МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Харківський національний університет радіоелектроники

Факультет комп’ютерних наук .

(повна назва)

Кафедра програмної інженерії .

(повна назва)

**КОМПЛЕКСНИЙ КУРСОВИЙ ПРОЄКТ**

**Пояснювальна записка**

рівень вищої освіти перший (бакалаврський) .

Програмна система для керування розумним гітарним процесором

засобами мобільної розробки .

(тема)

Виконав:

здобувач 3 курсу, групи ПЗПІ-22-5

Ярослав ЗАХАРЧЕНКО

(Власне ім’я, ПРІЗВИЩЕ)

Спеціальність 121 – Інженерія програмного забезпечення

(код і повна назва спеціальності)

Тип програми освітньо-професійна . Освітня програма Програмна інженерія

(повна назва освітньої програми)

Керівник ст. викл. кафедри ПІ Гліб ТЕРЕЩЕНКО

(посада, Власне ім’я, ПРІЗВИЩЕ)

Члени комісії (Власне ім’я, ПРІЗВИЩЕ, підпис)

доц. Олексій НАЗАРОВ .

ст. викл. Сергій МАР’ЇН .

ст.викл. Марія ШИРОКОПЕТЛЄВА .

2025 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерних наук

Кафедра програмної інженерії

Рівень вищої освіти перший (бакалаврський)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-професійна

Освітня програма Програмна Інженерія

(шифр і назва)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Курс | 3 | Група | ПЗПІ-22-5 | Семестр | 6 |

**ЗАВДАННЯ**

***на курсовий проект(роботу) студента***

здобувачеві Захарченку Ярославу Вадимовичу

(прізвище, ім’я, по батькові)

1. Тема роботи Програмна система для створення та редагування пресетів розумного гітарного процесора засобами мобільних технологій

2. Термін здачі студентом закінченої роботи „ 19 ” серпня 2025 р.\_

3. Вихідні дані до проєкту Розробити мобільний застосунок для керування розумним гітарним процесором. Застосунок має забезпечувати можливість створення, редагування та збереження пресетів із різними ефектами, відтворення треків із локального пристрою, а також взаємодію з REST API для отримання та збереження даних. Передбачити підтримку авторизації користувача, адаптивний інтерфейс, відображення ефектів у вигляді інтерактивних елементів, а також можливість оцінювання (лайки) і коментування пресетів.

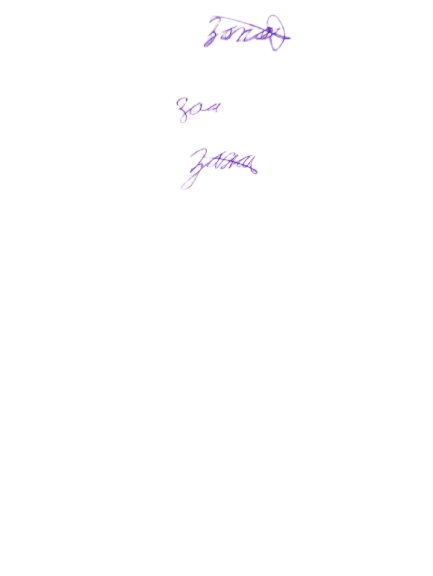
4. Перелік питань, що потрібно опрацювати в роботі

Вступ, аналіз вимог до інтерфейсу користувача, вибір технологічного стеку, проєктування структури застосунку та архітектурного підходу, реалізація інтерфейсу користувача, взаємодія з REST API, робота з локальними файлами, інтеграція функціоналу лайків і коментарів, обробка помилок і станів застосунку адаптивність інтерфейсу, тестування, висновки, додатки.

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
| 1 | Аналіз предметної галузі | 08.05.25 – 12.05.25 | виконано |
| 2 | Розробка постановки задачі | 13.05.25 – 14.05.25 | виконано |
| 3 | Проектування ПЗ | 15.05.25 – 22.06.25 | виконано |
| 4 | Програмна реалізація | 23.06.25 – 04.07.25 | виконано |
| 5 | Аналіз результатів | 05.07.25 – 12.07.25 | виконано |
| 6 | Підготовка пояснювальної записки. | 13.07.25 – 17.07.25 | виконано |
| 7 | Перевірка на наявність ознак академічного плагіату | 23.08.25 | виконано |
| 8 | Захист роботи | 20.06.25 | виконано |

Дата видачі завдання “ 23 ” серпня 2025р.



Здобувач \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис)

Керівник роботи ст.викл. кафедри ПІ .

(підпис) Гліб ТЕРЕЩЕНКО

(посада, Власне ім’я, ПРІЗВИЩЕ)

**РЕФЕРАТ / ABSTRACT**

Пояснювальна записка містить: 61 стор., 21 рис., 3 табл., 9 джерел.

РОЗУМНИЙ ГІТАРНИЙ ПРОЦЕСОР, ANDROID, KOTLIN, RETROFIT, REST API, ROOM, MVVM, MEDIA PLAYER, MOBILE APP

Об’єкт розробки – мобільний застосунок для створення, редагування та організації гітарних пресетів з можливістю використання аудіотреків, ефектів та інтеграції з backend-сервісом.

Мета розробки – створення зручного, функціонального та адаптивного мобільного інтерфейсу, який дозволяє гітаристам управляти власними пресетами, додавати ефекти, редагувати параметри, слухати результат у реальному часі та взаємодіяти з іншими користувачами (лайки, коментарі).

Метод рішення – використання сучасної технології Android-розробки на мові Kotlin із застосуванням архітектурного підходу MVVM; інтеграція з REST API через бібліотеку Retrofit; збереження даних у локальній базі Room/SQLite; використання MediaPlayer для відтворення треків; розробка адаптивного та привабливого інтерфейсу засобами XML-розмітки та стилів.

SMART GUITAR PROCESSOR, ANDROID, KOTLIN, RETROFIT, REST API, ROOM, MVVM, MEDIA PLAYER, MOBILE APP

The development object is a mobile application for creating, editing, and organizing guitar presets with the ability to use audio tracks, apply effects, and integrate with a backend service.

The development goal is to create a convenient, functional, and adaptive mobile interface that allows guitarists to manage their presets, add effects, adjust parameters, listen to the result in real time, and interact with other users (likes, comments).

The solution method is to use modern Android development technologies with the Kotlin programming language and the MVVM architectural approach; integrate with the REST API via Retrofit; store data in a local Room/SQLite database; use the MediaPlayer for audio playback; and design an adaptive and attractive interface through XML layouts and styles.

**ЗМІСТ**

[Перелік скорочень 7](#_Toc201611507)

[Вступ 8](#_Toc201611508)

[1 Аналіз предметної галузі 10](#_Toc201611509)

[1.1 Аналіз предметної галузі 10](#_Toc201611510)

[1.2 Виявлення та вирішення проблем 11](#_Toc201611511)

[1.3 Аналіз аналогів програмного забезпечення 12](#_Toc201611512)

[1.3.1 AmpliTube 12](#_Toc201611513)

[1.3.2 Guitar Rig 13](#_Toc201611514)

[1.3.3 Висновки щодо аналізу аналогів 14](#_Toc201611516)

[2 Постановка задачі 16](#_Toc201611517)

[3 Архітектура та проєктевання програмного забезпечення 18](#_Toc201611518)

[3.1 UML проєктування ПЗ 18](#_Toc201611519)

[3.2 Проектування архітекрути ПЗ 19](#_Toc201611520)

[3.3 Проєктування структури зберігання даних 21](#_Toc201611521)

[3.4 Приклади алгоритмів та методів 23](#_Toc201611522)

[3.5 Створення UI/UX дизайну системи 25](#_Toc201611523)

[3.6 Опис підготовки даних 27](#_Toc201611524)

[4 Опис прийнятих програмних рішень 29](#_Toc201611525)

[4.1 Вибір архітектури клієнтської частини 29](#_Toc201611526)

[4.2 Вибір бібліотек та технологій 30](#_Toc201611527)

[4.3 Організація навігації та управління станом у мобільному застосунку 31](#_Toc201611528)

[4.4 Робота з редактором пресетів 32](#_Toc201611529)

[4.5 Адаптивність інтерфейсів під мобільний застосунок 34](#_Toc201611530)

[4.6 Продуктивність та оптимізація 35](#_Toc201611531)

[5 Аналіз отриманих результатів 39](#_Toc201611532)

[5.1 Порівняння результатів з початковими вимогами 39](#_Toc201611533)

[5.2 Аналіз ефективності прийнятих рішень 40](#_Toc201611535)

[5.3 Обмеження та недоліки 41](#_Toc201611536)

[Висновки 42](#_Toc201611537)

[Перелік джерел посилання 44](#_Toc201611538)

[ДОДАТОК А 45](#_Toc201611538)

[ДОДАТОК Б 47](#_Toc201611538)

[ДОДАТОК В 53](#_Toc201611538)

# **ПЕРЕЛІК СКОРОЧЕНЬ**

API – Application Programming Interface

BPM – Beats Per Minute

DB – Database

DSP – Digital Signal Processing

IDE – Integrated Development Environment

ISO – International Organization for Standardization

JSON – JavaScript Object Notation

Kotlin – мова програмування Kotlin

MIDI – Musical Instrument Digital Interface

MP3 – MPEG Audio Layer III

MVVM – Model–View–ViewModel

OOP – Object-Oriented Programming

REST API – Representational State Transfer Application Programming Interface

Retrofit – бібліотека для роботи з REST API в Android

Room – бібліотека Android Jetpack для роботи з локальною БД SQLite

SDK – Software Development Kit

SQL – Structured Query Language

UI – User Interface

UX – User Experience

WAV – Waveform Audio File Format

XML – eXtensible Markup Language

# **ВСТУП**

Управління звуком та ефектами для музичних інструментів потребує сучасного, комплексного підходу, який поєднує гнучкість, багатофункціональність та зручність для кінцевого користувача. В умовах швидкого розвитку цифрових технологій зростає попит на інструменти, що дозволяють музикантам експериментувати з власним звучанням, створювати унікальні звукові палітри та легко ділитися результатами з іншими. Важливою складовою таких рішень є можливість створення і редагування гітарних пресетів, використання аудіотреків та інтеграція з інтелектуальними алгоритмами, які допомагають автоматизувати частину процесів і оптимізувати робочий процес музиканта.

Для досягнення цих завдань застосовується клієнт-серверна архітектура, що забезпечує ефективний розподіл обробки даних між сервером, відповідальним за збереження, обробку та управління інформацією, та мобільним клієнтом, який надає інтуїтивно зрозумілий і доступний інтерфейс. Такий підхід дозволяє розвантажити мобільний пристрій, забезпечити масштабованість рішення та гарантувати надійність роботи системи.

Мобільний застосунок відіграє ключову роль у забезпеченні зручності та ефективності взаємодії користувача з процесором ефектів. Саме він відповідає за інтерактивний контроль, відтворення аудіо, зміну параметрів ефектів та створення комфортного середовища для музиканта під час репетицій чи виступів. Важливо, щоб інтерфейс такого застосунку був не лише естетично привабливим, а й функціональним, адаптивним та відповідним сучасним вимогам до продуктивності й ергономіки.

Темою комплексного курсового проєкту є розробка «розумного» гітарного процесора на базі мобільного застосунку. Основними можливостями системи є створення, редагування та організація пресетів, додавання звукових ефектів, регулювання їх параметрів у реальному часі, а також інтерактивна взаємодія з іншими користувачами за допомогою лайків і коментарів. Це не лише підвищує індивідуальність звучання, але й створює елемент соціальної взаємодії та формує спільноту музикантів навколо продукту.

Метою розробки є створення клієнтської частини мобільного застосунку для гітаристів, що забезпечить зручний, інтуїтивно зрозумілий та адаптивний інтерфейс із підтримкою роботи з аудіотреками. Також передбачена інтеграція з REST API для збереження та завантаження даних, а також функціонал для управління ефектами та пресетами.

У процесі реалізації проєкту використовувалися сучасні технології та інструменти мобільної розробки. Зокрема, мова програмування Kotlin, архітектурний підхід MVVM для відокремлення бізнес-логіки від інтерфейсу, бібліотека Retrofit для взаємодії з REST API та Room для роботи з локальною базою даних. Окрім цього, застосовувалися стандартні Android-компоненти для побудови інтерфейсу та роботи з мультимедіа, а також сучасні підходи до оптимізації продуктивності.

Основна увага приділялася створенню інтуїтивно зрозумілого та інтерактивного інтерфейсу, що забезпечує комфортну роботу з ефектами та треками, сприяє покращенню користувацького досвіду та відповідає сучасним трендам мобільної розробки.

# **АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ**

## Аналіз предметної галузі

Сучасні музиканти потребують інноваційних інструментів для керування гітарними ефектами та створення унікального звучання. Традиційні апаратні процесори залишаються актуальними, проте вони часто є дорогими, громіздкими та складними у налаштуванні. У зв’язку з розвитком мобільних технологій зросла популярність програмних рішень, які забезпечують зручне керування ефектами, створення пресетів і взаємодію з треками без використання додаткового обладнання. Такі застосунки, як Tonebridge чи AmpliTube, демонструють попит на подібні рішення, однак обмежені у функціоналі. Це визначає актуальність розробки «розумного» гітарного процесора у вигляді мобільного застосунку, який поєднує гнучкість, простоту та широкі можливості персоналізації.

Одним із визначальних чинників ефективності сучасних мобільних застосунків є наявність зручного та зрозумілого інтерфейсу, що дозволяє користувачеві швидко опановувати функціонал і працювати з ним у реальному часі. Для музичних інструментів, зокрема гітари, це означає можливість легко створювати, редагувати й комбінувати ефекти, а також персоналізувати звучання під власні уподобання. Інтерфейс повинен бути однаково доступним як для початківців, так і для досвідчених музикантів, пропонуючи прості налаштування та водночас гнучкі інструменти для тонкої роботи зі звуком. Важливу роль відіграє інтеграція з додатковими сервісами — наприклад, з бібліотеками треків, хмарними сховищами або штучним інтелектом для автоматичної генерації пресетів. Мобільний фронтенд у цьому контексті стає ключовим елементом, адже саме через нього реалізується взаємодія користувача із системою: вибір та редагування ефектів, робота з треками, налаштування параметрів звучання. Висока якість інтерфейсу, його адаптивність, підтримка світлої й темної теми та орієнтація на позитивний користувацький досвід (UX) визначають успішність і конкурентоспроможність «розумного» гітарного процесора.

## Виявлення та вирішення проблем

У ході розробки мобільного застосунку для «розумного» гітарного процесора було проаналізовано ключові труднощі, з якими стикаються музиканти під час роботи з подібними системами. Попри наявність широкого вибору інструментів і процесорів на ринку, більшість із них мають недоліки, пов’язані з перевантаженим інтерфейсом, складними налаштуваннями та відсутністю зручних засобів швидкого редагування параметрів ефектів. Це створює бар’єр для новачків і знижує ефективність використання програм у концертних чи студійних умовах.

Однією з найпоширеніших проблем є надмірна кількість вкладених меню для зміни параметрів звучання. У запропонованому рішенні було реалізовано систему динамічних блоків, де користувач може редагувати потрібні ефекти за допомогою слайдерів і простих візуальних контролів без необхідності перемикатися між екранами. Це значно спрощує роботу з процесором і робить її більш інтуїтивною.

Ще однією проблемою виявилося зниження продуктивності при використанні декількох ефектів одночасно. Для подолання цього було застосовано оптимізовану обробку даних та ефективну взаємодію з API, що дало змогу мінімізувати затримки й забезпечити плавність роботи навіть за складних налаштувань.

Крім цього, було враховано потребу в інтеграції з інтелектуальними інструментами. Реалізовано можливість використання функцій штучного інтелекту для генерації рекомендованих налаштувань ефектів, автоматичного створення пресетів і персоналізації звучання відповідно до стилю гри користувача.

Важливою частиною проєкту стала також реалізація системи сповіщень і підказок. Застосунок інформує користувача про успішне збереження пресету, помилки при роботі з параметрами або інші важливі події. Це дозволяє уникати плутанини й забезпечує більш комфортний досвід використання.

Окремо слід зазначити потребу в інтеграції з інтелектуальними сервісами, які можуть допомагати користувачу під час створення контенту. У рамках цього проєкту реалізовано можливість використовувати функції штучного інтелекту – зокрема, доповнення, покращення або переклад тексту – через інтерфейс, доступний безпосередньо в редакторі.

Таким чином, у процесі розробки було визначено й усунуто основні проблеми, притаманні подібним рішенням: складність інтерфейсу, низьку продуктивність, відсутність адаптивності та інтеграції з AI. Запропоноване рішення поєднує простоту використання, сучасні технології та розширені можливості, що робить його зручним інструментом як для аматорів, так і для професійних музикантів.

## Аналіз аналогів програмного забезпечення

На ринку існує низка програмних рішень для гітаристів. Вони пропонують широкий набір інструментів для налаштування звучання та роботи з ефектами. Водночас у більшості систем спостерігаються типові обмеження: складний інтерфейс, орієнтація лише на професіоналів, відсутність інтеграції з AI та недостатня адаптація під мобільні пристрої. Це створює потребу в новому застосунку з простим та інтуїтивним UI, гнучким налаштуванням ефектів і підтримкою інтелектуальних технологій.

### AmpliTube

AmpliTube – це популярний програмний гітарний процесор, який надає широкий набір інструментів для створення та моделювання звучання. Він підтримує роботу з віртуальними підсилювачами, кабінетами, педалями та студійними ефектами. Завдяки цьому користувач може формувати власні ланцюги сигналів та експериментувати з налаштуваннями у режимі реального часу (див. рис. 1.3).



Рисунок 1.3 – Інтерфейс роботи у AmpliTube

Основними недоліками AmpliTube є часткова платність ключових функцій, а також обмежена гнучкість кастомізації у безкоштовній версії. Крім того, інтерфейс може здаватися перевантаженим для початківців, що ускладнює процес освоєння.

### Guitar Rig

Guitar Rig – це професійний софт для моделювання гітарного та басового звучання, розроблений компанією Native Instruments. Він надає широкий вибір підсилювачів, кабінетів та ефектів, які можна комбінувати у власні ланцюги обробки сигналу. Додатково присутні студійні інструменти для мікшування та мастерингу, що робить програму корисною не лише для гітаристів, але й для звукорежисерів (див. рис. 1.4).



Рисунок 1.4 – Інтерфейс робочого простору у Guitar Rig

До недоліків Guitar Rig належать висока вимогливість до системних ресурсів, що може викликати затримки на слабших пристроях, а також складність для новачків через велику кількість доступних параметрів. Частина функцій доступна лише у повній платній версії.

### Висновки щодо аналізу аналогів

Проведений аналіз існуючих програмних рішень для гітаристів, зокрема AmpliTube та Guitar Rig, дав змогу виявити їхні сильні сторони та обмеження. Обидві системи забезпечують користувача широким набором можливостей для роботи зі звуком: вони пропонують велику кількість підсилювачів, ефектів, готових пресетів, а також високу якість обробки аудіо. Завдяки цьому такі продукти набули значної популярності серед музикантів та добре зарекомендували себе у професійних студійних умовах. Вони дозволяють досягати високого рівня гнучкості в налаштуванні звучання та реалізовують практично необмежені можливості для творчих експериментів.

Разом із тим виявлено низку суттєвих обмежень, які знижують ефективність використання подібних рішень у повсякденній практиці. Зокрема, їхні інтерфейси часто виявляються перевантаженими великою кількістю параметрів і меню, що ускладнює процес швидкого освоєння для новачків і навіть створює труднощі для досвідчених користувачів. Багато корисних функцій доступні виключно у платних версіях, що обмежує можливості музикантів із невеликим бюджетом. Додатковою проблемою є те, що мобільні версії таких програм поступаються за зручністю використання настільним аналогам і не завжди відповідають потребам сучасних користувачів, які прагнуть працювати у будь-якому місці та в будь-який час. Також важливою виявилася відсутність інтеграції зі штучним інтелектом, який міг би значно спростити процес створення і налаштування пресетів, пропонуючи користувачеві автоматичні рекомендації та допомогу у виборі оптимальних параметрів.

Таким чином, запропонований «розумний» гітарний процесор позиціонується як інноваційне рішення, яке не тільки успадковує найкращі практики перевірених продуктів, а й виходить за їхні межі завдяки спрощеному інтерфейсу, мобільній оптимізації та застосуванню сучасних AI-технологій. Це дозволить забезпечити користувачів інструментом нового покоління, що поєднує професійні можливості зі зручністю і доступністю.

# **ПОСТАНОВКА ЗАДАЧІ**

Метою цього проєкту є створення клієнтської частини мобільного застосунку «розумного» гітарного процесора, призначеного для роботи з ефектами, формуванням індивідуальних пресетів та інтеграцією з допоміжними сервісами на основі штучного інтелекту.

Основна увага приділяється забезпеченню інтуїтивності та простоти використання, аби програмою могли користуватися як початківці, так і досвідчені музиканти. Інтерфейс має бути зручним, адаптивним до різних розмірів екранів і оптимізованим під щоденну роботу в умовах репетицій або виступів.

Клієнтська частина виконує функції відображення даних, керування пресетами та взаємодії з сервером через REST API. Вона повинна забезпечувати динамічне редагування параметрів ефектів, збереження налаштувань користувача та швидку обробку запитів без затримок.

Основними функціональними задачами клієнтської частини є:

* реалізацію вибору та підключення ефектів до створюваних пресетів;
* редагування параметрів ефектів у зручному форматі;
* взаємодію з REST API для отримання даних про доступні ефекти та збереження налаштувань на сервер;
* відтворення аудіотреків із можливістю застосування створених пресетів;
* підтримку зворотного зв’язку у вигляді повідомлень про успішне виконання дій чи помилки;
* адаптивність і зрозумілу навігацію у межах мобільного інтерфейсу;

Технологічна реалізація базується на використанні мови програмування Kotlin у середовищі Android. Для взаємодії з серверною частиною застосовується бібліотека Retrofit, що забезпечує обмін даними через REST API. Асинхронна робота організована за допомогою Kotlin Coroutines, а для локального збереження даних (наприклад, пресетів або налаштувань) може використовуватися Room. Відтворення аудіо реалізується на основі MediaPlayer або ExoPlayer, що дозволяє користувачам працювати з треками безпосередньо в інтерфейсі програми.

У межах курсової роботи ставиться задача побудувати мобільний застосунок, який поєднує простоту використання та гнучкість у налаштуваннях, забезпечуючи музикантам зручний інструмент для створення та організації гітарних пресетів.

# **АРХІТЕКТУРА ТА ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

## UML проєктування ПЗ

Мобільний застосунок для роботи з гітарними ефектами та створення пресетів проєктувався із застосуванням UML-методології, що дозволило формалізувати основні сценарії взаємодії користувача та внутрішню структуру системи. Архітектура клієнтської частини побудована за модульним принципом, що спрощує розширення та подальшу підтримку.

На етапі планування була створена діаграма випадків використання (див. рис. 3.1), яка відображає основні дії користувача: додавання нового ефекту до пресету, зміна параметрів (наприклад, delay, mix, feedback), відтворення або зупинка треку, збереження та редагування пресетів, перегляд списку наявних налаштувань. Окремим сценарієм передбачено взаємодію із серверною частиною через REST API для отримання даних про ефекти та збереження користувацьких налаштувань.

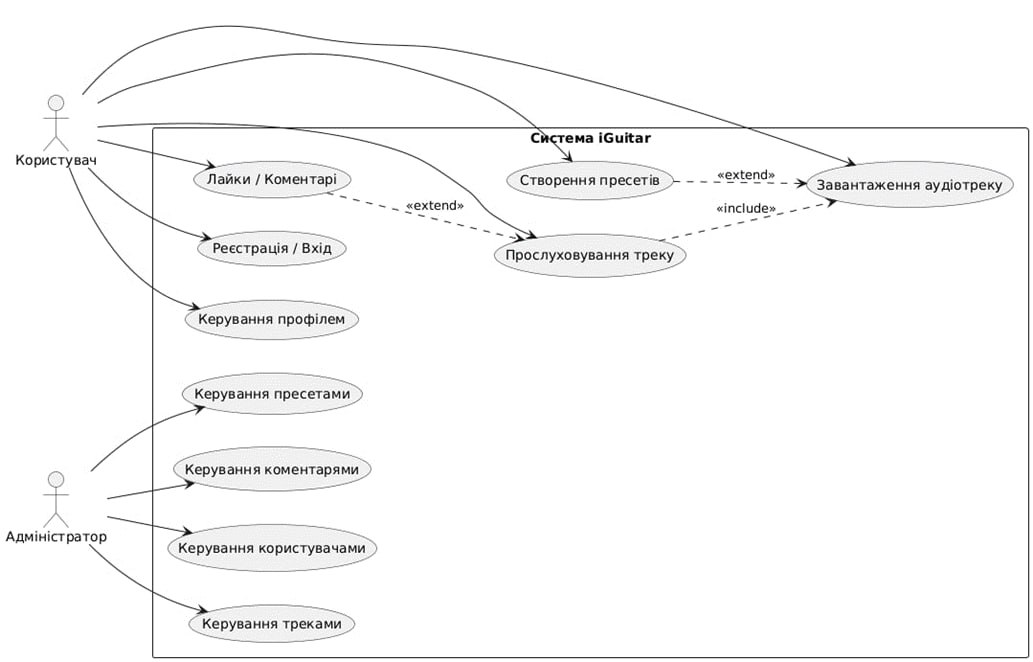


Рисунок 3.1 – Діаграма випадків використання користувача

у клієнтському інтерфейсі

Для відображення внутрішньої логіки мобільного застосунку була створена діаграма компонентів (рис. 3.2). Вона показує основні модулі системи та їхню взаємодію: створення і редагування пресетів, редагування параметрів ефектів, відтворення треків, управління збереженими пресетами, роботу з сервером через API та локальне збереження даних.



Рисунок 3.2 – Діаграма компонентів інтерфейсу користувача

Для відображення архітектури мобільного застосунку було побудовано діаграму розгортання (див. рис. 3.3). Вона демонструє, як клієнтська частина у вигляді Android-додатку встановлюється на мобільному пристрої та взаємодіє із серверною частиною через REST API. Сервер, у свою чергу, забезпечує обробку запитів, збереження даних у базі даних та надання необхідних відповідей клієнту. Такий підхід дозволяє поєднати локальні можливості пристрою (редагування пресетів, відтворення треків) із централізованим керуванням і синхронізацією даних на сервері.

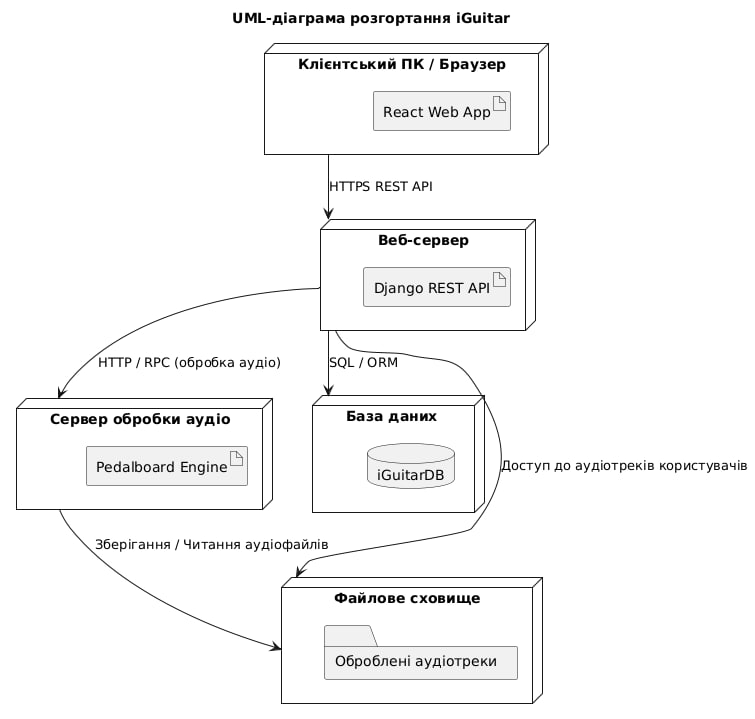


Рисунок 3.3 – Діаграма розростання

UML-діаграми дозволили систематизувати логіку роботи мобільного застосунку, окреслити ключові сценарії використання та визначити структуру компонентів таким чином, щоб забезпечити простоту підтримки, масштабування і повторне використання елементів у майбутньому розвитку системи.

## Проектування архітекрути ПЗ

Архітектура клієнтської частини мобільного застосунку побудована з урахуванням вимог до зручності використання, масштабованості та повторного застосування основних модулів. За основу взято нативний підхід розробки під Android на мові Kotlin, що забезпечує високу продуктивність, інтеграцію з системними можливостями та підтримку сучасних інструментів.

Застосунок реалізовано із використанням сучасного технологічного стеку, що включає:

* Kotlin як основну мову розробки;
* Android Jetpack Components (Activity, Fragment, ViewModel, LiveData) для організації життєвого циклу та роботи з UI;
* Retrofit для взаємодії з сервером через REST API;
* Zustand для управління глобальним станом (вибір активного документа, користувач, тема оформлення тощо);
* Room / SharedPreferences для локального збереження пресетів і налаштувань користувача;
* ExoPlayer для відтворення треків із застосованими ефектами;
* Material Design для побудови адаптивного та сучасного інтерфейсу.

Архітектурно застосунок умовно поділено на кілька рівнів:

* рівень представлення (UI) містить компоненти інтерфейсу, зокрема Sidebar, Editor, DocumentCard, WorkspaceMenu, AIContextMenu, UserMenu, а також глобальні компоненти на кшталт Layout, ThemeSwitcher чи Notifications;
* UI-рівень представлений активностями та фрагментами, зокрема CreatePresetActivity для створення та редагування пресетів, PresetListActivity для перегляду та вибору збережених наборів, а також EffectEditor для налаштування параметрів ефектів;
* рівень логіки реалізовано через ViewModel та локальні сховища, які відповідають за управління станом користувацьких даних, відстеження змін і синхронізацію з сервером;
* рівень взаємодії з API побудований на основі Retrofit і містить сервіси для отримання, створення й оновлення пресетів;
* Модуль відтворення реалізовано за допомогою PlayerModule, що відповідає за роботу з треками, відтворення з ефектами та управління відтворенням у реальному часі;

Особливу увагу приділено адаптивності інтерфейсу. Усі основні екрани оптимізовано під різні розміри дисплеїв і орієнтації, а налаштування теми зберігаються в локальному сховищі, що забезпечує персоналізацію досвіду користувача.

Запропонована архітектура забезпечує модульність, гнучкість та простоту підтримки, дозволяючи у майбутньому розширювати функціональність мобільного застосунку без суттєвих змін у його базовій структурі.

## Проєктування структури зберігання даних

У розробленому застосунку дані зберігаються як локально, так і на віддаленому сервері: у локальному сховищі пристрою фіксується токен авторизації та індивідуальні налаштування користувача, що забезпечує швидке відновлення сесії без повторного входу, тоді як основні сутності, такі як пресети, ефекти та аудіотреки, передаються і зберігаються на сервері за допомогою REST API, реалізованого через бібліотеку Retrofit; обмін відбувається за допомогою спеціалізованих моделей даних, які описують параметри пресету та ефектів, а користувацький інтерфейс, побудований на основі XML-розмітки, забезпечує інтеграцію з цими механізмами і дозволяє створювати та редагувати ресурси, поєднуючи переваги локального та централізованого зберігання.

## Приклади алгоритмів та методів

У клієнтській частині застосунку реалізовано низку рішень, що підвищують ефективність та зручність взаємодії з інтерфейсом, зокрема інтеграцію зі штучним інтелектом, централізоване управління станом, кешування запитів і локальне збереження параметрів користувача; інтеграція з AI дозволяє виділяти текст у редакторі та викликати дії, наприклад, «доповнити» або «перекласти», які обробляються через асинхронну функцію executeAction, де запит надсилається на сервер, а результат автоматично вставляється у документ, наприклад:

**const executeAction = async (actionId: string, action: () => Promise<string | string[]>) => {**

**if (!selectedText.trim() || isLoading) return**

**setIsLoading(actionId)**

**try {**

**const result = await action()**

**const textResult = Array.isArray(result) ? result.join('\n• ') : result**

**if (['improve', 'translate', 'fix'].includes(actionId)) {**

**onReplaceText(Array.isArray(result) ? '• ' + textResult : textResult)**

**} else {**

**onInsertText(Array.isArray(result) ? '• ' + textResult : textResult)**

**}**

**} catch (error) {**

**console.error(`AI ${actionId} failed:`, error)**

**} finally {**

**setIsLoading(null)**

**}**

**}**

Для управління глобальним станом застосунку використовується бібліотека Zustand, де зберігаються вибраний робочий простір, активний документ і режим теми, наприклад:

**interface AppState {**

**export const useAppStore = create<AppState>()(**

**persist(**

**setCurrentWorkspace: (workspace: Workspace | null) => { set({ currentWorkspace: workspace }) },**

**setTheme: (theme: 'light' | 'dark') => { set({ theme }); applyTheme(theme) },**

**{ name: 'app-storage' }**

**)**

**)**

Кешування серверних запитів реалізовано через React Query, що дозволяє отримувати дані лише один раз і повторно їх не запитувати при навігації, наприклад:

**const useFavoriteDocuments = () => {**

**return useQuery({**

**queryKey: ['users', 'documents', 'favorites'],**

**queryFn: usersApi.getFavoriteDocuments,**

**staleTime: 2 \* 60 \* 1000,**

**})**

**}**

Локальне збереження теми інтерфейсу через localStorage забезпечує збереження індивідуальних налаштувань між сесіями, а при завантаженні сторінки відповідна тема застосовується до HTML-документа; усі ці рішення демонструють сучасний підхід до розробки SPA-застосунків, поєднуючи асинхронну обробку, інтеграцію AI, централізоване управління станом і персоналізацію інтерфейсу, що дозволяє досягти високої продуктивності, стабільності та зручності користувацького досвіду.

.

## Створення UI/UX дизайну системи

Одним із ключових етапів розробки мобільного застосунку стало проєктування користувацького інтерфейсу та взаємодії з ним. Основною метою було створення зручного середовища для роботи з гітарними ефектами, яке поєднує простоту використання з гнучкими можливостями налаштування.

Дизайн інтерфейсу побудовано на принципах мінімалізму, чіткої ієрархії екранів і адаптивності під різні розміри дисплеїв. Основні елементи було розташовано в центральній частині екрана, що забезпечує легку доступність навіть під час живого виконання. Для користувача передбачено головні екрани: список збережених пресетів, модуль створення та редагування нового набору ефектів, а також програвач треків із можливістю застосування обраних налаштувань (див. рис. 3.4).

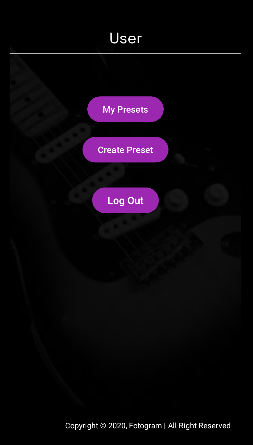


Рисунок 3.4 – Загальний вигляд застосунку

Особлива увага приділена модулю редагування ефектів. Він реалізований у вигляді динамічних блоків, кожен з яких відображає параметри конкретного ефекту. Користувач може змінювати значення за допомогою слайдерів, перемикачів та спіннерів, що дозволяє швидко отримувати бажане звучання. У межах UX було закладено підтримку миттєвого зворотного зв’язку: усі зміни параметрів одразу відображаються у звуці, що спрощує процес експериментування (див. рис. 3.5).

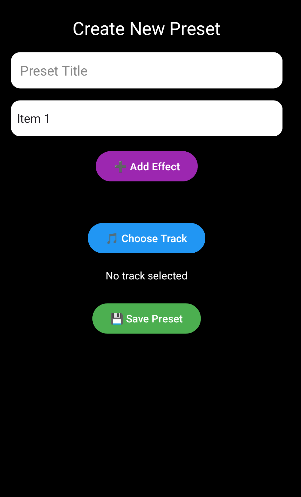


Рисунок 3.5 – Редагування ефеків

Важливим елементом UX є система візуального зворотного зв’язку: застосунок інформує користувача про успішне виконання дій, помилки чи інші події за допомогою сповіщень, індикаторів стану та плавних анімацій. Це дає змогу швидко орієнтуватися в процесі роботи та знижує когнітивне навантаження.

Розроблений дизайн забезпечує простоту та зрозумілість навіть для недосвідчених користувачів, але водночас дозволяє реалізувати широкий спектр налаштувань. Такий підхід робить застосунок зручним у щоденному використанні як під час репетицій, так і в умовах сцени.

## Опис підготовки даних

У мобільному застосунку iGuitar підготовка даних починається з отримання інформації із серверної частини через REST API. Ефекти завантажуються у форматі JSON, після чого вони перетворюються у модельні об’єкти, що використовуються в інтерфейсі. Наприклад, при завантаженні списку ефектів клієнт виконує асинхронний запит до API, а отримані назви відображаються у випадаючому списку Spinner.

**lifecycleScope.launch {**

**val response = RetrofitInstance.api.getEffects()**

**if (response.isSuccessful) {**

**val effects = response.body() ?: return@launch**

**val effectNames = effects.map { it.name ?: "Unnamed" }**

**spinner.adapter = ArrayAdapter(**

**this@CreatePresetActivity,**

**android.R.layout.simple\_spinner\_item,**

**effectNames**

**)**

**}**

**}**

Коли користувач завершує налаштування ефектів і вводить назву пресета, всі дані збираються у структуру PresetRequest і надсилаються на сервер. Таким чином, процес підготовки даних охоплює завантаження параметрів ефектів із API, їх трансформацію у придатний для відображення вигляд, інтерактивну зміну значень у межах інтерфейсу та формування підсумкового запиту для збереження пресета.

**btnSave.setOnClickListener {**

**val title = etTitle.text.toString().trim()**

**val preset = PresetRequest(title, selectedEffects)**

**lifecycleScope.launch {**

**val res = RetrofitInstance.api.createPreset(preset)**

**if (res.isSuccessful) {**

**Toast.makeText(this@CreatePresetActivity, "Preset saved", Toast.LENGTH\_SHORT).show()**

**finish()**

**}**

**}**

**}**

Таким чином, клієнтська частина застосунку забезпечує послідовну взаємодію з API, завантаження та перетворення даних у зручний для користувача формат, можливість динамічного налаштування ефектів у межах інтерфейсу та формування підсумкових запитів для збереження пресетів на сервері.

# **ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ**

## Вибір архітектури клієнтської частини

Клієнтська частина системи реалізована у вигляді нативного мобільного застосунку для Android, розробленого мовою Kotlin. Архітектурно програма побудована на основі принципів розділення логіки та інтерфейсу: в окремих пакетах винесені моделі даних, сервіси для роботи з API та активності з елементами користувацького інтерфейсу.

Для організації взаємодії з сервером використовується бібліотека Retrofit разом із конвертером Gson, що забезпечує виконання асинхронних HTTP-запитів та обробку результатів у форматі JSON. Отримані дані трансформуються у модельні об’єкти (Effect, PresetRequest), після чого відображаються у компонентах інтерфейсу, зокрема у випадаючих списках (Spinner) та формах для введення параметрів.

Усі мережеві виклики виконуються в межах корутин Kotlin з використанням lifecycleScope, що дозволяє зберігати стабільність роботи інтерфейсу та уникати блокування головного потоку. Для збереження та надсилання даних про пресети передбачено окремі класи-запити, які збирають налаштування користувача та надсилають їх на сервер.

Завдяки такій архітектурі клієнтська частина має зрозумілу структуру, легко підтримується, дозволяє масштабувати функціонал і забезпечує плавну роботу застосунку навіть при частій взаємодії з API.

## Вибір бібліотек та технологій

Для реалізації клієнтської частини мобільного застосунку iGuitar було обрано набір перевірених інструментів та бібліотек, які забезпечують стабільну роботу, простоту інтеграції та підтримку сучасних підходів до розробки Android-застосунків. Основними критеріями вибору стали продуктивність, активна спільнота, легкість налаштування та сумісність із Kotlin.

Основу розробки становить Android SDK у поєднанні з Kotlin, що гарантує типобезпечність, зручність роботи з корутинами та сумісність із сучасними інструментами Android-екосистеми. Для архітектурного розділення логіки та UI використано стандартний підхід з Activity, ViewBinding та окремими моделями даних.

Для роботи з API застосовано бібліотеку Retrofit 2 разом із конвертером Gson, які забезпечують просте оголошення HTTP-запитів, перетворення JSON-даних у модельні об’єкти та зручну інтеграцію з асинхронним кодом. Виклики виконуються у Kotlin coroutines з використанням lifecycleScope, що дозволяє зберігати стабільність інтерфейсу та уникати блокувань головного потоку.

Візуальні компоненти реалізовані з використанням стандартних елементів Android UI (Spinner, EditText, RecyclerView), а для їх гнучкого налаштування застосовується Material Components for Android, що забезпечує сучасний дизайн і відповідність гайдлайнам Material Design.

Для локального збереження даних використовується Room як ORM-рішення поверх SQLite. Це дозволяє кешувати налаштування користувача, збережені пресети та історію вибраних ефектів.

Роботу з асинхронними операціями значно спрощує Kotlin Flow, який дає змогу будувати реактивні ланцюжки подій, наприклад, при оновленні списку ефектів у Spinner або при збереженні нового пресета.

Для логування подій та налагодження використовується Timber, що дозволяє централізовано керувати логами та зберігати їх у зручному форматі.

Такий набір інструментів обрано невипадково: Retrofit + Gson забезпечують зручну роботу з API, Coroutines + Flow — стабільність асинхронних процесів, а Room — ефективне кешування даних. Використання Material Components робить інтерфейс сучасним та інтуїтивно зрозумілим, а Timber спрощує процес налагодження.

Сукупність цих технологій гарантує продуктивність, масштабованість та адаптивність клієнтської частини iGuitar, що дозволяє розширювати її функціонал без значних змін у базовій архітектурі.

## Організація навігації та управління станом у мобільному застосунку

Клієнтська частина мобільного застосунку iGuitar побудована за принципом багатосторінкової архітектури з використанням Activity, що дозволяє організувати зрозумілу маршрутизацію між основними екранами без складних переходів. Переміщення користувача між інтерфейсами реалізовано через Intent, що забезпечує передачу параметрів і збереження контексту роботи. Наприклад, після успішної автентифікації користувач перенаправляється з LoginActivity у UserActivity, де відображається його профіль та доступ до функцій створення і перегляду пресетів:

**val intent = Intent(this@LoginActivity, UserActivity::class.java)**

**intent.putExtra("username", username)**

**startActivity(intent)**

**finish()**

Доступ до певних частин інтерфейсу обмежується станом авторизації. Для цього застосовується локальне сховище SharedPreferences, у якому зберігається токен користувача. Якщо дані авторизації відсутні, користувач автоматично повертається на екран входу:

**val prefs = getSharedPreferences("auth", MODE\_PRIVATE)**

**val token = prefs.getString("token", null)**

**if (token == null) {**

**startActivity(Intent(this, LoginActivity::class.java)**

**finish()**

**}**

Глобальний стан застосунку організовано через поєднання локального сховища та внутрішніх моделей даних. У CreatePresetActivity використовується список обраних ефектів, який формується у пам’яті додатку та передається на сервер лише після натискання кнопки збереження. Завдяки цьому користувач має можливість редагувати параметри ефектів у реальному часі, а кінцевий результат зберігається централізовано одним запитом.

Крім того, застосунок підтримує відновлення стану між сесіями: дані користувача залишаються у сховищі після перезапуску програми, що робить процес взаємодії більш зручним. Таким чином, маршрутизація та керування станом у iGuitar реалізовані на рівні Android-архітектури, забезпечуючи простоту переходів між екранами, перевірку авторизації та збереження ключової інформації без надмірного дублювання логіки.

## Робота з редактором пресетів

Редактор пресетів є ключовим компонентом мобільного застосунку iGuitar. Він реалізований у вигляді окремої активності (CreatePresetActivity), яка дозволяє користувачам формувати власні звукові налаштування на основі ефектів. Інтерфейс редактора складається з кількох основних елементів: вибору треку, додавання ефектів, редагування параметрів та збереження результату.

**for ((key, value) in effect.default\_parameters) {**

**if (value is Number) {**

**val seek = SeekBar(this@CreatePresetActivity).apply {**

**max = 100**

**progress = (value.toDouble() \* 100).toInt()**

**setOnSeekBarChangeListener(object SeekBar.OnSeekBarChangeListener {**

**override fun onProgressChanged(seekBar: SeekBar?, progress: Int, fromUser: Boolean) {**

**val newVal = progress / 100.0**

**paramLabel.text = "$key: %.2f".format(newVal)**

**params[key] = newVal**

**}**

**override fun onStartTrackingTouch(seekBar: SeekBar?) {}**

**override fun onStopTrackingTouch(seekBar: SeekBar?) {}**

**})**

**}**

**container.addView(seek)**

**}**

**}**

Редагування параметрів виконується миттєво, і користувач одразу бачить, як змінюються значення. Це забезпечує інтерактивність процесу і спрощує налаштування складних ефектів. Усі внесені зміни зберігаються у локальному стані застосунку у вигляді списку обраних ефектів. Лише після натискання кнопки «Save Preset» сформований пресет надсилається на сервер одним запитом:

**val preset = PresetRequest(title, selectedEffects)**

**val res = RetrofitInstance.api.createPreset(preset)**

Редактор також підтримує вибір треку з локальної пам’яті телефону. Для цього використовується стандартний інтент ActivityResultContracts.GetContent, який дозволяє користувачеві обрати будь-який аудіофайл. Інформація про трек відображається у вигляді текстового елементу без необхідності перезавантаження активності.

Таким чином, редактор пресетів у iGuitar поєднує простоту мобільного інтерфейсу з гнучкістю налаштувань, забезпечуючи користувачу можливість створювати унікальні звукові комбінації. Використання асинхронної взаємодії з сервером дозволяє швидко отримувати список ефектів та зберігати результати без затримок, а динамічне формування інтерфейсу гарантує масштабованість і легке розширення у майбутньому.

**const executeAction = async (actionId: string, action: () => Promise<string | string[]>) => {**

**if (!selectedText.trim() || isLoading) return**

**setIsLoading(actionId)**

**try {**

**const result = await action()**

**const textResult = Array.isArray(result) ? result.join('\n• ') : result**

**if (['improve', 'translate', 'fix'].includes(actionId)) {**

**onReplaceText(Array.isArray(result) ? '• ' + textResult : textResult)**

**toast.success(`Text ${actionId}d successfully`)**

**} else {**

**onInsertText(Array.isArray(result) ? '• ' + textResult : textResult)**

**toast.success(`${actionId.charAt(0).toUpperCase() + actionId.slice(1)} inserted`)**

**}**

**onClose()**

**} catch (error) {**

**console.error(`AI ${actionId} failed:`, error)**

**toast.error(`Failed to ${actionId} text`)**

**}**

**finally { setIsLoading(null)}**

**}**

Функція executeAction викликає відповідний запит (наприклад, переклад, доповнення або рефакторинг тексту), отримує результат від AI (див. рис. 4.2) та вставляє його у документ, де був курсор (див. рис. 4.3).

## Адаптація інтерфейсу під мобільний застосунок

Інтерфейс мобільного застосунку розроблено з урахуванням особливостей різних розмірів екранів смартфонів та планшетів. Для цього використовуються стандартні інструменти Android: XML-розмітка, атрибути layout\_weight, match\_parent, wrap\_content, а також ресурси у папках values, values-sw600dp тощо, що дозволяє змінювати пропорції елементів залежно від ширини екрана.

**<LinearLayout**

**android:layout\_width="match\_parent"**

**android:layout\_height="wrap\_content"**

**android:gravity="center"**

**android:orientation="vertical">**

**<Button**

**android:id="@+id/btn\_my\_presets"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="My Presets"**

**android:textColor="@android:color/white"**

**android:backgroundTint="@color/purple\_500"**

**android:layout\_marginTop="16dp" />**

**<Button**

**android:id="@+id/btn\_create\_preset"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Create Preset"**

**android:textColor="@android:color/white"**

**android:backgroundTint="@color/purple\_500"**

**android:layout\_marginTop="12dp" />**

**</LinearLayout>**

Крім базової розмітки, у застосунку використано стилі та теми Android, які зберігаються у файлах res/values/styles.xml. Це дозволяє задати єдиний вигляд для кнопок, текстових полів і заголовків на всіх екранах.

**<style name="AppTheme"**

**parent="Theme.MaterialComponents.DayNight.DarkActionBar">**

**<item name="android:textColor">#FFFFFF</item>**

**<item name="android:buttonStyle">@style/CustomButton</item>**

**</style>**

**<style name="CustomButton" parent="Widget.MaterialComponents.Button">**

**<item name="android:backgroundTint">#9C27B0</item>**

**<item name="android:textColor">#FFFFFF</item>**

**</style>**

Таким чином, інтерфейс застосунку є узгодженим і зручним на різних пристроях. Завдяки використанню ресурсів, стилів і атрибутів Android забезпечується масштабованість, повторне використання елементів і легка підтримка зовнішнього вигляду застосунку у майбутньому.

## Продуктивність та оптимізація

Для мобільного застосунку одним із ключових завдань є забезпечення стабільної швидкодії та плавності роботи інтерфейсу навіть на пристроях із обмеженими ресурсами. Оптимізація виконана як на рівні взаємодії з сервером, так і на рівні відображення інтерфейсу.

З боку мережевих операцій застосунок використовує Retrofit у поєднанні з корутинами Kotlin, що забезпечує асинхронне виконання запитів без блокування головного потоку. Отримані дані попередньо обробляються та кешуються локально, щоб знизити кількість повторних звернень до API. Для тимчасового зберігання використовується база Room або стандартний SharedPreferences, що дозволяє швидко відображати дані навіть у режимі обмеженого з’єднання.

**viewModelScope.launch {**

**try {**

**val response = api.getPresets()**

**if (response.isSuccessful) {**

**val presets = response.body() ?: emptyList(**

**presetDao.insertAll(presets)**

**\_uiState.value = UiState.Success(presets)**

**} else {**

**\_uiState.value = UiState.Error("Server error")**

**}**

**} catch (e: Exception) {**

**val cached = presetDao.getAll()**

**\_uiState.value = UiState.Success(cached)**

**}**

**}**

Для покращення відгуку інтерфейсу застосовується підхід оптимістичного оновлення: зміни відображаються на екрані відразу після дії користувача, а підтвердження зберігається у фоні. У разі помилки виконується відкат до попереднього стану. Це створює відчуття високої швидкодії навіть при повільному інтернет-з’єднанні.

Візуальна оптимізація досягається завдяки використанню RecyclerView для списків пресетів, що дозволяє переробляти елементи при прокрутці й економити пам’ять. Зображення завантажуються через бібліотеку Glide із вбудованим кешем і масштабуванням, щоб уникати затримок і надмірного використання ресурсів.

**Glide.with(context)**

**.load(preset.imageUrl)**

**.centerCrop()**

**.placeholder(R.drawable.placeholder)**

**.into(holder.imageView)**

Додатково застосовано механізм лінивого завантаження: ресурси та важкі екрани (наприклад, сторінка редагування пресету з плеєром) відкриваються лише тоді, коли користувач переходить до них. Це скорочує час початкового запуску застосунку та зменшує використання пам’яті.

Таким чином, оптимізація продуктивності мобільного застосунку базується на ефективній роботі з мережею, локальному кешуванні, оптимістичному оновленні та візуальній оптимізації. Це дозволяє досягти плавності, швидкого відгуку і комфортного використання навіть у складних умовах.

# **АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ**

## Порівняння результатів з початковими вимогами

У результаті розробки мобільного застосунку вдалося реалізувати основні функціональні вимоги, що були сформульовані на етапі постановки задачі (див. розділ 2). Інтерфейс вийшов інтуїтивно зрозумілим, підтримує зручну навігацію між екранами та забезпечує повноцінну взаємодію з даними, що отримуються через REST API.

Таблиця 5.1 – Відповідність функціональності початковим вимогам

|  |  |  |
| --- | --- | --- |
| № | Вимога | Реалізація |
| 1 | Реєстрація, вхід та вихід користувача | Реалізовано з використанням Retrofit та токен-автентифікації, стан зберігається у SharedPreferences |
| 2 | Навігація між основними екранами | Реалізовано через Android Navigation Component з підтримкою стеку переходів |
| 3 | Інтеграція з API для роботи з ефектами | Реалізовано через Retrofit: отримання списку ефектів та збереження змінених параметрів |
| 4 | Створення пресетів | Реалізовано в окремій активності з динамічним відображенням параметрів ефектів |
| 5 | Відображення сповіщень та повідомлень про помилки | Реалізовано через Toast та Snackbar у відповідних сценаріях |
| 6 | Відображення користувацького профілю | Реалізовано у UserActivity з підвантаженням даних із сервера |
| 7 | Адаптивність інтерфейсу для різних розмірів екранів | Забезпечено за допомогою ConstraintLayout та ресурсів для різних density |
| 8 | HTTP-взаємодія з сервером, обробка помилок | Реалізовано через Retrofit + Kotlin Coroutines з try/catch для винятків |
| 9 | Збереження налаштувань користувача | Реалізовано у SharedPreferences (авторизація, останні дії, кеш даних) |

Розроблений мобільний застосунок повністю відповідає поставленим цілям проєкту. Завдяки вибору сучасних технологій Android-розробки вдалося досягти високої швидкодії, зручності використання та готовності до подальшого розширення функціоналу.

## Аналіз ефективності прийнятих рішень

У процесі реалізації мобільного застосунку було прийнято низку архітектурних і технологічних рішень, які позитивно вплинули на стабільність, продуктивність та зручність підтримки системи.

Обрана архітектура MVVM дала змогу розділити бізнес-логіку та інтерфейс, що значно спростило тестування та подальший розвиток проєкту. Використання ViewModel та LiveData забезпечило реактивність інтерфейсу: зміни даних відображаються на екрані без зайвого дублювання код.

Для взаємодії з сервером застосовано Retrofit у поєднанні з Kotlin Coroutines, що забезпечило зручну організацію асинхронних запитів. Такий підхід зменшив кількість помилок, пов’язаних з багатопоточністю, і підвищив стабільність роботи застосунку навіть при нестабільному інтернет-з’єднанні.

Локальне збереження даних реалізовано через Room та SharedPreferences. Це дозволило зберігати налаштування користувача та кешувати результати запитів для роботи в офлайн-режимі. У результаті знизилось навантаження на мережу і прискорилось завантаження основних екранів.

Інтерфейс побудований з використанням ConstraintLayout та ресурсів для різних розмірів екранів, що забезпечило адаптивність та однаковий рівень зручності на смартфонах з різною діагоналлю.

Застосування принципів модульності та повторного використання компонентів дозволило зменшити обсяг коду та полегшити його підтримку. За результатами тестування прийняті рішення показали високу ефективність: застосунок швидко завантажується, стабільно працює навіть при великій кількості даних і залишається зрозумілим для користувача.

## Обмеження та недоліки

Попри досягнуту стабільність і реалізацію основних функцій мобільного застосунку, існують певні обмеження, які можуть бути вдосконалені в майбутньому. Одним із ключових факторів є залежність від серверної частини: збереження та синхронізація пресетів можливі лише за наявності інтернет-з’єднання, що ускладнює роботу користувачів у повністю офлайн-режимі. Хоча локальне кешування реалізоване, воно обмежене за обсягом і не передбачає повної синхронізації після відновлення мережі.

Ще одним аспектом є відтворення треків з ефектами. Поточна реалізація плеєра забезпечує базові можливості, проте оптимізація обробки аудіосигналу може покращити плавність відтворення при великій кількості одночасно застосованих ефектів.

Додатковим недоліком можна вважати обмежені можливості налаштувань інтерфейсу. Хоча інтерфейс адаптивний, він не містить розширених варіантів персоналізації, що може знизити зручність для окремих користувачів.

Подальша робота над цими напрямками сприятиме покращенню продуктивності, масштабованості та якості користувацького досвіду, роблячи застосунок більш гнучким та надійним у різних умовах використанн.

# **ВИСНОВКИ**

У процесі виконання комплексного курсового проєкту було створено повноцінну клієнтську частину мобільного застосунку для управління гітарними пресетами та звуковими ефектами. Основний акцент було зроблено на розробці інтуїтивно зрозумілого та адаптивного інтерфейсу, який відповідає сучасним вимогам продуктивності, зручності та масштабованості. Використання відповідних архітектурних підходів та перевірених бібліотек дозволило забезпечити швидку та стабільну роботу застосунку навіть при значному навантаженні на систему.

Користувацький інтерфейс було реалізовано з використанням Android SDK та мови Kotlin, що дало змогу побудувати гнучку модульну структуру та застосувати компонентний підхід у розробці. XML-розмітка дозволила створити адаптивний дизайн, який коректно відображається як на смартфонах, так і на планшетах. Управління станом і збереження даних реалізовано за допомогою локального кешування та інтеграції з API, що забезпечує баланс між роботою в онлайн- та офлайн-режимах.

Особливу увагу приділено роботі з пресетами та ефектами. Користувач отримує можливість додавати ефекти з дефолтними параметрами, змінювати їхні значення за допомогою інтерактивних елементів (наприклад, SeekBar), а також зберігати налаштування у власні пресети. У процесі редагування передбачена інтеграція з аудіоплеєром, що дає змогу в реальному часі прослуховувати зміни звучання. Передбачено також можливість завантаження треків із пам’яті пристрою та їх використання у зв’язці з ефектами, що значно розширює функціонал системи.

Важливим елементом проєкту стала система персоналізації інтерфейсу. Користувач може працювати у зручних умовах завдяки підтримці темного та світлого режимів, налаштування яких зберігаються між сесіями. Крім того, у застосунку реалізовано систему повідомлень та індикаторів, які інформують про успіх виконання дій або про помилки під час роботи з API. Це підвищує загальну стабільність і зрозумілість взаємодії.

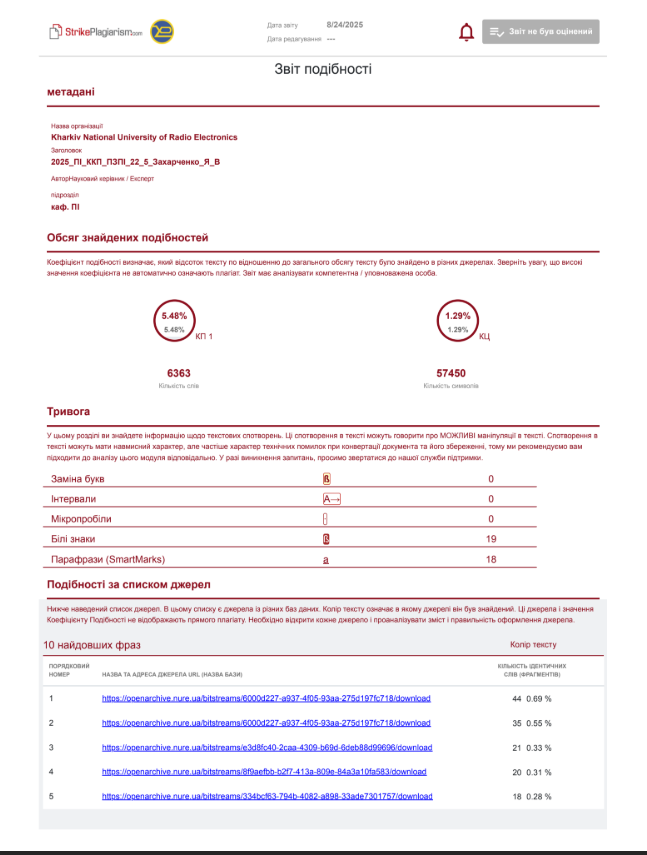
Завдяки комплексному підходу вдалося створити зручний, гнучкий і сучасний мобільний застосунок, що повністю відповідає функціональним вимогам і технічним очікуванням. Реалізоване рішення має перспективу подальшого розвитку: розширення бібліотеки ефектів, удосконалення офлайн-режиму та інтеграція із зовнішнім обладнанням (гітарними процесорами). Отриманий результат може використовуватись як самостійний інструмент для музикантів або як частина більшої екосистеми програмного забезпечення для роботи зі звуком.

# **ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ**

1. Android Developers – Official Documentation [Електронний ресурс]. – Режим доступу: [https://developer.android.com/docs](https://developer.android.com/docs%20) (дата звернення 15.05.2025)].
2. Kotlin Documentation [Електронний ресурс]. – Режим доступу: [https://kotlinlang.org/docs/home.html](https://kotlinlang.org/docs/home.html%20) (дата звернення 16.05.2025)
3. Android Jetpack – Android development tools [Електронний ресурс]. – Режим доступу[: https://developer.android.com/jetpack](:%20https:/developer.android.com/jetpack%20) (дата звернення 17.05.2025)
4. Material Design Guidelines [Електронний ресурс]. – Режим доступу: [https://m3.material.io/](https://m3.material.io/%20) (дата звернення 17.05.2025)
5. Retrofit – HTTP client for Android [Електронний ресурс]. – Режим доступу: <https://square.github.io/retrofit/>(дата звернення 18.05.2025)
6. OkHttp – HTTP client for Android and Java [Електронний ресурс]. – Режим доступу: <https://square.github.io/okhttp/>(дата звернення 19.05.2025)
7. Gson – A Java serialization/deserialization [Електронний ресурс]. – Режим доступу: <https://github.com/google/gson> (дата звернення 20.05.2025)
8. Android Architecture Components [Електронний ресурс]. – Режим доступу: <https://developer.android.com/topic/libraries/architecture> (дата звернення 01.06.2025)
9. GitHub репозиторій автора. Режим доступу: <https://github.com/Yarosslavo/2025-_KKP_PZPI-22-5_Zakharchenko_Y_V.git>

# **ДОДАТОК А**

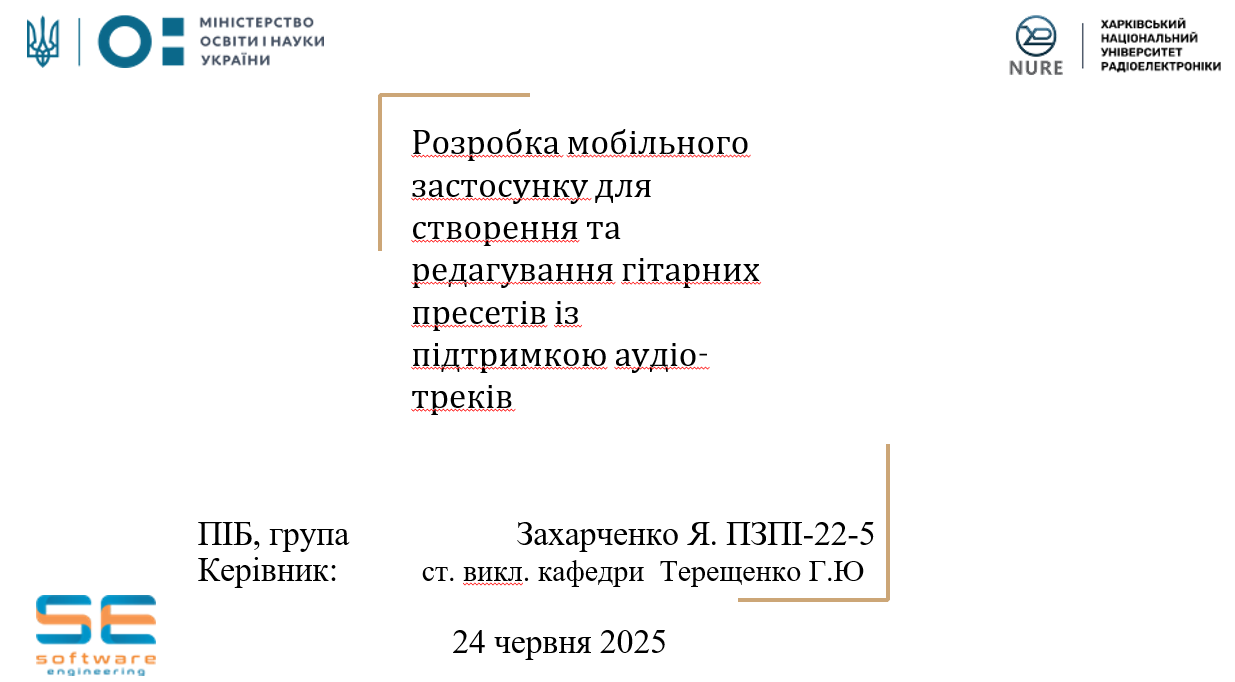
Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ

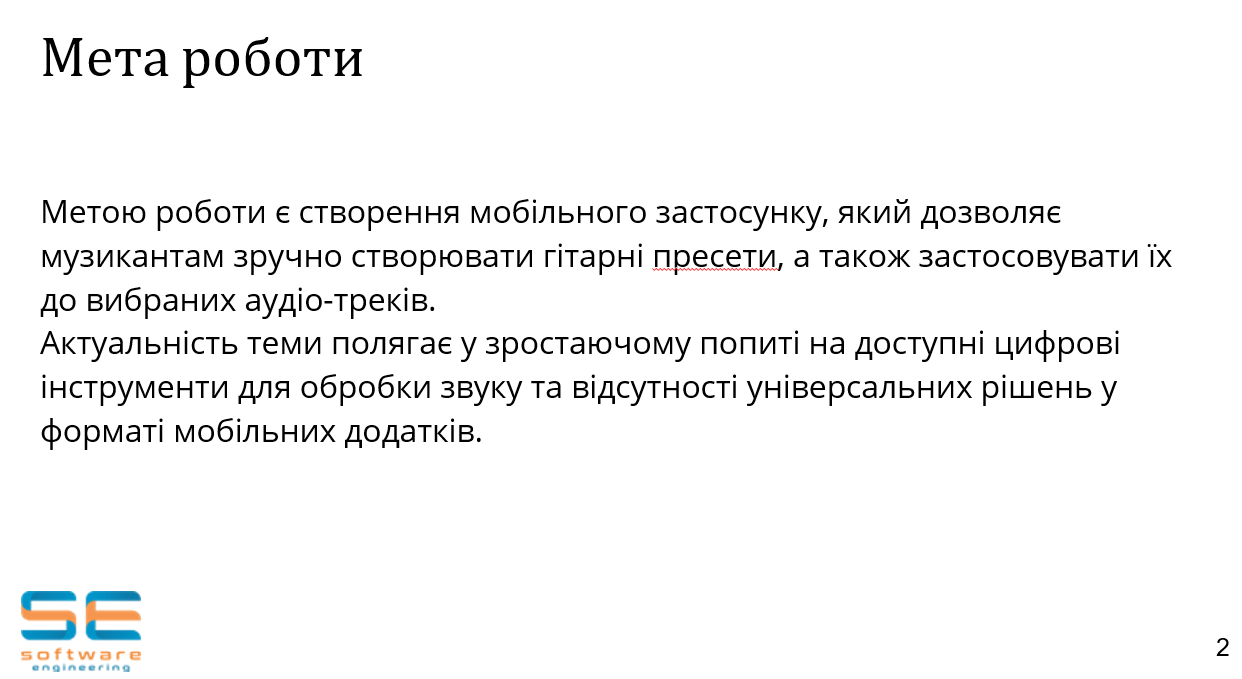


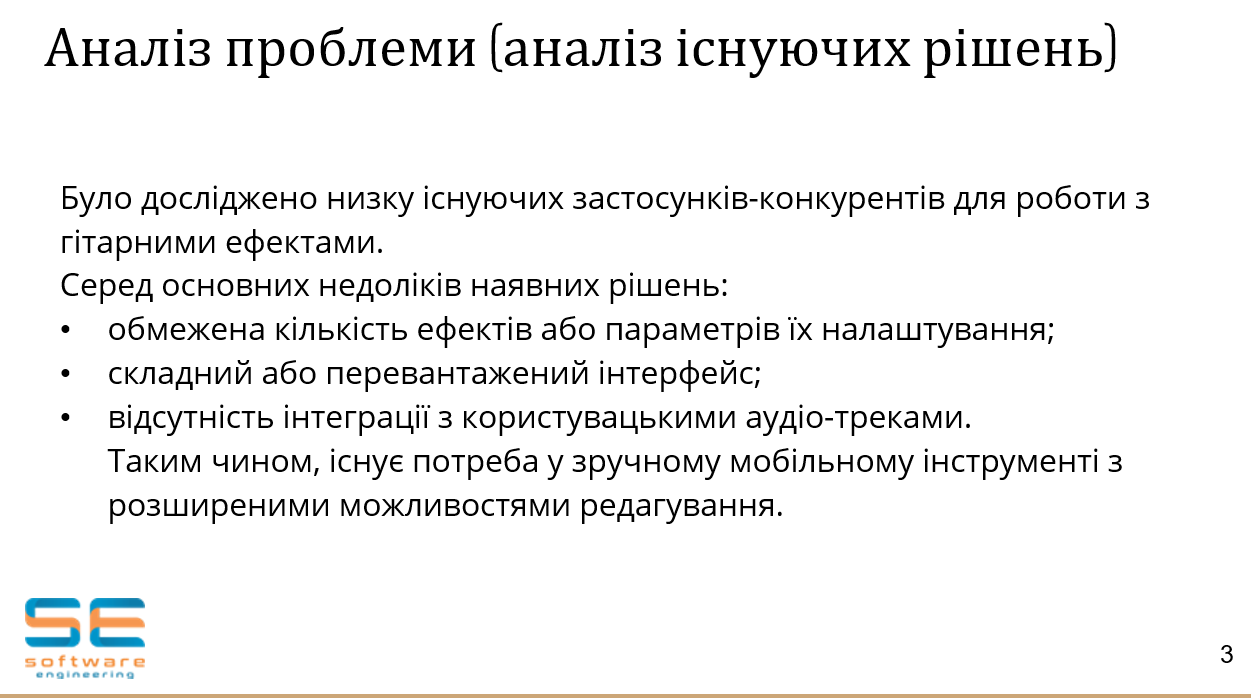


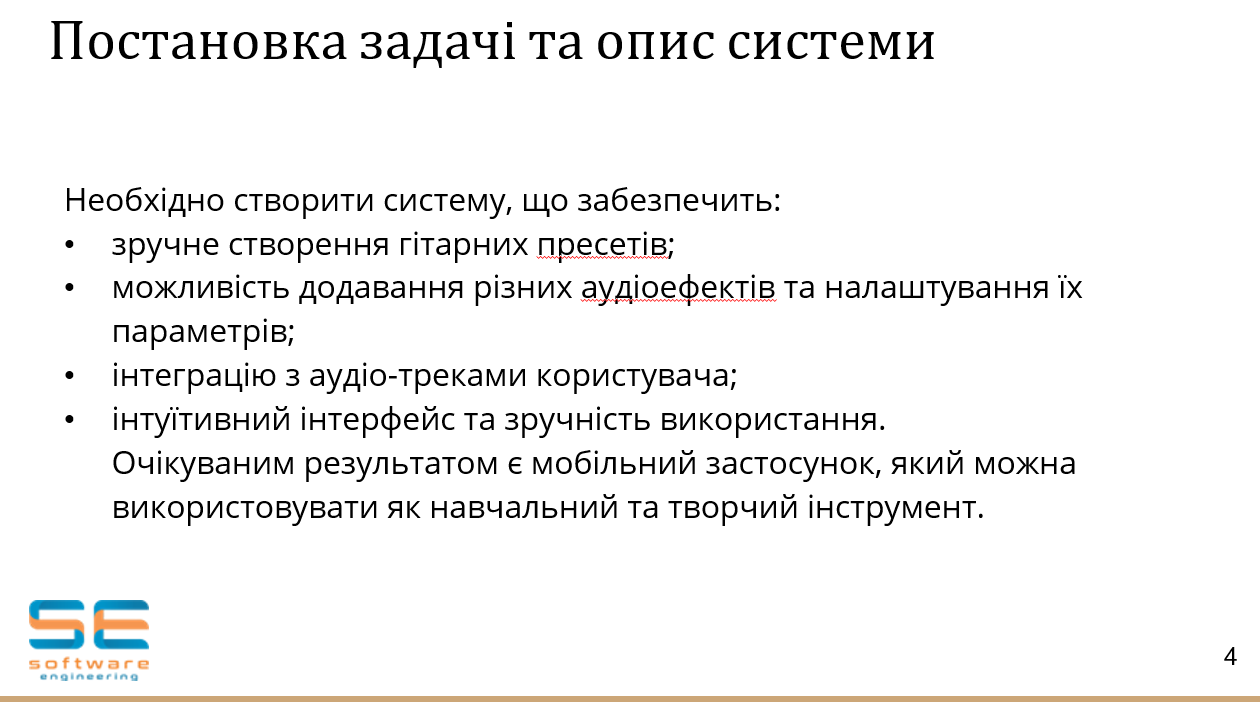
# **ДОДАТОК Б**

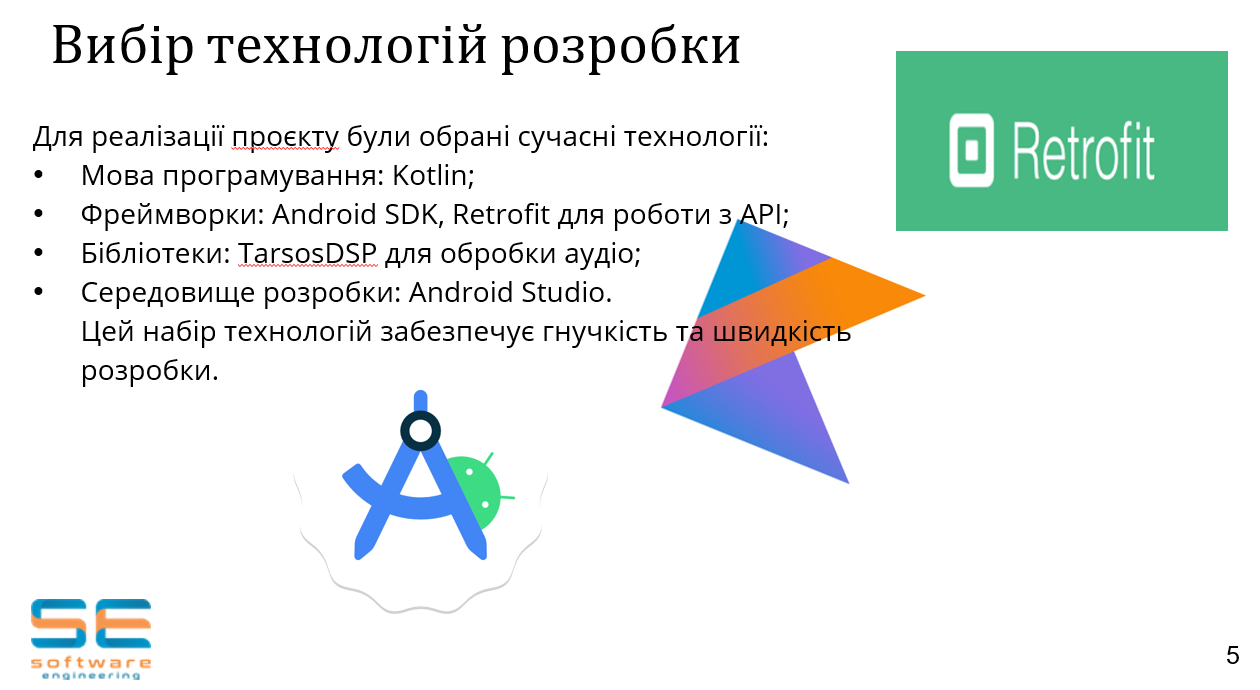
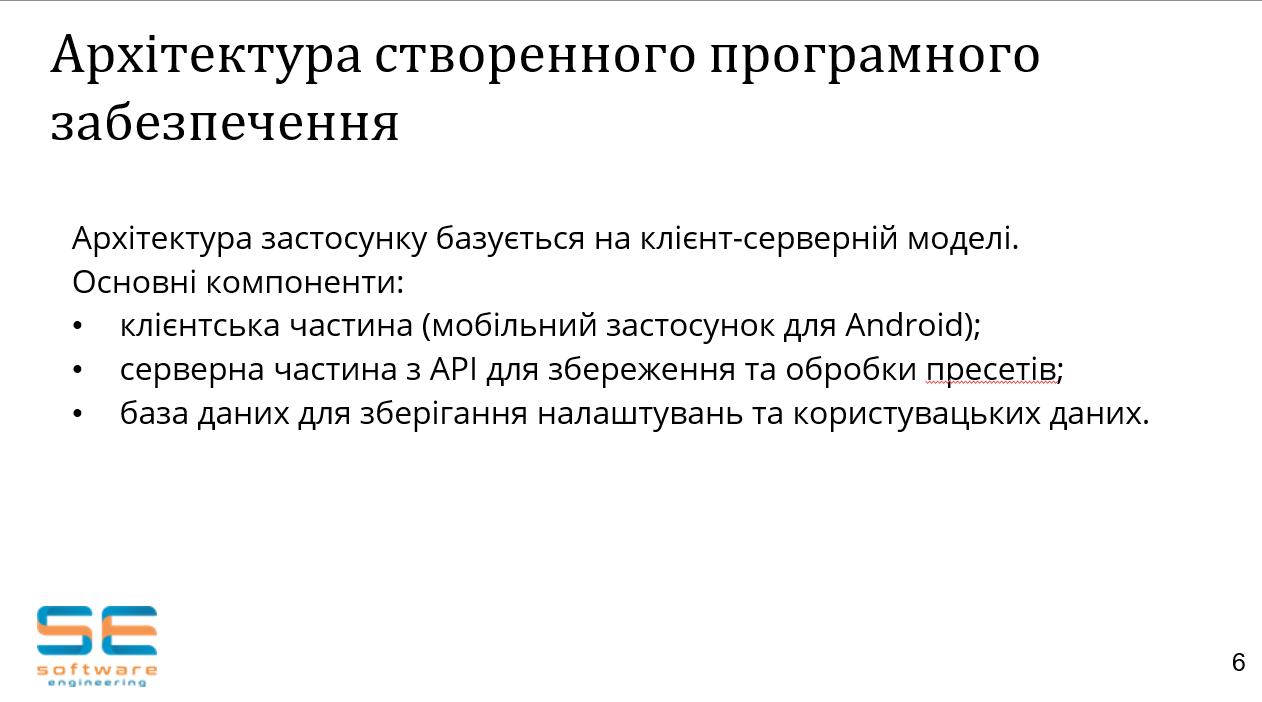
Слайди презентації

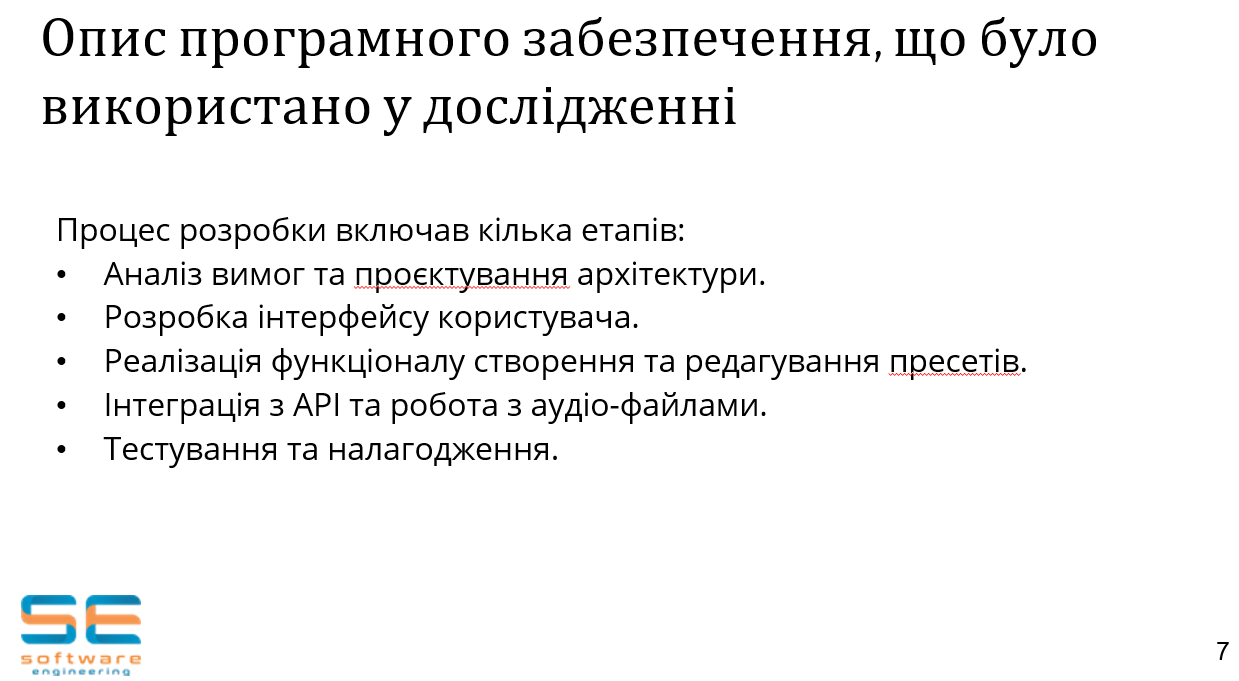


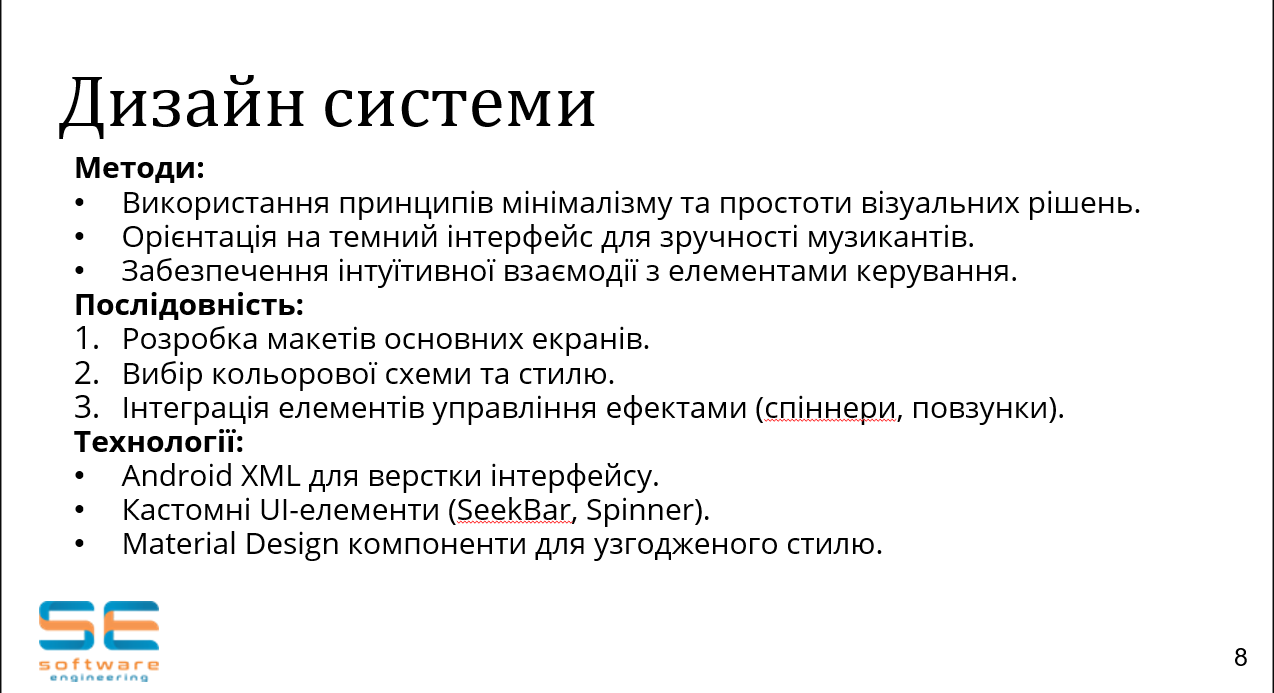


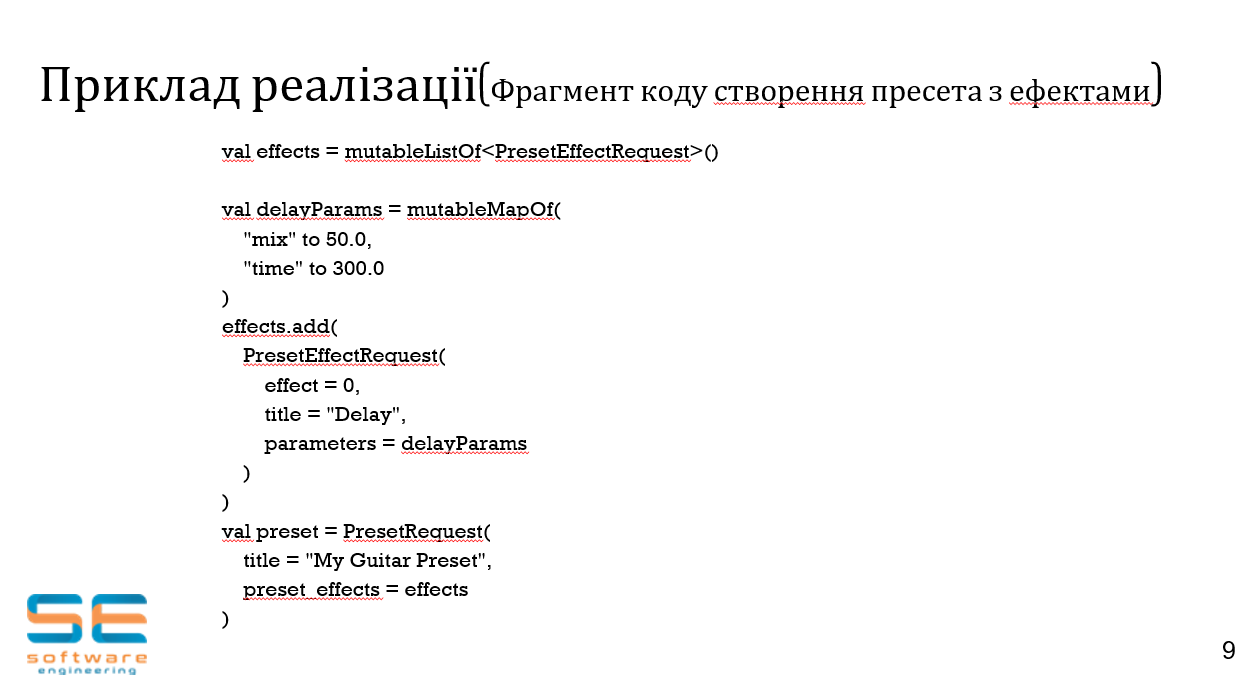


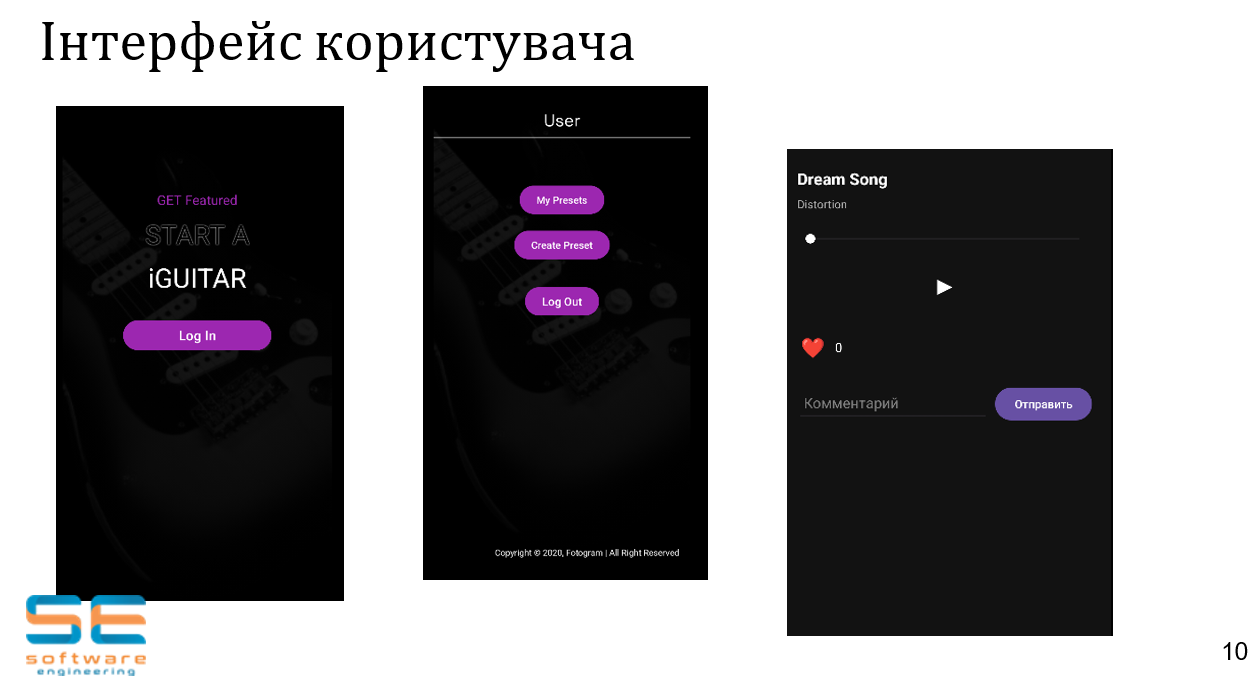


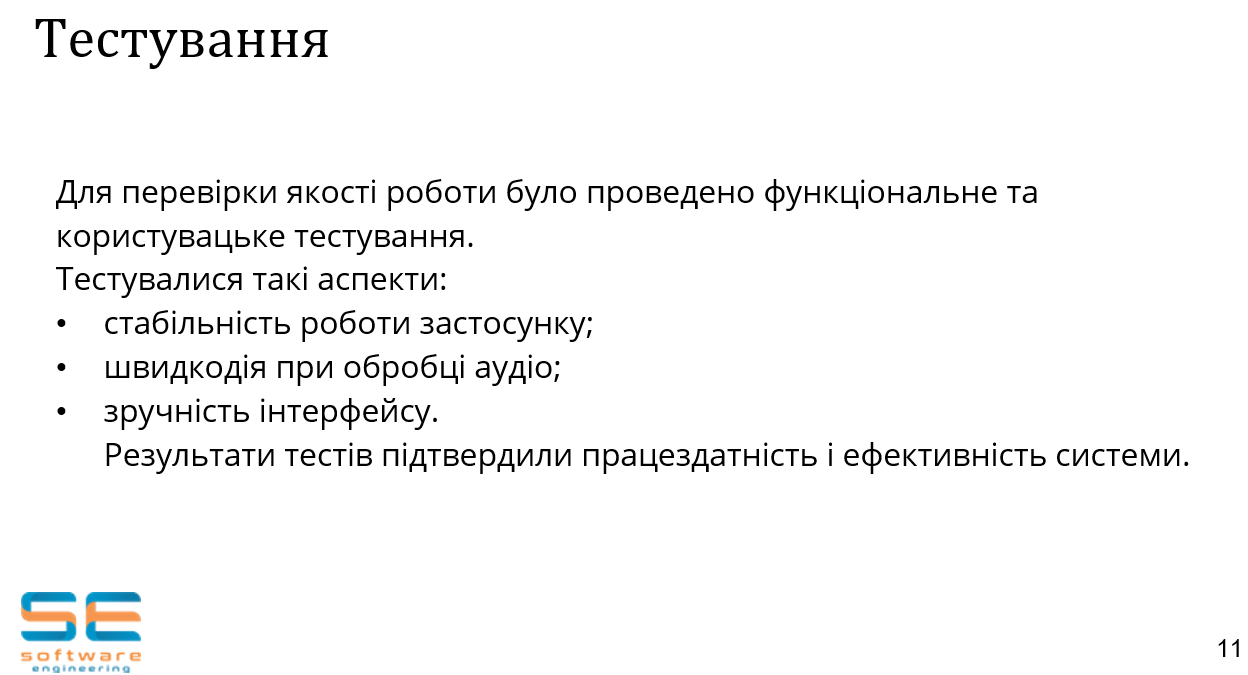
\ 

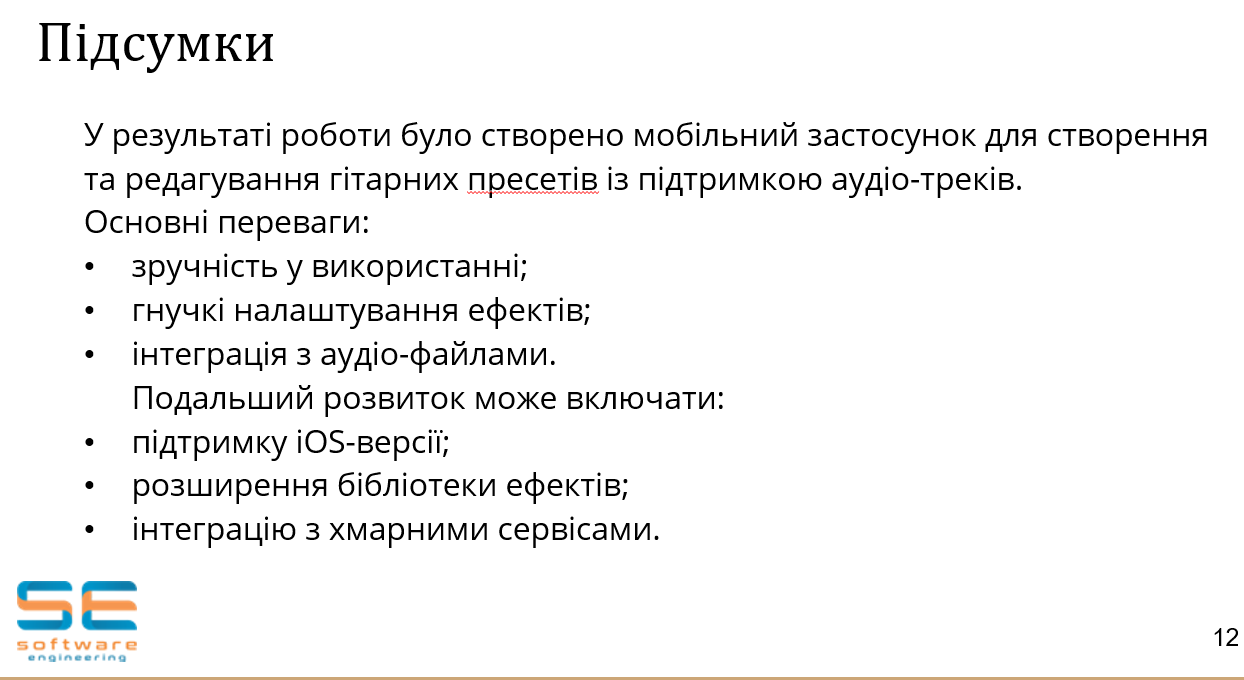












# ДОДАТОК В Специфікація програмного забезпечення

**Vision and Scope Document**

**for**

**«Розробка мобільного застосунку для створення та редагування гітарних пресетів із підтримкою аудіо-треків»**

**Version 1.0**

**Prepared by**

**Захарченко Ярослав Вадимович**

**ХНУРЕ**

**27.08.2025**

**ЗМІСТ**

[1 ВИМОГИ ДО БІЗНЕСУ 55](#_Toc207121700)

[1.1 Передумови 55](#_Toc207121701)

[1.2 Бізнес-можливість 55](#_Toc207121702)

[1.3 Бізнес-цілі та критерії успіху 55](#_Toc207121703)

[1.4 Потреби ринку 56](#_Toc207121704)

[1.5 Бізнес-ризики 56](#_Toc207121705)

[2 БАЧЕННЯ РІШЕННЯ 57](#_Toc207121706)

[2.1 Заява про бачення 57](#_Toc207121707)

[2.2 Основні функції 57](#_Toc207121708)

[2.3 Припущення та залежності 58](#_Toc207121709)

[3 МЕЖІ ТА ОБМЕЖЕННЯ 59](#_Toc207121710)

[3.1 Межі першого релізу 59](#_Toc207121711)

[3.2 Межі наступних релізів 59](#_Toc207121712)

[3.3 Обмеження та виключення 60](#_Toc207121713)

[4 БІЗНЕС-КОНТЕКСТ 60](#_Toc207121714)

[4.1 Профілі зацікавлених сторін 60](#_Toc207121715)

[4.2 Пріоритети проєкту 60](#_Toc207121716)

[4.3 Операційне середовище 61](#_Toc207121717)

# **ВИМОГИ ДО БІЗНЕСУ**

## Передумови

Попри наявність багатьох музичних додатків, більшість орієнтовані або на професійних музикантів, або на великі навчальні екосистеми. Це ускладнює їх використання початківцями та звичайними користувачами, які прагнуть простого й зрозумілого інструмента для роботи зі звуком.

## Бізнес-можливість

iGuitar надає музикантам інтерактивний мобільний інструмент для створення й редагування гітарних пресетів, прослуховування у реальному часі та керування власними аудіо-треками.Це дозволяє уникнути потреби у складному обладнанні чи професійних студійних комплексах. Додаток орієнтований як на індивідуальних користувачів, так і на викладачів чи невеликі гурти.

## Бізнес-цілі та критерії успіху

Для успішної реалізації проєкту необхідно врахувати як технічні, так і користувацькі аспекти, щоб забезпечити ефективну роботу застосунку. Головна мета – створити інструмент, який буде простим у використанні, гнучким у налаштуваннях та здатним підвищити продуктивність завдяки інтеграції сучасних технологій.

Основні цілі:

* створити мобільний застосунок для управління гітарними пресетами;
* надати можливість редагування параметрів ефектів;
* забезпечити інтеграцію з аудіо-треками;
* впровадити авторизацію й збереження даних на сервері через Django REST API;

Критерії успіху:

* реалізоване ядро: створення пресетів, редагування параметрів, застосування до треків;
* працююча система входу/реєстрації користувачів;
* попереднє прослуховування в реальному часі без затримок;
* адаптивний та стабільний інтерфейс;
* відсутність критичних помилок під час тестування.

Досягнення цих цілей забезпечить створення ефективного та сучасного інструменту, що відповідатиме потребам користувачів і дозволить їм комфортно працювати зі звуком та керувати робочим простором.

## Потреби ринку / користувачів

Багато сучасних музичних застосунків занадто складні або дорогі для пересічного користувача. Саме тому існує потреба в простому та доступному рішенні.

Потреби ринку та користувачів:

* проста альтернатива складним і дорогим музичним платформам;
* англомовний інтерфейс;
* підтримка роботи з аудіо у реальному часі;

Врахування цих факторів дозволить створити продукт, який максимально відповідатиме потребам сучасних користувачів, надаючи їм інструменти для зручної та ефективної роботи.

## Бізнес-ризики

Розробка мобільного застосунку iGuitar супроводжується низкою ризиків, які необхідно враховувати ще на етапі проєктування. Одним із ключових викликів є залежність від браузерних технологій, адже Web Audio API може по-різному підтримуватися різними браузерами, що потенційно вплине на стабільність роботи. Також існує загроза перевантаження серверної частини при великій кількості одночасних користувачів, що може спричинити зниження продуктивності системи. Додатковим фактором є ризики, пов’язані з безпекою: недостатній контроль доступу чи неналежна реалізація механізмів автентифікації (JWT) можуть призвести до уразливостей та витоку даних.

# **БАЧЕННЯ РІШЕННЯ**

## Заява про бачення

iGuitar – це клієнт-серверний мобільний застосунок для створення та редагування гітарних пресетів із підтримкою роботи з аудіо-треками.

Користувач може створювати та налаштовувати ефекти, застосовувати їх у реальному часі до треків, зберігати результати на сервері та працювати через зручний адаптивний інтерфейс.

## Основні функції

Для забезпечення зручного та функціонального користувацького досвіду необхідно реалізувати набір ключових можливостей, які дозволять ефективно працювати з документами, налаштовувати середовище та інтегрувати додаток із зовнішніми сервісами. Важливо, щоб кожна функція була не лише корисною, а й інтуїтивно зрозумілою для кінцевого користувача.

Основні функції:

* Авторизація та управління профілем.
* Створення, редагування та видалення пресетів.
* Бібліотека ефектів: Delay, Reverb, Chorus, Distortion, Compressor, Noise Gate, Overdrive.
* Налаштування параметрів ефектів у зручному редакторі.
* Попереднє прослуховування у реальному часі.
* Завантаження та відтворення власних аудіо-треків.
* Інтеграція з Django REST API для синхронізації даних..

Щоб забезпечити широкий спектр можливостей для користувачів, додаток включає набір ключових функцій, спрямованих на зручність та ефективність роботи зі звуком .

## Припущення та залежності

Для стабільної роботи платформи необхідно врахувати технічні залежності та припущення, що визначають умови функціонування системи.

Припущення та залежності:

* Збереження та обробка пресетів здійснюється на сервері (Django + ORM).
* Web Audio API використовується для відтворення й обробки звуку у реальному часі.
* Додаток підтримує Android 7.0 і вище.
* Потрібне стабільне інтернет-з’єднання.

Завдяки продуманому підходу до основних функцій та врахуванню технічних залежностей, система забезпечує зручну роботу користувачів із створенням та керуванням аудіо-пресетами, попереднім прослуховуванням ефектів та збереженням результатів для подальшого використання.

# **МЕЖІ ТА ОБМЕЖЕННЯ**

## Межі першого релізу

Перший реліз iGuitar сфокусований на стабільній роботі основних функцій для створення та редагування аудіо-пресетів ефектів. Головний акцент зроблено на застосуванні ефектів у реальному часі та попередньому прослуховуванні аудіо.

Межі першого релізу:

* Реєстрація, авторизація, профіль користувача.
* Створення пресетів.
* Бібліотека базових ефектів (Delay, Reverb, Chorus, Distortion).
* Попереднє прослуховування в реальному часі.
* Завантаження аудіо-треків та застосування до них ефектів.
* Збереження даних на сервері.

Цей реліз закладає фундамент для майбутніх розширень, поступово доповнюючи систему новими функціональними можливостями.

## Межі наступних релізів

Наступні оновлення будуть спрямовані на розширення можливостей платформи, покращення взаємодії між користувачами та повноцінну оптимізацію для різних пристроїв.

Межі наступних релізів:

* Додаткові ефекти (Compressor, Noise Gate, Overdrive).
* Імпорт/експорт оброблених треків.
* Збереження історії змін пресетів.
* Коментування та спільна робота з пресетами.
* Інтернаціоналізація інтерфейсу (українська, англійська).
* iOS-версія.;

Втілення цих функцій забезпечить не лише більшу гнучкість у роботі з платформою, але й покращить користувацький досвід та зручність взаємодії.

## Обмеження та виключення

Попри широкий набір можливостей, платформа має певні обмеження, які необхідно враховувати в межах її першої версії. Деякі функції можуть бути реалізовані у майбутніх оновленнях або залишитися поза межами основного функціоналу.

Обмеження та виключення:

* Немає офлайн-режиму в першій версії.
* Відсутня розширена система аналітики.
* Робота лише з поширеними аудіо-форматами (mp3, wav, flac).Інтернаціоналізація інтерфейсу (українська, англійська) та покращена мобільна адаптивність;

Ці аспекти можуть бути розглянуті для впровадження у майбутніх версіях платформи.

# **БІЗНЕС-КОНТЕКСТ**

## Профілі зацікавлених сторін

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Зацікавлена сторона** | **Основна цінність** | **Ставлення** | **Основні інтереси** | **Обмеження** |
| Користувач | Простота, стабільність | Позитивне | Легкий інтерфейс, швидка робота | Не має технічних знань |
| Розробники | Архітектура, модульність | Залучені | Чистий код, підтримуваність | Обмеження часу/ресурсів |
| Замовник (викладач) | Навчальна цінність | Нейтральне | Відповідність вимогам | Обмежений час на перевірку |

## Пріоритети проєкту

|  |  |  |  |
| --- | --- | --- | --- |
| **Параметр** | **Ключовий драйвер** | **Обмеження** | **Ступінь свободи** |
| Функціональність | Так | Часткова реалізація | Високий |
| Продуктивність | Так | Мінімальні затримки | Середній |
| Масштабованість | Ні | |  | | --- | |  |  |  | | --- | | Тимчасово обмежена | | В майбутньому можливо |
| Безпека | Так | JWT, ролі | Низький |
| Терміни | Так | |  | | --- | |  |  |  | | --- | | Навчальний графік | | Визначено чітко |

## Операційне середовище

Для коректної роботи застосунку iGuitar необхідне стабільне операційне середовище, яке забезпечує узгоджену взаємодію клієнтської та серверної частин. На рівні клієнта система орієнтована на мобільні пристрої під управлінням Android версії 7.0 і вище, що дозволяє охопити більшість сучасних смартфонів. Серверна частина функціонує на базі Django з використанням REST API, завдяки чому забезпечується швидкий та безпечний обмін даними між клієнтом і сервером. Для зберігання інформації про користувачів і пресети застосовується PostgreSQL або інша сумісна SQL-система керування базами даних, яка гарантує надійність і масштабованість. У фронтенді інтегровано Web Audio API, що дає змогу реалізувати обробку та попереднє відтворення аудіоефектів безпосередньо у браузері. Водночас важливим технічним чинником є стабільне інтернет-з’єднання зі швидкістю не нижче 2 Мбіт/с, адже воно забезпечує безперервність роботи, швидке завантаження даних і коректну взаємодію користувачів із платформою.