НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___» _____ 2025 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Мобільний застосунок «Temporun» для тренувань з бігу»

Виконав:
студент IV курсу, групи КП-12
Заварзін Олександр Олександрович _____

Керівник:
доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
доцент каф. СПіСКС ФПМ, к.т.н, доцент
Тарасенко-Клятченко Оксана Володимирівна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Євгенія СУЛЕМА
«___»_____2024 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Заварзіну Олександру Олександровичу

1. Тема проєкту «Мобільний застосунок «TempoRun» для тренувань з бігу», керівник проекту Заболотня Тетяна Миколаївна, доцент кафедри ПЗКС, к.т.н., доцент, затверджені наказом по університету №1808-С від «29» травня 2025 р.

2. Термін подання студентом проєкту «13» червня 2025 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   – аналіз предметної галузі та існуючих рішень;
   – аналіз мов програмування та технологій розроблення мобільних застосунків;
   – розроблення мобільного застосунку для тренувань з бігу;
   – аналіз розроблених алгоритмів та підпрограм;
   – аналіз розробленого мобільного застосунку.

5. Перелік обов'язкового графічного матеріалу:
   – алгоритм трансформації фітнес-даних у цільовий музичний темп (креслення);
   – алгоритм добору музичного супроводу (креслення);
   – архітектура мобільного застосунку (плакат);
   – дерево проблем потенційних користувачів системи (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2024 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	01.12.2024	
2.	Розроблення та узгодження технічного завдання	16.12.2024	
3.	Розроблення структури мобільного застосунку	25.12.2024	
4.	Підготовка матеріалів першого розділу дипломного проєкту	08.01.2025	
5.	Розроблення дизайну застосунку та графічних елементів	01.02.2025	
6.	Підготовка матеріалів другого розділу дипломного проєкту	19.02.2025	
7.	Програмна реалізація мобільного застосунку	20.03.2025	
8.	Тестування мобільного застосунку	25.03.2025	
9.	Підготовка матеріалів третього розділу дипломного проєкту	15.04.2025	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	28.04.2025	
11.	Підготовка графічної частини дипломного проєкту	14.05.2025	
12.	Оформлення документації дипломного проєкту	26.05.2025	

Студент Олександр ЗАВАРЗІН

Керівник проєкту Тетяна ЗАБОЛОТНЯ

АНОТАЦІЯ

Дипломний проєкт присвячений розробленню мобільного застосунку для тренувань з бігу, який поєднує дані про фізичну активність користувача зі стримінговим сервісом Spotify, щоб персоналізувати музичний добір під поточні фітнес характеристики користувача. Актуальність роботи зумовлена тим, що чинні рішення або зосереджені лише на вимірюванні фітнес-показників, або відтворюють власний контент, не даючи можливості інтегрувати персональні плейлисти зі сторонніх платформ.

Застосунок реалізується за клієнт-серверною схемою. Крос-платформний клієнт, створений у середовищі Expo, зчитує темп, дистанцію та геодані користувача, а також інтегрується зі Spotify через OAuth 2.0 для імпорту плейлистів і керування відтворенням треків. Серверна частина побудована на хмарній платформі Firebase, що забезпечує автентифікацію, зберігання даних у Firestore та виконання допоміжних Cloud Functions без адміністрування інфраструктури.

Алгоритм добору працює у чотирьох режимах – Timer, Map, Length і Pace. Кожен режим перетворює відповідний індикатор (час, маршрут, відстань або поточний темп) у цільове значення музичного темпу, фільтрує композиції плейлисту і формує чергу відтворення, що динамічно адаптується до змін активності користувача. Інтерфейс дає змогу налаштовувати параметри режиму, запускати й зупиняти спортивну сесію, а після її завершення переглядати узагальнені показники часу та дистанції.

Метою проєкту є розроблення програмного забезпечення для тренувань з бігу, що надасть користувачам можливість підбирати музичний супровід з плейлиста у відповідності до фітнес-характеристик бігу.

У даному дипломному проєкті розроблено: архітектуру серверної та клієнтської частин системи, клієнтську частину у вигляді мобільного застосунку, а також алгоритми підбору музичного супроводу під поточні фітнес показники користувача.

ABSTRACT

The diploma project is dedicated to the development of a mobile application for running training that combines data on the user's physical activity with the Spotify streaming service to personalize the music selection to the user's current fitness characteristics. The relevance of the work is that existing solutions either focus only on measuring fitness indicators or play their own content, without the ability to integrate personal playlists from third-party platforms.

The application is implemented using a client-server scheme. The cross-platform client, built in the Expo environment, reads the user's pace, distance, and geodata, and integrates with Spotify via OAuth 2.0 to import playlists and control track playback. The server side is built on the Firebase cloud platform, which provides authentication, data storage in the Firestore, and auxiliary Cloud Functions without infrastructure administration.

The selection algorithm works in four modes – Timer, Map, Length, and Pace. Each mode converts the corresponding indicator (time, route, distance, or current pace) into the target value of the music tempo, filters the playlist songs, and creates a playlist queue that dynamically adapts to changes in user activity. The interface allows you to customize the mode parameters, start and stop a sports session, and view a summary of time and distance data after it is completed.

The goal of the project is to develop software for running training that will allow users to select music from an imported playlist in accordance with their current fitness characteristics.

In this diploma project, the following were developed: the architecture of the server and client parts of the system, the client side in the form of a mobile application, as well as algorithms for selecting music for the user's current fitness performance.

ДП.045440-01-90 Мобільний застосунок «TempoRun» для тренувань з бігу.
Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Мобільний застосунок	5	
	«TempoRun» для		
	тренувань з бігу.		
	Технічне завдання		
ДП.045440-03-81	Мобільний застосунок	70	
	«TempoRun» для		
	тренувань з бігу.		
	Пояснювальна записка		
ДП.045440-04-51	Мобільний застосунок	4	
	«TempoRun» для		
	тренувань з бігу.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Мобільний застосунок	19	
	«TempoRun» для		
	тренувань з бігу.		
	Керівництво		
	користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Мобільний застосунок	1	
	«TempoRun» для		
	тренувань з бігу.		
	Трансформація		
	фітнес-даних у		
	цільовий музичний		
	темп. Блок-схема		
	алгоритму		
ДП.045440-07-99	Мобільний застосунок	1	
	«TempoRun» для		
	тренувань з бігу.		
	Підбір музичного		
	супроводу під час бігу.		
	Блок-схема алгоритму		
ДП.045440-08-98	Мобільний застосунок	1	
	«TempoRun» для		
	тренувань з бігу.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК «TEMPORUN» ДЛЯ ТРЕНУВАНЬ З БІГУ

Технічне завдання

ДП.045440-02-91

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олександр ЗАВАРЗІН

2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: мобільний застосунок «TempoRun» для тренувань з бігу.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання людьми, які займаються бігом, велосипедними тренуваннями, спортивною ходьбою та іншими видами навантажень на витривалість і прагнуть персоналізувати процес добору музичного супроводу під час тренувань. Користувачі отримують змогу підключити власні стримінгові сервіси, налаштовувати алгоритми підбору треків відповідно до поточних фітнес-показників (темп, час, дистанція, місцеположення), а також переглядати детальну статистику тренувань.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Мобільний застосунок повинен забезпечувати наступні функції:

1) можливість обробки фітнес-даних користувача: темп, дистанція, тривалість сесії, місцеположення;
2) підтримка інтеграції зі Spotify з імпортом плейлистів і керуванням програвачем;

3) наявність алгоритму підбору треків із режимами за темпом, місцеположенням, часом або дистанцією;
4) можливість налаштування параметри, за якими алгоритм підбору буде опрацьовувати обраний тип фітнес-даних користувача;
5) наявність синхронізації профілю для автоматичного відновлення даних після входу на іншому пристрої.

Розробку виконати на мобільній платформі використовуючи фреймворк React Native.

Додаткові вимоги:

1) наявність авторизації використовуючи акаунт Google;
2) наявність відображення поточних фітнес-показників під час тренування в користувацькому інтерфейсі;
3) наявність можливості зміни кольорової теми застосунку (Світла, Темна);
4) наявність можливості зміни мови застосунку (Англійська, Українська);
5) наявність екрана підсумків з результатами після завершення сесії.

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

Після виконання проєкту повинно бути написана наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Трансформація фітнес-даних у цільовий музичний темп. Схема алгоритму»;
   - «Добір музичного супроводу. Схема алгоритму».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи 01.12.2024
Розроблення та узгодження технічного завдання 16.12.2024
Розроблення структури мобільного застосунку 25.12.2024
Розроблення дизайну застосунку та графічних елементів..... 01.02.2025
Програмна реалізація мобільного застосунку 20.03.2025
Тестування мобільного застосунку 25.03.2025
Підготовка матеріалів текстової частини проєкту 28.04.2025
Підготовка матеріалів графічної частини проєкту..... 14.05.2025
Оформлення технічної документації проєкту 26.05.2025

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"
Завідувач кафедри
_____ Євгенія СУЛЕМА
"___" _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК «TEMPORUN» ДЛЯ ТРЕНУВАНЬ З БІГУ

Пояснювальна записка

ДП.045440-03-81

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олександр ЗАВАРЗІН

2025

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення.

IT – Information Technology (інформаційні технології).

Стримінговий музичний сервіс – сервіс, що надає безперервний доступ до власної музичної бібліотеки за визначену ціну.

Фітнес характеристики – характеристики спортивної сесії користувача (час сесії, темп переміщення, пройдена дистанція, тощо).

Геолокація – географічне місцеположення.

Плейлист – сукупність музичних треків у певній послідовності.

Гейміфікація – залучення ігрових практик для покращення монотонних довгих, або не цікавих дій для кінцевого користувача.

HTTP – HyperText Transfer Protocol (протокол передавання гіпертекстових документів).

REST – Representational State Transfer (стиль проєктування вебсервісів).

API – Application Programming Interface (інтерфейс програмування застосунків).

JVM – Java Virtual Machine (віртуальна машина Java).

Композит – елемент який складається з інших елементів у певній композиції.

Ко-рутини – конструкції в мові програмування Kotlin для асинхронного програмування.

LLVM – Low Level Virtual Machine (універсальна система оптимізації програм завдяки реалізації віртуальної машини як оптимізувального компілятора).

UI – User Interface (інтерфейс користувача).

GPU – Graphics Processing Unit (Графічний процесор).

Нативний – написаний системною мовою, що дозволяє мати найкращу продуктивність що може надавати ця система.

Фреймворк – інфраструктура програмного рішення з готовим набором інструментарію, що полегшую розробку ПЗ.

SDK – Software Development Kit (набір засобів розробника ПЗ).

CI/CD – Continuous Integration/Continuous Delivery (безперервна інтеграція та доставка).

KMM – Kotlin Multiplatform Mobile (мобільна платформа Kotlin, що підтримується на декількох платформах).

JSX – JavaScript XML (Розширення синтаксису мови програмування JavaScript, що дозволяє писати HTML елементи в коді бізнес логіки).

CLI – Command Line Interface (Текстова консоль для вводу відповідних команд для роботи з системою).

AR – Augmented reality (Доповнена реальність).

BaaS – Backend as a Service (Сервіс, що надає можливість швидкої побудови серверної частини завдяки готовим рішенням та інструментам розробки).

OAuth – Open Authorization (протокол делегованої авторизації).

MFA – Multi-Factor Authentication (багатофакторна автентифікація).

СУБД – Система Керування Базами Даних.

MVP – Minimum Viable Product (продукт з мінімальним функціоналом, який забезпечує основний функціонал ПЗ).

UML – Unified Modeling Language (стандарт, що використовує графічні елементи для відображення абстрактної моделі системи).

BPMN – Business Process Model and Notation (стандарт, що використовує графічні елементи для відображення бізнес процесів).

UID – Unique identifier (унікальний ідентифікатор).

AI – Artificial Intelligence (штучний інтелект).

ВСТУП

Сучасний світ перебуває в періоді активного розвитку інформаційних технологій, які все більше інтегруються в повсякденне життя. Особливо це помітно у сфері спорту та здорового способу життя, де інновації стають ключовим інструментом для підвищення ефективності тренувань і створення персоналізованих рішень для користувачів.

Здоровий спосіб життя стає не лише трендом, а й важливою частиною життя мільйонів людей. Водночас розвиток носимих пристроїв, стримінгових сервісів та алгоритмів штучного інтелекту відкриває нові можливості для автоматизації процесів тренувань. І одним із ключових аспектів такого розвитку є створення технологій, які дозволяють об’єднувати дані про фізичну активність із розважальним контентом, зокрема музикою.

Персоналізовані спортивні рішення підвищують ефективність тренувань і комфорт користувачів, поєднуючи дані активності з можливостями сучасних IT-технологій. Вони задають нові стандарти ринку, так як наявні сервіси здебільшого зосереджені на точному вимірі фітнес-показників і майже не інтегруються з музичними платформами. Якщо інтерактивний розважальний контент наявний в спортивному застосунку, то він переважно власного виробництва, і користувач не має можливості його змінювати.

Даний дипломний проєкт присвячено розробленню мобільному застосунку TempoRun, метою якого є покращення взаємодії користувачів зі стримінговими платформами під час сесії спорту, завдяки об’єднанню монотонних вправ на витривалість з персоналізованим підбором музики під поточні фітнес-характеристики користувача.

1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

В першому розділі пояснювальної записки подано інформацію про проблеми, наявні у людей, що полюбляють займатися бігом прослуховуючи музику з обраного музичного стримінгового сервісу, а також наведено результати огляду і порівняння наявних існуючих програмних рішень. В рамках цього розділу виконано наступні завдання:

1) визначення основної проблеми, що характерна для бігунів, які слухають музику зі стримінгових сервісів під час бігу;
2) здійснення аналізу найпопулярніших програмних рішень, що пропонують автоматизований підбір музики під час бігу, а також визначення їх переваг й недоліків;
3) складення порівняльної таблиці з результатами аналізу наявних рішень;
4) визначення переліку функціональних вимог до програмного забезпечення, що розробляється.

1.1. Огляд проблеми, яку вирішуватиме ПЗ

В спільноті людей, що займаються бігом, дуже розповсюджене явище, коли бігун прослуховує музику під час бігу. Улюблена музика не тільки допомагає відволікти мозок від фізичної втоми, а також може спричинити виділення адреналіну, якщо темп музики відповідатиме темпу бігу.

Як правило користувач перед бігом обирає свій плейлист з улюбленою музикою, одягає навушники, і розпочинає біг. Але при цьому виникає проблема: музичний сервіс може ввімкнути невідповідний трек (наприклад, занадто повільний під час спроби прискорення). У таких випадках користувач змушений витрачати час і докладати додаткових зусиль на ручне перемикання музики, що може знизити ефективність тренування. Для уникнення цього, користувачі можуть спробувати створити заздалегідь сплановану послідовність треків, але вона також не є оптимальним

рішенням, оскільки не здатна гнучко реагувати на зміни темпу чи стану бігуна.

Таким чином, метою даного дипломного проєкту є забезпечення гнучкого і прозорого підбору музики обраного плейлиста з музичного стримінгового сервісу користувача у відповідності до поточних фітнес-характеристик бігу шляхом розроблення відповідного ПЗ.

Для досягнення поставленої мети визначено наступні основні завдання:

1) аналіз ринку мобільних застосунків для підбору музики під час бігу;
2) виявлення та порівняння конкурентних рішень, а також визначення їхніх переваг і недоліків;
3) формування функціональних вимог до ПЗ, яке розроблятиметься;
4) обґрунтування вибору інструментів і технологій розроблення ПЗ;
5) реалізація мобільного ПЗ для автоматизованого підбору музики під час бігу;
6) тестування розробленого ПЗ та оцінювання його ефективності в реальних умовах.

1.2. Аналіз існуючих рішень

Аналіз наявних рішень дає змогу виявити переваги та недоліки конкурентних програмних продуктів, а також сформулювати обґрунтовану стратегію розвитку проєкту.

Передусім, для початку аналізу нам необхідно провести пошук мобільних застосунків, які б задовольняли основні потреби наших потенційних користувачів. Зазвичай інтеграція з підбором музики є вбудована в спортивні застосунки, які вимірюють характеристики бігу, так як користувачу зручніше заміряти поточні фітнес-показники і підбирати у відповідності до них музику з одного застосунку. Серед найпопулярніших таких застосунків можна виділити Strava: Run, Bike, Hike [1]. Утім, існують

також і винятки з цього правила, як, на приклад, Running Beats BPM for Spotify [2], або ZRX: Zombies Run + Marvel Move [3]. Такі застосунки можуть бути використані як і в додаток до основного бігового ПЗ, так і в заміну нього.

Для проведення аналізу і порівняння визначених програмних забезпечень сформульовано наступні критерії:

1) підбір музики з імпортованого особистого плейлиста;
2) залежність музичного супроводу від поточних фітнес-характеристик користувача;
3) інтеграція з особистим профілем музичного стримінгового сервісу;
4) можливість змінювати перелік треків під час бігу без втручання в профіль стримінгового сервісу;
5) гнучкий вибір фітнес-показників, що визначають підбір музичного супроводу;
6) наявність панелі зі зручним відображенням поточних характеристик користувача.

1.2.1. Strava: Run, Bike, Hike

Strava: Run, Bike, Hike є одним із найпоширеніших мобільних застосунків для відстеження спортивної активності, зокрема велосипедних поїздок, плавання, ходьби та бігу. Застосунок дає змогу фіксувати показники тренувань і зберігати їх у внутрішній соціальній мережі [1].

Для початку роботи користувач повинен зареєструватися, надати доступ до геолокації й (за наявності) спортивних сенсорів. Після цього можна розпочати запис активності, попередньо обравши вид спорту, маршрут, сенсори, параметри спільного доступу до геоданих та інші налаштування в екрані підготовки (рис. 1.1).

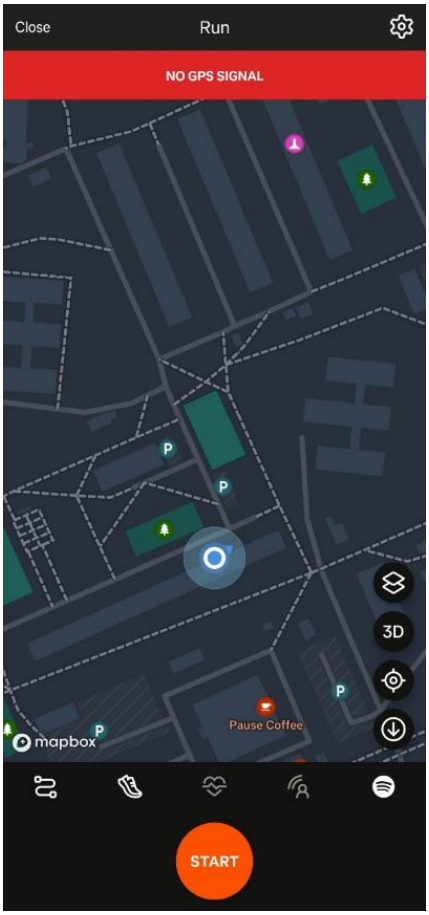


Рис. 1.1. Вигляд екрана перед початком бігу в ПЗ Strava

Інтеграція зі Spotify реалізована у вигляді кнопки, після натискання на яку відкривається вікно музичного сервісу (рис. 1.2). В цьому вікні користувач заходить в свій акаунт, обирає потрібний плейлист з запропонованого списку, і пісня одразу починає грати з обраного плейлиста.

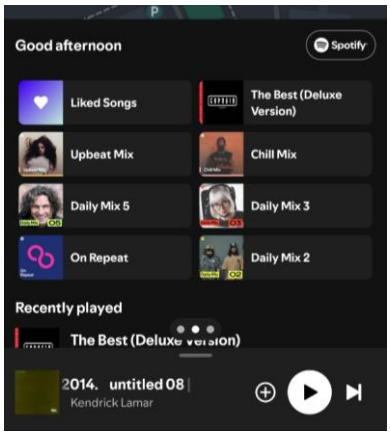


Рис. 1.2. Фрагмент вікна інтеграції з особистим профілем музичного стримінгового сервісу Spotify

З недоліків такої інтеграції зазначено наступне:

1) повторна авторизація у Spotify під час кожного запуску Strava;
2) відсутність власного алгоритму підбору музики, оскільки все відбувається на стороні Spotify;
3) необхідність взаємодії з інтерфейсом Strava або самим застосунком Spotify для керування музикою під час бігу;
4) музика починає програвати одразу під час обрання плейлиста перед початком бігу, а також продовжує відтворюватися й після завершення бігу, доки її не зупинять у Spotify.

Загалом інтеграція ПЗ Strava зі музичним сервісом Spotify поверхнева, але можна зазначити і свої плюси даного ПЗ. Такими плюсами є:

1) швидкий доступ до заздалегідь створених бігових плейлистів;
2) зручний і простий у використанні інтерфейс;
3) велика спільнота користувачів, які займаються аналогічними видами спорту;
4) глибоке та якісне відстеження поточних фітнес-показників (загальна дистанція, середній темп руху, графік часу витраченого на кожний пройдений кілометр);
5) підтримка широкого спектра спортивної активності (біг, плавання, їзда на велосипеді, спортивна ходьба, і інші види фізичної діяльності).

Отже, після проведення аналізу, не дивлячись на обмежені можливості інтеграції, Strava залишається популярним спортивним застосунком і соціальною платформою для спортсменів різного рівня підготовки. Хоч ПЗ не задовольняє більшу частину поставлених вимог, або задовольняє їх частково з величною кількістю недоліків, його переваги – наявність готових тематичних плейлистів, підтримка великої кількості різновидів фізичних вправ, ефективне вимірювання показників активності й чисельна спільнота – можуть бути вирішальними для частини користувачів.

У процесі розроблення власного ПЗ слід врахувати зазначені фактори для підвищення привабливості майбутнього рішення.

1.2.2. Running Beats BPM for Spotify

Running Beats BPM for Spotify є додатковим мобільним інструментом, що дає змогу формувати індивідуальні плейлисти для бігу. Для початку роботи користувач має увійти у свій обліковий запис Spotify, аби застосунок міг імпортувати наявні плейлисти. Далі слід вказати середній темп бігу, після чого стають доступними основні функції програми [2]. У межах цього застосунку користувач може обрати улюблені плейлисти (рис. 1.3), із яких система відфільтровує композиції відповідно до заданого темпу та формує окремий новий плейлист.

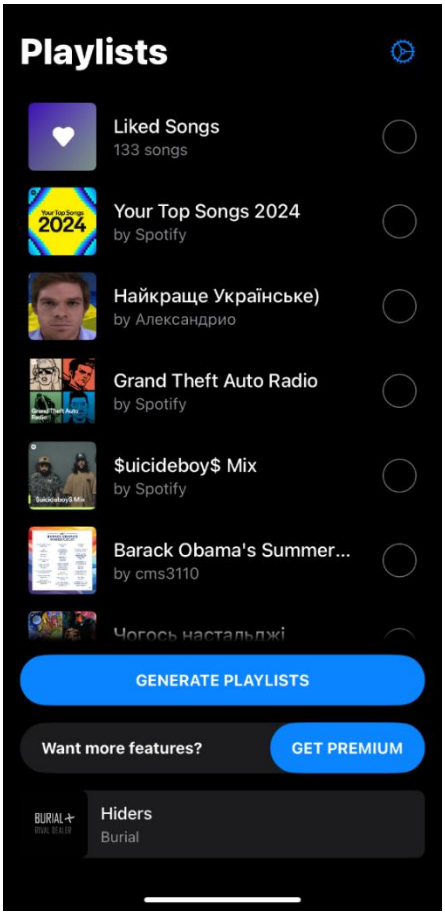


Рис. 1.3. Вигляд основного екрана перед початком бігу в мобільному застосунку Running Beats BPM for Spotify

Створений плейлист зберігається на стороні Spotify, що дає змогу повторно ним скористатися. Для повторної генерації потрібно використати меню бігу (рис. 1.4). Кількість згенерованих плейлистів є обмеженою у безплатній версії. Платна версія знімає це обмеження та додає рекомендації музичних треків і більш «розумний» алгоритм формування плейлистів.

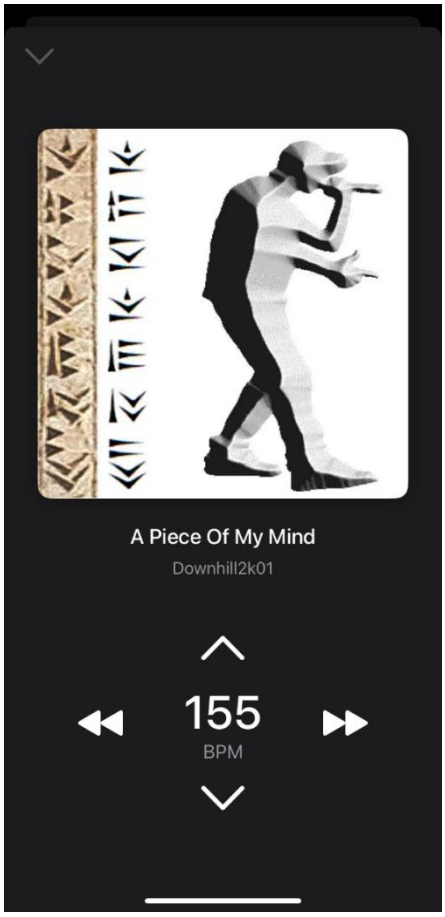


Рис. 1.4. Вигляд екрана під час бігу в мобільному застосунку Running Beats BPM for Spotify

Хоч це програмне забезпечення є корисним інструментом, яке користувач може використовувати перед початком бігу, але у нього виявлено низку недоліків, а саме:

1) генерація плейлиста базується виключно на середньому темпі бігу та не передбачає гнучких налаштувань;
2) для нової генерації слід знову взаємодіяти з інтерфейсом програми;

3) кожна нова генерація формує окремий плейлист;
4) остаточний підбір музики відбувається безпосередньо у стримінговому сервісі, а не в самому застосунку;
5) створений плейлист зберігається в обліковому записі користувача (Spotify). Після завершення сесії його необхідно видаляти вручну, якщо користувачу він буде не потрібен;
6) застосунок не надає фітнес-характеристик користувача та є додатковим модулем до основного бігового ПЗ;
7) доступне лише на операційній системі iOS;
8) безплатна версія обмежує кількість генерованих плейлистів.

Як ми бачимо, інтеграція зі музичним стримінговим сервісом Spotify глибша ніж це було в Strava, але вона досі повністю не покриває потреби нашого потенційного користувача. Все ж таки у цього застосунку можна визначити деякі плюси, такі як:

1) відбір треків відповідно до заданого темпу можливий з декількох плейлистів одночасно;
2) створені плейлисти залишаються доступними після завершення сесії, що дає змогу повертатися до них у майбутньому;
3) у платній версії наявна функція рекомендацій нових треків для автоматичної побудови плейлистів.

Таким чином, Running Beats BPM for Spotify не повністю відповідає визначеним вимогам, проте реалізує одну з ключових функцій – формування списку відтворення на основі налаштувань користувача щодо темпу бігу. Серед вагомих переваг варто відзначити рекомендації нових композицій у платній версії та можливість повторного доступу до згенерованого плейлиста після завершення тренування.

1.2.3. ZRX: Zombies Run + Marvel Move

ZRX: Zombies Run + Marvel Move – це мобільний застосунок, розроблений із метою гейміфікації процесу бігу за допомогою

інтерактивних історій. Користувач має змогу обрати одну з доступних історій, яка відтворюється під час пробіжки й може змінюватися залежно від поточних фітнес-характеристик [3].

Для початку роботи не потрібна попередня реєстрація. Достатньо надати застосунку необхідні дозволи, обрати історію, завантажити серію та розпочати біг. Під час бігу у застосунку відображаються поточні фітнес-характеристики (рис. 1.5), а результат буде збережений в окремому вікні.

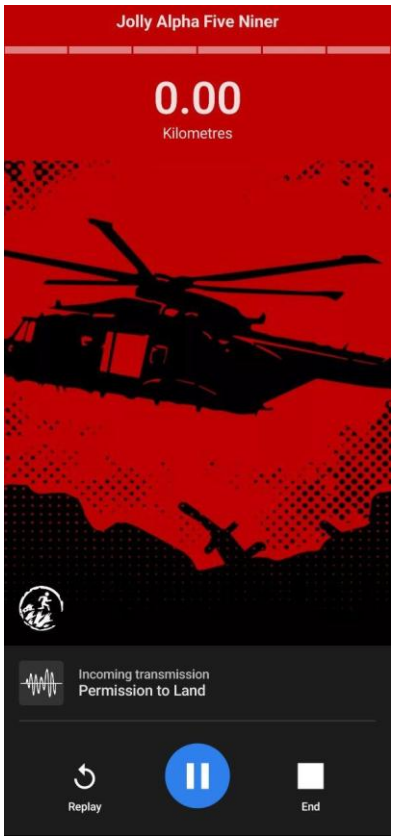


Рис. 1.5. Вигляд основного екрана під час бігу в мобільному застосунку ZRX: Zombies Run + Marvel Move

Перед початком бігу у вікні налаштувань (рис. 1.6) користувач може обрати тривалість сесії, активувати або деактивувати режим переслідування, налаштувати музику та відстеження місцеперебування. Окрім використання GPS, застосунок підтримує підрахунок кроків та інші альтернативні методи відстеження, що робить його придатним для бігової доріжки чи іншої фізичної активності.

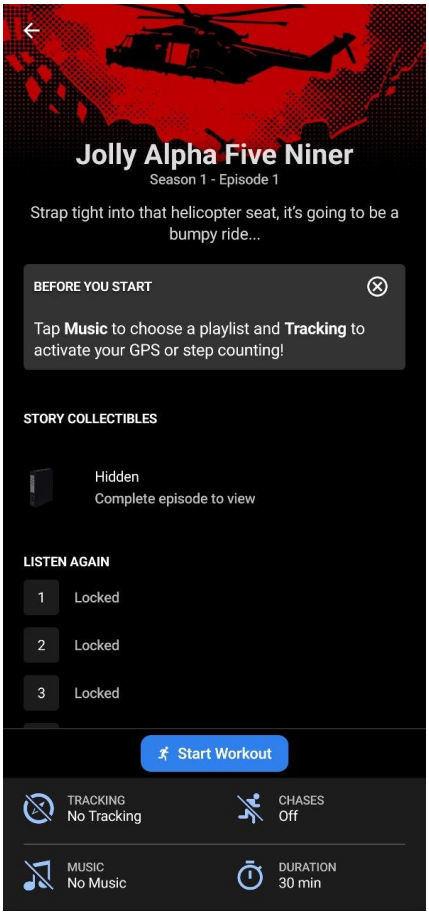


Рис. 1.6. Вигляд основного екрана перед початком бігу в мобільному застосунку ZRX: Zombies Run + Marvel Move

У програмі передбачена платна підписка, що вилучає рекламу, надає необмежений доступ до нових історій та відкриває додаткові можливості налаштування режиму переслідування. Механіка переслідування призначена для мотивації підвищувати темп бігу: між діалогами персонажів відтворюються звуки переслідування зомбі, наближення яких супроводжується відповідними аудіо-сигналами. Щоб втекти від них, користувачу потрібно буде збільшити темп бігу.

Інтеграція зі стримінговими сервісами музики реалізована на базовому рівні: користувач самостійно обирає плейлист, а застосунок приглушує його під час відтворення власного інтерактивного контенту чи звукових ефектів переслідування. Додатково наявна можливість прослуховувати локально збережені музичні треки. В такому випадку

застосунок може чекати до того, як музика закінчиться, щоб включити своє власне аудіо без переривань користувацької аудіо бібліотеки.

Серед переваг цього застосунку зазначено наступне:

1) інтерактивний контент адаптується до поточних фітнес-характеристик;
2) сюжетна історія розгортається впродовж кількох бігових сесій;
3) програма придатна не лише для бігу на відкритому повітрі, а й для інших видів спортивної активності;
4) не вимагає реєстрації для базового використання застосунку і його безкоштовних історій;
5) режим переслідування сприяє підвищенню темпу завдяки гейміфікації процесу;
6) застосунок може автоматично приглушувати музику, незалежно від джерела її відтворення;
7) підтримує режим роботи без мережі Інтернет.

Серед недоліків цього застосунку виявлено наступне:

1) музика повністю контролюється зовнішнім сервісом; застосунок може лише тимчасово зменшувати гучність;
2) у безплатній версії обмежено можливості налаштування режиму переслідування;
3) мала кількість доступних інтерактивних історій без придбання платної підписки;
4) налаштування сесії дає змогу обирати лише між заданою дистанцією або часом.

Таким чином, ZRX не вирішує повністю проблему автоматизованого підбору музики під біг, проте робить увагу на гейміфікації процесу та мотивуванні користувача збільшувати або підтримувати темп. Водночас поверхнева інтеграція зі стримінговими сервісами та акцент на власному аудіо-контенті не задовольняють основну потребу у використанні користувацької музики як динамічного супроводу бігу. Утім, завдяки

тривалому періоду існування й значній популярності у своєму сегменті цей застосунок може слугувати корисним прикладом для розробки аналогічних рішень, зокрема у мотивації користувачів.

1.3. Результати проведеного аналізу

За результатами аналізу наявного програмного забезпечення, яке бігуни використовують для супроводу музикою під час тренувань, можна зробити такі висновки:

1. Strava: Run, Bike, Hike (як представник великих спортивних сервісів і соцмереж) забезпечує базову інтеграцію зі Spotify та надає швидкий доступ до заздалегідь створених бігових плейлистів. Водночас гнучкість підбору музики обмежується функціональними можливостями самого Spotify.
2. Running Beats BPM for Spotify (як додатковий інструмент для бігових тренувань) надає змогу створювати плейлисти на основі імпортованих користувацьких треків і заданого темпу. Проте розрахунок темпу виконується вручну, а не базується на актуальних фітнес-характеристиках бігуна.
3. ZRX: Zombies Run + Marvel Move (як сервіс, орієнтований на гейміфікацію бігу) адаптує власний інтерактивний аудіоконтент відповідно до поточних показників користувача, проте взаємодія із зовнішнім музичним сервісом обмежується лише приглушенням у потрібні моменти.

На етапі планування та визначення функціональних вимог до системи сформовано ключові критерії для автоматизованого підбору музики згідно з актуальними фітнес-показниками користувача. Нижче наведено порівняльну таблицю (табл. 1.1), у якій зазначені програмні рішення за визначеними критеріями.

Таблиця 1.1

Порівняння існуючих програмних рішень

Функціональна вимога \ Сервіс	Strava	Running Beats	ZRX
Підбір музики з імпортованого особистого плейлиста	–	+	–
Залежність музичного супроводу від поточних фітнес-характеристик користувача	–	–	+
Інтеграція з особистим профілем музичного стримінгового сервісу	+	+	+/–
Можливість змінювати перелік треків під час бігу без втручання в профіль стримінгового сервісу	–	–	–
Гнучкий вибір фітнес-показників, що визначають підбір музичного супроводу	–	–	+
Наявність панелі зі зручним відображенням поточних характеристик користувача	+	–	+

Порівняльний аналіз показав, що жоден із розглянутих застосунків не задовольняє усі критичні потреби бігунів щодо динамічного музичного супроводу. На підставі отриманих результатів сформовано наступний перелік ключових функціональних можливостей, які має забезпечувати розроблюване ПЗ:

1) відображення актуальних фітнес-показників у вигляді інтерактивної панелі під час пробіжки;
2) підключення до особистого облікового запису користувача в музичному сервісі Spotify з доступом до його бібліотеки;
3) можливість визначити, за якими фітнес-показниками здійснюватиметься підбір треків у реальному часі;

4) можливість виключати окремі треки із сесії спорту у без зміни вихідного плейлиста на стороні Spotify;
5) додавання, пропуск та реорганізація треків у поточному списку відтворення на стороні Spotify під час спортивної сесії у відповідності до поточних фітнес-характеристик.

1.4. Висновки до розділу

Під час написання першого розділу дипломного проєкту виконано наступні задачі:

1) проаналізовано ринок спортивних застосунків із музичною підтримкою;
2) виявлено ключові проблеми користувачів таких систем;
3) складено перелік найпопулярніших рішень для бігунів;
4) визначено критерії їх порівняння у форматі функціональних вимог;
5) проведено порівняльний аналіз рішень із визначенням їх переваг і недоліків;
6) порівняно результати аналізів у вигляді таблиці;
7) сформульовано перелік функціональних вимог до розроблюваного ПЗ у рамках дипломного проєкту.

2. ОБГРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РОЗРОБЛЕННЯ

У другому розділі пояснювальної записки сформульовано вимоги до засобів розроблення, виконано їх аналіз з наведенням списку особливостей кожного з них, і подано порівняльну таблицю за визначеними критеріями.

Розроблюваний сервіс персоналізованого музичного супроводу реалізовано за клієнт-серверною схемою, що підвищує безпеку завдяки обов'язковій автентифікації та забезпечує відновлення профілю з будь-якого пристрою через централізоване зберігання даних.

Серверна частина здійснює керування базою даних, обробляє та повертає HTTP-запити, тоді як клієнтська надає інтерфейс користувача, синхронізує дані з музичним сервісом, запускає тренувальні сесії, формує музичні рекомендації й обмінюється інформацією з сервером через REST-API.

2.1. Вибір мови програмування для реалізації клієнтської частини

Серед поширених засобів розроблення клієнтської частини мобільних застосунків виокремлюють наступні інструменти: мови програмування Kotlin та Swift, а також фреймворк React Native. Саме ці три інструменти обрано для подальшого системного аналізу, за результатами якого буде визначено оптимальний варіант для створення клієнтської частини ПЗ.

До інструмента, що реалізовуватиме мобільну клієнтську частину, висуваються наступні вимоги:

1) підтримка компонентної архітектури;

2) лаконічний і читабельний код;

3) наявність розвинутої екосистеми бібліотек;

4) висока продуктивність;

5) кросплатформна підтримка.

2.1.1. Kotlin

Kotlin – сучасна типізована мова для Android-розробки [4]. Але також існує і технологія Kotlin Multiplatform Mobile, що дає змогу спільно використовувати бізнес-логіку на Android та iOS [5]. Завдяки компіляції у JVM-байткод або системний машинний код, Kotlin забезпечує продуктивність опрацьовування бізнес логіки і виведення результату на екран, що є наближена до системної мови, так як взаємодія з елементами системи відбувається на пряму, без додаткових шарів трансформації запитів.

У Kotlin-екосистемі UI будується компонентним підходом. Для Android застосовується Jetpack Compose, де екран розглядається як дерево незалежних композитів, кожен із яких можна повторно використовувати чи вкладати в інші [6]. Така декомпозиція спрощує масштабування та ізолює побічні ефекти, а декларативна модель забезпечує швидке оновлення стану інтерфейсу під час розроблення.

Лаконічність мови досягається шляхом виведення типів, системи null-безпеки, використанням розширюваних функцій і ко-рутин – це скорочує шаблонний код і підвищує читабельність. Повна сумісність із Java відкриває доступ до багаторічної екосистеми бібліотек, а власні пакети – Kotlinx-coroutines, Serialization, Ktor, Compose Multiplatform – покривають роботу з мережею, паралелізм, серіалізацію та роботу з UI без втрати продуктивності.

Таким чином, Kotlin відповідає основним вимогам до інструмента розробки мобільної клієнтської частини: забезпечує кросплатформність бізнес-логіки (через KMM), підтримує компонентну архітектуру, відзначається лаконічним та безпечним синтаксисом, спирається на широку екосистему бібліотек і демонструє високу продуктивність завдяки системній компіляції.

2.1.2. Swift

Swift – статично типізована мова для програмування під системи iOS, iPadOS, watchOS, tvOS та macOS і компільована безпосередньо у машинний код через LLVM, що забезпечує швидкодію опрацьовування бізнес логіки близьку до швидкодії системного коду без проміжних шарів виконання [7]. Завдяки optionals, type inference і розширенням мовних конструкцій Swift зменшує обсяг шаблонного коду, водночас підтримуючи суворий контроль нульових значень, що суттєво підвищує читабельність і безпечність програмного коду.

Декларативний фреймворк SwiftUI реалізує компонентний підхід, оскільки інтерфейс описується як ієрархія незалежних view-компонентів, які можна вільно використовувати в композиції користувацького інтерфейсу та повторно використовувати в інших місцях [8]. Модель оновлення стану базується на механізмі @State/@Binding, що дає поточний перегляд інтерфейсу під час розробки, а пізніше генерує оптимізоване дерево відображення для GPU, мінімізуючи витрати ресурсів.

Екосистема Swift містить багаторічні фреймворки Cocoa/UIKit, власний Swift Package Manager, а також тисячі пакетів на GitHub, що охоплюють роботу з мережею, базами даних, графікою та машинним навчанням; крім того, завдяки відкритості компілятора мовою можна користуватися на Linux-серверах або, за підтримки спільноти, збирати бібліотеки під Android і Windows, що частково задовольняє вимогу кросплатформності.

Таким чином, Swift повністю відповідає вимогам щодо лаконічності, компонентної архітектури та високої продуктивності, має розвинену бібліотечну базу та офіційну підтримку інструментарію (Xcode, Swift Tools). Обмеження полягає у тому, що повноцінний кросплатформний досвід наразі зосереджений переважно в екосистемі Apple, хоча відкритість мови дає можливість розширювати сферу її застосування поза межами нативних платформ iOS/macOS.

2.1.3. React Native

React Native – фреймворк для кросплатформної розробки мобільних застосунків на JavaScript або TypeScript [9]. Під час компіляції бізнес-логіка оброблюється у віртуальній машині Hermes, тоді як UI-шар збирається з системних компонентів Android та iOS; таким чином досягається продуктивність, близька до системної, без додаткових веб-обгорток.

Архітектура базується на компонентній моделі React: екрани користувацького інтерфейсу описуються декларативним деревом компонентів, що можна вільно вставляти в композиції користувацького інтерфейсу, а також повторно використовувати й ізолювати [10]; оновлення стану керується спеціальними hook-функціями та технологією Fast Refresh, яка забезпечує миттєву повторну візуалізацію користувацького під час розробки.

Лаконічність коду для фреймворку React Native досягається за допомогою синтаксису JSX, системи хуків React і типовим структуруванням проєктів, що зменшує шаблонний код. Через менеджер пакунків npm доступна величезна екосистема користувацьких бібліотек: від навігації, керуванням станом, анімації, до керування Bluetooth, AR та інших API. Додатково існує окрема екосистема інструментів Expo, що пропонує керований пакет CLI-інструментів і сервісів збирання застосунку [11]. При необхідності легко можливо підключити власні системні модулі мовами Kotlin, Swift або Objective-C/Java, що задовольняє вимогу інтеграції з платформо-специфічними функціями.

В підсумку, React Native повністю відповідає критеріям кросплатформності, компонентної архітектури, лаконічності коду, наявності великої бібліотечної бази й достатнього рівня продуктивності, забезпечуючи при цьому гнучку інтеграцію з системною інфраструктурою мобільних платформ.

2.1.4. Результати проведеного аналізу

Після проведеного аналізу побудуємо порівняльну таблицю технологій для реалізації клієнтської частини за критеріями, що були визначені перед початком проведення аналізу (табл. 2.1).

Таблиця 2.1

Порівняння технологій для реалізації клієнтської частини застосунку

Технологія / Критерій порівняння	Kotlin	Swift	React Native
Кросплатформна підтримка	+/–	–	+
Підтримка компонентної архітектури	+	+	+
Лаконічний і читабельний код	+	+	+
Наявність розвинутої екосистеми бібліотек	+/–	+/–	+
Висока продуктивність	+	+	+/–

Порівняльний аналіз показав, що React Native найкраще впорався зі сформульованими критеріями. Фреймворк забезпечує повноцінну кросплатформну підтримку Android та iOS, реалізує гнучку компонентну архітектуру й має розвинену багаторічну екосистему пакетів, що скорочує час розроблення та подальшої підтримки. Код на TypeScript/JavaScript залишається лаконічним і легко читабельним; єдина часткова невідповідність стосується продуктивності на застарілих пристроях, однак практичні втрати можуть бути компенсовані завдяки віртуалізації великих листів, мемоїзації компонентів та багатьох інших методів підвищення продуктивності.

Що стосується Kotlin (KMM), то ця мова задовольняє вимоги до продуктивності, компонентного підходу та лаконічності, проте лише

частково покриває кросплатформність – UI для iOS необхідно розробляти окремо – і має обмеженішу бібліотечну базу, що зумовлено відносною новизною технології.

Якщо говорити про Swift, то ця мова демонструє високі показники продуктивності та зручну декларативну архітектуру SwiftUI, але вона прив'язана лише до екосистеми Apple і володіє порівняно невеликою спільнотою сторонніх пакетів, що збільшує витрати часу при створенні логіки, спільної для кількох платформ.

Отже, урахувавши ці чинники, для розроблення клієнтської частини обрано фреймворк React Native.

2.2. Вибір сервісу для реалізації серверної частини

Сервіс, призначений для реалізації серверної частини програмного забезпечення, має відповідати таким функціональним вимогам:

1) вбудовані засоби автентифікації;
2) повноцінні інструменти роботи з базою даних;
3) розвинена екосистема модулів;
4) можливість простого масштабування бази даних і користувачів;
5) спрощене підключення клієнтської частини;
6) докладна й актуальна документація з низьким порогом вступу;
7) аналітична панель із поточними показниками стану сервера.

2.2.1. Firebase

Firebase – хмарна платформа Google класу BaaS, що об'єднує систему автентифікації користувачів, бази даних, файлове сховище, середовище для написання serverless-функції та інформаційні панелі з аналітикою в єдиному SDK для Android, iOS, Web і Unity [12]. Ядром ідентифікації виступає Firebase Authentication, яка підтримує авторизацію за поштою, телефоном, або використовуючи один з OAuth-провайдерів, а також підтримує MFA,

забезпечуючи повний цикл реєстрації та входу без додаткового серверного коду [13].

Для зберігання даних платформа надає дві serverless NoSQL-опції – Realtime Database та Cloud Firestore. Обидві синхронізують дані у реальному часі, працюють offline і автоматично масштабуються до мільйонів підключень, залишаючись прозорими для клієнтського коду [14]. Обчислювальні задачі виконуються через Cloud Functions for Firebase, що запускаються подіями БД або HTTP-запитами й розгортаються на інфраструктурі Google без адміністрування серверів.

Екосистема містить сотні офіційних і сторонніх пакетів, а вся функціональність керується з консолі Firebase: там доступні управління проєктами, налаштування сервісів, а також інтегрована Google Analytics – панель із звітами подій та метриками продуктивності, що задовольняє вимогу інформаційної аналітики. Документація охоплює всі SDK і сценарії масштабування, включаючи інструкції з розгортання та найкращих практик використання сервісу.

Таким чином, Firebase повністю відповідає висунутим критеріям: має вбудовані засоби автентифікації, потужні інструменти роботи з даними, розгалужену екосистему модулів, автоматичне масштабування й спрощене підключення клієнтів через єдиний кросплатформний SDK, доповнене вичерпною документацією та аналітичною консоллю.

2.2.2. Supabase

Supabase – відкрита платформа BaaS, побудована на PostgreSQL, що об’єднує аутентифікацію, реляційну базу даних, файлове сховище та edge-функції в єдиному SDK для Web, Android, iOS та інших клієнтів [15]. Завдяки реплікації СУБД Postgres події з’являються у підписаних клієнтів без окремого брокера, а вся інфраструктура керується через браузерну консоль Supabase Studio чи локальний консольний інтерфейс. Підтримуються авторизації через електронну пошту, OAuth-логіни та MFA;

дані користувачів зберігаються у виділеній схемі БД, що спрощує контроль доступу і відповідає вимогам автентифікації та безпеки [16].

Реляційна модель забезпечує повноцінну підтримку SQL-запитів, транзакцій та розширення PostGIS. Масштабування відбувається горизонтально для edge-функцій і вертикально, або кластером, для СУБД Postgres, тому у проєктах із різко мінливим трафіком може знадобитися додаткове налаштування системи. Екосистема модулів швидко зростає, але поки поступається конкурентам за кількістю інтегрованих сервісів.

Документація покриває усі SDK і сценарії розгортання, однак інструменти моніторингу ще оформлюються; аналітична панель надає базові метрики запитів, але глибока інтеграція з Google Analytics або BigQuery відсутня. Цінова модель фіксована при високих обсягах записів, однак безплатний рівень має нижчі ліміти зберігання й продуктивності.

У підсумку Supabase відповідає ключовим вимогам (автентифікація, робота з БД, консоль керування, клієнтські SDK) й приваблює відсутністю прямої залежності від постачальника сервісу та можливістю розміщення на власних серверах або у приватній хмарі. Проте менша кількість сторонніх користувацьких бібліотек, обмежена кількість вбудованих сервісів та менш автоматизоване масштабування ставлять її в менш вигідне становище для проєктів, що потребують миттєвого розгортання з максимальною кількістю готових інструментів.

2.2.3. AWS Amplify

AWS Amplify – це інтегрований набір керованих сервісів AWS, що через CLI, SDK та веб-конструктор Amplify Studio надає повну Back-end функціональність: автентифікацію на базі Amazon Cognito, GraphQL- і REST-API через AppSync та API Gateway, offline-синхронізацію даних DataStore з DynamoDB, наявність файлового сховища S3, serverless функції Lambda, а також хостинг і CI/CD [17].

Усі ресурси описуються інфраструктурним кодом CloudFormation та адмініструються з єдиної консолі, де доступні базові метрики трафіку й помилок завдяки Amazon Pinpoint і CloudWatch. Платформа повністю покриває вимоги до автентифікації, роботи з базою даних, масштабування й клієнтських SDK, водночас спираючись на широку екосистему AWS-сервісів.

Втім, у порівнянні зі своїми конкурентами, Amplify характеризується крутішою кривою навчання: налаштування Cognito, IAM-ролей і CloudFormation вимагає глибшого розуміння хмарних концепцій, а початкова конфігурація є складнішою через фрагментований набір сервісів.

Ціноутворення у даного проєкту теж менш прозоре: безплатний рівень обмежений 12 місяцями, а подальші витрати розподілені між AppSync-запитами, Cognito-користувачами, трафіком S3 та іншими компонентами, що ускладнює прогноз бюджету. Крім того, платформа не підтримує розгортання на власних серверах й створює жорстку залежність від своїх сервісів на AWS-інфраструктуру.

У підсумку аналізу можна зазначити, хоч Amplify задовольняє усі функціональні вимоги, рівень складності старту й адміністративні витрати ускладнюють його використання для проєктів, що потребують швидкого впровадження та єдиного консольного середовища.

2.2.4. Результати проведеного аналізу

Після проведеного аналізу побудуємо порівняльну таблицю характеристик сервісів для реалізації серверної частини за критеріями, що були визначені перед початком проведення аналізу (табл. 2.2).

З порівняльної таблиці можна побачити, що Firebase задовольняє всі поставлені критерії порівняння до засобу розроблення серверної частини. Платформа має готові модулі автентифікації, дві керовані бази даних, широку екосистему вбудованих і користувацьких бібліотек і автоматичне масштабування без додаткових налаштувань. Підключення клієнтських

SDK займає мінімальну кількість часу, документація докладна й орієнтована на швидкий старт, а вбудована консоль надає повний зріз аналітики про стан сервера та використання ресурсів.

Таблиця 2.2

Порівняння технологій для реалізації серверної частини застосунку

Технологія / Критерій порівняння	Firebase	Supabase	AWS Amplify
Вбудовані засоби автентифікації	+	+	+
Повноцінні інструменти роботи з базою даних	+	+	+/–
Розвинена екосистема модулів	+	+/–	+
Можливість простого масштабування бази даних і користувачів	+	+/–	+
Спрощене підключення клієнтської частини	+	+	+/–
Докладна й актуальна документація з низьким порогом вступу	+	+	+/–
Аналітична панель із поточними показниками стану сервера	+	+/–	+/–

Що стосується Supabase, цей сервіс покриває такі критерії, як автентифікація, робота з базою даних і спрощене підключення клієнтів завдяки тому, що воно пропонує Postgres+SQL, готовому Auth і простій JavaScript SDK. Проте, разом з тим, сервіс має обмеженішу екосистему інтегрованих бібліотек, його горизонтальне масштабування потребує ручного конфігурування, а Supabase Studio надає лише базові метрики.

Якщо говорити про AWS Amplify, то базові функції також присутні, але є обмеження зі схемою NoSQL DynamoDB+AppSync, оскільки вона

підходить не для всіх сценаріїв; підключення SDK може забирати додатковий час, а документація має крутішу криву навчання. Також можна додати, що аналітичні панелі CloudWatch і Pinpoint забезпечують лише базовий рівень моніторингу порівняно зі своїми конкурентами.

Таким чином, саме Firebase найкраще відповідає сформульованим вимогам, що робить його доцільним вибором для серверної частини застосунку.

2.3. Висновки до розділу

Під час написання другого розділу дипломного проєкту було виконано наступні завдання:

1) було сформовано список функціональних вимог до засобів реалізації програмного забезпечення: мови програмування для створення клієнтської частини, а також технології для створення серверної частини;
2) визначено найпопулярніші засоби реалізації цих вимог;
3) проведено порівняльний аналіз обраних інструментів із висвітленням їхніх переваг і недоліків;
4) складено порівняльну таблицю за заданими критеріями;
5) сформульовано висновки та обрано оптимальні інструменти для розроблення клієнтської й серверної частин.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ ЗАСТОСУНКУ

У третьому розділі пояснювальної записки представлено результати виконання таких завдань:

1) проаналізувати проблематику та визначити цілі мобільного застосунку;
2) сформулювати вимоги до програмного забезпечення;
3) побудувати архітектуру системи;
4) розробити структури даних;
5) розробити алгоритми функціонування мобільного застосунку.

3.1. Опис вимог до розроблюваного мобільного застосунку

Функціональна вимога – це формалізований опис необхідної поведінки програмної системи, зрозумілий всім учасникам проєкту та узгоджений між замовником і виконавцем [18]. Щоб така вимога була придатною до реалізації, вона має відповідати критеріям SMART:

- S (Specific) – вимога чітко визначена і стисла;
- M (Measurable) – вимога містить чіткі критерії, за якими можна визначити ступінь її досягнення;
- A (Achievable) – вимога реалістична і досяжна з поточними можливостями;
- R (Relevant) – вимога відповідає поточним цілям проєкту;
- T (Time-bound) – вимога має конкретні часові рамки, за якими вона повинна бути досягнена.

Виявлення й опрацювання вимог є критично важливим початковим етапом життєвого циклу ПЗ: зміни, внесені на пізніх фазах розроблення, потребують значно більше ресурсів, ніж корекція на стадії збору вимог. Тому всі функціональні вимоги слід ретельно перевіряти, та документувати, щоб мінімізувати ризики невідповідності очікуванням зацікавлених сторін.

За рівнем деталізації вимоги умовно поділяють на:

1) бізнес-вимоги, що фіксують користь від системи для організації;
2) вимоги зацікавлених сторін, які відображають потреби всіх власників у рамках Програмного Забезпечення;
3) функціональні вимоги, що описують, які саме дії має виконувати система;
4) нефункціональні вимоги, що визначають умови, у яких ці дії повинні відбуватися, а також властивості якості ПЗ;
5) перехідні вимоги, що регламентують умови міграції й подальшого масштабування [19].

За допомогою визначення вимог з різним рівнем деталізації, команді розробників стає доступним поглиблене розуміння проблематики, яку вирішує розроблюване ПЗ, а також стає можливою і побудова подальшого плану розроблення.

3.1.1. Аналіз проблематики та цілей мобільного застосунку

З метою надання реальної користі кінцевим споживачам розроблюваного програмного забезпечення під час визначення вимог до мобільного застосунку для автоматизованого підбору музики під час бігу доцільно фокусуватися на чинниках, що безпосередньо впливають на досвід спортсменів, які тренуються з музичним супроводом. На підставі аналізу особливостей предметної галузі та виявлених потреб потенційних користувачів встановлено такі ключові проблеми:

1) неможливість прив’язати алгоритм підбору музики до зовнішніх показників (зокрема темп, місцеположення, тощо);
2) відсутність механізму приховування окремих композицій без змін обраного плейлиста;
3) брак засобів керування параметрами роботи алгоритму.

Структурну декомпозицію зазначених проблем представлено на рис. 3.1.

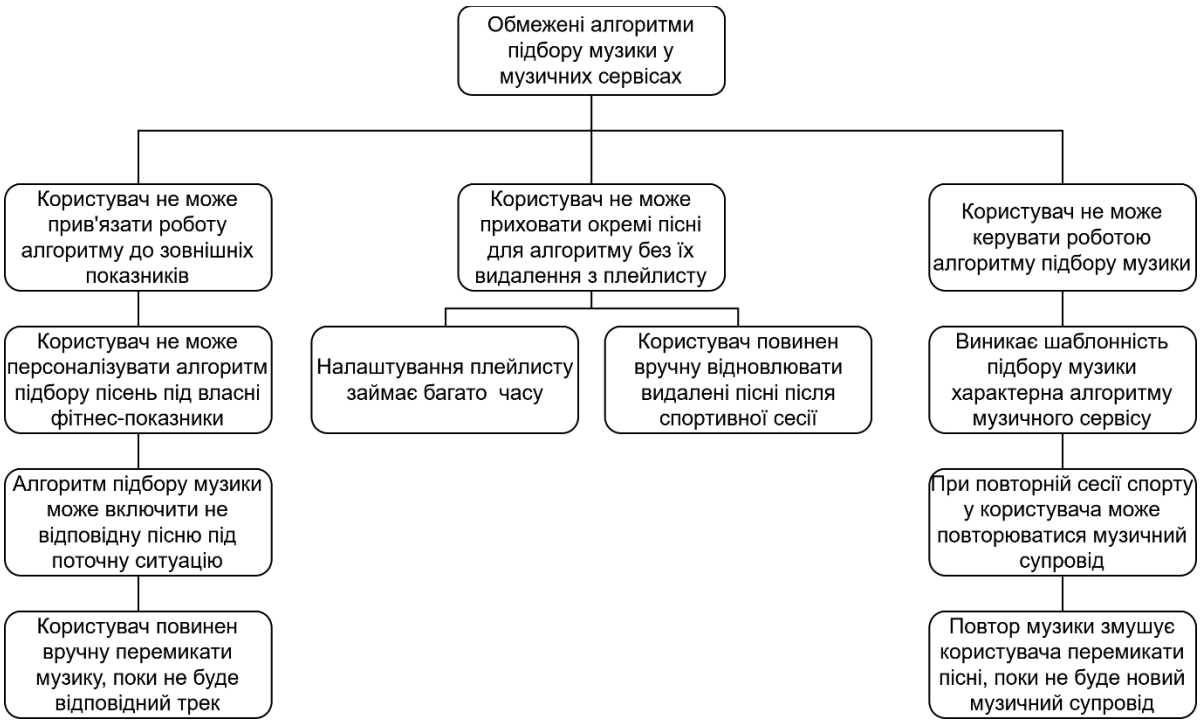


Рис. 3.1. Дерево проблем людей, що займаються спортом з музичним супроводом

Головною метою проєкту є розроблення програмного продукту, який забезпечує автоматизований добір музичних композицій з урахуванням поточних показників спортивної активності користувача. Для досягнення зазначеної мети передбачено створення п’яти функціональних модулів:

1. Модуль імпорту музичних даних – інтеграція зі Spotify API для завантаження даних про користувацькі плейлисти і користувача.
2. Модуль керування імпортованими даними – операції з імпортованими плейлистами та окремими музичними композиціями.
3. Модуль оброблення фітнес-даних – зчитування та нормалізація поточних спортивних показників користувача.
4. Модуль музичного супроводу – формування черги відтворення на основі актуальних фітнес-характеристик і заданих параметрів добору.
5. Панель налаштування роботи алгоритму – панель налаштування критеріїв роботи алгоритму підбору музики.

Враховуючи попередньо зазначене дерево проблем, усі визначені цілі були об’єднані в дерево цілей, що наведено на рис. 3.2.

Рис. 3.2. Дерево цілей розроблення мобільного застосунку

На рис. 3.3 представлено дерево результатів розроблення мобільного застосунку автоматизованого музичного супроводу для бігу; воно сформоване на основі узгодженого дерева цілей, визначених для виконання поточного проєкту.

Рис. 3.3. Дерево результатів розроблення мобільного застосунку

3.1.2. Формулювання вимог до програмного забезпечення

Вимоги до програмного забезпечення можуть фіксуватися різними методами, зокрема: користувацькі історії (User Story), сценарії використання (Use Case), критерії приймання (Acceptance Criteria), UML-діаграми або BPMN-діаграми і т.д. Формат обирається з огляду на те, наскільки доступно він передає інформацію усім групам зацікавлених сторін, а саме: замовнику для співвіднесення з бізнес-процесами, команді розроблення для реалізації функціональних можливостей, фахівцям QA для перевірки відповідності готового продукту затвердженим вимогам.

Для розроблюваного мобільного застосунку підбору музичного супроводу обрано використовувати сценарії використання (Use Case), оскільки цей формат найкраще відтворює послідовну взаємодію користувача із системою.

Кожен сценарій містить наступні структурні елементи:

1) ідентифікатор і назву;

2) акторів (користувачі чи зовнішні системи);

3) початкові умови;

4) очікуваний результат;

5) основний потік подій;

6) альтернативні (виняткові) потоки.

Вимоги до розроблюваного мобільного застосунку було сформовано у вказаному форматі сценаріїв використання (додаток 3).

Виходячи зі сформульованих сценаріїв використання мобільного застосунку автоматизованого музичного супроводу та з урахуванням узагальненого дерева проблем, підготовлено зведену таблицю функціональних вимог, обов’язкових для реалізації у межах даного дипломного проєкту. Реєстр функціональних вимог наведений в таблиці 3.1.

Таблиця 3.1

Реєстр функціональних вимог до мобільного застосунку

Код вимоги	Зміст вимоги	Атрибути	
		Пріоритет	Складність
F1	Можливість зареєструвати нового користувача за електронною поштою	Високий	Середня
F2	Можливість авторизувати зареєстрованого користувача за електронною поштою та паролем	Високий	Середня
F3	Можливість авторизувати зареєстрованого користувача за акаунтом Google	Високий	Середня
F4	Можливість під’єднання (OAuth) облікового запису Spotify та зберігання access-токена	Високий	Середня
F5	Можливість імпорту користувацьких плейлистів зі Spotify	Високий	Середня
F6	Можливість відновлення користувацьких даних (дані інтегрованого профілю Spotify, налаштування плейлистів, налаштування алгоритму) при повторному вході в систему	Високий	Висока
F7	Запуск тренувальної сесії з автоматизованим добором музики в реальному часі	Високий	Висока
F8	Можливість приховування окремих треків для алгоритму без зміни оригінального списку	Середній	Середня
F9	Можливість налаштування характеристик роботи алгоритму за обраним режимом	Високий	Середній

Продовження табл. 3.1

F10	Можливість вибору типу фітнес-характеристик (темп, час, місцеположення, дистанція), у відповідності до яких буде працювати алгоритм вибору музичного супроводу	Високий	Середній
F11	Перегляд поточних фітнес характеристик користувача під час тренування (час, дистанція, середній темп, поточний трек)	Середній	Середня

На наступному етапі сформульовано нефункціональні вимоги, тобто обмеження та характеристики, що регламентують роботу системи. Відповідно до класифікації вони згруповані за категоріями:

1) безпека – гарантує конфіденційність, цілісність і доступність даних;
2) продуктивність – визначає цільові показники часу відгуку, пропускної здатності, споживання ресурсів тощо;
3) реалізаційні аспекти – встановлює вимоги до портативності, масштабованості, сумісності, можливості супроводження та інші.

Перелік нефункціональних вимог для розроблюваного мобільного застосунку подано далі (табл. 3.2-3.4).

Таблиця 3.2

Нефункціональні вимоги до безпеки

Код вимоги	Зміст вимоги
SEC-1	Доступ до даних користувача має лише цей користувач
SEC-2	Використання OAuth 2.0 для авторизації в музичних сервісах
SEC-3	Надавання доступу до API музичних стримінгових платформ винятково через токенізовані запити з обмеженим терміном дії

Таблиця 3.3

Нефункціональні вимоги до продуктивності

Код вимоги	Зміст вимоги
PER-1	Система повинна підтримувати одночасне підключення 100 користувачів
PER-2	Система повинна виконувати повне завантаження даних користувача під час авторизації не більше, ніж 5 секунд
PER-3	Система повинна підтримувати оброблення імпортованих плейлистів з 150 треків

Таблиця 3.4

Нефункціональні вимоги до реалізації

Код вимоги	Зміст вимоги
IMP-1	Інтерфейс має підтримувати щонайменше дві мови (Українська і Англійська) з можливістю додавання нових мов
IMP-2	Мобільний застосунок повинен коректно працювати на операційних системах Android 13 і вище
IMP-3	Застосунок не повинен споживати більш ніж 5 % заряду батареї під час 30-хвилинної тренувальної сесії

3.2. Опис архітектури мобільного застосунку

Архітектура програмного забезпечення – це сукупність принципів, що визначають поділ системи на логічні складові та спосіб їх взаємодії [20]. Вибір архітектури впливає на масштабованість і загальну якість сервісу.

Під час проєктування ПЗ було враховано такі чинники:

1) функціональні та нефункціональні вимоги (підбір музики у реальному часі, зчитування фітнес-даних користувача);
2) терміни розроблення MVP і обмежені DevOps-ресурси;
3) масштабованість для підключення нових музичних сервісів.

З огляду на це TempoRun реалізовано у вигляді клієнт-серверної системи з «тонким» serverless-бекендом. Усі тривалі та ресурсномісткі операції (оброблення фітнес-показників користувача, підбір темпу музики під них, синхронізація даних користувача) виконує клієнт, тоді як Back-end, побудований на платформі Firebase (BaaS), забезпечує лише автентифікацію (Firebase Auth) та зберігання документів (Cloud Firestore). Подібний розподіл надає такі переваги:

1) автоматичне масштабування інфраструктури;

2) централізоване та безпечне зберігання даних користувачів;

3) real-time синхронізація змін без ручної реалізації WebSocket-шару;

4) мінімальні витрати на супровід та розгортання нових версій.

Клієнтська частина побудована за шаблоном Model-View-ViewModel з репозиторіями. Рівень View (екрани React Native) відповідає за відображення даних і навігацію; ViewModel містять бізнес-логіку, зокрема алгоритм добору треків відповідно до поточного темпу бігу; Repository інкапсулює роботу з Firebase SDK, Spotify API та SoundStat API, виконує перетворення даних. Жодна критична бізнес-логіка не зберігається на сервері, що відповідає принципу thin-backend. Така структура забезпечує ізольовану зміну будь-якого шару без впливу на решту системи.

Під час проєктування клієнтської частини застосунку було застосовано архітектурну модель Redux для централізованого керування глобальним станом. У межах цієї моделі формується єдиний об’єкт стану (State), ієрархічно поділений на тематичні сегменти (Slices). Односторонній потік даних через ці сегменти забезпечується виконанням дій (Actions), за допомогою яких бізнес-логіка змінює дані, що відображаються у користувацькому інтерфейсі. Дії, пов’язані з ключовими даними користувача – зокрема параметрами роботи алгоритму та налаштуваннями імпортованих плейлистів – синхронно оновлюють локальний стан і передають відповідні зміни до Firestore, підтримуючи актуальність інформації у віддаленій базі даних.

Узагальнену архітектуру застосунку з використанням патерну проєктування Model-View-ViewMode наведено на рис. 3.4.

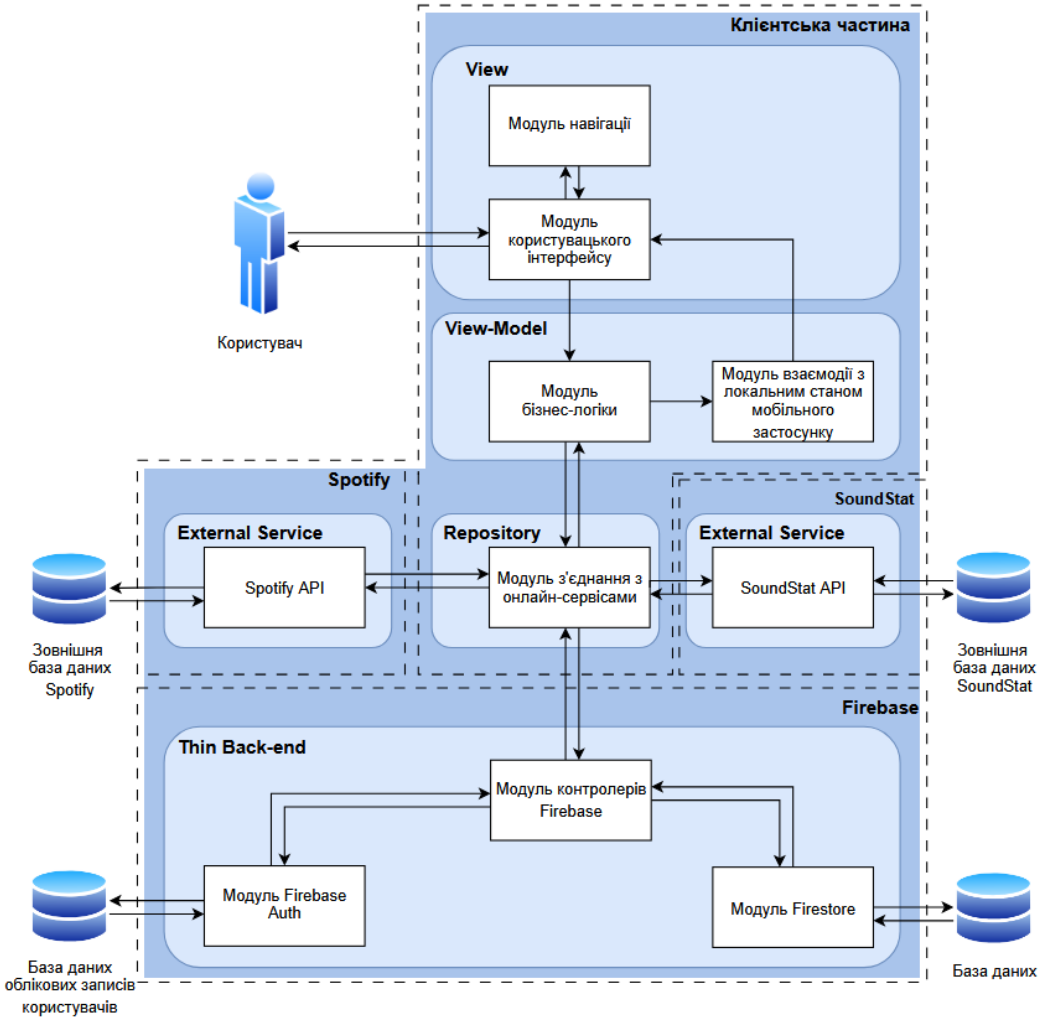


Рис. 3.4. Узагальнена архітектура мобільного застосунку

3.3. Опис структур даних мобільного застосунку

Після успішної реєстрації через Firebase Auth обліковий запис користувача додається до системної таблиці Users в Firebase Authentication. Служба автоматично:

- генерує унікальний ідентифікатор UID;
- хешує пароль відповідно до налаштованого алгоритму;
- фіксує електронну адресу, спосіб реєстрації та дату створення.

Отриманий UID використовується як первинний ключ для всіх пов’язаних документів у Firestore. Під час першого входу клієнтський

застосунок автоматично створює порожні документи з ключем UID у таких колекціях:

- user_mode – налаштування алгоритму добору музики;
- user_playlist – імпортовані плейлисти та треки;
- user_streaming – параметри інтеграції із Spotify.

Кожний документ в колекції user_mode відповідає одному користувачу. В таблиці 3.5 наведено структуру документів відповідної колекції.

Таблиця 3.5

Структура документа з колекції user_mode

Поле	Тип	Призначення
selectedMode	string	Ідентифікатор активного режиму роботи алгоритму ("timer", "map", "pace", "length")
modesInfo	object	Для кожного з чотирьох режимів містить масив об'єктів { indicator, musicTempo }, де indicator – порогове значення зовнішньої характеристики, а musicTempo – цільовий BPM

Кожний документ в колекції user_playlist відповідає одному користувачу, а також містить в собі вкладену під-колекцію playlist. В цій вкладеній під-колекції для кожного імпортованого плейлиста створюється документ з ідентифікатором playlistId (ідентифікатор Spotify). В кожному документі імпортованого плейлиста також наявна під-колекція tracks, яка містить документи про кожний наявний музичний трек в імпортованому плейлистів зі статусом видимості для алгоритму підбору музики. В різних плейлистах може бути інформація про трек з однаковим ідентифікатором, так як один і той самий трек може мати різний статус видимості в різних плейлистах. В таблиці 3.6 наведено структуру документів відповідної колекції.

Таблиця 3.6

Структура документа з колекції user_playlist

Поле	Тип	Опис
active	boolean	Ознака, що плейлист вибраний для поточної пробіжки
streamingId	string	Ідентифікатор власника плейлиста у Spotify; використовується для валідації під час синхронізації
tracks	колекція	Колекція документів-треків, де кожний документ містить значення активності відповідного треку для алгоритму підбору

Кожний документ в колекції user_streaming відповідає одному користувачу. Зміст документу оновлюється кожен раз, коли користувач оновлює токен сесії. Якщо сесію не вийшло оновити, документ очищується і користувачу потрібно буде перезайти у власний акаунт Spotify. В таблиці 3.7 наведено структуру документів відповідної колекції.

Таблиця 3.7

Структура документа з колекції user_streaming

Поле	Тип	Призначення
accessToken	string	Поточний JWT-токен сесії доступу до Spotify API
expiresIn	number	Час життя токена в секундах, за замовчуванням 3600
issuedAt	number	Unix-мітка моменту видачі токена
refreshToken	string	Токен для оновлення accessToken в разі закінчення часу активності

3.4. Опис реалізованих алгоритмів функціонування застосунку

Основними алгоритмами у мобільного застосунку для підбору музичного супроводу під час бігу є алгоритм трансформації фітнес-даних у цільовий темп музики і алгоритм добору музичного супроводу під нього. Ці алгоритми активуються один за одним виключно під час спортивної сесії.

Після натискання користувачем кнопки «Run» система відкриває сторінку Running, де кожні пів секунди пристрій опрацьовує поточне положення користувача і порівнює його зі списком попередніх зафіксованих положень. З цього порівняння система отримує актуальні фітнес-показники: темп руху, пройдена дистанція, час з початку руху, і місцеположення користувача. З урахуванням індивідуальних налаштувань алгоритму підбору ці показники трансформуються у цільове значення темпу музики.

Трансформація відбувається завдяки іншому алгоритму трансформації, який відповідно до користувацької таблиці налаштувань з опціями алгоритму перекладає фітнес-дані в темп музики (рис. 3.5).

Алгоритм запам’ятовує індекс поточної позиції і перевіряє, чи дійшли обрані дані користувача до наступної позиції. Якщо дані дійшли, то цей алгоритм перекладу запам’ятовує індекс наступної позиції і передає відповідне цільове значення темпу музики зазначене в наступній позиції.

Але, якщо обраним режимом роботи алгоритму є режим «Pace», то алгоритм завжди перевіряє індикатор першої опції, а саме:

1. Перетворює індикатор опції на проміжок значень темпу бігу.
2. Обирає темп бігу з поточних фітнес-даних користувача.
3. Порівнює поточний темп бігу з граничними значеннями проміжку:
   - при темпі бігу, що більше найбільшого значення проміжку опції, алгоритм обирає темп музики «Low»;
   - при темпі бігу, що міститься між граничними значеннями з проміжку опції, алгоритм обирає темп музики «Medium»;
   - при темпі бігу, що менше найменшого значення проміжку опції, алгоритм обирає темп музики «High».

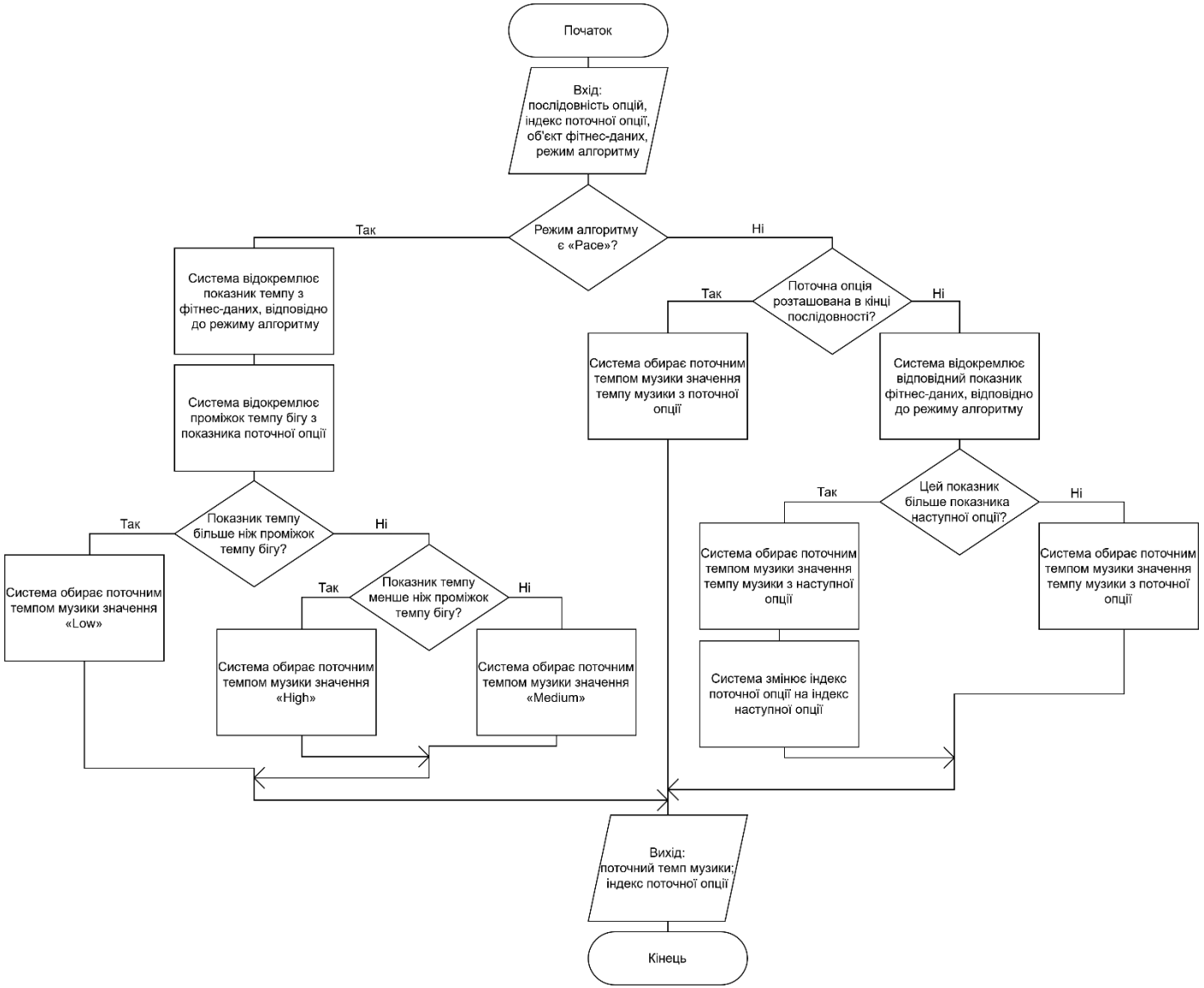


Рис. 3.5. Блок-схема алгоритму трансформації фітнес-даних у цільовий музичний темп

Якщо перерахунок призводить до зміни цільового значення темпу музики, запускається алгоритм добору треків (рис. 3.6). Алгоритм запускається лише зі зміною темпу музики для того, щоб у користувача не збивався музичний супровід при однаковому цільовому значенні темпу музики і для економії ресурсів пристрою користувача.

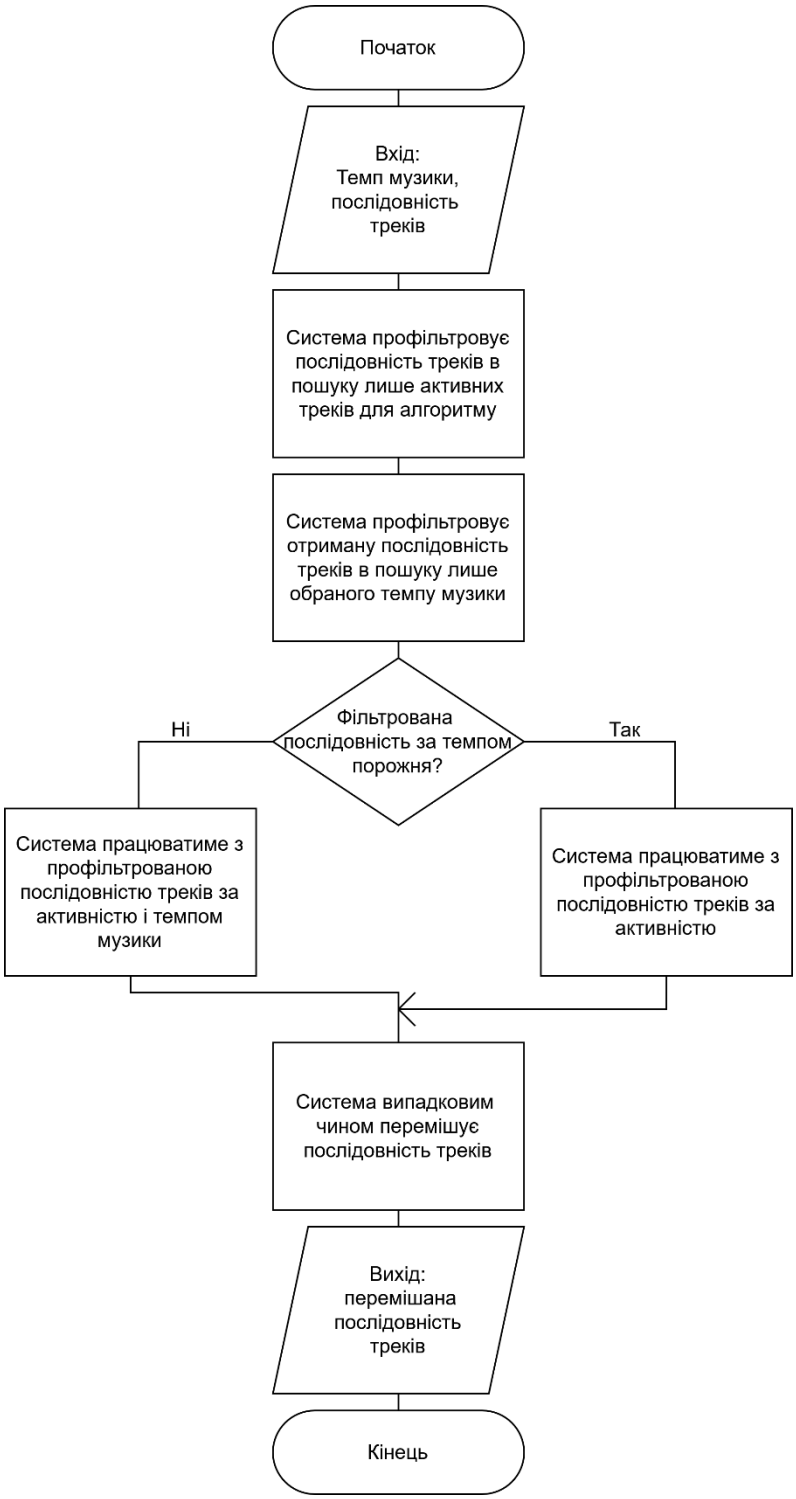


Рис. 3.6. Блок-схема алгоритму добору музичного супроводу

Цим алгоритмом передбачено виконання наступних дій:

1) формування вибірки активних треків з обраного плейлиста;
2) фільтрування композиції усередині вибірки, темп яких відхиляється від потрібного темпу;
3) використання усіх активних треків за відсутності релевантних композицій до потрібного темпу;
4) перемішування обраних композицій у псевдовипадковому порядку алгоритмом Fisher-Yates;
5) повертання впорядкованого масиву ідентифікаторів треків до клієнтського модуля.

Отримана послідовність ідентифікаторів треків передається через Spotify API для створення нового контексту відтворення. Якщо протягом наступних ітерацій значення цільового темпу музики залишається незмінним, а черга треків відпрацьована повністю, Spotify автоматично перезапускає сформований список.

3.5. Висновки до розділу

Під час написання третього розділу дипломного проєкту виконано наступні задачі:

1) проведено проблемно-цільовий аналіз розроблюваного застосунку, побудовано дерева проблем, цілей і запланованих результатів;
2) сформульовано вимоги до системи: функціональні у форматі сценаріїв використання; і нефункціональні, а саме вимоги до безпеки, продуктивності, і реалізації;
3) спроєктовано й описано архітектуру застосунку;
4) визначено й описано організацію даних у Firebase;
5) розроблено алгоритми перекладу фітнес-даних в цільовий темп музики і алгоритм добору музичного супроводу під нього.

4. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ЗАСТОСУНКУ

Мета цього розділу – висвітлити особливості реалізації клієнтської та серверної частини, детально описати створений користувацький інтерфейс, подати план проведення тестувань у вигляді окремих текстових випадків, а також окреслити напрями подальшого розвитку та удосконалення мобільного застосунку для автоматизованого підбору музичного супроводу.

4.1. Опис використаних сторонніх програмних інструментів

Розроблення мобільного застосунку для автоматизованого підбору музичного супроводу здійснювалося згідно з правилами «Clean Code» [21]. Ідентифікатори змінних мають семантично точні назви; повторювані фрагменти коду винесено в окремі функції та компоненти; забезпечено модульність архітектури, високу читабельність і інформативність програмного коду. Для всіх змінних визначено відповідні типи даних використовуючи синтаксис TypeScript, що гарантує більш сувору типізацію в межах проєкту. Такий підхід істотно спрощує подальшу підтримку й масштабування системи, зводячи до мінімуму витрати часу на рефакторинг існуючих рішень.

Серверна частина розроблюваного програмного забезпечення відповідає за автентифікацію користувачів і збереження їхніх даних за допомогою сервісів Firebase. Процес ідентифікації реалізовано через Firebase Authentication, тоді як інформація про користувачів зберігається у Firestore Database.

Ключем кожного запису в Firestore виступає унікальний UID, що був сформований ще під час реєстрації користувача, що гарантує однозначність і швидкий доступ до даних. Застосунок завантажує дані і взаємодіє з базою даних використовуючи UID авторизованого користувача в системі. При видаленні користувача з системи, також видаляються відповідні документи в Firestore Database, що мали ключем UID цього користувача.

Для розроблення клієнтської частини мобільного застосунку обрано екосистему Expo, яка забезпечує ширший набір готових інструментів порівняно з React Native CLI. Серед таких інструментів можна зазначити:

1) expo-auth-session – автентифікація зі Spotify за протоколом OAuth 2.0 [22];
2) expo-router – маршрутизація між екранами застосунку [23];
3) expo-location – визначення поточного місцеперебування користувача [24];
4) expo-web-browser – відкриття зовнішніх посилань у вбудованому браузері [25].

Інтеграцію із сервісами Firebase забезпечують бібліотеки @react-native-firebase/app, @react-native-firebase/auth та @react-native-firebase/firestore [26], які відповідають відповідно за ініціалізацію проєкту, автентифікацію й роботу з базою даних Firestore. Для централізованого керування глобальним станом застосунку використано @reduxjs/toolkit [27], що спрощує підтримку прогнозованої логіки оновлення даних. Підтримку багатомовного інтерфейсу реалізовано за допомогою react-i18next [28], а відображення інтерактивних карт і геоданих користувача – через react-native-maps [29] у поєднанні з Google Maps Platform.

Також варто зазначити, що, через закриття відповідного API для отримання темпу музики зі сторони Spotify [30], було використано сторонній сервіс SoundStat [31]. При імпорті плейлиста застосунок також відправляє ідентифікатори імпортованих треків в систему, і у відповідь він отримує дані аналізу відповідного треку, в яких міститься інформація про темп музики. Але є обмеження в доступності інформації, так як не вся музична бібліотека Spotify була проаналізована. Якщо музична композиція не була проаналізована в системі SoundStat до цього, то сервіс ініціалізує відповідний аналіз, а система TempoRun призначає відповідному треку тимчасове значення Темпу музики «Medium».

4.2. Опис реалізованого користувацького інтерфейсу

Користувацький інтерфейс системи для підбору музичного супроводу під час бігу реалізовано у вигляді мобільного застосунку, що містить в собі наступні екрани:

1) екран реєстрації та авторизації користувача;
2) екран головного меню;
3) екран налаштування режимів алгоритму;
4) екран імпортованих плейлистів;
5) екран налаштування імпортованого плейлиста;
6) екран імпорту плейлистів з інтегрованого профілю Spotify;
7) екран налаштувань застосунку;
8) екран налаштування акаунту;
9) екран спортивної сесії;
10) екран результатів спортивної сесії.

Екран автентифікації TempoRun містить логотип застосунку (рис. 4.1). Нижче розташовано два текстові поля для введення облікових даних. Перше поле призначене для електронної адреси. Друге поле призначене для пароля, містить іконку-перемикач у вигляді перекресленого ока, що дає змогу приховувати або показувати введені символи.

Під полями розміщено блок інтерактивних елементів керування. Кнопка «Sign up» ініціює реєстрацію нового користувача: введені email та пароль надсилаються на Firebase Auth, де перевіряється унікальність адреси, після чого за успішного результату створюється новий обліковий запис.

Кнопка «Log In» ініціює авторизацію користувача: введені email та пароль надсилаються на Firebase Auth, де вони перевіряються відповідність до існуючого акаунту і правильність введених даних. Після перевірки за успішного результату система входить в обліковий запис.

У разі не правильного вводу облікових даних, система змінює колір рамки полів вводу на червоне і відображає текст помилки під ними.

Остання кнопка «Log in with Google» при натисканні відкриває системний діалог вибору акаунта Google, завдяки якому користувач зможе пройти авторизацію OAuth 2.0 через Google Identity Services.

У разі успіху будь-якого з процесів система автоматично відображає головне меню застосунку.

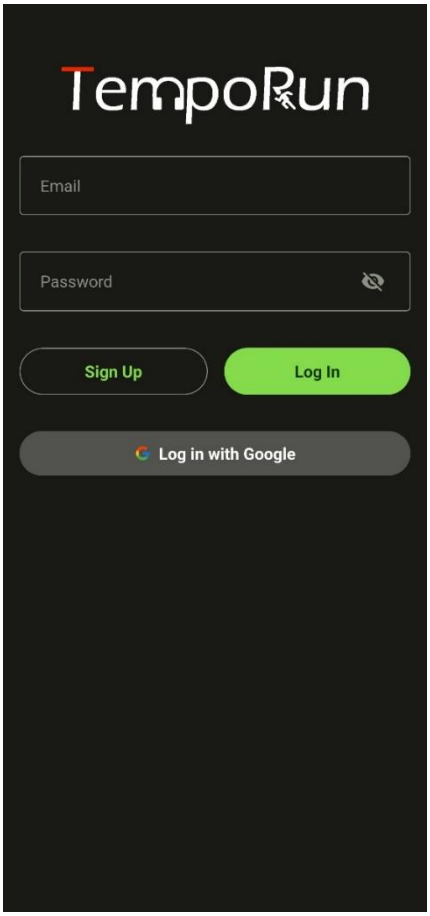


Рис. 4.1. Екран реєстрації та авторизації користувача

Екран головного меню відображає логотип «TempoRun», центральну кнопку «Run» для старту тренування, інформаційну картку з поточним режимом роботи алгоритму (іконка і назва) та картку активного плейлиста: обкладинка і назва (рис. 4.2). Торкання картки режиму відкриває налаштування алгоритму, а картки плейлиста – перелік імпортованих плейлистів. Унизу розташована панель навігації з чотирма піктограмами: режим, біг, плейлисти та налаштування.

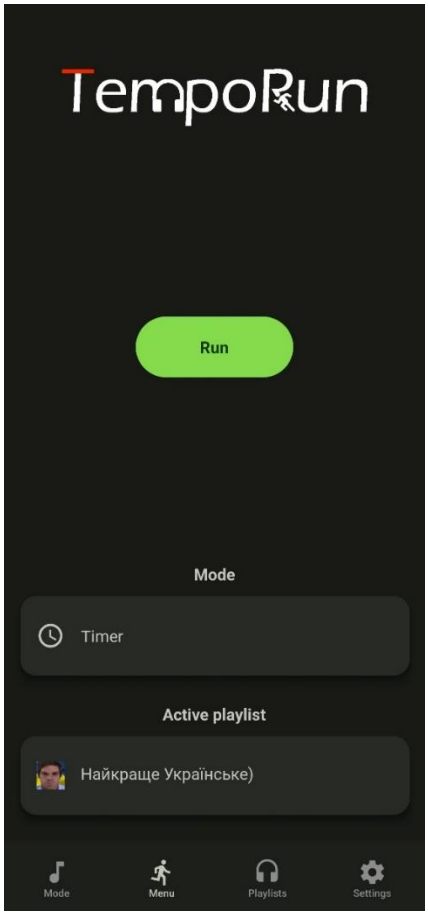


Рис. 4.2. Екран головного меню

Екран налаштування режимів алгоритмів показує перемикач режимів: «Timer», «Map», «Length» і «Pace»; активний режим підсвічено, а при виборі іншого режиму, система одразу підвантажує відповідну таблицю налаштувань (рис. 4.3).

У таблиці кожен рядок містить значення обраного індикатора (час, відстань тощо), бажаний темп музики («Low»/«Medium»/«High») та, окрім стартового рядка, кнопку видалення.

Значення індикатора редагується через модальне вікно (для всіх режимів, крім «Map» і «Pace») і після підтвердження автоматично сортується за зростанням; темп змінюється з випадаючого списку.

Натискання на кнопку видалення з іконкою хрестику прибирає рядок, а кнопка «+ Add Option» додає новий пустий запис, який з’являється внизу списку.

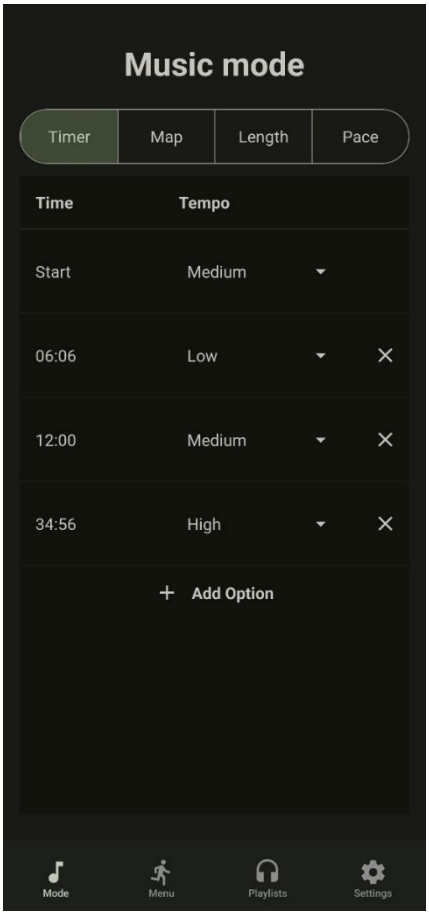


Рис. 4.3. Екран налаштування роботи алгоритму

З активним режимом «Map» під перемикачем режимів з’являється вбудована Google-карта (рис. 4.4). На ній відображаються маркери вибраних точок із напівпрозорими радіусами: перша точка позначає поточне місце користувача (старт), тому вона не відображається на мапі, а решта точок відображаються на мапі з автоматично присвоєними літерними позначками A, B, C і тд. Маркер можна перетягнути, щоб уточнити координати.

Натискання кнопки «+ Add Option» додає нову точку поруч із останньою (або на позиції користувача, якщо це друга точка після старту).

Таблиця під картою працює так само, як у решті режимів: для кожної точки задається темп «Low»/«Medium»/«High», рядки можна видаляти, а стартовий рядок залишається незмінним (окрім значення темпу обраного темпу музики).

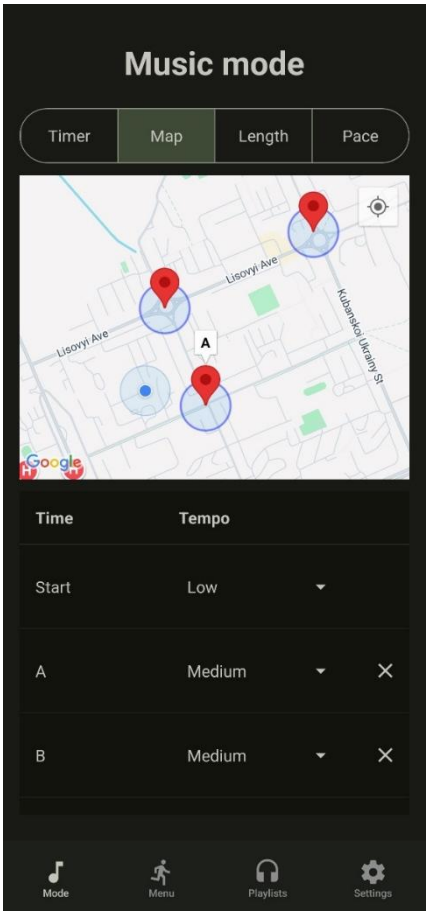


Рис. 4.4. Екран налаштування роботи алгоритму з обраним режимом «Map»

З активним режимом «Pace» під перемикачем режимів з'являється повзунок з двома точками (рис. 4.5). За допомогою переносу цих точок, користувач має можливість змінювати проміжок за яким система буде підбирати «Medium» темп музики. Якщо користувач матиме значення темпу бігу менше ніж найменше значення цього проміжку, то система буде підбирати «High» темп музики. Якщо буде навпаки більший темп ніж найбільше значення проміжку, то система буде підбирати «Low» темп музики.

Нижче під цим повзунком розташована пояснювальна таблиця, яка показує при яких значення темпу систему буде обирати відповідне значення темпу музики. Ця таблиця оновлюється з кожним перенесенням точок на повзунку.

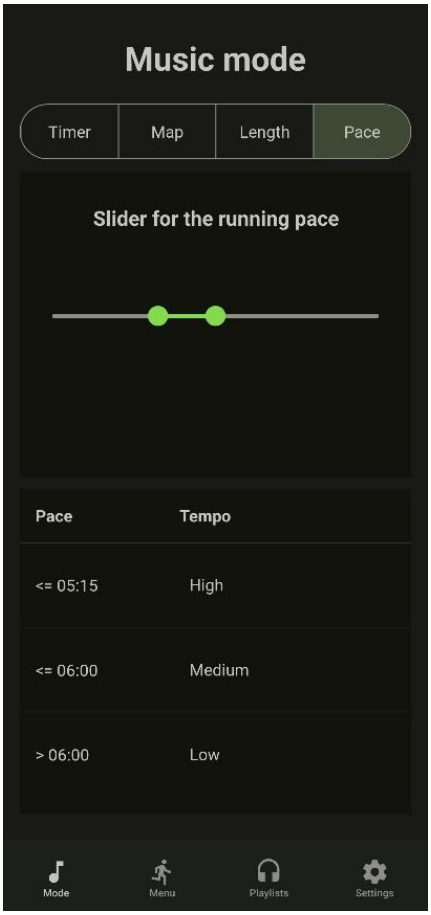


Рис. 4.5. Екран налаштування роботи алгоритму з обраним режимом «Pace»

Екран імпортованих плейлистів відображає перелік імпортованих плейлистів із підключеного музичного сервісу (рис. 4.6). Кожен елемент списку містить обкладинку, назву плейлиста, джерело (в нашому випадку, це Spotify) та кнопку з іконкою трьох крапок для переходу до його налаштувань. Натискання по картці робить плейлист активним і підсвічує його.

Якщо користувач ще не імпортував плейлисти до цього, система відображає повідомлення «No playlists to activate». Повідомлення зникне після того, як користувач імпортує свій перший плейлист в систему.

У правому нижньому куті розташована плаваюча кнопка (FAB) з іконкою плюса («+»), що відкриває екран імпорту нових плейлистів.

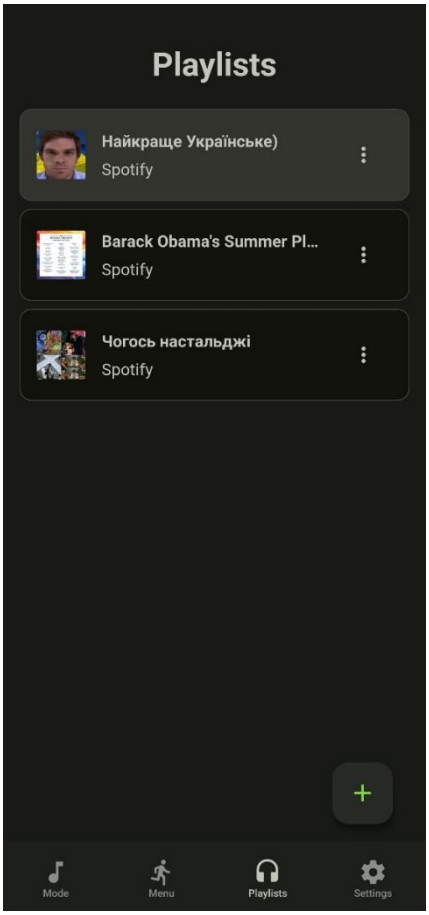


Рис. 4.6. Екран зі списком імпортованих плейлистів

Екран налаштування плейлиста містить верхню панель із кнопкою з іконкою стрілки («←») для повернення, назвою списку та кнопкою «Delete» для видалення плейлиста з бази даних TempoRun (рис. 4.7).

Нижче відображається перелік музичних композицій: кожен рядок показує обкладинку альбому, назву композиції, виконавця та значок ока. Неперекреслене око означає, що композиція враховуватиметься алгоритмом; перекреслене – буде пропущений. Торкання рядка перемикає стан видимості.

Якщо плейлист порожній система відображає повідомлення «No music tracks to manage». Повідомлення зникне після того, як користувач наповнить треками обраний плейлист на стороні Spotify і перезапустить застосунок для оновлення даних.

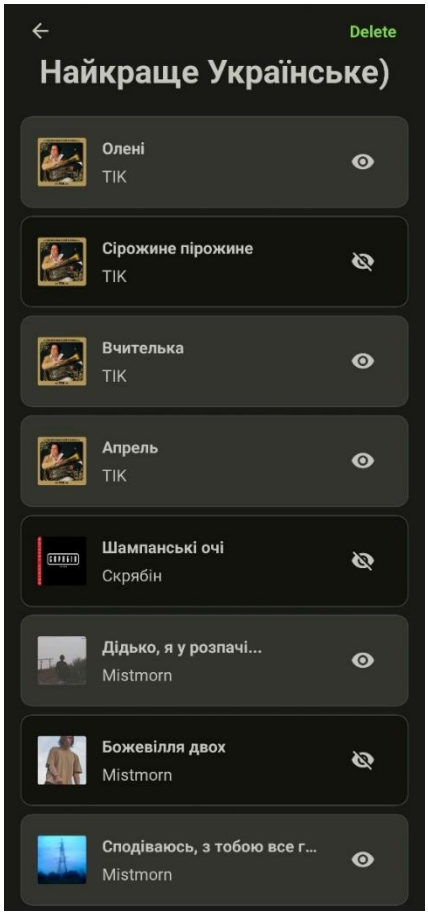


Рис. 4.7. Екран налаштування імпортованого плейлиста

Екран імпорту плейлистів містить верхню панель із кнопкою повернення та іконкою шестерні, що відкриває налаштування (рис. 4.8).

Під панеллю розташовано заголовок «Import playlist» і карточку із логотипом «Spotify» і назвою під'єднаного профілю. Нижче в цій карточці виводиться перелік плейлистів, яких ще немає у TempoRun: кожна картка містить обкладинку, назву списку, джерело (в нашому випадку це «Spotify») та кнопку «+». Натискання на кнопку з іконкою плюса («+») запускає процедуру імпорту вибраного плейлиста, після чого його картка зникає зі списку.

Якщо профіль Spotify ще не інтегровано, або на інтегрованому профілі немає особистих плейлистів, система відображає повідомлення «No playlists to import». Повідомлення зникне після того, як користувач інтегрує профіль Spotify з плейлистами, або добавить плейлисти на стороні Spotify до

інтегрованого профілю (після додавання плейлистів необхідно перезапустить застосунок для оновлення даних).

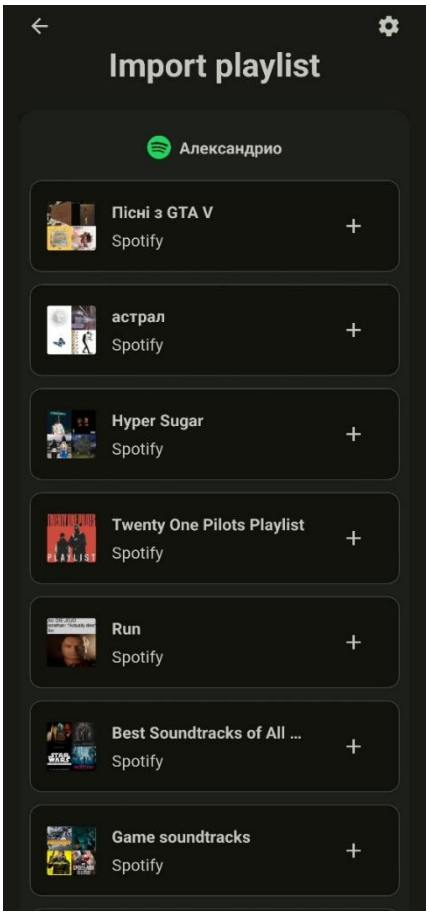


Рис. 4.8. Екран імпорту плейлиста

Екран налаштування застосунку відкриває картку «Linked account»: якщо «Spotify» ще не під’єднано, відображається лише іконка сервісу для запуску інтеграції, а після успішного зв’язування з’являються назва профілю й аватар у вигляді карточки (рис. 4.9). Натискання на цю карточку переводить користувача безпосередньо у «Spotify» (або на його вебсторінку, якщо застосунок не встановлено). Далі розташовано параметри застосунку: перемикач теми змінює світле й темне оформлення, а пункт мови переключає інтерфейс між англійською та українською. У блоці «Account» можна відкрити форму редагування особистих даних або вийти з облікового запису, після чого користувача повертає на екран авторизації.

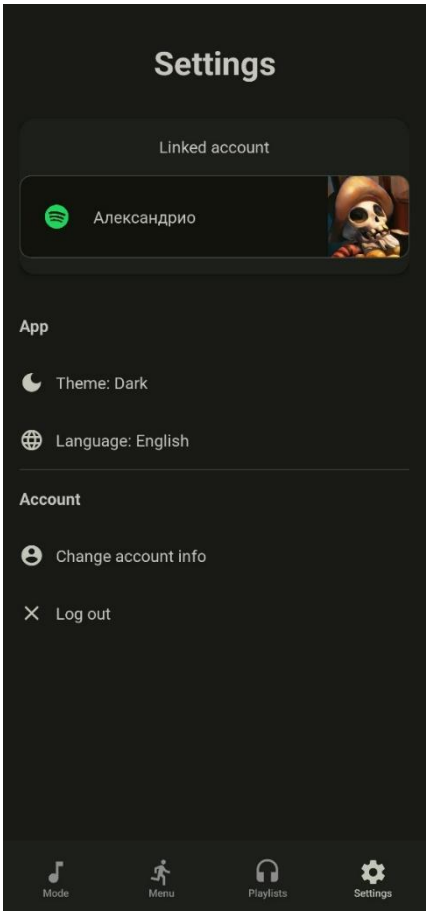


Рис. 4.9. Екран імпорту плейлиста

Екран налаштування акаунту подає перелік керувальних дій для облікового запису: змінити електрону пошту, змінити пароль, від’єднати «Spotify» та видалити аккаунт в системі TempoRun (рис. 4.10). Пункти відображаються залежно від типу авторизації: зміна пошти й пароля доступні лише користувачам із логіном-паролем, а від’єднання – лише після під’єднання акаунту «Spotify».

Натискання на пункт відкриває відповідний модальне вікно введення нових даних або підтвердження дії. Для зміни пошти відкривається модальне вікно з одним полем вводу, а для зміни паролю – з двома полями (для паролю і його повтору). Для видалення інтеграції з профілем Spotify і для видалення акаунту відкривається підтверджувальне модальне вікно.

Кнопка з іконкою стрілки («←») у заголовку використовується для повернення на попередній екран.

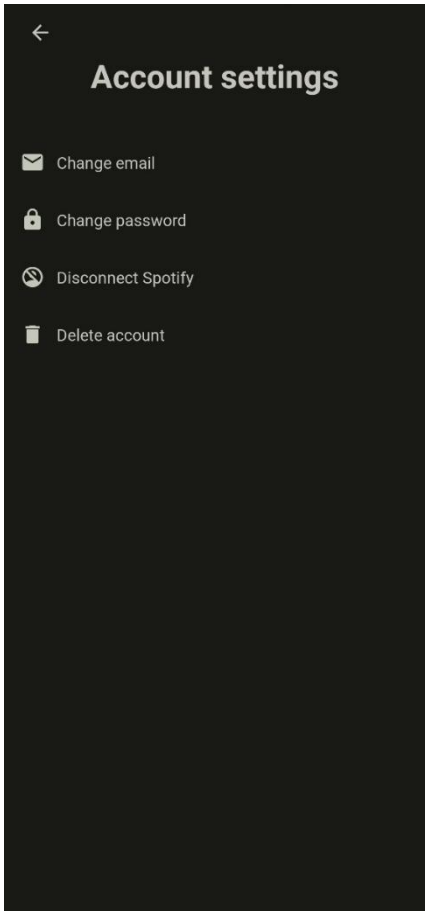


Рис. 4.10. Екран налаштування акаунту

Екран тренування відображає три поточні показники – час від старту в секундах, поточний темп бігу в хвилинах та подолану дистанцію в кілометрах (рис. 4.11). Значення таблиці поточних показників оновлюється двічі на секунду.

Нижче розташована картка активної опції алгоритму: іконка й назва вибраного режиму, цільовий музичний темп та позиція опції у черзі. Якщо обраний режимом є «Pace», то позиція опції у черзі не буде відображена на інформаційній карточці активної опції алгоритму.

У нижній частині екрана знаходиться велика кнопка-стоп з іконкою квадрата. Натискання на неї завершує сесію й відкриває сторінку результатів.

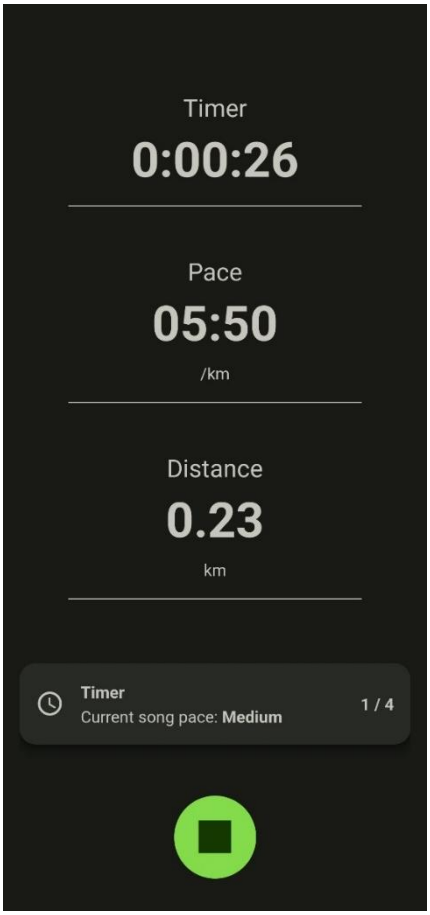


Рис. 4.11. Екран спортивної сесії

Після завершення сесії тренування відображається підсумковий екран: у центрі картка з даними про дистанцію, тривалість, середній темп і кількість пройдених опцій, а під нею розташовано кнопку «Go back», при натисканні на яку система відкриває головне меню (рис. 4.12).

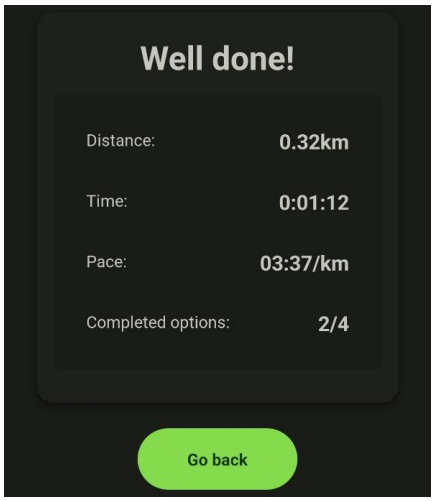


Рис. 4.12. Таблиця результатів спортивної сесії

4.3. Особливості проведеного тестування

З метою системної та вичерпної перевірки відповідності програмного продукту встановленим вимогам було сформовано відповідний набор тест-кейсів. Для оцінювання працездатності мобільного застосунку з підбору музичного супроводу обрано метод димного-тестування, який передбачає послідовне виконання всіх користувацьких сценаріїв у режимі реального використання.

Нижче наведено інструкцію проведених випробувань (табл. 4.1).

Таблиця 4.1

Тестові випадки

№	Назва	Передумова	Сценарій дій	Очікуваний результат
1	Можливість реєстрації у системі	Користувач не має облікового запису	1. Користувач вводить email і пароль. 2. Користувач натискає на кнопку «Sign up».	Аккаунт створено; система показує головне меню
2	Можливість авторизації у системі	Користувач зареєстрований у системі	1. Користувач вводить коректний email і пароль. 2. Користувач натискає на кнопку «Log in».	Виконано вхід; система завантажила користувацькі дані; система показує головне меню
4	Можливість зміни теми застосунку	Користувач авторизований; Темна тема обрана як тема застосунку	1. Користувач натискає на кнопку «Settings» на панелі знизу. 2. Користувач натискає на пункт «Theme: Dark».	Тема застосунку перемикається з темної на світлу тему

Продовження табл. 4.1

5	Можливість під’єднати акаунт Spotify до профілю в системі	Користувач авторизований у системі; користувач не має під’єднаного акаунту Spotify	1. Користувач натискає на кнопку «Settings» на панелі знизу. 2. Користувач натискає на іконку Spotify в панелі інтеграції акаунту. 3. Користувач авторизується у Spotify і надає доступ до особистих даних.	Система запам’ятала токен сесії Spotify; система зберегла токен сесії на Firebase Database; система відображає ім’я і логотип профілю користувача
6	Можливість імпорту плейлиста з особистого профілю Spotify	Користувач авторизований в системі; профіль Spotify під’єднано; на стороні Spotify в профілі користувача доступні персональні плейлисти	1. Користувач натискає на кнопку «Playlists» на панелі. 2. Користувач натискає на плавучу кнопку з іконкою плюса. 3. Користувач натискає на кнопку з іконкою плюса на обраній карточці ще не імпортованого плейлиста.	Плейлист додано до системи; прибрано картку доданого плейлиста зі списку імпорту; додано картку доданого плейлиста в список вже імпортованих плейлистів
7	Можливість зміни режиму алгоритму на режим «Length»	Користувач авторизований в системі; обраним режимом алгоритму не є режим «Length»	1. Користувач натискає на кнопку «Mode» на панелі знизу в головному меню. 2. Користувач натискає на панелі вибору режимів на кнопку «Length».	Обраним режимом роботи алгоритму в системі став режим «Length»; в таблиці опцій налаштування алгоритму оновлено вміст на опції режиму «Length»

Продовження табл. 4.1

8	Можливість додавання опції для режиму «Timer»	Користувач авторизований в системі; режим «Timer» обраний як поточний режим роботи алгоритму	1. Користувач натискає на кнопку «Mode» на панелі знизу. 2. Користувач натискає на кнопку «+ Add Option» в таблиці опцій.	Система додає нову опцію в кінці списку для режиму «Timer»; нова опція має порожній вміст як індикатор і значення «Medium» як цільове значення музики
9	Можливість зміни індикатору опції для режиму «Timer»	Користувач авторизований в системі; режим «Timer» обраний як поточний режим роботи алгоритму; в таблиці опцій для режиму «Timer» доступні декілька опцій окрім початкової	1. Користувач натискає на кнопку «Mode» на панелі знизу. 2. Користувач натискає на клітинку в першому стовпчику другої опції з обраним часом. 3. Користувач вводить бажану кількість хвилин і секунд. 4. Користувач натискає на кнопку «OK».	Система змінює індикатор обраної опції на новий; система сортує список опції за зростанням індикатора опцій; в таблиці відображено оновлений вміст
10	Можливість запуску спортивної сесії	Користувач авторизований; алгоритм налаштовано; плейлист обрано; користувач надав доступ до свого місце-положення	1. Користувач натискає на кнопку «Run» на головному меню.	Розпочато спортивну сесію; клієнт Spotify починає програвати підібрану музику під цільовий темп, що був визначений початковою опцією

Продовження табл. 4.1

11	Можливість зупинки спортивної сесії	Користувач авторизований; алгоритм налаштовано; плейлист обрано; користувач надав доступ до свого місце-положення; розпочато спортивну сесію	1. Користувач пересувається з телефоном 10 секунд. 2. Користувач натискає на кнопку стоп.	Завершено спортивну сесію; клієнт Spotify перестає програвати музику; відкрито екран результатів із коректними показниками часу й дистанції

4.4. Рекомендації до подальшого вдосконалення

У результаті аналізу створеного мобільного застосунку для підбору музичного супроводу відповідно до фітнес-даних користувача було визначено список рекомендацій для подальшого розвитку розроблюваного ПЗ, а саме:

1. Забезпечити підтримку інтеграції профілів з Apple Music, YouTube Music, Deezer, SoundCloud та інших популярних музичних стрімінгових платформ.
2. Удосконалити алгоритм добору треків, а саме:
   a. Розширити режими роботи новими типами фітнес даних.
   b. Пропонувати користувачу вже заготовані набори опцій для бігу (інтервальний біг, прогресивне прискорення тощо).
   c. Впровадити алгоритми штучного інтелекту для персоналізованих рекомендацій замість послідовного проходження по опціям користувача.
   d. Аналізувати темп музики не лише як середнє значення треку, а й по сегментах композиції, для динамічного зміщування точки переходу до наступної композиції, мінімізуючи зміни темпу музики.

3. Удосконалити систему зворотного зв’язку із користувачем, а саме:
    a. Відстежувати реакцію користувача на згенерований добір музики у вигляді покращення фітнес-показників під окремі підібрані треки.
    b. Надати можливість оцінювати добір музики й коригувати запропонований темп для окремих імпортованих треків.
4. Зберігати історію спортивних сесій і відображати динаміку показників у вигляді графіків із прив’язкою до відтворених композицій.
5. Інтеграція з персональними фітнес пристроями для покращення зчитування фітнес-характеристик користувача і для зчитування серцевого ритму.
6. Співпраця з музичними сервісами у вигляді спільних підписок, спеціально створених музичних добірок і інших ексклюзивних пропозицій.

4.5. Висновки до розділу

Під час написання четвертого розділу дипломного проєкту було виконано наступні завдання:

1) схарактеризовано особливості програмної реалізації клієнтської та серверної частини ПЗ;
2) описано активні елементи розробленого користувацького інтерфейсу, що був узгоджений згідно з рекомендаціями Google Material 3;
3) підготовлено таблицю реалізованих тестових випадків, що були використані під час димного тестування ПЗ;
4) сформульовано список рекомендацій щодо подальшого удосконалення розроблюваного ПЗ.

ВИСНОВКИ

Метою даного дипломного проєкту було розроблення мобільного застосунку TempoRun, призначеного для автоматизованого добору музичного супроводу під час бігу на основі актуальних фізичних показників користувача. Підсумком виконаної роботи став повнофункціональний програмний продукт у формі мобільного застосунку, який у режимі реального часу інтегрується з особистою бібліотекою Spotify.

У процесі розроблення досліджено предметну галузь персоналізованих спортивних застосунків, проаналізовано існуючі рішення та виявлено їхні обмеження щодо гнучкого керування музичним супроводом. Показано, що через наявні обмеження у задоволенні потреб потенційних користувачів розроблення мобільного застосунку для підбору музичного супроводу є актуальним.

За результатами порівняльного аналізу інструментів розроблення ПЗ для клієнтської частини було обрано React Native разом із платформою Expo, а для серверної складової – Firebase. Система спроєктована за клієнт-серверною моделлю з використанням патерну Model-View-ViewModel у поєднанні з концепцією Thin Back-End, що мінімізує навантаження на сервер та спрощує масштабування сервісу. У межах реалізації дипломного проєкту створено інтуїтивний користувацький інтерфейс, впроваджено алгоритм трансформації фітнес-характеристик у цільовий діапазон музичного темпу і забезпечено механізм добору треків з імпортованих плейлистів відповідно до розрахованого темпу музики.

Розробку виконано у повному обсязі, а отриманий програмний продукт відповідає функціональним і нефункціональним вимогам, визначеним технічним завданням та пояснювальною запискою. Комплексне тестування, проведене згідно із затвердженою методикою, підтвердило відповідність системи поставленим критеріям якості.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Strava: Run, Bike, Walk [Електронний ресурс]. – Режим доступу: https://www.strava.com. – Дата доступу: 20.03.2025 р.
2. Running Beats BPM for Spotify [Електронний ресурс]. – Режим доступу: https://apps.apple.com/us/app/running-beats-bpm-for-spotify/id1609099381. – Дата доступу: 20.03.2025 р.
3. ZRX: Zombies Run + Marvel Move [Електронний ресурс]. – Режим доступу: https://zrx.app. – Дата доступу: 21.03.2025 р.
4. Kotlin for Android | Kotlin Documentation [Електронний ресурс]. – Режим доступу: https://kotlinlang.org/docs/android-overview.html. – Дата доступу: 08.04.2025 р.
5. Kotlin Multiplatform | Kotlin Documentation [Електронний ресурс] – Режим доступу: https://kotlinlang.org/docs/multiplatform.html. – Дата доступу: 08.04.2025 р.
6. Jetpack Compose UI App Development Toolkit – Android Developers [Електронний ресурс]. – Режим доступу: https://developer.android.com/compose. – Дата доступу: 08.04.2025 р.
7. Swift – Apple Developer [Електронний ресурс]. – Режим доступу: https://developer.apple.com/swift. – Дата доступу: 09.04.2025 р.
8. View fundamentals | Apple Developer Documentation [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/SwiftUI/View-fundamentals. – Дата доступу: 09.04.2025 р.
9. React Native Learn once, write anywhere [Електронний ресурс]. – Режим доступу: https://reactnative.dev. – Дата доступу: 10.04.2025 р.
10. React Fundamentals · React Native [Електронний ресурс]. – Режим доступу: https://reactnative.dev/docs/intro-react. – Дата доступу: 10.04.2025 р.

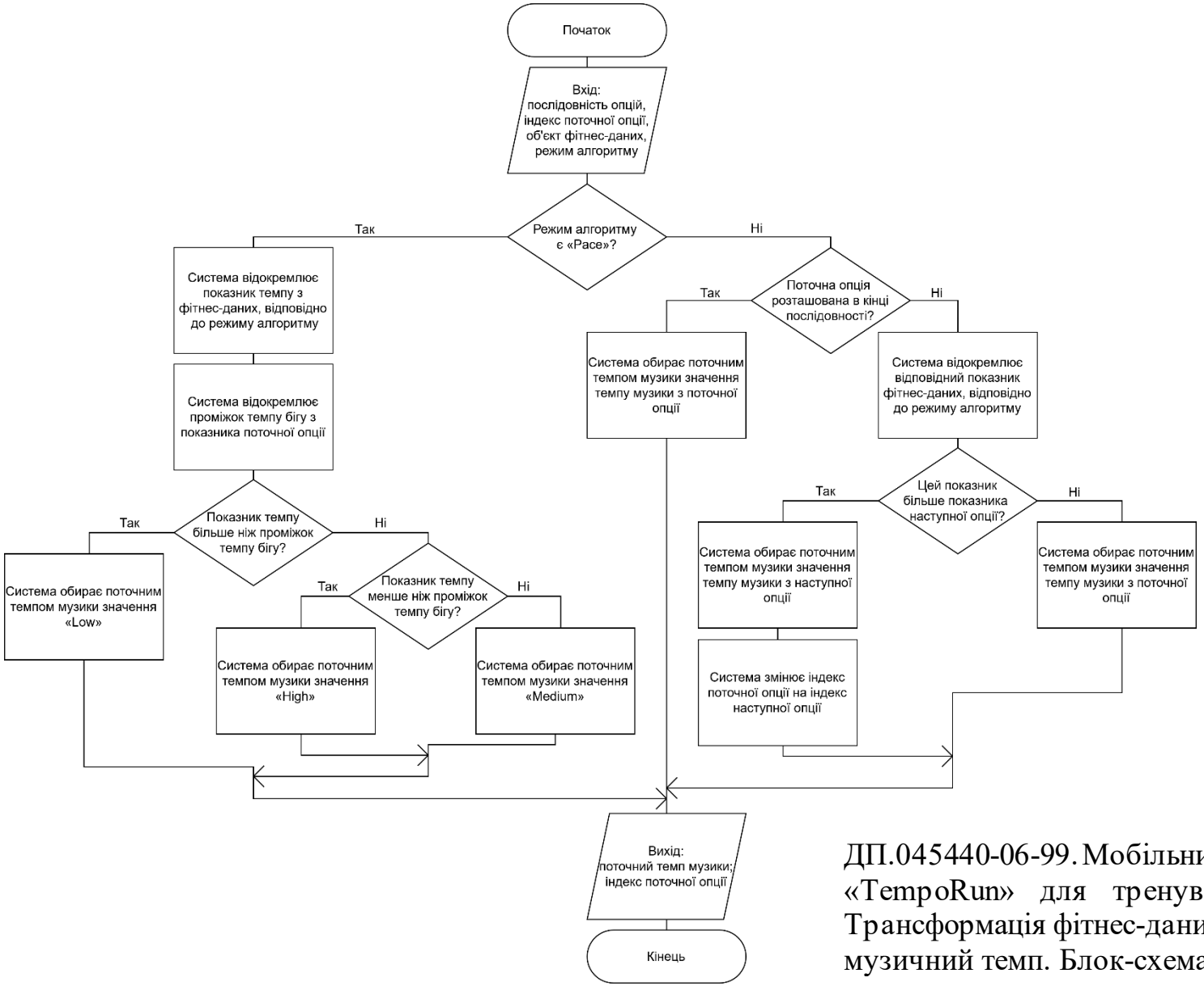
11. Introduction – Expo Documentation [Електронний ресурс]. – Режим доступу: https://docs.expo.dev/get-started/introduction. – Дата доступу: 10.04.2025 р.
12. Firebase | Google's Mobile and Web App Development Platform [Електронний ресурс]. – Режим доступу: https://firebase.google.com. – Дата доступу: 15.04.2025 р.
13. Firebase Authentication [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/auth. – Дата доступу: 15.04.2025 р.
14. Firestore | Firebase [Електронний ресурс]. – Режим доступу: https://firebase.google.com/docs/firestore. – Дата доступу: 15.04.2025 р.
15. Supabase | The Open Source Firebase Alternative [Електронний ресурс]. – Режим доступу: https://supabase.com/. – Дата доступу: 16.04.2025 р.
16. Auth | Built-in user management [Електронний ресурс]. – Режим доступу: https://supabase.com/auth. – Дата доступу: 16.04.2025 р.
17. Full Stack Development – Web and Mobile Apps – AWS Amplify [Електронний ресурс]. – Режим доступу: https://aws.amazon.com/amplify. – Дата доступу: 16.04.2025 р.
18. Що таке SMART-цілі? | HURMA [Електронний ресурс]. – Режим доступу: https://hurma.work/blog/smart-czili-pomylky-ta-porady. – Дата доступу: 25.04.2025 р.
19. Ну і чого ти хочеш?. Поговоримо про вимоги і які вони… | by BAsha | Medium [Електронний ресурс]. – Режим доступу: https://medium.com/@olenabochulia/ну-і-чого-ти-хочеш-a7ed99a42134. – Дата доступу: 25.04.2025 р.
20. Архітектура програмного забезпечення – Wikiwand [Електронний ресурс]. – Режим доступу: https://www.wikiwand.com/uk/articles/Архітектура_програмного_забезпечення. – Дата доступу: 25.04.2025 р.
21. ryanmcdermott/clean-code-javascript: Clean Code concepts adapted for JavaScript [Електронний ресурс]. – Режим доступу:

https://github.com/ryanmcdermott/clean-code-javascript. – Дата доступу: 10.04.2025 р.
22. AuthSession – Expo Documentation [Електронний ресурс]. – Режим доступу: https://docs.expo.dev/versions/latest/sdk/auth-session. – Дата доступу: 29.04.2025 р.
23. Introduction to Expo Router – Expo Documentation [Електронний ресурс]. – Режим доступу: https://docs.expo.dev/router/introduction. – Дата доступу: 29.04.2025 р.
24. Location – Expo Documentation [Електронний ресурс]. – Режим доступу: https://docs.expo.dev/versions/latest/sdk/location. – Дата доступу: 29.04.2025 р.
25. WebBrowser – Expo Documentation [Електронний ресурс]. – Режим доступу: https://docs.expo.dev/versions/latest/sdk/webbrowser. – Дата доступу: 29.04.2025 р.
26. React Native Firebase | React Native Firebase [Електронний ресурс]. – Режим доступу: https://rnfirebase.io. – Дата доступу: 29.04.2025 р.
27. Redux Toolkit | Redux Toolkit [Електронний ресурс]. – Режим доступу: https://redux-toolkit.js.org. – Дата доступу: 29.04.2025 р.
28. Introduction | react-i18next documentation [Електронний ресурс]. – Режим доступу: https://react.i18next.com. – Дата доступу: 29.04.2025 р.
29. react-native-maps – Expo Documentation [Електронний ресурс]. – Режим доступу: https://docs.expo.dev/versions/latest/sdk/map-view. – Дата доступу: 29.04.2025 р.
30. Introducing some changes to our Web API | Spotify for Developers [Електронний ресурс]. – Режим доступу: https://developer.spotify.com/blog/2024-11-27-changes-to-the-web-api. – Дата доступу: 18.04.2025 р.
31. SoundStat.info – Audio Analysis API [Електронний ресурс]. – Режим доступу: https://soundstat.info. – Дата доступу: 18.04.2025 р.

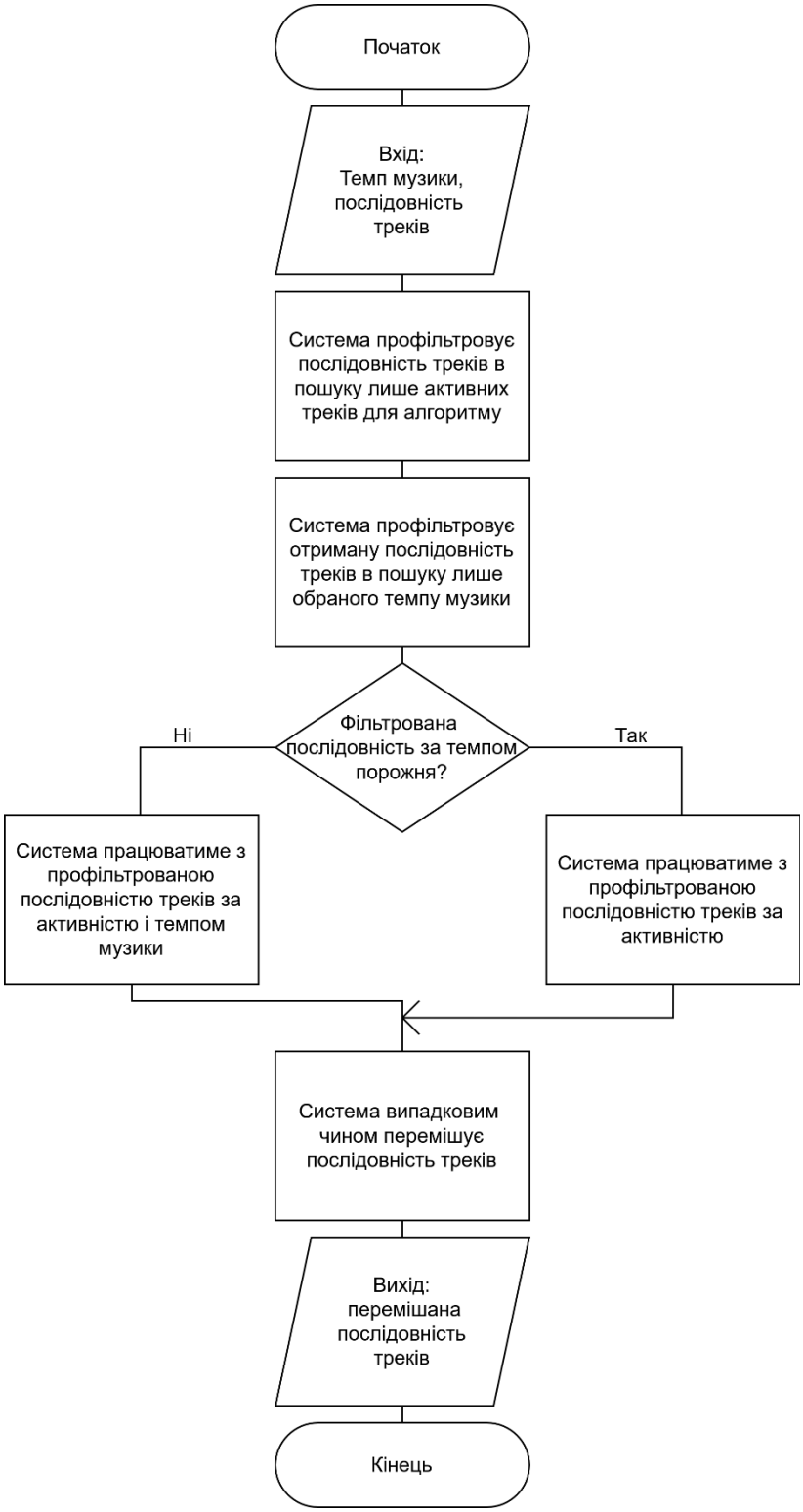
ДОДАТКИ

Додаток 1

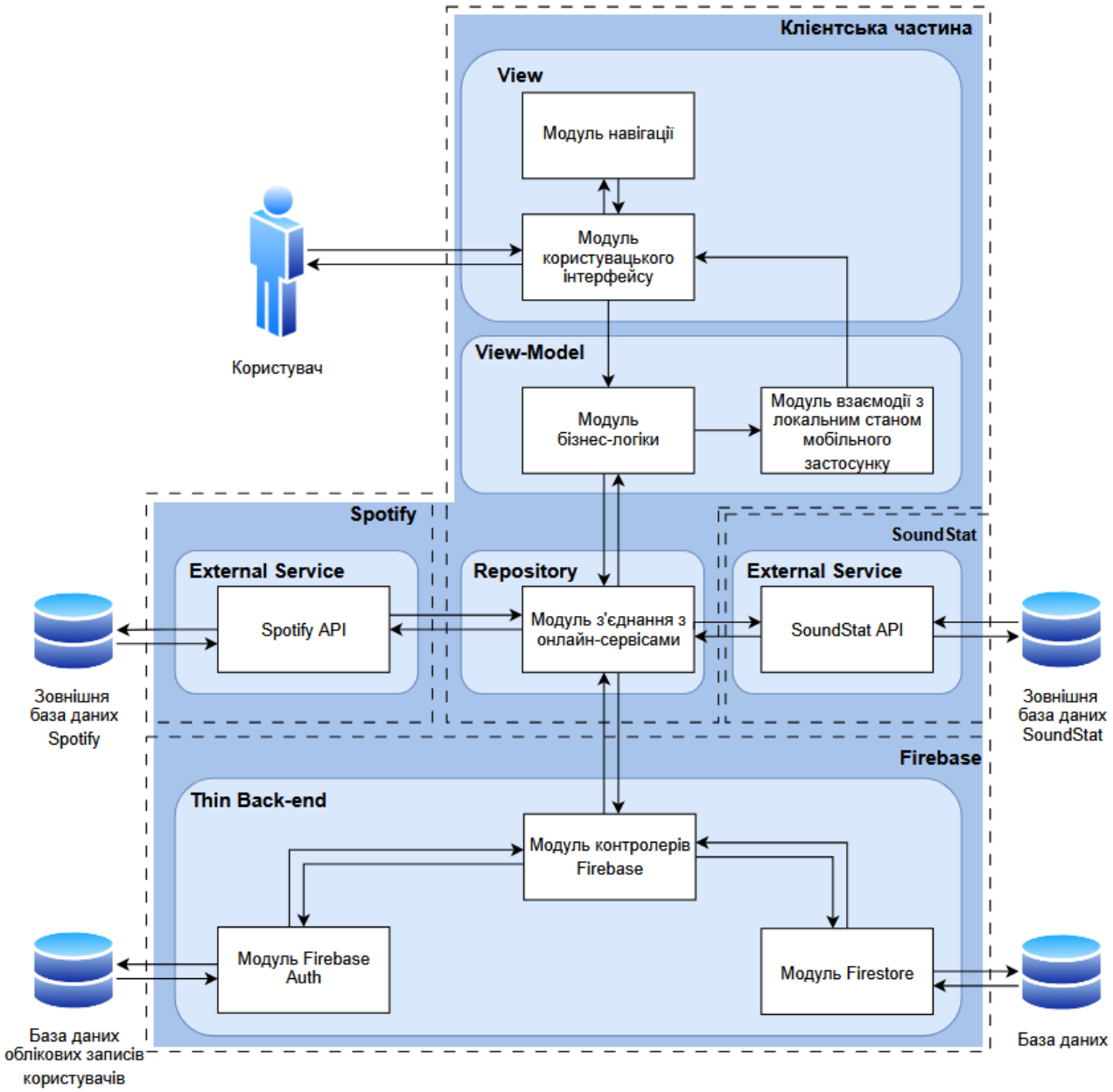
Копії графічних матеріалів



ДП.045440-06-99. Мобільний застосунок «TempoRun» для тренувань з бігу. Трансформація фітнес-даних у цільовий музичний темп. Блок-схема алгоритму

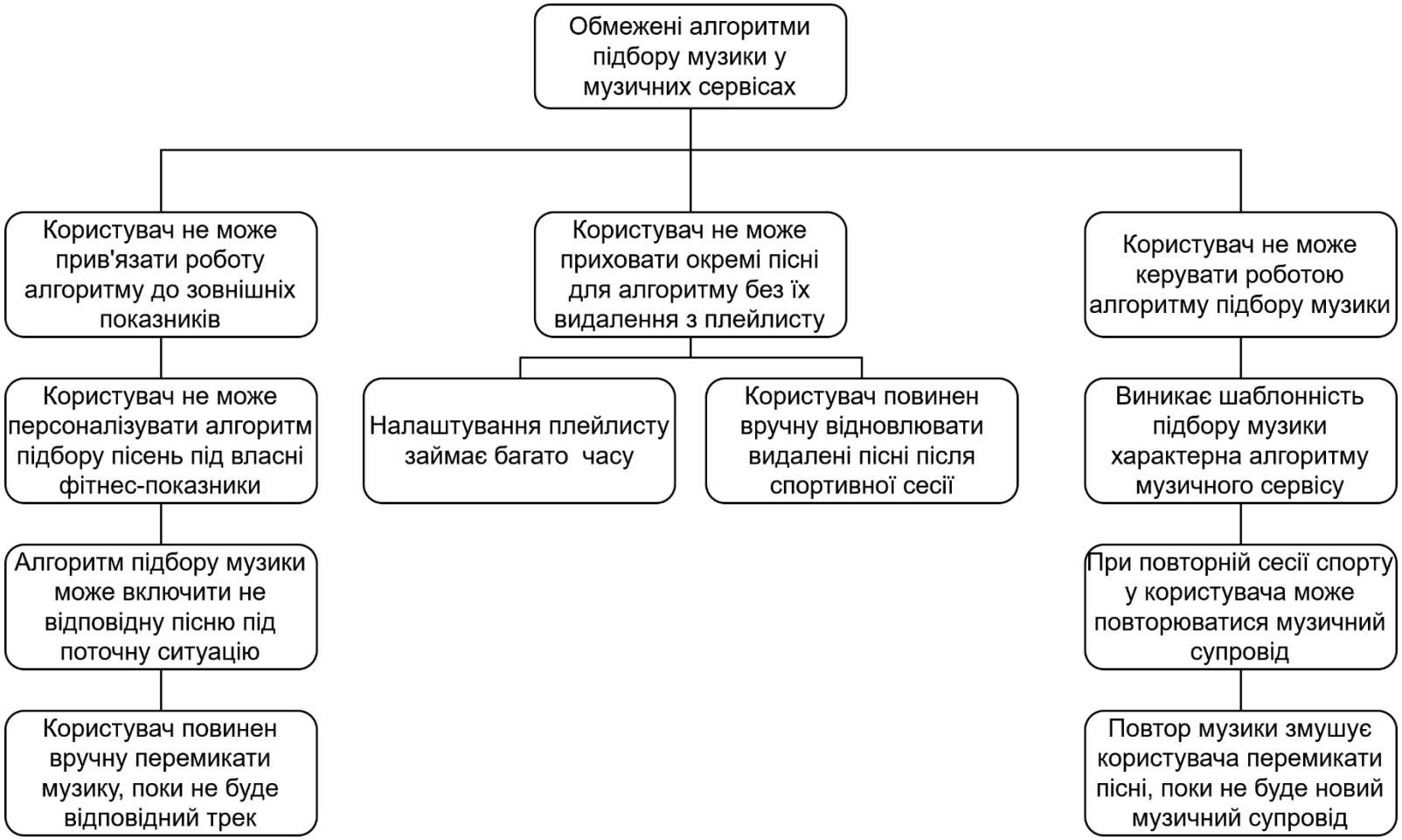


ДП.045440-07-99. Мобільний застосунок «TempoRun» для тренувань з бігу. Підбір музичного супроводу під час бігу. Блок-схема алгоритму



Архітектура мобільного застосунку.

Заварзін Олександр, група КП-12



Дерево проблем потенційних користувачів.

Заварзін Олександр, група КП-12

Додаток 2

Лістинг програми

SessionContext.tsx

```
const LOCATION_TRACKING = "background-location-tracking";
const TIME_UPDATE_INTERVAL = 1000;
const DISTANCE_UPDATE_INTERVAL = 10;

declare global {
  interface Window {
    sessionUpdateCallback?: (location: Location.LocationObject) => void;
  }
}

export interface SessionData {
  startTime: number;
  distance: number;
  pace: number;
  lastLocation?: Location.LocationObject | null;
}

interface FitnessData {
  sessionData: SessionData;
  paceHistory: number[];
  currentKilometerPaces: number[];
}

interface SessionContextType {
  sessionData: SessionData | null;
  sessionStarted: boolean;
  startSession: () => Promise<void>;
  endSession: () => Promise<void>;
  getAveragePace: () => number;
  currentOptionIndex: number;
  setCurrentOptionIndex: (index: number) => void;
}

const SessionContext = createContext<SessionContextType | undefined>(undefined);

const handleLocationUpdate = (
  location: Location.LocationObject,
  prevData: FitnessData
): FitnessData => {
  const paceHistory = [...prevData.paceHistory];
  let currentKilometerPaces = [...prevData.currentKilometerPaces];

  const travelledDistance = prevData.sessionData.lastLocation
    ? calculateDistance(
        prevData.sessionData.lastLocation.coords,
        location.coords
      )
    : 0;
  const newDistance = prevData.sessionData.distance + travelledDistance;

  const timeElapsed =
    location.timestamp -
    (prevData.sessionData.lastLocation?.timestamp ??
      prevData.sessionData.startTime);
  const currentPace = calculatePace(travelledDistance, timeElapsed);

  const currentKilometer = Math.floor(newDistance / 1000);
  const lastKilometer = prevData.sessionData.lastLocation
    ? Math.floor(prevData.sessionData.distance / 1000)
    : currentKilometer;
```

```
  if (currentKilometer !== lastKilometer) {
    paceHistory.push(calculateAveragePace(currentKilometerPaces));
    currentKilometerPaces = [currentPace];
  } else {
    currentKilometerPaces.push(currentPace);
  }

  return {
    sessionData: {
      ...prevData.sessionData,
      distance: newDistance,
      pace: calculateAveragePace(currentKilometerPaces),
      lastLocation: location,
    },
    paceHistory,
    currentKilometerPaces,
  };
};

TaskManager.defineTask(
  LOCATION_TRACKING,
  async (
    body: TaskManager.TaskManagerTaskBody<{
      locations: Location.LocationObject[];
    } | null>
  ) => {
    if (body.error) {
      console.error(body.error);
      return;
    }
    if (body.data?.locations[0]) {
      window.sessionUpdateCallback?.(body.data.locations[0]);
    }
  }
);

const createInitialFitnessData = (): FitnessData => ({
  sessionData: {
    startTime: Date.now(),
    distance: 0,
    pace: 0,
    lastLocation: null,
  },
  paceHistory: [],
  currentKilometerPaces: [],
});

const startLocationUpdates = async () => {
  await Location.startLocationUpdatesAsync(LOCATION_TRACKING, {
    accuracy: Location.Accuracy.High,
    timeInterval: TIME_UPDATE_INTERVAL,
    distanceInterval: DISTANCE_UPDATE_INTERVAL,
    pausesUpdatesAutomatically: false,
    activityType: Location.ActivityType.Fitness,
  });
};

export function SessionProvider({ children }: { children: ReactNode }) {
  const [fitnessData, setFitnessData] = useState<FitnessData | null>(null);
  const [currentOptionIndex, setCurrentOptionIndex] = useState<number>(-1);
  const [sessionStarted, setSessionStarted] = useState<boolean>(false);
  const { getToken } = useStreamingServiceToken();
```

```
  useEffect(() => {
    window.sessionUpdateCallback = (location: Location.LocationObject) => {
      setFitnessData((prev) => {
        if (!prev) return null;
        return handleLocationUpdate(location, prev);
      });
    };

    return () => {
      delete window.sessionUpdateCallback;
    };
  }, []);

  const startSession = async () => {
    setFitnessData(createInitialFitnessData());
    setCurrentOptionIndex(-1);
    await startLocationUpdates();
    setSessionStarted(true);
  };

  const endSession = async () => {
    try {
      await Location.stopLocationUpdatesAsync(LOCATION_TRACKING);
      const accessToken = await getToken();
      if (accessToken) {
        await pausePlayback(accessToken);
      }
      setSessionStarted(false);
    } catch (error) {
      console.error("Error stopping location updates:", error);
    }
  };

  const getAveragePace = () => {
    if (!fitnessData?.currentKilometerPaces) return 0;
    const allPaces = [
      ...fitnessData.currentKilometerPaces,
      calculateAveragePace(fitnessData.currentKilometerPaces),
    ];
    return calculateAveragePace(allPaces);
  };

  return (
    <SessionContext.Provider
      value={{
        sessionData: fitnessData?.sessionData ?? null,
        startSession,
        endSession,
        getAveragePace,
        currentOptionIndex,
        setCurrentOptionIndex,
        sessionStarted,
      }}
    >
      {children}
    </SessionContext.Provider>
  );
}

export function useSession() {
  const context = useContext(SessionContext);
  if (!context) {
```

```
    throw new Error("useSession must be used within a SessionProvider");
  }
  return context;
}
```

useAlgorithm.ts

```
const INTERVAL_DELAY = 1000;

export function useAlgorithm() {
  const [currentTempo, setCurrentTempo] = useState<MusicTempo | null>(null);
  const intervalRef = useRef<NodeJS.Timeout | null>(null);

  const { sessionData, currentOptionIndex, setCurrentOptionIndex } =
    useSession();
  const totalOptions = useSelector(getSelectedOptionsLength);
  const mode = useSelector(getSelectedMode);

  const { getToken } = useStreamingServiceToken();

  const activePlaylist = useSelector(getActivatedPlaylist);
  const activeTrackIds =
    activePlaylist?.tracks
      ?.filter((track) => track.active)
      .map((track) => track.id) ?? [];

  const tracks = useSelector(getTracksByIds(activeTrackIds));
  const tracksByTempo = groupBy(tracks, (track) => track.info.tempo);

  const nextOptionIndex = Math.min(currentOptionIndex + 1, totalOptions - 1);
  const nextOption = useSelector(getSelectedOptionByIndex(nextOptionIndex));
  const convert = useMemo(() => convertDictionary[mode], [mode]);

  const updateTempoAndTrack = async (tempo: MusicTempo) => {
    setCurrentTempo(tempo);
    setCurrentOptionIndex(nextOptionIndex);
    if (tempo === currentTempo) return;

    const token = await getToken();
    if (!token) return;

    const trackQueue = randomizeArray(
      tracksByTempo[tempo]?.map((track) => track.id) ?? activeTrackIds
    );
    await setPlayback(token, trackQueue);
  };

  useEffect(() => {
    const isLastOption = currentOptionIndex === totalOptions - 1;
    if (isLastOption && mode !== Mode.PACE) {
      if (intervalRef.current) {
        clearInterval(intervalRef.current);
        intervalRef.current = null;
      }
      return;
    }

    const checkAndUpdateTempo = () => {
      if (!sessionData) return;

      const tempo = convert(sessionData, nextOption);
      if (tempo) {
        updateTempoAndTrack(tempo);
```

```
      }
    };

    intervalRef.current = setInterval(checkAndUpdateTempo, INTERVAL_DELAY);

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

  return {
    currentOptionIndex,
    currentTempo,
    totalOptions,
    mode,
  };
}
```

Algorithm.ts

```
const RADIUS_METERS = 100;

type ConvertFunction = (
  data: SessionData,
  option: ModeOptionType
) => MusicTempo | null;

const convertLength: ConvertFunction = (data, option) => {
  const distance = data.distance;
  const optionDistance = parseFloat(option.indicator) * 1000;

  return distance >= optionDistance ? option.musicTempo : null;
};

const convertTimer: ConvertFunction = (data, option) => {
  const time = (Date.now() - data.startTime) / 1000;
  const [minutes, seconds] = option.indicator.split(":").map(Number);
  const optionTime = minutes * 60 + seconds;

  return time >= optionTime ? option.musicTempo : null;
};

const convertPosition: ConvertFunction = (data, option) => {
  const [latitude, longitude] = option.indicator.split(",").map(parseFloat);
  const optionPosition = { latitude, longitude };

  const isDefaultPosition = latitude === 0 && longitude === 0;
  if (isDefaultPosition) return option.musicTempo;

  if (!data.lastLocation) return null;

  const currentPosition = {
    latitude: data.lastLocation.coords.latitude,
    longitude: data.lastLocation.coords.longitude,
  };

  const distanceToTarget = calculateDistance(currentPosition, optionPosition);

  return distanceToTarget <= RADIUS_METERS ? option.musicTempo : null;
};
```

```
const convertPace: ConvertFunction = (data, option) => {
  const [min, max] = option.indicator.split(":").map(Number);
  const pace = data.pace;
  if (!pace) return MusicTempo.MEDIUM;

  if (pace <= min) return MusicTempo.HIGH;
  else if (pace <= max) return MusicTempo.MEDIUM;
  else return MusicTempo.LOW;
};

export const convertDictionary: Record<Mode, ConvertFunction> = {
  [Mode.PACE]: convertPace,
  [Mode.MAP]: convertPosition,
  [Mode.LENGTH]: convertLength,
  [Mode.TIMER]: convertTimer,
};
```

firestoreService.ts

```
import { Mode, ModeOptionType, ModeType, MusicTempo } from "@/state/mode/types";
import { PlaylistType, TrackType } from "@/state/playlists/types";
import { StreamingServiceCredentialsType } from "@/state/streaming/types";
import { UserModeType, UserStreamingType } from "@/types/firebase";
import firestore from "@react-native-firebase/firestore";

// User mode settings
export async function getUserModeInfo(userId: string) {
  const modes = await firestore().collection("user_mode").doc(userId).get();
  return modes.data() as UserModeType;
}

export async function initUserModeInfo(userId: string) {
  await firestore()
    .collection("user_mode")
    .doc(userId)
    .set({
      selectedMode: Mode.TIMER,
      modsInfo: {
        [Mode.TIMER]: [
          {
            indicator: "00:00",
            musicTempo: MusicTempo.MEDIUM,
          },
        ],
        [Mode.MAP]: [
          {
            indicator: "0,0",
            musicTempo: MusicTempo.MEDIUM,
          },
        ],
        [Mode.LENGTH]: [
          {
            indicator: "0",
            musicTempo: MusicTempo.MEDIUM,
          },
        ],
        [Mode.PACE]: [
          {
            indicator: "300:360",
            musicTempo: MusicTempo.MEDIUM,
          },
        ],
```

```
      },
    });
}

export async function setUserModeInfo(
  userId: string,
  mode: Omit<ModeType, "currentOptionIndex">
) {
  await firestore().collection("user_mode").doc(userId).set(mode);
}

export async function updateSelectedUserModeInfo(userId: string, mode: Mode) {
  await firestore()
    .collection("user_mode")
    .doc(userId)
    .update({ selectedMode: mode });
}

export async function updateUserModeInfo(
  userId: string,
  mode: Mode,
  modeOption: ModeOptionType[]
) {
  const key = "modsInfo." + mode;
  await firestore()
    .collection("user_mode")
    .doc(userId)
    .update({ [key]: modeOption });
}

// User Streaming info
export async function getUserStreamingInfo(userId: string) {
  const streamingServices = await firestore()
    .collection("user_streaming")
    .doc(userId)
    .get();
  return streamingServices.data() as UserStreamingType;
}

export async function initUserStreamingInfo(userId: string) {
  await firestore().collection("user_streaming").doc(userId).set({});
}

export async function setUserStreamingInfo(
  userId: string,
  service: StreamingServiceCredentialsType
) {
  await firestore().collection("user_streaming").doc(userId).set(service);
}

export async function resetUserStreamingInfo(userId: string) {
  await firestore().collection("user_streaming").doc(userId).set({});
}

// Playlists
export async function getUserPlaylistInfo(
  userId: string
): Promise<PlaylistType[]> {
  const playlists = await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
```

```
    .get();
  const res = await Promise.all(
    playlists.docs.map(async (doc) => {
      const playlist = { ...doc.data(), id: doc.id } as PlaylistType;
      const tracks = await getUserPlaylistTracks(userId, playlist.id);
      return {
        ...playlist,
        tracks,
      };
    })
  );
  return res;
}

export async function initUserPlaylistInfo(userId: string) {
  const userPlaylistsRef = firestore().collection("user_playlists").doc(userId);
  await userPlaylistsRef.set({});
  await userPlaylistsRef.collection("playlists").doc().set({});
}

export async function addUserPlaylistInfo(
  userId: string,
  playlist: PlaylistType
) {
  await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
    .doc(playlist.id)
    .set({
      active: false,
      streamingServiceId: playlist.streamingServiceId,
    });
  if (playlist.tracks) {
    setUserPlaylistTracks(userId, playlist.id, playlist.tracks);
  }
}

export async function removeUserPlaylistInfo(
  userId: string,
  playlist: PlaylistType
) {
  await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
    .doc(playlist.id)
    .collection("tracks")
    .get()
    .then((docs) => {
      docs.docs.forEach((doc) => {
        doc.ref.delete();
      });
    });
  await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
    .doc(playlist.id)
    .delete();
}
```

```
export async function resetUserPlaylistInfo(userId: string) {
  await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
    .get()
    .then((docs) => {
      docs.docs.forEach(async (doc) => {
        await doc.ref
          .collection("tracks")
          .get()
          .then((docs) => {
            docs.docs.forEach((doc) => {
              doc.ref.delete();
            });
          });
        doc.ref.delete();
      });
    });
}

export async function setUserPlaylistActive(
  userId: string,
  playlistId: string,
  active: boolean
) {
  await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
    .doc(playlistId)
    .update({ active });
}

// Playlists Tracks
export async function getUserPlaylistTracks(
  userId: string,
  playlistId: string
): Promise<TrackType[]> {
  const tracks = await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
    .doc(playlistId)
    .collection("tracks")
    .get();
  return tracks.docs.map((doc) => ({
    ...doc.data(),
    id: doc.id,
  })) as TrackType[];
}

export async function setUserPlaylistTracks(
  userId: string,
  playlistId: string,
  tracks: TrackType[]
) {
  tracks.forEach(async (track) => {
    await firestore()
      .collection("user_playlists")
      .doc(userId)
      .collection("playlists")
```

```
      .doc(playlistId)
      .collection("tracks")
      .doc(track.id)
      .set({
        active: track.active,
      });
  });
}

export async function removeUserPlaylistTracks(
  userId: string,
  playlistId: string,
  trackIds: string[]
) {
  trackIds.forEach(async (trackId) => {
    await firestore()
      .collection("user_playlists")
      .doc(userId)
      .collection("playlists")
      .doc(playlistId)
      .collection("tracks")
      .doc(trackId)
      .delete();
  });
}

export async function addUserPlaylistTracks(
  userId: string,
  playlistId: string,
  tracks: TrackType[]
) {
  tracks.forEach(async (track) => {
    await firestore()
      .collection("user_playlists")
      .doc(userId)
      .collection("playlists")
      .doc(playlistId)
      .collection("tracks")
      .doc(track.id)
      .set({
        active: track.active,
      });
  });
}

export async function toggleUserPlaylistTrackActive(
  userId: string,
  playlistId: string,
  trackId: string,
  active: boolean
) {
  await firestore()
    .collection("user_playlists")
    .doc(userId)
    .collection("playlists")
    .doc(playlistId)
    .collection("tracks")
    .doc(trackId)
    .update({ active });
}

export async function deleteUserData(userId: string) {
```

```
  await resetUserPlaylistInfo(userId);
  await firestore().collection("user_playlists").doc(userId).delete();
  await firestore().collection("user_streaming").doc(userId).delete();
  await firestore().collection("user_mode").doc(userId).delete();
}
```

spotifyService.ts

```
import { PlaylistType, TrackType } from "@/state/playlists/types";
import {
  deviceInfoType,
  DeviceType,
  playlistInfoType,
  playlistTrackInfo,
  trackInfoType,
} from "@/types/spotifyAPI";
import { Spotify } from "@/constants/API";
import { Images } from "@/constants/Images";
import { StreamingServiceCredentialsType } from "@/state/streaming/types";
import { refreshAsync } from "expo-auth-session";

export async function getSpotifyUserProfile(accessToken: string) {
  try {
    const response = await fetch(`${Spotify.url.apiV1Endpoint}/me`, {
      headers: {
        Authorization: `Bearer ${accessToken}`,
      },
    });
    const userInfo = await response.json();

    return {
      id: userInfo.id as string,
      info: {
        name: userInfo.display_name as string,
        profileImage: (userInfo.images[0]
          ? userInfo.images[0].url
          : Images.notFound.image) as string,
      },
    };
  } catch (error) {
    console.error("Failed to fetch user info:", error);
    return null;
  }
}

export async function getSpotifyUserPlaylists(
  accessToken: string,
  userId: string
): Promise<PlaylistType[] | null> {
  try {
    const response = await fetch(`${Spotify.url.apiV1Endpoint}/me/playlists`, {
      headers: {
        Authorization: `Bearer ${accessToken}`,
      },
    });
    const playlistsInfo = (await response.json()).items;

    return playlistsInfo.map((playlistInfo: playlistInfoType) => {
      return {
        id: playlistInfo.id,
        streamingServiceId: userId,
        info: {
          name: playlistInfo.name,
```

```
          imageUrl: playlistInfo.images[0]
            ? playlistInfo.images[0].url
            : Images.notFound.image,
        },
        active: false,
        imported: false,
      };
    });
  } catch (error) {
    console.error("Failed to fetch user playlists:", error);
    return null;
  }
}

export async function getSpotifyPlaylistTracks(
  accessToken: string,
  paylistId: string
): Promise<TrackType[] | null> {
  try {
    const response = await fetch(

`${Spotify.url.apiV1Endpoint}/playlists/${paylistId}/tracks?fields=items%28track%2
8id%2Cduration_ms%29%29`,
      {
        headers: {
          Authorization: `Bearer ${accessToken}`,
        },
      }
    );
    const playlistsTracks: Array<playlistTrackInfo> = (await response.json())
      .items;

    return playlistsTracks
      .filter(({ track }) => track.duration_ms > 0)
      .map(({ track }) => {
        return {
          id: track.id,
          active: true,
        };
      });
  } catch (error) {
    console.error("Failed to fetch playlist's tracks:", error);
    return null;
  }
}

export async function getSpotifyTrackInfo(
  accessToken: string,
  trackId: string
) {
  try {
    const response = await fetch(
      `${Spotify.url.apiV1Endpoint}/tracks/${trackId}`,
      {
        headers: {
          Authorization: `Bearer ${accessToken}`,
        },
      }
    );
    const trackInfo: trackInfoType = await response.json();

    return {
```

```
      id: trackId,
      info: {
        name: trackInfo.name,
        artist: trackInfo.artists.map((v) => v.name).join(`, `),
        imageUrl: trackInfo.album.images[0]
          ? trackInfo.album.images[0].url
          : Images.notFound.image,
      },
    };
  } catch (error) {
    console.error("Failed to fetch track info:", error);
    return null;
  }
}

export async function getSpotifyTracksInfo(
  accessToken: string,
  trackIds: Array<string>
) {
  try {
    const ids = trackIds.join(`,`);
    const response = await fetch(
      `${Spotify.url.apiV1Endpoint}/tracks?ids=${ids}`,
      {
        headers: {
          Authorization: `Bearer ${accessToken}`,
        },
      }
    );
    const tracksInfo: Array<trackInfoType> = (await response.json()).tracks;
    return tracksInfo.map((trackInfo) => ({
      id: trackInfo.id,
      info: {
        name: trackInfo.name,
        artist: trackInfo.artists.map((v) => v.name).join(`, `),
        imageUrl: trackInfo.album.images[0]
          ? trackInfo.album.images[0].url
          : Images.notFound.image,
      },
    }));
  } catch (error) {
    console.error("Failed to fetch track info:", error);
    return null;
  }
}

export async function getRefreshedToken(refreshToken: string) {
  try {
    const tokenResult = await refreshAsync(
      {
        clientId: Spotify.env.clientId,
        refreshToken: refreshToken,
      },
      {
        tokenEndpoint: Spotify.url.tokenEndpoint,
      }
    );
    return {
      accessToken: tokenResult.accessToken,
      expiresIn: tokenResult.expiresIn ?? 3600,
      issuedAt: tokenResult.issuedAt,
      refreshToken: tokenResult.refreshToken ?? refreshToken,
```

```
    };
  } catch (error) {
    console.error("Failed to refresh user token:", error);
    throw error;
  }
}

export async function getValidSpotifyToken(
  credentials: StreamingServiceCredentialsType
): Promise<StreamingServiceCredentialsType> {
  const { refreshToken, expiresIn, issuedAt } = credentials;

  const lastRefreshedDate = new Date(issuedAt);
  const now = Date.now() / 1000;
  if (now - lastRefreshedDate.getTime() > expiresIn - 600) {
    const newCreds = await getRefreshedToken(refreshToken);
    return {
      ...credentials,
      ...newCreds,
    };
  }

  return credentials;
}

export async function getAvailableDevices(
  accessToken: string
): Promise<deviceInfoType[] | null> {
  try {
    const response = await fetch(
      `${Spotify.url.apiV1Endpoint}/me/player/devices`,
      {
        headers: {
          Authorization: `Bearer ${accessToken}`,
        },
      }
    );

    if (!response.ok) {
      throw new Error(`Failed to fetch devices: ${response.status}`);
    }

    const { devices = [] } = await response.json();
    return devices.filter(
      (device: deviceInfoType) => device.type === DeviceType.Smartphone
    );
  } catch (error) {
    console.error("Failed to get available devices:", error);
    return null;
  }
}

export async function setPlayback(
  accessToken: string,
  trackIds: Array<string>,
  position = 0,
  positionMs = 0
) {
  try {
    const devices = await getAvailableDevices(accessToken);
    const deviceId = devices?.[0]?.id;
    if (!deviceId) {
```

```
      throw new Error("No device found");
    }

    const playbackOptions = {
      uris: trackIds.map((id) => `spotify:track:${id}`),
      offset: { position },
      position_ms: positionMs,
    };

    const response = await fetch(
      `${Spotify.url.apiV1Endpoint}/me/player/play?device_id=${deviceId}`,
      {
        method: "PUT",
        headers: {
          Authorization: `Bearer ${accessToken}`,
          "Content-Type": "application/json",
        },
        body: JSON.stringify(playbackOptions),
      }
    );

    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData);
    }

    return true;
  } catch (error) {
    console.error("Failed to start playback:", error);
    return null;
  }
}

export async function pausePlayback(
  accessToken: string
): Promise<boolean | null> {
  try {
    const devices = await getAvailableDevices(accessToken);
    const deviceId = devices?.[0]?.id;
    if (!deviceId) {
      throw new Error("No device found");
    }
    const response = await fetch(
      `${Spotify.url.apiV1Endpoint}/me/player/pause?device_id=${deviceId}`,
      {
        method: "PUT",
        headers: {
          Authorization: `Bearer ${accessToken}`,
          "Content-Type": "application/json",
        },
      }
    );
    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData);
    }
    return true;
  } catch (error) {
    console.error("Failed to pause playback:", error);
    return null;
  }
}
```

Додаток 3

Таблиці сценаріїв використання

Таблиця 3.1

UC-1: Реєстрація користувача

UC-1	Реєстрація користувача
Актори	Незареєстрований користувач сервісу
Початкові умови	Доступ до Інтернету
Очікуваний результат	Створити обліковий запис у TempoRun
Основний потік	1. Користувач відкриває застосунок. 2. Система відображає форму для входу в систему, яка містить поля вводу електронної адреси і паролю, кнопок «Sign Up» і «Log In», а також кнопки «Log in with Google». 3. Користувач вводить дані у відповідні поля. 4. Користувач натискає на кнопку «Sign up». 5. Система перевіряє введені дані. 5a. Одне або декілька полів порожні. 5b. Формат пошти некоректний. 5c. Формат паролю некоректний. 5d. Пошта вже була зареєстрована. 7. Відбувається автоматичний вхід в систему.
Альтернативні потоки	5a. Одне або декілька полів порожні. 5a-1. Система виділяє червоним рамки пропущених полів вводу і відображає помилку «Email is required» і «Password is required» відповідно до пропущених полів. 5a-2. Користувач заповнює пропущені поля. 5a-3. Повернення до кроку 4 основного потоку.
	5b. Формат пошти некоректний. 5b-1. Система виділяє рамки поля вводу електронної пошти червоним і відображає помилку «Invalid email address». 5b-2. Користувач повторно заповнює поле електронної пошти. 5b-3. Повернення до кроку 4 основного потоку.
	5c. Формат паролю некоректний. 5c-1. Система виділяє рамки поля вводу паролю червоним і відображає помилку «Password must be at least 6 characters long».

Продовження табл. 3.1

Альтернативні потоки	5c-2. Користувач повторно заповнює поле паролю. 5c-3. Повернення до кроку 4 основного потоку.
	5d. Пошта вже була зареєстрована. 5d-1. Система виділяє рамки поля вводу електронної пошти червоним і відображає помилку «Email already in use». 5d-2. Користувач повторно заповнює поле електронної пошти. 5d-3. Повернення до кроку 4 основного потоку.

Таблиця 3.2

UC-2: Авторизація користувача

UC-2	Авторизація користувача
Актори	Зареєстрований користувач сервісу
Початкові умови	Доступ до Інтернету
Очікуваний результат	Вхід в систему TempoRun
Основний потік	1. Користувач відкриває застосунок. 2. Система відображає форму для входу в систему, яка містить поля вводу електронної адреси і паролю, кнопок «Sign Up» і «Log In», а також кнопки «Log in with Google». 3. Користувач вводить дані у відповідні поля. 4. Користувач натискає на кнопку «Log In». 5. Система перевіряє введені дані. 5a. Одне або декілька полів порожні. 5b. Формат пошти некоректний. 5c. Формат паролю некоректний. 5d. Пошта ще не була зареєстрована. 5e. Пароль не відповідає паролю, з яким користувач реєструвався. 6. Відбувається автоматичний вхід і система виводить на екран головне меню застосунку.

Продовження табл. 3.2

Альтернативні потоки	5a. Одне або декілька полів порожні. 5a-1. Система виділяє червоним рамки пропущених полів і відображає помилку «Email is required» і «Password is required» відповідно до пропущених полів. 5a-2. Користувач заповнює пропущені поля. 5a-3. Повернення до кроку 4 основного потоку.
	5b. Формат пошти некоректний. 5b-1. Система виділяє рамки поля електронної пошти червоним і відображає помилку «Invalid email address». 5b-2. Користувач повторно заповнює поле електронної пошти. 5b-3. Повернення до кроку 4 основного потоку.
	5c. Формат паролю некоректний. 5c-1. Система виділяє рамки поля вводу паролю червоним і відображає помилку «Password must be at least 6 characters long». 5c-2. Користувач повторно заповнює поле паролю. 5c-3. Повернення до кроку 4 основного потоку.
	5d. Пошта ще не була зареєстрована. 5d-1. Система виділяє рамки полів вводу електронної пошти і паролю червоним і відображає помилку «Invalid email or password». 5d-2. Користувач повторно заповнює поле електронної пошти. 5d-3. Повернення до кроку 4 основного потоку.
	5e. Пароль не відповідає паролю, з яким користувач реєструвався. 5e-1. Система виділяє рамки поля вводу електронної пошти і паролю червоним і відображає помилку «Invalid email or password». 5e-2. Користувач повторно заповнює поле паролю. 5e-3. Повернення до кроку 4 основного потоку.

Таблиця 3.3

UC-3: Підключення облікового запису Spotify

UC-3	Підключення облікового запису Spotify
Актори	Авторизований користувач, сервіс Spotify
Початкові умови	Інтернет-з’єднання; дійсний акаунт Spotify
Очікуваний результат	Підключити обліковий запис Spotify до системи
Основний потік	1. Користувач відкриває застосунок TempoRun. 2. Користувач відкриває вікно «Settings». 3. У налаштуваннях користувач натискає на іконку Spotify у панелі під’єднання музичного сервісу. 4. Система ініціює OAuth-процес з Spotify. 5. Spotify відображає форму входу. 6. Користувач вводить свої дані. 7. Spotify перевіряє правильність введених даних на своїй стороні. 8. Користувач підтверджує доступ. 9. Система зберігає токен і повертає користувача назад у вікно з налаштуваннями. 10. Система відображає імпортований профіль Spotify у панелі з імпортованими музичними сервісами.

Таблиця 3.4

UC-4: Вибір плейлиста

UC-4	Реєстрація користувача
Актори	Авторизований користувач
Початкові умови	Доступ до Інтернету, користувач підключив власний профіль Spotify, і імпортував плейлист
Очікуваний результат	Встановити джерело для підбору музики під час тренування
Основний потік	1. Користувач відкриває застосунок TempoRun. 2. Користувач відкриває вікно «Playlists».

Продовження табл. 3.4

Основний потік	3. Система відображає список імпортованих плейлистів користувача, у вигляді карточок з їх іконками і назвами. 4. Користувач натискає на карточку обраного плейлиста. 5. Система перевіряє, чи є обраний плейлист. 5a. Система вже має інший обраний плейлист. 5b. Система вже має цей плейлист обраним. 6. Обраний плейлист набуває статусу обраного і система змінює колір його інформаційної карточки.
Альтернативні потоки	5a. Система вже має інший обраний плейлист. 5a-1. Система прибирає статус обраного у попереднього плейлиста і змінює його колір інформаційної карточки. 5a-2. Перехід до наступного 6 кроку.
	5b. Система вже має цей плейлист обраним. 5b-1. Система прибрає статус обраного у цього плейлиста і змінює його колір інформаційної карточки.

Таблиця 3.5

UC-5: Налаштування параметрів алгоритму в режимі Таймер

UC-5	Налаштування параметрів алгоритму
Актори	Авторизований користувач
Початкові умови	Доступ до Інтернету
Очікуваний результат	Задати правила підбору музики в режимі Таймер
Основний потік	1. Користувач відкриває застосунок. 2. Користувач відкриває вікно «Mode». 3. Система відображає меню налаштування режиму підбору пісень, а саме таблицю з характеристиками підбору і кнопки перемикання режимів алгоритму. 4. Користувач обирає режим «Timer».

Продовження табл. 3.5

Основний потік	5. Система відображає таблицю з характеристиками підбору під режим таймер з двома стовпчиками «Time» і «Tempo». 6. Користувач натискає на кнопку «+ Add Option». 7. Система створює новий рядок в таблиці характеристик з порожнім значенням в стовпчику «Time» і зі значенням за замовчуванням Medium в стовпчику «Tempo». 8. Користувач натискає на порожнє місце в новоствореному рядку таблиці в стовпчику «Time». 9. Система відображає модальне вікно з полями вводу даних про тривалість у форматі “хвилини:секунди” а також двома кнопками «Cancel» і «OK». 10. Користувач вводить відповідні дані про потрібний йому час в секундах і хвилинах у поле вводу. 11. Користувач натискає на кнопку «OK». 12. У відповідному рядку з’являються введені дані користувача. 13. Система автоматично сортує рядки за зростанням від найменшого значення в полі «Time» до найбільшого з порожніми полями в кінці. 14. Користувач натискає на клітинку з стовпчика «Tempo» у відповідному рядку. 15. Система виводить спадне меню зі списком темпу музики на відповідний час. В списку наявні такі значення як «Low», «Medium», «High». 16. Користувач обирає значення «Low». 17. Система запам’ятовує відповідні значення характеристики і буде використовувати їх в сукупності з іншими при наступному старті сесії спорту.

Таблиця 3.6

UC-6: Початок тренування

UC-6	Початок тренування
Актори	Авторизований користувач, Spotify
Початкові умови	Доступ до Інтернет, імпортовані плейлисти, інтегрований профіль Spotify
Очікуваний результат	Розпочалася сесія тренування
Основний потік	1. Користувач відкриває застосунок TempoRun. 2. Користувач відкриває вікно «Menu». 3. У відкритому екрані «Menu» система відображає логотип застосунку, кнопку «Run» а також обраний режим алгоритму і активний плейлист. 4. Користувач натискає на кнопку «Run». 5. Система перевіряє передумови сесії бігу. 5a. Система не має обраного плейлиста. 5b. Обраний режим Таймер, Відстань або Карта, і система не має додаткових характеристик до нього, окрім стандартного на початок сесії. 5c. Користувач не надав доступ до свого місцеположення. 6. Система відкриває екран спортивної сесії. 7. Система відображає таймер, поточний темп пересування, пройдену дистанцію, інформаційну карточку треку, що програється, і кнопку зупинки. 8. Система вираховує темп першої пісні під обрані користувачем характеристики і вмикає її на стороні Spotify.
Альтернативні потоки	5a. Система не має обраного плейлиста. 5a-1. Система виводить інформаційне модальне вікно з текстом «There is no selected playlists. Please go to the Playlists menu and select your playlist» і кнопка «Ok».

Продовження табл. 3.6

Альтернативні потоки	5b. Обраний режим Таймер, Відстань або Карта, і система не має додаткових характеристик до нього, окрім стандартного на початок сесії. 5b-1. Система виводить інформаційне модальне вікно з текстом «There are no additional settings for the selected music selection mode. Please go to the Mode menu and create at least one additional option on the table» і кнопка «Ok».
	5c. Користувач не надав доступ до свого місцеположення. 5c-1. Система виводить системне повідомлення про надання доступу застосунку до місцеположення користувача.

Додаток 4

Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

МОБІЛЬНИЙ ЗАСТОСУНОК «TEMPORUN» ДЛЯ ТРЕНУВАНЬ З БІГУ

Виконав: Заварзін Олександр Олександрович

Керівник: доц. кафедри ПЗКС, к.т.н., доц. Заболотня Тетяна Миколаївна

Київ-2025

ПОСТАНОВКА ЗАДАЧІ

Мета проєкту: розробити мобільне програмне забезпечення для тренувань з бігу, що надасть користувачам можливість підбирати музичний супровід з імпортованого плейлиста у відповідності до поточних фітнес-характеристик бігу.

Завдання:

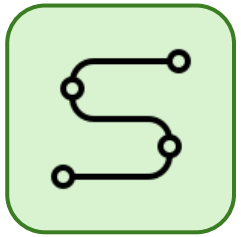
1. Проаналізувати ринок мобільних застосунків для підбору музики під час бігу
2. Виявити та порівняти конкурентні рішення, а також визначити їхні переваги і недоліки
3. Сформувати функціональні вимоги до ПЗ, яке розроблятиметься
4. Обґрунтувати вибір інструментів і технологій розроблення ПЗ
5. Реалізувати мобільне ПЗ для автоматизованого підбору музики під час бігу
6. Протестувати розроблене ПЗ та оцінити його ефективність в реальних умовах
7. Визначити можливості подальшого вдосконалення

ФІТНЕС-ХАРАКТЕРИСТИКИ

Пройдений час

Поточне місцеположення

Загальна дистанція



Темп переміщення

АКТУАЛЬНІСТЬ

Публічні опитування проведені в 2018-2019 роках виявили, що більше **60%** всіх бігунів прослуховують музику під час бігу. Більше того, **43 %** опитуваних вже роками бігають під один плейлист

Підібраний музичний супровід не відповідає поточній біговій ситуації

Підготовка послідовності музичного супроводу вручну займає багато часу

ІСНУЮЧІ РІШЕННЯ

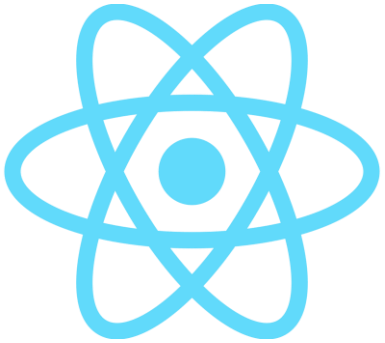
Strava

ZRX

Running Beats

ДЕРЕВО ПРОБЛЕМ

ОСНОВНІ ТЕХНОЛОГІЇ РОЗРОБЛЕННЯ



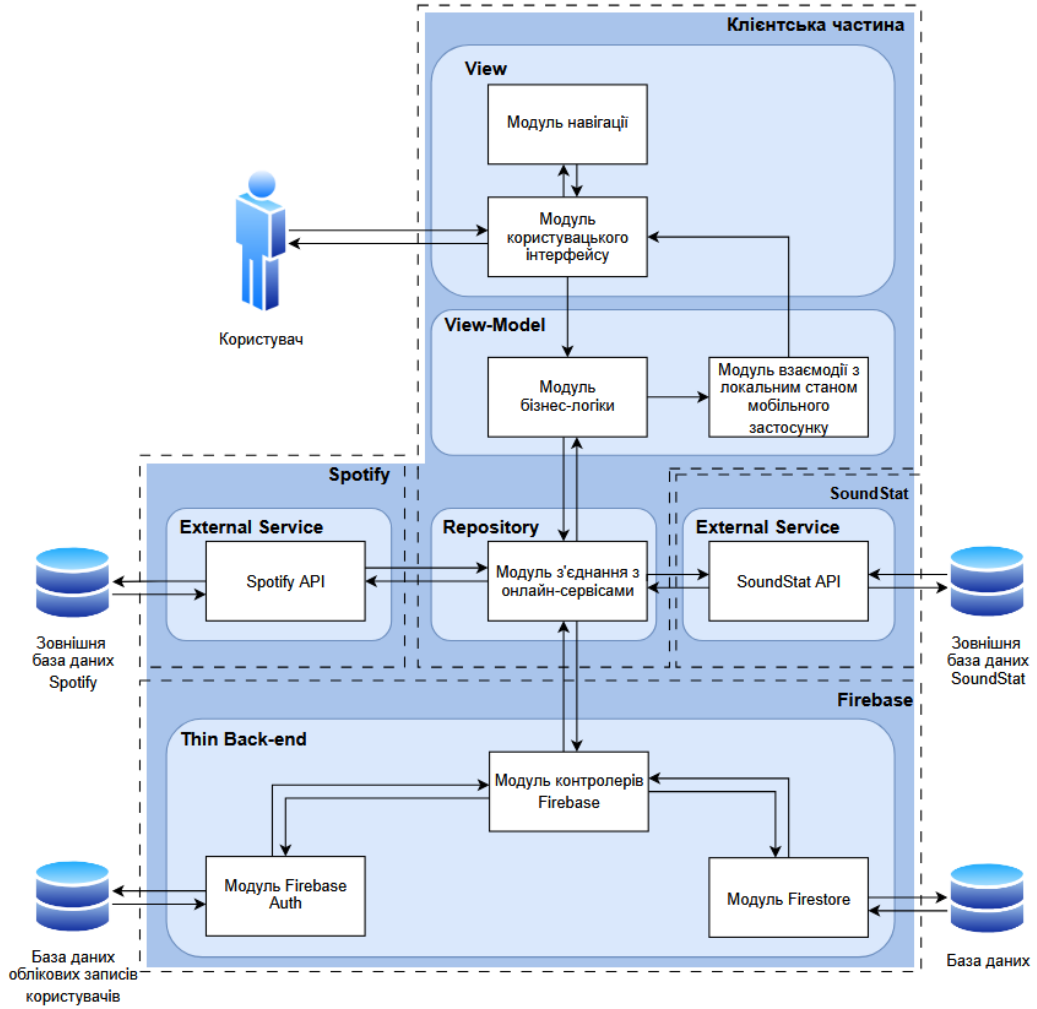
React Native

Firebase

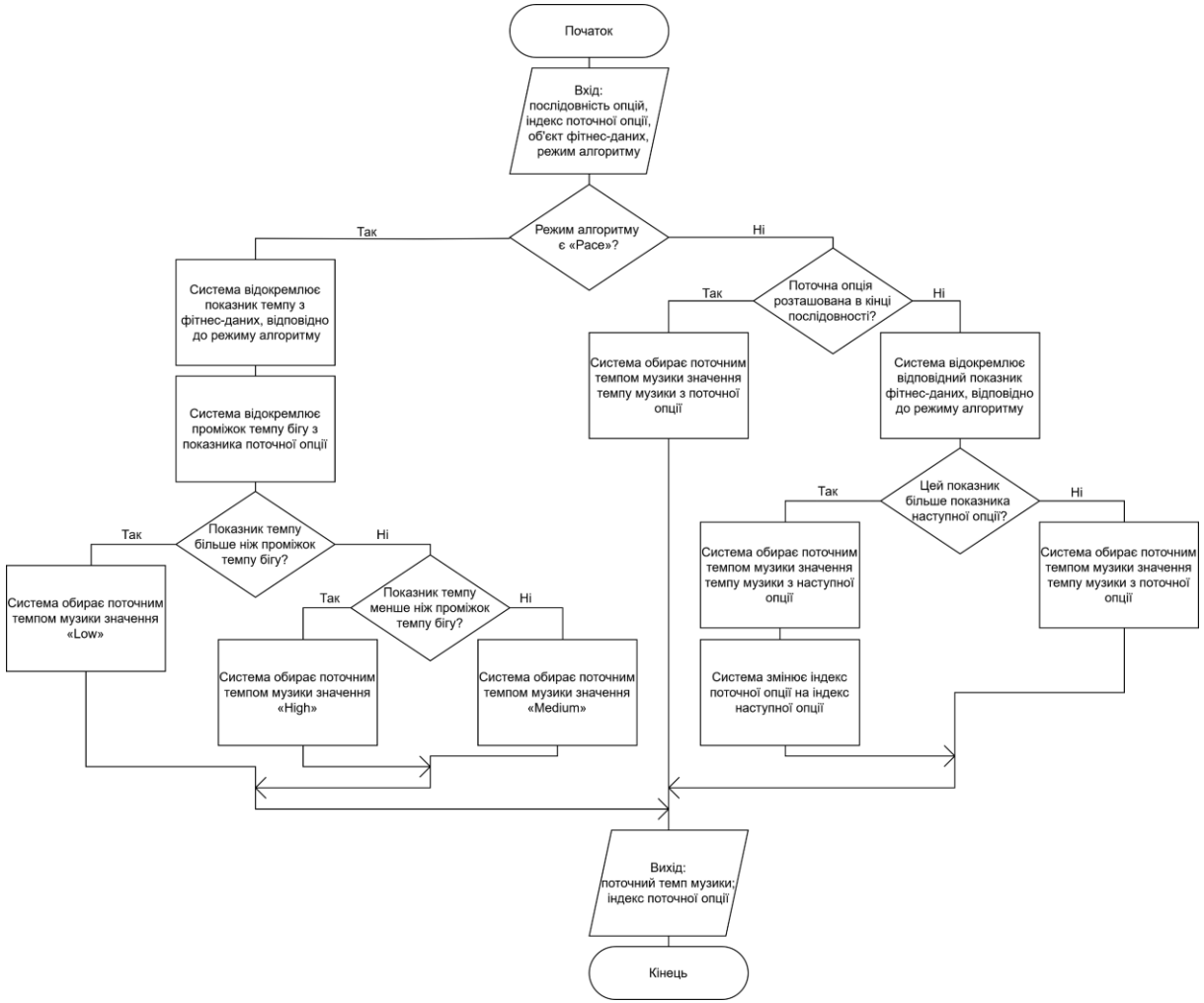
Expo

Redux

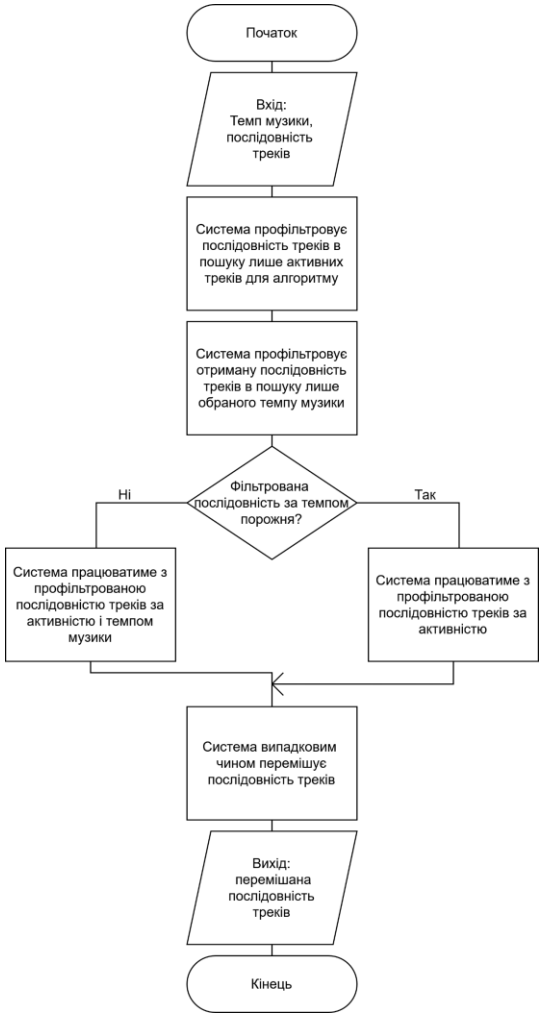
АРХІТЕКТУРА СИСТЕМИ



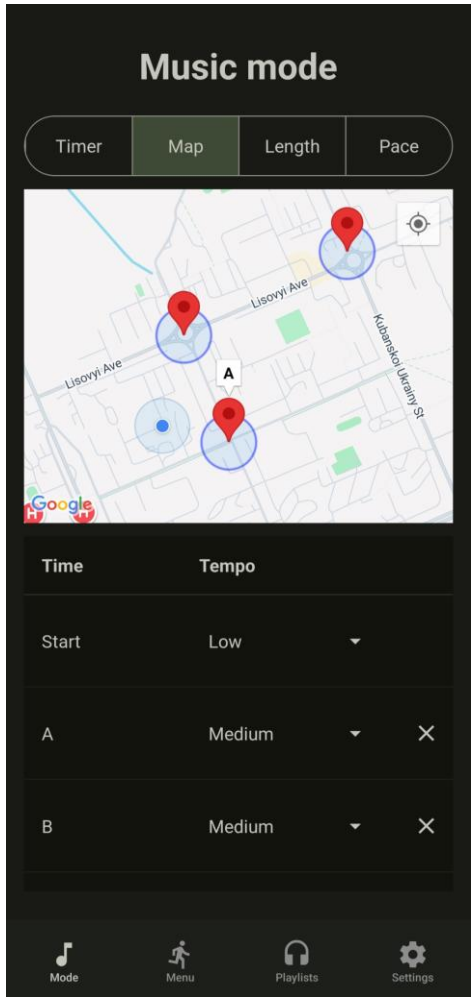
ТРАНСФОРМАЦІЯ ФІТНЕС-ДАНИХ У ЦІЛЬОВИЙ МУЗИЧНИЙ ТЕМП. БЛОК-СХЕМА АЛГОРИТМУ

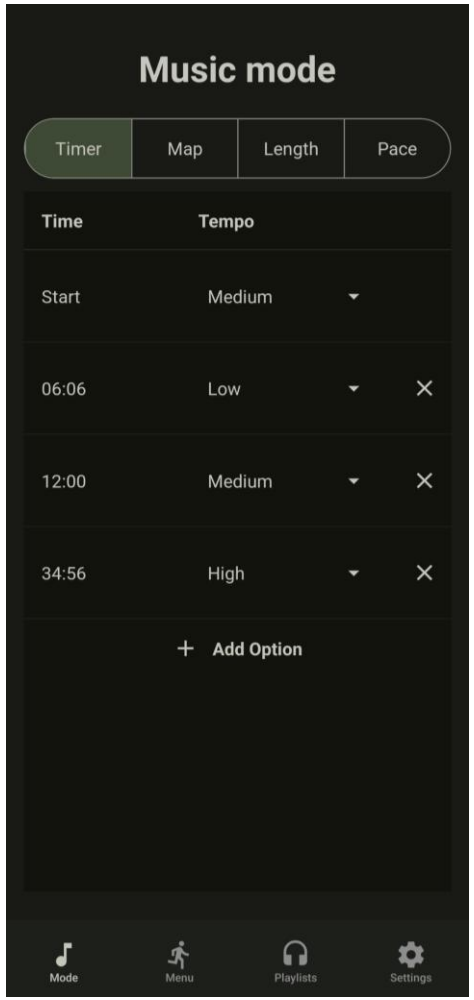


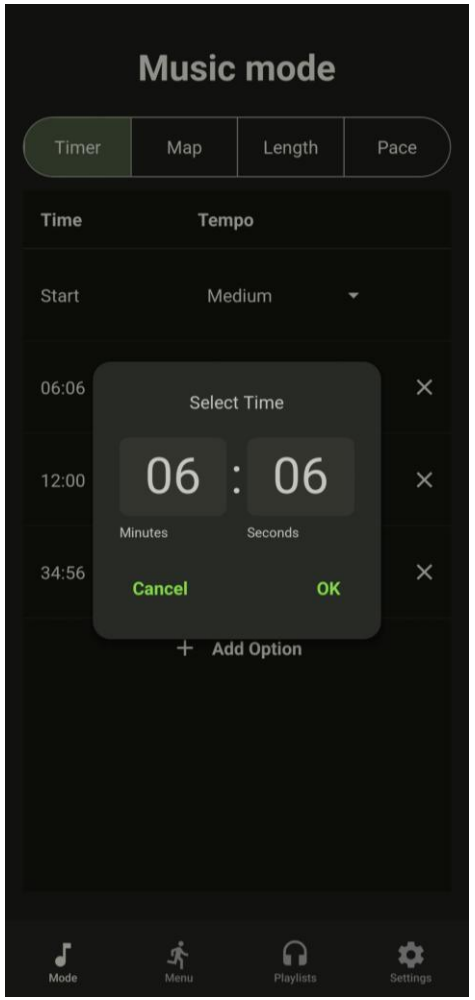
ПІДБІР МУЗИЧНОГО СУПРОВОДУ ПІД ЧАС БІГУ. БЛОК-СХЕМА АЛГОРИТМУ

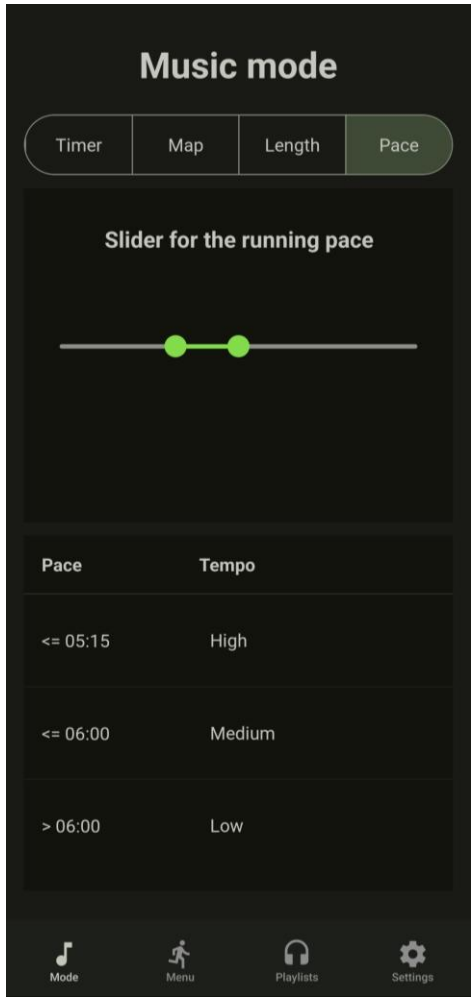


ЕКРАН НАЛАШТУВАННЯ АЛГОРИТМУ

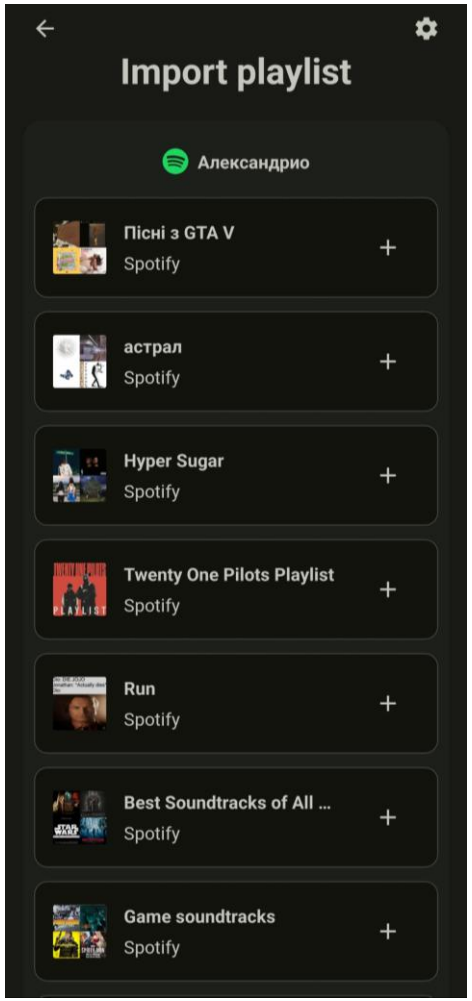


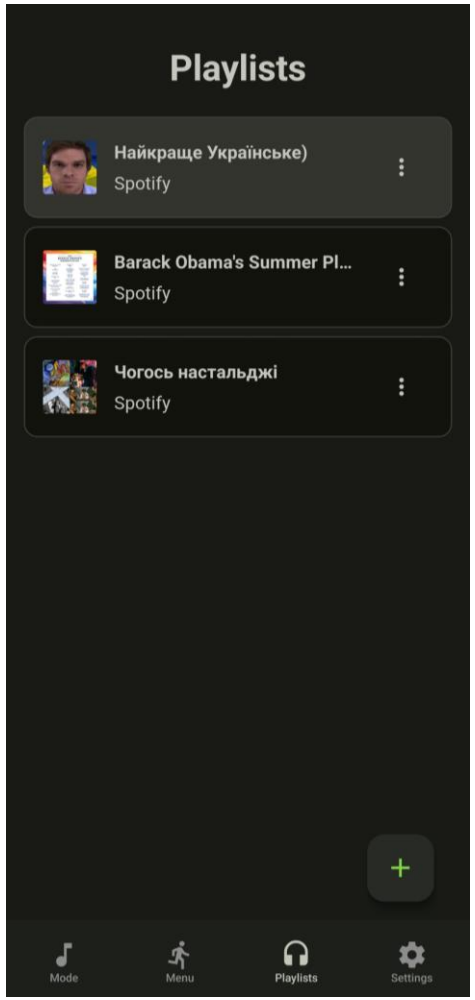


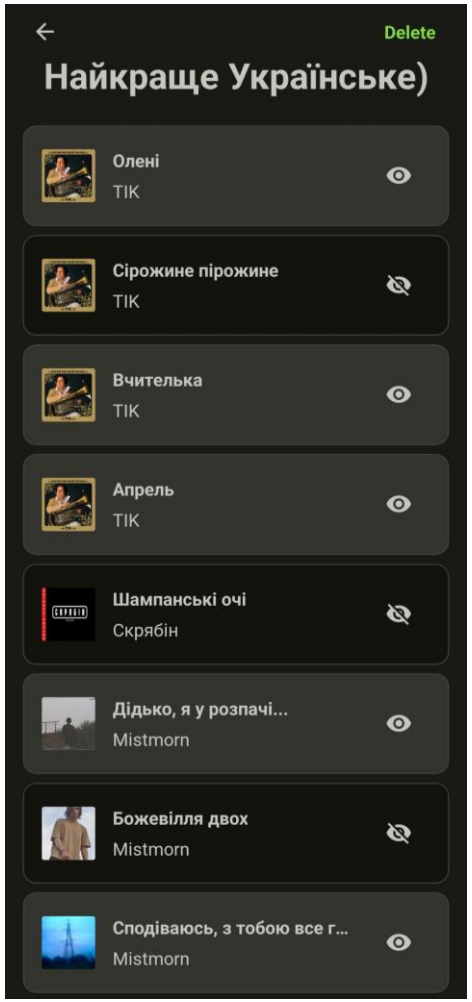




ЕКРАНИ НАЛАШТУВАННЯ ПЛЕЙЛИСТІВ







ОСНОВНІ ЗАСОБИ ОЦІНЮВАННЯ ЯКОСТІ РОЗРОБЛЕНОГО ПЗ

1. ручне тестування із введенням граничних даних у поля введення
2. динамічне ручне тестування на відповідність функціональним вимогам
3. статичний аналіз вихідного коду
4. тестування зручності використання
5. перевірка інтерфейсу на відповідність до рекомендацій Material UI

РЕЗУЛЬТАТИ ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПЗ ТА АНАЛОГІВ

Критерій	Strava	Running Beats	ZRX	TempoRun
Підбір музики з імпортованого особистого плейлиста	–	+	–	+
Залежність музичного супроводу від поточних фітнес-характеристик користувача	–	–	+	+
Інтеграція з особистим профілем музичного стримінгового сервісу	+	+	+/–	+
Можливість змінювати перелік треків під час бігу без втручання в профіль стримінгового сервісу	–	–	–	+
Гнучкий вибір фітнес-показників, що визначають підбір музичного супроводу	–	–	+	+
Наявність панелі зі зручним відображенням поточних характеристик користувача.	+	–	+	+
Підбір музики з імпортованого особистого плейлиста	–	+	–	+

ШЛЯХИ ПОДАЛЬШОГО РОЗВИТКУ РОЗРОБЛЮВАНОГО ПЗ

Підтримка інтеграції з іншими музичними сервісами

Історія спортивних сесій з графіками зміни показників

Система зворотного зв’язку з користувачем

Інтеграція з персональними фітнес пристроями

Удосконалення алгоритму підбору треків

ВИСНОВКИ

Досліджено предметну галузь персоналізованих спортивних застосунків

Визначено основні проблеми, що характерні для бігунів, які слухають музику під час бігу

Проаналізовано існуючі рішення та виявлено їхні обмеження щодо гнучкого керування музичним супроводом

Спроєктовано архітектуру ПЗ для вирішення основних проблем потенційних користувачів

Розроблено мобільний застосунок, що забезпечує автоматизований добір музичного супроводу під час бігу на основі фітнес показників користувача

Протестовано і порівняно розроблене ПЗ з існуючими програмними рішеннями, та визначено, що розробка повністю відповідає вимогам

Визначено шляхи подальшого розвитку розробленого ПЗ

УНІКАЛЬНІСТЬ

Звіт подібності

метадані

Назва організації
National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute

Заголовок
КП_12_Заварзін_Олександр

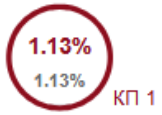
Автор
Заварзін Олександр Олександрович

Науковий керівник / Експерт
Заболотня Тетяна Миколаївна

підрозділ
ФПМ, К-ра програмного забезпечення комп'ютерних систем

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1

10	8290	66195
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

ДЯКУЮ ЗА УВАГУ!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК «TEMPORUN» ДЛЯ ТРЕНУВАНЬ З БІГУ

Програма та методика тестування

ДП.045440-04-51

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олександр ЗАВАРЗІН

2024

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Мобільний застосунок «TempoRun», створений на фреймворку React Native з використанням екосистеми Expo та інтеграції зі Spotify API.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

1) функціональна працездатність елементів мобільного застосунку;
2) можливість інтеграції особистого профілю Spotify;
3) можливість перегляду списку наявних плейлистів і імпортувати їх в систему з особистого профілю Spotify;
4) можливість налаштовувати критерії, у відповідності до яких система буде опрацьовувати фітнес-дані;
5) включення відповідно підібраного музичного супроводу під час спортивної сесії в програвачу на стороні Spotify;
6) забезпечення відповідного рівня безпеки до користувацьких даних;
7) зручність роботи з мобільним застосунком;
8) відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування мобільного застосунку виконується методом Gray Box Testing, тобто перевіряється і вихідний код, і готовий продукт на рівні системних випробувань. Застосовуються такі методи:

1) функціональне тестування у вигляді інтеграційної перевірки модулів та сценаріїв взаємодії користувача;
2) тестування продуктивності для оцінки стабільності роботи й поведінки застосунку під навантаженням;
3) тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Працездатність мобільного застосунку перевіряється такими способами:

1) динамічне ручне тестування із введенням граничних і недопустимих даних у поля введення;
2) динамічне ручне тестування на відповідність функціональним вимогам;
3) статичний аналіз вихідного коду;
4) перевірка роботи на різних версіях Android на різних розмірах екрана;
5) навантажувальне тестування при пікових запитах до Spotify та сенсорних даних;
6) тестування стабільності за умов обмеженого інтернет-зв’язку, низького заряду батареї та тривалої роботи у фоні;
7) тестування зручності використання;
8) перевірка інтерфейсу на відповідність до рекомендацій платформ.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

"ЗАТВЕРДЖЕНО"

Завідувач кафедри

_____ Євгенія СУЛЕМА

"___" _____ 2025 р.

МОБІЛЬНИЙ ЗАСТОСУНОК «TEMPORUN» ДЛЯ ТРЕНУВАНЬ З БІГУ

Керівництво користувача

ДП.045440-05-34

"ПОГОДЖЕНО"

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Олександр ЗАВАРЗІН

2025

ЗМІСТ

1. Опис структури мобільного застосунку

Користувацький інтерфейс системи для підбору музичного супроводу під час бігу реалізовано у вигляді мобільного застосунку, що містить в собі наступні екрани:

1) екран реєстрації і авторизації користувача;
2) екран головного меню;
3) екран налаштування режимів алгоритму;
4) екран імпортованих плейлистів;
5) екран налаштування імпортованого плейлиста;
6) екран імпорту плейлистів з інтегрованого профілю Spotify;
7) екран налаштувань застосунку;
8) екран налаштування акаунту;
9) екран спортивної сесії;
10) екран результатів спортивної сесії.

Користуватися застосунком можуть лише авторизовані користувачі, тому всі екрани, окрім екрану реєстрації і авторизації користувача, доступні лише авторизованим користувачам системи.

2. Процедура авторизації користувача

При першому запуску застосунку «TempoRun» користувач потрапляє на екран автентифікації (рис. 1). В залежності від того, як саме він хоче авторизуватися і чи був він зареєстрований до цього взагалі, користувач вирішує що йому зробити: авторизуватися чи зареєструватися за поштою і паролем, або зробити вхід за допомогою Google сервісів.

Якщо користувач бажає авторизуватися чи зареєструватися використовуючи пошту і пароль, він вводить свої дані у поля «Email» та «Password», після чого натискає на кнопку «Sign up» для реєстрації, або на

кнопку «Log In» для авторизації. Після цього система перевіряє коректність введених даних.

В разі некоректного вводу даних, рамки полів вводу стануть червоними і з’явиться текст помилки під відповідним полем (рис. 2).

Якщо користувач бажає зробити вхід за допомогою Google сервісів, він натискає на кнопку «Log in with Google», що відкриває діалог вибору акаунта Google та виконує OAuth-автентифікацію з обраним акаунтом.

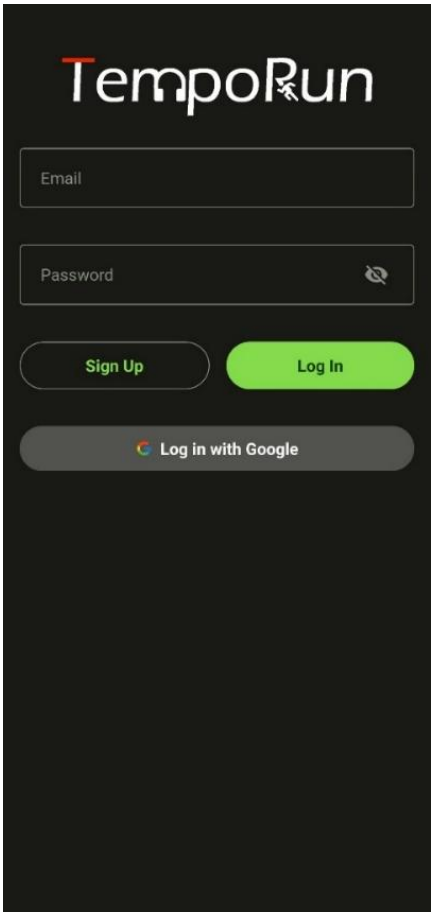


Рис. 1. Панель реєстрації і авторизації користувача

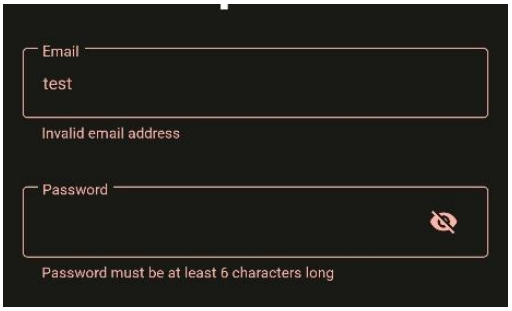


Рис. 2. Панель реєстрації і авторизації користувача з помилкою

3. Екран головного меню

При успішній авторизації користувач потрапляє на екран головного меню (рис. 3). На цьому екрані користувач зможе побачити обраний режим роботи алгоритму («Timer», «Map», «Length» і «Pace»), активний плейлист (якщо плейлист не обрано то показується повідомлення “Not selected”. При натисканні на інформаційну картку режиму, система відкриває налаштування алгоритму, а при натисканні на картку активного плейлиста відкриває перелік імпортованих плейлистів.

В низу екрану розташована панель навігації з чотирма піктограмами: режим, біг, плейлисти та налаштування, що забезпечує швидкий перехід між відповідними розділами.

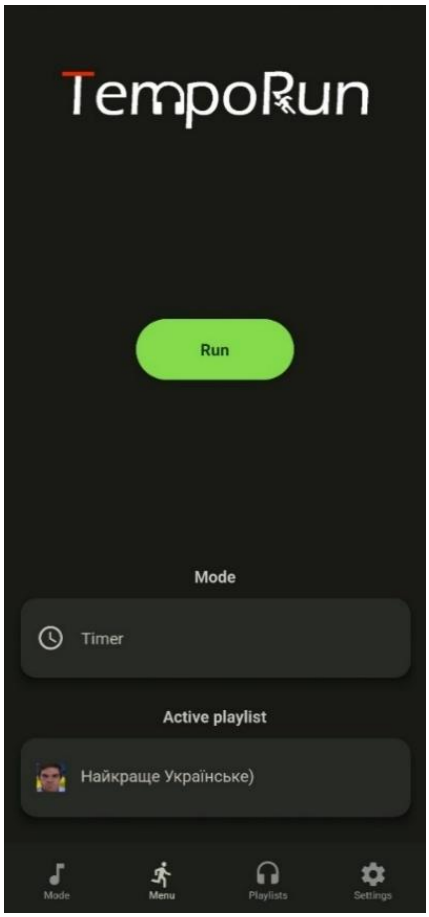


Рис. 3. Екран головного меню

По середині екрану наявна кнопка «Run». При натисканні на неї користувач може розпочати тренування. Перед початком система перевіряє щоб акаунт відповідав початку, а саме перевіряється: доступ до місцеположення користувача, наявність активного не порожнього плейлиста, наявність додаткових параметрів для обраного режиму роботи алгоритму (окрім режиму «Pace»), наявність активного програвача Spotify.

Якщо якась з цих умов не виконується, система виводить відповідне модальне вікно з помилкою (рис. 4). При виконанні всіх умов система відкриває вікно спортивної сесії.

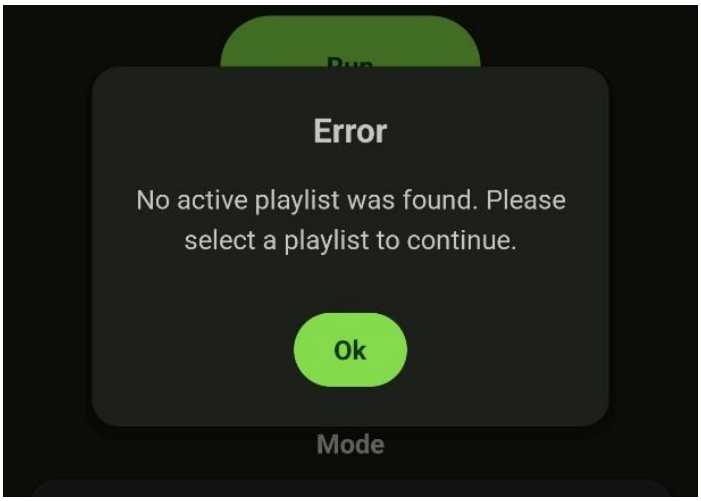


Рис. 4. Інформаційне модальне вікно з помилкою про відсутність активного плейлиста

4. Екран налаштування режимів алгоритму

Якщо користувачу потрібно налаштувати поточний режим алгоритму підбору музики, йому потрібно відкрити екран налаштування режимів алгоритму. Це він може зробити натиснувши на відповідну кнопку «Mode» на нижній панелі навігації, або натиснути на інформаційну картку активного режиму в головному меню.

При переході на екран налаштування режимів алгоритмів, користувач зможе побачити перемикач режимів – «Timer», «Map», «Length» і «Pace».

Активний режим підсвічено, а натискання на інший спонукає систему відобразити відповідну таблицю налаштувань обраного режиму.

У цій таблиці налаштувань кожен рядок містить значення обраного індикатора (час, відстань тощо), бажаний темп музики («Low»/«Medium»/«High») та, окрім стартового рядка, кнопку видалення.

Для редагування значення індикатора, користувачу потрібно натиснути на відповідний індикатор, і система відобразить модальне вікно, в якому користувач зможе змінити значення індикатора для всіх режимів: «Timer» (рис. 5), і «Length» (рис. 6). Після підтвердження зміни система автоматично сортує рядки таблиці за зростанням значення індикатору.

Для зміни темпу музики у відповідний момент, користувачу потрібно буде натиснути на темп музики на рядку обраного індикатора і система відобразить випадаюче вікно. І саме через нього користувач може обрати відповідне значення темпу музики з наявного списку («Low»/«Medium»/«High»).

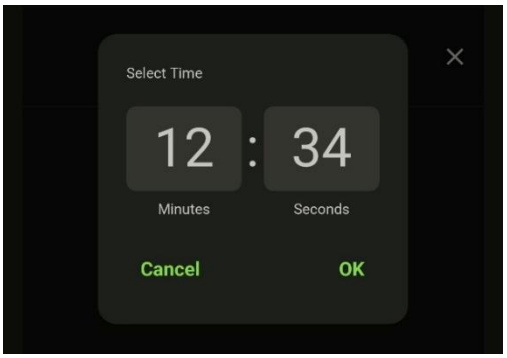


Рис. 5. Модальне вікно зміни значення індикатора в режимі «Timer»

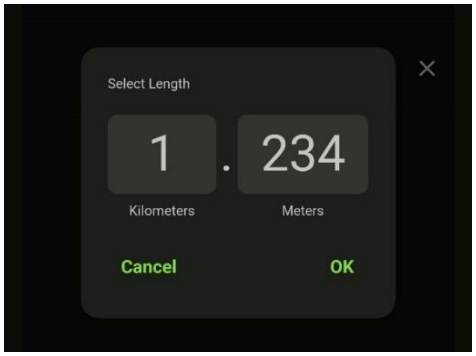


Рис. 6. Модальне вікно зміни значення індикатора в режимі «Length»

Для видалення рядку з таблиці (рис. 7), користувачу потрібно буде натиснути на кнопку видалення з іконкою хрестику, і саме після цього система видалить обрану опцію.

Для додавання нових опцій, користувачу потрібно буде натиснути на кнопку «+ Add Option». Після цього система додає новий запис, який з’являється внизу списку з порожнім значенням індикатора і обраним темпом музики «Medium».

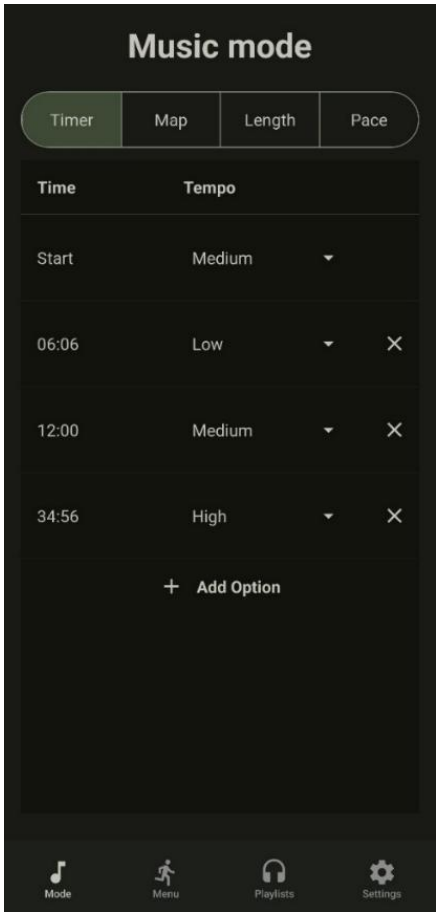


Рис. 7. Таблиця налаштування роботи алгоритму

Якщо користувач обирає активним режимом режим «Map», під перемикачем режимів з’являється вбудована Google-карта (рис. 8). На ній відображаються маркери вибраних точок із напівпрозорими радіусами: перша точка позначає поточне місце користувача (старт), тому вона не відображається на мапі, а решта точок відображаються на мапі з автоматично присвоєними літерними позначками A, B, C і тд. За бажанням,

користувач може перетягувати маркери, щоб змінити їх координати. Після натискання кнопки «+ Add Option», система додає нову точку поруч із останньою (або на позиції користувача, якщо це друга точка після старту).

Таблиця під картою працює так само, як у решті режимів: для кожної точки задається темп «Low»/«Medium»/«High», рядки можна видаляти, а стартовий рядок залишається незмінним. Єдина різниця – це розмір таблиці, так як частину екрану займає інтерактивна мапа.

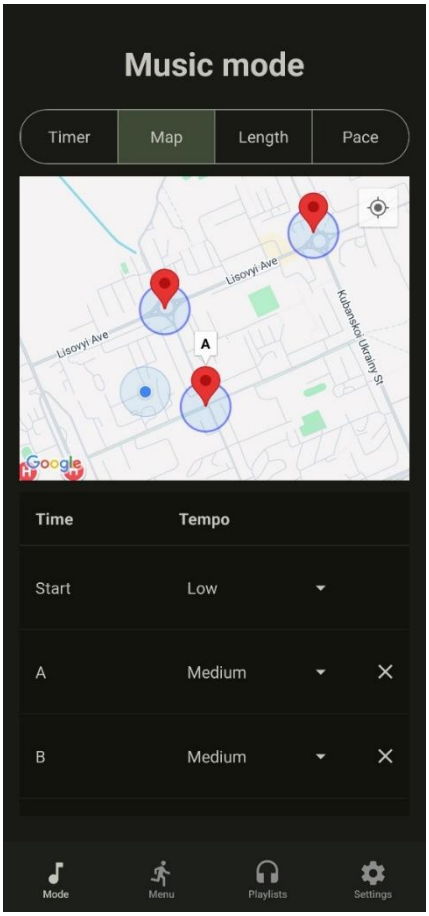


Рис. 8. Екран налаштування роботи алгоритму з обраним режимом «Map»

Якщо користувач обирає активним режимом режим «Pace», то система відображає під перемикачем режимів повзунок з двома точками і областю між ними (рис. 9). Саме завдяки переносу цих точок, користувач має можливість змінювати проміжок темпу бігу за яким система буде підбирати темп музики «Medium». Якщо користувач матиме значення темпу бігу менше ніж найменше значення проміжку (правіше на повзунку), то

система буде підбирати темпу музики «High». Якщо буде навпаки більший темп ніж найбільше значення проміжку, то система буде підбирати темп музики «Low» (лівіше на повзунку). Повзунок має максимальним значенням 10:00, а мінімальним значенням 8:00. При зміні положення однієї з точок на повзунку, система відображає поточне значення цієї точки під нею.

Нижче, під цим повзунком, розташована пояснювальна таблиця, яка показує з якими значеннями темпу система буде обирати значення темпу музики. Таблиця оновлюється з кожним перенесенням точок на повзунку.

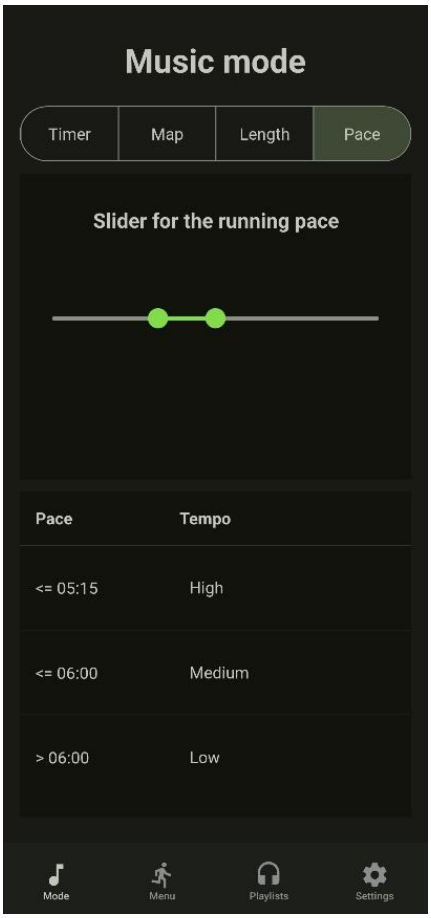


Рис. 9. Екран налаштування роботи алгоритму з обраним режимом «Pace»

5. Екран налаштування імпортованих плейлистів

Якщо користувач бажає налаштувати імпортовані плейлисти, або активувати інший плейлист, йому потрібно відкрити екран налаштування імпортованих плейлистів. Це він може зробити натиснувши на відповідну

кнопку «Playlists» на нижній панелі навігації, або натиснути на інформаційну картку активного плейлиста в головному меню.

Після відкриття екрану імпортованих плейлистів, користувач зможе побачити перелік імпортованих плейлистів із підключеного музичного сервісу (рис. 10). Кожен елемент списку містить обкладинку, назву плейлиста, джерело та кнопку з іконкою трьох крапок для переходу до його налаштувань.

Для того щоб активувати плейлист, користувачу потрібно натиснути по картці плейлиста, після чого система підсвічує його. Якщо до цього вже був активованим інший плейлист, він втрачає свій стан активності.

У правому нижньому куті розташована плаваюча кнопка (FAB) з іконкою плюса («+»), при натисканні якого система відображає користувачу екран імпорту нових плейлистів.

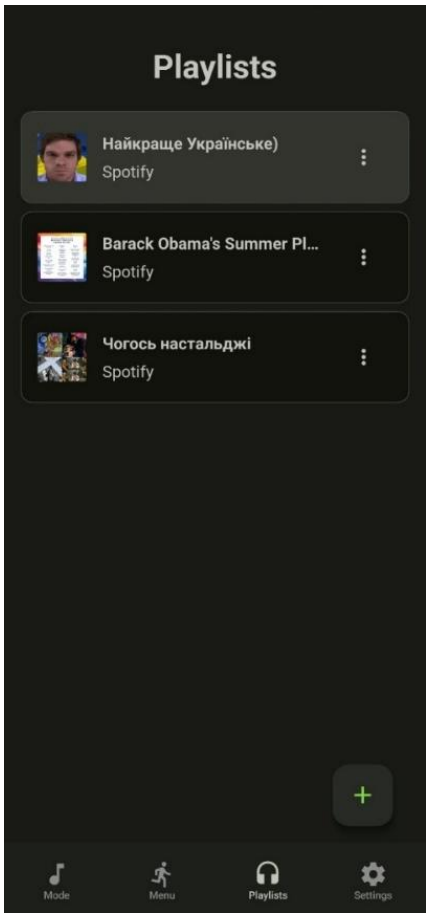


Рис. 10. Список імпортованих плейлистів

6. Екран налаштування плейлиста

Якщо користувач бажає налаштувати треки імпортованого плейлиста, або видалити плейлист зі свого профілю TempoRun, йому потрібно відкрити екран налаштування плейлиста (рис. 11). Саме після натискання користувачем на іконку трьох крапок на інформаційній карточці потрібного плейлиста, система відкриває екран налаштування цього плейлиста.

Після відкриття відповідного екрану, користувач зможе побачити верхню панель із кнопкою з іконкою стрілки («←») для повернення, назвою списку та кнопкою «Delete» для видалення плейлиста з бази даних TempoRun. Нижче відображається перелік треків: кожен рядок показує обкладинку альбому, назву композиції, виконавця та значок ока. Не перекреслене око означає, що трек враховуватиметься алгоритмом; перекреслене – буде пропущений. Щоб перемикнути стан видимості відповідний трек користувачу достатньо торкнутися карточки трека.

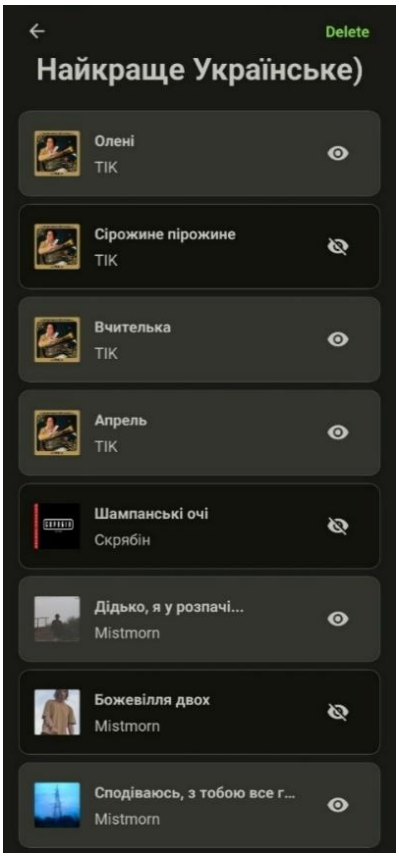


Рис. 11. Екран налаштування імпортованого плейлиста

7. Екран імпорту плейлистів

Якщо користувач бажає імпортувати плейлист інтегрованого профілю Spotify, йому потрібно відкрити екран імпорту плейлистів (рис. 12). Це він зможе зробити натиснувши на плаваючу кнопку (FAB) з іконкою плюса («+»), що розташована на екрані налаштування імпортованих плейлистів.

Після того, як система відобразить екран імпорту плейлистів, користувач зможе побачити верхню панель із кнопкою повернення та іконкою шестерні, що відкриває налаштування при натисканні на неї. Під цією верхньою панеллю розташовано заголовок «Import playlist» із логотипом «Spotify» і назвою під'єднаного профілю. Нижче виводиться перелік плейлистів, яких ще немає у TempoRun: кожна картка показує обкладинку, назву списку, джерело імпорту, та кнопку «+». Щоб запустити процес імпорту плейлиста, користувачу потрібно натиснути на кнопку з іконкою плюса («+»), після чого його інформаційна картка зникає зі списку.

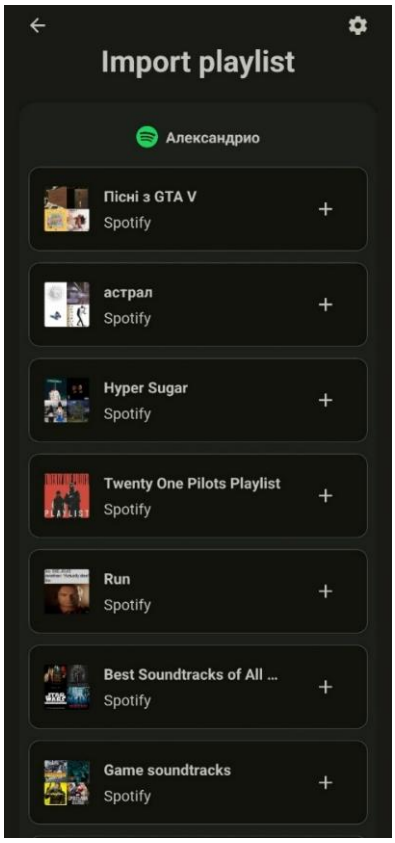


Рис. 12. Екран імпорту плейлиста

8. Екран налаштування застосунку

Якщо користувач бажає інтегрувати власний акаунт Spotify, або налаштувати застосунок, йому потрібно відкрити екран налаштування застосунку. Це він може зробити натиснувши на відповідну кнопку «Settings» на нижній панелі навігації, або натиснути на іконку шестерні з верхньої панелі екрану імпорту плейлистів.

Після відкриття екрану налаштування застосунку, система відображає картку інтеграції акаунту музичного сервісу. Якщо профіль Spotify ще не інтегровано, то система відображає лише іконка сервісу для запуску інтеграції разом з текстом «Link» (рис. 13).

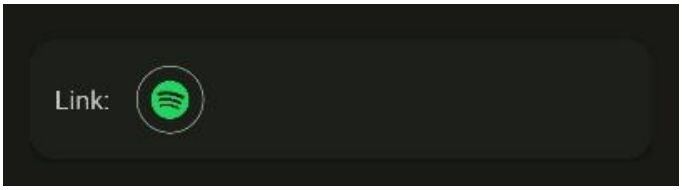


Рис. 13. Карточка інтеграції акаунту Spotify без інтегрованого Spotify акаунта

Після успішної інтеграції профілю з’являються назва профілю й аватар у вигляді карточки разом з текстом «Linked account» (рис. 14). Після натискання користувачем на цю карточку, система відкриває безпосередньо застосунок Spotify (або його вебсторінку, якщо застосунок не встановлено).

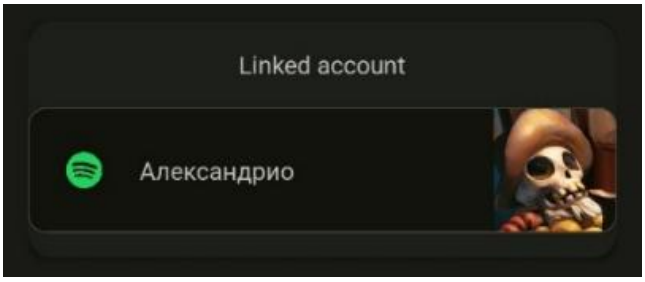


Рис. 14. Карточка інтеграції акаунту Spotify після успішної інтеграції Spotify акаунта

Далі, під карточкою інтеграції профілю Spotify, розташовано параметри застосунку у вигляді текстових пунктів з відповідною іконкою, розділених на два блоки: налаштування застосунку і налаштування акаунту користувача (рис. 15).

Для зміни кольорової теми застосунку, користувачу потрібно натиснути на пункт «Theme: …», після чого система змінить тему застосунку з темної на світлу і навпаки.

Для зміни мови застосунку користувачу потрібно натиснути на пункт «Language: …», після чого система переключить інтерфейс між англійською та українською.

У блоці «Account», користувач може відкрити форму редагування особистих даних акаунту, натиснувши на пункт «Change account info», або вийти з облікового запису, натиснувши на пункт «Log out», після чого система відкриває екран авторизації.

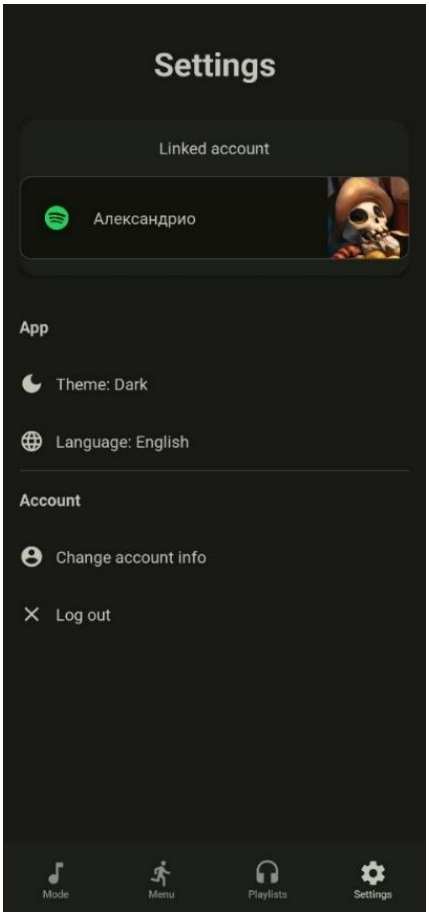


Рис. 15. Екран налаштування застосунку

9. Екран налаштування акаунту

Якщо користувач бажає змінити пошту або пароль, який використовується під час авторизації в систему, або видалити інтеграцію з поточним акаунтом Spotify чи власний акаунт в TempoRun, йому потрібно відкрити екран налаштування акаунту (рис. 16). Це він може зробити натиснувши на пункт кнопку «Change account info» в екрані налаштування застосунку.

Після відкриття екрану налаштування акаунту, система відображає користувачу перелік керувальних дій для облікового запису: змінити електрону пошту, змінити пароль, від’єднати «Spotify» та видалити аккаунт в системі TempoRun. Пункти відображаються залежно від типу авторизації: зміна пошти й пароля доступні лише користувачам із логіном-паролем, а від’єднання – лише після під’єднання акаунту «Spotify».

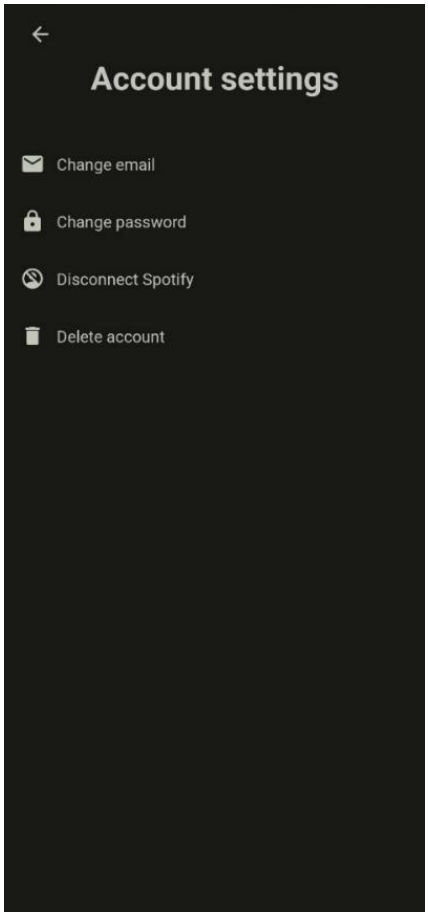


Рис. 16. Екран налаштування акаунту

Після натискання на одне з цих налаштувань, система відкриває модальне вікно введення нових даних або підтвердження дії.

Якщо користувач бажає змінити пошту, він натискає на налаштування і вводить нову пошту у відкрите модальне вікно (рис. 17).

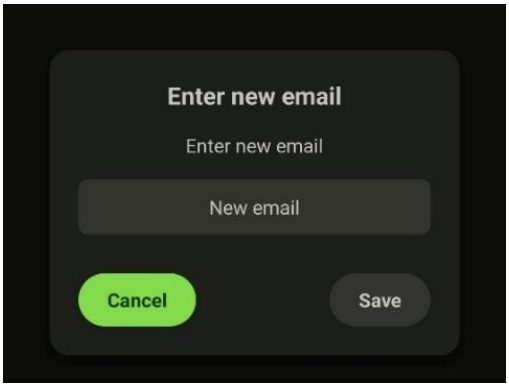


Рис. 17. Модальне вікно зміни пошти акаунту

Якщо користувач бажає змінити пароль, то він вводить новий пароль і його повтор у відкритому модальному вікні (рис. 18).

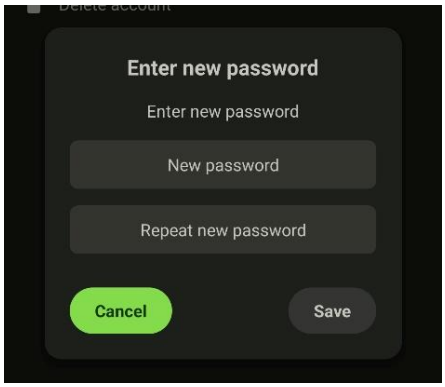


Рис. 18. Модальне вікно зміни паролю акаунту

Після підтвердження користувачем, система перевіряє чи введені дані коректні. Якщо дані були введені не коректно, то під полем вводу з’являється текст відповідної помилки. Якщо все правильно, то система відправляє відповідні зміни на сервер і закриває модальне вікно.

Якщо користувач бажає видалити свій акаунт або від’єднати акаунт Spotify, то він натискає на відповідне налаштування і перед ним з’являється

модальне вікно підтвердження дії. Лише після підтвердження, система робить відповідну дію (рис. 19).

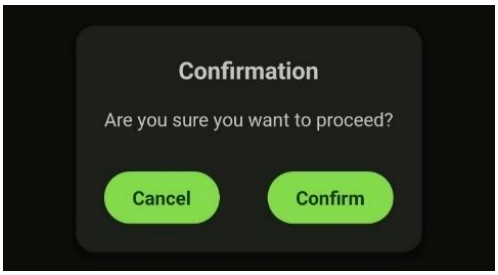


Рис. 19. Модальне вікно підтвердження дії

Для повернення на попередній екран, користувачу потрібно натиснути на кнопку з іконкою стрілки («←») на панелі зверху.

10. Екран спортивної сесії

Якщо користувач бажає розпочати спортивну сесію, йому потрібно відкрити екран спортивної сесії (рис. 20). Це він може зробити натиснувши на кнопку «Run» на екрані головного меню, тільки якщо він надав дозволи до свого місцеположення, додав додаткові опції до обраного режиму роботи алгоритму, активував плейлист з активними треками і має активним музичний програвач Spotify.

Після відкриття екрана спортивної сесії, система відображає три поточні показники – час від старту, поточний темп бігу та подолану дистанцію. Значення оновлюються показників оновлюються двічі на секунду.

Нижче розташована картка активної опції алгоритму: іконка й назва вибраного режиму, цільовий музичний темп та позиція опції у черзі.

Під карткою активної опції алгоритму, у нижній частині екрана, знаходиться велика кнопка-стоп. Натиснувши на неї, користувач може завершити спортивну сесію. При завершенні система зупиняє програвання поточного треку в Spotify і відкриває сторінку результатів.

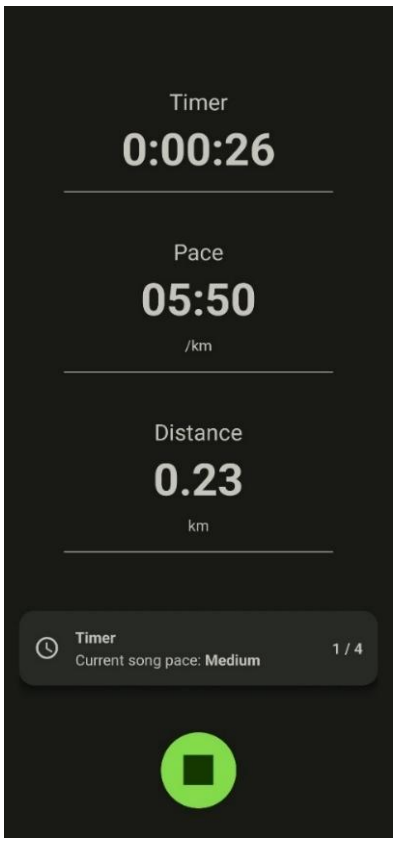


Рис. 20. Екран спортивної сесії

11. Екран результатів спортивної сесії

Після завершення сесії тренування, система відображає підсумковий екран (рис. 21). У центрі екрану картка з даними про пройдену дистанцію, загальну тривалість сесії, середній темп переміщення і кількість пройдених опцій, а під нею розташовано кнопку «Go back», при натисканні на яку система відкриває для користувача головне меню.

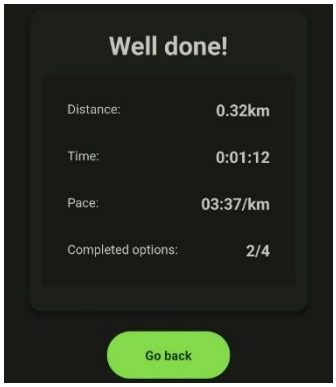


Рис. 21. Таблиця результатів спортивної сесії