимальну довжину, використану параметром для відсічення або доповнення. Та визначена головна функція preprocess_dataset, яка

– додає форматовані заготовки до кожної вибірки у наборі даних;

– токенізує набір даних у партіях за допомогою функції preprocess_batch та видаляє оригінальні ключі словника (інструкція, вхід, вихід та текст);

– відфільтровує заготовки, у яких розмір вхідних токенів перевищує максимальну довжину;

– перемішує набір даних.

Далі потрібно підготувати модель для тренування за допомогою PEFT, щоб ефективно використовувати ресурси та зменшити розмір моделі. Для створення конфігурації PEFT було використано один з методів Parameter-Efficient Fine-tuning від бібліотеки PEFT, Hugging Face. А саме метод QLoRA, який використовує квантизацію на 4 біти для стиснення попередньо навченої мовної моделі. За допомогою конфігурації LoraConfig з бібліотеки peft, було підготовлено модель для додавання навчальних параметрів. Для застосування LoRA (Low-Rank Adapters) до моделі потрібно було визначити модулі, до яких будуть додаватися ці адаптери, це було зроблено через функцію find_all_linear_names. А функція print_trainable_parameters дозволила визначити кількість та відсоток навчальних параметрів моделі. Після визначення параметрів для QLoRA та TrainingArguments було викликано функцію fine_tune для початку процесу тренування моделі.

Після збереження ваг моделі та після тонкого налаштування, ваги були об’єднані з базовою моделлю, і збережені разом із токенізатором. Ця модель та пов’язаний з нею токенізатор був завантажений на Hugging Face Hub (рис. 3.3) для загального використання.

Отже, тонко налаштована модель має наступний вигляд:

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(32000, 4096, padding_idx=0)
    (layers): ModuleList(
```

```
      (0-31): 32 x LlamaDecoderLayer(
        (self_attn): LlamaAttention(
          (q_proj): Linear(in_features=4096, out_features=4096,
bias=False)
          (k_proj): Linear(in_features=4096, out_features=4096,
bias=False)
          (v_proj): Linear(in_features=4096, out_features=4096,
bias=False)
          (o_proj): Linear(in_features=4096, out_features=4096,
bias=False)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear(in_features=4096, out_features=11008,
bias=False)
          (up_proj): Linear(in_features=4096, out_features=11008,
bias=False)
          (down_proj): Linear(in_features=11008, out_features=4096,
bias=False)
          (act_fn): SiLUActivation()
        )
        (input_layernorm): LlamaRMSNorm()
        (post_attention_layernorm): LlamaRMSNorm()
      )
    )
    (norm): LlamaRMSNorm()
  )
  (lm_head): Linear(in_features=4096, out_features=32000, bias=False)
)
```

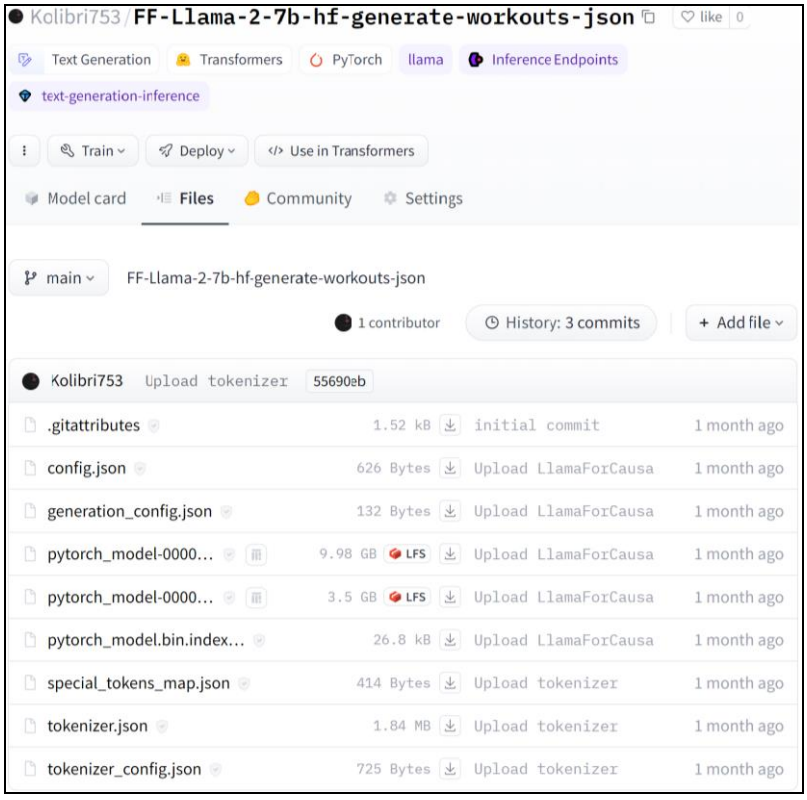


Рисунок 3.3 – Репозиторій тонко налаштованої моделі на Hugging Face Hub

3.3 Розробка бази даних

На основі концептуальної ER-діаграми, яка була розроблена у попередньому розділі, було створено фізичне моделювання даних у базі даних Firestore. Firestore [38] є потужною NoSQL документно-орієнтованою базою даних, що надає гнучкість та масштабованість. У порівнянні з реляційними базами даних SQL, Firestore пропонує альтернативну структуру збереження даних: замість таблиць та рядків, дані організовані у вигляді документів, які групуються в колекції для зручності навігації та управління.

У Firestore дані зберігаються у вигляді колекцій та документів. Документ є записом, який містить різні поля, а колекції об'єднують документи. Хоча документ може також містити вкладені колекції, така можливість на даний момент не підтримується на платформі Android. У контексті SQL-бази даних, колекція відповідає таблиці, а документ представляє собою запис у цій таблиці. Одна колекція може містити документи з різним набором полів.

Колекція «Користувачі» (Users) містить інформацію про користувачів, яка надходить під час реєстрації з використанням Firebase Authentication. У кожному документі «Користувачі» зберігається наступна інформація (рис. 3.4):

– email: поле, для збереження електронної адреси;

– password: поле, призначене для зберігання пароля;

– name: поле, в якому зберігається ім’я користувача, за замовчуванням це електронна пошта без імені домена;

– imageUrl: необов’язкове поле, для фотографії облікового запису;

– favoriteExercises: поле, що містить посилання на колекцію «Улюблені вправи» (Favorite Exercises) даного користувача;

– userActivities: поле, що містить посилання на колекцію «Статистика тренувань» (User Activities) даного користувача;

– userWorkouts: поле, яке містить посилання на колекцію «Власні тренування» (User Workouts) для даного користувача.

Кожен користувач в Firestore має свій власний документ з унікальним ідентифікатором, що спрощує доступ до його даних для зберігання та отримання інформації. Ця унікальність ідентифікаторів дозволяє ефективно керувати даними користувачів у застосунку, забезпечуючи зручну та безпроблемну роботу з інформацією про них.

```
Users (Collection)
├ {userId} (Document)
│   ├ email: "string"
│   ├ password: "string"
│   ├ name: "string"
│   ├ imageUrl: "string"
│   ├ favoriteExercises: (Reference to Favorite Exercises Collection)
│   ├ userActivities: (Reference to User Activities Collection)
│   └ userWorkouts: (Reference to User Workouts Collection)
└
```

Рисунок 3.4 – Колекція «Користувачі»

Колекція «Улюблені вправи» містить інформацію про улюблені вправи користувачів (рис. 3.5). Вона складається з багатьох документів вправ. У кожному документі колекції «Улюблені вправи» зберігається наступна інформація про улюблену вправу:

– bodyPart: поле, що містить інформацію про частину тіла, для якої призначена вправа;

– description: містить опис вправи, який може включати в себе інструкції з виконання та корисну інформацію про неї;

– equipment: поле, що вказує на обладнання, яке потрібно для виконання вправи;

– gifUrl: містить URL-адресу GIF-зображення, яке демонструє виконання вправи;

– name: поле, що містить назву вправи;

– target: поле, що вказує на м’язову групу, яка використовується при виконанні вправи;

– starredAt: містить мітку часу (timestamp), яка вказує на час, коли вправа була додана до списку улюблених.

Рисунок 3.5 – Колекція «Улюблені вправи»

Колекція «Власні тренування» містить інформацію про тренування користувачів (рис. 3.6). Вона містить доповнену інформацію про вправи, так як користувач додатково додав інформацію, яка стосується тренування. У кожному документі зберігається наступна інформація про тренування:

– name: поле, що містить назву власного тренування користувача;

– description: містить опис тренування, який може включати в себе інструкції щодо його виконання, поради та рекомендації;

– image: поле, що містить URL-адресу зображення тренування;

– exercises: колекція документів, яка містить документи вправ, які включені до даного тренування.

Кожен документ колекції «Вправи» (exercises) містить інформацію про окрему вправу, включаючи її назву, опис, частину тіла, для якої вона призначена, обладнання, необхідне для виконання, URL-адресу GIF-зображення, а також порядок у тренуванні, час відпочинку та час виконання.

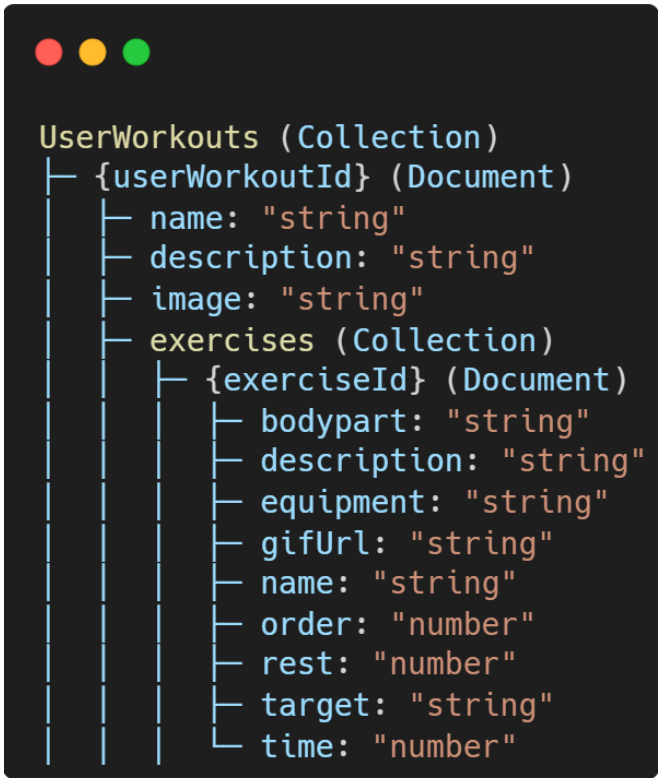


Рисунок 3.6 – Колекція «Улюблені вправи»

Колекція «Статистика тренувань» зберігає інформацію про пройдені тренування з метою аналізу цільових м'язових груп, на які було спрямовано кожне тренування (рис. 3.7). Кожен документ містить наступну інформацію:

– targets: поле, що містить масив цільових м'язів;
– timestamp: містить дату та час, коли було пройдено тренування;
– workoutName: містить назву пройденого тренування.

Рисунок 3.7 – Колекція «Статистика тренувань»

3.4 Тестування системи

3.4.1 Тестування мобільного застосунку

Тестування є невід'ємною частиною життєвого циклу розробки програмного забезпечення. Воно є ключовим етапом у забезпеченні якості продукту та впевненості у його працездатності.

Типи тестування для веб- та мобільних гібридних застосунків [39]:

– юніт-тестування: фокусується на перевірці індивідуальних частин коду для визначення їх коректної роботи, тому дозволяє виявляти та виправляти помилки на ранніх етапах розробки;

– тестування компонентів (модульне): оцінює роботу окремих компонентів програми та їх взаємодію з користувацьким інтерфейсом, це допомагає гарантувати їх правильне відображення та функціональність;

– інтеграційне тестування: перевіряє, як взаємодіють різні компоненти програми між собою та зовнішніми сервісами, що важливо для визначення сумісності та стабільності системи;

– тестування за знімками (Snapshot testing): використовується для виявлення візуальних змін у відображенні компонентів програми та допомагає уникнути неочікуваних змін у користувацькому інтерфейсі;

– тестування з кінця в кінець (End-to-end testing, E2E): перевіряє поведінку програми від початку до кінця, включаючи реальні сценарії взаємодії з користувачем.

Для здійснення тестування у мобільному застосунку були встановлені наступні бібліотеки:

– jest (версія ^29.7.0) – це платформа для тестування JavaScript-коду, яка дозволяє робити тестування з використанням різноманітних підходів, таких як модульне тестування, тестування функцій та інтеграційне тестування;

– jest-expo (версія ^49.0.0) – це плагін Jest для тестування застосунків, розроблених з використанням Expo;

– @testing-library/react-native (версія ^12.5.0) – це бібліотека для тестування React Native застосунків, яка дозволяє писати тестові сценарії, які відтворюють поведінку користувача;

– react-test-renderer (версія 18.2.0) – це бібліотека для створення віртуального DOM та виконання тестів компонентів React, яка дозволяє перевіряти відображення компонентів та їх стан під час тестування.

В таблиці 3.1 наведено опис тестування директорії components/common.

Таблиця 3.1 – Тестування компонентів (common components)

Назва тесту	Категорія	Пояснення
renders correctly with title (TopNavigation)	Юніт-тест	Перевіряє, чи рендериться компонент TopNavigation правильно з вказаною назвою
renders back button and calls navigation.goBack when navigation prop is provided (TopNavigation)	Інтеграційний тест	Перевіряє, чи рендериться кнопка «назад» у компоненті TopNavigation та чи викликається функція navigation.goBack при натисканні
renders correctly (BottomNavigation)	Юніт-тест	Перевіряє, чи рендериться компонент BottomNavigation правильно з вказаними вкладками («Home», «Exercises», «Workouts», «Profile»)
calls navigation.navigate when a tab is pressed (BottomNavigation)	Інтеграційний тест	Перевіряє, чи викликається функція navigation.navigate при натисканні вкладки в компоненті BottomNavigation
renders correctly (CustomModal)	Юніт-тест	Перевіряє, чи рендериться компонент CustomModal правильно з вказаним заголовком і вмістом.
calls closeModal function when close button is pressed (CustomModal)	Інтеграційний тест	Перевіряє, чи викликається функція closeModal при натисканні кнопки закриття модального вікна
renders correctly with default props (InputField)	Модульний тест	Перевіряє, чи рендериться компонент InputField правильно зі стандартними параметрами
renders correctly with provided props (InputField)	Модульний тест	Перевіряє, чи рендериться компонент InputField правильно з вказаними параметрами
calls onChangeText function when input value changes (InputField)	Юніт-тест	Перевіряє, чи викликається функція onChangeText при зміні значення вводу
toggles secureTextEntry when icon is pressed (InputField)	Модульний тест	Перевіряє, чи перемикається параметр secureTextEntry при натисканні іконки ока.
displays error message when error prop is provided (InputField)	Модульний тест	Перевіряє, чи відображається повідомлення про помилку при переданій властивості error

Продовження таблиці 3.1

renders correctly (Logo)	Інтеграційний тест	Перевіряє, чи рендериться компонент Logo правильно з усіма його частинами: текстом «Gym», SVG логотипом та текстом «Rat»
renders correctly with default props (Pagination)	Модульний тест	Перевіряє, чи рендериться компонент Pagination правильно зі значеннями за замовчуванням
calls handlePrevPage when prev button is pressed (Pagination)	Юніт-тест	Перевіряє, чи викликається функція handlePrevPage при натисканні кнопки «попередня сторінка» (<)
calls handleNextPage when next button is pressed (Pagination)	Юніт-тест	Перевіряє, чи викликається функція handleNextPage при натисканні кнопки «наступна сторінка» (>)
calls setPage with correct page number when a pagination button is pressed (Pagination)	Юніт-тест	Цей тест перевіряє, чи викликається функція setPage з правильним номером сторінки, коли натиснуто кнопку пагінації з певним номером сторінки.
renders title correctly (TargetMusclePieChart)	Модульний тест	Цей тест перевіряє, чи рендериться заголовок «Target Muscles» правильно в компоненті TargetMusclePieChart.
renders 'Add exercises' text when no data is provided (TargetMusclePieChart)	Модульний тест	Цей тест перевіряє, чи рендериться текст «Add exercises to this workout!» правильно в компоненті TargetMusclePieChart, коли немає наданих даних.
renders legend items correctly (TargetMusclePieChart)	Модульний тест	Цей тест перевіряє, чи рендеряться елементи легенди правильно в компоненті TargetMusclePieChart.

Також до кожного з цих компонентів було додано snapshot-test (тестування за знімками). На рисунку 3.8 продемонстровано виконання тестування у консольному вікні.

```
vira@vira-ZenBook:~/Desktop/University/mobile_development/FitnessApp$ npm test /components/common

> fitnessapp@1.0.0 test
> jest /components/common

 PASS  components/common/__tests__/TargetMusclePieChart.test.js
 PASS  components/common/__tests__/Logo.test.js
 PASS  components/common/__tests__/Pagination.test.js
 PASS  components/common/__tests__/TopNavigation.test.js
 PASS  components/common/__tests__/InputField.test.js
 PASS  components/common/__tests__/CustomModel.test.js
 PASS  components/common/__tests__/BottomNavigation.test.js

Test Suites: 7 passed, 7 total
Tests:       26 passed, 26 total
Snapshots:   7 passed, 7 total
Time:        3.019 s
Ran all test suites matching /\/components\/common/i.
```

Рисунок 3.8 – Результат тестування у консольному вікні

Тестування з кінця в кінець (End-to-end testing, E2E) було проведено згідно з функціональними вимогами, описаними в ТЗ, та з урахуванням діаграм діяльності, наведених у додатку В. End-to-end тестування перевіряє весь системний потік і збільшує впевненість за рахунок своєчасного виявлення проблем і збільшення покриття тестами підсистем.

Діаграми діяльності (Activity diagrams) були використані для моделювання функціонування системи, оскільки вони надають можливість відобразити потік управління між об’єктами та ілюструвати послідовність процесів та їх розгалуження.

Процес авторизації, зображений на діаграмі діяльності (додаток В.1), починається з того, що користувач переходить на екран авторизації та вводить свої облікові дані, такі як електронна пошта та пароль. Після цього відбувається виклик функції, яка обробляє авторизацію, збираючи облікові дані користувача та виконуючи необхідні перевірки перед передачею даних на сервер Firebase. Після отримання облікових даних сервер передає їх до Firebase Authentication для перевірки. Якщо дані коректні, сервер надає застосунку токен авторизації. Отже, результатом даного процесу є перехід користувача на головний екран застосунку при введених коректних даних, або на екран реєстрації у випадку відсутності облікового запису.

Процес відображення категорій від RapidAPI з ExerciseDB, також зображений на діаграмі діяльності (додаток В.2) та починається з переходу користувача на головний екран. Коли користувач переходить на головний екран, відбувається запит на отримання категорій до RapidAPI. Запит перевіряється за допомогою API ключа для автентифікації. Якщо ключ коректний, запит пересилається до кінцевої точки, де міститься база даних з категоріями. Далі отримуються дані з бази даних, які відповідають запиту. На основі отриманих даних формується відповідь, яка відправляється застосунку, і на головному екрані з’являється назви категорій та відповідна картинка. Користувач може обрати будь-яку категорію зі списку категорій, натиснувши на

неї, після чого застосунок відкриє екран, де відображаються вправи, пов’язані з обраною категорією.

Отже, результати перевірки функціональності занесені в таблицю 3.2.

Таблиця 3.2 – Перевірка функціональних вимог

Функціональність	Перевірено
Реєстрація та авторизація	1. Процес реєстрації нових користувачів та авторизації існуючих. 2. Введення коректних та некоректних даних під час. 3. Можливість користувача авторизуватись після реєстрації за допомогою введених даних.
Перегляд вправ	1. Коректне відображення вправ, що надаються з RapidAPI (ExerciseDB). 2. Відповідність вправ та даних про них.
Проведення тренувань	1. Можливість проходження тренування з вправами від RapidAPI (ExerciseDB). 2. Коректність відображення списку вправ перед тренуванням та детального опису тренування. 3. Правильна робота таймера під час тренування та можливість переходу між вправами. 4. Відображення екрану відпочинку після виконання вправи та екрану привітання після завершення тренування. 5. Коректність роботи кнопок під час тренування. 6. Портретна та альбомна орієнтації. 7. Недоступність функції проходження тренування за відсутності вправ у тренуванні.
Створення списку улюблених вправ	1. Можливість додавання та перегляду вправ у списку улюблених. 2. Коректність видалення вправ зі списку улюблених.
Створення власного тренування	1. Можливість користувача створювати власні тренування. 2. Додавання до тренування вправ з RapidAPI (ExerciseDB). 3. Видалення вправ з тренування. 4. Генерація опису та назви тренування використовуючи штучний інтелект (тонко налаштовану модель). 5. Редагування власного тренування. 6. Видалення створеного тренування.
Збереження даних користувача	1. Правильність збереження даних користувача в обліковому записі Firebase. 2. Коректність збереження улюблених вправ, створених тренувань та іншої активності користувача.
Перегляд активності	1. Можливість користувача переглядати свою активність за допомогою календаря та діаграми. 2. Відповідність виконаних тренувань з відміченими у календарю. 3. Коректність визначення цільової групи м’язів на круговій діаграмі.

Для виявлення і усунення несправностей у застосунку спочатку розглядали можливість використання інструменту Flipper. Flipper є потужним інструментом для відлагодження застосунків для iOS, Android та React Native, який дозволяє візуалізувати, інспектувати та контролювати розробку застосунків, використовуючи зручний інтерфейс. Однак, після рекомендацій проєкту (RFC0641 [40]) від React Native, використання Flipper більше не є доцільним. Багато функцій з Flipper, такі як React Developer Tools та мережевий інспектор, тепер доступні безпосередньо через Expo CLI з версії SDK 49 і вище. Тому було прийнято рішення використовувати Expo CLI.

3.4.2 Тестування роботи зовнішніх сервісів

Оскільки в застосунку «GymRat» (рис. 2.14) використовуються зовнішні сервіси для розширення власних можливостей, важливо переконатися, що ці сервіси працюють правильно та надійно. У разі недоступності або непрацездатності зовнішніх сервісів, у застосунку передбачено механізми обробки виняткових ситуацій.

Для тестування API з вправами було використано Postman [41]. Postman – це надійний та легкий інструмент для тестування веб-API. Він дозволяє створювати, управляти та тестувати API, автоматизувати тести в процесах CI/CD, та допомагає розробникам візуалізувати дані, автентифікувати та керувати API в форматах REST, GraphQL та SOAP.

Спочатку було створено середовище ExercisesInfo (рис. 3.9), в якому була створена колекція запитів. Кожен запит містив необхідні заголовки, такі як X-RapidAPI-Key та X-RapidAPI-Host, а також вказувався URL, за яким потрібно звертатися до API. Після запуску тестів Postman автоматично відправляв запити до API, а потім, отримавши відповіді, перевіряв, чи повернулися очікувані результати. У разі успішного проходження тестів у Postman відображалися результати тестування, підтверджуючи, що API працює як очікувалося.

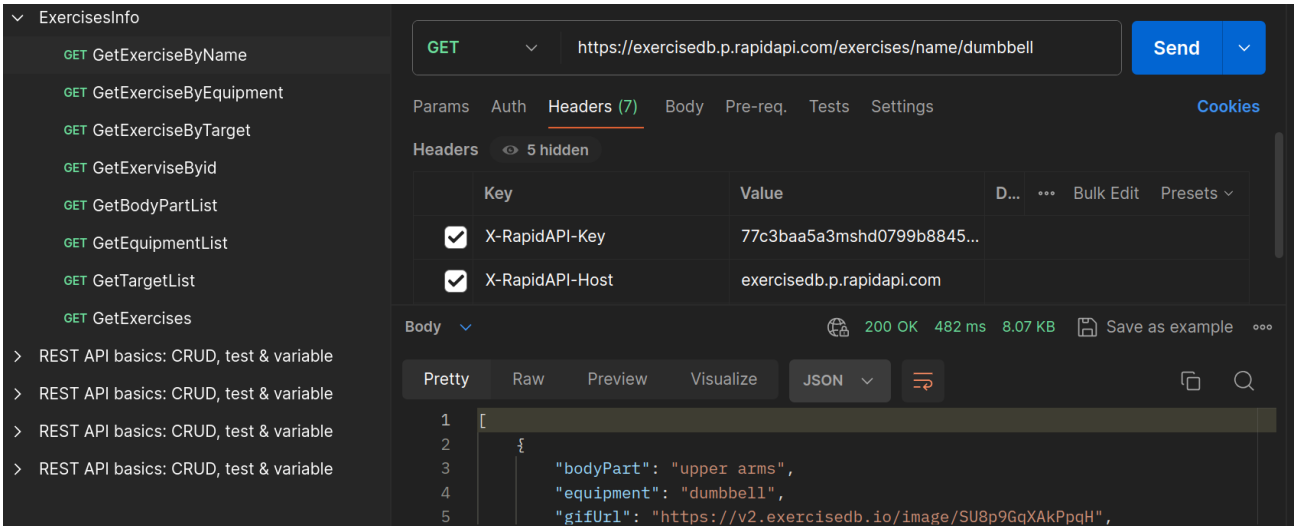


Рисунок 3.9 – Виконання запиту на сторінці Postman

У розділі 3.2 тонко налаштована модель була завантажена на Hugging Face Hub, і далі вона була опублікована через Inference Endpoints. Inference Endpoints [42] – це керований сервіс від Hugging Face, який дозволяє розгортати (практично) будь-яку модель з Hugging Face Hub. Коли створюється кінцева точка, сервіс створює артефакти зображення, які створюються на основі вибраної моделі або спеціально наданого контейнера (Docker Hub, AWS ECR, Azure ACR, або Google GCR). Артефакти зображення повністю відокремлені від вихідних репозиторіїв Hugging Face Hub, щоб забезпечити найвищий рівень безпеки та надійності. Після налаштувань точку ff-llama-2-7b-workouts-json (рис. 3.10) можна запустити та провести тестування.

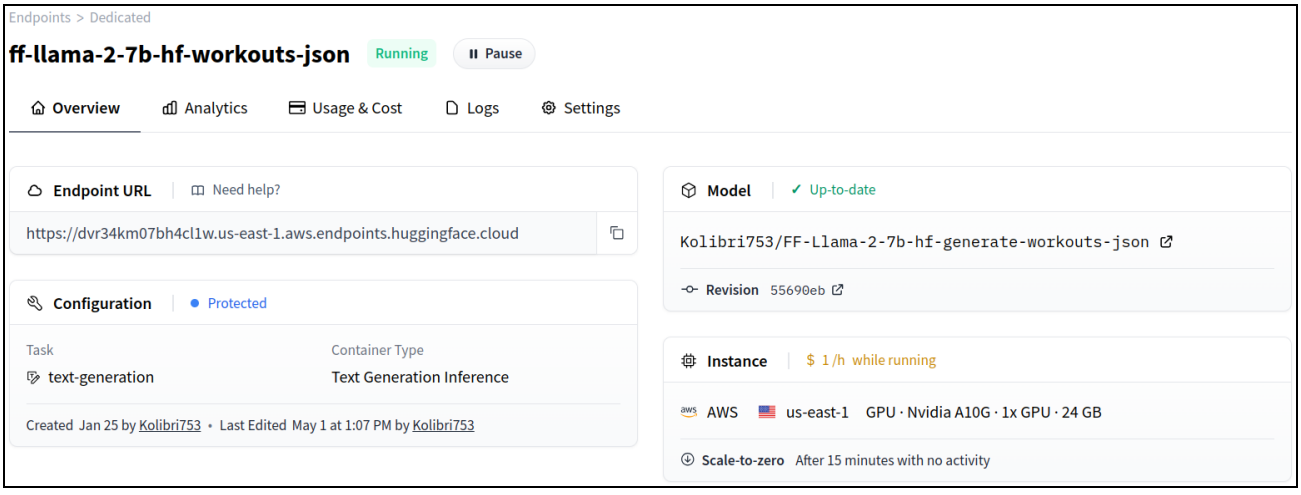


Рисунок 3.10 – Конфігурація Endpoint для генерації опису тренувань

Запити до Inference Endpoints можна відправляти за допомогою користувацького інтерфейсу Inference Widget (рис. 3.11) або програмно, наприклад, з використанням cURL, @huggingface/inference, huggingface_hub або будь-якого клієнта REST. Endpoints також надає не лише інтерактивний віджет для тестування Endpoints, а й генерує код для Python, JavaScript та curl.

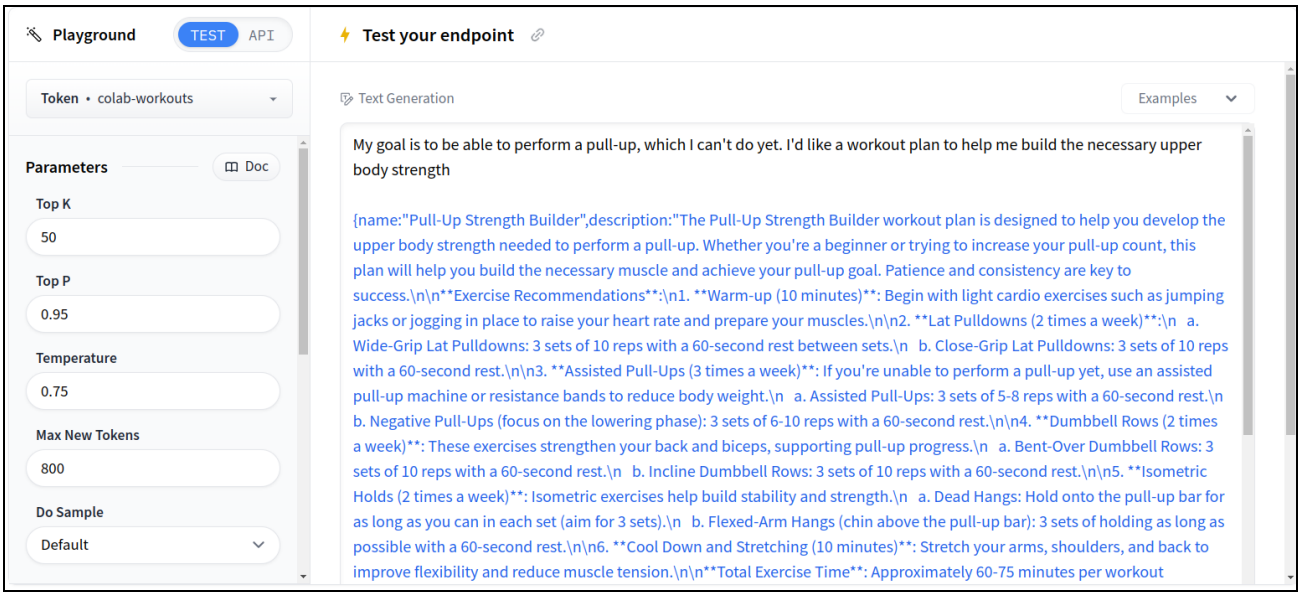


Рисунок 3.11 – Тестування роботи моделі з використанням Inference Widget

З результатів проведеного тестування можна зробити висновок, що модель працює коректно і надає відповіді у правильному форматі, хоча не завжди повну, через малу кількість наданих кредитів.

У даному розділі кваліфікаційної роботи описано процес реалізації та тестування системи та її компонентів. Був реалізований мобільний застосунок «GymRat» (https://github.com/kolibri753/FitnessApp), опубліковано модель Kolibri753/FF-Llama-2-7b-hf-generate-workouts-json для генерації опису тренувань та описано розробку бази даних на основі концептуальної ER-діаграми. Проведено тестування мобільного застосунку з використанням бібліотек, а також описано тестування інтеграції зовнішніх сервісів: для тестування RapidAPI (ExerciseDB) використовувався Postman, а запити до Inference Endpoints надсилалися за допомогою користувацького інтерфейсу Inference Widget.

ВИСНОВКИ

У першому розділі кваліфікаційної роботи було проведено детальний аналіз предметної області та функціональних особливостей мобільних застосунків у сфері фітнесу. Вибір гібридного типу застосунку був обумовлений його здатністю поєднувати переваги веб- та мобільних застосунків. Це дозволить створити універсальний продукт, який буде доступний для широкого кола користувачів, незалежно від операційної системи їх пристроїв. Далі було розглянуто два відомих застосунки для занять спортом – Nike Training Club (NTC) та Jefit, і проведено порівняльний аналіз їх функціональностей та можливостей. Детально описано функції кожного застосунку, відмінності у дизайні та можливості користувачів. Виявлені переваги та недоліки кожного застосунку, були враховані при формуванні вимог до програмного забезпечення. На основі аналізу вимог та функціональних особливостей існуючих застосунків було сформульовано вимоги до майбутнього програмного продукту. Для кращого розуміння взаємозв'язків була розроблена контекстна діаграма та діаграми декомпозиції першого та другого рівнів. Крім того, були визначені актори модельованої системи, їх варіанти використання та зв'язки між ними. У результаті було сформульовано технічне завдання, яке стало основою для подальшої розробки програмного продукту. Аналізовані функціональні та структурні особливості фітнес-застосунків допомогли зрозуміти вимоги користувачів та визначити шляхи подальшого вдосконалення мобільного застосунку «GymRat».

У другому розділі було проведено проєктування архітектури та структури системи. Було обрано безсерверну архітектуру через можливість зосередитися на функціональності застосунку. Для управління станом застосунку було обрано Redux, що забезпечує централізоване сховище стану та спрощує роботу з даними, забезпечуючи легкість у відстеженні їх змін. Далі було проведено проєктування логічної моделі бази даних за допомогою нотації IDEF1X.

Розроблена ER-діаграма є прикладом концептуальної діаграми, що дозволяє візуалізувати зв’язки між сутностями та відображати їх структуру. У розділі також розглянуто проєктування інтерфейсу користувача з використанням матеріального дизайну та вибором трьох основних кольорів: чорного, сірого та жовтого. Було розроблено навігаційне меню та описано склад екранів та форм, що у результаті забезпечує зручний та інтуїтивний інтерфейс для користувачів. Вибір конкретних технологій був здійснений з урахуванням вимог проєкту описаних в ТЗ. Було вирішено використовувати платформу Firebase для зберігання даних у хмарі. Додатково, для генерації описів тренувальних програм, було вирішено замість навчання моделі з нуля, використати технологію тонкого навчання на вже натренованій моделі LLama 2, яка використовує передові технології обробки природної мови (NLP) та здатна генерувати текст високої якості на запит користувача. Та для отримання вправ було обрано ExerciseDB з маркетплейсу RapidAPI, який надає доступ до понад 1300 вправ з анімаціями правильного виконання. Результатом цього етапу є конкретизована діаграма архітектури системи, що відображає архітектуру застосунку та взаємозв’язки з обраними сервісами.

У третьому розділі описано процес реалізації та тестування системи та її компонентів. Було реалізовано основну функціональність застосунку «GymRat». Далі у розділі було зосереджено увагу на реалізації та налаштуванні штучного інтелекту для генерації описів тренувальних програм. Було реалізовано тонко налаштовану модель на основі LLama 2 та завантажено на Hugging Face Hub (Kolibri753/FF-Llama-2-7b-hf-generate-workouts-json). Також вона була успішно інтегрована у мобільний застосунок через Inference Endpoints від Hugging Face. Наступним кроком була розробка бази даних, яка була створена на основі концептуальної ER-діаграми, попередньо розробленої в попередньому розділі. Фізичне моделювання даних було втілено у базі даних Firestore, що забезпечує ефективне зберігання та доступ до інформації. Після реалізації системи було проведено тестування для перевірки правильності роботи застосунку та його

компонентів. Для цього були встановлені відповідні бібліотеки, такі як jest, jest-expo, @testing-library/react-native, react-test-renderer. Також було організовано тестування зовнішніх сервісів: тестування RapidAPI (ExerciseDB) відбувалось за допомогою Postman, а запити до Inference Endpoints надсилалися через користувацький інтерфейс Inference Widget. Отже, у третьому розділі було реалізовано та проведено тестування ключових елементи мобільного застосунку «GymRat», включаючи функціональність, інтеграцію з зовнішніми сервісами та ефективне управління базою даних.

Отже, у результаті виконання кваліфікаційної роботи було реалізовано мобільний застосунок «GymRat», що виступає в ролі віртуального фітнес-тренера. Цей застосунок надає користувачам можливість знаходити вправи, проходити тренування та створювати персоналізовані тренувальні програми. Варто відзначити дві основні переваги цього програмного забезпечення. По-перше, воно використовує тонко налаштовану модель для генерації опису тренувань, що дозволяє користувачам отримати персоналізовані тренування, враховуючи їхні власні потреби. По-друге, користувачі можуть переглядати історію своїх тренувань, що дозволяє відстежувати прогрес у досягненні своїх фітнес-цілей. Такий підхід підвищує мотивацію та сприяє постійному покращенню результатів.

Розробка також була опублікована у збірнику наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп’ютерних наук АПКН-2023» [43].

Можливі напрямки подальшої роботи можуть включати: розширення функціональності застосунку, вибір інших постачальників послуг або перехід на новий план, створення власного API з базою даних вправ, щоб уникнути залежності від третіх осіб, та вдосконалення тонко налаштованої моделі для кращого створення опису тренувань. Також варто розглянути можливість інтеграції додаткових функцій, наприклад, моніторингу харчування або взаємодії з іншими спортивними пристроями, наприклад з фітнес-браслетами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Spraul T. Why the fitness industry is growing | exercise.com. *Exercise.com | Software to grow your fitness business*. URL: https://www.exercise.com/grow/why-the-fitness-industry-is-growing/ (дата звернення: 02.01.2024).

2. Fitness Apps - Worldwide | Statista Market Forecast. *Statista*. URL: https://www.statista.com/outlook/dmo/digital-health/digital-fitness-well-being/health-wellness-coaching/fitness-apps/worldwide (дата звернення: 03.01.2024).

3. Ceci L. Top fitness and workout app download growth 2024 | Statista. *Statista*. URL: https://www.statista.com/statistics/1239806/growth-top-fitness-mobile-apps-downloads/ (дата звернення: 04.03.2024).

4. Gough C. COVID-19: impact on the US home fitness routines 2020 | Statista. *Statista*. URL: https://www.statista.com/statistics/1327102/fitness-routine-covid-impact/ (дата звернення: 03.01.2024).

5. Raturi G. 15+ Top Features of Fitness Apps that Could Make Your App Popular. *LinkedIn*. URL: https://www.linkedin.com/pulse/15-top-features-fitness-apps-could-make-your-app-popular-raturi-gne4c (дата звернення: 12.01.2024).

6. Raturi G. Transforming Workouts: How AI is Revolutionizing Personalized Fitness Plans. *LinkedIn*. URL: https://www.linkedin.com/pulse/transforming-workouts-how-ai-revolutionizing-fitness-plans-raturi-vvfyc (дата звернення: 12.01.2024).

7. Hanna K. T., Wigmore I. What is a mobile app (mobile application)? – TechTarget Definition. *WhatIs*. URL: https://www.techtarget.com/whatis/definition/mobile-app (дата звернення: 15.01.2024).

8. King M. Types of Mobile Apps (2024). *App Developers*. URL: https://www.businessofapps.com/app-developers/research/types-of-mobile-apps/ (дата звернення: 15.01.2024).

9. Maxym Z. IDEF. *Maxym Zosym*. URL: https://www.maxzosim.com/idef/ (дата звернення: 20.01.2024).

10. Health & fitness - android apps on google play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/category/HEALTH_AND_FITNESS?hl=en (дата звернення: 22.01.2024).

11. Ceci L. Topic: Health and fitness apps. *Statista*. URL: https://www.statista.com/topics/9204/health-and-fitness-apps/#topicOverview (дата звернення: 29.02.2024).

12. Nike Training Club App. Home Workouts & More. *Nike.com*. URL: https://www.nike.com/ntc-app (дата звернення: 25.01.2024).

13. Inc N. Nike Training Club: Fitness - Apps on Google Play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=com.nike.ntc (дата звернення: 25.01.2024).

14. Inc J. JEFIT Gym Workout Plan Tracker - Apps on Google Play. *Android Apps on Google Play*. URL: https://play.google.com/store/apps/details?id=je.fit (дата звернення: 29.01.2024).

15. JEFIT - your ultimate fitness app for workout tracking & progress monitoring. *JEFIT - Your Ultimate Fitness App for Workout Tracking & Progress Monitoring*. URL: https://www.jefit.com/ (date of access: 29.01.2024).

16. Maxym Z. Варіанти використання та сценарії (Use Cases and Scenarios). *Maxym Zosym*. URL: https://www.maxzosim.com/use-cases-andscenarios/ (дата звернення: 18.02.2024).

17. Rational software architect standard edition 7.5.5. *IBM in Deutschland, Österreich und der Schweiz*. URL: https://www.ibm.com/docs/en/rsas/7.5.0?topic=diagrams-include-relationships (дата звернення: 18.02.2024).

18. Rational software architect standard edition 7.5.5. *IBM in Deutschland, Österreich und der Schweiz*. URL: https://www.ibm.com/docs/en/rsas/7.5.0?topic=diagrams-extend-relationships (дата звернення: 18.02.2024).

19. Gorton I. Foundations of scalable systems: designing distributed architectures. Sebastopol, California : O'Reilly Media, 2022. 337 с.

20. Shashi A. The rise of serverless computing: what architects need to know. *Medium*. URL: https://medium.com/@ashutoshshashi/the-rise-of-serverless-computing-what-architects-need-to-know-1c848f428c4e (дата звернення: 22.02.2024).

21. Dolynyuk T. Serverless architecture: when to use this approach and what benefits it gives. *Apiko | Outsourced Software Engineering Company*. URL: https://apiko.com/blog/serverless-architecture-benefits/ (дата звернення: 24.02.2024).

22. Adetunji D. What is an API gateway and why is it useful?. *freeCodeCamp.org*. URL: https://www.freecodecamp.org/news/what-are-api-gateways/ (дата звернення: 25.02.2024).

23. He H. Creating apps with React Native: deliver cross-platform 0 crash, 5 star apps. New York: Apress : Apress L. P., 2022. 449 с.

24. A S. Flux vs redux. *Medium*. URL: https://medium.com/nerd-for-tech/flux-vs-redux-6cb572f8d7f8 (дата звернення: 01.03.2024).

25. Piórkowski M. Flux and redux: differences | sunscrapers. *Sunscrapers*. URL: https://sunscrapers.com/blog/flux-and-redux-differences/ (дата звернення: 02.03.2024).

26. Lee C. What Is a Logical Data Model?. *GoodData*. URL: https://www.gooddata.com/blog/how-build-logical-data-models-scale-analytical-applications/ (дата звернення: 05.03.2024).

27. Baránov A. How to create a color palette for design systems | Blog. *Design systems*. URL: https://imperavi.com/blog/how-to-create-a-color-palette-for-design-systems/ (дата звернення: 08.03.2024).

28. Tidwell J., Brewer C., Valencia A. Designing Interfaces: Patterns for Effective Interaction Design. 3-тє вид. Sebastopol, California : O'Reilly Media, Incorporated, 2020. 599 с.

29. Sakhniuk M., Boduch A., Derks R. React and React Native: build cross-platform JavaScript applications with native power for the web, desktop, and mobile. 4-те вид. Birmingham : Packt Publishing, Limited, 2022. 606 с.

30. Padolsey J. Clean Code in JavaScript: Develop reliable, maintainable, and robust JavaScript. Birmingham : Packt Publishing, 2020. 548 с.

31. Introduction - Expo documentation. *Expo - framework | Expo*. URL: https://docs.expo.dev/overview/ (дата звернення: 10.03.2024).

32. Redux Toolkit: Overview | Redux. *Redux - A JS library for predictable and maintainable global state management*. URL: https://redux.js.org/redux-toolkit/overview (дата звернення: 20.04.2024).

33. Firebase - Introduction - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/firebase-introduction/ (дата звернення: 25.04.2024).

34. What is llama 2? | IBM. *IBM in Deutschland, Österreich und der Schweiz*. URL: https://www.ibm.com/topics/llama-2 (дата звернення: 26.04.2024).

35. Lutkevich B. What is Hugging Face? | definition from Techtarget. *WhatIs*. URL: https://www.techtarget.com/whatis/definition/Hugging-Face (дата звернення: 28.04.2024).

36. What is rapidapi.com?. *RapidAPI*. URL: https://docs.rapidapi.com/docs/what-is-rapidapi (дата звернення: 29.04.2024).

37. Jaiman A. Large Language Models [LLMs]. *Medium*. URL: https://ashishjaiman.medium.com/large-language-models-llms-260bf4f39007 (дата звернення: 30.04.2024).

38. Cloud Firestore data model | Firebase. *Firebase*. URL: https://firebase.google.com/docs/firestore/data-model (дата звернення: 01.05.2024).

39. Kahhum. Guide to React Native testing library. *AIA Singapore Technology Blog*. URL: https://ashishjaiman.medium.com/large-language-models-llms-260bf4f39007 (дата звернення: 02.05.2024).

40. Hunt A., Corti N., Andelkovic A. discussions-and-proposals/proposals/0641-decoupling-flipper-from-react-native-core.md at main · react-native-community/discussions-and-proposals. *GitHub*. URL: https://github.com/react-native-community/discussions-and-proposals/blob/main/proposals/0641-decoupling-flipper-from-react-native-core.md (дата звернення: 08.05.2024).

41. How to use Postman for API testing?. *Software Testing & QA - QAlified*. URL: https://qalified.com/blog/postman-for-api-testing/ (дата звернення: 02.05.2024).

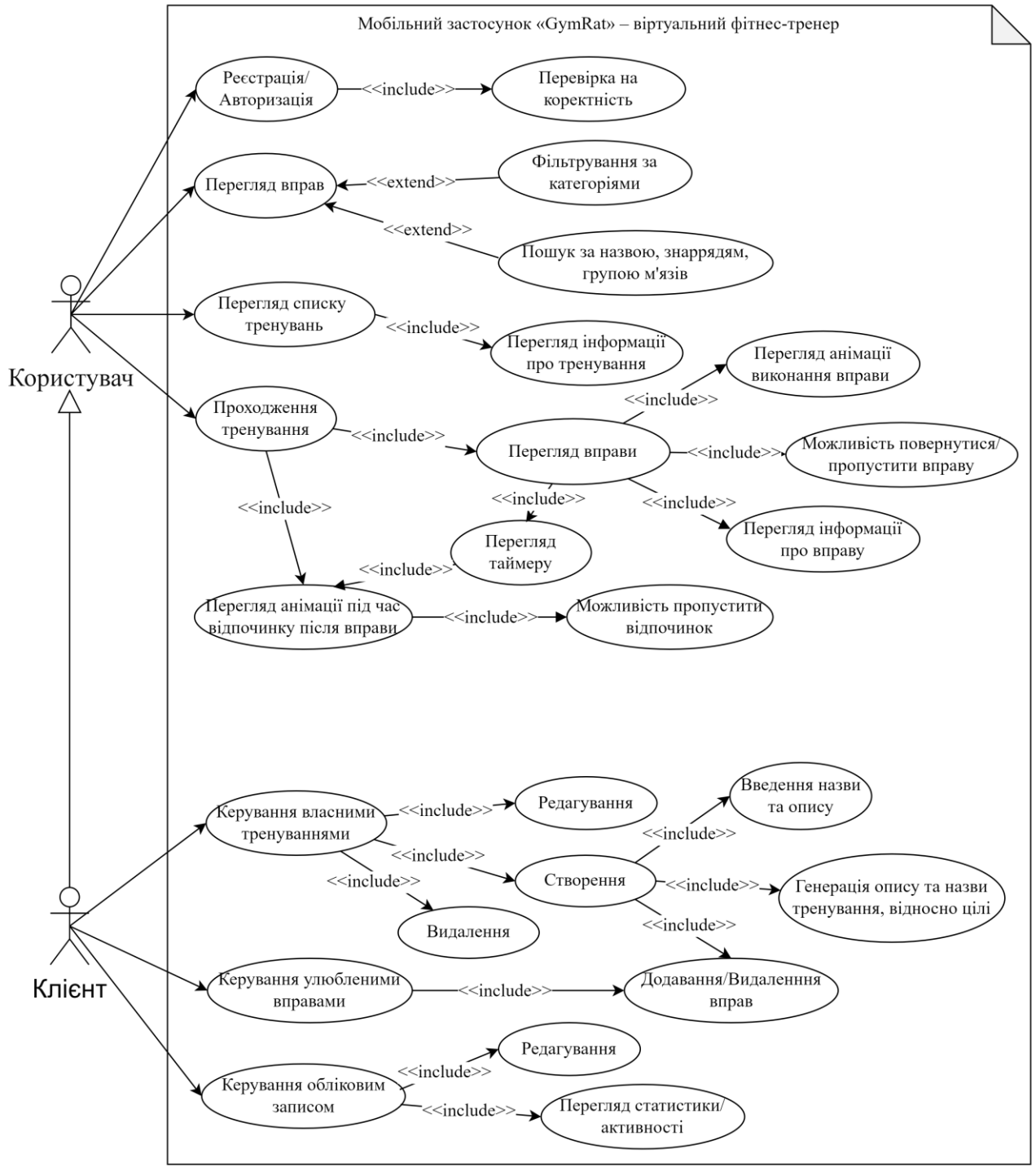
42. 🤗 Inference Endpoints. *Hugging Face – The AI community building the future.* URL: https://huggingface.co/docs/inference-endpoints/index (дата звернення: 08.05.2024).

43. Ваховська В. М. МОБІЛЬНИЙ ДОДАТОК «GYMRAT» – ВІРТУАЛЬНИЙ ФІТНЕС ТРЕНЕР. *Актуальні проблеми комп'ютерних наук АПКН-2023* : зб. наук. пр., м. Хмельницький, 17–18 листоп. 2023 р. Хмельницький, 2023. С. 43–46. URL: https://kn.khmnu.edu.ua/wp-content/uploads/sites/18/apkn-2023-corpuspaper.pdf (дата звернення: 10.05.2024).

ДОДАТОК А

(обов’язковий)

ДІАГРАМА ВАРІАНТ ВИКОРИСТАННЯ



Додаток А.1 – рисунок «Діаграма варіантів використання»

ДОДАТОК Б

(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проєкту розробки мобільного застосунку спрямованого на поліпшення фізичної активності та здоров’я користувачів, шляхом надання інструменту для здійснення фітнес тренувань в будь-який час та в будь-якому місці, також маючи можливість створювати власні тренувальні програми, використовуючи штучний інтелект, та моніторити прогрес тренувань.

Умовне позначення системи: «GymRat» – віртуальний фітнес-тренер.

1 Підстава для розробки

Підставою для розробки є «Завдання на дипломний проєкт», затверджене завідувачем кафедри інженерії програмного забезпечення.

Найменування розробки: Мобільний застосунок «GymRat» – віртуальний фітнес-тренер.

2 Призначення розробки

Мобільного фітнес застосунок «GymRat» призначений для надання користувачем зручного та простого інструменту для здійснення фітнес тренувань. Додаток дозволить користувачам переглядати вправи, проходити тренування, створювати список улюблених вправ, переглядати свій прогрес та створювати власні тренування з використанням вже доданих вправ, а також з можливістю генерації порад щодо тренування, використовуючи штучний інтелект.

Функціональне призначення мобільного фітнес застосунку «GymRat» включає в себе наступні можливості для користувачів:

– перегляд різноманітних вправ та стандартних тренувань;

– проведення тренувань за вибором користувача;

– збереження улюблених вправ для швидкого доступу;

– створення власних тренувань, використовуючи доступні вправи;

– генерація опису (порад) та назви тренування, під потреби користувача, використовуючи штучний інтелект;

– моніторинг та статистика проходження тренувань для відстеження прогресу.

Користувачами застосунку є звичайні користувачі телефонів на платформі Android.

Експлуатаційне призначення, система може використовуватися на будь-якому гаджеті з ОС Android версії вище 5, попередньо встановивши додаток, додаткових налаштувань для початку роботи застосунок не потребує.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Перелік функціональних можливостей, які має надавати програма:

1. Реєстрація та авторизація: додаток повинен мати можливість реєстрації нових користувачів і авторизації існуючих. Для реєстрації необхідно ввести ім’я, електронну пошту, пароль та підтвердження пароля. Після реєстрації користувач повинен мати можливість авторизуватись за допомогою введених під час реєстрації електронної пошти та пароля.

2. Перегляд вправ: додаток повинен надавати користувачам можливість переглядати вправи, що надаються з RapidAPI (ExerciseDB). Кожна вправа повинна містити назву, анімацію демонстрації правильного виконання вправи та кнопки додавання до списку улюблених.

3. Проведення тренувань: користувач повинен мати можливість проводити тренування з використанням вправ, що надаються з RapidAPI (ExerciseDB) та добавленого до них опису. Перед тренуванням користувач має можливість побачити список вправ, які потрібно буде виконати та опис самого тренування (назву, для якого рівня, скільки часу займе, тощо). Під час самого тренування користувач дивиться анімацію виконання вправи, детальний опис її виконання та бачить таймер, який відліковує час до закінчення виконання вправи. Має бути можливість переходу до наступної вправи до закінчення часу, або повернення до попередньої. Після виконаної вправи з'являється екран відпочинку, а після завершення тренування – екран привітання.

4. Створення списку улюблених вправ: користувач повинен мати можливість додавати вправи до свого списку улюблених вправ, щоб зручно і швидко знайти їх у майбутньому. Користувач повинен мати можливість переглядати свій список улюблених вправ та видаляти вправи зі списку.

5. Створення власного тренування: користувач повинен мати можливість створювати свої власні тренування з використанням вправ, що надаються з RapidAPI (ExerciseDB).

6. Збереження даних користувача: дані повинні зберігатись в обліковому записі користувача Firebase, щоб забезпечити конфіденційність і безпеку. Користувач матиме можливість зберігати свої дані, улюблені вправи, створені тренування та власну активність.

7. Перегляд активності: користувач може переглянути свою активність за допомогою календаря, де відображається історія пройдених тренувань. Для кожного тренування та дня надається аналіз у вигляді діаграми, на якій показано, які м'язи були задіяні під час цих тренувань.

3.2 Вимоги до надійності

Застосунок повинен виконувати наступні вимоги до надійності:

– розмежування прав користувачів у системі;

– збереження персональних даних користувачів у Firebase.
– наявність контактної інформації для зв’язку з розробниками.
– валідація користувацьких даних на стороні клієнта і сервера.
– обробка помилок та надсилання повідомлень про це;
– блокування некоректних дій користувача при взаємодії з системою.

3.3 Умови експлуатації

Умови експлуатації повинні відповідати санітарним і технічним нормам експлуатації мобільного пристрою, при температурі та відносній вологості навколишнього середовища, визначених розробником гаджету.

Застосунок може використовуватись на смартфонах та планшетах.

3.4 Вимоги до складу та параметрів технічних засобів

Додаток призначено для використання на смартфонах та планшетах, що працюють під управлінням операційної системи Android. Зазначається, що на одному пристрої може бути встановлено лише одна копія застосунку. Проте, варто враховувати, що коректна робота програми не може бути гарантована у випадку, якщо користувач використовує сторонні засоби для клонування програмного забезпечення.

Для виконання більшості функцій застосунку необхідне стабільне з’єднання з Інтернетом. У подальшому планується масштабування застосунку також на платформу iOS, проте наразі немає гарантії стабільної роботи на цій операційній системі.

Для успішного запуску застосунку пристрій повинен відповідати наступним рекомендованим вимогам:

– операційна система: Android версії 5.0 або вище;
– частота процесора: не менше 500 МГц;
– обсяг оперативної пам’яті (ОЗП): не менше 1 ГБ;
– вільне місце на пристрої: не менше 200 МБ.

3.5 Вимоги до інформаційної та програмної сумісності

Застосунок «GymRat» буде розроблений на платформі React Native Expo з використанням Redux для ефективного управління станом. З цієї причини, він забезпечить сумісність із смартфонами та планшетами, які працюють під управлінням операційної системи Android версії 5 і вище. Планується масштабування застосунку на платформу iOS у майбутньому, але наразі стабільність його роботи на цій операційній системі не гарантується.

Щодо інтеграцій, застосунок використовуватиме RapidAPI для отримання доступу до широкого спектру вправ, що забезпечить різноманітність тренувань для користувачів. Firebase відіграє ключову роль у збереженні різноманітної інформації, такої як облікові записи користувачів (Firebase Authentication), статичні файли застосунку (Firebase Storage), персоналізовані дані користувачів (Firestore Database), а також стандартні та доступні всім користувачам тренування (Firebase Realtime Database).

Hugging Face використовуватиметься для використання тонко налаштованої моделі, яка генеруватиме поради для тренувань, допомагаючи створювати описи та назви тренувань відповідно до вподобань користувача.

3.6 Спеціальні вимоги

Вимоги до інтерфейсу:

– контрастна кольорова гамма, включаючи жовтий, чорний і сірий;

– застосунок повинен мати навігаційне меню внизу екрану для швидкого переключення між екранами та візуальне відображення глибини (глибина взаємодії або рівень меню) та назви екрану.

4 Вимоги до програмної документації

У момент здачі проєкту замовнику надається наступний набір документів:

– код програми з відповідними коментарями;

– опис функціоналу програми;

– технічне завдання (цей документ);

– керівництво користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки проєкту «Мобільний застосунок «GymRat» – віртуальний фітнес-тренер» наведено в таблиці.

Таблиця А.1 – Стадії та етапи розробки проєкту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.2024 – 20.02.2024	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проєкт 21.02.2024– 20.03.2024	Розробка ескізного проєкту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури
Технічний проєкт 21.03.2024– 30.04.2024	Розробка технічного проєкту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми
Робочий проєкт 01.05.2024– 10.05.2024	Розробка програмного забезпечення	Реалізація програмного забезпечення; відлагодження; проведення попереднього тестування
Розробка програмної документації 11.05.2024– 25.05.2024	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 23.04.2024– 30.04.2024	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення

6 Порядок контролю та приймання

Види випробувань:

– юніт-тестування: перевірка окремих частин коду;

– тестування компонентів (модульне): оцінка роботи окремих компонентів програми та їх взаємодії з користувацьким інтерфейсом;

– інтеграційне тестування: перевірка взаємодії різних компонентів програми між собою та зовнішніми сервісами;

– тестування за знімками (Snapshot testing): виявлення візуальних змін у відображенні компонентів програми;

– тестування з кінця в кінець (End-to-end testing, E2E): перевірка поведінки програми від початку до кінця, включаючи реальні сценарії взаємодії з користувачем;

– відмовостійкість: перевірка стабільності та надійності застосунку під час навантаження та використання на різних пристроях.

Вимоги до приймання:

– відповідність вимогам: застосунок повинен повністю відповідати вимогам, зазначеним у технічному завданні;

– відмовостійкість та стабільність: застосунок повинен бути стійким до можливих відмов та надійно працювати під час реального використання на різних пристроях та умовах мережі;

– функціональне тестування: перевірка коректності виконання основних функцій застосунку, включаючи створення тренувань, додавання вправ улюблених користувачем, авторизацію та реєстрацію;

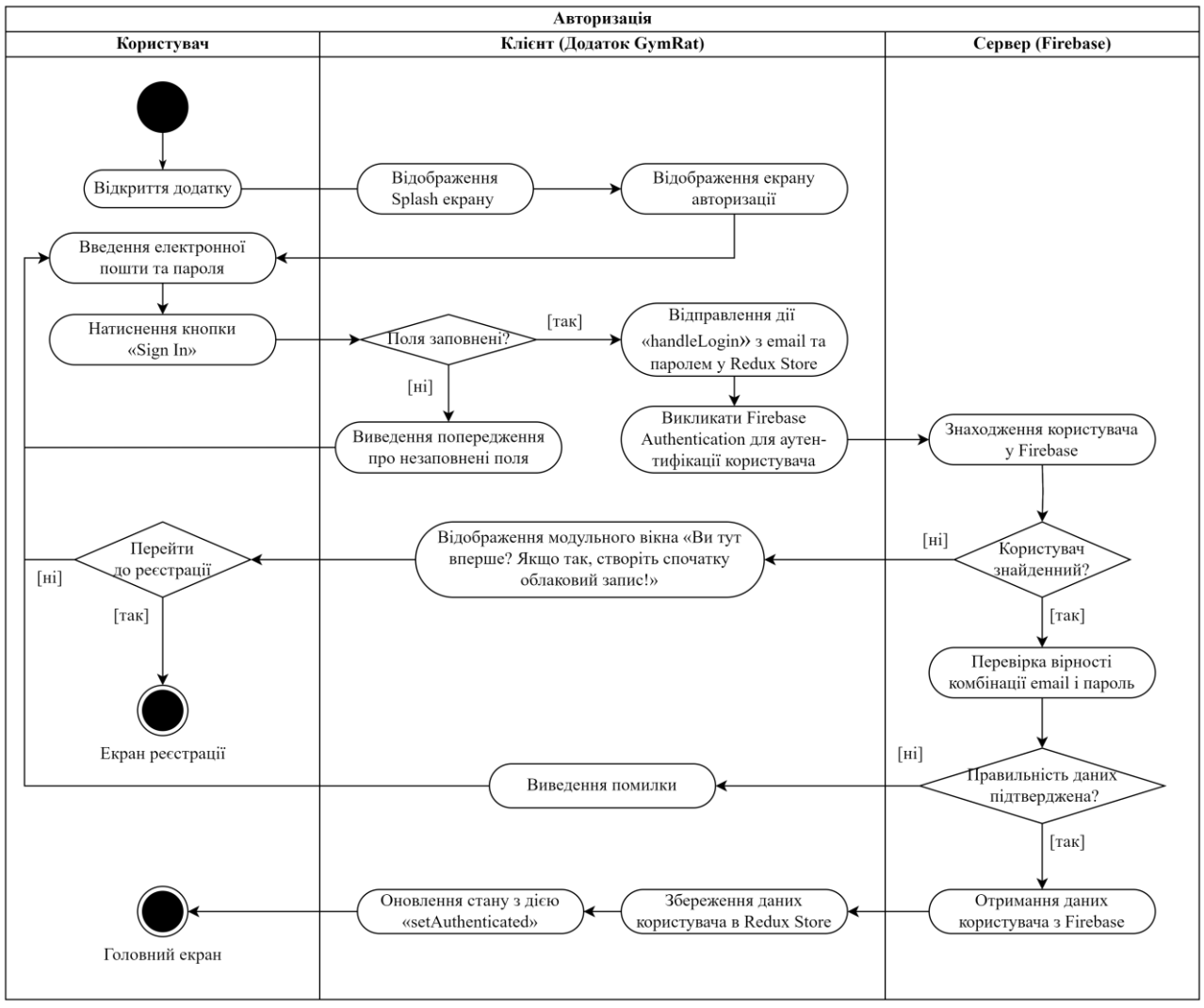
– взаємодія з сервісами: перевірка правильності взаємодії застосунку зі сторонніми сервісами, такими як RapidAPI, Hugging Face та Firebase;

– відповідність документації: всі документи та інструкції, пов’язані з застосунком, повинні бути відповідні та актуальні, забезпечуючи зрозумілу та повну інформацію для користувачів.

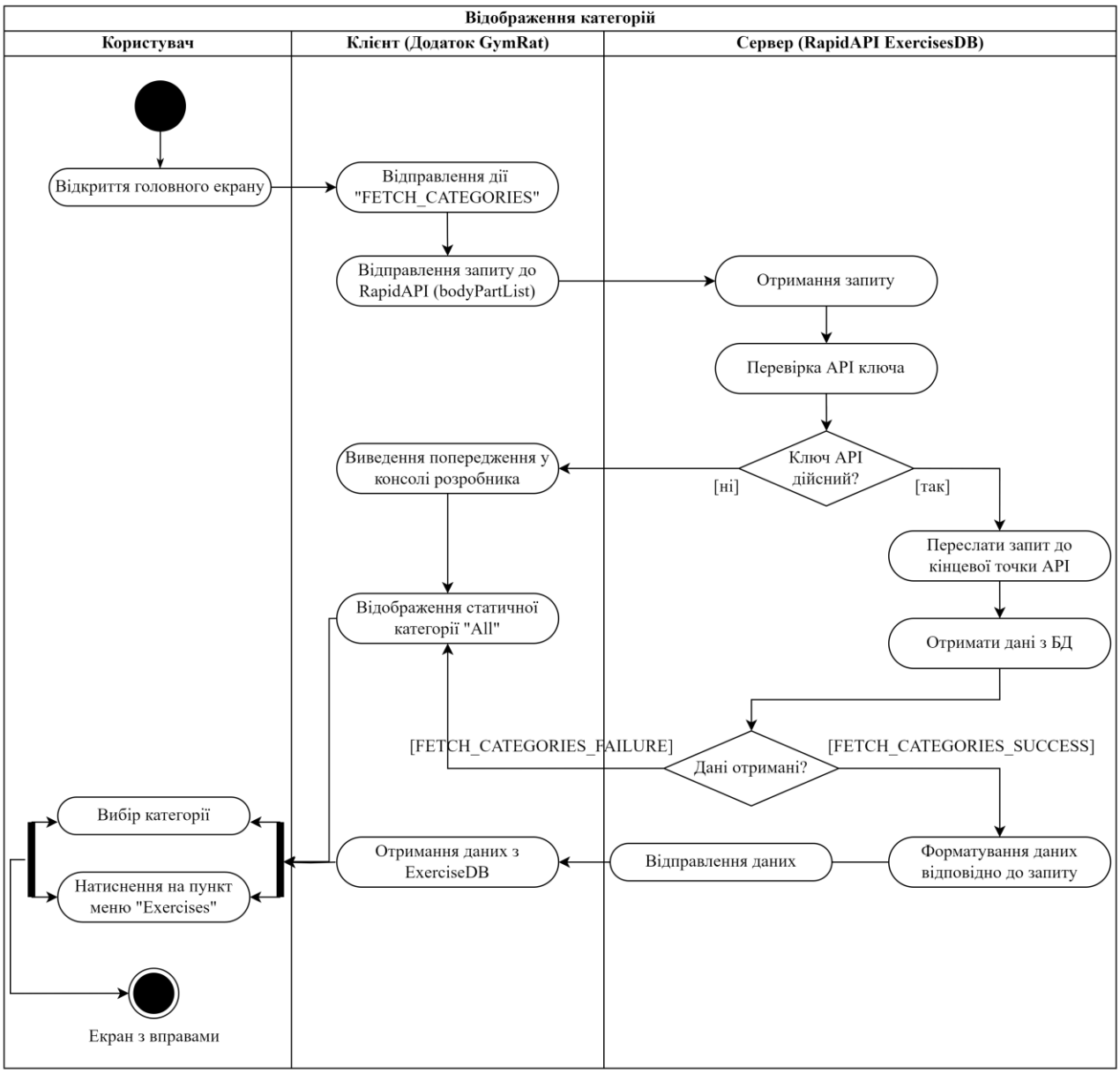
ДОДАТОК В

(обов’язковий)

ДІАГРАМИ ДІЯЛЬНОСТІ



Додаток В.1 – рисунок «Діаграма діяльності процесу авторизації»



Додаток В.2 – рисунок «Діаграма діяльності процесу відображення категорій»

ДОДАТОК Г

(обов’язковий)

ФРАГМЕНТИ КОДУ

package.json

```
{
  "name": "fitnessapp",
  "version": "1.0.0",
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web",
    "test": "jest"
  },
  "jest": {
    "preset": "jest-expo",
    "transformIgnorePatterns": [
      "node_modules/(?!((jest-)?react-native|@react-native(-
community)?)|expo(nent)?|@expo(nent)?/.*|@expo-google-fonts/.*|react-
navigation|@react-navigation/.*|@unimodules/.*|unimodules|sentry-expo|native-
base|react-native-svg)"
    ]
  },
  "dependencies": {
    "@react-navigation/bottom-tabs": "^6.5.7",
    "@react-navigation/native": "^6.1.6",
    "@react-navigation/stack": "^6.3.16",
    "@reduxjs/toolkit": "^1.9.3",
    "expo": "^49.0.22",
    "expo-image-picker": "~14.3.2",
    "expo-screen-orientation": "~6.0.6",
    "expo-status-bar": "~1.6.0",
    "firebase": "^9.18.0",
    "lottie-react-native": "5.1.6",
    "react": "18.2.0",
    "react-native": "0.72.10",
    "react-native-calendars": "^1.1303.0",
    "react-native-pie-chart": "^3.0.2",
    "react-native-root-toast": "^3.4.1",
    "react-native-safe-area-context": "4.6.3",
    "react-native-svg": "13.9.0",
    "react-redux": "^8.0.5",
    "react-test-renderer": "18.2.0",
    "redux": "^4.2.1",
    "react-native-gesture-handler": "~2.12.0"
  },
  "devDependencies": {
    "@babel/core": "^7.20.0",
    "@testing-library/react-native": "^12.5.0",
    "jest": "^29.7.0",
    "jest-expo": "^49.0.0",
    "react-native-svg-transformer": "^1.0.0"
```

```
  },
  "private": true
}
```

App.js

```
import React, { useState, useEffect } from "react";
import { createStackNavigator } from "@react-navigation/stack";
import { NavigationContainer } from "@react-navigation/native";
import { Provider } from "react-redux";
import { configureStore } from "@reduxjs/toolkit";
import { useScreenLock } from "./hooks/useScreenLock";
import AuthorizationScreen from "./screens/AuthorizationScreen";
import RegistrationScreen from "./screens/RegistrationScreen";
import BottomTabNavigator from "./navigation/BottomTabNavigator";
import WorkoutExerciseScreen from "./screens/WorkoutExerciseScreen";
import WorkoutCompleteScreen from "./screens/WorkoutCompleteScreen";
import SplashScreen from "./screens/SplashScreen";
import rootReducer from "./redux/reducers/rootReducer";
import { RootSiblingParent } from "react-native-root-siblings";

const Stack = createStackNavigator();

const store = configureStore({
      reducer: rootReducer,
});

const App = () => {
      const [loading, setLoading] = useState(true);

      useScreenLock();

      useEffect(() => {
            setTimeout(() => {
                  setLoading(false);
            }, 3000); // 3 seconds
      }, []);

      if (loading) {
            return <SplashScreen />;
      }

      return (
            <Provider store={store}>
                  <RootSiblingParent>
                        <NavigationContainer>
                              <Stack.Navigator>
                                    <Stack.Screen
                                          name="AuthorizationScreen"
                                          component={AuthorizationScreen}
                                          options={{
                                                headerShown: false,
                                          }}
                                    />
                                    <Stack.Screen
                                          name="RegistrationScreen"
                                          component={RegistrationScreen}
                                          options={{
                                                headerShown: false,
                                          }}
                                    />
                                    <Stack.Screen
                                          name="Main"
```

```
                                    component={BottomTabNavigator}
                                    options={{
                                          headerShown: false,
                                    }}
                              />
                              <Stack.Screen
                                    name="WorkoutExerciseScreen"
                                    component={WorkoutExerciseScreen}
                                    options={{
                                          headerShown: false,
                                    }}
                              />
                              <Stack.Screen
                                    name="WorkoutCompleteScreen"
                                    component={WorkoutCompleteScreen}
                                    options={{
                                          headerShown: false,
                                    }}
                              />
                        </Stack.Navigator>
                  </NavigationContainer>
            </RootSiblingParent>
      </Provider>
  );
};

export default App;
```

metro.config.js

```
const { getDefaultConfig } = require('@expo/metro-config');

module.exports = (() => {
  const config = getDefaultConfig(__dirname);
  config.resolver.assetExts.push('cjs');

  const { transformer, resolver } = config;

  config.transformer = {
    ...transformer,
    babelTransformerPath: require.resolve("react-native-svg-transformer"),
  };
  config.resolver = {
    ...resolver,
    assetExts: resolver.assetExts.filter((ext) => ext !== "svg"),
    sourceExts: [...resolver.sourceExts, "svg"],
  };

  return config;
})();
```

firebaseConfig.js

```
import { initializeApp } from 'firebase/app';
import { getAuth } from "firebase/auth";
import { getFirestore } from "firebase/firestore";
import { getDatabase } from "firebase/database";
import { getStorage } from 'firebase/storage';
import Constants from 'expo-constants';

// Initialize Firebase
const firebaseConfig = {
  apiKey: Constants.expoConfig.extra.FIREBASE_API_KEY,
  authDomain: Constants.expoConfig.extra.FIREBASE_AUTH_DOMAIN,
```

```
  projectId: Constants.expoConfig.extra.FIREBASE_PROJECT_ID,
  storageBucket: Constants.expoConfig.extra.FIREBASE_STORAGE_BUCKET,
  messagingSenderId: Constants.expoConfig.extra.FIREBASE_MESSAGING_SENDER_ID,
  appId: Constants.expoConfig.extra.FIREBASE_APP_ID,
  measurementId: Constants.expoConfig.extra.FIREBASE_MEASUREMENT_ID,
  databaseURL: Constants.expoConfig.extra.FIREBASE_DATABASE_URL,
};

// const firebaseConfig = Constants.manifest.extra.firebase;

const app = initializeApp(firebaseConfig);
export const auth = getAuth();
export const db =  getFirestore(app);
export const realTimeDb = getDatabase(app);
export const storage = getStorage(app);
```

redux/store.js

```
import { configureStore } from '@reduxjs/toolkit';
import rootReducer from './slices';

const store = configureStore({
  reducer: rootReducer,
});

export default store;
```

redux/reducers/rootReducer.js

```
import { combineReducers } from "@reduxjs/toolkit";
import authorizationReducer from "../slices/authorizationSlice";
import registrationReducer from "../slices/registrationSlice";
import categoriesReducer from "../slices/categoriesSlice";
import exercisesReducer from "../slices/exercisesSlice";

const rootReducer = combineReducers({
  authorization: authorizationReducer,
  registration: registrationReducer,
  categories: categoriesReducer,
  exercises: exercisesReducer,
});

export default rootReducer;
```

redux/slices/categoriesSlice.js

```
import { createSlice, createAsyncThunk } from "@reduxjs/toolkit";
import { fetchData, exerciseOptions } from "../../helpers/fetchData";

export const fetchCategories = createAsyncThunk(
  "categories/fetchCategories",
  async () => {
    console.log("Fetching categories data...");
    const categoriesData = await fetchData(
      "https://exercisedb.p.rapidapi.com/exercises/bodyPartList",
      exerciseOptions
    );
    return categoriesData;
  }
);

const categoriesSlice = createSlice({
      name: "categories",
      initialState: {
```

```
            data: [],
            loading: false,
            error: null,
      },
      reducers: {},
      extraReducers: (builder) => {
            builder
                  .addCase(fetchCategories.pending, (state) => {
                        state.loading = true;
                        state.error = null;
                  })
                  .addCase(fetchCategories.fulfilled, (state, action) => {
                        state.loading = false;
                        state.data = ["all", ...action.payload];
                  })
                  .addCase(fetchCategories.rejected, (state, action) => {
                        state.loading = false;
                        state.error = action.error.message;
                  });
      },
});

export default categoriesSlice.reducer;
```

redux/slices/authorizationSlice.js

```
import { createSlice } from "@reduxjs/toolkit";
import { auth } from "../../firebaseConfig";
import { onAuthStateChanged, signInWithEmailAndPassword } from "firebase/auth";

const initialState = {
      email: "",
      password: "",
      errors: [],
      authenticated: false,
};

const authorizationSlice = createSlice({
      name: "authorization",
      initialState,
      reducers: {
            setEmail: (state, action) => {
                  state.email = action.payload;
                  state.errors = [];
            },
            setPassword: (state, action) => {
                  state.password = action.payload;
                  state.errors = [];
            },
            setErrors: (state, action) => {
                  state.errors = action.payload;
            },
            setAuthenticated: (state, action) => {
                  state.authenticated = action.payload;
            },
      },
});

export const { setEmail, setPassword, setErrors, setAuthenticated } =
      authorizationSlice.actions;

export const handleLogin = (email, password) => async (dispatch) => {
      try {
```

```
            const userCredential = await signInWithEmailAndPassword(
                  auth,
                  email,
                  password
            );
            dispatch(setAuthenticated(true));
      } catch (error) {
            const errorCode = error.code;
            let errorMessage = error.message;
            if (errorCode === "auth/email-already-exists") {
                  errorMessage = "This email is already used.";
            } else if (errorCode === "auth/invalid-email") {
                  errorMessage = "Wrong email format.";
            } else if (errorCode === "auth/user-not-found") {
                  errorMessage = "This email is not registered yet.";
            }
            dispatch(setErrors([errorMessage]));
            console.log(errorCode, errorMessage);
      }
};

export const checkAuthenticated = () => async (dispatch) => {
      onAuthStateChanged(auth, (user) => {
            if (user) {
                  // User is logged in
      console.log("User is logged in");
                  dispatch(setAuthenticated(true));
            } else {
                  // User is not logged in
                  console.log("User is not logged in");
            }
      });
};

export const handleLogout = () => async (dispatch) => {
  try {
    await auth.signOut();
    dispatch(setAuthenticated(false));
  } catch (error) {
    console.log(error);
  }
};


export default authorizationSlice.reducer;
```

helpers/fetchData.js

```
import Constants from 'expo-constants';

const rapidApiKey = Constants.expoConfig.extra.RAPID_API_EXERCISEDB_KEY;

export const fetchData = async (url, options, limit = 10) => {
  let allData = [];
  let res;
  let data;

  try {
    options.headers["X-RapidAPI-Key"] = rapidApiKey;

    do {
      res = await fetch(url, options);
      data = await res.json();
```

```
      if (res.status === 429) {
        console.log(`API limit exceeded with key: ${rapidApiKey}`);
      }

      if (data && Symbol.iterator in Object(data)) {
        allData.push(...data);
      }
    } while (data && allData.length < limit);

    if (res.status === 429) {
      const error = new Error("Exceeded maximum number of API requests");
      error.key = rapidApiKey;
      throw error;
    }

    console.log(allData);

    return allData.slice(0, limit);
  } catch (error) {
    console.log(error);
  }
};

export const exerciseOptions = {
  method: "GET",
  headers: {
    "X-RapidAPI-Host": "exercisedb.p.rapidapi.com",
    "X-RapidAPI-Key": rapidApiKey,
  },
};
```

screens/Exercises.js

```
import React, { useState, useEffect } from "react";
import {
      Text,
      View,
      StyleSheet,
      ScrollView,
      TextInput,
      TouchableOpacity,
} from "react-native";
import { colors } from "../styles/colors";
import { AntDesign } from "@expo/vector-icons";
import { SafeAreaView } from "react-native-safe-area-context";
import Exercise from "../components/Exercise";
import Pagination from "../components/common/Pagination";
import TopNavigation from "../components/common/TopNavigation";
import useExercisesPagination from "../hooks/useExercisesPagination";
import { addWorkoutExercise } from "../utils/firebase/workoutsUtils";
import { showSuccessToast, showErrorToast } from "../utils/toastUtils";
import { useSelector, useDispatch } from "react-redux";
import { fetchExercisesData } from "../redux/slices/exercisesSlice";

const ExercisesScreen = ({ route, navigation }) => {
      const { showSelectButton } = route.params || false;

      // Use useSelector to get the selected category from Redux state
      const { selectedCategory } = useSelector((state) => state.exercises);
      const dispatch = useDispatch();
```

```
const [searchQuery, setSearchQuery] = useState("");
const [fetchedExercises, setFetchedExercises] = useState([]);

const {
      exercises,
      setExercises,
      page,
      setPage,
      scrollViewRef,
      scrollToTop,
      handleNextPage,
      handlePrevPage,
      getPaginatedExercises,
} = useExercisesPagination([]);

useEffect(() => {
      // Dispatch the fetchExercisesData thunk to fetch exercises data
      dispatch(fetchExercisesData(selectedCategory));
}, [dispatch, selectedCategory]);

// Use the useSelector hook to get the updated Redux state
const reduxExercises = useSelector((state) => state.exercises.exercises);

useEffect(() => {
      setExercises(reduxExercises); // Use the updated Redux state
      setFetchedExercises(reduxExercises);
}, [reduxExercises, setExercises]);

const handleSelectExercise = async (exerciseData) => {
      try {
            const workoutId = route.params.workoutId;

            await addWorkoutExercise(
                  workoutId,
                  exerciseData,
                  handleSuccess,
                  handleError
            );
      } catch (error) {
            console.error("Error handling select exercise: ", error);
      }
};

const handleSuccess = (message) => {
      showSuccessToast(message);
};

const handleError = (error) => {
      showErrorToast(error);
};

const handleSearch = () => {
      let filteredExercises = [];

      if (searchQuery === "") {
            filteredExercises = fetchedExercises;
      } else {
            filteredExercises = exercises.filter(
                  (exercise) =>

exercise.name.toLowerCase().includes(searchQuery.toLowerCase()) ||
```

```
      exercise.equipment.toLowerCase().includes(searchQuery.toLowerCase()) ||

      exercise.bodyPart.toLowerCase().includes(searchQuery.toLowerCase())
            );
      }

      setExercises(filteredExercises);
};

// const handleCategoryChange = (newCategory) => {
//    // Dispatch the selectCategory action to update the Redux state
//    dispatch(selectCategory(newCategory));
// };

return (
      <SafeAreaView style={styles.container}>
            <TopNavigation
                  title={`Exercises: ${selectedCategory}`}
                  activeDot={2}
                  navigation={navigation}
            />
            <ScrollView style={styles.exercisesContainer}
ref={scrollViewRef}>
                  <View style={styles.searchBarContainer}>
                        <TextInput
                              style={styles.searchInput}
                              placeholder="Search exercises..."
                              value={searchQuery}
                              onChangeText={setSearchQuery}
                        />
                        <TouchableOpacity style={styles.searchButton}
onPress={handleSearch}>
                              <AntDesign name="search1" size={24}
color={colors.black} />
                        </TouchableOpacity>
                  </View>
                  {getPaginatedExercises().map((exercise, index) => (
                        <Exercise
                              key={index}
                              exercise={exercise}
                              navigation={navigation}
                              handleSelectExercise={handleSelectExercise}
                              showSelectButton={showSelectButton}
                        />
                  ))}
                  {getPaginatedExercises().length === 0 && (
                        <Text style={styles.headerText}>
                              This is a limited version with only a few
exercises. Upgrade to premium to
                              get access to the full list of exercises.
                        </Text>
                  )}
                  <Pagination
                        page={page}
                        handlePrevPage={handlePrevPage}
                        handleNextPage={handleNextPage}
                        scrollToTop={scrollToTop}
                        setPage={setPage}
                  />
            </ScrollView>
      </SafeAreaView>
```

```
      );
};

const styles = StyleSheet.create({
      container: {
            flex: 1,
            backgroundColor: colors.black,
      },
      exercisesContainer: {
            flex: 1,
            paddingHorizontal: 20,
            paddingVertical: 10,
            backgroundColor: colors.grey,
      },
      searchBarContainer: {
            flexDirection: "row",
            alignItems: "center",
            marginBottom: 10,
      },
      searchInput: {
            flex: 1,
            height: 40,
            paddingHorizontal: 10,
            borderWidth: 1,
            borderColor: colors.black,
            backgroundColor: colors.white,
            borderRadius: 5,
            marginRight: 10,
      },
      searchButton: {
            backgroundColor: colors.white,
            paddingHorizontal: 7,
            paddingVertical: 7,
            borderRadius: 5,
      },
      headerText: {
            fontSize: 14,
            color: colors.white,
            marginBottom: 20,
      },
});

export default ExercisesScreen;
```

screens/AuthorizationScreen.js

```
import React, { useEffect } from "react";
import {
      StyleSheet,
      View,
      Text,
      TouchableOpacity,
      KeyboardAvoidingView,
} from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";
import { useSelector, useDispatch } from "react-redux";
import { colors } from "../styles/colors";
import Logo from "../components/common/Logo";
import TopNavigation from "../components/common/TopNavigation";
import InputField from "../components/common/InputField";
import { showSuccessToast } from "../utils/toastUtils";

import {
```

```
      setEmail,
      setPassword,
      setErrors,
      handleLogin,
      checkAuthenticated,
} from "../redux/slices/authorizationSlice";

const AuthorizationScreen = ({ navigation }) => {
      const dispatch = useDispatch();
      const { email, password, errors, authenticated } = useSelector(
            (state) => state.authorization
      );

      useEffect(() => {
            dispatch(checkAuthenticated());
      }, [dispatch]);

      useEffect(() => {
            if (authenticated) {
                  navigation.navigate("Main");
                  const successMessage = `You logged in using\n${email}`;
                  showSuccessToast(successMessage);
            }
      }, [authenticated, navigation]);

      const handleUseWithoutAuthorization = () => {
            navigation.navigate("Main");
      };

      const handleLoginButtonPress = () => {
            let newErrors = [];

            if (!email.trim()) {
                  newErrors.push("Please enter your email address.");
            }
            if (!password.trim()) {
                  newErrors.push("Please enter your password.");
            }

            dispatch(setErrors(newErrors));

            if (newErrors.length === 0) {
                  dispatch(handleLogin(email, password));
            }
      };

      return (
            <SafeAreaView style={styles.container}>
                  <TopNavigation title="Sign In" activeDot={1} />
                  <View style={styles.content}>
                        <Logo style={styles.logo} />
                        <Text style={styles.motivation}>
                              "Transform your body, transform your life. Let this
fitness app be your
                              guide on the journey to a healthier and happier
you."
                        </Text>
                        <KeyboardAvoidingView
                              style={styles.form}
                              behavior={Platform.OS === "ios" ? "padding" :
"height"}
                              keyboardVerticalOffset={50}
```

```
                        >
                              <InputField
                                    placeholder="Email"
                                    value={email}
                                    onChangeText={(text) => {
                                          dispatch(setEmail(text));
                                          dispatch(setErrors(""));
                                    }}
                              />
                              <InputField
                                    placeholder="Password"
                                    value={password}
                                    onChangeText={(text) => {
                                          dispatch(setPassword(text));
                                          dispatch(setErrors(""));
                                    }}
                                    secureTextEntry={true}
                              />
                              {errors.length > 0 &&
                                    errors.map((error, index) => (
                                          <Text key={index} style={styles.error}>
                                                {error}
                                          </Text>
                                    ))}
                              <TouchableOpacity style={styles.button}
onPress={handleLoginButtonPress}>
                                    <Text style={styles.buttonText}>Login</Text>
                              </TouchableOpacity>
                        </KeyboardAvoidingView>
                        <View style={styles.signup}>
                              <Text style={styles.signupText}>
                                    Don't have an account?{"\n"}
                                    <Text
                                          style={styles.signupLink}
                                          onPress={() =>
navigation.navigate("RegistrationScreen")}
                                    >
                                          Create one here
                                    </Text>
                              </Text>
                        </View>
                        <TouchableOpacity
                              style={styles.appLink}
                              onPress={handleUseWithoutAuthorization}
                        >
                              <Text style={styles.appLinkText}>Use app without
authorization</Text>
                        </TouchableOpacity>
                  </View>
            </SafeAreaView>
      );
};

const styles = StyleSheet.create({
      container: {
            flex: 1,
            backgroundColor: colors.black,
      },
      content: {
            flex: 1,
            alignItems: "center",
            justifyContent: "center",
```

```
            paddingHorizontal: 20,
            backgroundColor: colors.grey,
      },
      logo: {
            marginTop: "auto",
      },
      title: {
            fontSize: 32,
            fontWeight: "bold",
            marginBottom: 20,
            color: colors.white,
      },
      motivation: {
            fontSize: 16,
            fontStyle: "italic",
            marginBottom: 50,
            color: colors.white,
            textAlign: "center",
      },
      form: {
            alignItems: "center",
            width: "100%",
      },
      button: {
            backgroundColor: colors.yellow,
            borderRadius: 5,
            padding: 10,
            width: "80%",
            alignItems: "center",
            marginTop: 20,
      },
      buttonText: {
            color: colors.black,
            fontSize: 18,
            fontWeight: "bold",
      },
      error: {
            color: "red",
      },
      signup: {
            marginTop: 50,
      },
      signupText: {
            color: colors.white,
            fontSize: 16,
            fontStyle: "italic",
            textAlign: "center",
      },
      signupLink: {
            color: colors.yellow,
            fontWeight: "bold",
      },
      appLink: {
            marginTop: "auto",
            marginBottom: 20,
      },
      appLinkText: {
            color: colors.white,
            fontStyle: "italic",
            textAlign: "center",
      },
});
```

```
export default AuthorizationScreen;
```

screens/ProfileScreen.js

```
import React, { useState, useEffect } from "react";
import { useDispatch } from "react-redux";
import {
      StyleSheet,
      Text,
      View,
      TouchableOpacity,
      Alert,
      Image,
      KeyboardAvoidingView,
} from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";
import { colors } from "../styles/colors";
import { Entypo, Ionicons } from "@expo/vector-icons";
import TopNavigation from "../components/common/TopNavigation";
import InputField from "../components/common/InputField";
import Stats from "../components/Profile/Stats";
import { handleLogout } from "../redux/slices/authorizationSlice";
import {
      fetchUserProfile,
      updateUserName,
      updatePhotoURL,
} from "../utils/firebase/authUtils";

const ProfileScreen = ({ navigation }) => {
      const [email, setEmail] = useState("");
      const [name, setName] = useState("");
      const [newName, setNewName] = useState("");
      const [photoURL, setPhotoURL] = useState("");
      const [showInputField, setShowInputField] = useState(false);

      const dispatch = useDispatch();

      useEffect(() => {
            const userProfile = fetchUserProfile();
            if (userProfile) {
                  const { email, name, photoURL } = userProfile;
                  setEmail(email);
                  setName(name);
                  setNewName(name);
                  setPhotoURL(photoURL);
            }
      }, [name, photoURL]);

      const handleLogoutBtnPress = () => {
            dispatch(handleLogout())
                  .then(() => {
                        navigation.reset({
                              index: 0,
                              routes: [{ name: "AuthorizationScreen" }],
                        });
                  })
                  .catch((error) => {
                        console.log(error);
                  });
      };

      const handleChangeName = async () => {
            try {
```

```
                  const message = await updateUserName(newName);
                  if (newName.trim() !== name) {
                        setName(newName.trim());
                        Alert.alert("Success", message);
                  }
                  setShowInputField(false);
            } catch (error) {
                  setShowInputField(false);
                  Alert.alert("Error", error.message || "Failed to update
name.");
            }
      };

      const handleChangePhotoURL = async () => {
            try {
                  const message = await updatePhotoURL(setPhotoURL);
                  if (message) {
                        const userProfile = fetchUserProfile();
                        const { photoURL } = userProfile;
                        setPhotoURL(photoURL);
                        Alert.alert("Success", message);
                  }
            } catch (error) {
                  Alert.alert("Error", error.message || "Failed to update photo
URL.");
            }
      };

      return (
            <SafeAreaView style={styles.container}>
                  <TopNavigation title="Profile" activeDot={1} />
                  <View style={styles.content}>
                        <TouchableOpacity onPress={handleChangePhotoURL}>
                              <Image
                                    style={styles.profilePhoto}
                                    source={
                                          photoURL
                                                ? { uri: photoURL }
                                                : require("../assets/default-
profile-img.jpg")
                                    }
                              />
                        </TouchableOpacity>
                        <View style={styles.profileInfo}>
                              <Text style={styles.email}>{email}</Text>
                              <KeyboardAvoidingView
                                    style={styles.form}
                                    behavior={Platform.OS === "ios" ? "padding" :
"height"}
                                    keyboardVerticalOffset={0}
                              >
                                    {email && (
                                          <>
                                                {showInputField ? (
                                                      <InputField
                                                            placeholder={"Write
new name here"}
                                                            value={newName}
                                                            onChangeText={(text)
=> {

      setNewName(text);
```

```
                                                      }}

      onSubmitEditing={handleChangeName}
                                                      />
                                                ) : (
                                                      <Text
style={styles.title}>{name ? name : "unknown"}</Text>
                                                )}
                                                <TouchableOpacity
                                                      style={styles.buttonIcon}
                                                      onPress={
                                                            showInputField
                                                                  ?
handleChangeName
                                                                  : () =>
setShowInputField(!showInputField)
                                                      }
                                                >
                                                      <Entypo name="new-message"
size={24} color={colors.yellow} />
                                                </TouchableOpacity>
                                          </>
                                    )}
                              </KeyboardAvoidingView>
                              {email ? (
                                    <Stats />
                              ) : (
                                    <Text style={styles.email}>
                                          Statistics available for registered
users only
                                    </Text>
                              )}
                        </View>

                        <TouchableOpacity
                              style={styles.navButton}
                              onPress={() =>
navigation.navigate("FavoriteExercisesScreen")}
                        >
                              <Ionicons name="heart" size={24}
color={colors.black} />
                              <Text style={styles.navButtonText}>Favorites</Text>
                        </TouchableOpacity>
                        <TouchableOpacity
                              style={styles.navButton}
                              onPress={() =>
navigation.navigate("MyWorkoutsScreen")}
                        >
                              <Ionicons name="barbell" size={24}
color={colors.black} />
                              <Text style={styles.navButtonText}>My
Workouts</Text>
                        </TouchableOpacity>
                        <TouchableOpacity
                              style={styles.navButton}
                              onPress={() =>
navigation.navigate("ActivityCalendarScreen")}
                        >
                              <Ionicons name="calendar-sharp" size={24}
color={colors.black} />
                              <Text style={styles.navButtonText}>Activity
Calendar</Text>
```

```
                        </TouchableOpacity>
                        <TouchableOpacity
                              style={styles.navButton}
                              onPress={() =>
navigation.navigate("AboutAppScreen")}
                        >
                              <Ionicons name="information-circle" size={24}
color={colors.black} />
                              <Text style={styles.navButtonText}>About App</Text>
                        </TouchableOpacity>

                        <TouchableOpacity
                              style={styles.logoutButton}
                              onPress={handleLogoutBtnPress}
                        >
                              <Text style={styles.logoutText}>{email ? "Logout" :
"Register"}</Text>
                        </TouchableOpacity>
                  </View>
            </SafeAreaView>
      );
};

const styles = StyleSheet.create({
      container: {
            flex: 1,
            backgroundColor: colors.black,
      },
      content: {
            flex: 1,
            alignItems: "center",
            paddingHorizontal: 20,
            paddingVertical: 10,
            backgroundColor: colors.grey,
      },
      profilePhoto: {
            height: 200,
            width: 200,
            borderRadius: 100,
            backgroundColor: colors.white,
            marginBottom: 10,
      },
      profileInfo: {
            width: "100%",
            backgroundColor: colors.black,
            marginBottom: 20,
            borderRadius: 10,
      },
      title: {
            padding: 12,
            fontSize: 19,
            fontWeight: "bold",
            color: colors.white,
            marginBottom: 20,
      },
      email: {
            fontSize: 16,
            padding: 10,
            color: colors.white,
            textAlign: "center",
      },
      form: {
```

```
            position: "relative",
      },
      input: {
            backgroundColor: colors.white,
            paddingVertical: 10,
            paddingHorizontal: 20,
            borderRadius: 5,
            width: "80%",
      },
      buttonIcon: {
            position: "absolute",
            top: 0,
            right: 0,
            paddingVertical: 10,
            paddingHorizontal: 10,
      },
      navButton: {
            flexDirection: "row",
            alignItems: "center",
            justifyContent: "center",
            backgroundColor: colors.yellow,
            borderRadius: 5,
            padding: 10,
            width: "80%",
            marginVertical: 10,
      },
      navButtonText: {
            color: colors.black,
            fontSize: 18,
            fontWeight: "bold",
            marginLeft: 10,
      },
      logoutButton: {
            marginTop: "auto",
            marginBottom: 10,
      },
      logoutText: {
            fontSize: 18,
            fontWeight: "bold",
            color: colors.yellow,
      },
});

export default ProfileScreen;
```

screens/ActivityCalendarScreen.js

```
import React, { useState, useEffect } from "react";
import { StyleSheet } from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";
import TopNavigation from "../components/common/TopNavigation";
import CustomCalendar from "../components/Calendar";
import { colors } from "../styles/colors";
import { fetchUserActivity } from "../utils/firebase/userActivitiesUtils";
import { checkLoggedInAndAlert } from "../utils/firebase/generalUtils";

const ActivityCalendarScreen = ({ navigation }) => {
      const [workoutData, setWorkoutData] = useState([]);

      useEffect(() => {
            const fetchData = async () => {
                  try {
                        if (checkLoggedInAndAlert(navigation)) {
```

```
                              fetchUserActivity(
                                    (data) => setWorkoutData(data),
                                    (error) => console.error("Error fetching
workout data: ", error)
                              );
                        }
                  } catch (error) {
                        console.error("Error fetching workout data: ", error);
                  }
            };

            fetchData();
      }, [navigation]);

      return (
            <SafeAreaView style={styles.container}>
                  <TopNavigation
                        title="Activity Calendar"
                        activeDot={2}
                        navigation={navigation}
                  />
                  <CustomCalendar workoutData={workoutData} />
            </SafeAreaView>
      );
};

const styles = StyleSheet.create({
      container: {
            flex: 1,
            backgroundColor: colors.black,
      },
});

export default ActivityCalendarScreen;
```

screens/WorkoutExerciseScreen.js

```
import React, { useState } from "react";
import {
      View,
      Text,
      StyleSheet,
      Image,
      TouchableOpacity,
      useWindowDimensions,
} from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";
import { colors } from "../styles/colors";
import { AntDesign } from "@expo/vector-icons";
import ExerciseTimer from "../components/Exercise/Timer";
import WorkoutRest from "../components/Workout/Rest";
import { useScreenUnlock } from "../hooks/useScreenUnlock";
import { createUserActivity } from "../utils/firebase/userActivitiesUtils";

const WorkoutExerciseScreen = ({ route, navigation }) => {
      const exercises = Array.isArray(route.params.exercises)
            ? route.params.exercises
            : [route.params.exercises];
      const [currentIndex, setCurrentIndex] = useState(0);
      const exercise = exercises[currentIndex];
      const [timeLeft, setTimeLeft] = useState(exercise.time);
      const [isResting, setIsResting] = useState(false);
```

```
      useScreenUnlock();

      const handleNextPress = () => {
            if (isResting) {
                  setIsResting(false);
                  setCurrentIndex((currentIndex + 1) % exercises.length);
                  setTimeLeft(exercises[(currentIndex + 1) %
exercises.length].time);
            } else {
                  setIsResting(true);
                  setTimeLeft(0);
            }
      };

      const handlePrevPress = () => {
            if (Array.isArray(exercises)) {
                  if (currentIndex === 0) {
                        navigation.goBack();
                  } else {
                        setCurrentIndex((currentIndex - 1 + exercises.length) %
exercises.length);
                        setTimeLeft(
                              exercises[(currentIndex - 1 + exercises.length) %
exercises.length].time
                        );
                  }
            }
      };

      const handleFinishPress = async () => {
            try {
                  await createUserActivity(route.params.workoutName, exercises);
                  // Navigate to WorkoutCompleteScreen
                  navigation.navigate("WorkoutCompleteScreen");
            } catch (error) {
                  console.error("Error creating user activity: ", error);
            }
      };

      const { width, height } = useWindowDimensions();
      const isLandscape = width > height;

      return (
            <SafeAreaView
                  style={[styles.container, isLandscape &&
styles.landscapeContainer]}
            >
                  {isResting ? (
                        <WorkoutRest
                              restTime={exercise.rest}
                              handleNextPress={handleNextPress}
                        />
                  ) : (
                        <>
                              <View style={styles.imageContainer}>
                                    <Image
                                          source={{ uri: exercise.gifUrl }}
                                          style={[styles.image, isLandscape &&
styles.landscapeImage]}
                                    />
                              </View>
                              <View
```

```
                                    style={[
                                          styles.detailsContainer,
                                          isLandscape &&
styles.landscapeDetailsContainer,
                                    ]}
                              >
                                    <View>
                                          <Text
style={styles.title}>{exercise.name}</Text>
                                          <Text
style={styles.description}>{exercise.description}</Text>
                                          <View style={styles.indexContainer}>
                                                <Text
style={styles.indexText}>{`Exercise ${currentIndex + 1} of ${
                                                      exercises.length
                                                }`}</Text>
                                          </View>
                                    </View>
                                    <View style={styles.bottomContainer}>
                                          <TouchableOpacity
onPress={handlePrevPress} style={styles.button}>
                                                <AntDesign name="arrowleft"
size={24} color="black" />
                                                <Text
style={styles.buttonText}>Prev</Text>
                                          </TouchableOpacity>
                                          <ExerciseTimer
                                                timeLeft={timeLeft}
                                                setTimeLeft={setTimeLeft}
                                                handleNextPress={handleNextPress}
                                          />
                                          <TouchableOpacity
                                                onPress={
                                                      currentIndex ===
exercises.length - 1
                                                            ? handleFinishPress
                                                            : handleNextPress
                                                }
                                                style={styles.button}
                                          >
                                                <Text style={styles.buttonText}>
                                                      {currentIndex ===
exercises.length - 1 ? "Finish" : "Next"}
                                                </Text>
                                                <AntDesign
                                                      name={currentIndex ===
exercises.length - 1 ? "check" : "arrowright"}
                                                      size={24}
                                                      color="black"
                                                />
                                          </TouchableOpacity>
                                    </View>
                              </View>
                        </>
                  )}
            </SafeAreaView>
      );
};

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

```
      backgroundColor: colors.black,
      padding: 20,
      justifyContent: "space-between",
},
landscapeContainer: {
      flexDirection: "row",
},
imageContainer: {
      flex: 1,
      justifyContent: "center",
      alignItems: "center",
},
image: {
      width: "100%",
      height: 400,
      resizeMode: "contain",
},
landscapeImage: {
      height: 350,
},
detailsContainer: {
      // flex: 1,
      flex: 0,
      justifyContent: "space-between",
      alignItems: "center",
},
landscapeDetailsContainer: {
      flex: 1,
},
bottomContainer: {
      flexDirection: "row",
      justifyContent: "space-between",
      alignItems: "center",
      width: "100%",
},
title: {
      fontSize: 24,
      fontWeight: "bold",
      color: colors.white,
      marginBottom: 10,
      textAlign: "center",
      textTransform: "capitalize",
},
description: {
      fontSize: 16,
      color: colors.white,
      marginBottom: 20,
      textAlign: "center",
},
indexContainer: {
      marginBottom: 10,
},
indexText: {
      fontSize: 16,
      color: colors.white,
      textAlign: "center",
},
button: {
      display: "flex",
      flexDirection: "row",
      justifyContent: "space-between",
      alignItems: "center",
```

```
            backgroundColor: colors.yellow,
            padding: 10,
            borderRadius: 5,
            marginBottom: 10,
            width: 100,
            height: 50,
      },
      buttonText: {
            fontSize: 18,
            fontWeight: "bold",
            color: colors.black,
            textAlign: "center",
      },
});

export default WorkoutExerciseScreen;
```

screens/CreateWorkoutScreen.js

```
import React, { useState } from "react";
import {
      StyleSheet,
      View,
      Text,
      TouchableOpacity,
      KeyboardAvoidingView,
      Platform,
      Image,
} from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";
import { FontAwesome } from "@expo/vector-icons";
import { colors } from "../styles/colors";
import TopNavigation from "../components/common/TopNavigation";
import InputField from "../components/common/InputField";
import GenerateWorkoutModal from "../components/Workout/GenerateModal";
import ImageSelector from "../components/Workout/ImageSelector";
import { createUserWorkout } from "../utils/firebase/workoutsUtils";
import { showSuccessToast, showErrorToast } from "../utils/toastUtils";
import useKeyboardListener from "../hooks/useKeyboardListener";
import useWorkoutForm from "../hooks/useWorkoutForm";

const CreateWorkoutScreen = ({ navigation }) => {
      const {
            name,
            setName,
            description,
            setDescription,
            errors,
            setErrors,
            image,
            setImage,
      } = useWorkoutForm();
      const [isGenerateModalVisible, setGenerateModalVisible] = useState(false);
      const isKeyboardOpen = useKeyboardListener();

      const handleCreate = async () => {
            let newErrors = [];

            if (!name.trim()) {
                  newErrors.push("Please enter workout name");
            }

            if (!description.trim()) {
```

```
                  newErrors.push("Please enter workout description");
            }

            setErrors(newErrors);

            if (newErrors.length === 0) {
                  createUserWorkout(name, description, image, handleSuccess,
handleError);
            }
      };

      const handleGenerateWorkout = async () => {
            setGenerateModalVisible(true);
      };

      const handleSuccess = (message) => {
            showSuccessToast(message);
            navigation.navigate("MyWorkoutsScreen");
      };

      const handleError = (error) => {
            showErrorToast(error);
      };

      return (
            <SafeAreaView style={styles.container}>
                  <TopNavigation
                        title="Create Workout"
                        activeDot={2}
                        navigation={navigation}
                  />
                  <View style={styles.content}>
                        <KeyboardAvoidingView
                              style={styles.form}
                              behavior={Platform.OS === "ios" ? "padding" :
"height"}
                              keyboardVerticalOffset={100}
                        >
                              {!isKeyboardOpen && (
                                    <ImageSelector onSelectImage={setImage}
image={image} />
                              )}
                              <TouchableOpacity
                                    style={[styles.button, styles.magicButton]}
                                    onPress={handleGenerateWorkout}
                              >
                                    <FontAwesome name="magic" size={24}
color="black" />
                                    <Text style={styles.buttonText}>Generate
Workout using our AI</Text>
                                    <FontAwesome name="magic" size={24}
color="black" />
                              </TouchableOpacity>
                              <GenerateWorkoutModal
                                    isVisible={isGenerateModalVisible}
                                    onClose={() =>
setGenerateModalVisible(false)}
                                    onGenerate={(generatedName,
generatedDescription) => {
                                          setName(generatedName);
                                          setDescription(generatedDescription);
                                    }}
```

```
                                    />
                                    <InputField
                                          placeholder="Workout Name"
                                          value={name}
                                          onChangeText={(text) => {
                                                setName(text);
                                                setErrors("");
                                          }}
                                    />
                                    <InputField
                                          placeholder="Workout Description"
                                          value={description}
                                          onChangeText={(text) => {
                                                setDescription(text);
                                                setErrors("");
                                          }}
                                          multiline={true}
                                          numberOfLines={4}
                                    />
                                    {errors.length > 0 &&
                                          errors.map((error, index) => (
                                                <Text key={index} style={styles.error}>
                                                      {error}
                                                </Text>
                                          ))}
                                    <TouchableOpacity style={styles.button}
onPress={handleCreate}>
                                          <Text style={styles.buttonText}>Save
Workout</Text>
                                    </TouchableOpacity>
                              </KeyboardAvoidingView>
                        </View>
                  </SafeAreaView>
            );
};

const styles = StyleSheet.create({
      container: {
            flex: 1,
            backgroundColor: colors.black,
      },
      content: {
            flex: 1,
            justifyContent: "center",
            alignItems: "center",
            paddingHorizontal: 20,
            backgroundColor: colors.grey,
      },
      form: {
            alignItems: "center",
            width: "100%",
      },
      magicButton: {
            flexDirection: "row",
            justifyContent: "space-between",
            width: "100%",
            marginBottom: 40,
      },
      button: {
            backgroundColor: colors.yellow,
            borderRadius: 5,
            padding: 10,
```

```
            width: "50%",
            alignItems: "center",
            marginTop: 20,
      },
      buttonText: {
            fontSize: 18,
            fontWeight: "bold",
            textAlign: "center",
            color: colors.black,
      },
      error: {
            color: "red",
            marginBottom: 10,
      },
});

export default CreateWorkoutScreen;
```

components/Workout/GenerateModal.js

```
import React, { useState } from "react";
import {
      StyleSheet,
      TouchableOpacity,
      Text,
      Alert,
      ActivityIndicator,
} from "react-native";
import { colors } from "../../styles/colors";
import CustomModal from "../common/CustomModal";
import InputField from "../common/InputField";
import { generateWorkout } from "../../utils/huggingFaceUtils";
import { showSuccessToast, showErrorToast } from "../../utils/toastUtils";

const GenerateWorkoutModal = ({ isVisible, onClose, onGenerate }) => {
      const [prompt, setPrompt] = useState("");
      const [loading, setLoading] = useState(false);

      const handleGenerateWorkout = async () => {
            if (prompt.trim()) {
                  try {
                        setLoading(true);
                        const { name, description } = await
generateWorkout(prompt);

                        onGenerate(name, description);
        showSuccessToast("Check out the generated name and description!");
                  } catch (error) {
                        console.error("Error generating AI workout:", error);
                        showErrorToast(`Failed to generate workout. ${error}`);
                  } finally {
                        setLoading(false);
                        onClose();
                  }
            } else {
                  Alert.alert("Warning", "Please enter a prompt to generate the
workout.");
            }
      };

      return (
            <CustomModal
                  isModalVisible={isVisible}
```

```
                  closeModal={onClose}
                  handleSave={handleGenerateWorkout}
                  title="Define your Workout Goals"
            >
                  <InputField
                        placeholder="Example: I'm a student with a busy schedule
and need quick, effective workout to stay active and fit"
                        value={prompt}
                        onChangeText={(text) => {
                              setPrompt(text);
                        }}
                        multiline={true}
                        numberOfLines={4}
                  />
                  <TouchableOpacity
                        style={styles.button}
                        onPress={handleGenerateWorkout}
                        disabled={loading}
                  >
                        {loading ? (
                              <>
                                    <Text style={styles.buttonText}>
                                          Creating your Workout
<ActivityIndicator color={colors.black} />
                                    </Text>
                              </>
                        ) : (
                              <Text style={styles.buttonText}>Get AI-Designed
Workout</Text>
                        )}
                  </TouchableOpacity>
            </CustomModal>
      );
};

const styles = StyleSheet.create({
      button: {
            backgroundColor: colors.yellow,
            borderRadius: 5,
            padding: 10,
            width: "50%",
            alignItems: "center",
            marginTop: 20,
      },
      buttonText: {
            fontSize: 18,
            fontWeight: "bold",
            textAlign: "center",
            color: colors.black,
      },
});

export default GenerateWorkoutModal;
```

components/Exercise/index.js

```
import React, { useState } from "react";
import { View, Text, StyleSheet, Image, TouchableOpacity } from "react-native";
import { MaterialIcons } from "@expo/vector-icons";
import { colors } from "../../styles/colors";
import { toggleStarredExercise } from
"../../utils/firebase/favoriteExercisesUtils";
import useStarredExercise from "../../hooks/useStarredExercise";
```

```
import AddExerciseToWorkoutModal from "./AddToWorkoutModal";

const Exercise = ({
      navigation,
      exercise,
      handleSelectExercise,
      showSelectButton,
}) => {
      const [isStarred, setIsStarred] = useStarredExercise(exercise.id);
      const [isModalVisible, setIsModalVisible] = useState(false);
      const [description, setDescription] = useState("");
      const [time, setTime] = useState("");
      const [rest, setRest] = useState("");

      const handleStarPress = async () => {
    try {
      const toggledStar = await toggleStarredExercise(exercise, navigation);
      setIsStarred(toggledStar);
    } catch (error) {
      console.error("Error toggling starred exercise:", error);
    }
  };

      const openModal = () => {
            setIsModalVisible(true);
      };

      const closeModal = () => {
            setIsModalVisible(false);
      };

      const handleSaveExercise = async () => {
            if (time.trim() === "" || rest.trim() === "") {
                  return;
            }

            const { id, ...exerciseDataWithoutId } = exercise;
            const updatedExerciseData = {
                  ...exerciseDataWithoutId,
                  description: description,
                  time: time,
                  rest: rest,
            };

            handleSelectExercise(updatedExerciseData);
            closeModal();
      };

      return (
            <View style={styles.container}>
                  {showSelectButton && (
                        <TouchableOpacity style={styles.selectButton}
onPress={openModal}>
                              <MaterialIcons name="check" size={28}
color={colors.black} />
                        </TouchableOpacity>
                  )}
                  <TouchableOpacity style={styles.button}
onPress={handleStarPress}>
                        <MaterialIcons
                              name={isStarred ? "star" : "star-border"}
                              size={28}
```

```
                              color={isStarred ? colors.yellow : colors.black}
                        />
                  </TouchableOpacity>
                  <View style={styles.imageContainer}>
                        <Image style={styles.image} source={{ uri:
exercise.gifUrl }} />
                  </View>
                  <View style={styles.details}>
                        <Text style={styles.title}>{exercise.name}</Text>
                  </View>
                  <AddExerciseToWorkoutModal
                        isModalVisible={isModalVisible}
                        closeModal={closeModal}
                        handleSaveExercise={handleSaveExercise}
                        exercise={exercise}
                        description={description}
                        setDescription={setDescription}
                        time={time}
                        setTime={setTime}
                        rest={rest}
                        setRest={setRest}
                  />
            </View>
      );
};

const styles = StyleSheet.create({
      container: {
            backgroundColor: colors.white,
            borderRadius: 10,
            padding: 20,
            marginBottom: 10,
            elevation: 2,
            position: "relative",
      },
      button: {
            position: "absolute",
            top: 10,
            right: 10,
            zIndex: 2,
            padding: 5,
            borderRadius: 50,
            backgroundColor: colors.lightGrey,
      },
      selectButton: {
            position: "absolute",
            top: 10,
            left: 10,
            zIndex: 2,
            padding: 5,
            borderRadius: 50,
            backgroundColor: colors.lightGrey,
      },
      imageContainer: {
            alignItems: "center",
            marginVertical: 10,
      },
      image: {
            width: "100%",
            height: 300,
            borderRadius: 10,
            resizeMode: "cover",
```

```
      },
      details: {
            marginVertical: 10,
      },
      title: {
            fontSize: 24,
            fontWeight: "bold",
            textAlign: "center",
      },
});

export default React.memo(Exercise);
```

components/Exercise/Timer.js

```
import React, { useEffect } from "react";
import { View, Text, StyleSheet } from "react-native";
import { colors } from "../../styles/colors";

const ExerciseTimer = ({ timeLeft, setTimeLeft, handleNextPress }) => {
  useEffect(() => {
    const timer = setInterval(() => {
      setTimeLeft((prevTimeLeft) => prevTimeLeft - 1);
    }, 1000);

    return () => clearInterval(timer);
  }, []);

  useEffect(() => {
    if (timeLeft === 0) {
      handleNextPress();
    }
  }, [timeLeft]);

  const minutesLeft = Math.floor(timeLeft / 60);
  const secondsLeft = timeLeft % 60;
  const formattedTimeLeft = `${minutesLeft}:${secondsLeft < 10 ? "0" :
""}${secondsLeft}`;

  return (
    <View style={styles.timerContainer}>
      <Text style={styles.timerText}>{formattedTimeLeft}</Text>
    </View>
  );
};


const styles = StyleSheet.create({
      timerContainer: {
            marginBottom: 20,
            borderWidth: 3,
            borderColor: colors.yellow,
            borderRadius: 10,
            padding: 10,
      },
      timerText: {
            fontSize: 48,
            fontWeight: "bold",
            color: colors.white,
      },
});

export default ExerciseTimer;
```

components/Exercise/AddToWorkoutModal.js

```
import React from "react";
import {
      View,
      Text,
      StyleSheet,
      TextInput,
      TouchableOpacity,
} from "react-native";
import CustomModal from "../common/CustomModal";
import { colors } from "../../styles/colors";

const AddExerciseToWorkoutModal = ({
      isModalVisible,
      closeModal,
      handleSaveExercise,
      exercise,
      description,
      setDescription,
      time,
      setTime,
      rest,
      setRest,
}) => {
      return (
            <CustomModal
                  isModalVisible={isModalVisible}
                  closeModal={closeModal}
                  title={`Add |${exercise.name}| to your workout`}
            >
                  <View style={styles.inputContainer}>
                        <Text style={styles.inputLabel}>Description</Text>
                        <TextInput
                              style={[styles.input, styles.multilineInput]}
                              value={description}
                              onChangeText={(text) => setDescription(text)}
                              multiline={true}
                              numberOfLines={4}
                        />
                  </View>
                  <View style={styles.inputContainer}>
                        <Text style={styles.inputLabel}>Time for exercise</Text>
                        <TextInput
                              style={styles.input}
                              keyboardType="numeric"
                              value={time}
                              onChangeText={(text) => setTime(text)}
                        />
                  </View>
                  <View style={styles.inputContainer}>
                        <Text style={styles.inputLabel}>Rest time</Text>
                        <TextInput
                              style={styles.input}
                              keyboardType="numeric"
                              value={rest}
                              onChangeText={(text) => setRest(text)}
                        />
                  </View>
                  <TouchableOpacity style={styles.saveButton}
onPress={handleSaveExercise}>
                        <Text style={styles.saveButtonText}>SAVE</Text>
```

```
                </TouchableOpacity>
            </CustomModal>
      );
};

const styles = StyleSheet.create({
      inputContainer: {
            marginBottom: 20,
            width: "100%",
      },
      inputLabel: {
            fontSize: 14,
            marginBottom: 5,
      },
      input: {
            backgroundColor: "white",
            borderRadius: 10,
            padding: 10,
            fontSize: 16,
            width: "100%",
      },
      multilineInput: {
            height: 100,
            textAlignVertical: "top",
      },
      saveButton: {
            backgroundColor: colors.yellow,
            borderRadius: 10,
            padding: 10,
            width: "100%",
            alignItems: "center",
            marginTop: 10,
      },
      saveButtonText: {
            color: colors.black,
            fontSize: 18,
            fontWeight: "900",
      },
});

export default AddExerciseToWorkoutModal;
```

components/common/BottomNavigation.js

```
import React from 'react';
import { StyleSheet, View, TouchableOpacity, Text } from 'react-native';
import { Ionicons } from '@expo/vector-icons';
import { colors } from '../../styles/colors';

const BottomNavigation = ({ state, descriptors, navigation }) => {
  return (
    <View style={styles.container}>
      {state.routes.map((route, index) => {
        const { options } = descriptors[route.key];
        const label = options.title || route.name;

        const isFocused = state.index === index;

        const onPress = () => {
          const event = navigation.emit({
            type: 'tabPress',
            target: route.key,
            canPreventDefault: true,
```

```
          });

          if (!isFocused && !event.defaultPrevented) {
            navigation.navigate(route.name);
          }
        };

        return (
          <TouchableOpacity
            key={route.key}
            onPress={onPress}
            style={[styles.tab, isFocused && styles.activeTab]}
          >
            <Ionicons name={options.iconName} size={24} color={isFocused ?
colors.yellow : colors.white} />
            <Text style={[styles.tabLabel, isFocused &&
styles.activeTabLabel]}>{label}</Text>
          </TouchableOpacity>
        );
      })}
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flexDirection: 'row',
    justifyContent: 'space-around',
    alignItems: 'center',
    backgroundColor: colors.black,
    height: 60,
    borderTopWidth: 1,
    borderTopColor: colors.lightGrey,
    paddingHorizontal: 20,
  },
  tab: {
    alignItems: 'center',
  },
  tabLabel: {
    fontSize: 12,
    color: colors.white,
  },
  activeTab: {
    borderBottomWidth: 2,
    borderBottomColor: colors.yellow,
  },
  activeTabLabel: {
    color: colors.yellow,
  },
});

export default BottomNavigation;
```

components/common/__tests__/BottomNavigation.test.js

```
import React from "react";
import { render, fireEvent } from "@testing-library/react-native";
import renderer from "react-test-renderer";
import BottomNavigation from "../BottomNavigation";

describe("BottomNavigation", () => {
  test("renders correctly", () => {
    const state = {
```

```
      routes: [
        { key: "Home", name: "Home" },
        { key: "Exercises", name: "Exercises" },
        { key: "Workouts", name: "Workouts" },
        { key: "Profile", name: "Profile" },
      ],
      index: 0,
    };

    const descriptors = {
      Home: { options: { title: "Home" } },
      Exercises: { options: { title: "Exercises" } },
      Workouts: { options: { title: "Workouts" } },
      Profile: { options: { title: "Profile" } },
    };

    const navigation = {
      emit: jest.fn(),
      navigate: jest.fn(),
    };

    const { getByText } = render(
      <BottomNavigation
        state={state}
        descriptors={descriptors}
        navigation={navigation}
      />
    );

    expect(getByText("Home")).toBeTruthy();
    expect(getByText("Exercises")).toBeTruthy();
    expect(getByText("Workouts")).toBeTruthy();
    expect(getByText("Profile")).toBeTruthy();
  });

  test("calls navigation.navigate when a tab is pressed", () => {
    const state = {
      routes: [
        { key: "Home", name: "Home" },
        { key: "Exercises", name: "Exercises" },
        { key: "Workouts", name: "Workouts" },
        { key: "Profile", name: "Profile" },
      ],
      index: 0,
    };

    const descriptors = {
      Home: { options: { title: "Home" } },
      Exercises: { options: { title: "Exercises" } },
      Workouts: { options: { title: "Workouts" } },
      Profile: { options: { title: "Profile" } },
    };

    const navigation = {
      emit: jest.fn((eventName, eventPayload) => {
        // Mocking navigation.emit to return an object with defaultPrevented
property
        return {
          defaultPrevented: false,
        };
      }),
      navigate: jest.fn(),
```

```
    };

    const { getByText } = render(
      <BottomNavigation
        state={state}
        descriptors={descriptors}
        navigation={navigation}
      />
    );

    fireEvent.press(getByText("Exercises"));
    expect(navigation.navigate).toHaveBeenCalledWith("Exercises");
  });

  it("renders correctly", () => {
    const state = {
      routes: [
        { key: "Home", name: "Home" },
        { key: "Exercises", name: "Exercises" },
        { key: "Workouts", name: "Workouts" },
        { key: "Profile", name: "Profile" },
      ],
      index: 0,
    };

    const descriptors = {
      Home: { options: { title: "Home" } },
      Exercises: { options: { title: "Exercises" } },
      Workouts: { options: { title: "Workouts" } },
      Profile: { options: { title: "Profile" } },
    };

    const navigation = {
      emit: jest.fn(),
      navigate: jest.fn(),
    };

    const tree = renderer
      .create(
        <BottomNavigation
          state={state}
          descriptors={descriptors}
          navigation={navigation}
        />
      )
      .toJSON();
    expect(tree).toMatchSnapshot();
  });
});
```

hooks/useWorkoutForm.js

```
import { useState } from "react";

const useWorkoutForm = () => {
      const [name, setName] = useState("");
      const [description, setDescription] = useState("");
      const [errors, setErrors] = useState([]);
      const [image, setImage] = useState("");

      return {
            name,
            setName,
```

```
            description,
            setDescription,
            errors,
            setErrors,
            image,
            setImage,
        };
};

export default useWorkoutForm;
```

utils/calendarUtils.js

```
import { colors, markedDatesColors } from "../styles/colors";

const calculateDots = (workoutsCount) => {
  if (workoutsCount >= 5) {
    return [
      { key: "first", color: markedDatesColors.first },
      { key: "second", color: markedDatesColors.second },
      { key: "third", color: markedDatesColors.third },
    ];
  } else if (workoutsCount >= 3) {
    return [
      { key: "first", color: markedDatesColors.first },
      { key: "second", color: markedDatesColors.second },
    ];
  } else if (workoutsCount < 3) {
    return [{ key: "first", color: markedDatesColors.first }];
  }
  return [];
};

export const generateMarkedDates = (workoutData, selectedDate) => {
  const markedDates = {};

  workoutData.forEach((workout) => {
    const date = new Date(workout.timestamp).toISOString().split("T")[0];
    const workoutsCount = workoutData.filter(
      (w) => w.timestamp.toISOString().split("T")[0] === date
    ).length;

    markedDates[date] = {
      dots: calculateDots(workoutsCount),
      selected: false,
      disableTouchEvent: false,
      selectedDotColor: colors.yellow,
      marked: true,
    };
  });

  markedDates[selectedDate] = {
    ...markedDates[selectedDate],
    selected: true,
  };

  return markedDates;
};
```

utils/firebase/workoutsUtils.js

```
import { auth, db, realTimeDb } from "../../firebaseConfig";
import { ref, get } from "firebase/database";
import {
```

```
      collection,
      doc,
      getDoc,
      getDocs,
      setDoc,
      addDoc,
      updateDoc,
      deleteDoc,
      onSnapshot,
      query,
} from "firebase/firestore";
import { createId, countDocumentsInCollection } from "./generalUtils";

export const fetchUserWorkouts = (onSuccess, onError) => {
      const uid = auth.currentUser.uid;
      const unsubscribe = onSnapshot(
            collection(db, "users", uid, "userWorkouts"),
            (snapshot) => {
                  const workouts = snapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() }));
                  onSuccess(workouts);
            },
            onError
      );

      return unsubscribe;
};

export const fetchUserWorkout = async (workoutId) => {
      try {
            const userRef = doc(db, "users", auth.currentUser.uid);
            const workoutDocRef = doc(userRef, "userWorkouts", workoutId);
            const workoutDoc = await getDoc(workoutDocRef);

            return workoutDoc.exists() ? workoutDoc.data() : null;
      } catch (error) {
            console.error("Error fetching workout: ", error);
            return null;
      }
};

export const deleteUserWorkout = async (workoutId, onSuccess, onError) => {
      try {
            const uid = auth.currentUser.uid;
            await deleteDoc(doc(db, "users", uid, "userWorkouts", workoutId));
            onSuccess("Workout deleted successfully");
      } catch (error) {
            onError(`Error deleting workout: ${error}`);
      }
};

export const createUserWorkout = async (
      name,
      description,
      image,
      onSuccess,
      onError
) => {
      try {
            const userRef = doc(db, "users", auth.currentUser.uid);
            const workoutDocRef = doc(userRef, "userWorkouts", createId());
            await setDoc(workoutDocRef, { name, description, image });
```

```
            onSuccess("Workout created successfully");
      } catch (error) {
            onError(`Error adding workout: ${error}`);
      }
};

export const updateUserWorkout = async (
      workoutId,
      name,
      description,
      image,
      onSuccess,
      onError
) => {
      try {
            const userRef = doc(db, "users", auth.currentUser.uid);
            const workoutDocRef = doc(userRef, "userWorkouts", workoutId);
            await updateDoc(workoutDocRef, { name, description, image });

            onSuccess("Workout updated successfully");
      } catch (error) {
            onError(`Error updating workout: ${error}`);
      }
};

export const fetchWorkoutExercises = (workoutId, setExercises) => {
      const uid = auth.currentUser.uid;
      const userWorkoutsRef = collection(
            db,
            "users",
            uid,
            "userWorkouts",
            workoutId,
            "exercises"
      );

      const unsubscribe = onSnapshot(userWorkoutsRef, (snapshot) => {
            const data = snapshot.docs.map((doc) => ({ id: doc.id, ...doc.data()
}));
            const sortedData = data.sort((a, b) => a.order - b.order);
            setExercises(sortedData);
      });

      return unsubscribe;
};

export const addWorkoutExercise = async (
      workoutId,
      exerciseData,
      onSuccess,
      onError
) => {
      try {
            const uid = auth.currentUser.uid;
            const userWorkoutsRef = collection(
                  db,
                  "users",
                  uid,
                  "userWorkouts",
                  workoutId,
                  "exercises"
```

```
            );

            const q = query(userWorkoutsRef);
            const querySnapshot = await getDocs(q);

            const numExercises = querySnapshot.size;

            await addDoc(userWorkoutsRef, { ...exerciseData, order: numExercises
+ 1 });
            onSuccess("Exercise added to workout successfully");
      } catch (error) {
            onError(`Error adding exercise: ${error}`);
      }
};

export const deleteWorkoutExercise = async (workoutId, exerciseId) => {
      try {
            const uid = auth.currentUser.uid;
            const exerciseRef = doc(
                  db,
                  "users",
                  uid,
                  "userWorkouts",
                  workoutId,
                  "exercises",
                  exerciseId
            );

            await deleteDoc(exerciseRef);
            console.log("Exercise deleted successfully!");
      } catch (error) {
            console.error("Error deleting exercise: ", error);
      }
};

export const fetchWorkoutsFromRealTimeDb = async () => {
      try {
            const workoutsRef = ref(realTimeDb, "workouts");
            const snapshot = await get(workoutsRef);

            if (snapshot.exists()) {
                  const data = snapshot.val();
                  return Object.entries(data).map(([id, workout]) => ({
                        id,
                        ...workout,
                        exercises: workout?.exercises || [],
                  }));
            } else {
                  console.log("No data available");
                  return [];
            }
      } catch (error) {
            console.error("Error fetching workouts:", error);
            throw error;
      }
};

export const countUserWorkouts = async () => {
      return await countDocumentsInCollection("userWorkouts");
};
```

utils/firebase/userActivitiesUtils.js

```
import { auth, db } from "../../firebaseConfig";
import {
      collection,
      getDocs,
      addDoc,
      serverTimestamp,
} from "firebase/firestore";
import { countDocumentsInCollection } from "./generalUtils";

export const fetchUserActivity = async (onSuccess, onError) => {
      try {
            const uid = auth.currentUser.uid;
            const userActivitiesRef = collection(db, "users", uid,
"userActivities");

            const querySnapshot = await getDocs(userActivitiesRef);

            const data = querySnapshot.docs.map((doc) => {
                  const { workoutName, timestamp, targets } = doc.data();
                  return {
                        workoutName,
                        timestamp: timestamp.toDate(),
                        targets,
                  };
            });

            onSuccess(data);
      } catch (error) {
            console.error("Error fetching workout data: ", error);
            onError(error);
            throw error;
      }
};

export const createUserActivity = async (workoutName, exercises) => {
      try {
            const uid = auth.currentUser?.uid;

            if (!uid) return;

            const userActivitiesRef = collection(db, "users", uid,
"userActivities");
            const timestamp = serverTimestamp();

            await addDoc(userActivitiesRef, {
                  workoutName,
                  timestamp,
                  targets: exercises.map((exercise) => exercise.target),
            });

            console.log("New user activity created successfully!");
      } catch (error) {
            console.error("Error creating user activity: ", error);
            throw error;
      }
};

export const countFinishedWorkouts = async () => {
      return await countDocumentsInCollection("userActivities");
};
```

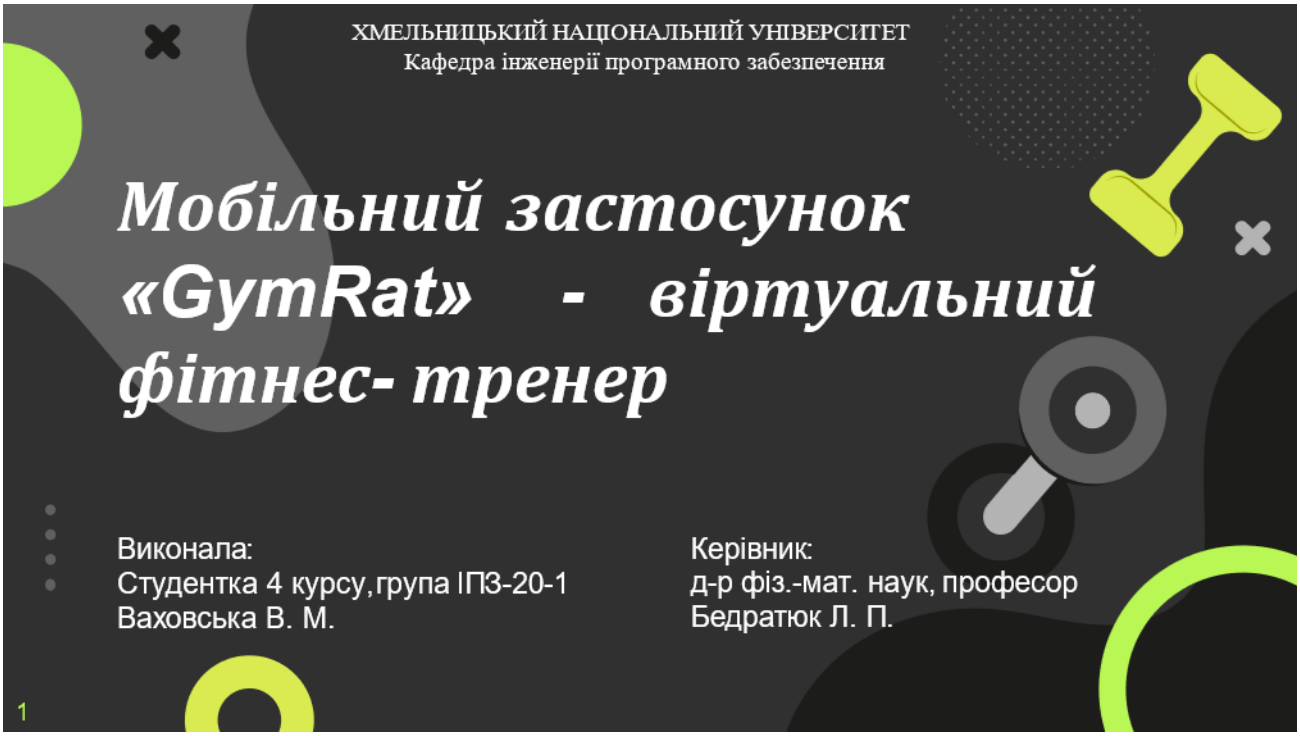
utils/huggingFaceUtils.js

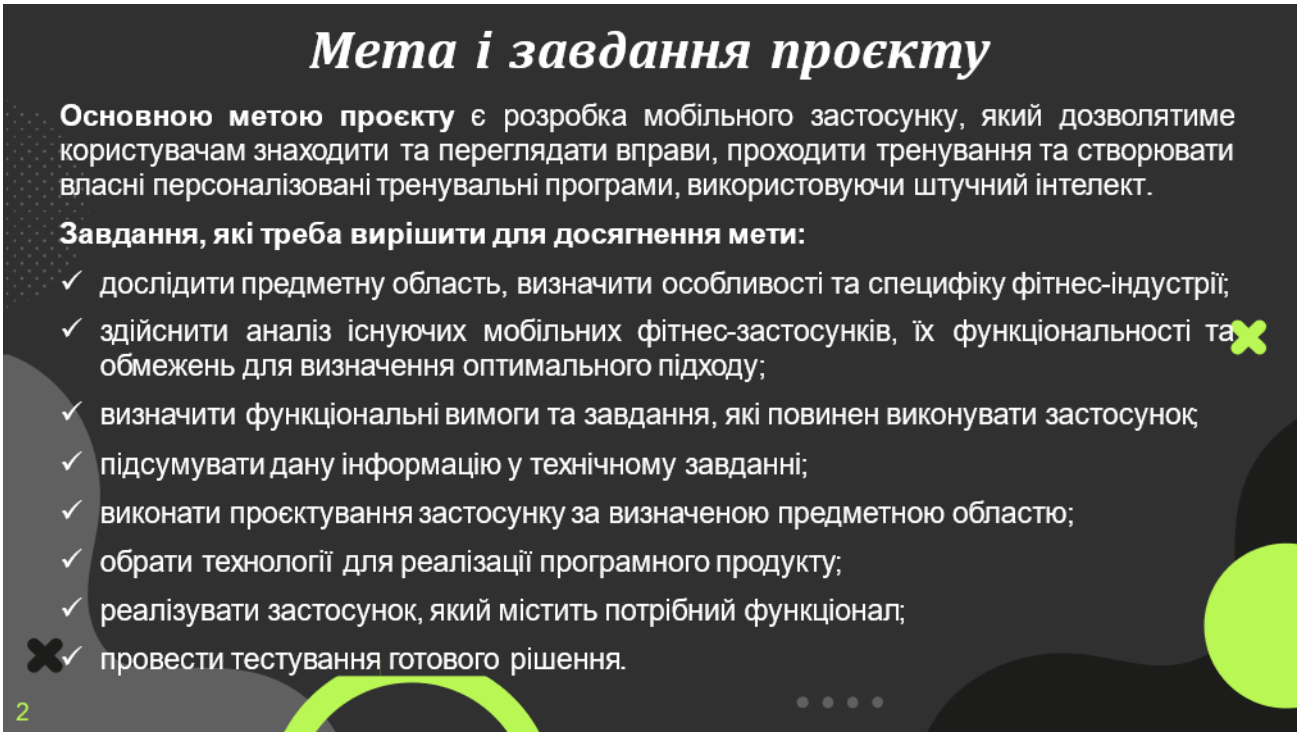
```
import Constants from "expo-constants";

export async function query(data) {
	try {
		const response = await
fetch(`${Constants.expoConfig.extra.HUGGINGFACE_API_ENDPOINT}`, {
			method: "POST",
			headers: {
				Authorization: `Bearer
${Constants.expoConfig.extra.HUGGINGFACE_API_KEY}`,
				"Content-Type": "application/json",
			},
			body: JSON.stringify(data),
		});

		if (!response.ok) {
			throw new Error(`HTTP error! Status: ${response.status}`);
		}

		const result = await response.json();
		return result;
	} catch (error) {
		throw new Error(`Error in API call: ${error.message}`);
	}
}

export async function generateWorkout(prompt) {
	try {
		const generatedWorkoutResponse = await query({
			inputs: prompt,
			parameters: {
				max_new_tokens: 500,
			},
		});

		console.log("Generated AI Workout:", generatedWorkoutResponse);

		const generatedText = generatedWorkoutResponse[0]?.generated_text ||
"";

		const nameMatch = generatedText.match(/\{name:"([^"]+)"/);
		const descriptionMatch =
generatedText.match(/\,description:"([^"]+)/);

		const name = nameMatch ? nameMatch[1] : null;
		const description = descriptionMatch ? descriptionMatch[1] :
generatedText;

		console.log("name", name);
		console.log("desc", description);

		return { name, description };
	} catch (error) {
		console.error("Error generating AI workout:", error);
		throw error;
	}
}
```

ДОДАТОК Д

(обов'язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ





Актуальність теми

Здоровий спосіб життя є однією з найважливіших тем сучасного світу. За статистикою, все більше людей починають стежити за своїм здоров'ям і підтримувати форму, займаючись спортом. Фітнес-індустрія швидко розвивається в усьому світі завдяки своїй комерційній спрямованості – прогнозований обсяг ринку 10,04 млрд доларів США до кінця 2028 року.

Незважаючи на наявність різноманітних мобільних фітнес-застосунків з вправами та готовими тренуваннями, багато з них не забезпечують зручного та персоналізованого підходу до індивідуальних потреб користувачів. Зазвичай такі застосунки обмежені стандартними тренувальними програмами, які не враховують особливостей кожного користувача. Бажання користувачів мати персоналізовані тренування та можливість самостійного створення тренувальних програм може стати ключовим елементом вибору фітнес-застосунків. Тому, на фоні існуючих обмежень, виникає потреба в розробці мобільного застосунку, який виступатиме в ролі віртуального фітнес-тренера та забезпечуватиме можливість користувачам створювати персоналізовані тренування як самостійно, так і з використанням штучного інтелекту.

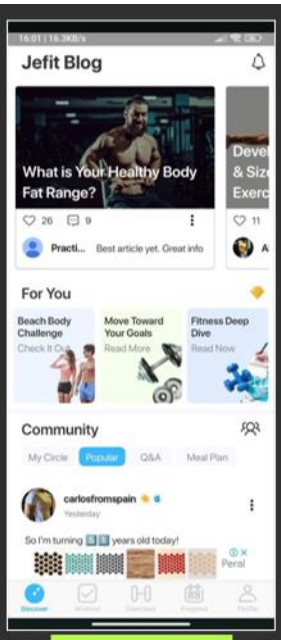
Наявне ПЗ

Недоліки:

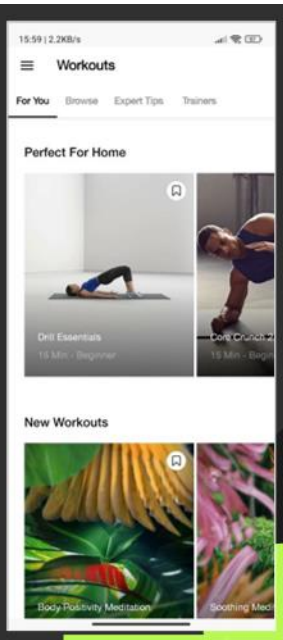
– обов'язкова авторизація;
– обмежений та недостатній функціонал безкоштовної версії;
– надмірна кількість реклами.

Застосунок повинен забезпечувати:

+ інтуїтивний та зручний інтерфейс;
+ створення власних тренувань
+ систему відстеження прогресу;
+ велику базу даних вправ;
+ легку навігацію у застосунку та реалізований пошук;
+ адаптованість до різних операційних систем (буде перевагою).



Jefit

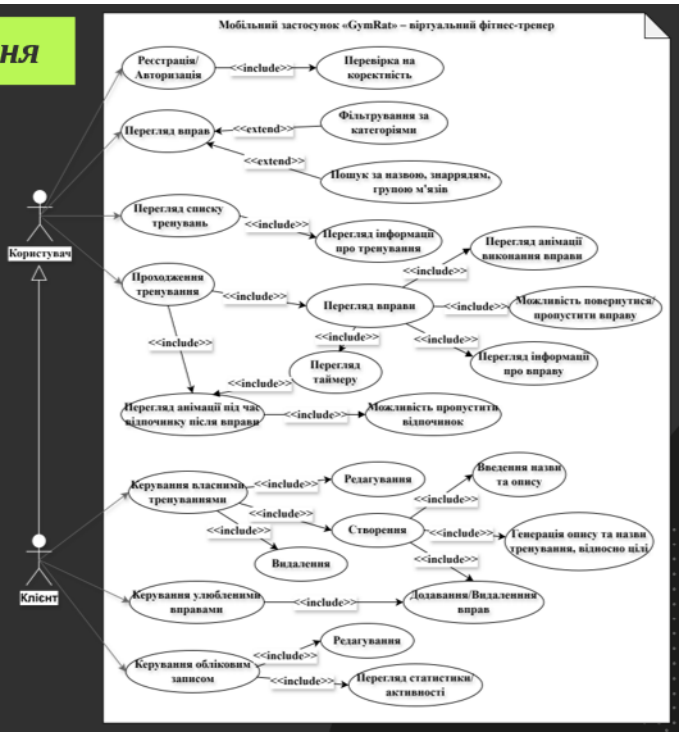


NTC

Діаграма варіантів використання

Функціональне призначення «GymRat»:

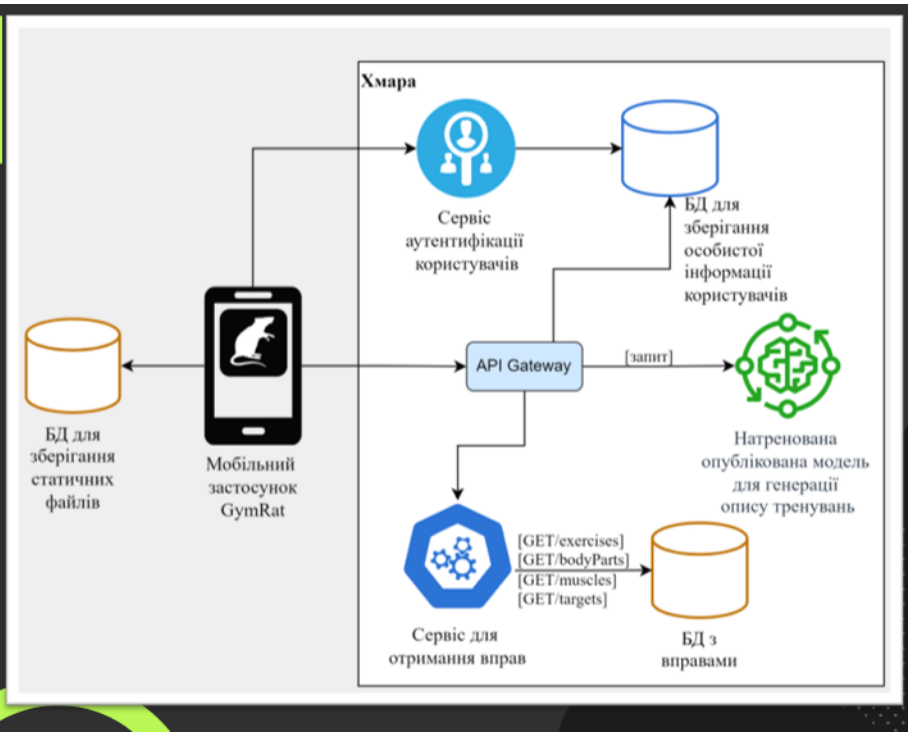
- перегляд різноманітних вправ, поділених на категорії;
- проходження стандартних тренувань;
- збереження улюблених вправ;
- створення власних тренувань, з використанням доступних вправ;
- генерація опису (порад) та назви тренування, під потреби користувача, використовуючи штучний інтелект;
- моніторинг статистики проходження тренувань для відстеження прогресу.

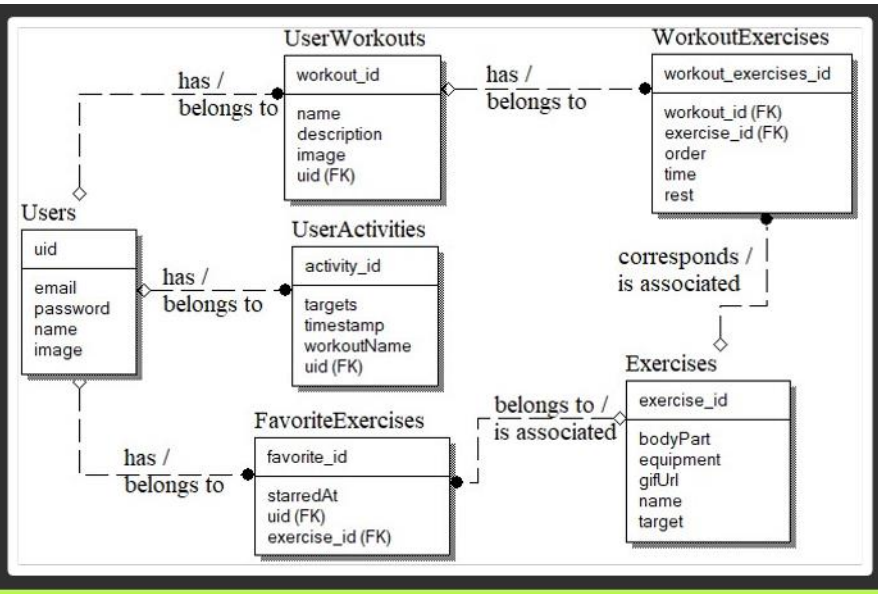


Загальна діаграма архітектури системи

Безсерверна архітектура (Serverless)

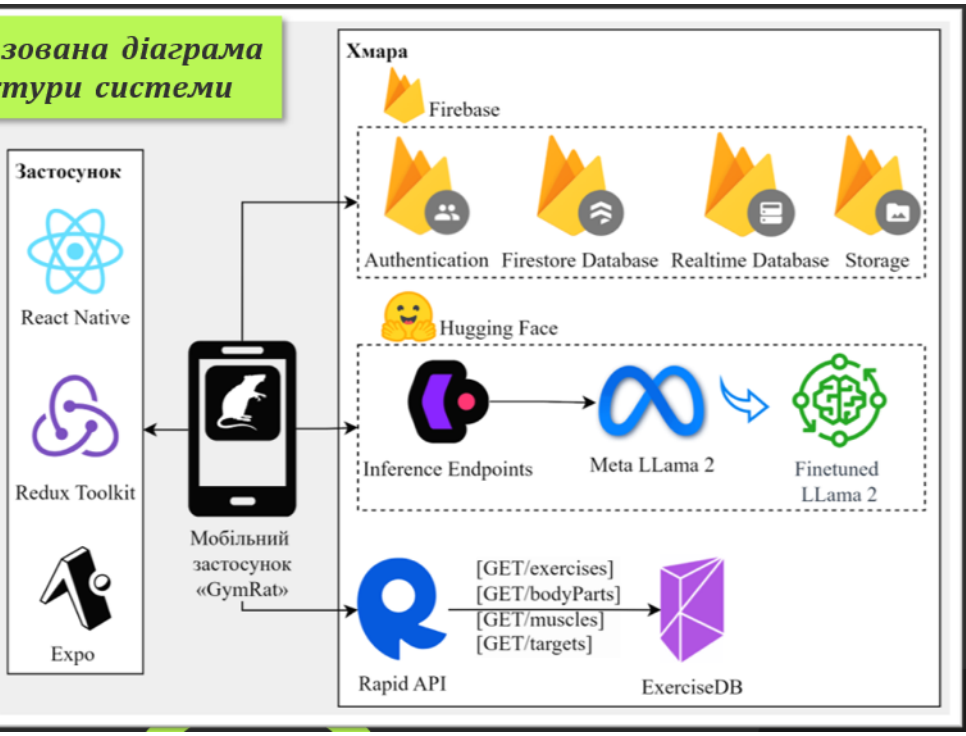
У такій архітектурі всі серверні обчислення та обробка даних відбуваються в хмарних сервісах, а не на власних серверах розробника.





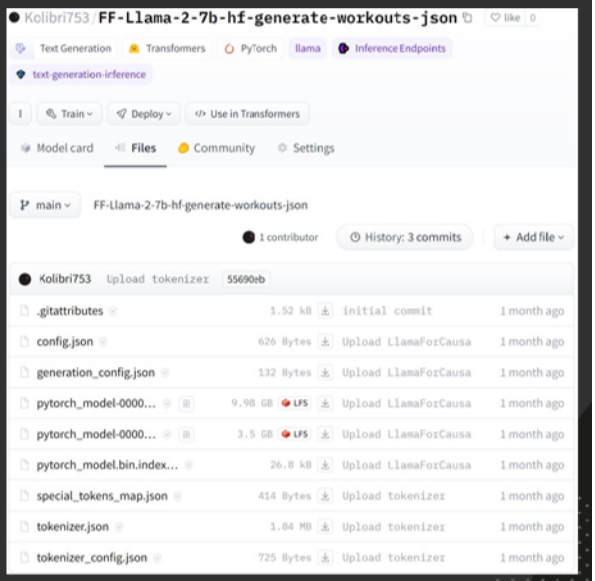
ER - діаграма (логічна модель бази даних)

Конкретизована діаграма архітектури системи



a) **підготовка до тонкого налаштування:**
1) встановлення та завантаження необхідних бібліотек transformers, bitsandbytes , accelerate, peft , trl ;
2) отримання дозволу на використання Llama 2 від Meta та вхід у Hugging Face CLI !huggingface - cli login ;
3) завантаження попередньо натренованої моделі та відповідного токенізатора load_model (model_name , bnb_config);
4) завантаження навчального набору даних load_dataset ("csv", data_files = dataset_name , split = "train") ;
5) попередня обробка навчального набору даних create_prompt_formats , preprocess_batch , preprocess_dataset ;

b) **тонке налаштування:**
1) налаштування PEFT (Parameter Efficient Fine - Tuning) метод QLoRA (квантизація на 4 біти);
2) тонке налаштування попередньо натренованої моделі Llama 2 Trainer() object ;
3) об'єднання ваг та публікування на Hugging Face Hub merge_and_unload , save_pretrained , push_to_hub .



Тренування моделі

Фізичне моделювання даних у базі даних Firebase Firestore .

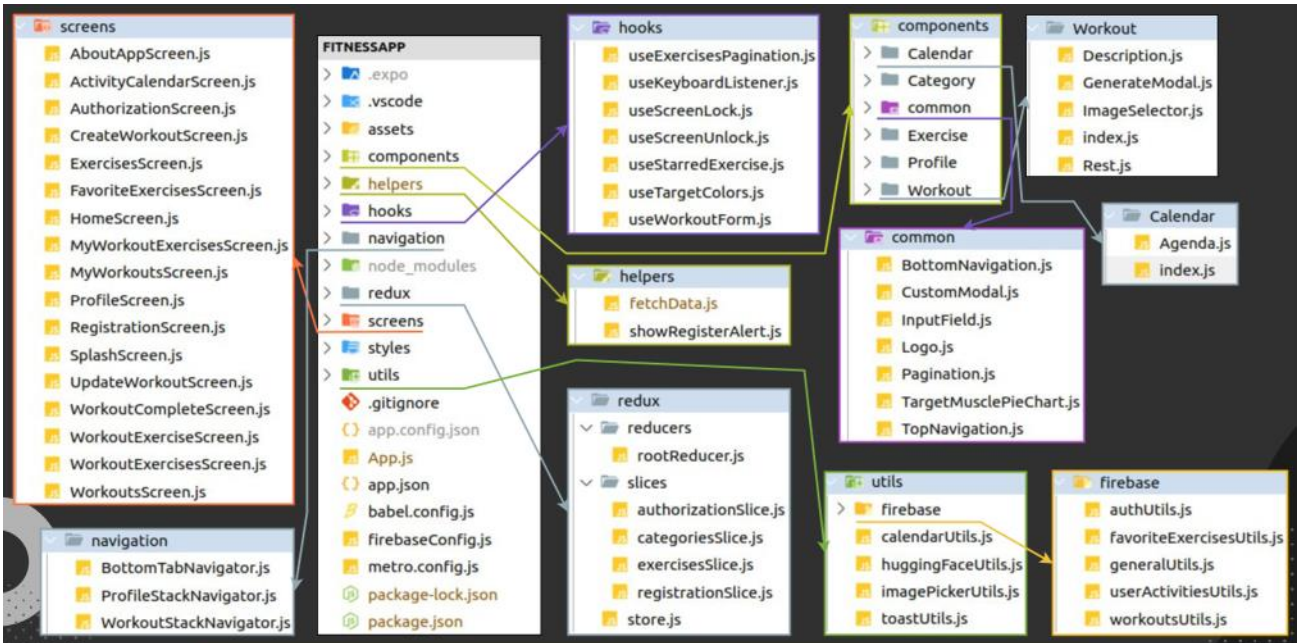
Замість таблиць та рядків, дані організовані у вигляді документів, які групуються в колекції для зручності навігації та управління .

```
2
FavoriteExercises (Collection)
├ {favoriteExerciseId} (Document)
│  ├ {exerciseId} (Document)
│  │  ├ bodypart: "string"
│  │  ├ description: "string"
│  │  ├ equipment: "string"
│  │  ├ gifUrl: "string"
│  │  ├ name: "string"
│  │  ├ target: "string"
│  │  └ starredAt: "timestamp"
```

```
2
UserWorkouts (Collection)
├ {userWorkoutId} (Document)
│  ├ name: "string"
│  ├ description: "string"
│  ├ image: "string"
│  ├ exercises (Collection)
│  │  ├ {exerciseId} (Document)
│  │  │  ├ bodypart: "string"
│  │  │  ├ description: "string"
│  │  │  ├ equipment: "string"
│  │  │  ├ gifUrl: "string"
│  │  │  ├ name: "string"
│  │  │  ├ order: "number"
│  │  │  ├ rest: "number"
│  │  │  ├ target: "string"
│  │  │  └ time: "number"
```

```
1
Users (Collection)
├ {userId} (Document)
│  ├ email: "string"
│  ├ password: "string"
│  ├ name: "string"
│  ├ imageUrl: "string"
│  ├ favoriteExercises: (Reference to Favorite Exercises Collection)
│  ├ userActivities: (Reference to User Activities Collection)
│  └ userWorkouts: (Reference to User Workouts Collection)
```

```
UserActivities (Collection)
├ {userActivityId} (Document)
│  ├ targets: ["string"]
│  ├ timestamp: "timestamp"
│  └ workoutName: "string"
└
```

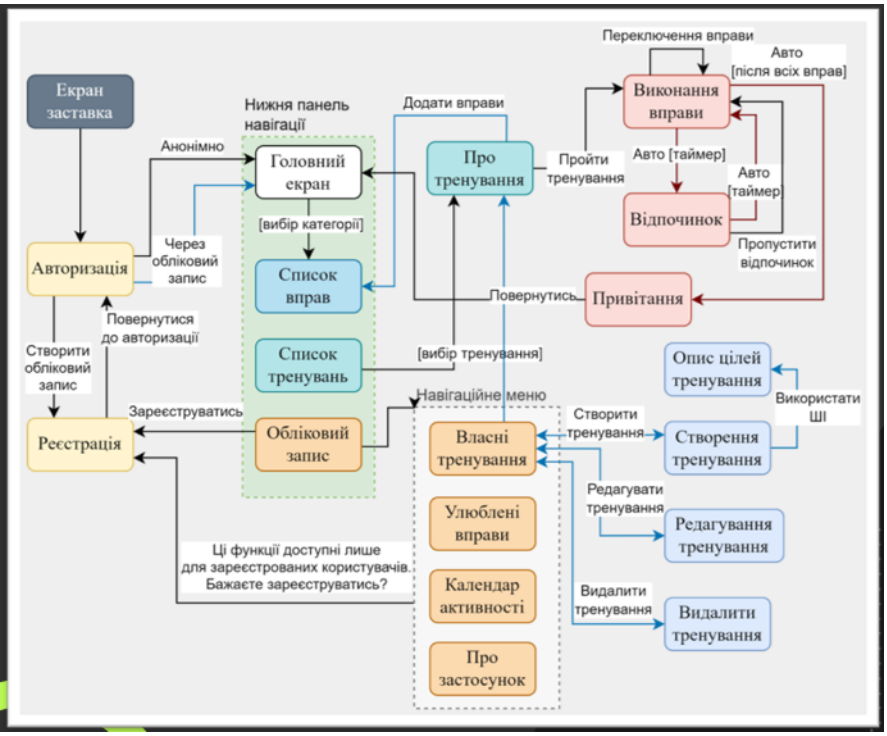


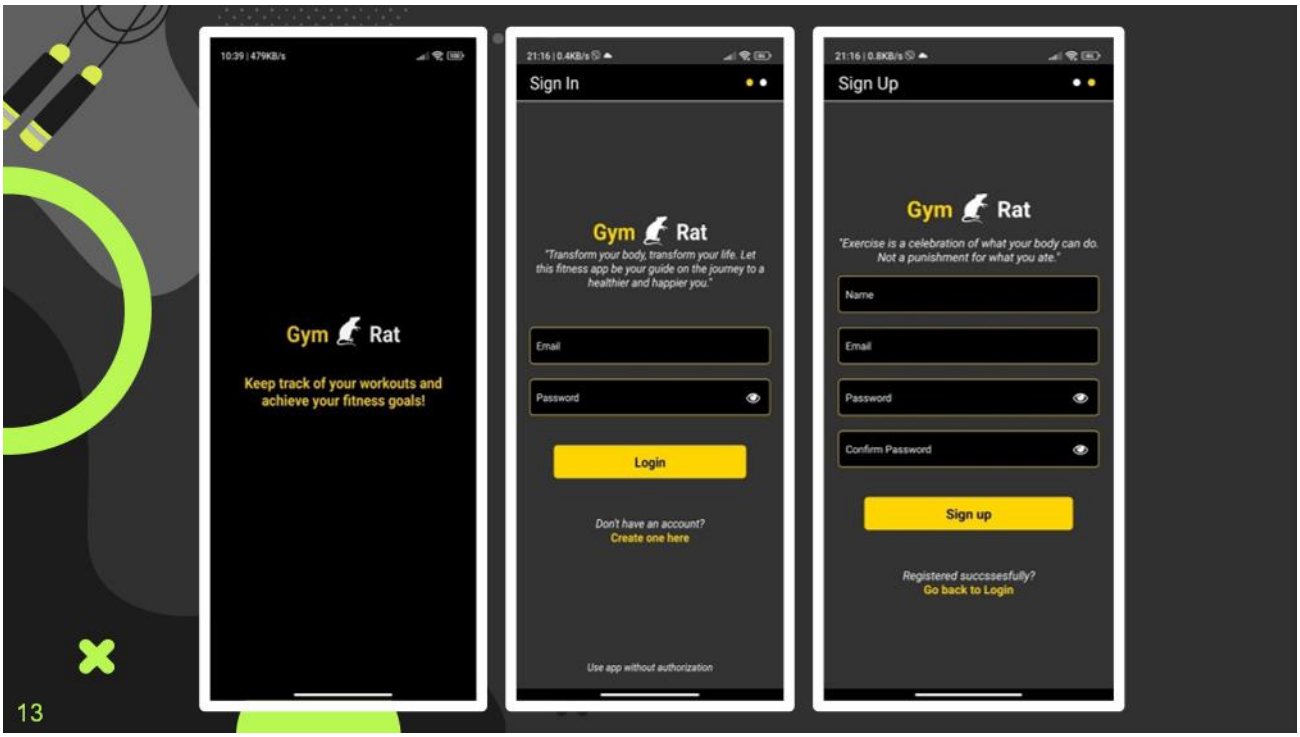
Структура проєкту

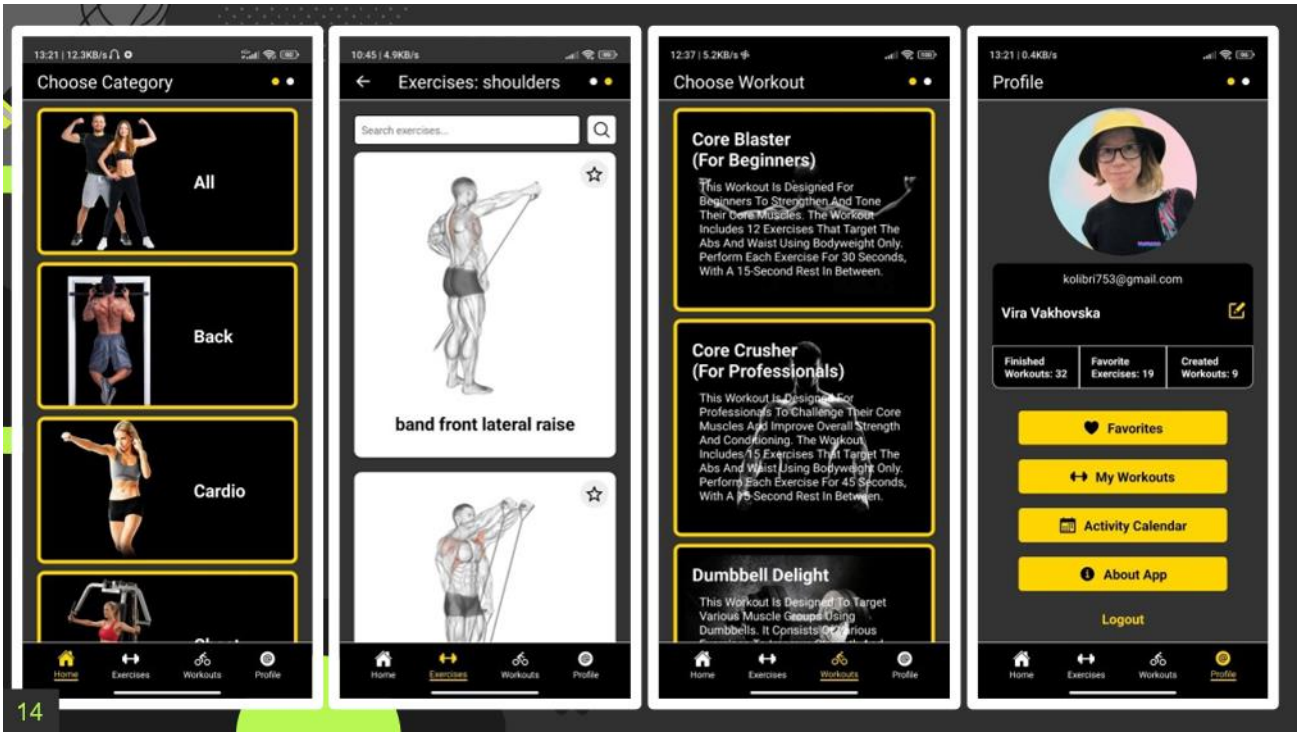
Схема переходів між екранами застосунку

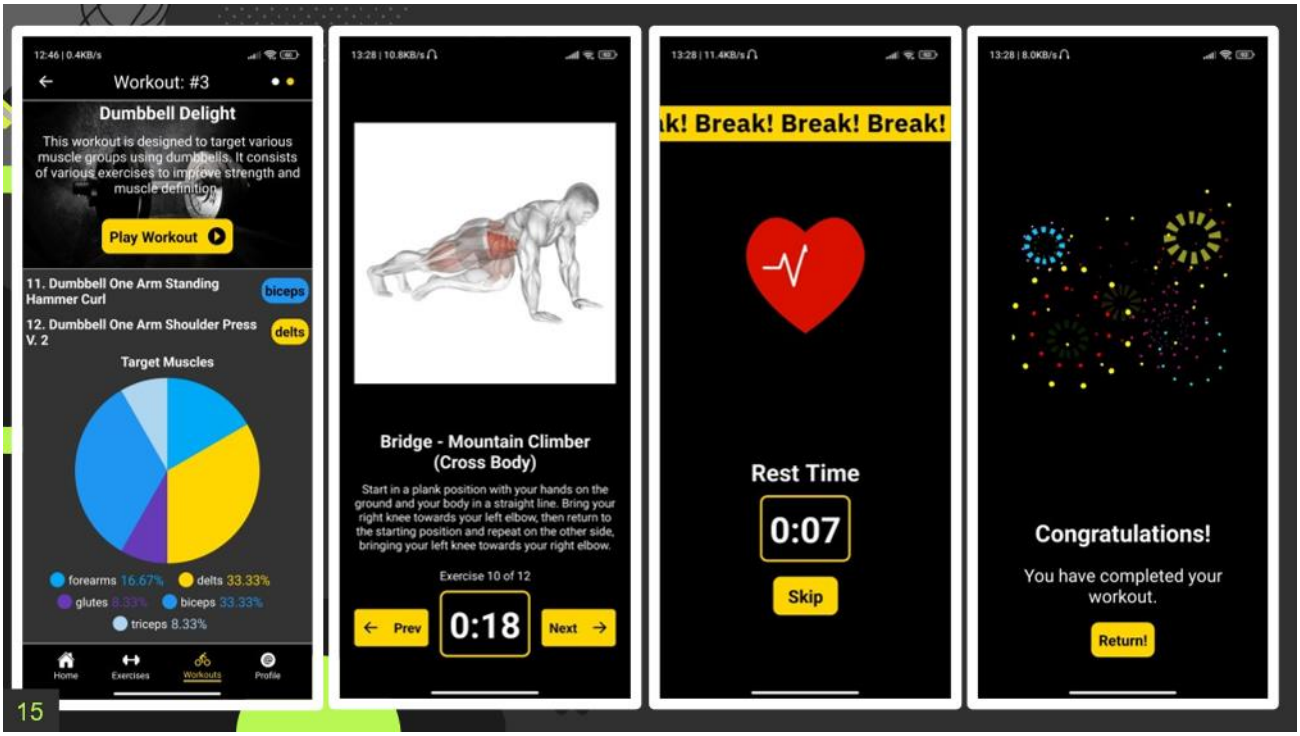
Для кращої візуалізації:

- сумісні екрани зафарбовані одним кольором;
- пунктирною лінією виділені навігаційні елементи, всередині групи, можна переходити на будь-який екран;
- синім кольором позначені кроки доступні лише авторизованому користувачеві.

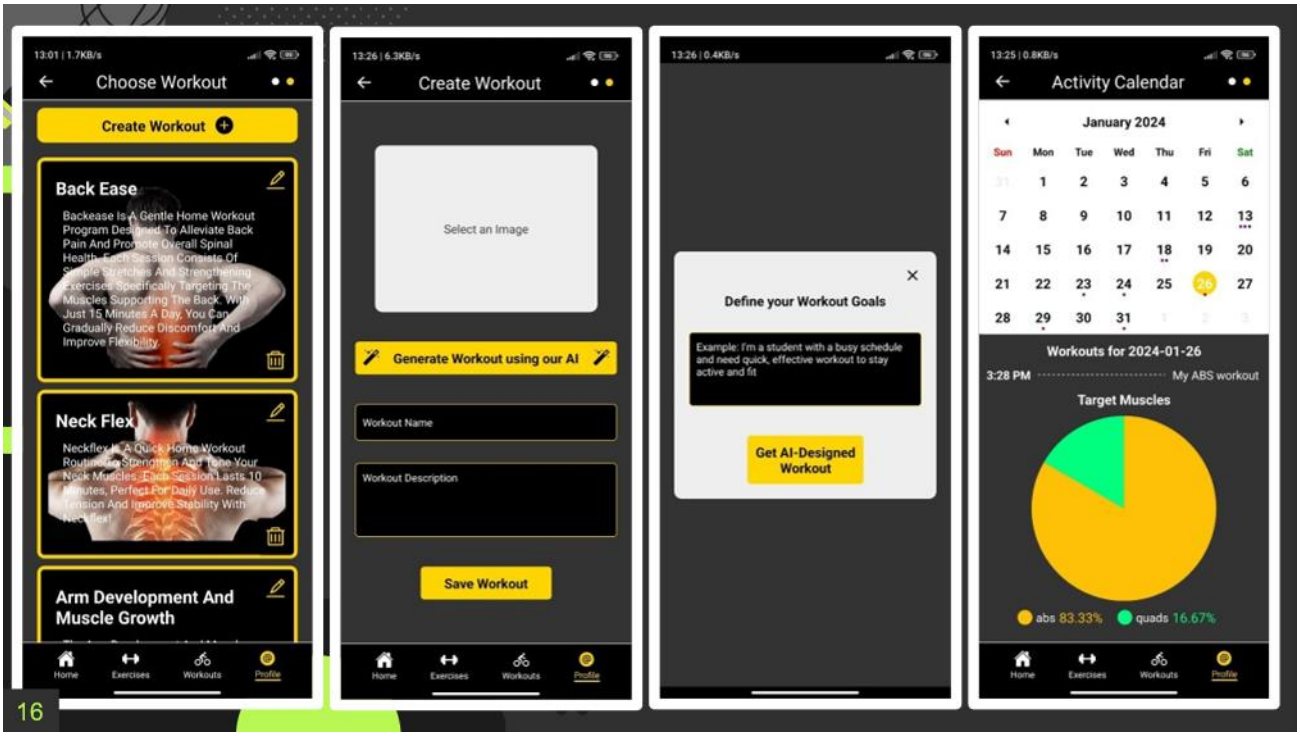



Gym Rat
Keep track of your workouts and achieve your fitness goals!
Sign In
Gym Rat
"Transform your body, transform your life. Let this fitness app be your guide on the journey to a healthier and happier you."
Email
Password
Login
Don't have an account?
Create one here
Use app without authorization
Sign Up
Gym Rat
"Exercise is a celebration of what your body can do. Not a punishment for what you ate."
Name
Email
Password
Confirm Password
Sign up
Registered succssesfully?
Go back to Login
13


Choose Category
All
Back
Cardio
Home
Exercises
Workouts
Profile
Exercises: shoulders
Search exercises...
band front lateral raise
Choose Workout
Core Blaster (For Beginners)
This Workout Is Designed For Beginners To Strengthen And Tone Their Core Muscles. The Workout Includes 12 Exercises That Target The Abs And Waist Using Bodyweight Only. Perform Each Exercise For 30 Seconds, With A 15-Second Rest In Between.
Core Crusher (For Professionals)
This Workout Is Designed For Professionals To Challenge Their Core Muscles And Improve Overall Strength And Conditioning. The Workout Includes 15 Exercises That Target The Abs And Waist Using Bodyweight Only. Perform Each Exercise For 45 Seconds, With A 15 Second Rest In Between.
Dumbbell Delight
This Workout Is Designed To Target Various Muscle Groups Using Dumbbells. It Consists Of Various
Profile
kolibri753@gmail.com
Vira Vakhovska
Finished Workouts: 32
Favorite Exercises: 19
Created Workouts: 9
Favorites
My Workouts
Activity Calendar
About App
Logout
14


Workout: #3
Dumbbell Delight
This workout is designed to target various muscle groups using dumbbells. It consists of various exercises to improve strength and muscle definition.
Play Workout
11. Dumbbell One Arm Standing Hammer Curl
biceps
12. Dumbbell One Arm Shoulder Press V. 2
delts
Target Muscles
forearms 16.67%
delts 33.33%
glutes 8.33%
biceps 33.33%
triceps 8.33%
Home
Exercises
Workouts
Profile
Bridge - Mountain Climber (Cross Body)
Start in a plank position with your hands on the ground and your body in a straight line. Bring your right knee towards your left elbow, then return to the starting position and repeat on the other side, bringing your left knee towards your right elbow.
Exercise 10 of 12
Prev
0:18
Next
Break! Break! Break!
Rest Time
0:07
Skip
Congratulations!
You have completed your workout.
Return!

15


Choose Workout
Create Workout
Back Ease
Backease Is A Gentle Home Workout Program Designed To Alleviate Back Pain And Promote Overall Spinal Health. Each Session Consists Of Simple Stretches And Strengthening Exercises Specifically Targeting The Muscles Supporting The Back. With Just 15 Minutes A Day, You Can Gradually Reduce Discomfort And Improve Flexibility.
Neck Flex
Neckflex Is A Quick Home Workout Routine To Strengthen And Tone Your Neck Muscles. Each Session Lasts 10 Minutes, Perfect For Daily Use. Reduce Tension And Improve Stability With Neckflex!
Arm Development And Muscle Growth
Home
Exercises
Workouts
Profile
Create Workout
Select an Image
Generate Workout using our AI
Workout Name
Workout Description
Save Workout
Define your Workout Goals
Example: I'm a student with a busy schedule and need quick, effective workout to stay active and fit
Get AI-Designed Workout
Activity Calendar
January 2024
Sun Mon Tue Wed Thu Fri Sat
Workouts for 2024-01-26
3:28 PM
My ABS workout
Target Muscles
abs 83.33%
quads 16.67%

16

Тестування

Flipper

[RFC0641]

Expo CLI (eas)

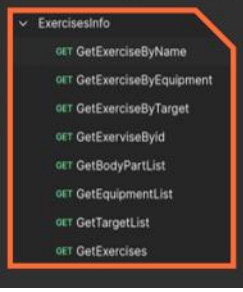
jest + jest-expo

@testing-library/react-native

react-test-renderer

Hugging Face (Inference Endpoints)

Postman

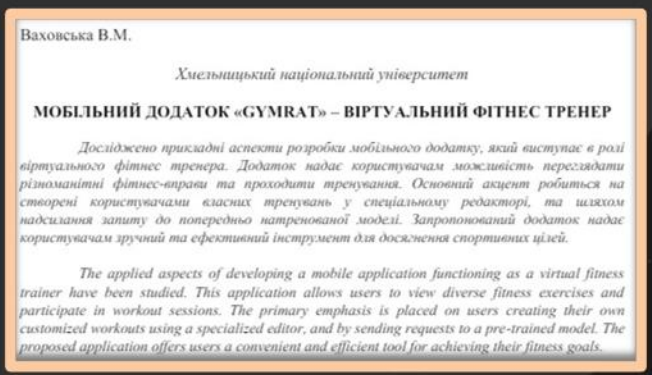


My goal is to be able to perform a pull-up, which I can't do yet. I'd like a workout plan to help me build the necessary upper body strength

{name:"Pull-Up Strength Builder",description:"The Pull-Up Strength Builder workout plan is designed to help you develop the upper body strength needed to perform a pull-up. Whether you're a beginner or trying to increase your pull-up count, this plan will help you build the necessary muscle and achieve your pull-up goal. Patience and consistency are key to success.\n\n**Exercise Recommendations**:\n1. **Warm-up (10 minutes)**: Begin with light cardio exercises such as jumping jacks or jogging in place to raise your heart rate and prepare your muscles.\n\n2. **Lat Pulldowns (2 times a week)**:\n a. Wide-Grip Lat Pulldowns: 3 sets of 10 reps with a 60-second rest between sets.\n b. Close-Grip Lat Pulldowns: 3 sets of 10 reps with a 60-second rest.\n\n3. **Assisted Pull-Ups (3 times a week)**: If you're unable to perform a pull-up yet, use an assisted pull-up machine or resistance bands to reduce body weight.\n a. Assisted Pull-Ups: 3 sets of 5-8 reps with a 60-second rest.\n b. Negative Pull-Ups (focus on the lowering phase): 3 sets of 6-10 reps with a 60-second rest.\n\n4. **Dumbbell Rows (2 times a week)**: These exercises strengthen your back and biceps, supporting pull-up progress.\n a. Bent-Over Dumbbell Rows: 3 sets of 10 reps with a 60-second rest.\n b. Incline Dumbbell Rows: 3 sets of 10 reps with a 60-second rest.\n\n5. **Isometric Holds (2 times a week)**: Isometric exercises help build stability and strength.\n a. Dead Hangs: Hold onto the pull-up bar for as long as you can in each set (aim for 3 sets).\n b. Flexed-Arm Hangs (chin above the pull-up bar): 3 sets of holding as long as possible with a 60-second rest.\n\n6. **Cool Down and Stretching (10 minutes)**: Stretch your arms, shoulders, and back to improve flexibility and reduce muscle tension.\n\n**Total Exercise Time**: Approximately 60-75 minutes per workout

Розробка також була опублікована у збірнику наукових праць за матеріалами XV Всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2023».

Ваховська В. М. МОБІЛЬНИЙ ДОДАТОК «GYMRAT» – ВІРТУАЛЬНИЙ ФІТНЕС ТРЕНЕР. *Актуальні проблеми комп'ютерних наук АПКН-2023* : зб. наук. пр., м. Хмельницький, 17–18 листоп. 2023 р. Хмельницький, 2023. С. 43–46. URL: https://kn.khmnu.edu.ua/wp-content/uploads/sites/ 18/apkn-2023-corpuspaper.pdf



Ваховська В.М.

Хмельницький національний університет

МОБІЛЬНИЙ ДОДАТОК «GYMRAT» – ВІРТУАЛЬНИЙ ФІТНЕС ТРЕНЕР

Досліджено прикладні аспекти розробки мобільного додатку, який виступає в ролі віртуального фітнес тренера. Додаток надає користувачам можливість переглядати різноманітні фітнес-вправи та проходити тренування. Основний акцент робиться на створені користувачами власних тренувань у спеціальному редакторі, та шляхом надсилання запиту до попередньо натренованої моделі. Запропонований додаток надає користувачам зручний та ефективний інструмент для досягнення спортивних цілей.

The applied aspects of developing a mobile application functioning as a virtual fitness trainer have been studied. This application allows users to view diverse fitness exercises and participate in workout sessions. The primary emphasis is placed on users creating their own customized workouts using a specialized editor, and by sending requests to a pre-trained model. The proposed application offers users a convenient and efficient tool for achieving their fitness goals.

Висновки

Під час виконання кваліфікаційної роботи було проведено аналіз предметної області мобільних застосунків для фітнесу та спорту. За даними на січень 2024 року, кількість завантажень таких застосунків > 14 мільйонів по всьому світу, що свідчить про великий попит. Для максимізації охоплення аудиторії був обраний гібридний тип застосунку, аналіз якого включав оцінку функціональності та ідентифікацію обмежень існуючих застосунків з метою визначення оптимальних вимог.

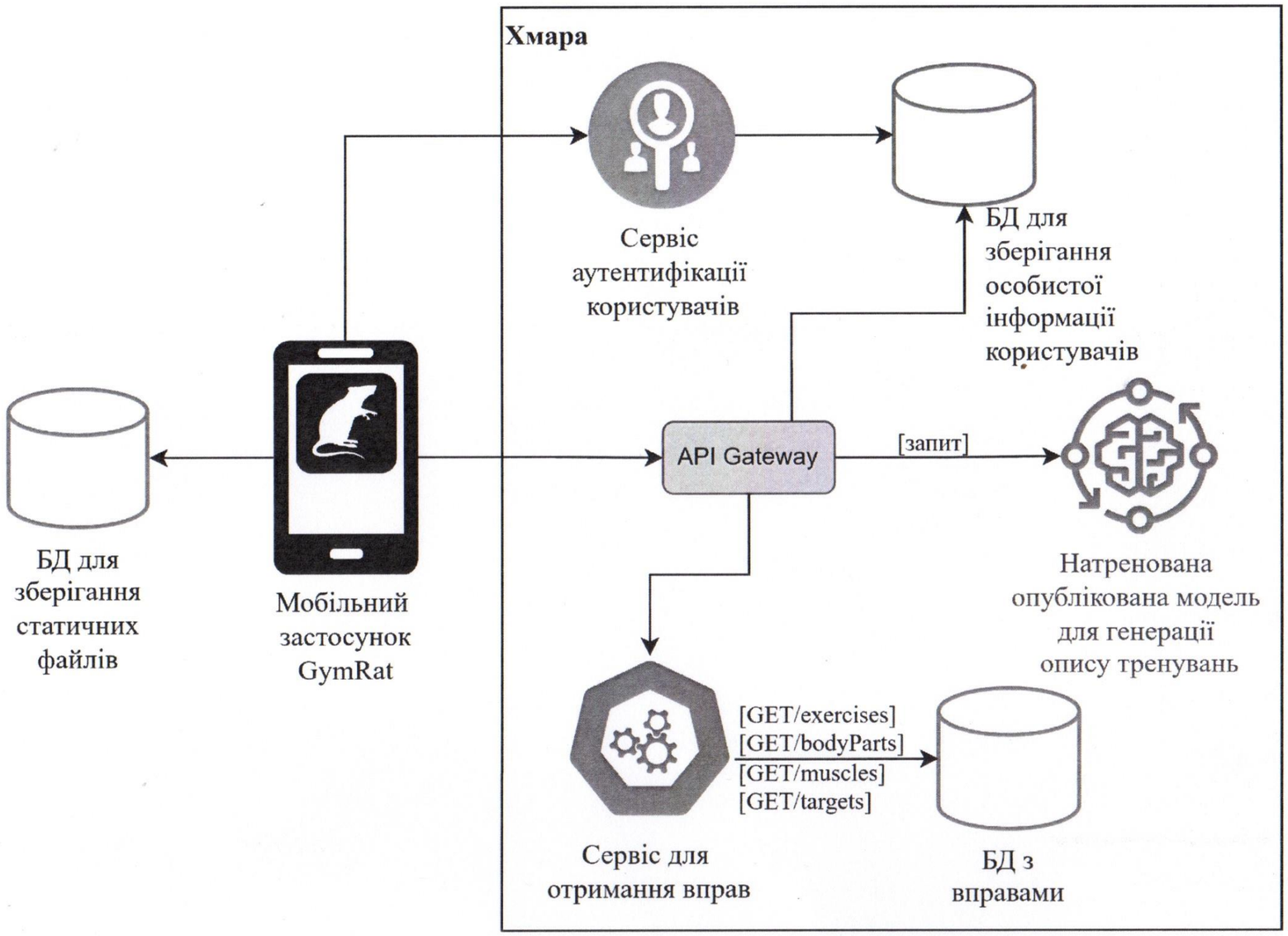
Була розроблена функціональна модель застосунку з використанням стандарту IDEF0 та виділено акторів системи, їх варіанти використання та взаємозв'язки, що стало основою для складання технічного завдання.

В процесі проєктування архітектури було обрано безсерверну архітектуру (Serverless) для зосередження уваги на функціональності. Для самого застосунку було обрано компонентну архітектуру з Redux для керування станом, через його простоту та ефективність.

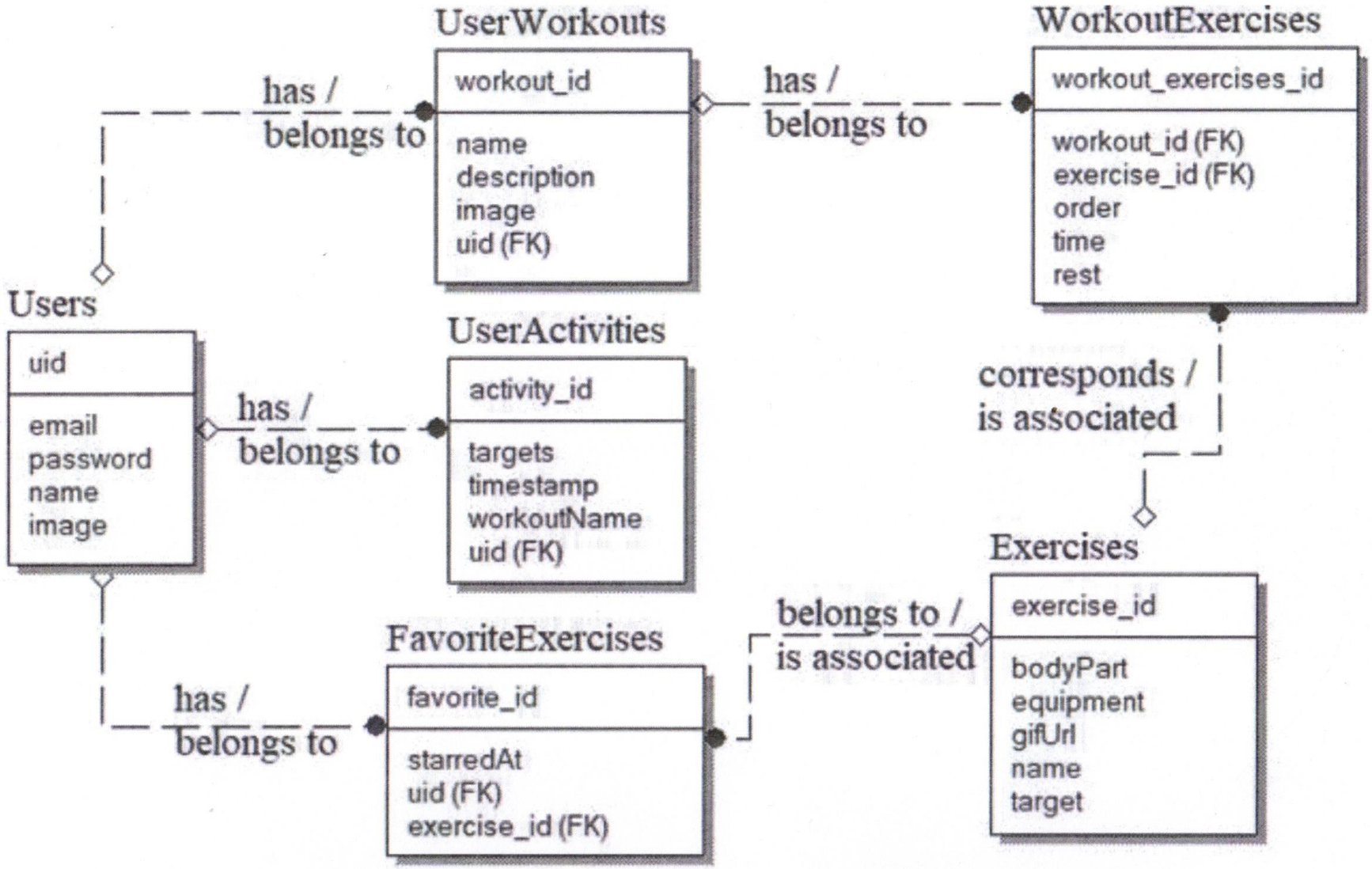
Виконано тренування моделі штучного інтелекту на внутрішніх даних, що призвело до створення тонко налаштованої моделі, яка була розміщена на платформі Hugging Face Hub та опублікована з використанням Inference Endpoints. Фізична БД була реалізована на основі спроєктованої раніше логічної моделі. Проведене тестування підтвердило працездатність системи та її відповідність технічному завданню.

У результаті було розроблено мобільний застосунок «GymRat», який не лише дозволяє переглядати вправи та проходити тренування, але й створювати власні тренувальні програми, з використанням спеціалізованої моделі штучного інтелекту.

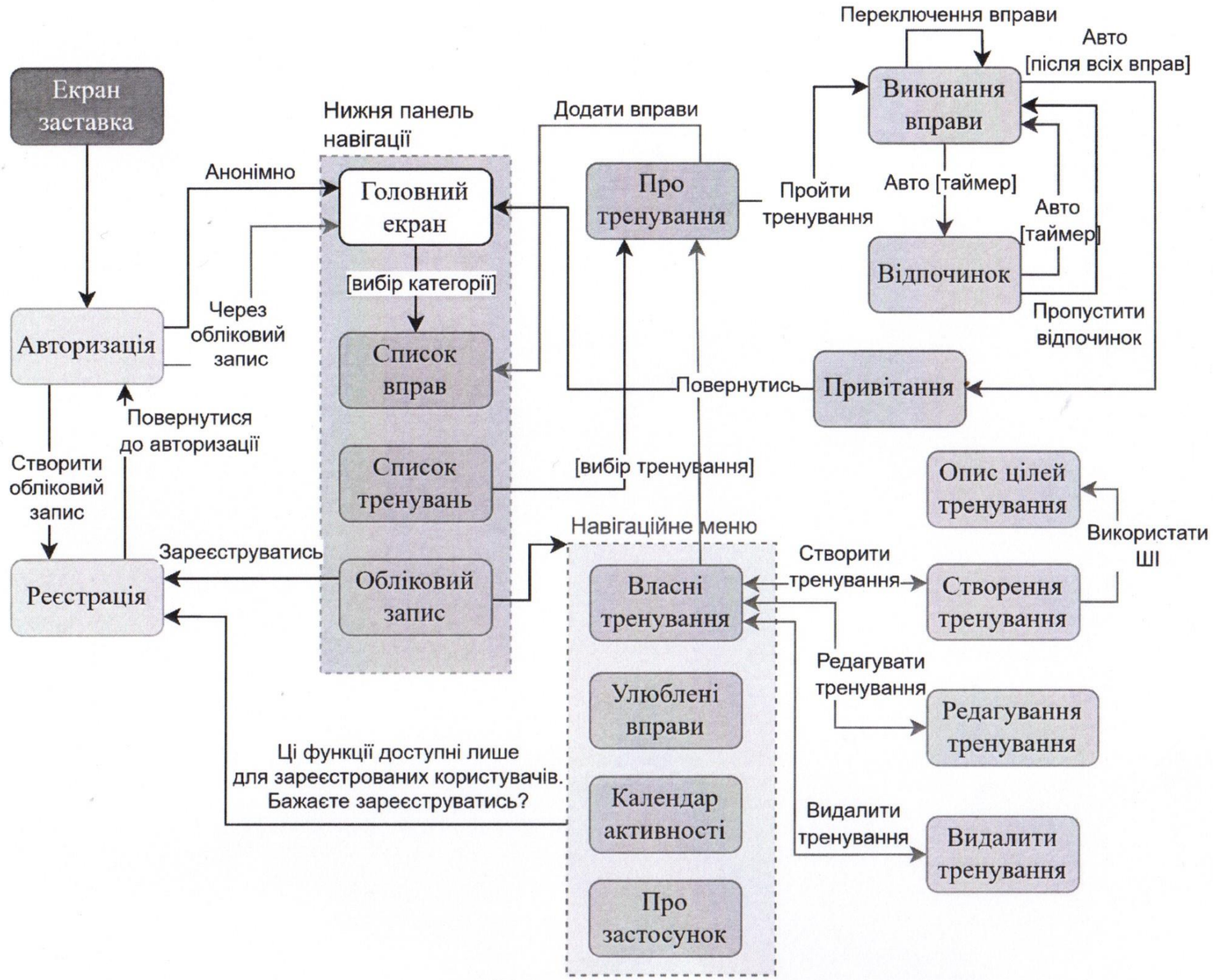
ГРАФІЧНА ЧАСТИНА



					КвРІПЗ.200245.01.04.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок «GymRat» – віртуальний фітнес-тренер	Літера / Маса / Масштаб
Розробив		Ваховська В.М.		6.06		
Керівник		Бедратюк Л.П.		6.06		
					Архітектура системи	Аркуш 1 / Аркушів 3
Н. Контр.		Бедратюк Г.І.		6.06		
Зав. каф.		Бедратюк Л.П.		6.06		



					КвРІПЗ.200245.01.04.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок «GymRat» – віртуальний фітнес-тренер Логічна модель бази даних	Літера / Маса / Масштаб
Розробив		Ваховська В.М.		6.06		
Керівник		Бедратюк Л.П.		6.06		Аркуш 2 / Аркушів 3
Н. Контр.		Бедратюк Г.І.		6.06		
Зав. каф.		Бедратюк Л.П.		6.06		



					КвРІПЗ.200245.01.04.Е8			
Зм.	Арк.	№ докум.	Підпис	Дата	Мобільний застосунок «GymRat» – віртуальний фітнес-тренер	Літера	Маса	Масштаб
Розробив		Ваховська В.М.		6.06				
Керівник		Бедратюк Л.П.		6.06				
					Схема екранів застосунку	Аркуш 3	Аркушів 3	
Н. Контр.		Бедратюк Г.І.		6.06				
Зав. каф.		Бедратюк Л.П.		6.06				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Ваховської В. М.

Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлена. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщена та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

30.05.2024

дата

підпис

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1016314086

Дата перевірки:
03.06.2024 12:09:38 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
03.06.2024 22:35:40 EEST

ID користувача:
100012953

Назва документа: **БКР Мобільний застосунок GymRat Ваховська В _Бедратюк Л.П**

Кількість сторінок: **72** Кількість слів: **13499** Кількість символів: **108024** Розмір файлу: **4.55 MB** ID файлу: **1016111291**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.44%
Схожість

Найбільша схожість: **1.51%** з джерелом з Бібліотеки (ID файлу: **1008408523**)

5.44% Джерела з Інтернету	648	Сторінка 74
3.76% Джерела з Бібліотеки	255	Сторінка 79

0.79% Цитат

Цитати	6	Сторінка 80

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	2
Підозріле форматування	26 сторінок

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 17%**

	Документ		Сумарний збіг по Базі Даних	
ID: 128056 Назва: БКР Мобільний застосунок «GymRat» Додано в БД: 2024-06-03 Автора: ВАХОВСЬКА Віра Керівники: БЕДРАТЮК Леонід Консультанти: д-р фіз.-мат. наук, професор Опоненти:	Символи	Лексеми	Символи	Лексеми
	84423	1251	4889 (6%)	66 (5%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Ваховська Віра Миколаївна

Тема Мобільний застосунок «GymRat» – віртуальний фітнес-тренер

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 69

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі Ваховської Віри Миколаївни розглянуто створення мобільного застосунку «GymRat» для фітнесу, який включає можливості створення персоналізованих тренувальних програм, використовуючи штучний інтелект. Проєкт охоплює аналіз предметної області, проєктування архітектури системи, програмну реалізацію та тестування.

2. Висновок про відповідність роботи поставленому завданню Робота повністю відповідає поставленому завданню. Виконано всі необхідні етапи розробки: дослідження предметної області, проєктування, реалізацію та тестування мобільного застосунку.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи Під час дослідження предметної області було виконано глибокий аналіз фітнес-індустрії та існуючих мобільних застосунків, виявлено їхні недоліки та визначено вимоги до нового продукту. Проєктування програмного забезпечення охоплювало розробку архітектури системи, логічної моделі бази даних та інтерфейсу користувача. При цьому використано методологію IDEF1X для проєктування бази даних та концепцію безсерверної архітектури. Програмна реалізація включала створення функціоналу мобільного застосунку за допомогою React Native, Expo та Redux Toolkit. Для зберігання даних і автентифікації користувачів використано Firebase, для отримання вправ – ExerciseDB з маркетплейсу RapidAPI, а для генерації опису тренувань – модель штучного інтелекту від Hugging Face. Тестування системи передбачало детальне тестування функціоналу застосунку, включаючи перевірку роботи зовнішніх сервісів. Використано сучасні технології розробки, такі як React Native та Firebase, інтеграція моделі штучного інтелекту для персоналізації тренувань демонструє високий рівень інноваційності. Безсерверна архітектура забезпечує ефективне використання ресурсів та спрощує масштабування системи, а Redux забезпечє централізоване зберігання даних та спрощує роботу з ними.

4. Позитивні сторони роботи Використання сучасних технологій і передових методів розробки сприяє ефективності процесів. Високий рівень деталізації проєктування та реалізації дозволяє досягти високої точності у виконанні завдань. Інноваційний підхід до персоналізації тренувань за допомогою штучного інтелекту відкриває нові можливості для індивідуального підходу до тренувань. Комплексне тестування забезпечує високу якість кінцевого продукту, що гарантує задоволення потреб користувачів.

5. Негативні сторони роботи Деякі аспекти роботи з оптимізацією продуктивності могли бути розглянуті більш детально, зокрема питання оптимізації швидкості завантаження даних з серверу та зменшення часу відгуку системи.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення та пояснювальна записка виконані на високому рівні, містять усі необхідні схеми, діаграми та ілюстрації, що робить їх зрозумілими і зручними для користувачів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота Ваховської Віри Миколаївни є завершеним проєктом, який демонструє високий рівень знань та навичок у галузі інженерії програмного забезпечення. Робота відповідає поставленим завданням та вимогам, містить теоретичні та практичні результати, які можуть бути використані в подальшій розробці мобільних застосунків.

8. Інші зауваження Рекомендується додати більше тестових сценаріїв для різних ситуацій використання, що дозволить більш повно оцінити стабільність та функціональність застосунку.

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота заслуговує на оцінку «відмінно» за змістом, якістю виконання та відповідністю поставленим завданням

РЕЦЕНЗЕНТ Говорущенко Тетяна Олександрівна, доктор технічних наук, професор, зав. кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

"6" 06 202 4 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Мобільний застосунок «GymRat» – віртуальний фітнес-тренер»

Автор: Ваховська Віра Миколаївна
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Бедратюк Леонід Петрович, д-р фіз.-мат. наук, професор
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, завдання, анотація, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій та у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 6,44% і адресується до 903 джерел, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 4.06.2024

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Леонід БЕДРАТЮК