КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра
за спеціальністю 122 Комп'ютерні науки
на тему:

ЗАСТОСУНОК ДЛЯ ПІДБОРУ МУЗИКИ ДО РИТМУ КРОКІВ

Виконав студент 4 курсу
ЗОТОВ Данило _____
(підпис)

Науковий керівник:
асистент, кандидат технічних наук
ФЕДОРУС Олексій _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики «____» _____
2023 р., протокол № ___
Завідувач кафедри
ТЕРЕЩЕНКО Василь _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 45 сторінок, 14 ілюстрацій, 3 таблиці, 12 джерел посилань, 1 додаток.

API, UNITY, ANDROID, МУЗИКА, АНАЛІЗ ДАНИХ, SPOTIFY WEB API, ІНТЕРФЕЙС ПРОГРАМНОГО ПРОДУКТУ, ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОДУКТУ, C#, ANDROID STUDIO, AUTH 2.0

Об’єктом роботи є розробка застосунку для підбору музики до ритму кроків користувача, реалізація алгоритму підбору музики для цього застосунку

Метою кваліфікаційної роботи є створення мобільного застосунку для автоматизації підбору музики відповідно до ритму кроків користувача

Методи дослідження: аналіз джерел. Мови програмування – C#, Java, інструменти – Unity, Spotify WEB API, середа розробки – Visual Studio 2019, Android Studio.

Результати роботи: була проведена практично-дослідна робота з розробки застосунку для підбору музики до ритму кроків, розроблено логіку алгоритму, створено технічне завдання для застосунку, створений сам застосунок згідно технічного завдання.

Результати роботи будуть корисні людям що займаються помірною фізичною активністю, власникам музичних платформ та фітнес-застосунків.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

УНХ – удари на хвилину;

API – Application Programming Interface;

URL – Uniform Resource Locator;

REST – Representational State Transfer;

JSON – JavaScript Object Notation;

ТЗ – технічне завдання;

AAR – Android Archive.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Оцінюючи сучасний стан речей на ринку музичних застосунків, можна прийти до висновку, що більшу частину займають застосунки створені для постійного програвання музики, які ще називають «стрімінговими». Такі застосунки як Spotify здатні оцінювати музичну бібліотеку за великою кількістю критеріїв, як фактичних, як гучність, ритм, так і оцінювати емоційність композицій.

Великий об’єм музичних композицій разом з усіма їх характеристиками накопичений на музичних платформах, дає змогу підлаштувати програвання музики до оточуючого середовища та ситуації. Наприклад, створити атмосферу, що сприяє легким фізичним навантаженням, таким як прогулянка. Деякі застосунки, такі як Google Fit пропонують користувачам йти на прогулянки з певною частотою кроків, щоб перетворити приємне проведення часу в легке фізичне навантаження. Проте такі застосунки не надають достатнього заохочення для людей, щоб підтримувати обраний ритм.

Створення зручного інструменту інтеграції звичних для людей застосунків для програвання музики у їх повсякденні фізичні навантаження спонукатимуть до більшого залучення в активний спосіб життя та покращення якості життя користувача.

Актуальність роботи та підстави для її виконання. Як показують дослідження [1], музика має активний вплив на те як багато часу людина приділяє фізичним навантаженням, до яких можна віднести й активну швидку прогулянку. Отже, допомога у підборі музики до ритму кроків користувача є важливою складовою заохочення його до фізичної активності.

Синхронізація рухів з музикою допомагає підтримувати стабільний темп, виконувати рухи точно та рівномірно, а також забезпечує відчуття гармонії та

задоволення від занять. Отже, розробка застосунків, які можуть допомогти підбирати музику до ритму кроків, має потенціал для поліпшення якісного виконання фізичних вправ та забезпечення задоволення від тренування.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення мобільного застосунку для автоматизації підбору музики відповідно до ритму кроків користувача. Для досягнення цієї мети поставлено такі завдання:

- Вивчити взаємозв'язку між рухами та музикою. Вплив ритму музики на швидкість руху користувача.
- Розробити метод організації аудіотеки користувача для роботи алгоритму
- Створити алгоритм, який буде враховувати ритмічні характеристики рухів та аналізувати відповідність музики.
- Розробити технічне завдання до застосунку.
- Створити зручний та інтуїтивно зрозумілий для користувачів інтерфейс.

Об'єкт, методи й засоби розроблення. Об'єктом розробки мобільного застосунку з підбору музики до ритму кроків є процес встановлення відповідності між даними отриманими від користувача, такими як швидкість ходьби, та характеристиками музичної композиції, такими як темп.

В якості інструменту створення застосунку було обрано ігровий рушій Unity 2021.2.7f1. Завдяки своїй гнучкості та багатоплатформенності, Unity дає великий вибір зручних інструментів для створення інтерфейсів, та застосування мови програмування C# у процесі створення мобільного застосунку. Ігровий рушій є безкоштовним для використання у навчальних не комерційних проєктах.

У якості середи розробки була обрана Microsoft Visual Studio 2019 Community Edition, як безкоштовна зручна IDE, що надає підтримку у розробці мовою програмування C# та дає інструменти для процесів налагодження коду при розробці за допомогою ігрового рушія Unity.

Мова програмування C# була обрана як основна мова розробки при використанні ігрового рушія Unity. Вона надає широкі можливості з використання класів для роботи з мережею, що важливо у розробці застосунку.

Джерелом інформації про музичні характеристики композицій у процесі вибору музики, а також платформою для створення готових музичних добірок буде виступати Spotify Web API. Даний ресурс дає можливість некомерційного застосування для розробки програм. Через нього можна отримати доступ до великого списку характеристик музичних композицій, що наявні в бібліотеці Spotify, а також до рекомендацій, створених для конкретного користувача на основі його бібліотеки.

Можливі сфери застосування. Даний застосунок може бути застосований спортсменами або особами, які займаються фізичною активністю самостійно, для підбору музики, що відповідає їхньому ритму та потребам. Це допоможе створити позитивну атмосферу та підвищити мотивацію під час тренувань. Часті прогулянки можуть допомогти групам осіб які страждають від проблем з ожирінням, депресією чи кров’яним тиском [1, 2].

РОЗДІЛ 1 ОСОБЛИВОСТІ ЗВ'ЯЗКУ МУЗИЧНИХ ХАРАКТЕРИСТИК З ФІЗИЧНОЮ АКТИВНІСТЮ

1.1 Вплив музики на ефективність фізичних вправ

У центрі багатьох медичних досліджень стоїть тема зв'язку між музикою та фізичною активністю людини. Досліджується вплив музики на якість тренувань, порівнюється ефективність синхронної та асинхронної музики у процесі тренувань на бігових доріжках, вплив музики на настрій та відчуття втоми.

Наприклад одним з досліджень була доведена перевага прослуховування музики синхронної до рухів під час занять на біговій доріжці, над прослуховуванням несинхронної. У бігунів, що прослуховували снхронізовану з рухами музику було зафіксовано нижче споживання кисню під час вправ [3].

Нижче споживання кисню свідчить про те, що бігуни досягали тих самих результатів використовуючи менше енергії. Однією з головних виявлених переваг нижчого споживання кисню є покращення аеробної витривалості. Коли організм ефективно використовує кисень, м'язи отримують достатню кількість палива для тривалої активності без надмірної втоми, що дозволяє тренуватися з більшою інтенсивністю і тривалістю, що сприяє покращенню загальної фізичної форми.

Крім того, низьке споживання кисню під час тренувань сприяє збільшенню сили і швидкості спортсмена. Оптимальне використання кисню дає можливість м'язам працювати ефективніше, забезпечуючи кращу координацію рухів.

Менше споживання кисню також може покращити відновлення після тренувань. Коли організм працює ефективніше, відновлення м'язів і тканин стає швидшим. Це дозволяє тренуватися частіше і зменшує ризик перенавантаження або травм [4].

Зниження споживання кисню може бути результатом кращої координації рухів під впливом синхронізованої з діями музики. Коли ритм музики та рухів синхронізовано, робота серцево-судинної системи бігуна покращується. Це впливає також і на тривалість виконання фізичних тренувань: що менше енергії людина використовує, то довше вона здатна продовжувати тренування [3].

Також важливим є те, що людина надає перевагу завжди тій музиці, що вона вже слухала до цього, або яка належить до знайомих їй жанрів, виконавців, тощо. Наприклад, одне з досліджень довело зв'язок між самостійно обраною музикою для тренувань та ефективністю людини протягом фізичних навантажень. Так, спортсмени показували більшу витривалість під час виконання силових вправ, порівняно з тими, хто займався без супроводу. Також учасники тренувань з музикою протягом тренувань мали кращий настрій та мотивацію до занять [5].

Синхронна музика, як і самостійно обрана, теж продемонструвала психологічний ефект. Так, велосипедисти яким запропонували займатись під синхронну музику, як і спортсмени, що займались під власноруч обрану, продемонстрували підвищення настрою та збільшення мотивації за результатами тренувань [3].

Також важливою є і швидкість музики для різних тренувань. Швидка та ритмічна музика підходить більше для силових тренувань та вправ високої інтенсивності, тоді як спокійна та мелодійна музика є атрибутом більш розслабляючих вправ, такі як заняття йогою.

Усі ці дослідження приводять до висновку, що вплив музики на ефективність виконання різних фізичних вправ, від пробіжок до силових тренувань є сильно індивідуальним та різниться від людини до людини. Це показує важливість особистих смаків людини при формуванні плейлиста для тренувань, на що ми звернемо увагу при розробці алгоритму для підбору музики до ритму кроків.

Також багато з досліджень проводились саме в умовах виконання помірних фізичних навантажень, таких як: заняття на біговій доріжці, велосипеді, тощо. Це підтверджує актуальність зв'язку синхронізації та ритмічності музики з помірними фізичними навантаженнями, до яких відносять також і ходьбу.

Згідно з дослідженням Градець-Краловського університету [6], було встановлено високий коефіцієнт кореляції між прослуховуванням музики та темпом ходьби в міському просторі. Людина в більшості випадків намагається синхронізувати швидкість своїх кроків з музикою, якщо це можливо. Це показує, що музика під час прогулянок може слугувати як зовнішній ритмічний стимул, що впливає на фізичну активність людини.

На рис. 1 зображено порівняння впливу від прослуховування музики з вищим показником темпу та з нижчим на проходження відрізків дороги у міських умовах на одному й тому самому шляху. Верхня крива відповідає вищому темпу музики, нижня – нижчому.

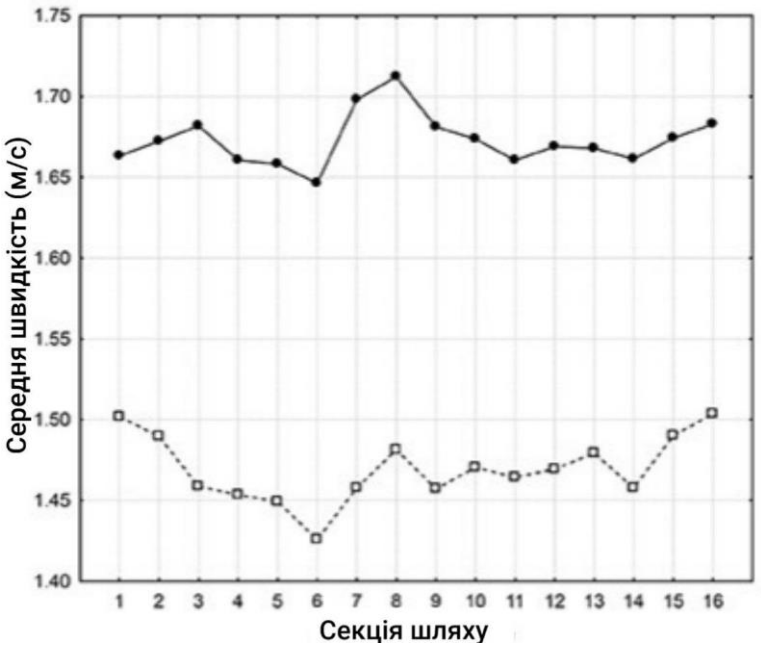

Рисунок 1 – Графік швидкості проходження однакових ділянок шляху з різним музичним супроводом

З цього графіку ми можемо зробити висновок, що не дивлячись на те, що на відповідних відрізках шляху учасники експерименту практично однаково прискорювались чи сповільнювались, різниця в середній швидкості ходьби залишалась практично сталою.

Також дослідження виявило індивідуальність впливу музики на швидкість ходьби. Деякі люди є менш чутливими до зовнішнього впливу музики, тоді як інші показуються високу чутливість. Це може також свідчити про важливість індивідуальних вподобань, так як під час експериментів використовувався узагальнений спільний набір композицій.

Додатково стаття звертає увагу на контекст використання музики під час фізичної активності. В міському середовищі, де існують такі ризики як рух транспорту та інші перепони, вибір темпу музики може впливати на безпеку прогулянки. Наприклад, швидка музика може спонукати до більшої швидкості ходьби, що може бути небезпечним у міському трафіку.

Загалом висновки зі статті підкреслюють важливість особистого підходу при підборі музики під час звичайних прогулянок. Правильно підібрана музика може сприяти більшій мотивації та задоволенню від прогулянок, дарувати відчуття відпочинку.

1.2 Зв'язок ритму композиції та її впливу на психологічний стан

Ритм музики є важливим елементом сприйняття композицій, що також впливає і на психологічний стан людини. Дослідження також показують, що ритм музики може впливати на настрій і емоційний досвід.

Ритм – це організований патерн часових відношень і повторень звуків, ударів або акцентів, які відтворюються впродовж певного проміжку часу. Ритм

визначається через регулярність, темп та акцентуацію музичних подій. Наша увага під час аналізу необхідних статей буде прикована саме до темпу.

Темп визначає швидкість ритму. Він відображає кількість музичних подій, що відбуваються за одиницю часу. Темп може бути повільним, помірним або швидким, в залежності від того, як швидко або повільно відтворюється мелодія.

Деякі дослідження [7] показують вплив темпу музики на сприйняття композиції людиною. Швидша музика з високим темпом сприяє позитивній оцінці. З іншого боку повільна музика з низьким темпом викликатиме заспокоєння.

Також важливою для цього дослідження є вплив регулярності ритму. Складні та нерегулярні ритмічні структури викликали сильну емоційну реакцію, яка не завжди була пов'язана з позитивними емоціями.

На рис. 2 показано як поєднання валентності та темпу змінює відношення людини до композиції. Так швидка в темпі музика з високою валентністю сприймається людиною як більш енергійна та викликає почуття радості.

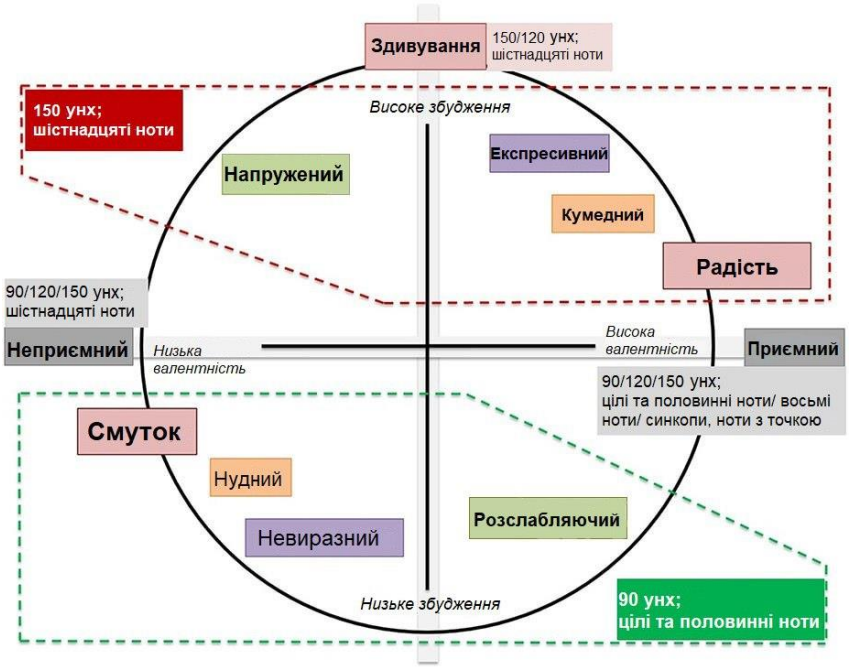

Рисунок 2 – Діаграма впливу темпу та валентності на емоційне забарвлення

Також важливу роль у сприйнятті емоцій відіграє динаміка музики – характеристика, що відображає гучність та виразність звуків. Згідно дослідження [8], композиції з більшою динамікою – тобто значними коливаннями гучності сприймаються як більш виразні та викликають яскравіші емоції.

Важливо зазначити загальний вплив музики на психологічний стан людини. Музика може мати значний вплив на емоційний досвід людини. Відповідно формувати її настрій на досить довгі періоди часу.

Крім того, важливо врахувати, що існують великі індивідуальні відмінності у сприйнятті музики різними людьми. Різні люди можуть мати різні емоційні відгуки на одну й ту ж музику в залежності від їх суб'єктивних вподобань, життєвого досвіду та інших впливових факторів [9].

Отже, при розробці алгоритму з підбору музики, важливо врахувати не лише схожість по ритму музики до швидкості кроків, а й інші параметри музики такі як її динаміка, темп, ритмічний рисунок для оптимального підбору музики, що відповідала б настрою користувача.

Складена на основі рис.2 та отриманих з різноманитних досліджень даних табл.1 демонструє характеристики музики на які варто звернути увагу та як зміна цих параметрів впливає на емоційне забарвлення музики.

Таблиця 1 – Параметри музичних характеристик музики з різним емоційним забарвленням

	Темп, УНХ	Динаміка	Валентність
Енергійна	120-150	Висока	Висока
Розслабляюча	90-120	Висока	Висока
Нервова	120-150	Висока	Низька
Сумна	90-120	Низька	Низька

РОЗДІЛ 2 РОЗРОБКА АЛГОРИТМУ ПІДБОРУ МУЗИКИ ДО РИТМУ КРОКІВ

2.1 Аналіз критеріїв для вибору музики

Для розробки якісного алгоритму з підбору музики, необхідно спочатку оцінити усі можливі вхідні параметри, які ми будемо давати на обробку та вихідні дані, які очікуємо від нього отримати.

Головним вхідним параметром для нашого алгоритму буде ритм кроків користувача. Відштовхуючись від цього параметру ми будемо формувати список композицій, що підходять та мають відповідний ритм.

Оскільки ритм у музиці рідко буває стабільним протягом усієї композиції, та знайти музику з ідеальним ритмом в, наприклад, 120 УНХ неможливо, необхідно збільшити вікно вибору композицій. Згідно досліджень, відомо, що не тренована людина погано розрізняє зміну ритму близько 10% від початкового [10].

Отже, якщо користувач йде з приблизним ритмом кроків у 120 кроків на хвилину, то ми можемо обирати для його прогулянки музику з ритмом у 108-132 УНХ. У відповідних умовах, якщо ритм кроків користувача всього 90 кроків на хвилину, то ритм музики може варіюватись лише між значеннями 81 та 99 УНХ.

Як було оговорено у першому розділі, вибір музики для користувача не може засновуватись лише на самому ритмі його кроків але також на його ритмічному рисунку та динаміці.

Для оцінки ритмічного рисунку ми будемо посилатись на два параметри: музичний розмір та валентність. Музичний розмір, також відомий як тактовість або метр – це кількість тактів у одному розмірі у композиції. Для прикладу розмір чотири четвертих означає, що в кожному такті є чотири четверті ноти. Валентність у свою чергу – це поняття емоційного відтінку композиції. Часто вона визначається

відносним значенням у межах від нуля до одиниці, де нуль присвоюється неприємній, сумній, дискомфортній музиці та одиниця – радісній, енергійній.

Щодо музичного розміру, то бажане значення для усіх підібранних композицій це саме чотири четвертих, тому що у такій конфігурації, кожному кроку користувача буде відповідати один такт у музиці, що дає можливість добре синхронізувати ритм кроків з ритмом власне композиції.

Щодо валентності, то її значення залежатиме від ритму кроків користувача та його побажань у настрої пісні, що шукається. Якщо ритм кроків низький, нижче за 120 кроків на хвилину та він очікує на сумну та спокійну музику, нам варто шукати композиції зі значенням валентності 0.3-0.5. Повільні композиції з нижчими значеннями валентності можуть бути невиразними. З тим самими ритмом але вищою валентністю, близько 0.5-1.0 пісні будуть розслабляючі. Для високого ритму кроків необхідно шукати композиції зі значеннями валентності не нижче 0.5-1.0, оскільки композиції з нижчою валентністю та високим ритмом будуть здаватись тривожними.

Також не варто забувати про динаміку. Чим вищий ритм кроків, тим вищу ми прагнемо динаміку для пісні, щоб вона створювала враження енергійної й мотивувала продовжувати рухатись у тому ж темпі.

2.2 Підходи до розробки алгоритму

Можна виділити декілька основних підходів, що можуть бути використані для ефективного вибору музики для користувача. Наприклад:

- Базовий аналіз;
- Аналіз емоційної валентності;
- Рекомендації на основі схожості;
- Аналіз на основі контексту.

Для вдалого вибору музики для користувача, необхідно використовувати усі ці підходи разом. Розберемо кожен з них докладніше.

Базовий аналіз та класифікація музичних композицій дозволяє нам заранння упорядкувати музику за її основними характеристиками, такими як темп, ритм, тон, динаміка, тощо. Частково ми будемо втілювати цей підхід у процесі аналізу бібліотеки користувача, для створення початкових даних для роботи алгоритму.

Аналіз емоційної валентності – використання якого ми досить докладно обґрунтували у попередніх розділах. Ми будемо його втілювати оцінюючи музичні композиції за їх значенням емоційної валентності, а також оцінювати вплив сумісної дії таких характеристик як темп, метр, динаміка на емоційне забарвлення музики.

Рекомендації на основі схожості – цей підхід використовує методи рекомендаційних систем для знаходження схожих музияних композицій. Він може враховувати попередні вподобання користувача, аналізувати їх історичний вибір музики та рекомендувати подібні композиції. Цей підхід буде головним з точки зору персоналізації запропонованого плейлисту для користувача. Для його реалізації ми будемо використовувати існуючі алгоритми Spotify Web API що знаходяться у вільному доступі.

Врахування контексту підбору музики. Цей підхід враховує контекстуальну інформацію таку як місце, час, діяльність, настрій Контекст у випадку даного алгоритму є наперед визначеним: підбір музики в умовах вуличної прогулянки чи пробіжки. Настрій при цьому користувач може задати з наперед визначених варіантів: Енергійна чи Спокійна.

У якості результату, алгоритм буде вертати набір пісень у формі плейлисту на музичній стримінговій платформі Spotify. Оптимальним вважаємо повертати плейлист обсягом п'ятдесят композицій, оскільки час програвання такого плейлисту

складає дві з половиною години, що є більш ніж достатнім часом для активної вуличної прогулянки.

2.3 Розробка алгоритму на основі Spotify Web API

2.3.1 Опис Spotify Web API

API – це набір правил та протоколів, які визначають спосіб взаємодії між різними програмами чи компонентами програмного забезпечення. API допомагає визначити які операції та функціональні можливості доступні для використання зовнішніми розробниками [11]. Дана форма реалізації інтерфейсу дозволяє розробникам використовувати готовий код та отримувати результати виконання певних запитів зовнішніми додатками.

Найбільш поширеним стандартом для API є Representational State Transfer API, або скорочено REST. На основі архітектури REST API побудований також і Spotify Web API.

Основні принципи REST API включають:

- Використання стандартних HTTP методів, таких як Get, Post, Put, Delete та деяких інших;
- Використання URL-адрес для ідентифікації ресурсів;
- Використання статус-кодів HTTP для передачі інформації про стан запиту до API;
- Відсутність залежності між клієнтом та сервером.

На рис.3 зображено спрощену схему процесу запиту до WEB API через мобільний застосунок. За унікальним ідентифікатором URL застосунок звертається до API з запитом. API у свою чергу звертається до бази даних, що знаходиться на віддаленому сервері, обробляє запит та повертає у відповідь дані у форматі JSON.

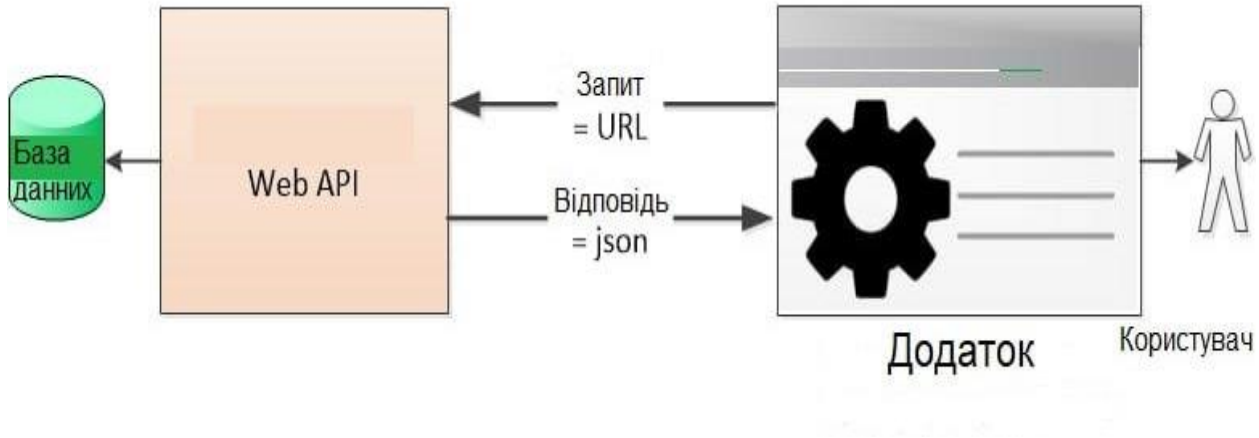

Рисунок 3 – Схема процесу запиту до WEB API

Структура запиту до WEB REST API, у тому числі й до Spotify API включає:

- HTTP метод, що вказує дію над ресурсом;
- URL-адресу ресурса на сервері;
- Заголовки, що включають додаткову інформацію про запит, таку як тип даних, мова, код аутентифікації, тощо;
- Тіло запиту за потреби у форматі JSON, XML, Web-Form чи іншому;
- Параметри запиту, що можуть бути передані у URL чи в тілі запиту.

Spotify WEB API – це набір API за REST архітектурою, які надаються популярною музичною платформою Spotify. Це офіційний інтерфейс, що дозволяє розробникам створювати додатки та сервіси, що взаємоді.ть з музичними даними та функціями Spotify.

За допомогою Spotify WEB API, розробники можуть отримати доступ до різних фунцій та даних, включаючи:

- Отримання інформації про користувача, доступу до бібліотеки його композицій, до його рекомендацій;

- Отримання широкого спектру характеристик музичної композиції, таких як темп, валентність, динаміка, тощо;
- Пошук композицій за критеріями;
- Керування програванням музики, включаючи створення та редагування плейлистів.

Як ми можем побачити, Spotify WEB API має широкий інструментарій, який є для нас необхідним у процесі створення алгоритму з підбору музики до ритму кроків.

2.3.2 Авторизація у Spotify WEB API

Для успішного використання більшості потрібних нам запитів, існує необхідність авторизуватись у нашому застосунку через акаунт користувача у Spotify. Авторизація Spotify WEB API проходить через процес OAuth 2.0, де наш мобільний застосунок буде авторизований як Spotify застосунок.

OAuth 2.0 – це відкритий протокол авторизації, що активно використовується для обміну даними між різними сервісами. Його мета це створити умови у яких користувачі мають змогу надавати доступ до своїх облікових записів третій стороні, не передаючи при цьому свої дані для авторизації. [12]

Як зображено на рис.4, протокол OAuth 2.0 працює на основі взаємодії між трьома основними суб’єктами:

- Користувачем – власником ресурсу, який надає доступ до своїх облікових даних;
- Клієнтом – застосунком, що робить запит до даних користувача;
- Сервером авторизації – сервером, що видає токени доступу до облікових даних Користувача.

Рисунок 4 – процес авторизації OAuth 2.0

На рис.4 зображена також схема основних етапів взаємодії в межах протоколу OAuth 2.0. Так, спочатку клієнт реєструється, отримуючи Client ID та Client Secret. У нашому випадку реєстрація відбувається на сайті Spotify Developers. Після цього Клієнт направляє Користувача на Сервер авторизації, для аутентифікації Користувача. Після аутентифікації, Сервер повертає клієнту код авторизації. Цей код Клієнт міняє в Сервера на Токен доступу. Використовуючи цей токен для авторизації під час виконання запитів до WEB API дозволяє Клієнту тимчасово отримати доступ до ресурсів Користувача.

Для отримання коду авторизації у Spotify використовується переадресація на адресу вказану в створеному Клієнті застосунку. Оскільки наш застосунок мобільний, для зручності, замість web-адреси, процес авторизації повертає Користувача назад до застосунку, й передає код авторизації. Детальніше як це відбувається ми розглянемо у третьому розділі.

Роблячи запит авторизації у Spotify, клієнт визначає усі необхідні доступи, які потрібні йому для функціонування застосунку. У нашому випадку це доступ до бібліотеки користувача, його рекомендацій та можливість створювати та редагувати від його імені не публічні альбоми, в які ми будемо зберігати підібрану музику.

2.3.3 Організація аудіотеки користувача для використання її в подальшому алгоритмом

Завдяки використанню Spotify WEB API ми маємо можливість персоналізувати вибір музики для користувача. Але для успішної роботи алгоритму треба виконати попередню обробку музичної бібліотеки користувача. Для цього Spotify WEB API надає нам інструменти у вигляді ресурсів для отримання самих композицій з бібліотеки користувача, та можливості отримати до них повну інформацію щодо музичних характеристик.

Спочатку ми обробляємо бібліотеку збережених композицій користувача за ресурсом User`s Saved Tracks. На рис.5 зображено як виглядає запитом по URL цього ресурсу у форматі cURL. Видно, що по запиту в URL передається три параметри: регіон в якому мають бути доступні композиції, їх кількість та зміщення від найновішої композиції в збережених.

```
curl --request GET \
  --url https://api.spotify.com/v1/me/tracks \
  --header 'Authorization: Bearer BQDKEN...15NEzQ'
```

Рисунок 5 – Приклад запиту до Spotify WEB API

Spotify WEB API має обмеження, щодо кількості треків, які він передає нам за раз, через що ми маємо поступово, порціями не більше ніж п’ятдесят композицій зберегти усі необхідні нам дані. У даному випадку ми зберігаємо лише ID композицій, яке знадобиться нам потім для отримання додаткової інформації про характеристики композиції та додавання цих композицій до майбутнього плейлиста.

Після цього, необхідно сформувати ще один запит, тепер до ресурсу який повертає музичні характеристики кількох композицій одночасно. На рис.6 показано приклад відповіді на запит до цього ресурсу.


```
{
  "acousticness": 0.00242,
  "analysis_url": "https://api.spotify.com/v1/audio-analysis/2takcwOaAZWiXQijPHIx7B",
  "danceability": 0.585,
  "duration_ms": 237040,
  "energy": 0.842,
  "id": "2takcwOaAZWiXQijPHIx7B",
  "instrumentalness": 0.00686,
  "key": 9,
  "liveness": 0.0866,
  "loudness": -5.883,
  "mode": 0,
  "speechiness": 0.0556,
  "tempo": 118.211,
  "time_signature": 4,
  "track_href": "https://api.spotify.com/v1/tracks/2takcwOaAZWiXQijPHIx7B",
  "type": "audio_features",
  "uri": "spotify:track:2takcwOaAZWiXQijPHIx7B",
  "valence": 0.428
}
```


Рисунок 6 – Приклад відповіді на запит до Spotify WEB API

В нас немає необхідності зберігати усі музичні характеристики композиції. Для роботи нашого алгоритму, як ми визначили у розділі 2.1 достатньо лиш зберігати значення таких полів:

- loudness, що відповідає за динаміку;
- tempo, що відповідає за ритм;
- time_signature, що відповідає за метр;
- valence, що відповідає за валентність.

На решту полів ми не будемо звертати уваги. Для збереження цих полів створимо модель класу Features, де будуть поля loudness, tempo, valence типу double та поле time_signature типу int. Зберігати треки будемо у словнику, де ключами слугуватимуть ID композицій, а значеннями екземпляри класу Features.

Варто звернути увагу, що у цього ресурсу теж є обмеження на кількість композицій для яких можна отримати список характеристик: не більше ста за один запит. Тому якщо кількість до цього отриманих збережених композицій користувача більше ста, то запит треба буде робити кілька разів, поки не будуть проаналізовані усі композиції.

Також для майбутнього використання у алгоритмі, зробимо запит на ресурс Get Available Genre Seeds. У відповідь, ресурс поверне нам усі ті жанри, які оцінюються як ті, що подобаються, чи можуть сподобатися користувачу. Ця інформація пізніше знадобиться нам для отримання рекомендованих композицій для користувача.

2.3.4 Написання алгоритму підбору музики

У результаті дій усіх попередніх розділів, ми зібрали та проаналізували достатню кількість інформації, для того щоб реалізувати логіку самого алгоритму підбору музики.

Викликаючи метод підбору музики, ми передаємо до нього наш цільовий темп та настрій, який задає користувач, доповнюючи контекстуальну складову алгоритму підбору музики.

Далі, на основі аналізу емоційної валентності та базового аналізу, необхідну інформацію для яких ми підготували у розділі 2.3.3, алгоритм може підготувати список з усіх можливих композицій, що вже є у бібліотеці користувача, а отже, були ним уподобані раніше, через що й потрапили до бібліотекі.

Як описувалось у розділі 2.1, композиції будуть підбиратись згідно меж, що встановлюються для кожного параметру: loudness, tempo, time_signature, valence. У табл.2 наведена інформація як комбінації вхідних параметрів будуть впливати на межі цільових параметрів. Для значень tempo наведено на скільки відсотків розширяться межі для пошуку ритму. Решта значень є відносними, без одиниць виміру

Таблиця 2 – Орієнтовні значення цільових параметрів для підбору музики

	loudness	tempo, %	time_signature	valence
120-150, Енергійна	-20.0-0.0	12-15%	4	0.7-1.0
120-150, Спокійна	-30.0-(-20.0)	12-15%	4	0.5-0.7
90-120, Енергійна	-30.0-(-20.0)	9-12%	4	0.5-0.7
90-120, Спокійна	-50.0-(-30.0)	9-12%	4	0.3-0.5

Якщо з бібліотеки користувача згідно цільових параметрів вдасться обрати 50 чи більше композицій, тоді на цьому роботу алгоритму можна вважати завершеною, залишається лиш додати знайдені композиції у плейлист користувача. Якщо ж знайдених композицій менше за 50, тоді ми намагаємось додати необхідну кількість композицій на основі рекомендацій, що нам може запропонувати платформа Spotify для цього користувача.

Для того щоб отримати список рекомендацій, нам необхідно створити запит до ресурсу Get Recommendations. Цей запит потребує 5 параметрів генерації рекомендацій, які можуть бути трьох типів: seed_artist, seed_genres, seed_tracks.

У якості одного параметра типу seed_tracks ми можемо взяти випадкову композицію з числа уже відібраних для додавання в плейлист, чи будь-яку з бібліотеки користувача, якщо відібрати для плейлиста нічого не вдалось.

Для одного параметра типу seed_artists ми можемо додати ID виконавця відібраної композиції. Для того щоб його отримати, нам доведеться зробити окремий запит по ресурсу Get Track, якому надати ID відібраної композиції. У відповіді серед інших характеристик композицію буде об’єкт виконавця, з якого ми й можемо отримати ID виконавця для параметру seed_artists.

Решту три параметри ми можемо заповнити параметрами типу seed_genres. Назви жанрів для цього параметру ми вже завбачливо зберегли у пункті 2.3.3.

Також до запиту Get Recommendations ми можемо додати велику кількість опціональних параметрів, що будуть вимогами до треків, що рекомендуються. Серед цих параметрів:

- min_loudness, max_loudness та target_loudness – параметри мінімального, максимального та бажаного значення динаміки;
- min_tempo, max_tempo та target_tempo – параметри мінімального, максимального та бажаного значення темпу;
- target_time_signature – бажане значення метру;

- min_valence, max_valence та target_valence – параметри мінімального, максимального та бажаного значення валентності;
- А також limit – кількість композицій у відповіді та market – у якому регіоні композиції мають бути доступні.

Відповіддю на запит буде список композицій, які підходять усім заданим параметрам, кількістю не більше ніж limit. ID цих композицій ми додамо до загального списку відібраних ID щоб потім додати їх до створеного плейлисту.

Використання ресурсу Get Recommendations дозволяє знаходити достатню кількість композицій для користувача згідно більш жорстких обмежень до цільових параметрів, при цьому зберігаючи індивідуальність у підході до підбору музики для користувача.

2.3.5 Додавання музики до плейлисту

Останнє що буде виконувати алгоритм, це за допомогою Spotify WEB API утворювати в аккаунті користувача новий плейлист з 50 відібраних треків, згідно з ритмом ходьби користувача.

Для початку, необхідно звернутись до ресурсу Create Playlist. До його URL необхідно додати ID користувача, а також відправити з запитом тіло у форматі JSON, яке складається з хоча б одного поля name – назви створеного плейлисту. Назву плейлисту ми будемо генерувати відповідно від ритму ходьби для якого був створений плейлист. Робиться це для того, щоб користувач міг за потреби знов скористатись вже створеним плейлистом, замість того щоб витрачати час та енергію на створення нового, коли у цьому немає потреби.

У якості відповіді на свій запит, ми отримаємо повний опис нового плейлиста. Для майбутніх запитів нам необхідно зберегти ID створеного плейлиста.

Після створення плейлисту, ми звертаємось до ресурсу Add Items to Playlist. Схоже до створення плейлиста, тут необхідно додати в URL параметр ID плейлиста, а в тіло запиту записати у форматі JSON spotify_uris усіх композицій, що ми хочемо додати до плейлиста. До цього ми зберігали лише ID композицій, проте немає нічого складного в тому щоб утворити з ID – spotify_uri, додавши перед значенням ID частину "spotify:track:". Також у тіло ми додаємо параметр position зі значенням 0, оскільки ми додаємо композиції до початку пустого плейлиста.

В результаті наших дій та роботи алгоритму, користувач отримає на своєму Spotify аккаунті новий плейлист, що буде виглядати як на рис.7.

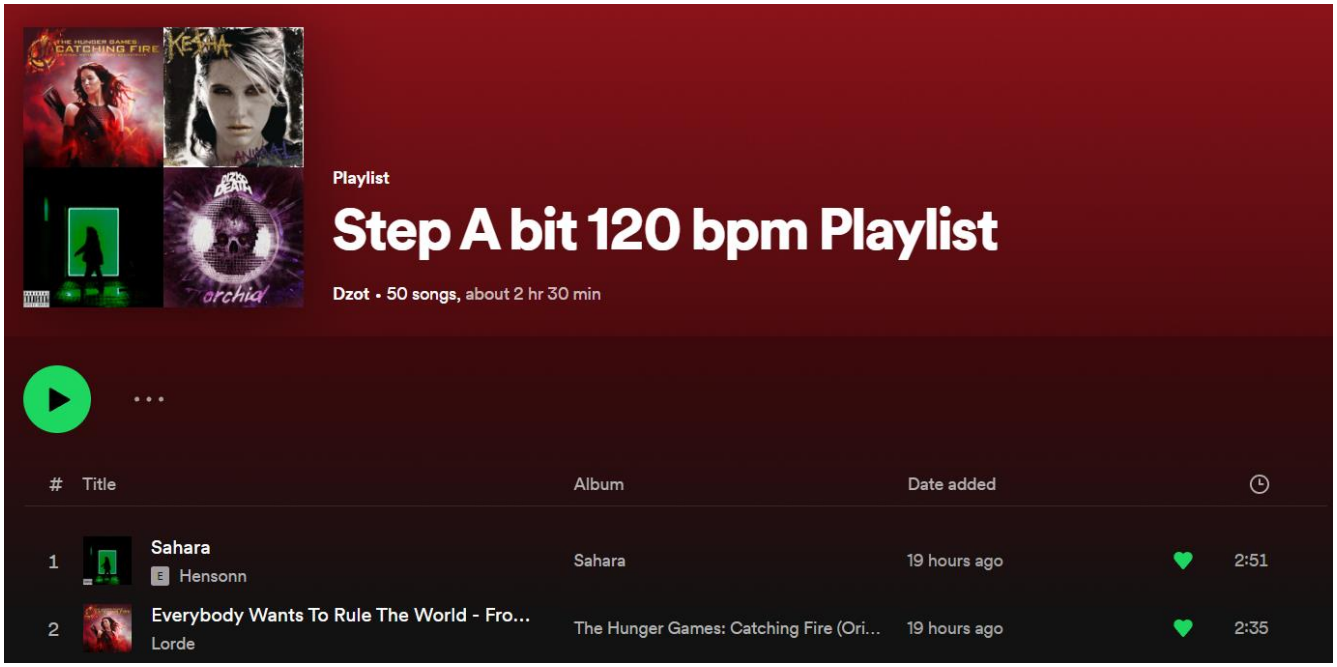

Рисунок 7 – Приклад Створеного плейлиста

На цьому алгоритм з підбору музики до ритму кроків завершує свою дію.

РОЗДІЛ 3 РОЗРОБКА ЗАСТОСУНКУ ДЛЯ ПІДБОРУ МУЗИКИ ПІД РИТМ КРОКІВ

3.1 Опис технічного завдання до застосунку

Технічне завдання, скорочено ТЗ – це документ який містить детальний опис вимог до функціональності, характеристик, архітектури чи інших аспектів програмного забезпечення. Цей документ є необхідним для забезпечення чіткості, однозначності визначення характеристик та можливостей продукту.

ТЗ для розробки андроїд застосунку має включати в себе:

- Опис проєкту;
- Вимоги до функціональності;
- Вимоги до інтерфейсу;
- Нефункціональні вимоги;
- Архітектура та технології.

Для опису ТЗ для «Застосунку для підбору музики до ритму кроків» послідовно опишемо ці пункти.

Опис проєкту: застосунок для підбору музики до ритму кроків покликаний зручний та персоналізований досвід від помірних фізичних навантажень, надаючи користувачам можливість слухати музику, що відповідає їх темпу руху.

Головна мета проєкту – забезпечити користувачу музичний супровід для його прогулянок чи тренувань. Аналізуючи його музичні вподобання та темп кроків, застосунок створить персоналізований плейлист з музичних композицій, які допоможуть користувачу покращити ефект від тренувань.

Вимоги до функціональності, що описують переліку функцій та можливостей, які повинні бути реалізовані у мобільному застосунку.

Вони включають:

- Збір даних про кроки – застосунок повинен мати здатність отримувати дані про швидкість кроків користувача з вбудованого педометра;
- Аналіз ритму кроків – застосунок повинен мати алгоритм, що здатний обробляти дані про кроки та виначати темп ходьби користувача цей алгоритм може використовувати розрахунок середнього темпу на основі кількості кроків за певний відрізок часу;
- Підбір музики за темпом – застосунок повинен мати доступ до бази даних музичних композицій, що також зберігає дані про музичні характеристики композицій, такі як темп, валентність, динаміка, тощо. На основі цієї бази даних, застосунок має здійснити вибір музики для користувача;
- Персоналізація вибору музики – застосунок повинен надати користувачеві музику згідно його вподобань у музиці, таких як жанри, улюблені виконавці, тощо;
- Інтеграція з музичними сервісами – застосунок повинен мати можливість інтеграції з популярними музичними платформами, такими як Spotify.

Вимоги до інтерфейсу: застосунок повинен мати інтуїтивно зрозумілий та зручний інтерфейс. Це включає головний екран з пунктами меню, до інших екранів застосунку, екран інтеграції з музичними сервісами, де користувач може надати дозволи для застосунку на маніпуляції з аккаунтом на зовнішній музичній платформі, екран з якого користувач може заміряти темп своїх кроків у реальному часіі, а також екран, де користувач може власноруч задати темп кроків.

Нефункціональні вимоги, що описують якості які має мати застосунок, не пов’язані з його безпосередніми функціями. Вони включають:

- Ефективність: застосунок має працювати швидко, забезпечуючи швидку обробку запитів користувача;

- Надійність: застосунок має бути стабільним у роботі та не мати помилок чи несправностей;
- Сумісність: застосунок має бути сумісним з різними версіями операційної системи Android та мати можливість запуску на якомога більшому обсязі мобільних телефонів;
- Адаптивність: застосунок повинен мати адаптивний інтерфейс, що підходить для використання на пристроях з різною апаратною конфігурацією та роздільною здатністю екрану;
- Безпека: застосунок повинен бути якомога більш конфіденційним, мати механізми захисту особистої інформації користувачів.

Архітектура та технології, де описано загальну архітектуру застосунку, середовище розробки, мови програмування, фреймворки, тощо. Воно включає:

- Архітектура: застосунок буде представляти собою клієнт який матиме змогу обробляти інформацію отриману з серверної частини популярних музичних платформ і робити запити відповідно до результатів попередньої обробки;
- Мова програмування: у якості основної мови програмування буде використано мову C#. Деякі плагіни для використання певних можливостей операційної платформи Android будуть написані мовою Java;
- Середовище розробки: для розробки застосунку буде використано ігровий рушій Unity, який надає велику кількість зручних інструментів в тому числі для розробки мобільних застосунків, а не лише ігор. Окремі плагіни для застосунку будуть написані у середовищі мобільної розробки Android Studio;
- Залучені API: Для отримання доступу до музичної колекції користувача та отримання інформації про музичні характеристики композицій буде використано Spotify WEB API, який знаходиться у вільному доступі.

З даним описом технічного завдання можна переходити безпосередньо до розробки застосунку.

3.2 Вимірювання ритму кроків користувача

Головними вхідними даними для алгоритму підбору музики описаному в розділі 2 є ритм кроків користувача, який виражається середнім значенням кількості кроків, що людина робить у хвилину. Для того щоб виміряти середню швидкість кроків хвилину, вимірювати кількість кроків користувача треба хоча б 2 хвилини. Отримавши значення кількості кроків за кожну з хвилин можна отримати середнє значення, яке можна надати для обробки алгоритму підбору музики.

Для того щоб виміряти кількість кроків які робить користувач, застосунку необхідно отримати доступ до апаратного педометру пристрою. Педометр у мобільних пристроях зазвичай є застосунком, що використовує вбудовані акселерометри та датчики руху для підрахунку кількості кроків користувача.

Для того щоб отримати доступ до вбудованого педометру в операційній системі Андроїд через застосунок, що розробляється можливостями ігрового рушія Unity, необхідно написати плагін для Android, який ми будемо підключати до застосунку під час роботи, та звертатись через нього до операційної системи пристрою.

Плагін – це додатковий модуль для розробки застосунків для операційної системи Android. Він надає розробникам засоби необхідні для розширення функціональності застосунку. Плагіни можуть включати в себе функціональні можливості по типу взаємодії з сенсорами пристрою, інтеграцію зі сторонніми сервісами, розширені можливості тестування, тощо.

Для написання плагіну, що надасть нам доступ до сенсору педометра необхідно створити новий проєкт в Android Studio. Це середовище розробки мобільних додатків має великий вибір готових архітектур, як показано на рис.8. Оскільки ми лиш створюємо плагін, обираємо пустий проєкт з назвою No Activity.

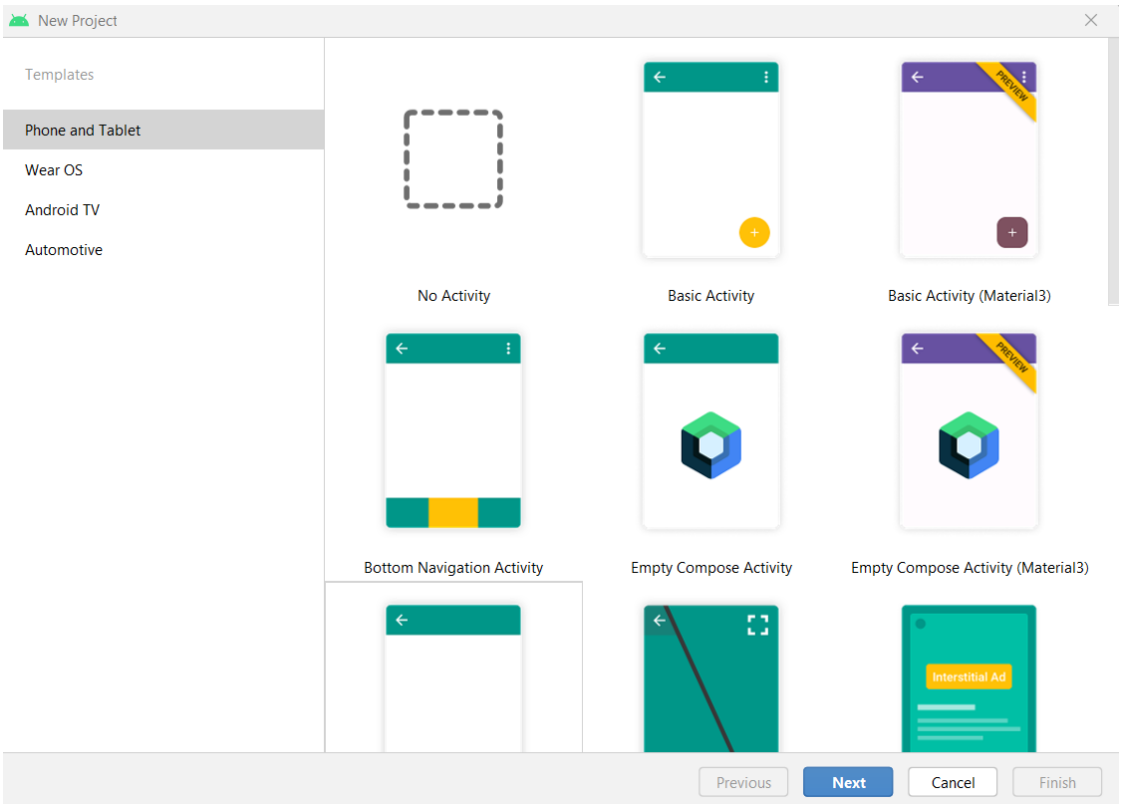

Рисунок 8 – можливості створення додатків Android Studio

Для того щоб створити саме плагін, створюємо у Android Studio нову Android бібліотеку. У ній ми створимо клас Pedometer, що буде імплементацією інтерфейсу з Android API SensorEventListener. Даний інтерфейс використовується для створення класів, здатних отримувати інформацію з сенсорів при їх зміні у режимі реального часу. Для цього, інтерфейс має два методи: onAccuracyChanged та onSensorChanged. Які ми маємо імплементувати. Оскільки сенсор педометру працює зі сталою точністю, метод onAccuracyChanged ми імплементуємо порожнім.

Нас цікавить метод onSensorChanged який приймає у якості параметра об’єкт класу SensorEvent, що має інформацію про зміну сенсору. На рис.9 показано імплементацію цього методу. Об’єкт delegate при цьому є функціональним інтерфейсом PedometerDelegate з реалізованим методом OnStep, що приймає два параметри: кількість кроків та відстань. За допомогою нього ми будемо отримувати інформацію про кроки та відстань пройдену користувачем вже безпосередньо в застосунку написаному в середовищу Unity.

```
public void onSensorChanged(SensorEvent event) {
    double STEP2METERS = 0.715D;
    int steps = (int)event.values[0];
    double distance = (double)steps * 0.715D;
    this.delegate.onStep(steps, distance);
}
```

Рисунок 9 – Реалізація методу класу SensorEventListener

Створений плагін ми білдимо засобами Android Studio у формат .aar – файловий формат, що використовується для розповсюдження плагінів для Android застосунків. У файлі формату AAR зберігається компільований код, ресурси та конфігураційні файли необхідні для використання бібліотеки у застосунку.

Для того щоб використовувати можливості плагіну, створимо C# клас Pedometer, що буде слугувати API між застосунком та плагіном. Він буде мати поля початкової кількості кроків, оскільки деякі педометри на пристроях рахують кількість кроків з моменту запуску пристрою, відповідно початкової дистанції та об’єкту делегату, аналогу функціонального інтерфейсу з плагіну, StepCallback. Цей делегат вказує на функцію, що має вхідні параметри кількості кроків та пройденої відстані. Повний код класу Pedometer представлений у додатках (див. додаток А)

Для використання Pedometer у застосунку, в класі де планується використання цього класу, необхідно створити функцію OnStep з вхідними параметрами цілочисельним steps та числом з плаваючою точкою distance. Потім необхідно ініціалізувати клас Pedometer з функцією OnStep і кожен раз, коли педометр пристрою буде оновлювати кількість пройдених кроків, буде виконуватись функція OnStep. Приклад використання ви можете побачити на рис.10.

```
private void Start () {

    pedometer = new Pedometer(OnStep);

    if (Pedometer.Implementation.IsSupported)
    {
        distanceText.text = "Supported";
    }
    else
    {
        distanceText.text = "Not supported";
    }
}

1 reference
private void OnStep (int steps, double distance) {
    stepText.text = steps.ToString();
}
```

Рисунок 10 – Приклад використання API Pedometer

Для фіксації ритму ходьби користувача, користувачу буде запропоновано почати замір кількості кроків та йти дві хвилини у бажаному темпі. В цей час кількість кроків буде фіксувати клас StepMeter. По закінченю двох хвилин

користувачу буде показано його результат, та на основі ритму його кроків створено плейлист з підібраною музикою.

Альтернативно, бажаний ритм музики користувач може вказати сам, тоді алгоритм створить плейлист на основі параметрів, що забажав користувач.

3.3 Реалізація звернень до Spotify WEB API

Для функціонування алгоритму вибору музики до ритму кроків, описаному в Розділі 2, необхідний доступ із застосунку до Spotify WEB API та можливість робити до цього API велику кількість різних запитів.

У класі SpotifyAPI буде проводитись контроль за успішністю виконання запитів, попередня обробка інформації для створення запитів, а також обробка отриманої інформації з метою надати її у зручному вигляді для роботи алгоритму. Але SpotifyAPI не буде створювати запити як такі.

За створення запитів до Spotify WEB API створено клас WebRequestHandler. Цей клас буде формувати запити на основі готових наданих даних, надсилати їх та повертати дані у відповідь, не проводячи ніякої їх обробки.

3.3.1 Опис класу WebRequestHandler

Клас WebRequestHandler, як було описано вище, відповідає за формування запитів, надсилання їх на сервер, отримання та подільшу передачу інформації до класу SpotifyAPI.

Для того щоб формувати запити до серверу, у класі WebRequestHandler використовується модуль UnityEngine.Networking. Це модуль, що надає можливості

до формування та надсилання запитів до віддалених серверів та отримання від них відповіді.

Для формування запиту у UnityEngine.Networking використовується клас UnityWebRequest. Він є ключовим компонентом для здійснення HTTP-запитів і взаємодії з віддаленими серверами. Даний клас підтримує HTTP-методи типу GET, POST, PUT, DELETE, дозволяє конфігурувати Header параметри запиту, додавати тіло запиту у вигляді www-form або рядку.

Даний клас має три статичних методи, які повертають об’єкти класу зарання конфігуровані для поширених HTTP запитів GET, POST та DELETE. На рис.11 показано як створюється GET запит за допомогою UnityWebRequest, до якого в Header додається поле авторизації користувача.

```
string url = "https://api.spotify.com/v1/me/tracks";
url += "?market=" + market;
url += "&limit=" + limit.ToString();
url += "&offset=" + offset.ToString();

UnityWebRequest www = UnityWebRequest.Get(url);
www.SetRequestHeader("Authorization", "Bearer "+PlayerPrefs.GetString("AuthCode"));
yield return www.SendWebRequest();
```

Рисунок 11 – Формування GET запиту

Для створення ж запиту власноруч, до об’єкту класу UnityWebRequest необхідно додати два класи: UploadHandler та DownloadHandler. Перший клас необхідний для того щоб додати до UnityWebRequest тіло запиту. UploadHandler оперує потоком байтів, тому ми конвертуємо тіло у форматі JSON до байтового вигляду та додаєм до запиту. DownloadHandler у свою чергу допомагає взаємодіяти з відповідю, що надходить з сервера після запиту. На рис.12 зображене формування.

```
string url = "https://api.spotify.com/v1/users/" + userId + "/playlists";

string form = "{\"name\":\""+name+"\", \"public\": false}";
UnityWebRequest request = new UnityWebRequest(url, "POST");
byte[] bodyRaw = Encoding.UTF8.GetBytes(form);
request.uploadHandler = (UploadHandler)new UploadHandlerRaw(bodyRaw);
request.downloadHandler = (DownloadHandler)new DownloadHandlerBuffer();
request.SetRequestHeader("Content-Type", "application/json");
request.SetRequestHeader("Authorization", "Bearer " + PlayerPrefs.GetString("AuthCode"));

yield return request.SendWebRequest();
```

Рисунок 12 – Формування POST запиту

Також для обробки відповідей у форматі JSON та отриманні необхідної інформації клас WebRequestHandler використовує модуль Newtonsoft.Json.Linq. Ця частина бібліотеки Newtonsoft.Json дозволяє зручно взаємодіяти та маніпулювати даними в форматі JSON. Він дозволяє звертатись до полів JSON файлу напряму за їх ключем, та отримувати їх значення у форматі рядка. На рис.13 показано як за допомогою Newtonsoft.Json.Linq з відповіді на запит отримуються усі поля “id” об’єктів “track” в масиві об’єктів “items”.

```
List<string> res = new List<string>();
JObject info = JObject.Parse(data);

foreach (var item in info["items"])
{
    res.Add(item["track"]["id"].ToString());
}
```

Рисунок 13 – Отримання полів з JSON

Загалом у WebRequestHandler є 9 методів, кожен з яких звертається до свого ресурсу. Оскільки запити робляться асинхронно, то кожен з цих методів окрім необхідних значень полів для запиту, в аргументах приймає значення типу Action<T>, де T – тип значень, яке ми хочемо повернути. Це дозволяє після закінчення виконання запиту на сервер, повернути з потоку значення які необхідні для подальшої роботи, перед тим як поток знищиться.

3.3.2 Опис класу SpotifyAPI

Для того щоб інкапсулювати виклик методів API та зробити звернення до них зручними, створений клас SpotifyAPI. Оскільки зручніше коли усі запити до API контролюються з одного місця, то цей клас за патерном проєктування є Singleton. Даний патерн чудово підходить для випадків коли нам необхідно створити клас, що буде керувати глобальним станом програми.

Клас SpotifyAPI має 5 методів, кожен з яких надсилає за допомогою класу WebRequestHandler один чи кілька запитів, оброблює значення, що отримав у відповідь та повертає ці значеня для обробки алгоритмом підбору музики до ритму кроків. Методи, їх параметри та функціонал описані табл.3

Таблиця 3 – Опис функцій класу SpotifyAPI

	Параметри	Значення	Функціонал
GetUsersSaved	Ніяких	List<string>	Опрацьовує бібліотеку користувача та повертає список з ідентифікаторів композицій
GetTracksBPM	List<string>	List<double>	Приймає список ідентифікаторів пісень та повертає список значень їх ритму
CreatePlaylist	string, List<string>	bool	Приймає назву плейлисту та список пісень, повертає логічне значення: істина якщо плейлист створено, хиба якщо

Кінець Таблиці 3

	Параметри	Значення	Функціонал
GetReferenceVals	string	List<string>	Приймає значення ідентифікатора композиції з бібліотеки користувача, повертає значення необхідні для створення запиту з отримання рекомендацій
GetRecommendations	int, double, string, string, string	List<string>	Приймає кількість рекомендацій яку треба отримати, бажані параметри, ідентифікатори пісні, виконавця та жанрів на основі яких будуть згенеровані рекомендації. Повертає список ідентифікаторів рекомендованих пісень.

3.4 Створення інтерфейсу застосунку

User Interface – скорочено UI – це спосіб взаємодії користувача з програмою. Це включає в себе елементи які користувач бачить на екрані, такі як кнопки, текстові поля, панелі, меню, іконки та інші графічні елементи.

UI визначає спосіб, яким користувачі взаємодіють з програмою і відіграє важливу роль у тому щоб застосунком було зручно користуватись. Якісний

інтерфейс допомагає створити позитивний користувацький досвід, спрощує навігацію та роботу з програмою.

Мета хорошого UI полягає в тому щоб користувачі могли легко зрозуміти як використовувати програму без зайвих зусиль.

Для створення користувацького інтерфейсу Unity пропонує використовувати UI Toolkit. Це великий набір різних скриптів та компонентів, таких як кнопки, тексти, списки, повзунки, панелі, тощо. Зручний інтерфейс Unity Editor дозволяє швидко розробляти необхідні екрани, додаючи елементи на екран в один клік.

За відображення елементів інтерфейсу в Unity відповідає об’єкт Canvas. Усі можливі UI елементи в структурі проєкту мають належати об’єктку Canvas як батьківському. Також для того щоб з об’єктами на екрані можна було взаємодіяти, у проєкті має бути об’єкт EventSystem. Він реєструє будь-які взаємодії користувача з застосунком, та здатний розрізняти велику кількість різних пристроїв вводу. На рис.14 зображена структура об’єктів користувацького інтерфейсу у застосунку.

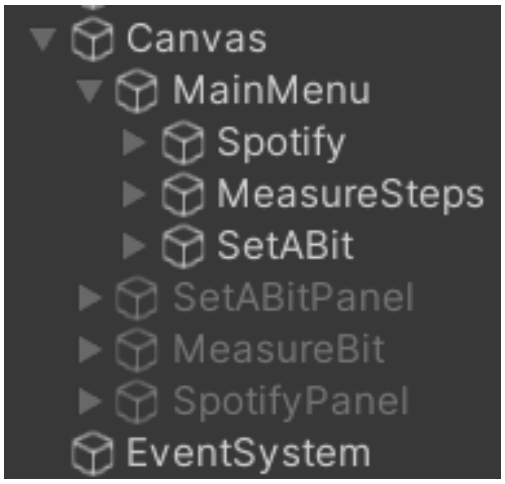

Рисунок 14 – Структура об'єктів користувацького інтерфейсу

Велика перевага Unity UI Toolkit у його кросплатформенності, що дозволяє легко переносити застосунок за потреби на інші платформи, такі як iOS.

Контроль над поведінкою елементів користувацького інтерфейсу відбувається через клас MenuButtonManager. Полями класу є різні панелі меню: Головне меню, Меню взаємодії з Spotify, Меню вимірювання ритму кроків та Меню задання ритму. Усі вони знаходяться на одному екрані, та перемикаються натисканнями кнопки. Це було зроблено для швидкодії, щоб уникнути повільного завантаження між екранами застосунку.

ВИСНОВКИ

У цій роботі була проведена практично-дослідна робота з розробки застосунку для підбору музики до ритму кроків.

У процесі аналізу було визначено взаємозв’язок між музикою та її впливом на ефективність у проведені помірних фізичних навантажень. Так, на основі оглянутих досліджень, було зроблено висновок, що музика під час тренувань здатна підвищити ефективність тренувань, знизити відчуття втоми, покращити психологічний стан та збільшити мотивацію до фізичних навантажень. Також було визначено сферу використання застосунку, як помічника у тренуваннях.

Також в процесі аналізу джерел було визначено які музичні характеристики є важливими для ефективної роботи алгоритму з підбору музики до ритму кроків. Так, було визначено, що окрім власне ритму музики, обираючи композиції варто звернути увагу на валентність композиції, її динаміку, та метр.

У процесі розробки алгоритму, було визначено референтні значення для попередньо визначених музичних характеристик. Також було визначено логіку алгоритму та обрано Spotify WEB API, як допоміжний API у процесі роботи алгоритму, який може надати усю необхідну інформацію для персоналізованого вибору треків для користувача.

Було також визначено технічне завдання для застосунку, встановлені вимоги до функціональності, інтерфейсу, засобів розробки

Був написаний плагін для отримання доступу до внутрішнього сенсору операційної системи Android, реалізовані поміжні класи для створення запитів до Spotify WEB API. Створено користувацький інтерфейс за допомогою інструментів Unity UI Toolkit.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Thakare A. Effect of music tempo on exercise performance and heart rate among young adults. / A. Thakare, R. Mehrotra, A. Singh. // Int J Physiol Pathophysiol Pharmacol. – 2017. – №9. – С. 35–39.
2. Hanson S. Is there evidence that walking groups have health benefits? A systematic review and meta-analysis. / S. Hanson, A. Jones. // Br J Sports Med.. – 2015. – №49. – С. 710–715.
3. Bacon C. Effect of music-movement synchrony on exercise oxygen consumption. / C. Bacon, T. Myers, C. Karageorghis. // J Sports Med Phys Fitness.. – 2012. – №52. – С. 359–65.’
4. Oxygen consumption and usage during physical exercise: the balance between oxidative stress and ROS-dependent adaptive signaling. / [Z. Radak, Z. Zhao, E. Koltai та ін.]. // Antioxid Redox Signal.. – 2013. – №18. – С. 1208–46.
5. Effects of self-selected music on strength, explosiveness, and mood. / [M. Biagini, L. Brown, J. Coburn та ін.]. // J Strength Cond Res.. – 2012. – №26. – С. 1934–8.
6. Tempo and walking speed with music in the urban context [Електронний ресурс] / Franěk Marek, van Noorden Leon, Režný Lukáš // Frontiers in Psychology – 2014. – №5 – Режим доступу до журн. : https://www.frontiersin.org/articles/10.3389/fpsyg.2014.01361/full#h3
7. Influence of Tempo and Rhythmic Unit in Musical Emotion Regulation [Електронний ресурс] / Fernández-Sotos Alicia, Fernández-Caballero Antonio, Latorre José // Frontiers in Computational Neuroscience – 2016. – №10 – Режим

доступу до журн. : https://www.frontiersin.org/articles/10.3389/fncom.2016.00080/full#h5

8. Kamenetsky S. Effect of Tempo and Dynamics on the Perception of Emotion in Music / S. Kamenetsky, H. David, T. Sandra. // Psychology of Music. – 1997. – №25. – С. 149–160.
9. Juslin P. Handbook of Music and Emotion: Theory, Research, Applications / Patrik Juslin., 2010. – (Online edn.).
10. Temporal Processing in Audition: Insights from Music. / Vani G. Rajendran, Sundeep Teki, Jan W.H. Schnupp // Neuroscience. – 2018. – №389. – С. 4–18.
11. Bloch J. How to design a good API and why it matters / Joshua Bloch. // OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. – 2006. – С. 506–507.
12. Richer J. OAuth 2 in Action / J. Richer, A. Sanso., 2017. – 360 с. – (Simon and Schuster).

ДОДАТОК А

```
namespace PedometerU {

    using UnityEngine;
    using System;
    using Platforms;

    6 references
    public sealed class Pedometer : IDisposable {


        1 reference
        public int updateCount { get; private set; }
        public static readonly IPedometer Implementation;

        private int initialSteps;
        private double initialDistance;
        private readonly StepCallback callback;

        1 reference
        public Pedometer (StepCallback callback) {
            if (Implementation == null) {
                Debug.LogError("Pedometer Error: Step counting is not supported on this platform");
                return;
            }
            if (callback == null) {
                Debug.LogError("Pedometer Error: Cannot create pedometer instance with null callback");
                return;
            }
            this.callback = callback;
            Implementation.OnStep += OnStep;
        }

        1 reference
        public void Dispose () {
            if (Implementation == null) {
                Debug.LogWarning("Pedometer Error: Step counting is not supported on this platform");
                return;
            }
```

Рисунок А.1 – Код класу Pedometer

```
            1 reference
            public void Dispose () {
                if (Implementation == null) {
                    Debug.LogWarning("Pedometer Error: Step counting is not supported on this platform");
                    return;
                }
                Implementation.OnStep -= OnStep;
            }

            2 references
            private void OnStep (int steps, double distance) {
                // Set initials and increment update count
                initialSteps = updateCount++ == 0 ? steps : initialSteps;
                initialDistance = steps == initialSteps ? distance : initialDistance;
                // If this is not the first step, then invoke the callback
                if (steps != initialSteps)
                    callback(steps - initialSteps, distance - initialDistance);
            }


        }
    }
```

Рисунок А.2 – Продовження коду

Посилання на Github репозиторій з кодом: https://github.com/Dzotz/StepABit