НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)

Кафедра інформатики та програмної інженерії
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення інформаційних систем»

спеціальності «121 Інженерія програмного забезпечення»

на тему: Мобільний застосунок для пошуку музичного гурту

Виконав студент IV курсу, групи ІП-02
(шифр групи)

Середюк Валентин Васильович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н., доц., Іванова Л.М. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Рецензент зав.каф. КІ ДУІКТ, к.т.н., доц., Лащевська Н.О. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

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

Київ –2024

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 Інженерія програмного забезпечення

Освітньо-професійна програма – Інженерія програмного забезпечення інформаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Едуард ЖАРІКОВ
(підпис) (ім'я прізвище)
"___"_____2024 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Середюк Валентин Васильович
(прізвище, ім'я, по батькові)

1. Тема проєкту Мобільний застосунок для пошуку музичного гурту
керівник проєкту Іванова Любов Миколаївна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «27» травня 2024 р. №2112-с

2. Термін подання студентом проєкту «17» червня 2024 року

3. Вихідні дані до проєкту: технічне завдання

4. Зміст пояснювальної записки
1) Передпроєктне обстеження предметної області: аналіз предметної області, аналіз існуючих рішень, опис бізнес-процесів, постановка задачі.
2) Розроблення вимог до програмного забезпечення: варіанти використання програмного забезпечення, розроблення функціональних вимог, розроблення нефункціональних вимог.
3) Конструювання та розроблення програмного забезпечення: архітектура програмного забезпечення, обґрунтування вибору засобів розробки, конструювання програмного забезпечення, аналіз безпеки даних.
4) Аналіз якості та тестування програмного забезпечення: аналіз якості програмного забезпечення, опис процесів тестування, опис контрольного прикладу
5) Розгортання та супровід програмного забезпечення: розгортання програмного забезпечення, супровід програмного забезпечення.

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «11» березня 2024 року _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	18.03.2024	
3	Постановка та формалізація задачі	25.03.2024	
4	Розробка інформаційного забезпечення	01.04.2024	
5	Алгоритмізація задачі	08.04.2024	
6	Обґрунтування вибору використаних технічних засобів	15.04.2024	
7	Розробка програмного забезпечення	20.04.2024	
8	Налагодження програми	05.05.2024	
9	Виконання графічних документів	12.05.2024	
10	Оформлення пояснювальної записки	16.05.2024	
11	Подання ДП на попередній захист	03.06.2024	
12	Подання ДП рецензенту	10.06.2024	
13	Подання ДП на основний захист	14.06.2024	

Студент _____ Валентин СЕРЕДЮК
(підпис) (ініціали, прізвище)

Керівник _____ Любов ІВАНОВА
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Пояснювальна записка дипломного проєкту складається з п’ятьох розділів, містить 54 таблиць, 35 рисунків та 21 джерел – загалом 84 сторінки.

Мета: полегшити процес інтеграції творчих людей у музичній сфері за рахунок розробки застосунку пошуку музичного гурту.

Об'єкт дослідження: програмне забезпечення для пошуку музичного гурту.

Предмет дослідження: процеси розроблення, тестування та впровадження програмного забезпечення та аналізу взаємозв’язків між музикантами.

У першому розділі було проведено аналіз предметної області, порівняння із існуючими аналогами, описано бізнес-процеси. Як результат було сформовано постановку задачі.

У другому розділі було створено діаграму використань програмного забезпечення, на основі якої були розроблені функціональні та нефункціональні вимоги.

У третьому розділі було досліджено та розроблено архітектуру програмного забезпечення, на основі якої було створено діаграму пакетів програмного забезпечення. Також розроблене саме програмне забезпечення та база даних.

У четвертому розділі було досліджено програмне забезпечення на якість коду, а також проведено мануальне тестування з описом контрольного прикладу.

У п’ятому розділі було розгорнуто програмне забезпечення на платформі Amazon Appstore, а також наведена інструкція для супроводу мобільного додатку.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ДОДАТОК, АНДРОЇД, МУЗИКА, ГУРТ, ПОШУК РОБОТИ, ВАКАНСІЯ, FIREBASE.

ABSTRACT

The explanatory note of the diploma project consists of five sections, contains 54 tables, 35 figures and 21 sources – in total 84 pages.

The purpose of the diploma project is to facilitate the process of integrating creative people in the music industry by developing a band search application.

Object of research: software for finding a music band.

Subject of research: the processes of developing, testing and implementing software and analyzing the relationships between musicians.

The first section analyzed the subject area, compared it with existing analogs, and described business processes. As a result, the task statement was formed.

In the second section, a software use case diagram was created, based on which functional and non-functional requirements were developed.

In the third section, the software architecture was researched and developed, on the basis of which a software package diagram was created. The software and the database were also developed.

In the fourth section, the software was examined for code quality, and manual testing was performed with a description of the control example.

In the fifth section, the software was deployed on the Amazon Appstore platform and instructions for maintaining the mobile application were provided.

KEYWORDS: MOBILE APPLICATION, ANDROID, MUSIC, BAND, JOB SEARCH, VACANCY, FIREBASE.

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-0222.045440.01.91	Технічне завдання	20	
3	А4	КПІ.ІП-0222.045440.02.81	Пояснювальна записка	84	
4	А4	КПІ.ІП-0222.045440.03.12	Текст програми	46	
5	А4	КПІ.ІП-0222.045440.04.51	Програма та методика тестування	6	
6	А4	КПІ.ІП-0222.045440.05.34	Керівництво користувача	18	
7	А3	КПІ.ІП-0222.045440.06.99	Графічний матеріал	3	

КПІ.ІП-0222.045440.00.90

Змін.	Арк.	№ докум.	Підп.	Дата
Розроб.		Середюк В.В.		
Перевір.		Іванова Л.М.		
Н.контр.		Шулькевич Т.В.		
Затв.		Жаріков Е.В.		

Відомість дипломного проєкту

Літ.	Аркуш	Аркушів
	1	

КПІ ім. Ігоря Сікорського
ФІОТ каф. ІПІ гр. ІП-02

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Завідувач кафедри

_____ Едуард ЖАРІКОВ

“___” _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПОШУКУ МУЗИЧНОГО ГУРТУ

Технічне завдання

КПІ.ІП-0222.045440.01.91

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Любов ІВАНОВА

Нормоконтроль:

_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:

_____ Валентин СЕРЕДЮК

Київ – 2024

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: мобільний застосунок для пошуку музичного гурту.

Галузь застосування: музична індустрія, повсякденне життя.

Наведене технічне завдання поширюється на розробку мобільного застосунку для пошуку музичного гурту КПІ.ІП-0222.045440, котра використовується для пошуку музичного гурту для роботи в ньому або кандидатів у свій гурт та призначена для полегшення процесу об’єднання різних музикантів у гурти.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки мобільного застосунку для пошуку музичного гурту є завдання на дипломне проєктування, затверджене кафедрою інформатики та програмної інженерії Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для знаходження музичного гурту для професійного зростання або учасників у свій гурт.

Метою розробки є полегшити процес інтеграції творчих людей у музичній сфері за рахунок розробки застосунку пошуку музичного гурту.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1 Користувацького інтерфейсу

– сторінка авторизації (рис. 4.1);

Рисунок 4.1 – Прототип сторінки авторизації

– сторінка реєстрації гурту (рис. 4.2);

Рисунок 4.2 – Прототип сторінки реєстрації гурту

– сторінка реєстрації кандидата (рис. 4.3);

Рисунок 4.3 – Прототип сторінки реєстрації кандидата

– сторінка головного пункту меню (рис. 4.4);

Рисунок 4.4 – Прототип сторінки головного пункту меню

– сторінка списку чатів користувача (рис. 4.5);

Рисунок 4.5 – Прототип сторінки чатів користувача

– сторінка чату із іншим користувачем (рис. 4.6);

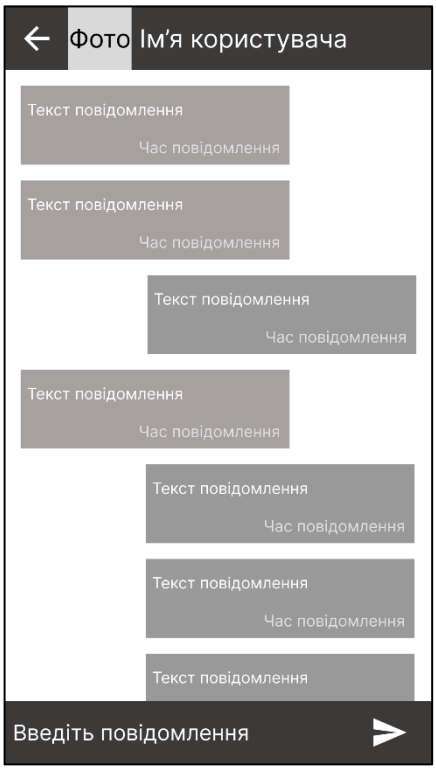


Рисунок 4.6 – Прототип сторінки чату з користувачем

– сторінка пошуку кандидата (рис. 4.7);

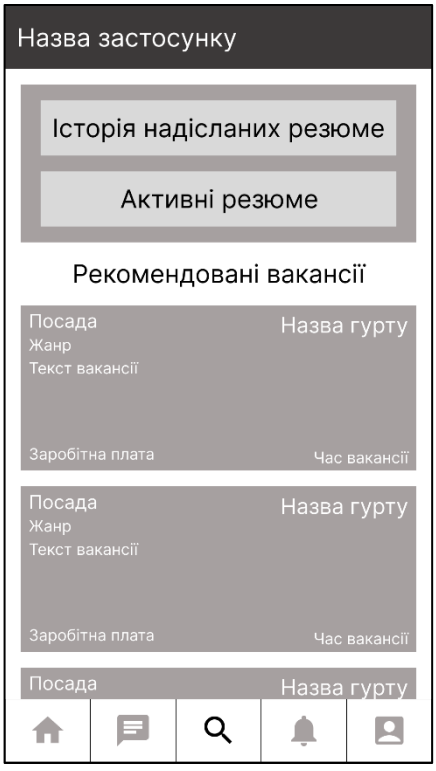


Рисунок 4.7 – Прототип сторінки пошуку кандидата

– сторінка перегляду історії резюме та активних резюме (рис. 4.8);

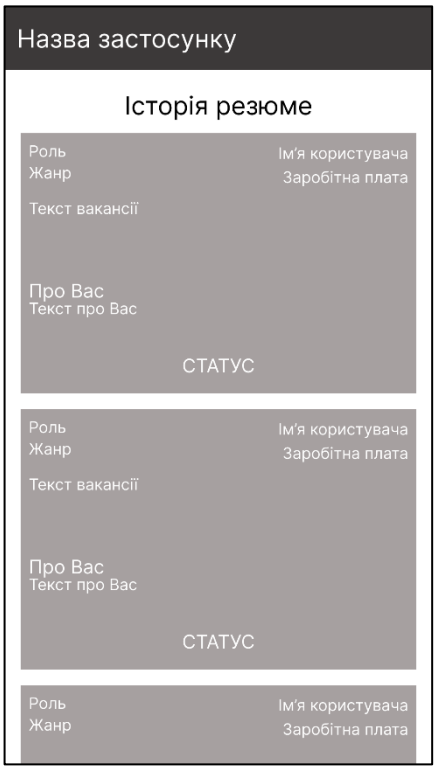


Рисунок 4.8 – Прототип сторінки історії резюме та активних резюме

– сторінка інформації про резюме (рис. 4.9);

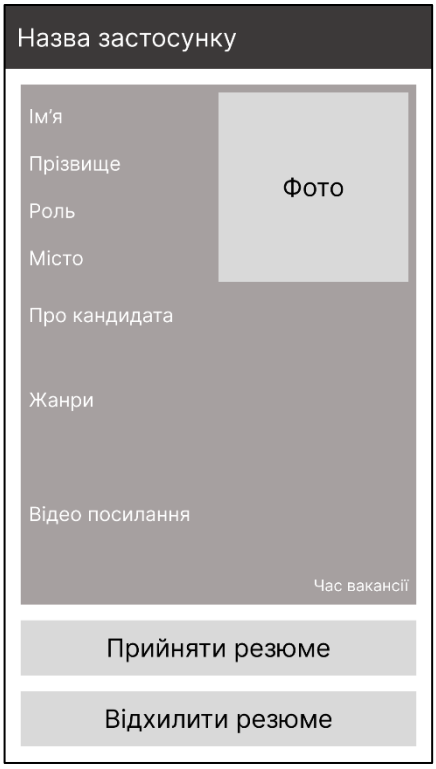


Рисунок 4.9 – Прототип сторінки інформації про резюме

– сторінка надсилання резюме (рис. 4.10);

Рисунок 4.10 – Прототип сторінки надсилання резюме

– сторінка пошуку власника гурту (рис. 4.11);

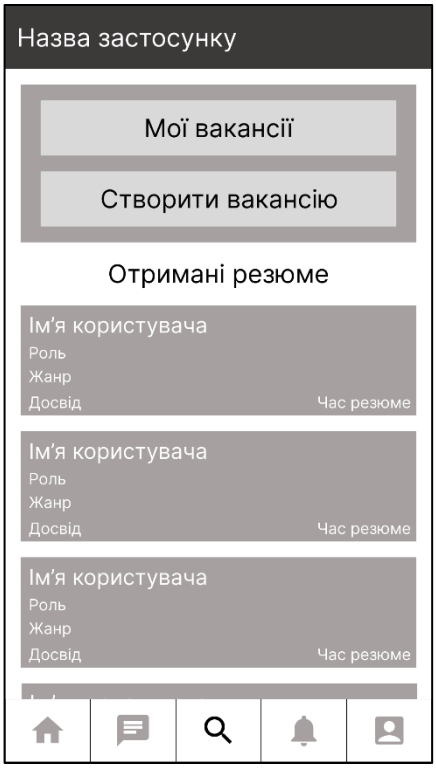


Рисунок 4.11 – Прототип сторінки пошуку власника гурту

– сторінка перегляду виставлених вакансій (рис. 4.12);

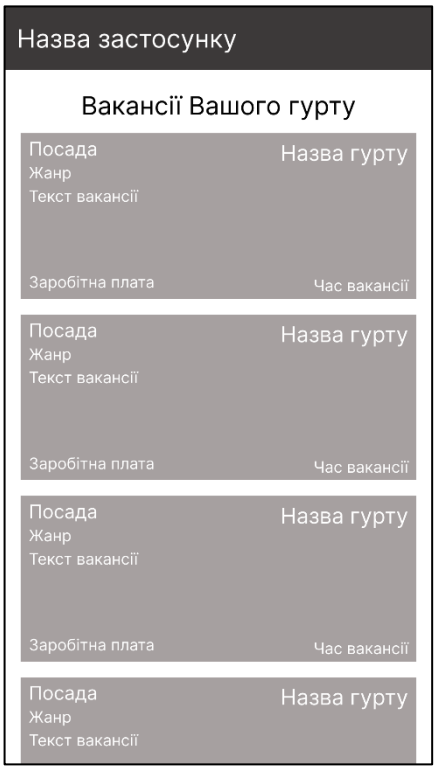


Рисунок 4.12 – Прототип сторінки виставлених вакансій гурту

– сторінка інформації про вакансію (рис. 4.13);

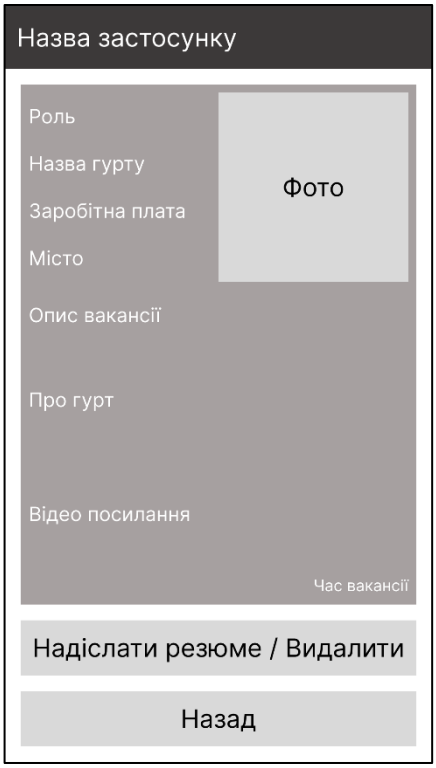


Рисунок 4.13 – Прототип сторінки інформації про вакансію

– сторінка створення вакансії (рис. 4.14);

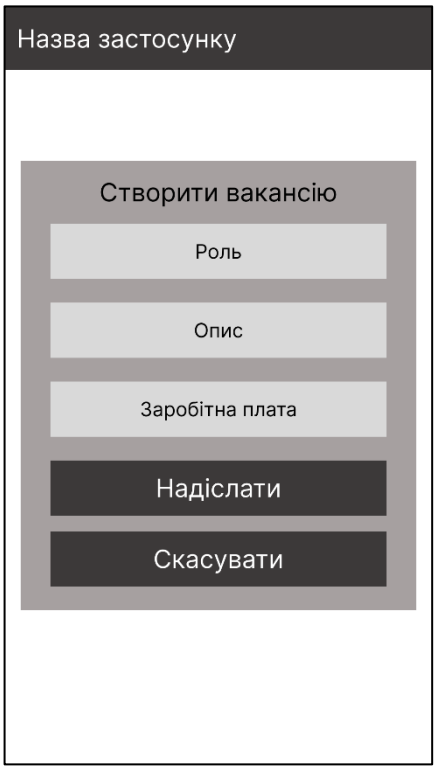


Рисунок 4.14 – Прототип сторінки створення вакансії

– сторінка списку сповіщень (рис. 4.15);

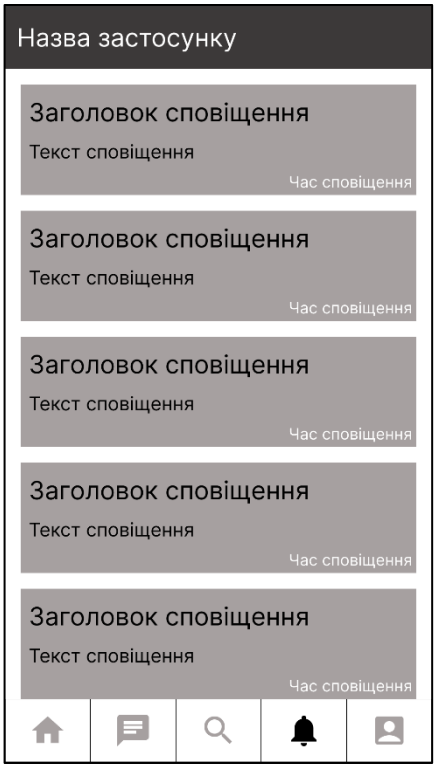


Рисунок 4.15 – Прототип сторінки отриманих сповіщень

– сторінка профілю користувача (рис. 4.16);

Рисунок 4.16 – Прототип сторінки профілю користувача

– сторінка редагування профілю користувача (рис. 4.17).

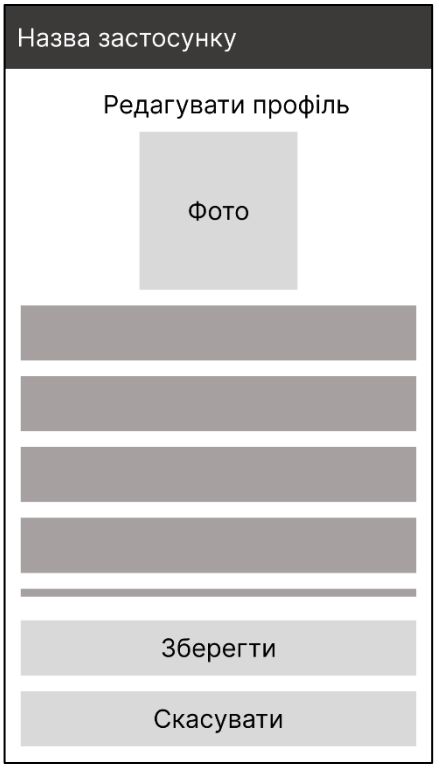


Рисунок 4.17 – Прототип сторінки редагування профілю користувача

4.1.2 Для користувача:

– авторизація;

– реєстрація кандидата;
– реєстрація власника гурту;
– перегляд та зміна профілю користувача;
– створення та перегляд чатів;
– надсилання текстових повідомлень у чати;
– отримання сповіщення про зміни у статусі резюме/вакансій;
– створення, перегляд та видалення вакансій;
– отримання рекомендованих вакансій;
– відповідь на вакансію;
– створення та перегляд резюме;
– отримання надісланих резюме;
– відповідь на резюме.

4.1.3 Додаткові вимоги:

– забезпечення автоматичної авторизації при повторному відкритті застосунку;
– робота чату у режимі реального часу.

4.2 Вимоги до надійності

Передбачити контроль введення інформації та захист від некоректних дій користувача. Забезпечити цілісність інформації в базі даних. Запобігти несанкціонованому доступу до даних користувача.

4.3 Умови експлуатації

Умови експлуатації згідно СанПін 2.2.2.542 – 96.

4.3.1 Вид обслуговування

Вимоги до виду обслуговування не висуваються

4.3.2 Обслуговуючий персонал

Вимоги до обслуговуючого персоналу не висуваються

4.4 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинно функціонувати на мобільних пристроях з операційною системою Android.

Мінімальна конфігурація технічних засобів:

– частота процесору: 1 ГГц;
– об‘єм ОЗП: 3 Гб;
– об’єм вільного місця на накопичувачі: 28 МБ;
– діагональ екрану 4 дюйма;
– підключення до мережі Інтернет зі швидкістю від 8 мегабіт;

Рекомендована конфігурація технічних засобів:

– частота процесору: 3 ГГц;
– об‘єм ОЗП: 8 Гб;
– об’єм вільного місця на накопичувачі: 50 МБ;
– діагональ екрану 6 дюймів;
– підключення до мережі Інтернет зі швидкістю від 100 мегабіт;

4.5 Вимоги до інформаційної та програмної сумісності

Програмне забезпечення повинно працювати під управлінням операційних систем Android версії 9 і вище (підтримка API level 33).

4.5.1 Вимоги до вхідних даних

Вхідні дані повинні бути представлені в наступному форматі: користувач вводить необхідну інформацію у спеціальні поля на різних сторінках. Також, присутні списки, з яких користувач може обрати готові вхідні дані.

4.5.2 Вимоги до вихідних даних

Результати повинні бути представлені в наступному форматі: уся необхідна інформація повинна бути відображена на відповідних сторінках застосунку і вигляді тексту або фото матеріалу.

4.5.3 Вимоги до мови розробки

Розробку виконати на мові програмування Java.

4.5.4 Вимоги до середовища розробки

Розробку виконати за допомогою середовища Android Studio.

4.5.5 Вимоги до представленню вихідних кодів

Вихідний код програми має бути представлений у документі «Код програми», а також у репозиторії GitHub.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Згенерувати інсталяційну версію програмного забезпечення. Зробити інсталяційну версію доступною на будь-які маркет-платформі мобільних застосунків.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

У склад супроводжувальної документації повинні входити наступні документи на аркушах формату А4:

– пояснювальна записка;
– технічне завдання;
– текст програми;
– програма та методика тестування;
– керівництво користувача;

Графічна частина повинна бути виконана на аркушах формату А3 та містити наступні документи:

– схема структурна варіантів використання;
– схема бази даних;
– креслення вигляду екранних форм.

5.2 Спеціальні вимоги до програмної документації

Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі у вигляді опису функціональних вимог, які покриває зазначений код.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк	Звітність
1	Вивчення рекомендованої літератури	11.03.2024	
2	Аналіз існуючих методів розв’язання задачі	18.03.2024	
3	Постановка та формалізація задачі	25.03.2024	Технічне завдання
4	Розробка інформаційного забезпечення	01.04.2024	Схема структурна програмного забезпечення та специфікація компонентів (пакетна діаграма, схема бази даних)
5	Алгоритмізація задачі	08.04.2024	
6	Обґрунтування вибору використаних технічних засобів	15.04.2024	Технічна документація
7	Розробка програмного забезпечення	20.04.2024	Тексти програмного забезпечення
8	Налагодження програми	05.05.2024	Програми та методика тестувань
9	Виконання графічних документів	12.05.2024	Графічний матеріал проекту
10	Оформлення пояснювальної записки	16.05.2024	Пояснювальна записка

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

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

Пояснювальна записка
до дипломного проєкту

на тему: **Мобільний застосунок для пошуку музичного гурту**

КПІ.ІП-0222.045440.02.81

Київ – 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК	–	Персональний комп’ютер
ПЗ	–	Програмне забезпечення
OS	–	Operating System, операційна система
MVC	–	Model-View-Controller, модель-представлення-контролер
MVP	–	Model-View-Presenter, модель-представлення-представник
MVVM	–	Model-View-ViewModel, модель-представлення-модель представлення
BPMN	–	Business Process Modeling and Notation, моделювання та нотація бізнес-процесів
DAO	–	Data Access Object, об’єкт доступу до даних
JSON	–	JavaScript Object Notation, нотація об’єкта мовою javascript
ADT	–	Android Development Tools, знаряддя для розробки Android-застосунків
SDK	–	Software Development Kit, набір для розробки програмного забезпечення
JDK	–	Java Development Kit, комплект для розробки мовою програмування Java
IDE	–	Integrated Development Enviroment, інтегроване середовище розробки
JVM	–	Java Virtual Machine, віртуальна машина Java
БД	–	База даних

ВСТУП

Мобільні пристрої, на сьогоднішній день, стали невід’ємною частиною життя людини. Складно навіть уявити хоча б один день, без тих зручностей, що надає нам смартфон: від будильнику до оплати товарів та послуг, від комунікацій до робочого простору. Музична індустрія також не стояла на місці. Взаємодія музикантів, менеджерів та агентів все більше переходить у цифровий формат, де мобільні застосунки стають невід'ємною частиною їхнього повсякденного життя. Таким чином, розробка мобільного застосунку для пошуку музичного гурту є надзвичайно актуальною задачею, що відповідає потребам сучасних користувачів.

У глобальному, спостерігається тенденція до створення спеціалізованих платформ для представників певних професій, особливо творчих. Наприклад музичні платформи могли б сприяти взаємодії між музикантами, допомагати знайти учасників гурту або замінити тимчасово відсутніх музикантів. Основний акцент у таких застосунках робиться на швидкості та зручності пошуку, а також на можливості представити свої музичні навички та репертуар.

Розроблений мобільний застосунок для пошуку роботи в музичному гурті може бути використаний як окремими музикантами, так і музичними гуртами, менеджерами та продюсерами. Він сприятиме швидкому та ефективному знаходженню роботи або нових учасників гурту, зважаючи на географічне розташування, музичні вподобання та рівень майстерності. Застосунок також може бути інтегрований із соціальними мережами та професійними платформами, що дозволить розширити його функціонал та підвищити ефективність пошуку.

1 ПЕРЕДПРОЄКТНЕ ОБСТЕЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз предметної області

Мобільний застосунок (або мобільний додаток) — це програмне забезпечення, розроблене для використання на мобільних пристроях, таких як смартфони та планшети. Ці застосунки можуть виконувати різноманітні функції, включаючи: комунікації, розваги, продуктивність, освіта, комерція [1].

Застосунок для пошуку музичного гурту – це сервіс, який дозволяє користувачам знаходити музичні гурти, у яких вони могли б себе проявити, а також знаходити нові знайомства серед колег у музичній сфері. Даний тип застосунку можна віднести до застосунків із пошуку роботи.

Музичний гурт — це група музикантів, які об'єдналися для спільного виконання музики. Гурти можуть виконувати різні жанри музики, такі як рок, поп, джаз, класичну музику тощо. Кожен гурт має свій унікальний стиль і звучання, яке формується за рахунок творчої співпраці його учасників.

Сервіс з пошуку роботи – це сервіс, що дозволяє швидко і ефективно знайти найбільш підходящого співробітника. Роботодавці розміщують вакансію і вимоги до кандидата, здобувач може залишити своє резюме і тестове завдання, яке він виконав. Всі резюме зберігаються в особистому кабінеті HR-менеджера [2].

На сьогоднішній день, рішення, які існують на ринку не пропонують гнучкого налаштування під музикантів. Гурти не виставляються вакансії у звичному всім розумінні, оскільки сучасні рішення є незручними для них і їм доводиться користуватись своїми соціальними мережами, що не є найкращим методом.

Викладання, робота в театрі чи оркестрі, наприклад, може забезпечити стабільну регулярну роботу, а іноді навіть гарантувати статус оплачуваної зайнятості. Це винятки з правил: лише 10% музикантів працюють на умовах повної зайнятості. Половина музикантів взагалі не мають постійної роботи.

Переважна більшість музикантів (94%) повністю або частково заробляють на життя фрілансом [3].

Також варто зазначити, що навіть серед працевлаштованих музикантів, представники жіночої статі мають не такі гарні умови, як представники чоловічої. Середня заробітна плата жінок у США на момент 2017 року складає 23,713$ на рік, у той момент, коли чоловіки мають 42,030$. На рисунку 1.1 наведено дані, щодо кількості працевлаштованих музикантів чоловіків та жінок у США за 2017 рік.

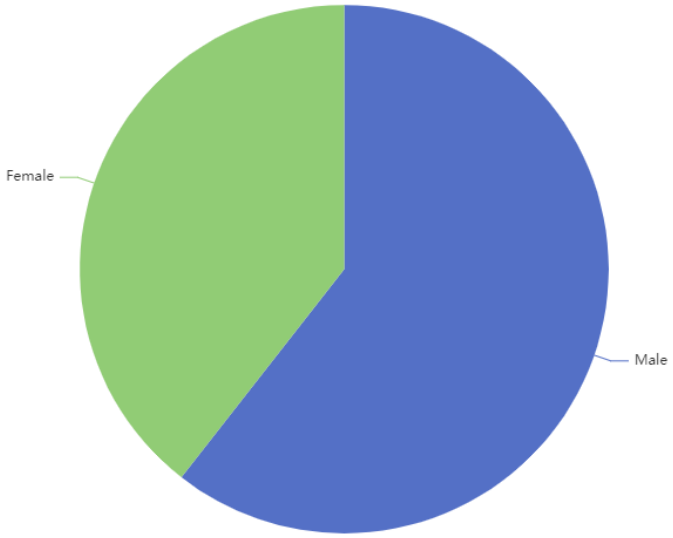


Рисунок 1.1 – Відношення кількості чоловік та жінок до загальної кількості музикантів [4]

Способом розв'язання цієї проблеми є розробка простору, де будь-які виконавці, музиканти або ж інші люди пов’язані із музичною сферою змогли б взаємодіяти незалежно від віку, статі чи досвіду. Застосунок такого плану допоможе знайти колег як для новачків, так і для досвідчених особистостей.

Отже, було вирішено розробити мобільний застосунок для пошуку музичного гурту, який дозволить не лише знайти вакансія для роботи, а й творчих однодумців із схожими інтересами.

1.2 Аналіз існуючих рішень

Розглянемо популярні сучасні технічні рішення, що допоможуть у реалізації мобільного застосунку для пошуку музичного гурту. Надалі розглянемо уже готові технічні та програмні рішення й інструменти розробки.

1.2.1 Аналіз відомих програмних продуктів

Зазначимо, що подібні ПЗ уже існують на ринку. Серед конкурентів можна виділити сервіси для пошуку роботи, наприклад – work.ua, robota.ua та сайт splice.com, який об’єднує в собі велику спільноту музикантів. Також є сервіс musicians.com.ua, який дозволяв знайти заробіток для музикантів, але останні публікації на ньому датуються 2 роками тому, тому його не будемо враховувати.

Work.ua

Work.ua — сайт з пошуку роботи в Україні. Він дозволяє компаніям розміщувати свої вакансії для знаходження співробітників у різних сферах діяльності. Його не можна виділити саме для якоїсь конкретної професії, але щодо музичних вакансій, то вони практично відсутні. Можна виділити лише вакансії для педагогів-музикантів, але ніяк не для гравців.

robota.ua

robota.ua – аналогічний до work.ua, сайт з пошуку роботи в Україні. Також дозволяє розміщувати вакансії та шукати співробітників. Являється універсальним, тобто можуть бути розміщені вакансії будь-яких компаній. На ньому також актуальні лише вакансії з пошуку педагогів-музикантів.

splice.com

splice.com – сайт, який спеціалізується на створенні музики та плагінах. Його особливістю є велика спільнота, у якій кожен може знайти однодумців або ж створити власну публічну сесію до якої зможе долучитись кожен.

Якщо не враховувати деякі сторінки в соціальних мережах, на разі відсутні сервіси, які надають змогу шукати собі професію та однодумців саме

у музичній сфері. Для наглядного порівняння мобільного застосунку для пошуку музичного гурту із своїми аналогами скористаємось таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогами

Особливості та функціонал застосунків	Мобільний застосунок для пошуку музичного гурту	work.ua	robota.ua	splice.com
Знаходження місця для роботи музикантом	Присутній	Присутній	Присутній	Відсутній (хіба при домовленості в спільноті)
Спілкування із однодумцями	Присутній	Відсутній	Відсутній	Присутній
Гнучке редагування профілю під музиканта	Присутній	Відсутній (хіба що вказувати в резюме усе детально)	Відсутній (хіба що вказувати в резюме усе детально)	Присутній (у спільноті можна про себе розповісти)
Можливість продемонструвати свої роботи на платформі	Присутній	Відсутній	Відсутній	Присутній
Наявність мобільного застосунку	Присутній	Присутній	Присутній	Присутній
Наявність крос-платформеності	Відсутній (в майбутньому планується розширення)	Присутній	Присутній	Присутній

1.2.2 Аналіз відомих алгоритмічних та технічних рішень

Під час проектування мобільних застосунків, перед розробником постає вибір між операційної системою, під яку буде розроблятись додаток. Серед найпопулярніших мобільних операційних систем можна виділити Android OS та IOS. Саме вони покривають 99,26% усіх пристроїв на планеті (рис. 1.2).

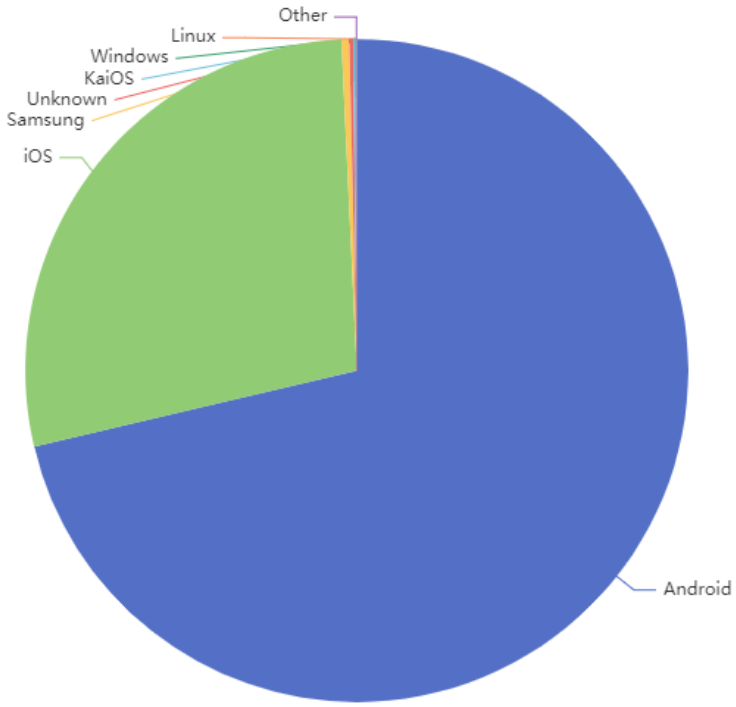


Рисунок 1.2 – Графік розподілу мобільних операційних систем по світовому ринку [5]

Як можна побачити із рисунку 1.2, операційна система Android є найуживанішою для мобільних пристроїв, а саме 71,31% від усіх користувачів. Android — це операційна система, розроблена компанією Google, яка використовується переважно на мобільних пристроях, таких як смартфони та планшети. Android також використовується на інших пристроях, включаючи телевізори, автомобільні системи, смарт-годинники, і навіть деякі ноутбуки [6]. Перевагами даної операційної системи є те, що вона пропонує широкі можливості налаштування як для користувачів, так і для розробників. Інтеграція з сервісами Google покращує користувацький досвід, а можливості багатозадачності підвищують продуктивність. Зручна для розробників екосистема, що включає Android Studio та Google Play Store, підтримує

розробку та розповсюдження додатків. Регулярні оновлення забезпечують доступ до найновіших функцій і методів для підтримки безпеки програмного забезпечення, а активна спільнота надає підтримку та сприяє інноваційному розвитку операційної системи.

Для мобільного застосунку було обрано підтримувати усі версії Android від 9 і вище, оскільки попередні версії уже не користуються популярністю серед користувачів. Детальніше, про популярність різних версій ОС Android можна дізнатись з рисунку 1.3. Версії із відсотком меншим за 1% не відображені.

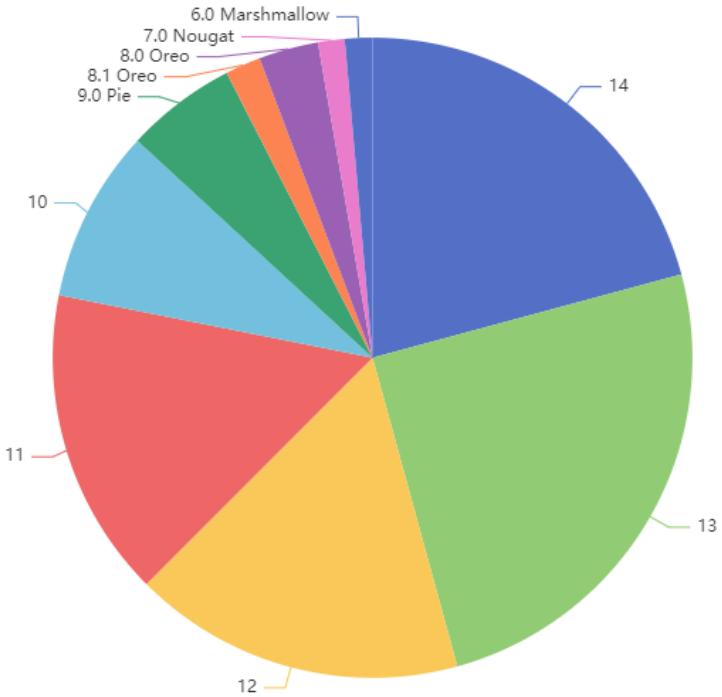


Рисунок 1.3 – Графік розподілу версій Android [5]

Найпопулярнішими архітектурними патернами під Android-застосунки, які використовують розробники: MVC, MVP та MVVM. Основна ідея всіх цих патернів полягає в тому, щоб організувати проект таким чином, щоб весь код був поділений на модулі. Крім того, це полегшує процес підтримки та оновлення програмного забезпечення, додаванні та видаленні функцій, розробники можуть тестувати різні логічні частини [7].

Model-View-Controller

Патерн MVC - це найстаріша архітектура додатків для Android, яка просто передбачає поділ коду на 3 різні шари (рис. 1.4):

Model: Рівень для зберігання даних. Він відповідає за обробку логіки бізнес-процесів і зв'язок з базою даних та різними рівнями передачі даних.

View: Рівень інтерфейсу користувача (UI), який відповідає за відображення та візуалізацію даних, отриманих з моделі зберігаються в моделі.

Controller: Рівень, який містить основну логіку, реагує на дії користувача та оновлює модель відповідно до вимог [7].

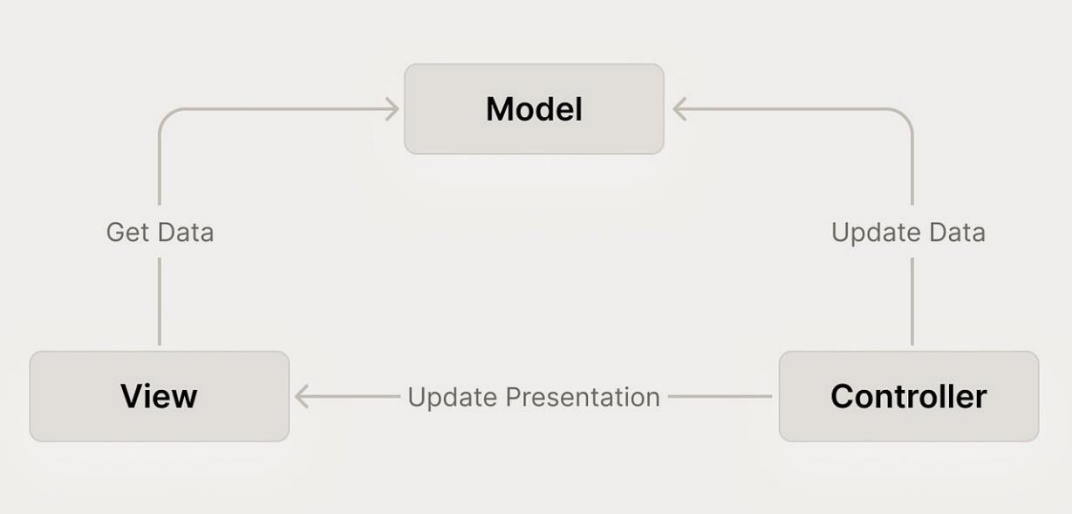


Рисунок 1.4 – Схема архітектурного патерну MVC [8]

Model-View-Presenter

Патерн MVP – це архітектурний патерн Android-додатків, який був призваний замінити MVC. Він є загальноприйнятим і також рекомендується для розробників. Призначення кожного компонента (рис. 1.5):

Model: Аналогічно до патерну MVC, це рівень для зберігання даних. Він відповідає за обробку бізнес-процесів і зв'язок з базою даних та мережевими рівнями.

View: Рівень інтерфейсу користувача (UI), який відповідає за відображення та візуалізацію даних та відстеження дій користувача, щоб інформувати про них презентера.

Presenter: Отримує дані з моделі і застосовує логіку інтерфейсу користувача для визначення того, що слід відобразити. Керує станом View і виконує дії відповідно до повідомлень про введення даних від користувача [7].

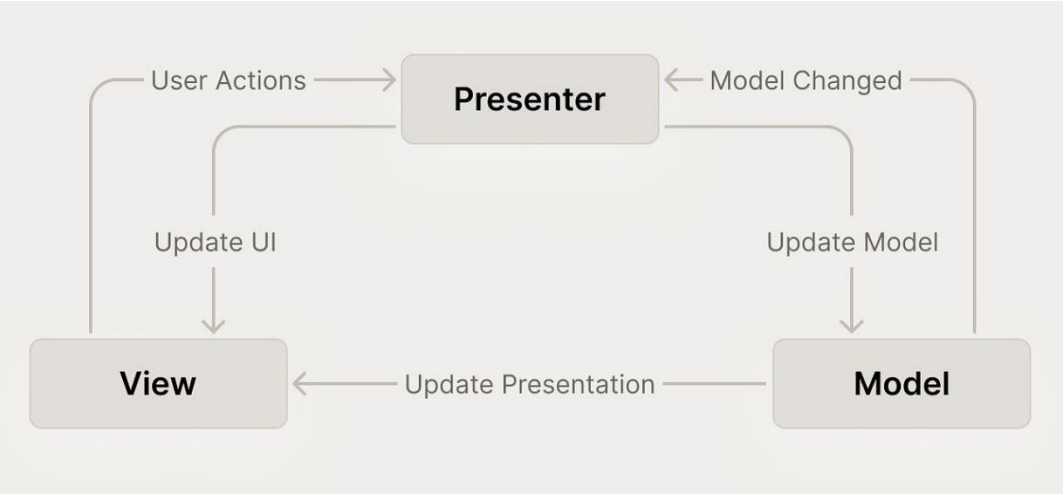


Рисунок 1.5 – Схема архітектурного патерну MVP [8]

Model-View-ViewModel

Третьою ітерацією архітектури Android є патерн MVVV. Під час випуску Android Architecture Components команда Android рекомендувала цей патерн архітектури. Нижче наведено окремі шари коду (рис. 1.6):

Model: Цей рівень забезпечує абстракцію джерел даних, а також співпрацює з ViewModel для отримання та збереження інформації.

View: Основна мета цього рівня полягає в тому, щоб передавати ViewModel інформацію про дії, виконані користувачем.

ViewModel: Показує ті потоки даних, які є релевантними для View [7].

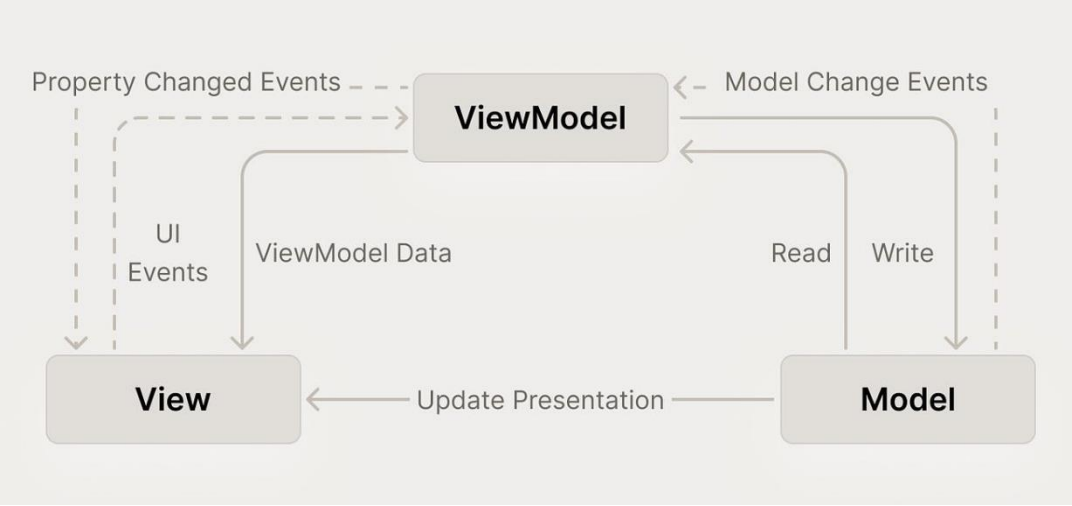


Рисунок 1.6 – Схема архітектурного патерну MVVM [8]

Патерни MVVM і MVP досить схожі, оскільки обидва ефективно абстрагуються від стану і поведінки шару View. У MVVM, View може зв'язати себе з потоками даних, які розкриваються у ViewModel.

Для детального порівняння усіх трьох архітектурних патернів, скористаємось таблицею 1.2.

Таблиця 1.2 – Порівняння архітектурних патернів

MVC	MVP	MVVM
Controller і View існують у відношенні "один до багатьох". Один Controller може вибирати інший View залежно від необхідної операції.	Зв'язок "один-до-одного" існує між Presenter та View, оскільки один клас Presenter керує одним View за один раз.	Кілька View можуть бути відображені за допомогою однієї ViewModel і, таким чином, між View і ViewModel існує зв'язок один-до-багатьох.
Важко вносити зміни та модифікувати функції програми, оскільки шари коду щільно пов'язані між собою.	Шари коду вільно з'єднані, тому легко вносити модифікації та зміни в код програми.	Легко вносити зміни в додаток. Однак, якщо логіка зв'язування даних занадто складна, налагоджувати додаток буде трохи складніше.
Обмежена підтримка модульного тестування.	Легко проводити модульне тестування, але тісний зв'язок між View та Presenter може дещо ускладнити його.	Можливість тестування модулів є найвищою в цій архітектурі.
Не дотримується модульного принципу та принципу єдиної відповідальності.	Дотримується модульного принципу та принципу єдиної відповідальності.	Дотримується модульного принципу та принципу єдиної відповідальності.

MVC є найпростішим з цих патернів, натомість MVP і MVVM – більш зручнішими, гнучкими і дозволяють більш чітко розділити завдання між різними рівнями додатку. Для розробки мобільного застосунку для пошуку музичного гурту було обрано саме MVVM, оскільки він забезпечує кращу підтримку та модифікацію коду, аніж інші патерни.

1.3 Опис бізнес-процесів

Опишемо основні бізнес-процеси, які будуть реалізовані у мобільному застосунку для пошуку музичного гурту.

Розглянемо BPMN модель для опису бізнес-процесу реєстрації нового користувача (рис. 1.7):

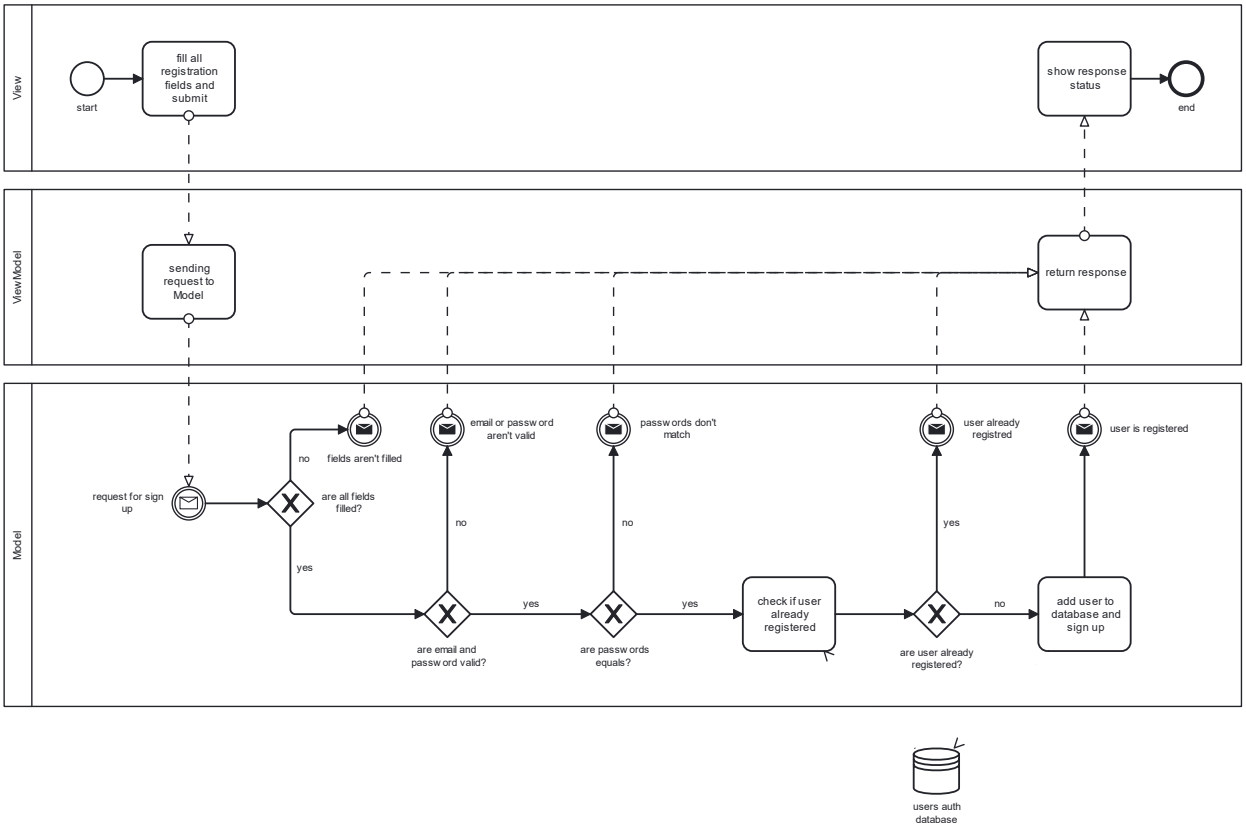


Рисунок 1.7 – BPMN модель бізнес-процесу реєстрації нового користувача

Опис послідовності реєстрації нового користувача:

– користувач заповнює усі поля реєстрації;
– користувач відправляє запит на реєстрацію;
– якщо введені не всі поля, повернеться помилка;

– якщо електронна пошта або пароль введені не вірного формату, повернеться помилка;

– якщо пароль та пароль підтвердження не співпадають, повернеться помилка;

– якщо користувач із такою ж електронною поштою вже був зареєстрований, повернеться помилка;

– інакше, користувач додається до бази даних зареєстрованих користувачів та повертається статус «зареєстровано»;

– користувачу відображається статус його реєстрації.

Розглянемо BPMN модель для опису бізнес-процесу входу в обліковий запис користувача (рис. 1.8):

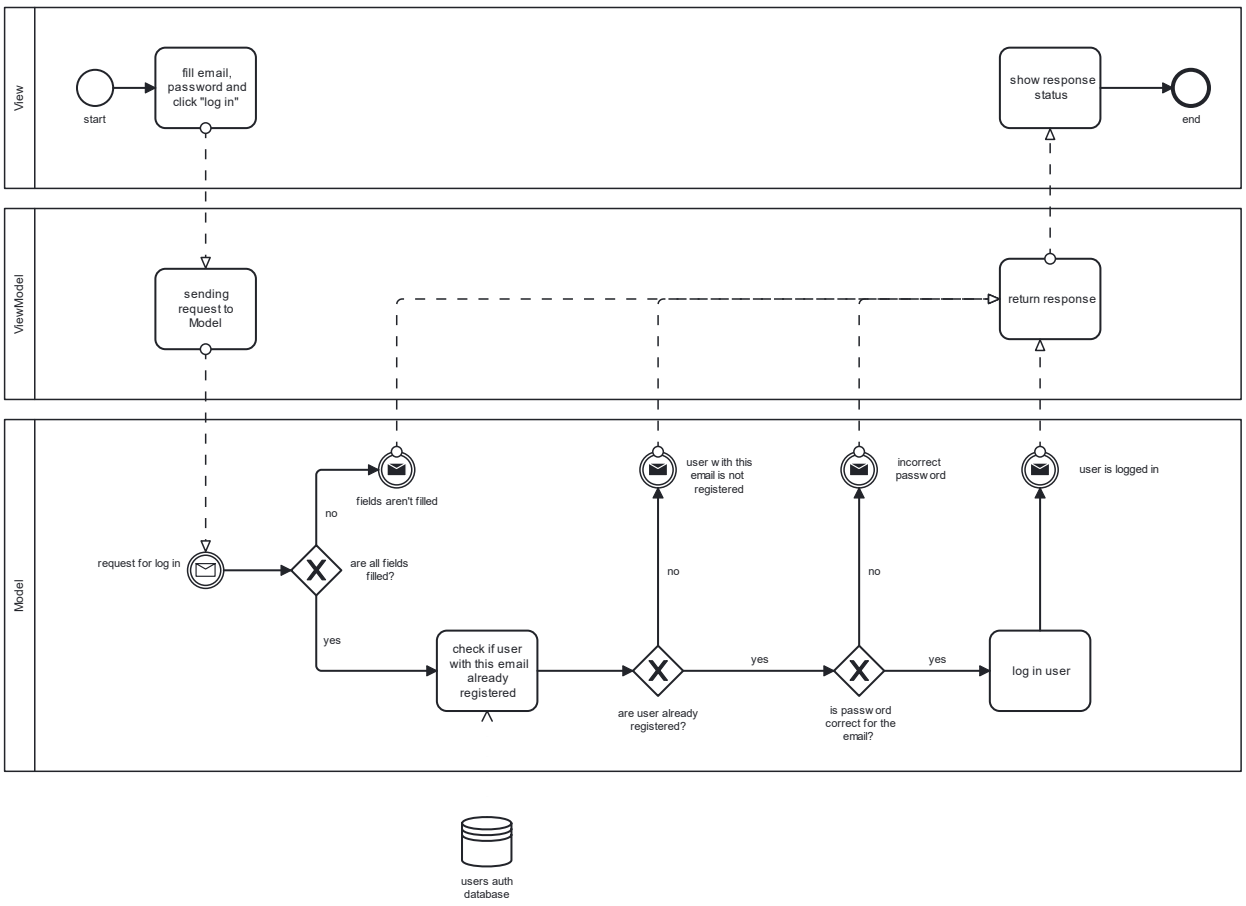


Рисунок 1.8 – BPMN модель бізнес-процесу входу користувача

Опис послідовності входу користувача до облікового запису:

– користувач заповнює електронну пошту, пароль та натискає «увійти»;

– користувач відправляє запит на вхід;

– якщо введені не всі поля, повернеться помилка;

– якщо електронна пошта не зареєстрована ще, повернуться помилка;

– якщо пароль не підходить до пошти, повернеться помилка;

– інакше, користувач успішно виконує вхід та повертає статус «увійдено»;

– користувачу відображається статус його реєстрації.

Розглянемо BPMN модель для опису бізнес-процесу відгуку на вакансію (рис. 1.9):

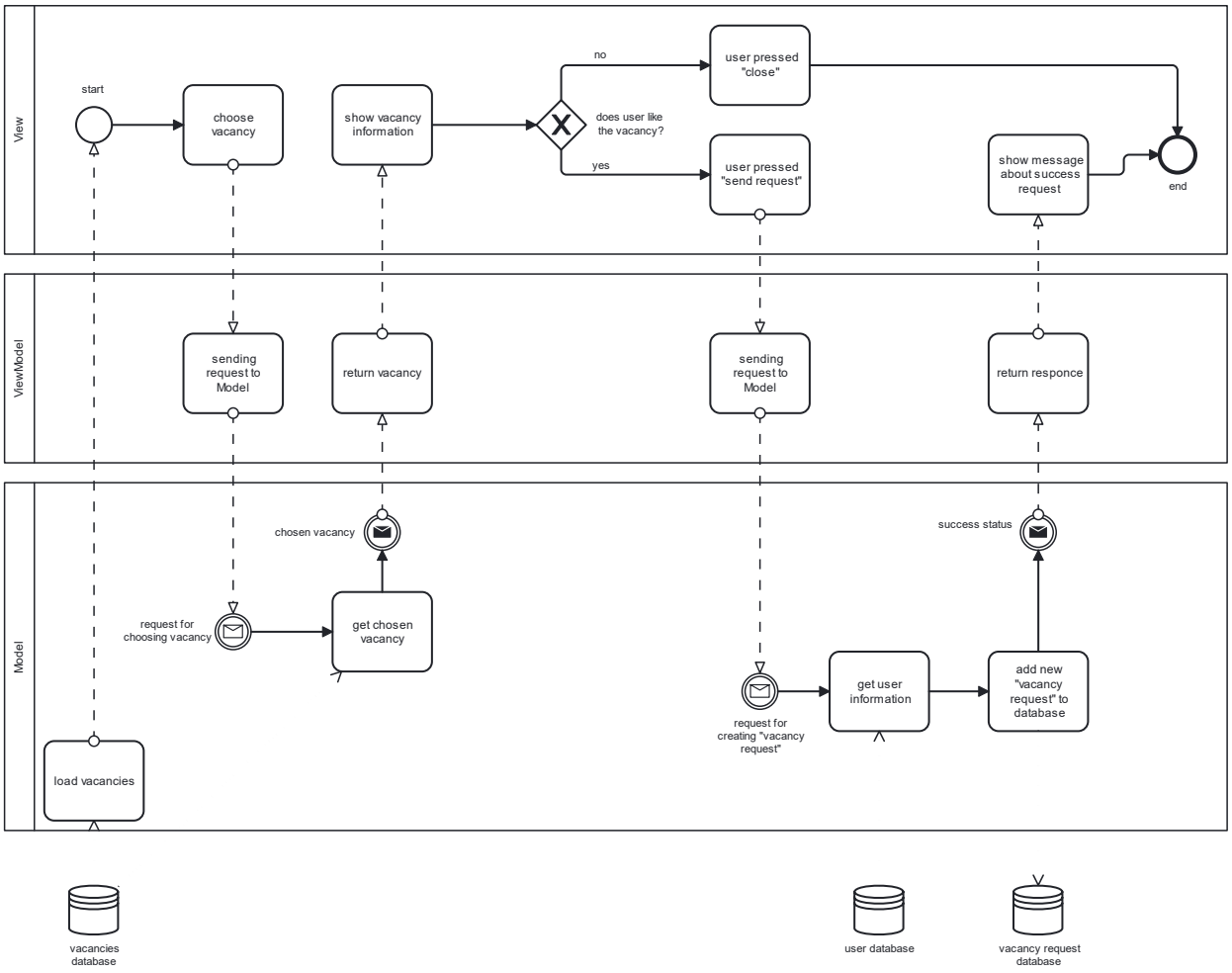


Рисунок 1.9 – BPMN модель бізнес-процесу відгуку на вакансію

Опис послідовності відгуку на вакансію:

– користувач отримує список вакансій;

– користувач вибирає вакансію зі списку;

– користувач отримує у відповідь інформацію про вибрану ним вакансію;

– користувач переглядає вакансію;

– якщо вакансія задовольняє користувача, користувач подає резюме на дану вакансію та закінчує процес;

– інакше, користувач просто закінчує процес.

Розглянемо BPMN модель для опису бізнес-процесу перегляду резюме (рис. 1.10):

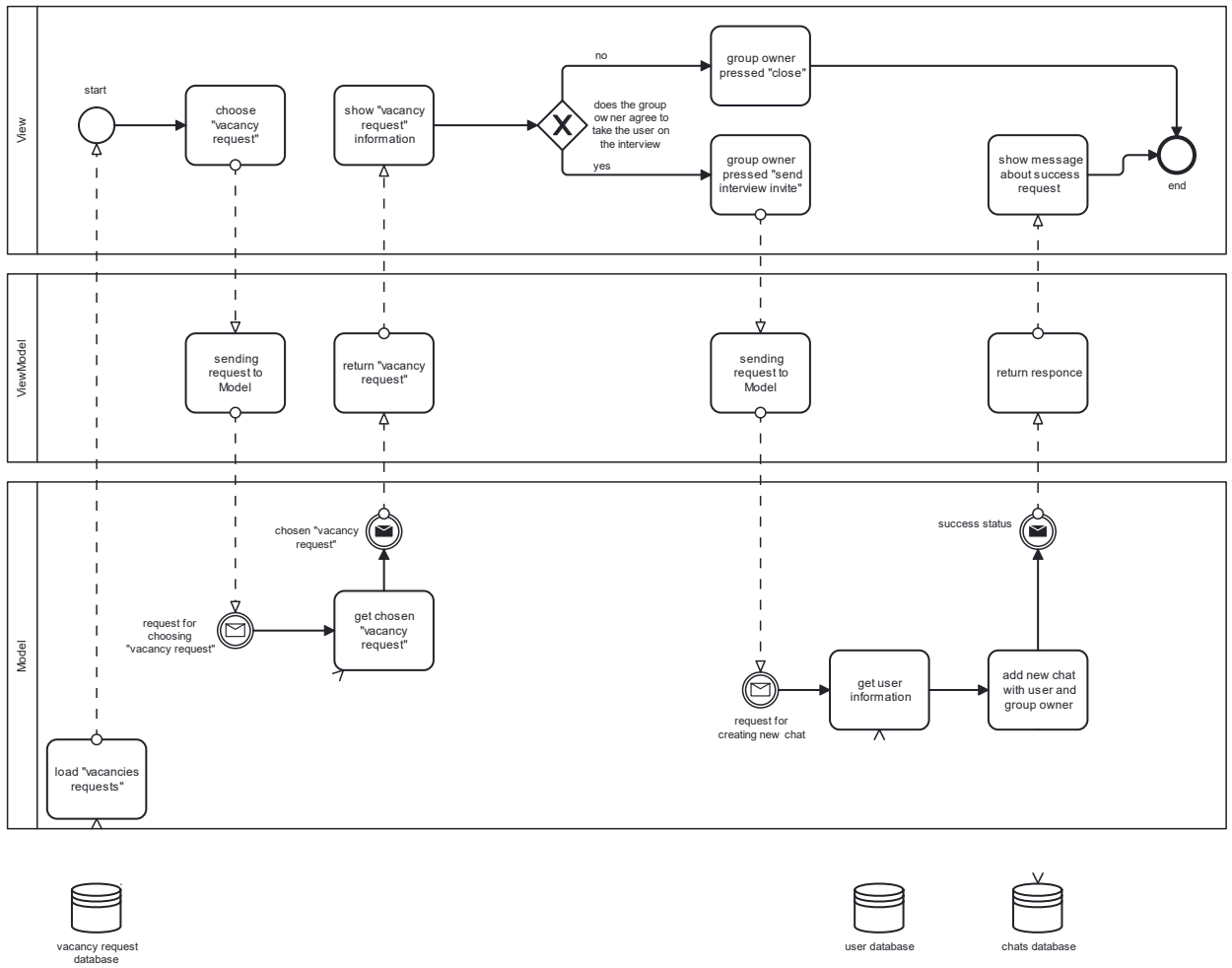


Рисунок 1.10 – BPMN модель бізнес-процесу перегляду резюме

Опис послідовності перегляду резюме:

– власник гурту отримує список надісланих резюме;

– власник гурту вибирає резюме зі списку;

– власник гурту отримує у відповідь інформацію про вибране ним резюме;

– власник гурту переглядає резюме;

– якщо резюме задовольняє власника гурту, власник гурту надсилає користувачу запрошення на співбесіду та створюється чат для спілкування один з одним, процес закінчується;

– інакше, власник гурту просто закінчує процес.

Розглянемо BPMN модель для опису бізнес-процесу роботи чату (рис. 1.11):

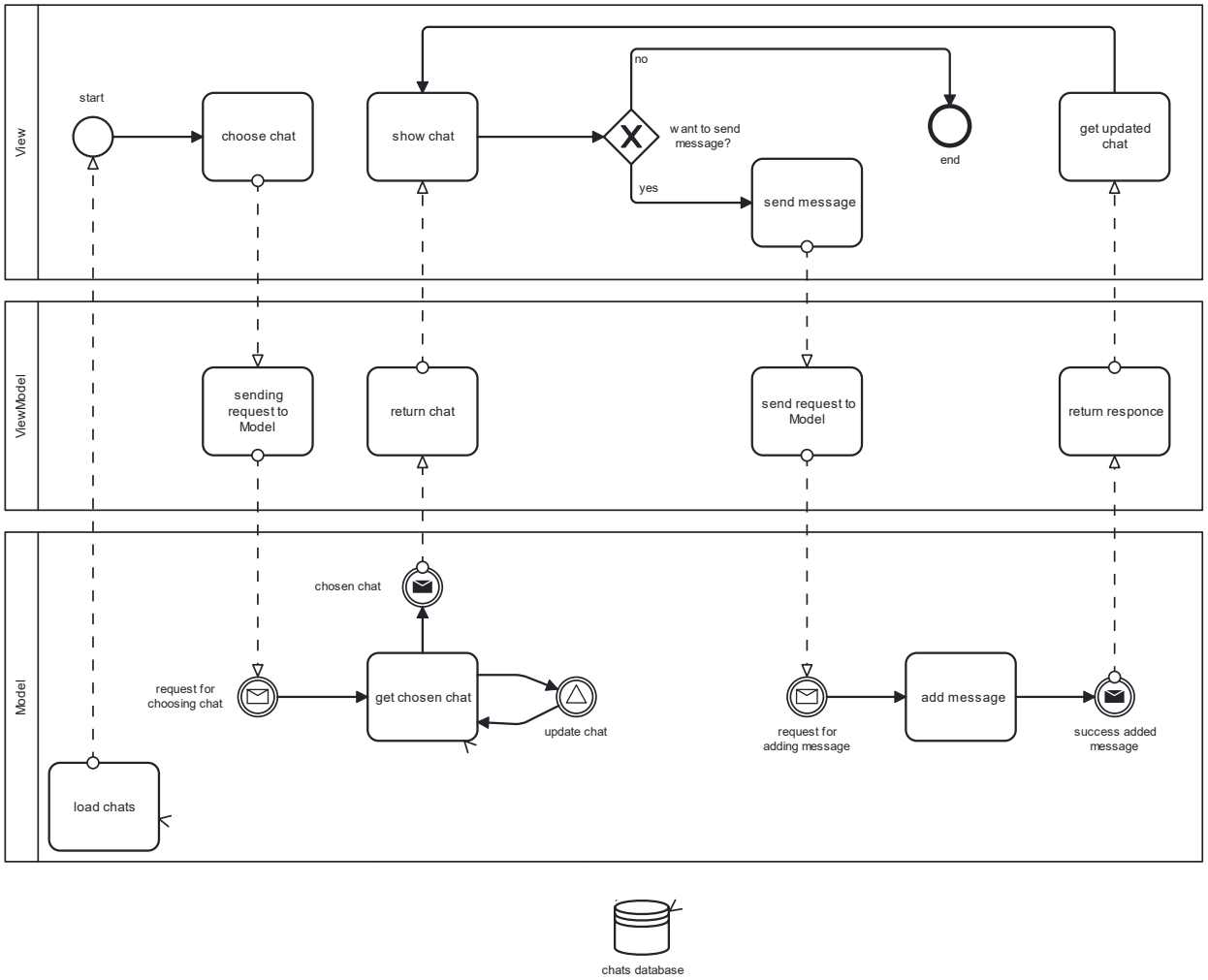


Рисунок 1.11 – BPMN модель бізнес-процесу роботи чату

Опис послідовності роботи чату:

– користувач отримує список своїх чатів;

– користувач вибирає чат зі списку;

– користувач отримує у відповідь увесь чат;

– також відбувається постійна перевірка оновлення чату;

– якщо користувач хоче надіслати повідомлення, він надсилає його;

– повідомлення додається до історії чату;

– повертає успішний статус;

– чат оновлюється, а користувач знову вирішує, чи хоче він надсилати щось;

– якщо так, то дії повторюються;

– якщо ні, то процес закінчується.

Зображені бізнес-процеси на рисунках 1.7-1.11 покривають значну частину функціональних задач. Інші бізнес-процеси не будуть на стільки ж масштабними, тому їх зображено не було.

1.4 Постановка задачі

Метою даної розробки є полегшити процес інтеграції творчих людей у музичній сфері за рахунок розробки застосунку пошуку музичного гурту.

Результатом виконання даного дипломного проєкту повинен бути мобільний застосунок для пошуку музичного гурту, який передбачає можливість створювати акаунти як для звичайних користувачів, так і для власників музичних колективів. Кожен користувач повинен мати змогу редагувати свій профіль. У залежності від типу облікового запису, користувач може або створювати вакансії у свою групу, або ж переглядати наявні вакансії на свою роль та відправляти резюме. Для власників музичних колективів, має бути можливість переглянути усі резюме, які були їм надіслані, та надіслати запрошення кандидатам. Кандидати в свою ж чергу, можуть відповідати та списуватись із власниками груп у особистих чатах. Також передбачити можливість інформувати користувачів про надіслані їм повідомлення.

Висновки до розділу

У результаті виконання даного розділу дипломного проєкту було проведено аналіз предметної області. Розглянуто що таке в загальному мобільний застосунок, а також детальніше про сервіси для пошуку роботи та

музичні гурти. Виконано аналіз стану ринку праці для музикантів на сьогоднішній день та проблеми, з якими вони зіштовхуються.

Проведено дослідження аналогів, їх можливості та виконано порівняння із існуючими конкурентами. Порівняння показало, що існуючі рішення не зовсім покривають предметну область. Через їх широку направленість, саме пошук музичного майбутнього не є актуальним напрямком їх використання.

Для актуальності розробки, було проведено дослідження платформи, яка підійде найбільше. Серед фаворитів Android OS та IOS було обрано Android, оскільки частка ринку перевищує практично втричі. Також, було визначено версії, які будуть підтримуватись. Так як більше 90% усіх Android смартфонів працюють на версії 9.0 Pie і вище, то її було взято за мінімальний рівень підтримки.

Для кращого розуміння побудови Android-додатків, було проведено аналіз архітектурних патернів, які найчастіше використовуються у розробці. Серед найпопулярніших було виділено MVC, MVP та MVVM. Після порівняння, для майбутнього застосунку було обрано використовувати архітектурний патерн MVVM.

Щоб полегшити процес розробки, було виділено 5 основних бізнес-процесів, які покривають значну частину функціоналу мобільного застосунку для пошуку музичного гурту. Для кожного з них, було побудовано BPMN модель та описано послідовності даних бізнес-процесів.

Як результат роботи, в кінці було визначено мету даної розробки, а також сформовано текст умови із усіма загальними задачами, які він повинен покривати.

2 РОЗРОБЛЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Варіанти використання програмного забезпечення

Основною функцією застосунку пошуку музичних гуртів є забезпечення усіх процесів для пошуку роботи в музичних гуртах. До неї відноситься створення вакансій, перегляд наявних вакансій, подання резюме на вакансію, відгук на резюме і також чатування між кандидатом та власником групи. Детальну інформацію щодо другорядних функцій зображено на рисунку 2.1.

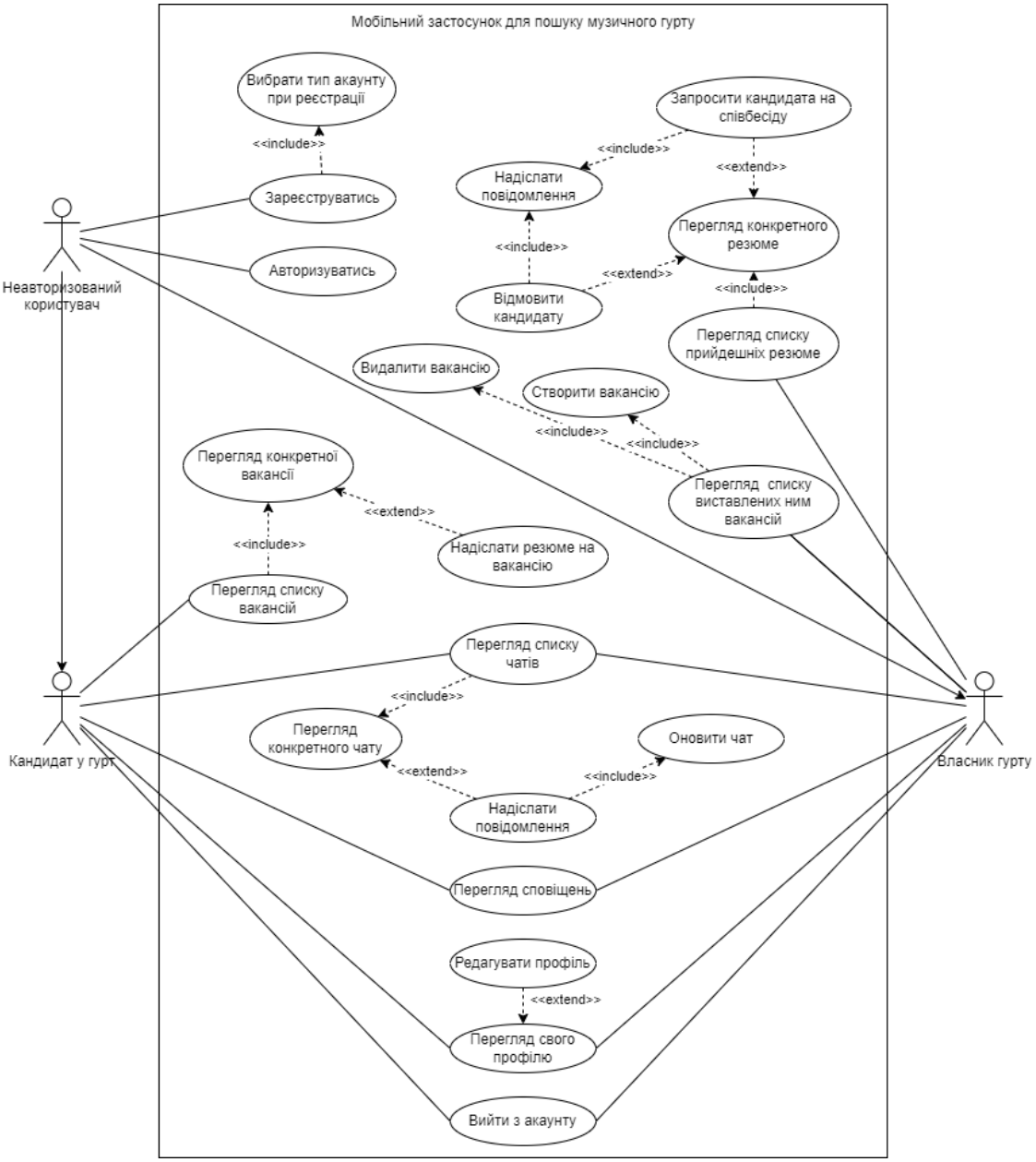


Рисунок 2.1 – Діаграма варіантів використання

У таблицях 2.1 – 2.18 наведені усі варіанти використання програмного забезпечення.

Таблиця 2.1 – Варіант використання UC-1

Use case name	Реєстрація користувача
Use case ID	UC-01
Goals	Реєстрація нового користувача в системі
Actors	Неавторизований користувач
Trigger	Користувач бажає зареєструватися
Pre-conditions	–
Flow of Events	Із сторінки авторизації, користувач переходить на сторінку реєстрації. У відповідні поля, користувач вводить електронну пошту, пароль та пароль підтвердження. Користувач повинен вибрати тип акаунту, який він хоче зареєструвати. Якщо це кандидат у колективи, то ще вводить, ім’я, прізвище, дату народження та місто, в якому знаходиться. Якщо це власник гурту, то вводиться назва гурту, жанр музики та місто. Після заповнення даних, користувач натискає кнопку реєстрації.
Extension	У разі введення невірних даних або ж залишення порожніх полів, користувач повідомляється про відповідну помилку.
Post-Condition	Користувач реєструється в системі та перенаправляється на головну сторінку.

Таблиця 2.2 – Варіант використання UC-2

Use case name	Авторизація користувача
Use case ID	UC-02
Goals	Авторизація користувача в систему
Actors	Неавторизований користувач
Trigger	Користувач бажає авторизуватись
Pre-conditions	У користувача вже є створений обліковий запис
Flow of Events	На сторінці авторизації, користувач вводить електронну пошту та пароль до обліково запису. Далі натискає кнопку входу.
Extension	У разі введення невірних даних або ж залишення порожніх полів, користувач повідомляється про відповідну помилку.
Post-Condition	Користувач авторизується в системі та перенаправляється на головну сторінку.

Таблиця 2.3 – Варіант використання UC-3

Use case name	Перегляд списку чатів
Use case ID	UC-03
Goals	Переглянути список чатів користувача
Actors	Кандидат у гурт, Власник гурту
Trigger	Користувач бажає переглянути чати
Pre-conditions	Користувач увійшов в обліковий запис
Flow of Events	Перейти на вкладку чатів у меню.
Extension	–
Post-Condition	Користувач бачить список його чатів.

Таблиця 2.4 – Варіант використання UC-4

Use case name	Перегляд конкретного чату
Use case ID	UC-04
Goals	Переглянути чат із певним користувачем
Actors	Кандидат у гурт, Власник гурту
Trigger	Користувач бажає переглянути конкретний чат
Pre-conditions	Користувач перейшов на вкладку чатів
Flow of Events	Натиснути на чат, який користувач бажає переглянути, зі списку усіх чатів.
Extension	–
Post-Condition	Користувач бачить історію повідомлень у чаті, який він вибрав.

Таблиця 2.5 – Варіант використання UC-5

Use case name	Надсилання повідомлення
Use case ID	UC-05
Goals	Надіслати повідомлення в чат із певним користувачем
Actors	Кандидат у гурт, Власник гурту
Trigger	Користувач бажає надіслати повідомлення іншому користувачу
Pre-conditions	Користувач відкрив потрібний йому чат
Flow of Events	У поле вводу, користувач вводить текст та натискає кнопку надсилання повідомлення.
Extension	Якщо користувач залишить поле породнім, нічого не відбудеться.
Post-Condition	Користувач бачить, що його повідомлення добавилось до історії повідомлень, оскільки чат оновився.

Таблиця 2.6 – Варіант використання UC-6

Use case name	Перегляд сповіщень
Use case ID	UC-06
Goals	Переглянути список сповіщень
Actors	Кандидат у гурт, Власник гурту
Trigger	Користувач бажає дізнатись, які сповіщення йому приходили
Pre-conditions	–
Flow of Events	Користувач натискає на пункт сповіщень у меню.
Extension	–
Post-Condition	Користувач бачить усі сповіщення, які приходили на аккаунт.

Таблиця 2.7 – Варіант використання UC-7

Use case name	Перегляд профілю
Use case ID	UC-07
Goals	Переглянути свій профіль
Actors	Кандидат у гурт, Власник гурту
Trigger	Користувач бажає переглянути свій профіль
Pre-conditions	–
Flow of Events	Користувач натискає на пункт профіль у меню.
Extension	–
Post-Condition	Користувач бачить свій профіль.

Таблиця 2.8 – Варіант використання UC-8

Use case name	Редагування профілю
Use case ID	UC-08
Goals	Відредагувати інформацію у своєму профілю
Actors	Кандидат у гурт, Власник гурту
Trigger	Користувач бажає відредагувати інформацію свого профілю
Pre-conditions	Користувач знаходить на вкладці профілю
Flow of Events	Користувач натискає на кнопку редагування профілю. Користувач редагує потрібні йому поля та зберігає оновлену інформацію.
Extension	При залишені поля місто порожнім, зберегти оновлення не вдасться.
Post-Condition	Користувача переносить назад на вкладку профілю та він бачить свій оновлений профіль.

Таблиця 2.9 – Варіант використання UC-9

Use case name	Вихід з облікового запису
Use case ID	UC-09
Goals	Вийти з облікового запису користувача
Actors	Кандидат у гурт, Власник гурту
Trigger	Користувач бажає вийти із свого облікового запису
Pre-conditions	–
Flow of Events	Користувач переходить на вкладку профілю, на якій натискає кнопку вийти з профілю. Користувач повинен підтвердити своє бажання залишити профіль.
Extension	При відхилені виходу з профілю, користувач залишиться на вкладці профілю.
Post-Condition	Користувача деавторизує та переносить на вкладку з авторизацією.

Таблиця 2.10 – Варіант використання UC-10

Use case name	Перегляд списку вакансій
Use case ID	UC-10
Goals	Переглянути список доступних вакансій
Actors	Кандидат у гурт
Trigger	Користувач бажає переглянути список підходящих йому вакансій
Pre-conditions	Користувач вказав свою роль у вкладці профілю
Flow of Events	Користувач переходить на вкладку пошуку вакансій.
Extension	При відсутності ролі у користувача, вакансії будуть відсутні.
Post-Condition	Користувач отримує список вакансії, які задовольняють його потреби.

Таблиця 2.11 – Варіант використання UC-11

Use case name	Перегляд конкретної вакансії
Use case ID	UC-11
Goals	Переглянути конкретну вакансію
Actors	Кандидат у гурт
Trigger	Користувач бажає переглянути інформацію про конкретну вакансію
Pre-conditions	Користувач перейшов на вкладку вакансій та отримав список актуальних вакансій
Flow of Events	Користувач вибирає потрібну йому вакансію із загального списку вакансій.
Extension	–
Post-Condition	Користувач переглядає інформацію про потрібну йому вакансію

Таблиця 2.12 – Варіант використання UC-12

Use case name	Надіслати резюме на вакансію
Use case ID	UC-12
Goals	Надіслати резюме власнику гурту
Actors	Кандидат у гурт
Trigger	Користувач бажає надіслати своє резюме на вакансію
Pre-conditions	Користувач вибрав потрібну йому вакансію
Flow of Events	Користувач на вкладці вакансії натискає кнопку для надсилання резюме.
Extension	–
Post-Condition	Власнику гурту надсилається резюме з даними кандидата. Користувач повертається до списку усіх вакансій.

Таблиця 2.13 – Варіант використання UC-13

Use case name	Переглянути список створених вакансій
Use case ID	UC-13
Goals	Переглянути список створених вакансій музичного гурту
Actors	Власник гурту
Trigger	Користувач бажає переглянути існуючі вакансії
Pre-conditions	–
Flow of Events	Користувач переходить на вкладку пошуку кандидатів та натискає на кнопку «Мої вакансії».
Extension	При відсутності виставлених вакансій, буде порожня сторінка.
Post-Condition	Користувача переносить на нову сторінку із створеними ним вакансіями.

Таблиця 2.14 – Варіант використання UC-14

Use case name	Створити вакансію
Use case ID	UC-13
Goals	Створити нову вакансію в гурт
Actors	Власник гурту
Trigger	Користувач бажає розмістити нову вакансію
Pre-conditions	Користувач перейшов на сторінку вакансії.
Flow of Events	Користувач натискає на кнопку «створити вакансію». У новому вікні користувач вибирає роль, яку він хоче знайти та бажаний досвід володіння музичним інструментом чи роботи.
Extension	При залишенні полів порожніми, користувач не зможе створити вакансію.
Post-Condition	Користувач створює нову вакансію та переноситься на екран з списком створених вакансій.

Таблиця 2.15 – Варіант використання UC-15

Use case name	Видалити вакансію
Use case ID	UC-15
Goals	Видалити наявну вакансію в гурт
Actors	Власник гурту
Trigger	Користувач бажає видалити наявну вакансію, користувач уже знайшов кандидата.
Pre-conditions	Користувач перейшов на сторінку вакансії.
Flow of Events	Користувач натискає на кнопку видалення вакансії. У новому вікні користувач підтверджує видалення вакансії.
Extension	При відхилені видалення, нічого не відбудеться.
Post-Condition	Користувач видаляє свою вакансію та переноситься на сторінку з оновленим списком його вакансій.

Таблиця 2.16 – Варіант використання UC-16

Use case name	Перегляд списку прийдешніх резюме
Use case ID	UC-16
Goals	Переглянути список резюме, які надійшли
Actors	Власник гурту
Trigger	Користувач бажає переглянути список резюме, які надіслали йому кандидати
Pre-conditions	–
Flow of Events	Користувач переходить на вкладку пошуку кандидатів.
Extension	При відсутності надісланих резюме, нічого відображатись не буде.
Post-Condition	Користувач отримує список резюме, які надіслали йому кандидати.

Таблиця 2.17 – Варіант використання UC-17

Use case name	Перегляд конкретного резюме
Use case ID	UC-17
Goals	Переглянути інформацію про конкретне резюме
Actors	Власник гурту
Trigger	Користувач бажає переглянути інформацію про резюме із списку
Pre-conditions	–
Flow of Events	Користувач переходить на вкладку пошуку кандидатів, де із списку надісланих йому резюме вибирає потрібне.
Extension	–
Post-Condition	Користувач переходить на нову сторінку із інформацією про потрібне йому резюме.

Таблиця 2.18 – Варіант використання UC-18

Use case name	Відповідь на резюме
Use case ID	UC-18
Goals	Надати відповідь на резюме
Actors	Власник гурту
Trigger	Користувачу прийшло нове резюме на його вакансію
Pre-conditions	–
Flow of Events	Користувач відкриває резюме, яке йому надійшло та обирає чи запросити кандидати на співбесіду, чи відхилити його заявку.
Extension	В обох випадках, з’являється попередження про вибрану дію.
Post-Condition	Автоматично формується повідомлення із запрошенням або відхиленням в залежності від вибраного пункту та надсилається кандидату. Користувач переноситься на вкладку із списком резюме.

2.2 Розроблення функціональних вимог

Для зручності розробки програмне забезпечення буде розділене на свого роду модулі, які дозволять поетапно розробляти набір функцій для кожного з них. У таблиці 2.19 наведено загальну модель вимог програмного

забезпечення, а в таблиці 2.20 наведений детальний опис кожної функціональної вимоги. У результатів було створено матрицю трасування вимог (рис. 2.2).

Таблиця 2.19 – Загальна модель вимог

№	Назва	ID вимоги	Пріоритети	Ризики
1	Система авторизації користувача	FR-1	Високий	Високий
1.1	Реєстрація облікового запису	FR-2	Високий	Високий
1.2	Вхід в обліковий запис	FR-3	Високий	Високий
1.3	Вихід з облікового запису	FR-4	Середній	Низький
2	Налаштування профілю користувача	FR-5	Середній	Низький
2.1	Зміна інформації про користувача	FR-6	Середній	Низький
2.1.1	Зміна ролі користувача	FR-7	Середній	Низький
2.1.2	Встановлення аватару користувача	FR-8	Низький	Низький
2.2	Зміна посилань на відео	FR-9	Низький	Низький
3	Система чату	FR-10	Середній	Низький
3.1	Перегляд списку чатів	FR-11	Середній	Низький
3.2	Відкриття конкретного чату	FR-12	Середній	Низький
3.2.1	Відображення історії чату	FR-13	Середній	Низький
3.3	Надсилання текстового повідомлення	FR-14	Середній	Середній
4	Отримання сповіщень	FR-15	Низький	Низький
5	Система роботи з вакансіями	FR-16	Високий	Високий
5.1	Отримання списку вакансій	FR-17	Високий	Низький
5.1.1	Отримання списку вакансій для кандидата	FR-18	Високий	Низький

Продовження таблиці 2.19

№	Назва	ID вимоги	Пріоритети	Ризики
5.1.2	Отримання списку вакансій для власника гурту	FR-19	Середній	Низький
5.2	Перегляд вибраної вакансії	FR-20	Високий	Низький
5.2.1	Надіслати резюме на вакансію	FR-21	Високий	Високий
5.3	Створити вакансію	FR-22	Високий	Середній
5.4	Видалити вакансію	FR-23	Низький	Високий
6	Система обробки резюме	FR-24	Високий	Високий
6.1	Отримання списку надісланих резюме	FR-25	Високий	Низький
6.2	Переглянути конкретне резюме	FR-26	Високий	Низький
6.3	Відповісти на резюме	FR-27	Високий	Високий
6.3.1	Позитивно відповісти на резюме	FR-28	Середній	Середній
6.3.2	Відхилити резюме	FR-29	Середній	Середній
6.4	Надіслати повідомлення кандидату із результатом	FR-30	Середній	Низький
6.4.1	Надіслати запрошення у разі успіху	FR-31	Середній	Низький
6.4.2	Надіслати відмову у разу невдачі	FR-32	Низький	Низький

Таблиця 2.20 – Перелік функціональних вимог

ID вимоги	Назва та опис
FR-1	Система авторизації користувача Система, яка дозволяє здійснювати маніпуляції із обліковими записами користувачів, такі як авторизація, реєстрація та вихід.
FR-2	Реєстрація облікового запису Неавторизований користувач може створити новий обліковий запис у системі, перейшовши на сторінку реєстрації.
FR-3	Вхід в обліковий запис Неавторизований користувач може увійти у свій обліковий запис із початкової сторінки.
FR-4	Вихід з облікового запису Авторизований користувач має змогу із сторінки профілю залишити свій обліковий запис.
FR-5	Налаштування профілю користувача Система має передбачити зміни в даних користувача.
FR-6	Зміна інформації про користувача Користувач може змінити дані про себе такі як прізвище, ім’я, місто, досвід.
FR-7	Зміна ролі користувача Користувач повинен мати змогу змінювати свою роль в залежності від бажаної.
FR-8	Встановлення аватару користувача Користувач може змінити своє фото облікового запису та встановити власне.
FR-9	Зміна посилань на відео Користувач може змінити посилання на відео із своїми прикладами музичних робіт.

Продовження таблиці 2.20

ID вимоги	Назва та опис
FR-10	Система чату Кандидати в гурти можуть спілкуватись з власниками гуртів в особистих чатах.
FR-11	Перегляд списку чатів При переході на сторінку з чатами, користувач має бачити список із своїх чатів з іншими користувачами.
FR-12	Відкриття конкретного чату При виборі чату зі списку чатів, користувач має переноситись у нове вікно вибраного чату.
FR-13	Відображення історії чату При відкритті чату, автоматично повинна завантажуватись історія попередніх повідомлень для цього чату.
FR-14	Надсилання текстового повідомлення У відкритому чаті має бути текстове поле для вводу повідомлення. При натисканні на кнопку надсилання, повідомлення повинно відобразитись в історії чату, текстове поле очиститись.
FR-15	Отримання сповіщень У випадках присилання резюме або відповіді на вакансії, користувач повинен отримувати повідомлення на вкладці сповіщень.
FR-16	Система роботи з вакансіями Користувач повинен мати можливість створювати, обробляти, переглядати та видаляти вакансії в залежності від типу користувача.
FR-17	Отримання списку вакансій Користувач повинен мати змогу переглядати наявні вакансії.

Продовження таблиці 2.20

ID вимоги	Назва та опис
FR-18	Отримання списку вакансій для кандидата Кандидат може переглядати вакансії, які підходять до його ролі, міста та досвіду.
FR-19	Отримання списку вакансій для власника гурту Власник гурту може переглядати вакансії, які він ж і створив.
FR-20	Перегляд вибраної вакансії При виборі вакансій із списку, повинна відкриватись сторінка із інформацією про вакансію.
FR-21	Надіслати резюме на вакансію На сторінці вакансії, для кандидата має бути можливість надіслати своє резюме, яке автоматично генерується із даних, які він вказав у своєму профілі.
FR-22	Створити вакансію Власник гурту має право створити вакансію у свій гурт із вказаною роллю.
FR-23	Видалити вакансію Власник гурту має право видалити вакансію у свій гурт в раз неактуальності вакансії або ж знаходження кандидата на вакансію.
FR-24	Система обробки резюме Користувач має право на надсилання, перегляд та відповідь на резюме, у залежності від типу користувача.
FR-25	Отримання списку надісланих резюме Власник гурту може переглянути список усіх резюме, які надіслали на його вакансії.

Продовження таблиці 2.20

ID вимоги	Назва та опис
FR-26	Переглянути конкретне резюме Власник гурту може переглянути конкретне резюме із усього списку та дізнатись інформацію з нього.
FR-27	Відповісти на резюме Власник гурту, для закриття резюме, має дати відповідь на резюме.
FR-28	Позитивно відповісти на резюме У випадку позитивної відповіді, власник гурту показує своє бажання запросити кандидата на співбесіду.
FR-29	Відхилити резюме У випадку відхилення резюме, власник гурту показує, що кандидат його не задовольняє.
FR-30	Надіслати повідомлення кандидату із результатом Після вибору відповіді на резюме, автоматично генерується повідомлення із відповіддю та надсилається кандидату.
FR-31	Надіслати запрошення у разі успіху Якщо власник бажає запросити кандидата на співбесіду, кандидат отримає повідомлення із запрошенням.
FR-32	Надіслати відмову у разі невдачі Якщо власник не бажає запрошувати кандидата, кандидат отримає повідомлення, про відмову у відборі.

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	FR-16	FR-17	FR-18	FR-19	FR-20	FR-21	FR-22	FR-23	FR-24	FR-25	FR-26	FR-27	FR-28	FR-29	FR-30	FR-31	FR-32
UC-01	+	+																														
UC-02	+		+																													
UC-03										+	+																					
UC-04										+		+	+																			
UC-05										+				+																		
UC-06															+																	
UC-07					+																											
UC-08					+	+	+	+	+																							
UC-09	+			+																												
UC-10																+	+	+	+													
UC-11																+				+												
UC-12																+					+			+								
UC-13																+			+													
UC-14																+						+										
UC-15																+							+									
UC-16																								+	+							
UC-17																								+		+						
UC-18																								+			+	+	+	+	+	+

Рисунок 2.2 – Матриця трасування вимог

2.3 Розроблення нефункціональних вимог

Усі нефункціональні вимоги, які висуваються до застосунку пошуку музичного гурту наведені у таблиці 2.21.

Таблиця 2.21 – Перелік нефункціональних вимог

Назва вимоги	Опис вимог
Продуктивність	Час відгуку застосунку на дії користувача повинен бути мінімальним.
Надійність	Застосунок повинен бути доступний практично завжди, щоб користувачі довіряли йому.
Безпека	Дані користувачів повинні бути у безпеці та захищенні від крадіжки.
Зручність у використанні	Застосунок повинен бути інтуїтивно зрозумілим.
Портативність	Застосунок повинен підтримуватись на всіх версіях операційної системи Android 9.0 Pie і вище.

Висновки до розділу

У цьому розділі дипломного проєкту були розроблені вимоги, які висуваються до мобільного застосунку для пошуку музичного гурту.

Для кращого розуміння усіх процесів, які будуть відбуватись всередині застосунку, було створено діаграму варіантів використання, на якій зображені усі дії, які можуть виконувати актори (користувачі розробленого програмного забезпечення). На основі діаграми варіантів використання, були визначенні усі варіанти використання (use cases), надані їм унікальні ідентифікатори та прописані усі умови виконання, тригери, послідовні кроки виконання та результати їх виконання.

З метою полегшення процесу розробки програмного забезпечення, була розроблена загальна модель функціональних вимог, яка дозволить розробнику чітко розуміти усі поставленні перед ним задачі. Кожній функціональній вимозі було надано унікальний ідентифікатор та додано додатковий опис самої вимоги.

На основі усіх варіантів використань та функціональних вимог, було створено матрицю трасувань вимог, яка дає можливість чітко зрозуміти, які варіанти використань, яких функціональних вимог потребують.

Також, для покращення вражень користувача від використання розробленого програмного забезпечення, були розроблені нефункціональні вимоги, дотримання яких забезпечить збільшення аудиторії.

За результатами розділу сформовано технічне завдання на розробку програмного забезпечення для пошуку музичного гурту.

3 КОНСТРУЮВАННЯ ТА РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Архітектура програмного забезпечення

У якості архітектурного патерну для розробки музичного застосунку з пошуку музичного гурту, як було сказано у 1 розділі, обрано MVVM. Розглянемо детально усі складові даного шаблону проектування:

Модель (Model) – об’єкти, які обробляють усю бізнес-логіку застосунку та керують даними. У даному застосунку, у моделях будуть проводитись різного роду маніпуляції із отриманими даними із бази даних для забезпечення правильності подання користувачу. Також, усі сутності бази даних відносяться до моделі.

Представлення (View) – об’єкти, які забезпечують відображення усіх елементів на екрані користувача та отримання вводу від користувача (дотики, натискання, свайпи і т.д.). У нашому випадку, представлення будуть виконувати відображення різних екранів, таких як екран авторизації, реєстрації, профілю, головної сторінки, чатів, конкретного чату, сповіщень, пошуку вакансій та перегляду резюме.

Модель представлення (ViewModel) – забезпечує зв’язок між представленням та моделлю. Перетворює дані від представлення до моделі і від моделі до представлення у зручний для них вид. Також забезпечує логіку інтерфейсу, тобто обробляє усі дії користувача з представленням. У мобільному застосунку для пошуку музичного гурту, виконує ключову роль у зв’язку між об’єктами та між різними представленнями (обробка переходів між представленнями) [7].

У даному випадку, розроблену архітектуру зручно буде подати у вигляді діаграми пакетів, яка зображена на рисунку 3.1.

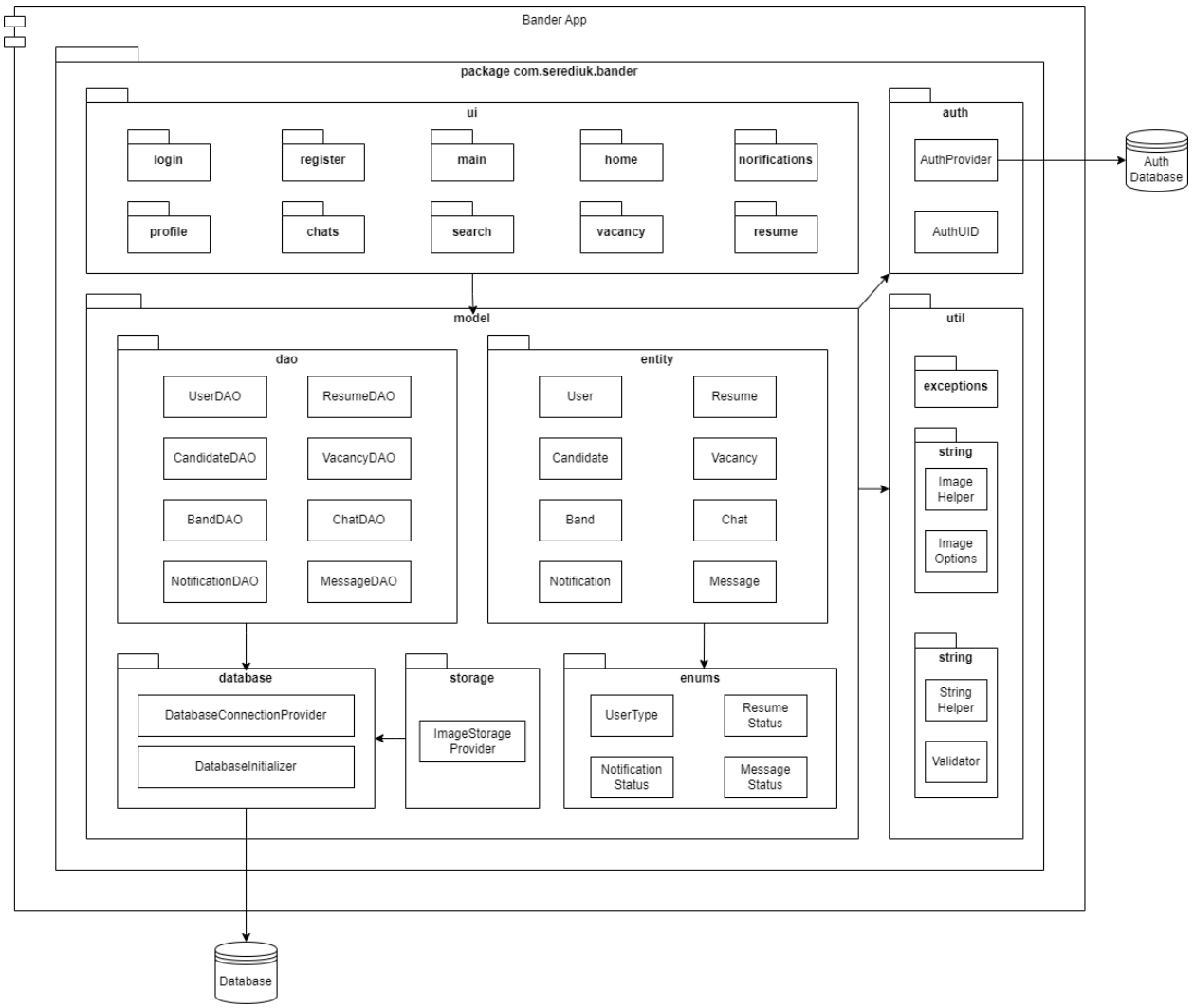


Рисунок 3.1 – Діаграма пакетів

Дана діаграма пакетів показує, як має виглядати структура проекту під час розробки. Пакет ui, який містить в собі ще інші пакети – це пакет для збережень представлень та моделі представлень. Тобто у ньому зберігаються усі класи, які відповідають за користувацький інтерфейс, а також їх обробку.

Пакет model – пакет, у якому зберігаються класи сутностей бази даних, а також логіка бізнес-процесів. Відповідає моделі в архітектурному патерні MVVM. Сама ж модель розділена на декілька пакетів. Є два допоміжних пакета, а саме auth та util.

Об’єкти пакету auth використовуються для здійснення авторизації за допомогою токенів авторизації, що забезпечує надійність та безпеку для користувачів застосунку. Варто відмітити, що його логіка реалізована так, що користувач зберігається в системі та не потребує постійного входу в обліковий запис. Об’єкти пакету util – це допоміжні об’єкти, які не виділяються із себе

окрему архітектурну одиницю, а лише слугують для спрощення розробки застосунку.

Пакет dao є надзвичайно важливим, оскільки він є частиною моделі, але його обов’язки це робота із даними. Саме він забезпечує отримання даних із бази даних, які потім обробляються у моделі та подаються користувачу.

Варто зазначити, що даний застосунок розробляється без серверної частини. Такий тип архітектури часто називають безсерверною (serverless architecture).

Безсерверна архітектура — це модель розробки програмного забезпечення, при якій постачальник хмарних послуг динамічно керує виділенням ресурсів і виконанням коду. Ця модель дає можливість розробникам зосередитися на написанні коду, не турбуючись про управління серверною інфраструктурою [9].

До переваг безсерверної архітектури відносять:

– Швидкість розробки: розробники можуть швидше створювати і розгортати застосунки, зосередившись на бізнес-логіці, а не на інфраструктурі.

– Зниження витрат: платіжна модель "оплата за використання" дозволяє значно зменшити витрати, особливо для застосунків зі змінним або рідкісним навантаженням.

– Автоматичне масштабування: динамічне масштабування без необхідності ручного втручання забезпечує високу доступність і продуктивність [9].

У нашому випадку, це дозволить приділити більше часу важливішим компонентам при розробці застосунків, а також використовувати вже готові та надійні рішення. Одним із таких є використання сервісу Firebase.

Firebase – це набір хмарних backend-сервісів і платформ для розробки додатків від Google. Він містить бази даних, сервіси, засоби автентифікації та інтеграції для різноманітних додатків.

Використання Firebase дозволить нам, як розробнику скористатись сервісами Firebase Authentication, Firebase Realtime Database та Firebase Storage. Дані сервіси закривають такі нефункціональні вимоги, як надійність, безпека та продуктивність.

Firebase Realtime Database – це хмарна база даних, яка зберігає дані у форматі JSON та синхронізує їх в режимі реального часу з усіма підключеними клієнтами. Всі користувачі мають доступ до єдиного екземпляра бази даних і автоматично отримують найсвіжіші оновлення [10].

Сама ж база даних відноситься до типу NoSQL, тобто не реляційна. Дані зберігаються у вигляді єдиного великого дерева JSON, де кожен вузол дерева є об'єктом, а кожне значення може бути примітивним типом, масивом або іншим об'єктом. Перевагами такого підходу є швидкість опрацювання, гнучкість та завантаження у режимі реального часу в порівнянні із звичними реляційними базами даних [11].

Firebase Realtime Database використовує власну мову правил безпеки, що дозволяє визначати, хто має доступ до певних даних і яким чином ці дані можуть бути змінені. Це забезпечує захист інформації від несанкціонованого доступу.

Технологічний стек, який склався на даний момент, наведено в таблиці 3.1.

Таблиця 3.1 – Технологічний стек

Тип технології	Назва технології
Операційна система	Android OS
Архітектурний патерн	MVVM
Хмарний backend-сервіс	Firebase
База даних	Firebase Realtime Database
Сервіс аутентифікації	Firebase Authentication

3.2 Обґрунтування вибору засобів розробки

Для завершення технологічного стеку, потрібно визначитись із мовою програмування та середовищем розробки. Серед IDE, виділимо найпопулярніші, а саме Android Studio, IntelliJ IDEA, Eclipse with ADT.

Android Studio – це офіційне інтегроване середовище розробки (IDE) від компанії JetBrains для проектування, розробки, відлагодження та тестування додатків для Android. Android Studio володіє багатьма властивостями та особливостями, які забезпечують збільшення продуктивності та ефективності під час створення додатків, таких як:

а) гнучка система збірки проектів, що базується на Gradle;
б) продуктивний та мультифункціональний Android-емулятор;
в) редагування в реальному часі застосунку для оновлення як на емуляторах, так і на фізичних пристроях;
г) широкі інструменти та фреймворки для тестування;
д) вбудована підтримка сервісів Google, що дозволяє легко підключити до проекту сервіси Firebase [12].

Android SDK – це комплект для розробки програмного забезпечення для програмної екосистеми Android, який включає в себе набір інструментів розробки, зокрема відладчик, бібліотеки, емулятор телефону, документацію, приклади коду та навчальні посібники. SDK є частиною офіційного середовища розробки Android Studio, але його різноманітні інструменти та ресурси можна використовувати незалежно [13].

IntelliJ IDEA — це офіційне IDE, розроблене тією ж компанією JetBrains, яке найчастіше використовується під час написання різного роду проектів мовами програмування Java та Kotlin. IntelliJ IDEA відома своєю потужною функціональністю, високим рівнем продуктивності та розширеними інструментами для підвищення ефективності розробників. Так як, Android Studio засновано поверх IntelliJ IDEA, то Android Studio має аналогічний функціонал, який доповнюється спеціальними плагінами під Android розробку [14].

Eclipse with ADT (Android Development Tools) — це набір плагінів для Eclipse, який раніше використовувався для розробки Android-застосунків до появи Android Studio. ADT забезпечував інтеграцію інструментів Android SDK з Eclipse, роблячи його повноцінною IDE для розробки Android-додатків. На даний момент, воно не є актуальним через появу Android Studio [15].

Порівнюючи із конкурентами, Android Studio є явним фаворитом серед IDE. Щодо мови програмування, то вибір залишається між Java та Kotlin

Java — це об'єктно-орієнтована мова програмування, випущена у 1995 році компанією Sun Microsystems (котра зараз належить Oracle). Щодо головних переваг використання Java для розробки мобільних додатків є її швидкість, простота та платформонезалежність. Тобто програми, написані мовою Java, можуть бути запущені на різних платформах без необхідності додаткових змін в майбутньому, а логіку застосунку можна буде використовувати для інших операційних систем [16].

Для мобільних додатків, які призначені для Android, Java є однією з основних мов програмування, яку підтримує ця платформа. Більше того, мова Java має надзвичайно велику та не менш активну спільноту розробників, що забезпечує її розвиток та доступ до великої кількості різноманітних ресурсів, бібліотек та інструментів розробки. Її об'єктно-орієнтований підхід дозволяє створювати складні, але добре організовані додатки з більшою ефективністю та легкістю обслуговування. Також Java відома своєю надійністю, високою швидкодією та широким спектром доступних інструментів, що сприяє розвитку продуктивних та функціональних мобільних додатків.

JDK (Java Development Kit) – це набір інструментів, необхідних для розробки, компіляції та виконання застосунків, написаних мовою програмування Java. Набір включає в себе компілятор Java, бібліотеки, документацію та інші корисні ресурси для розробників [16].

Kotlin – це молода, статично-типізована мова програмування, яка побудована на основі мови Java, тобто використовує Java Virtual Machine (JVM) для запуску і може бути альтернативою Java для розробки Android-

додатків. Вона лаконічна, безпечна, сумісна з Java та іншими мовами і забезпечує можливості для ефективного використання коду між різними платформами [17].

Для детального порівняння двох мов програмування скористаємось таблицею 3.2 [18].

Таблиця 3.2 – Порівняння мов програмування Java та Kotlin

Характеристика	Java	Kotlin
Сумісність та екосистема	Поширена і стабільна екосистема, використовується багато років, велика кількість бібліотек і фреймворків	Повна сумісність з Java, але сама екосистема ще розвивається
Продуктивність виконання	Висока продуктивність завдяки стабільному JVM, відлагоджена робота протягом десятиліть	Подібна продуктивність до Java, але корутини можуть додати деякі витрати на керування
Популярність та підтримка	Широке використання у великих проектах, велика спільнота, офіційна мова для Android до 2017 року	Зростаюча популярність, але менша спільнота у порівнянні з Java
Стабільність та зрілість	Зрілий та стабільний мовний стандарт, перевірений часом	Сучасніша мова, яка все ще отримує нові оновлення
Підтримка великих проектів	Використовується у багатьох великих корпоративних проектах, зрілі рішення для масштабування	Відмінно підходить для нових проектів, але менше історій успіху у великих компаніях

Продовження таблиці 3.2

Характеристика	Java	Kotlin
Інтеграція з Android	Офіційна мова для розробки під Android до Kotlin, все ще широко використовується	Офіційно підтримувана мова для Android з 2017 року, хороша інтеграція з Android Studio
Офіційна підтримка та ресурси	Велика кількість офіційних ресурсів, документація від Oracle та інших компаній	Офіційна підтримка від JetBrains та Google, але менше ресурсів у порівнянні з Java
Підтримка підприємств	Широке використання в корпоративному світі, підтримка від великих компаній як Oracle, IBM	Зростаюче прийняття у підприємствах, але все ще менше у порівнянні з Java

Зважаючи на результати порівняння, було обрано використовувати мову програмування Java.

3.3 Конструювання програмного забезпечення

Так як Java це об’єктно-орієнтована мова програмування, то програмне забезпечення буде складати із набору класів, які взаємо пов’язані між собою. Детальний опис основних класів наведено у таблиці 3.3.

Таблиця 3.3 – Опис основних класів мобільного застосунку

№	Ім’я класу	Опис класу
1	User	Клас описує користувача, слугує батьківським класом для кандидати та власника гурту
2	Candidate	Клас описує кандидата в музичний гурт, містить інформацію про кандидата.
3	Group	Клас описує музичного гурту, містить інформацію про музичний гурт.

Продовження таблиці 3.3

№	Ім’я класу	Опис класу
4	Chat	Клас описує чат між користувачами, містить історію повідомлень між двома користувачами.
5	Message	Клас описує одне сповіщення, містить усю необхідну інформацію про сповіщення.
6	Vacancy	Клас описує вакансію, містить інформацію про вакансію.
7	Resume	Клас описує резюме, містить інформацію про резюме.
8	Notification	Клас описує сповіщення, містить інформацію про сповіщення.
9	AuthProvider	Клас, який надає функціонал для авторизації та всіх дій пов’язаних з нею. При створені під’єднується до сервісу Firebase Authentication.
10	DatabaseConnection Provider	Клас, який встановлює з’єднання із базою даних Firebase Realtime Database. Містить інформацію про з’єднання із БД.
11	CandidateDAO	Клас, який дозволяє здійснити доступ до даних кандидатів із бази даних. Дозволяє виконувати певні маніпуляції із даними.
12	GroupDAO	Клас, який дозволяє здійснити доступ до даних гуртів та складу гуртів із бази даних. Дозволяє виконувати певні маніпуляції із даними.
13	ChatDAO	Клас, який дозволяє здійснити доступ до даних чатів та історії чатів із бази даних. Дозволяє виконувати певні маніпуляції із даними.
14	VacancyDAO	Клас, який дозволяє здійснити доступ до даних вакансій із бази даних. Дозволяє виконувати певні маніпуляції із даними.

Продовження таблиці 3.3

№	Ім’я класу	Опис класу
15	ResumeDAO	Клас, який дозволяє здійснити доступ до даних резюме із бази даних. Дозволяє виконувати певні маніпуляції із даними.
16	NotificationDAO	Клас, який дозволяє здійснити доступ до даних сповіщень із бази даних. Дозволяє виконувати певні маніпуляції із даними.

Усі допоміжні класи, та класи реалізації View та ViewModel були упущені, оскільки вони не відіграють ролі в бізнес логіці. Для кращого розуміння усієї структури програмного забезпечення, можна скористатись діаграмою пакетів, яка зображена на рисунку 3.1.

Як вже згадувалось раніше, в якості сервісу для роботи з базами даних було обрано сервіс Firebase Realtime Database. Сама база даних призначена для зберігання даних про користувачів, усіх дій та документів, які вони виконують. Хоч дані в ньому і зберігаються у вигляді JSON-файлів, але для кращого розуміння структури, побудуємо ER-діаграму (рис. 3.2). Опис усіх сутностей наведено у таблицях 3.4 – 3.11.

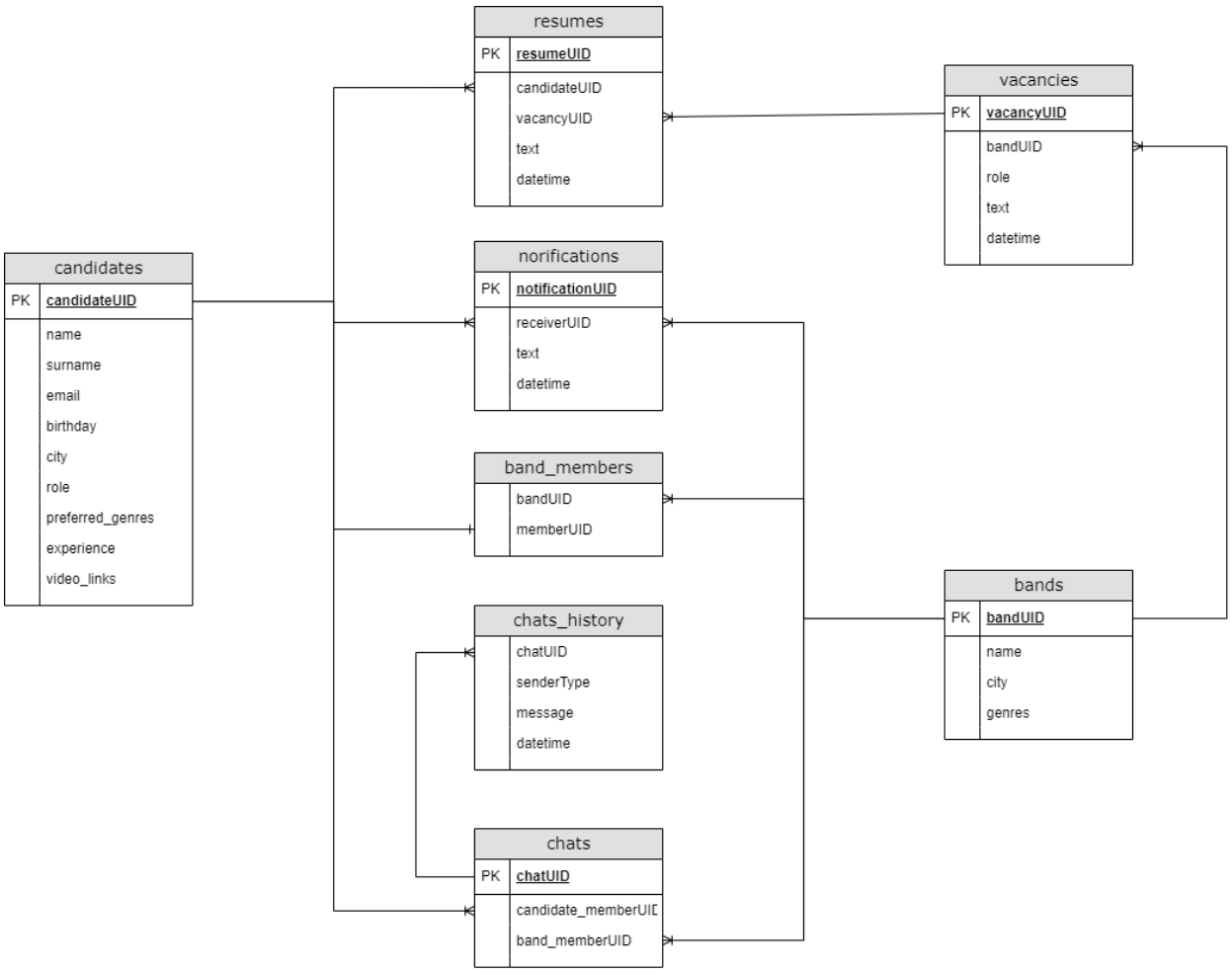


Рисунок 3.2 – ER-діаграма сутностей бази даних

Таблиця 3.4 – Опис сутності bands

Назва поля	Тип даних	Опис
bandUID	serial PK	Унікальний ідентифікаційний номер гурту
name	varchar	Назва гурту
city	varchar	Локація гурту
genres	varchar	Жанри гри музичного колективу

Таблиця 3.5 – Опис сутності candidates

Назва поля	Тип даних	Опис
candidateUID	serial PK	Унікальний ідентифікаційний номер кандидата
name	varchar	Ім’я користувача
surname	varchar	Прізвище користувача
email	varchar	Електронна адреса користувача
birthday	date	Дата народження користувача
city	varchar	Місто перебування користувача
role	varchar	Роль в музичному гурті, яку користувач може виконувати
preferred_genres	varchar	Жанри, які користувач хотів би виконувати
experience	float	Досвід користувача на певній ролі (подається як кількість років у дробовому числі)
video_links	varchar	Посилання на приклади виступів користувача

Таблиця 3.6 – Опис сутності band_members

Назва поля	Тип даних	Опис
bandUID	varchar	Ідентифікаційний номер гурту
memberUID	varchar	Унікальний ідентифікаційний номер кандидата

Таблиця 3.7 – Опис сутності chats

Назва поля	Тип даних	Опис
chatUID	serial PK	Унікальний ідентифікаційний номер чату
candidate_memberUID	varchar	Ідентифікаційний номер кандидата
band_memberUID	varchar	Ідентифікаційний номер гурту

Таблиця 3.8 – Опис сутності chats_history

Назва поля	Тип даних	Опис
chatUID	varchar	Ідентифікаційний номер чату
senderType	varchar	Тип відправника повідомлення (кандидат або власник гурту)
message	varchar	Текст повідомлення
datetime	date	Дата та час відправки повідомлення

Таблиця 3.9 – Опис сутності notifications

Назва поля	Тип даних	Опис
notificationUID	serial PK	Унікальний ідентифікаційний номер сповіщення
receiverUID	varchar	Ідентифікаційний номер отримувача сповіщення
text	varchar	Текст сповіщення
datetime	date	Дата та час відправки сповіщення

Таблиця 3.10 – Опис сутності vacancies

Назва поля	Тип даних	Опис
vacancyUID	serial PK	Унікальний ідентифікаційний номер вакансії
bandUID	varchar	Ідентифікаційний номер гурту, що виставив вакансію
role	varchar	Роль в музичному гурті на яку розшукують кандидата
text	varchar	Додаткова інформація про вакансію
datetime	date	Дата та час створення вакансії

Таблиця 3.11 – Опис сутності resumes

Назва поля	Тип даних	Опис
resumeUID	serial PK	Унікальний ідентифікаційний номер резюме
candidateUID	varchar	Ідентифікаційний номер кандидата, який надіслав резюме
vacancyUID	varchar	Ідентифікаційний номер вакансії, на яку було надіслано резюме
text	varchar	Додаткова інформація до резюме
datetime	date	Дата та час надсилання резюме

Опис усіх програмних засобів, які були залученні під час створення програмного застосунку, наведено в таблиці 3.12.

Таблиця 3.12 – Опис утиліт

№ п/п	Назва утиліти	Опис застосування
1	Android Studio	Інтегроване середовище розробки мобільного застосунку. Дозволяє розробляти, проектувати, тестувати та емулювати програмне забезпечення.
2	Firebase console	Програмне забезпечення необхідне для налаштування проекту Firebase. Саме у ньому виконуються взаємодії із сервісами Firebase Authentication (авторизація), Firebase Realtime Database (бази даних), Firebase Storage (місце зберігання файлів)
3	diagrams.net (draw.io)	Програмне забезпечення яке дозволяє створювати різного роду UML-діаграми і не тільки.

Тексти програмного коду наведені в окремому документі «Текст програми».

3.4 Аналіз безпеки даних

Безпека даних та її забезпечення відіграють значну роль у розробці та використанні будь-якого програмного забезпечення, в тому числі і мобільного застосунку для пошуку музичного гурту, оскільки користувачі будуть зберігати та обмінювати особисту інформацію, важливо гарантувати, що їхні дані будуть захищені від несанкціонованого доступу, втрати або компрометації. Для запобіганню проблем – виділимо та проведемо дослідження основних вразливостей розробленого програмного забезпечення. Результати дослідження наведено у таблицях 3.13 – 3.16.

Таблиця 3.13 – Вразливість аутентифікації та авторизації

Вразливість	Аутентифікація та авторизація
Загроза	Несанкціонований доступ до облікових записів користувачів
Виконане рішення	Під час розробки використано надійний метод аутентифікації Firebase Authentication
Варіанти покращення	Використання методів двофакторної аутентифікації, постійне оновлення паролю

Таблиця 3.14 – Вразливість безпеки зберігання даних

Вразливість	Безпечне зберігання даних
Загроза	Крадіжка особистих даних користувачів
Виконане рішення	Було використано сервіс Firebase Realtime Database, у якому налаштовано параметри доступу до даних, що не дозволяє кому-небудь отримати доступ до бази даних
Варіанти покращення	Зберігати дані у зашифрованому вигляді, кожного разу їх розшифровуючи при отриманні

Таблиця 3.15 – Вразливість передачі даних

Вразливість	Передача даних
Загроза	Крадіжка особистих даних користувачів при спробі передати дані
Виконане рішення	Firebase Realtime Database використовує прокол HTTPS для забезпечення захищеного зв’язку між застосунком та базою даних.
Варіанти покращення	Запровадити шифрування при передачу даних.

Таблиця 3.16 – Вразливість архітектури програмного застосунку

Вразливість	Вразлива архітектура
Загроза	Крадіжка даних напряму з мобільного застосунку
Виконане рішення	Було дотримано принципів інкапсуляції, що приховало важливі дані від третіх осіб
Варіанти покращення	Зберігати усі дані у захищеному хмарному сервісі без завантаження локально.

Як можна побачити, усі можливі вразливості були тим чи іншим чином виправлені. Варто зазначити, що є кращі методи для захисту застосунку, але на даний момент, застосунок достатньо захищений.

Висновки до розділу

У результаті виконання даного розділу дипломного проєкту проведено конструювання та розроблено програмне забезпечення. Було детально розглянуто архітектурний патерн MVVM та виділено основні компоненти мобільного застосунку для пошуку музичного гурту як представлення, модель та модель представлення. Як результат, було створено діаграму пакетів, яка показує взаємовідношення між різними компонентами архітектури.

У якості архітектурного рішення, було вирішено використовувати безсерверну архітектуру (serverless), що дозволило значно покращити та захистити застосунок за допомогою сервісів Firebase. Також надано опис сервісів Firebase, які були використані у розробці програмного забезпечення та виділено технологічний стек для розробки мобільного застосунку.

Для використання найкращих засобів розробки, було проведено дослідження та виділено найкраще IDE – Android Studio, а в якості мови програмування, було обрано Java.

Під час розробки програмного забезпечення було створено багато класів, основні з яких були описані у таблиці, де вказано їх основний функціонал. Багато з класів (DAO) забезпечувало роботу з базою даних, яка

була реалізована за допомогою сервісу Firebase Realtime Database. На основі змодельованої бази даних було створено ER-діаграму сутностей, а також, для кожної сутності було створено таблицю із детальним описом полів та їх типів. У результаті було виділено список утиліт, які використовувались під час розробки програмного забезпечення.

Для забезпечення найкращого досвіду використання користувачам, було проведено аналіз вразливостей мобільного застосунку та безпеки даних користувач. Аналіз показав, що усі вразливості були покриті відповідним їм рішенням проблеми.

4 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз якості ПЗ

Так як перед розробкою поставлено певні нефункціональні вимоги, було вирішено провести аналіз якості ПЗ. Метриками для оцінки якості обрано:

– захищеність;
– надійсність;
– повторюваність;
– читабельність.

Для полегшення процесу аналізу якості коду ПЗ, було вирішено використати сервіс sonarcloud.io. SonarCloud – це хмарний сервіс перевірки чистого коду (його якість та безпека), який є безкоштовним для проєктів з відкритим кодом і пропонує безкоштовну пробну версію для приватних проєктів. Цей сервіс підтримує 26 мов програмування різного роду, серед яких є Java, що чудово підходить для розробленого мобільного застосунку [19].

Щоб розпочати аналіз, потрібно увійти на сайт. На сторінці входу обираємо метод за допомогою платформи github, на якій власне і знаходиться увесь код проекту (рис. 4.1).

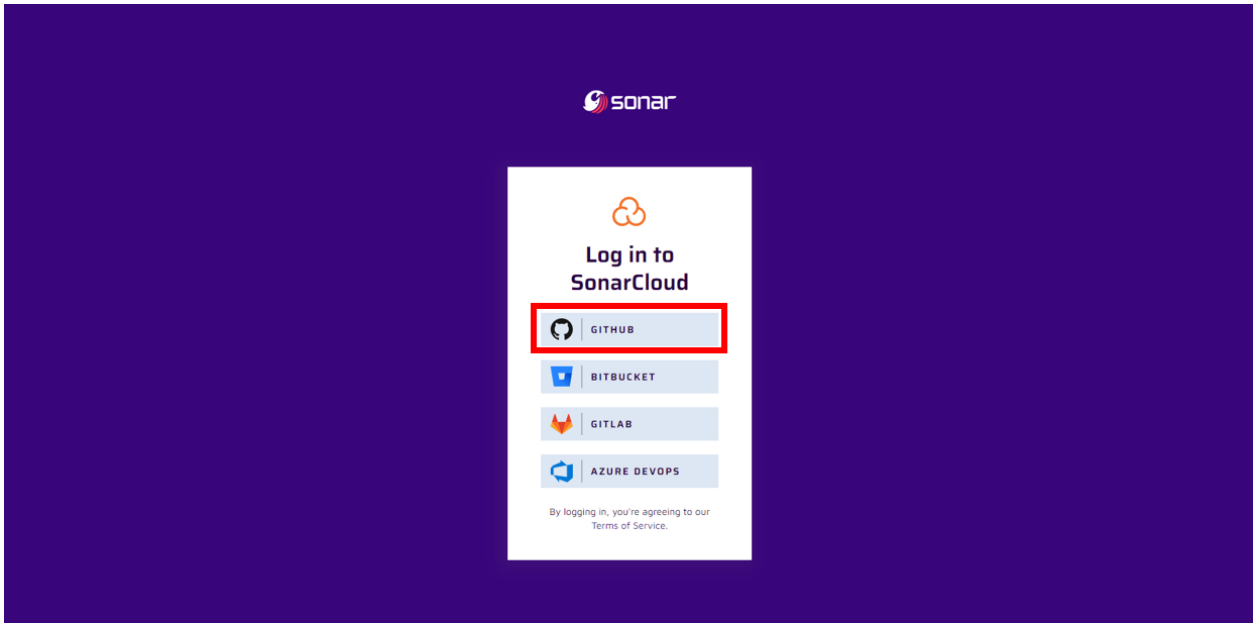


Рисунок 4.1 – Вхід до сервісу SonarCloud

Завдяки тому, що вхід було виконано через платформу github, сервіс дозволив швидко додати проєкт розробленого застосунку до аналізу. SonarCloud перевіряє такі властивості ПЗ як захищеність, надійність, читабельність, повторюваність і т.д. Отриманий результат для мобільного застосунку з пошуку музичного гурту зображено на рисунку 4.2.

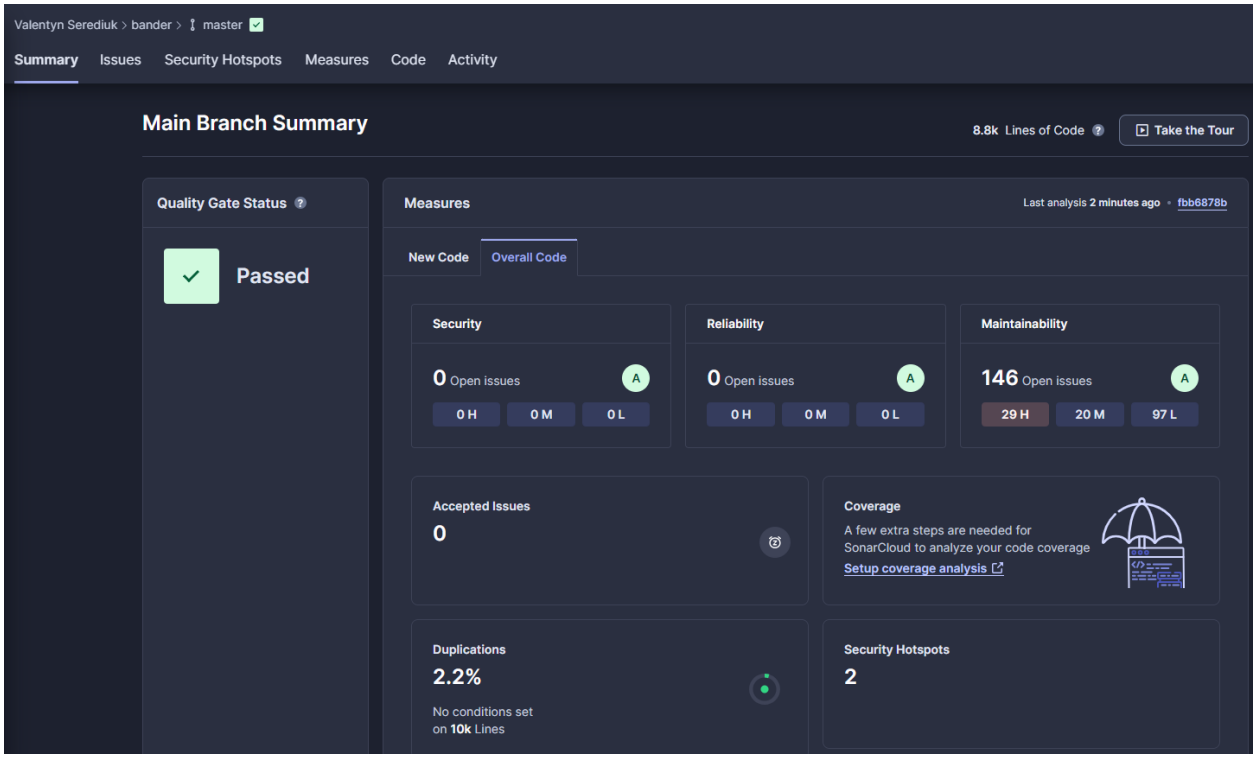


Рисунок 4.2 – Результат аналізу сервісом SonarCloud

Із результату, можна побачити, що у проєкта відсутні проблеми із захищеністю, надійністю та повторюваністю коду (2,2% це низький показник). Натомість, у проєкта є проблеми із читабельністю (146), але усі вони мають дуже низький ризик. Більшість цих проблем пов’язані із використанням патерна проектування класів Singleton, а також великою кількістю полів для виводу інформації, які часто повторюються.

4.2 Опис процесів тестування

Після завершення виконання програмного застосунку, було виконане мануальне тестування програмного забезпечення, згідно документу «Програма та методика тестування». Опис основної частини тестів наведено у таблицях 4.1 – 4.15.

Таблиця 4.1 – Тест 1.1 Реєстрація кандидата

Початковий стан системи	Кандидат знаходиться на сторінці реєстрації
Вхідні дані	Електронна пошта, ім’я, прізвище, дата народження, місто, пароль, підтвердження паролю
Опис проведення тесту	У відповідні поля екрану реєстрації кандидата вводяться: коректна електронна пошта, яка до цього не була зареєстрована в системі, особиста інформація користувача така як ім’я, прізвище, дата народження та місто, пароль від 8 символів, який містить хоча б з одну англійську літеру та одну арабську цифру, підтвердження паролю, яке повинне бути ідентичним раніше введеному паролю. Після цього натискається кнопка реєстрації.
Очікуваний результат	Реєстрація проходить успішно, користувач додається у базу даних авторизованих користувачів і перенаправляється на головну сторінку застосунку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.2 – Тест 1.2 Вхід користувача

Початковий стан системи	Користувач знаходиться на сторінці входу
Вхідні дані	Електронна пошта, пароль
Опис проведення тесту	У відповідні поля користувач вносить попередньо зареєстровані електронну пошту та відповідний їй пароль. Після цього натискається кнопка входу.
Очікуваний результат	Вхід проходить успішно, користувач перенаправляється на головну сторінку застосунку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.3 – Тест 1.3 Неправильний вхід користувача

Початковий стан системи	Користувач знаходиться на сторінці входу
Вхідні дані	Не відповідні електронна пошта та пароль
Опис проведення тесту	У відповідні поля користувач вносить не відповідні електронну пошту та пароль. Після цього натискається кнопка входу.
Очікуваний результат	Вхід не виконується. Користувач повідомляється про помилку вхідних даних.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.4 – Тест 1.4 Редагування профілю

Початковий стан системи	Користувач знаходиться на сторінці профілю
Вхідні дані	Дані, які користувач бажає змінити про себе (окрім електронної пошти та паролю)
Опис проведення тесту	Натискається кнопка редагування профілю. У відповідні поля, користувач вводить дані, які він хоче замінити. Натискається кнопка зберегти.
Очікуваний результат	Дані оновлюються в базі даних. Користувач переноситься на головну сторінку, де дані вже оновлені.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.5 – Тест 1.5 Зміна зображення профілю

Початковий стан системи	Користувач знаходиться на сторінці профілю
Вхідні дані	Нове зображення профілю
Опис проведення тесту	Натискається кнопка редагування профілю. Користувач натискає на фото. У новому вікні він обирає необхідне фото та підтверджує вибір. Натискається кнопка зберегти.
Очікуваний результат	Зображення оновлюється в базі даних. Користувач переноситься на головну сторінку із оновленим зображенням.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.6 – Тест 2.1 Створення вакансії

Початковий стан системи	Власник гурту знаходиться на сторінці пошуку
Вхідні дані	Роль, опис та заробітна плата для вакансії
Опис проведення тесту	Натискається кнопка створення вакансії. У відповідні поля, користувач вносить дані вакансії, також обирає валюту, у котрій буде виставлена заробітна плата. Натискається кнопка створити вакансію.
Очікуваний результат	Створюється нова вакансія у базі даних. Користувач переноситься на сторінку пошуку.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.7 – Тест 2.2 Перегляд усіх вакансій гурту

Початковий стан системи	Власник гурту знаходиться на сторінці пошуку
Вхідні дані	Відсутні
Опис проведення тесту	Натискається кнопка «мої вакансії».
Очікуваний результат	Користувач переноситься на сторінку із усіма створеними ним вакансіями.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.8 – Тест 2.3 Видалення вакансії гурту

Початковий стан системи	Власник гурту знаходиться на сторінці із списком його вакансій
Вхідні дані	Відсутні
Опис проведення тесту	Із переліку усіх вакансій, власник гурту натискає на ту, яку хоче видалити. Користувача переносить на сторінку із інформацією про вакансію. Натискається кнопка видалити вакансію та з’являється діалогове вікно підтвердження. Натискається кнопка «так».
Очікуваний результат	Вакансія видаляється із бази даних. Усі резюме, які були надіслані на дану вакансію отримали статус відхилені. Користувач переноситься на сторінку із списком вакансій гурту.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.9 – Тест 3.1 Відправлення резюме

Початковий стан системи	Кандидат знаходиться на сторінці пошуку
Вхідні дані	Додаткова інформація для резюме
Опис проведення тесту	Із переліку усіх рекомендованих вакансій, кандидат натискає на ту, на яку хоче відправити резюме. Користувача переносить на сторінку із інформацією про вакансію. Натискається кнопка надіслати резюме. Користувач переноситься на сторінку надсилання резюме. У поле вводиться додаткова інформація. Натискається кнопка надсилання.
Очікуваний результат	Резюме добавляється до бази даних. Користувач переноситься на головну сторінку та з’являється повідомлення про надсилання резюме.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.10 – Тест 3.2 Перегляд статусів резюме

Початковий стан системи	Кандидат знаходиться на сторінці пошуку
Вхідні дані	Відсутні
Опис проведення тесту	Натискається кнопка історії резюме.
Очікуваний результат	Користувач переноситься на сторінку з історією резюме, які отримали відповідь позитивну або негативну.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.11 – Тест 3.3 Підтвердження резюме

Початковий стан системи	Власник гурту знаходиться на сторінці із інформацією про резюме
Вхідні дані	Відсутні
Опис проведення тесту	Натискається кнопка прийняття резюме.
Очікуваний результат	Користувач переноситься на сторінку пошуку. З’являється повідомлення про прийняття резюме. Створюється новий чат із кандидатом. Кандидату надсилається повідомлення та сповіщення про прийняття резюме. Дані в базі даних оновлюються.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.12 – Тест 3.4 Відхилення резюме

Початковий стан системи	Власник гурту знаходиться на сторінці із інформацією про резюме
Вхідні дані	Відсутні
Опис проведення тесту	Натискається кнопка відхилення резюме.
Очікуваний результат	Користувач переноситься на сторінку пошуку. З’являється повідомлення про відхилення резюме. Дані в базі даних оновлюються.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.13 – Тест 4.1 Перегляд історії чату

Початковий стан системи	Користувач знаходиться на сторінці чатів
Вхідні дані	Відсутні
Опис проведення тесту	Користувач натискає на відповідний чат із списку, який йому потрібний.
Очікуваний результат	Користувач переноситься на сторінку, де відображається історія вибраного ним чату.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.14 – Тест 4.2 Надсилання повідомлення в чат

Початковий стан системи	Користувач знаходиться в чаті
Вхідні дані	Повідомлення
Опис проведення тесту	Користувач натискає на поле для вводу повідомлення. Вводиться необхідне повідомлення. Натискається кнопка надсилання.
Очікуваний результат	Історія повідомлень поповнюється ще одним повідомленням. Користувачі чату одразу ж мають змогу переглянути зміни.
Фактичний результат	Збігається з очікуваним.

Таблиця 4.15 – Тест 5.1 Відображення кількості на головній сторінці

Початковий стан системи	Користувач знаходиться на головній сторінці
Вхідні дані	Відсутні
Опис проведення тесту	Користувач отримує нові сповіщення або повідмлення
Очікуваний результат	Кількість повідомлень та сповіщень відображаються у відповідних місцях поблизу плиток чатів та сповіщень.
Фактичний результат	Збігається з очікуваним.

4.3 Опис контрольного прикладу

Розглянемо весь процес роботи із вакансією зі сторони власника гурту, який не працював ще з ними. Початковий стан для роботи з вакансіями, це перебування власника гурту на сторінці пошуку (рис. 4.3).

Рисунок 4.3 – Порожня сторінка пошуку

Дана сторінка не відображає резюме, оскільки ніхто їм ще нічого не відіслав. Також, вона має два варіанта використання щодо роботи із вакансіями – переглянути свої вакансії або ж створити вакансію. При переході на список своїх вакансій, буде відображатись сторінка із списком усіх вакансій гурту, яких на даний момент ще нема. При натисненні на кнопку створити вакансію буде відкрито вікно створення вакансій(рис. 4.4).

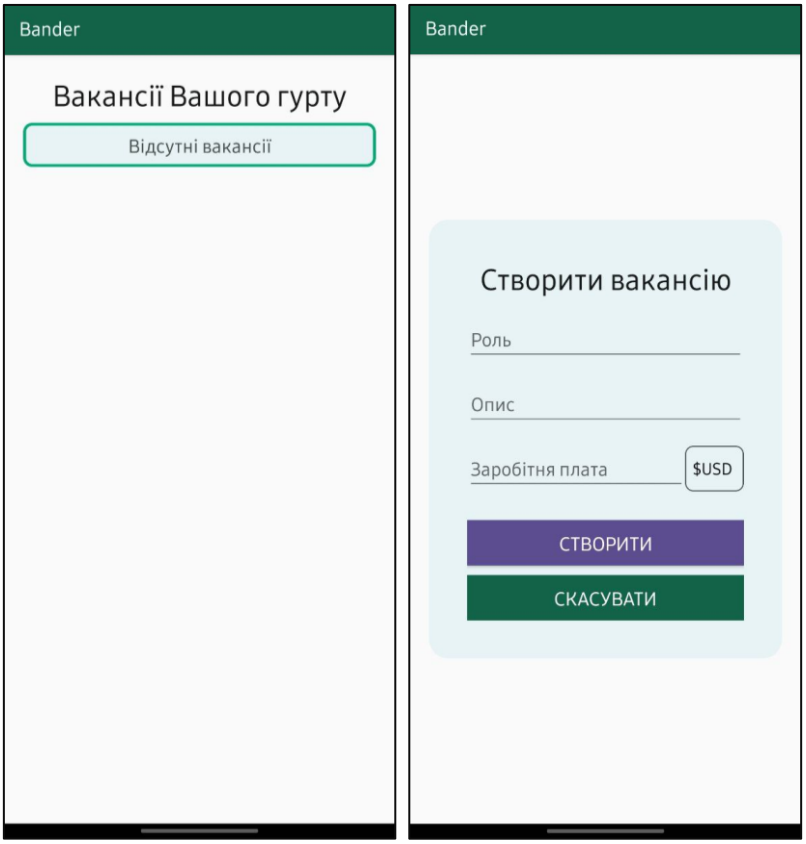
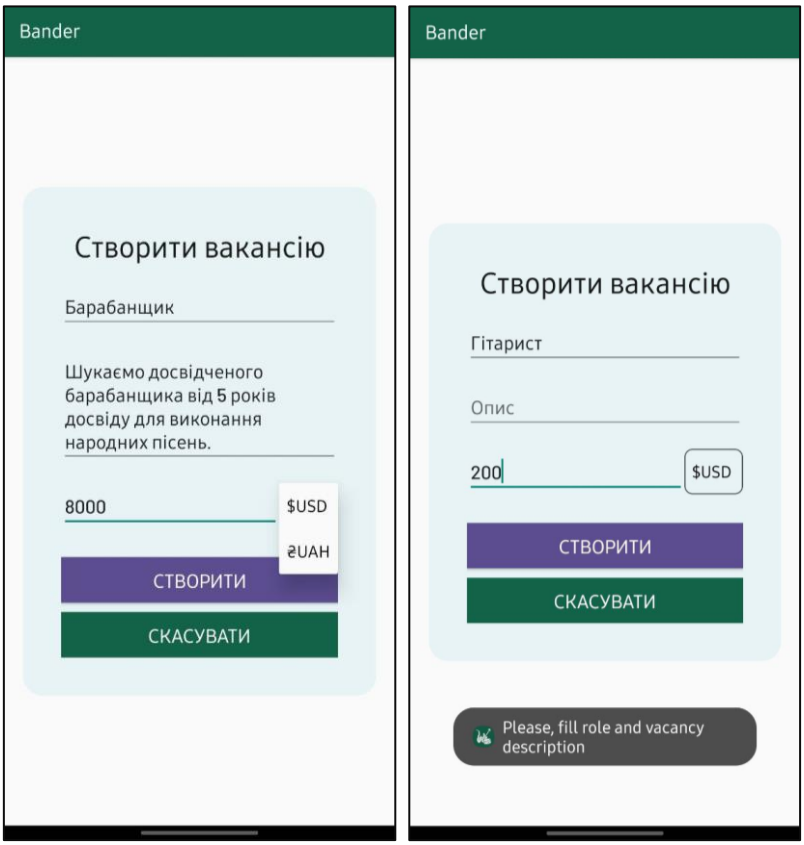


Рисунок 4.4 – Екрани списку вакансій гурту та створення вакансій

Розглянемо окремо сторінку створення вакансій. Користувач може ввести значення у три поля, а також вибрати валюту із випадаючого списку. Значення у полях роль та опис є обов’язковими, тому без введення даних, застосунок повідомиться про це. Значення у полі заробітна плата є опціональним. Без введення в нього, вакансія буде вважатись із не розголошеною заробітною платою (рис. 4.5).



Рисоунок 4.5 – Екрани із повністю заповненими полями та виборем валюти, а також приклад повністю не заповненої форми створення із повідомленням

У випадку, якщо ж власник гурту вніс дані у поля роль та опис, то відбудеться створення вакансії, про що власне і повідомиться користувач. Варто зазначити, що кількість вакансій на один гурт є необмеженою. Також, якщо повернутись до екрану із списком вакансій гурту, то там з’явиться створена вакансія (рис. 4.6).

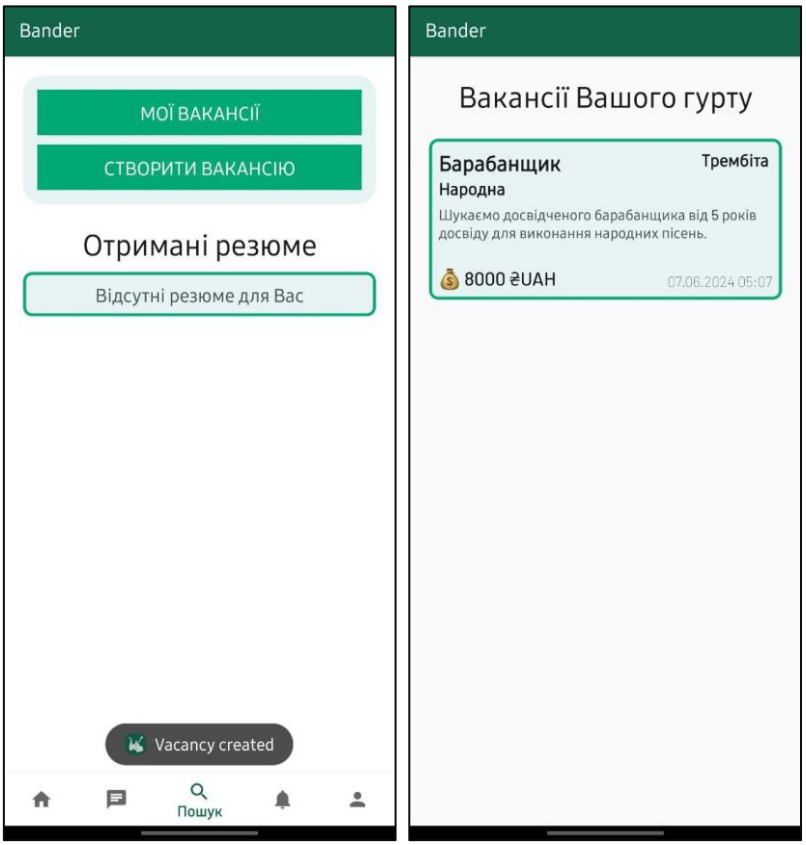


Рисунок 4.6 – Екрани із повідомлення про створення вакансії і відображення її у списку усіх вакансій

Користувач має можливість переглянути створені ним вакансії. При переході в список вакансії, а також натисненні на конкретну вакансію, користувача перемістить на сторінку цієї вакансії, на якій повністю відображається уся інформація про вакансію. Власник гурту має право або повернутись назад, або ж видалити вакансію. При натисканні кнопки назад, користувач повернеться до списку своїх вакансій. Якщо ж натиснути видалити вакансію, то з’явиться попереджувальне діалогове вікно, яке перепитає підтвердження видалення вакансії. У разі підтвердження видалення, вакансія видаляється та більше не відображається у списку (рис. 4.7).

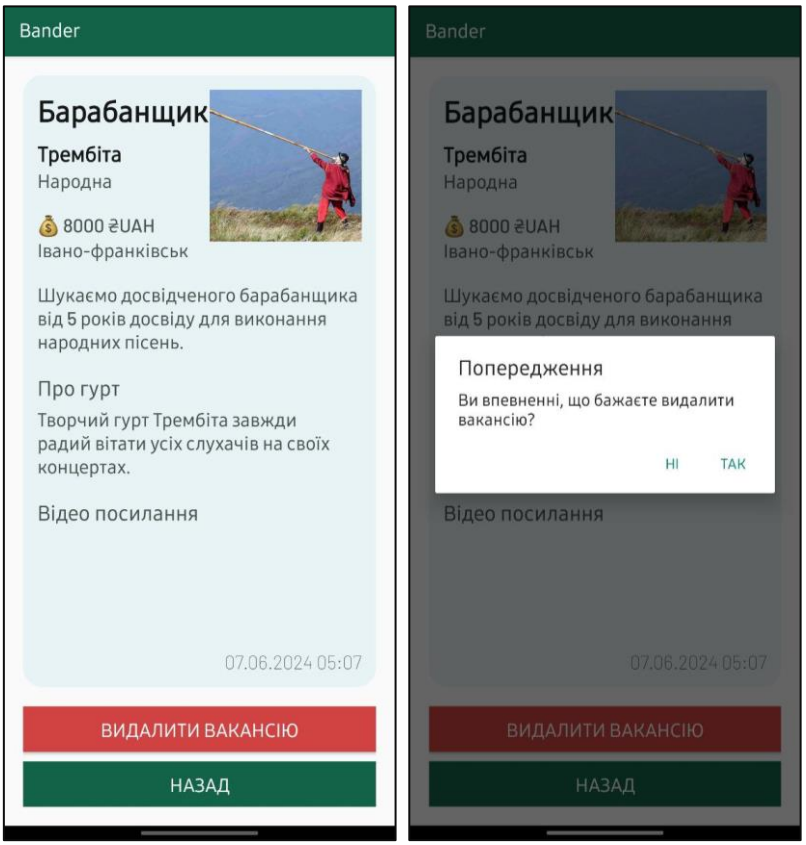


Рисунок 4.7 – Екрани перегляду інформації про вакансію, а також підтвердження видалення вакансії

Після видалення, можна спостерігати порожній екран, який зображено на рисунку 4.4.

Висновки до розділу

У цьому розділу дипломного проєкту, було здійснено оцінку якості програмного забезпечення, який виконувався сервісом автоматичного аналізу коду. Результат показав, що написаний код являється гарно захищеним, надійним, а також мав мало повторень. Єдиною проблемою написаного коду виявилась погана читабельність, яка зумовлена правилами перевірки даного сервісу, який скаржиться на використання деяких архітектурних патернів.

Також, було проведене мануальне тестування основних функціональних вимог програмного забезпечення. Тестування показало, що усі вимоги реалізовані коректно та працюють без жодних відхилень від очікуваних результатів. Варто зазначити, що було проведено тест на введення

неправильний вхідних даних, який показав, що розроблене програмне забезпечення передбачило помилку та не впустило користувача.

Як результат даного розділу дипломного проєкту, було наведено повний детальний опис одного контрольного прикладу. У даному прикладі було проведено аналіз усіх можливостей по роботі з вакансіями для власників гурту. Було наведено скріншоти усіх екранних форм та приведені варіанти введення різних даних при створенні вакансій, які повних так і не повних. Цей приклад показав, що система по роботі із вакансіями працює повністю правильно.

5 РОЗГОРТАННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Розгортання програмного забезпечення

Для розгортання Android застосунків є 3 основних варіанта: розгортання на локальному комп'ютері через емулятори, виготовлення .apk файлу та розсилання його, надання доступів і т.д., або ж найпопулярніший і найзручніший варіант це завантаження його на ринок мобільних застосунків для зручного поширення в маси. Розглянемо кожен із варіантів по порядку.

Якщо ж запускати даний проєкт локально, то для цього підійде Android Studio із вбудованим емулятором. Для запуску потрібно завантажити проєкт із github-репозиторію; відкрити його за допомогою Android Studio; увімкнути емулятор Android-пристрою та власне запустити проект. Останні дві дії зображені на рисунку 5.1.

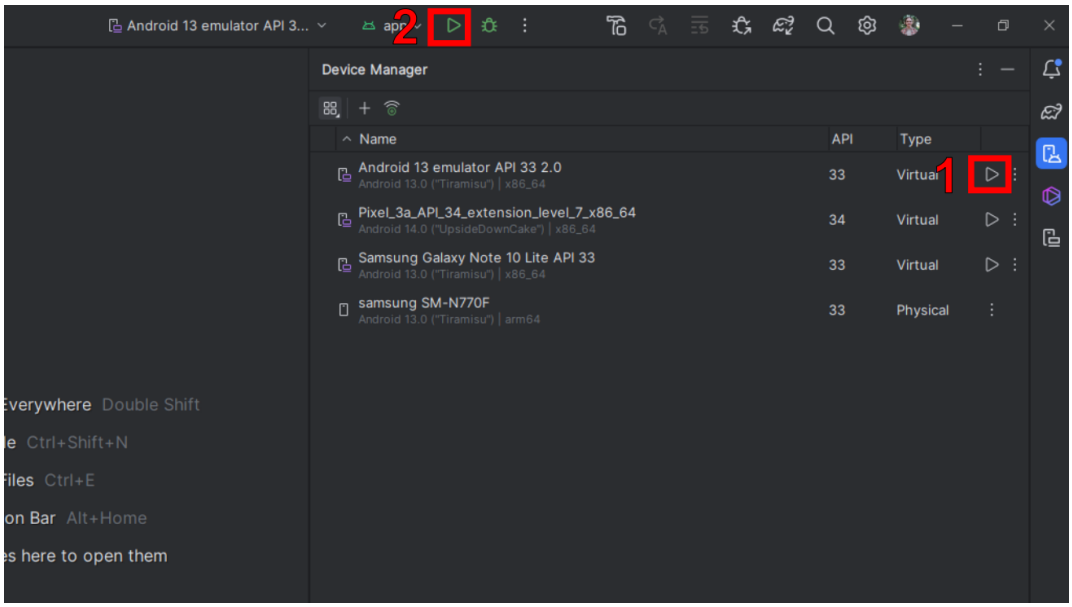


Рисунок 5.1 – Порядок запуску емулятора та застосунку

Даний метод є зручним для внесення швидких змін у застосунок, оскільки можна моментально запустити нову версію застосунку. Але для звичайного користувача це не має сенсу, йому потрібна зручність в інсталяції застосунку.

Другий варіант це розповсюдження .apk файлу. Даний файл, можна встановити на будь-якому фізичному пристрої чи емуляторі із версією Android 9 і вище. Для генерації даного файлу потрібно в Android Studio

слідувати наступним діям: Menu → Build → Build Bundle(s) / APK(s) → Build APK(s) (рис. 5.2).

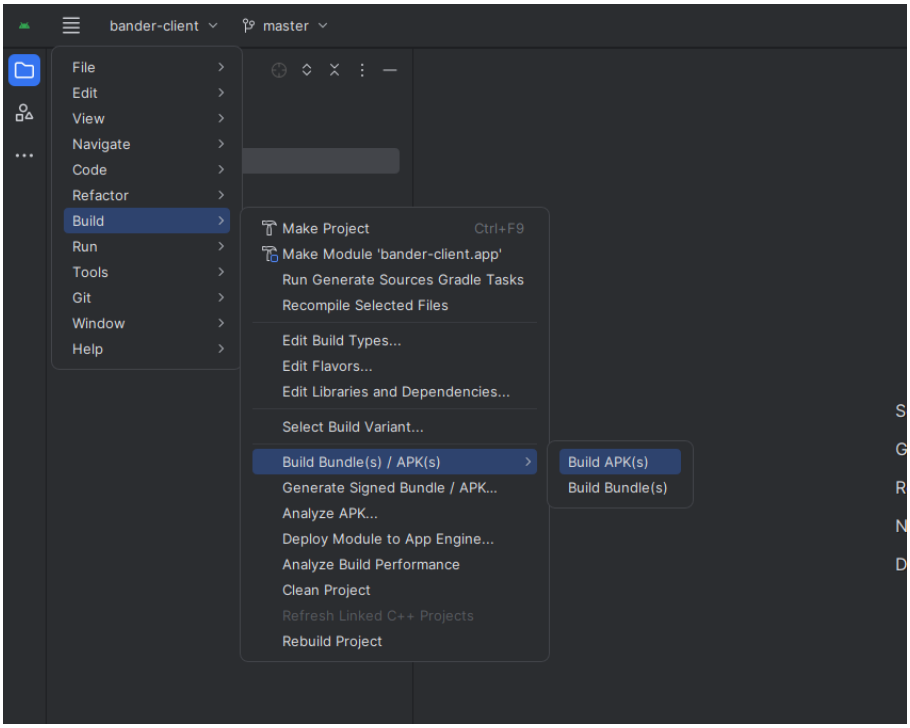


Рисунок 5.2 – Порядок дій для генерації .apk файлу в Android Studio

У результаті, отримаємо побудований .apk файл, який бути міститись у папці, яка буде вказана у повідомлені після завершення виготовлення файлу (рис. 5.3).

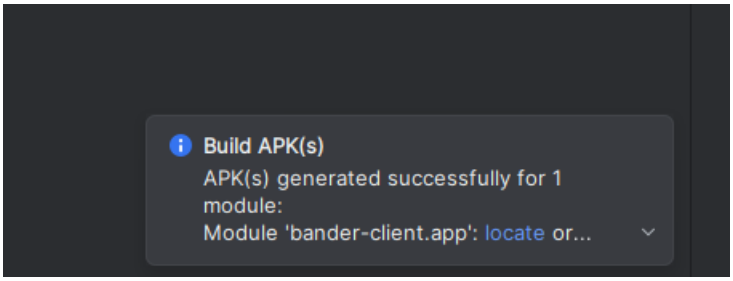


Рисунок 5.3 – Повідомлення із папкою .apk файлу після завершення побудови

На далі, отримуємо файл, який можна без перешкод надсилати кому завгодно, але таке розповсюдження не є зручним для користувачів також.

Переходимо до найпопулярнішого способу – завантаження застосунку у маркет. Серед найпопулярніших маркетів під Android можна виділити Google Play Market, який є явним фаворитом у цьому плані. Але у нього є один недолік – для створення акаунту користувач повинен заплатити 25$

одноразової оплати, що не є підходящим для усіх розробників. Тому даний сервіс розглядатись не буде, хоча і є найпопулярнішим.

Серед інших популярних маркетів для публікування застосунків лідирують Samsung Galaxy Store, Amazon Appstore, Huawei AppGallery, Xiaomi GetApps та інші. Варто відмітити, що практично всі вони розроблені певним виробником під свої пристрої, а Amazon Appstore може бути встановленою на будь-який Android пристрій [20].

Розглянемо повний процес добавлення застосунку у Amazon Appstore. Для початку, потрібно авторизуватись на сайті developer.amazon.com. Після входу, наводимо на «Add a New App» та натискаємо на «Android» (рис. 5.4).

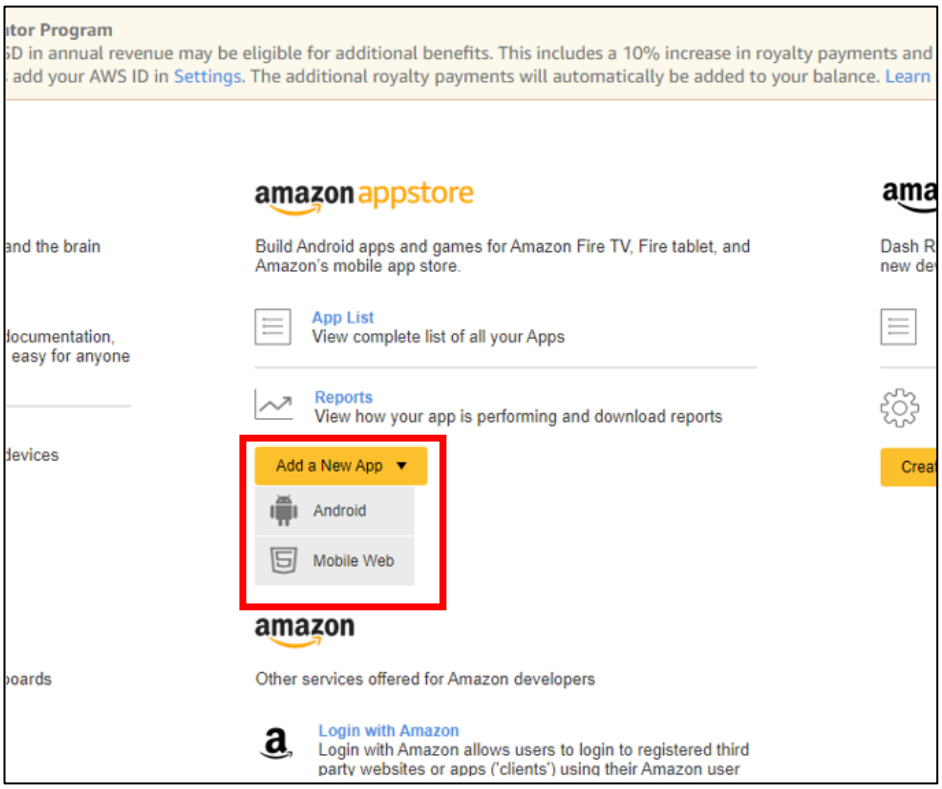


Рисунок 5.4 – Початок створення застосунку у Amazon Appstore

У новому вікні, вводимо усі необхідні дані про мобільний застосунок, а саме його назву, SKU (унікальне значення, яке буде використовуватись як id проекту), вказати категорію застосунку, а також нижче вказати електронну пошту для підтримки користувачів та натискаємо зберегти (рис. 5.5).

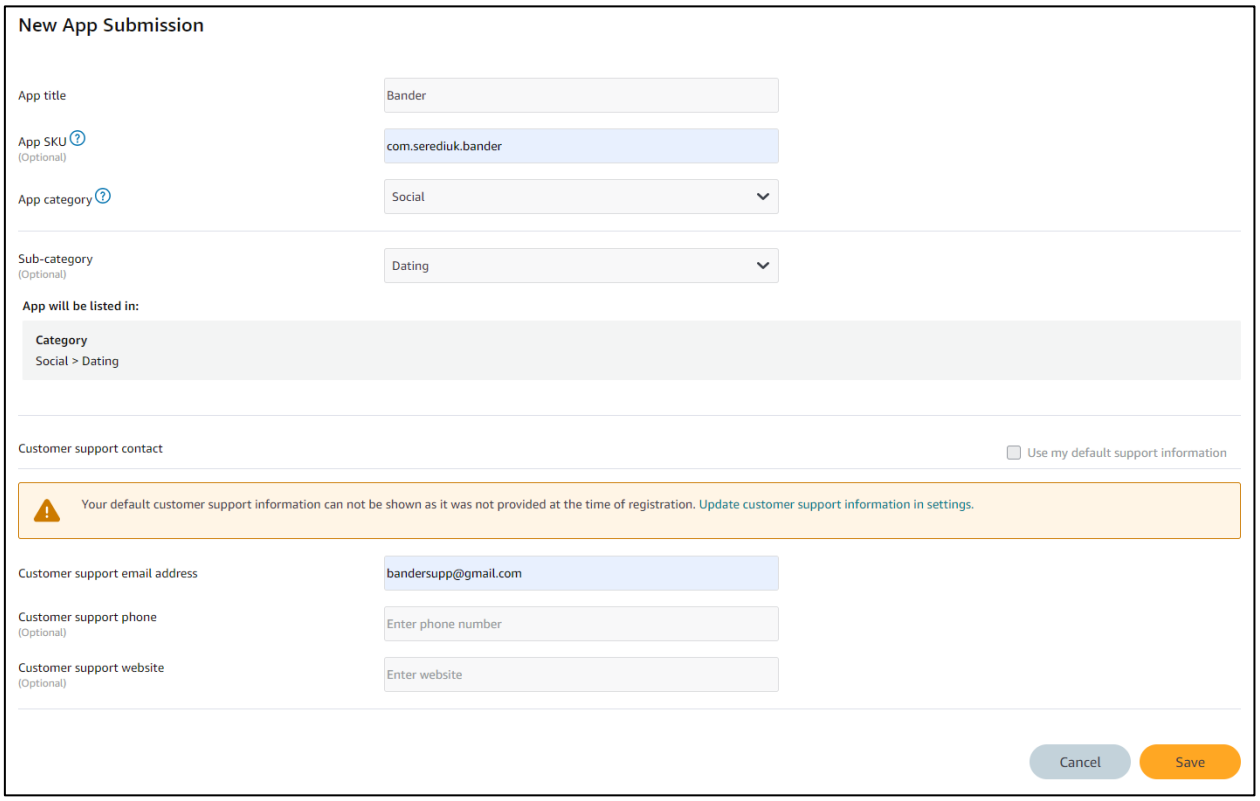


Рисунок 5.5 – Введення даних для створення застосунку

У відкритому вікні, завантажуємо .apk файл, який було створено за зразком попереднього варіанту (рис. 5.6).

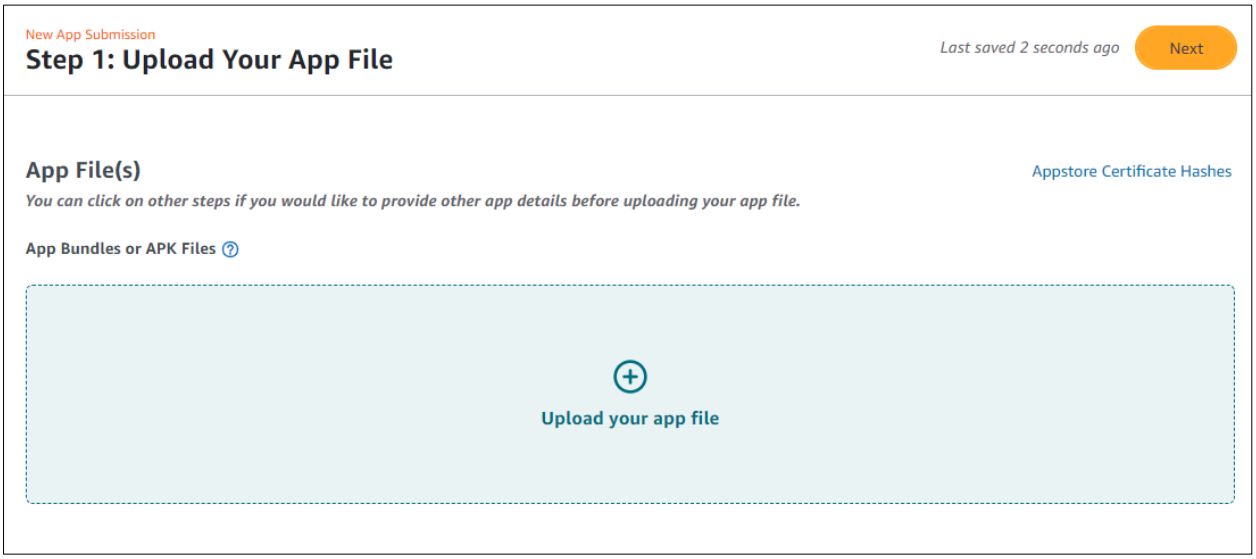


Рисунок 5.6 – Місце для завантаження .apk файлу

Очікуємо завершення процесу завантаження, перевіряємо дані, за необхідності виправляємо (рис. 5.7).

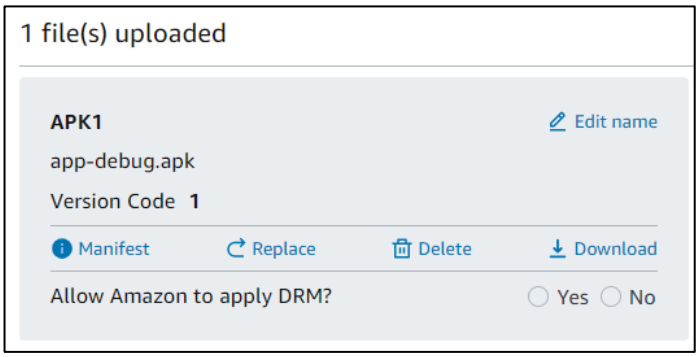


Рисунок 5.7 – Інформація про завантажений .apk файл

Переходимо до другого кроку та вказуємо необхідну інформацію про цільову аудиторію застосунку (рис. 5.8).

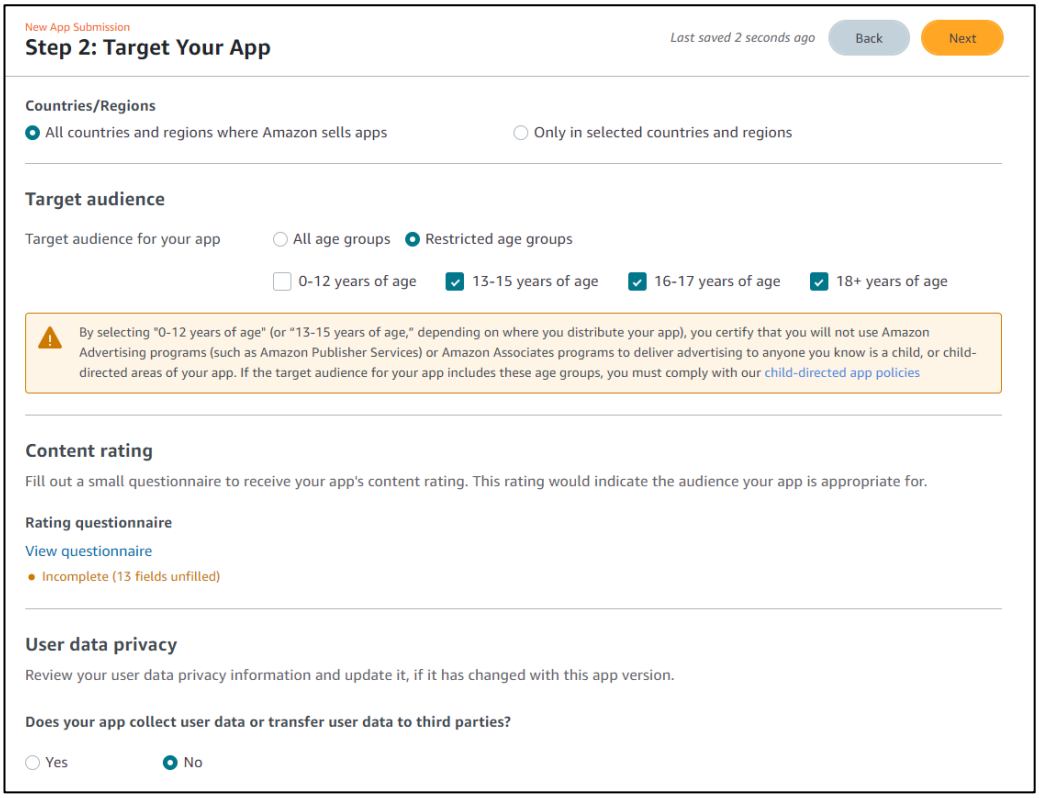


Рисунок 5.8 – Внесення даних про цільову аудиторію

Переходимо на наступний крок, на якому потрібно заповнити інформацію про додаток (рис. 5.9), додати іконки та скріншоти застосунку (рис. 5.10).

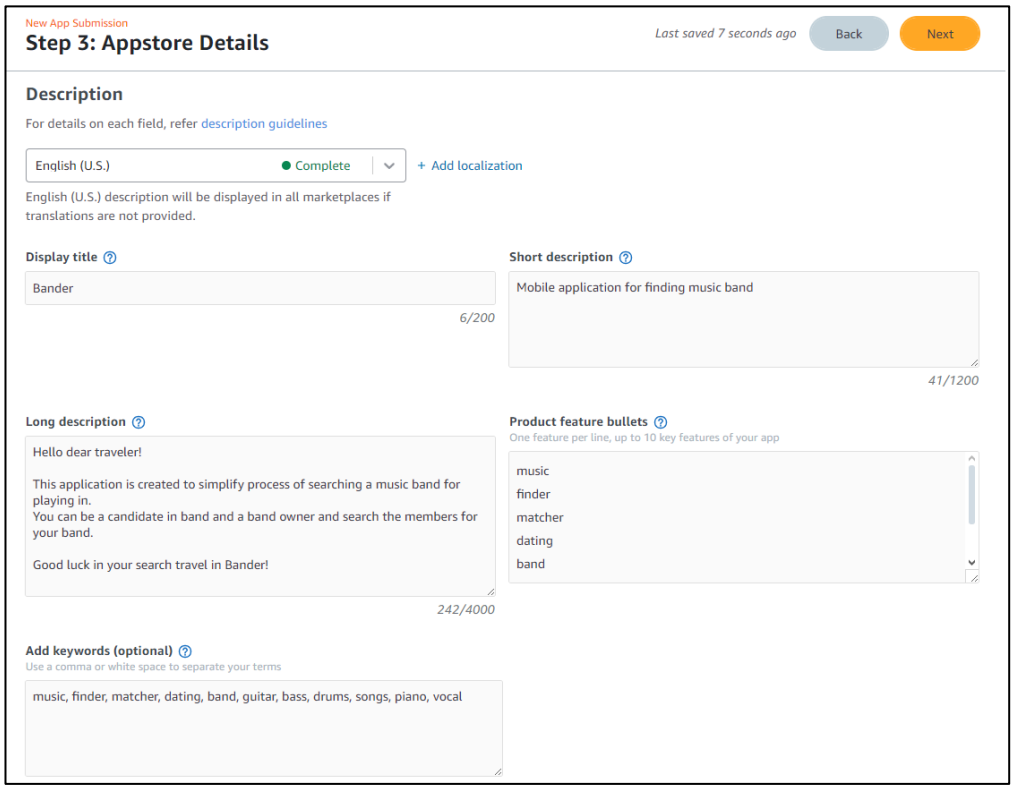


Рисунок 5.9 – Заповнення інформації про додаток

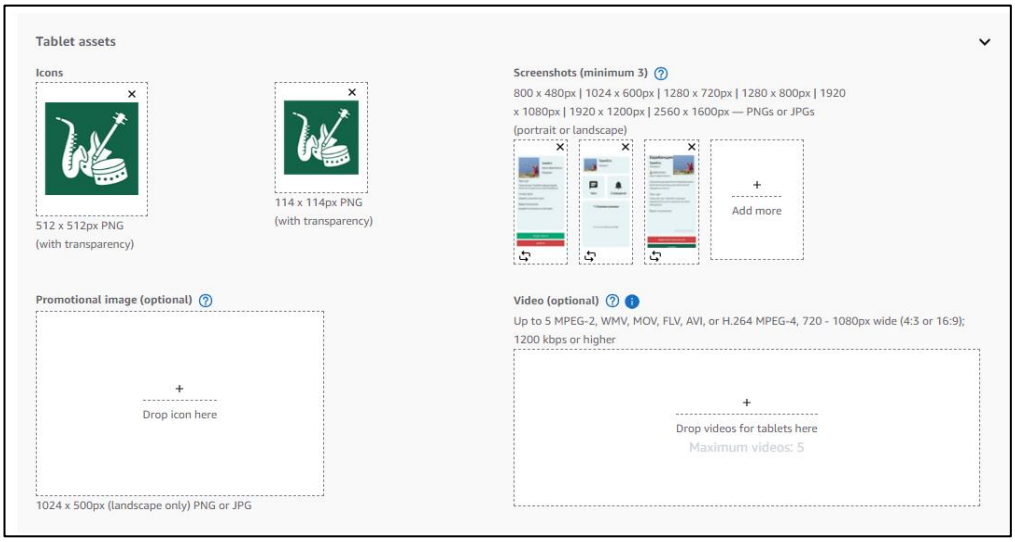


Рисунок 5.10 – Заповнення іконок та скріншотів програми

Перейшовши на 4 крок просто завершуємо заповнення (рис. 5.11).

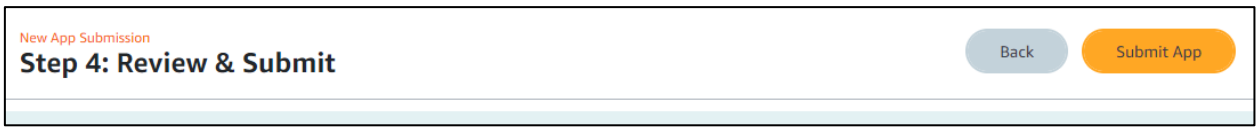


Рисунок 5.11 – Кнопка завершення створення застосунку

Після завершення, потрібно очікувати на підтвердження модерації, після чого застосунок стане публічно доступним. Таким чином, застосунок було вивантажено на платформу Amazon Appstore [21].

5.2 Супровід програмного забезпечення

В окремому документів наведено керівництво користувача.

Користувач повинен мати змогу оновити свій застосунок у випадку її створення. З цією метою, розробник повинен оновити версію .apk файлу, яка завантажена на Amazon Appstore.

Для оновлення версії застосунку, потрібну натиснути «Add Upcoming Version» (рис. 5.12).

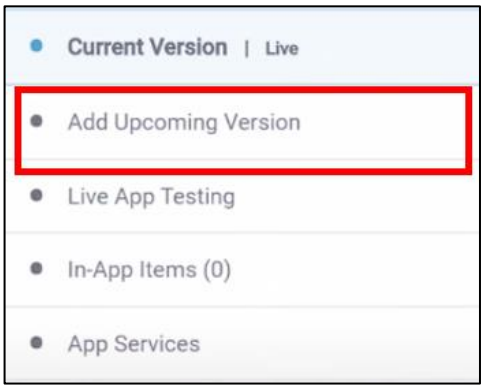


Рисунок 5.12 – Кнопка додавання нової версії програмного забезпечення

Далі, заходимо у пункт горизонтального меню «APK Files», внизу натискаємо «Edit» та отримуємо вікно, із вмістом як на рисунку 5.13.

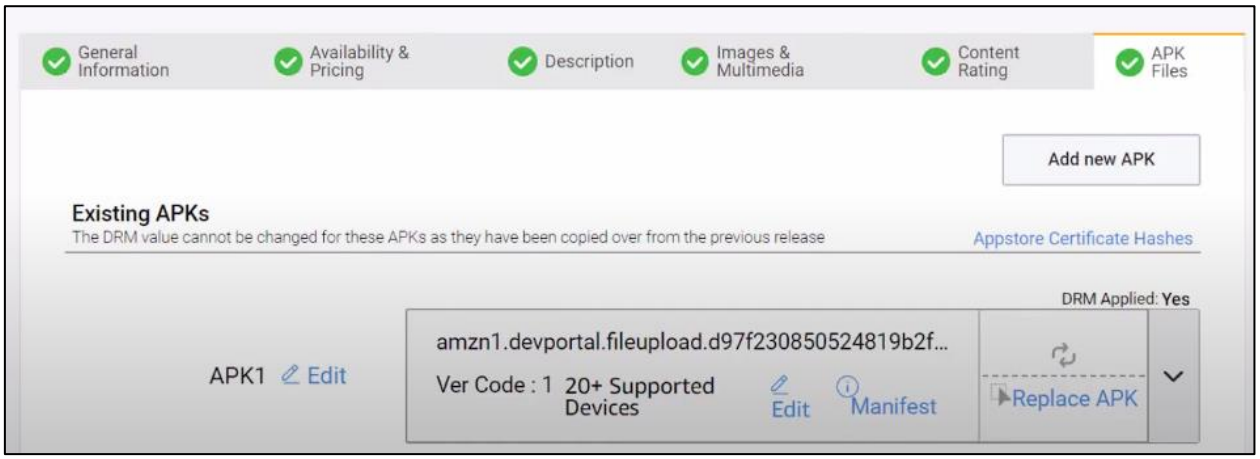


Рисунок 5.13 – Вікно із заміною .apk файлу

Для заміни ж самого фалу потрібно натиснути «Replace APK» та вибрати нову версію застосунку. Потім, для завершення натиснути «Submit App» як і у випадку з першим завантаженням. Після очікування на перевірку, застосунок буде оновлено у Amazon Appstore, де будь-який користувач вже зможе оновитись до найновішої версії просто натиснувши «оновити».

Висновки до розділу

У результаті виконання останнього розділу дипломного проєкту, було описано три варіанти для розгортання програмного забезпечення, а саме локальний запуск коду, створення та розповсюдження .apk файлу та завантаження .apk файлу на маркети застосунків. Локальний запуск виявився зручним для моментального внесення змін, але не зручним для користувача, який не володіє мовою програмування та середовищем розробки. Створення .apk файлу виявилось вже зручнішим, але була проблема у тому, як правильно розповсюджувати застосунок. Варіант із завантаженням на маркети мобільних застосунків вирішив усі недоліки попереднього варіант із .apk файлом. Як результат, було обрано Amazon Appstore як маркет для завантаження мобільного застосунку для пошуку музичного гурту, через його відносну універсальність та безкоштовність у розробці. Також було наведено детально усі кроки для створення .apk файлу програмного забезпечення, а також усі етапів публікування застосунку на маркет мобільних застосунків Amazon Appstore.

Для користувачів, які будуть потребувати оновлень, було наведено детальну інструкцію, яка показує як замінити попередню версію застосунку в Amazon Appstore на новішу, а також як звичайному користувачу отримати нову версію.

ВИСНОВКИ

У результаті виконання дипломного проєкту, було розроблено мобільний застосунок для пошуку музичного гурту, який дозволить полегшити процес інтеграції творчих людей у музичній сфері. Застосунку надано функціонал для роботи як з кандидатами в гурти, так і з власниками даних гуртів. Реалізовано системи для роботи з вакансіями та резюме. Вакансії можна створювати, переглядати, відзиватись на них та видаляти. Резюме, у свою чергу, можна надіслати у відповідь на вакансію та прийняти або відхилити. Для покращення контакту між кандидатами і власника гуртів, була додана можливість надсилати повідомлення один одному в режимі реального часу. Для кращого досвіду використання застосунку було додані сповіщення для інформування користувачів про важливі події стосовно їх профілів.

З метою досягнення поставленої мети, було здійснено аналіз предметної області, під час чого було виділено недоліки конкурентів, спроєктовані BPMN-моделі, функціональні та нефункціональні вимоги, на основі яких будувалась діаграма використань. Також, були виділені важливі архітектурні рішення та патерни, з яких було використано безсерверну архітектуру та патерн MVVM. У якості баз даних було обрано Firebase Realtime Database, що дозволило розробити застосунок, який працює з даними в режимі реального часу. У якості середовища розробки було обрано Android Studio через його ефективність у розробці Android додатків. На основі розробленого програмного забезпечення було сконструйовано пакетну діаграму.

Для підтвердження правильності роботи мобільного застосунку, було проведено аналіз якості коду, який показав хороший результат, а також ряд мануальних тестів, з якими додаток впорався на відмінно.

З метою розповсюдження програмного забезпечення, було виконано вивантаження додатку на маркет мобільних застосунків Amazon Appstore, що дозволить усім бажаючим спробувати мобільний застосунок для пошуку музичного гурту власноруч та переконатись у тому, що він дійсно полегшить життя усім музикантам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Mobile application. Techopedia. URL: https://www.techopedia.com/definition/2953/mobile-application-mobile-app.

2) Додаток для пошуку роботи - як об’єднати замовників та виконавців. KITAPP. URL: https://kitapp.pro/uk/dodatok-dlya-poshuku-roboti-yak-ob-yednati-zamovnikiv-ta-vikonavtsiv/.

3) DHA Communications. The Working Musician. The Musicians' Union | Trade Union in the UK. URL: https://musiciansunion.org.uk/MusiciansUnion/media/resource/Guides%20and%20reports/Education/The-Working-Musician-report.pdf?ext=.pdf.

4) Musicians, singers, & related workers | Data USA. Data USA. URL: https://datausa.io/profile/soc/musicians-singers-related-workers.

5) Mobile Operating System Market Share Worldwide | Statcounter Global Stats. StatCounter Global Stats. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide.

6) Android (operating system). Wikipedia. URL: https://en.wikipedia.org/wiki/Android_(operating_system).

7) Boudjnah E. Clean Architecture for Android: Implement Expert-led Design Patterns to Build Scalable, Maintainable, and Testable Android Apps (English Edition). BPB Publications, 2022. 262 p.

8) Android Architecture Patterns | Dashwave. Android Cloud Builds & Collaborative Emulators | Dashwave. URL: https://dashwave.io/blog/android-architecture-patterns.

9) Serverless Architectures. *Amazon Web Services, Inc*. URL: https://aws.amazon.com/lambda/serverless-architectures-learn-more/.

10) Firebase | Google's Mobile and Web App Development Platform. Firebase. URL: https://firebase.google.com/.

11) What Is a NoSQL Database? | IBM. *IBM in Deutschland, Österreich und der Schweiz*. URL: https://www.ibm.com/topics/nosql-databases.

12) Android Developers. URL: https://developer.android.com/tools/releases/platform-tools.

13) Android SDK. Wikipedia. URL: https://en.wikipedia.org/wiki/Android_SDK.

14) JetBrains. IntelliJ IDEA – the Leading Java and Kotlin IDE. *JetBrains*. URL: https://www.jetbrains.com/idea/.

15) ADT Plugin for Eclipse | Android Developers. Distributed Object Computing (DOC) Group for DRE Systems. URL: https://www.dre.vanderbilt.edu/~schmidt/android/android-4.0/out/target/common/docs/doc-comment-check/sdk/eclipse-adt.html.

16) Java documentation. Oracle. URL: https://docs.oracle.com/en/java/.

17) Kotlin Docs. *Kotlin Help*. URL: https://kotlinlang.org/docs/home.html.

18) Java vs. Kotlin. *Baeldung on Kotlin*. URL: https://www.baeldung.com/kotlin/java-vs-kotlin.

19) Driving continuous quality of your code with SonarCloud. Azure DevOps Hands-On Labs | Azure DevOps Hands-on-Labs. URL: https://azuredevopslabs.com/labs/vstsextend/sonarcloud/.

20) The Ultimate List of Mobile App Stores in 2024. MobiLoud - Convert Your Website to Native Mobile Apps. URL: https://www.mobiloud.com/blog/list-of-mobile-app-stores.

21) Submit Your App to the Appstore | Fire App Builder. Amazon Developers. URL: https://developer.amazon.com/docs/fire-app-builder/submit-your-app.html.

ДОДАТОК А ЗВІТ ПОДІБНОСТІ

UNICHECK
by Turnitin

Ім'я користувача:
Лісовиченко Олег Іванович

ID перевірки:
1016332248

Дата перевірки:
07.06.2024 23:49:57 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
08.06.2024 07:22:30 EEST

ID користувача:
76913

Назва документа: ІП-02_Середюк_ПЗ

Кількість сторінок: 80 Кількість слів: 11304 Кількість символів: 88224 Розмір файлу: 4.46 MB ID файлу: 1016132059

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

15%
Схожість

Найбільша схожість: 4.86% з джерелом з Бібліотеки (ID файлу: 1016099047)

5.01% Джерела з Інтернету 221 ..... Сторінка 82

14.5% Джерела з Бібліотеки 602 ..... Сторінка 85

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

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

Підозріле форматування 14 сторінок

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПОШУКУ МУЗИЧНОГО ГУРТУ

Текст програми

КПІ.ІП-0222.045440.03.12

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Валентин СЕРЕДЮК

Київ – 2024

Посилання на репозиторій з повним текстом програмного коду
https://github.com/serediukit/bander

Файл AuthProvider.java
Реалізація функціональної вимоги системи авторизації користувача.

```
public class AuthProvider {
    private static AuthProvider instance;

    private final FirebaseAuth auth;

    private AuthProvider() {
        auth = FirebaseAuth.getInstance();
    }

    public static AuthProvider getInstance() {
        if (instance == null) {
            instance = new AuthProvider();
        }
        return instance;
    }

    public void addAuthStateListener(FirebaseAuth.AuthStateListener listener)
{
        auth.addAuthStateListener(listener);
    }

    public void removeAuthStateListener(FirebaseAuth.AuthStateListener
listener) {
        auth.removeAuthStateListener(listener);
    }

    public Task<AuthResult> register(String email, String password, String
confirmPassword) {
        if (Validator.isValidEmail(email) &&
Validator.isValidPassword(password)) {
            if (password.equals(confirmPassword))
                return auth.createUserWithEmailAndPassword(email, password);
            else {
                return Tasks.forException(new AuthException("Passwords do not
match"));
            }
        }
        return Tasks.forException(new AuthException());
    }

    public Task<AuthResult> login(String email, String password) {
        return auth.signInWithEmailAndPassword(email, password);
    }

    public void signOut() {
        auth.signOut();
    }

    public String getUid() {
        return Objects.requireNonNull(auth.getCurrentUser()).getUid();
    }
}
```

Файл BandsDAO.java

Реалізація функціональної вимоги налаштування профілю користувача.

```
public class BandsDAO {
    private static BandsDAO instance;
    private final FirebaseDatabase database;

    private final ArrayList<Band> bandsList;

    private BandsDAO() {
        bandsList = new ArrayList<>();

        database = DatabaseConnectionProvider.getInstance().getDatabase();
        loadBands();
    }

    public static BandsDAO getInstance() {
        if (instance == null) {
            instance = new BandsDAO();
        }
        return instance;
    }

    public void createBand(Band band) {
        if (band.getBandUID() != null) {

database.getReference("bands").child(band.getBandUID()).setValue(band);

database.getReference("users").child(band.getBandUID()).setValue(band.getUser
());

            Log.d("BAND DAO", "Adding band:\n" + band);
        } else {
            Log.d("BAND DAO", "Key is null");
        }
    }

    public Band readBand(String bandUID) {
        Log.d("BAND DAO", "Reading band: " + bandUID);
        if (bandsList.size() != 0) {
            for (Band band : bandsList) {
                if (band.getBandUID().equals(bandUID)) {
                    return band;
                }
            }
        }
        return null;
    }

    public void updateBand(Band band) {

database.getReference("bands").child(band.getBandUID()).setValue(band);

database.getReference("users").child(band.getBandUID()).setValue(band.getUser
());

        for (int i = 0; i < bandsList.size(); i++) {
            if (bandsList.get(i).getBandUID().equals(band.getBandUID())) {
                bandsList.set(i, band);
                break;
            }
        }

        Log.d("BAND DAO", "Updating band:\n" + band);
```

```
    }

    public void loadBands() {
        DatabaseReference bandsReference = database.getReference("bands");

        bandsReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                bandsList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Band band = dataSnapshot.getValue(Band.class);
                    bandsList.add(band);
                }
                Log.d("BAND DAO", "Read " + bandsList.size() + " bands");
                DatabaseInitializer.inc();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        Log.d("BAND DAO", "Loaded " + bandsList.size() + " bands");
    }
}
```

Файл CandidatesDAO.java

Реалізація функціональної вимоги налаштування профілю користувача.

```
public class CandidatesDAO {
    private static CandidatesDAO instance;
    private final FirebaseDatabase database;

    private static ArrayList<Candidate> candidatesList;

    private CandidatesDAO() {
        candidatesList = new ArrayList<>();

        database = DatabaseConnectionProvider.getInstance().getDatabase();
        loadCandidates();
    }

    public static CandidatesDAO getInstance() {
        if (instance == null) {
            instance = new CandidatesDAO();
        }
        return instance;
    }

    public void createCandidate(Candidate candidate) {
        if (candidate.getCandidateUID() != null) {

database.getReference("candidates").child(candidate.getCandidateUID()).setVal
ue(candidate);

database.getReference("users").child(candidate.getCandidateUID()).setValue(ca
ndidate.getUser());

            Log.d("CANDIDATE DAO", "Adding candidate:\n" + candidate);
        } else {
            Log.d("CANDIDATE DAO", "Key is null");
```

```
        }
    }

    public Candidate readCandidate(String candidateUID) {
        Log.d("CANDIDATE DAO", "Reading candidate: " + candidateUID);
        if (candidatesList.size() != 0) {
            for (Candidate candidate : candidatesList) {
                if (candidate.getCandidateUID().equals(candidateUID)) {
                    return candidate;
                }
            }
        }
        return null;
    }

    public void updateCandidate(Candidate candidate) {

database.getReference("candidates").child(candidate.getCandidateUID()).setVal
ue(candidate);

database.getReference("users").child(candidate.getCandidateUID()).setValue(ca
ndidate.getUser());

        for (int i = 0; i < candidatesList.size(); i++) {
            if
(candidatesList.get(i).getCandidateUID().equals(candidate.getCandidateUID()))
{
                candidatesList.set(i, candidate);
                break;
            }
        }

        Log.d("CANDIDATE DAO", "Updating candidate:\n" + candidate);
    }

    public void loadCandidates() {
        DatabaseReference candidatesReference =
database.getReference("candidates");

        candidatesReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                candidatesList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Candidate candidate =
dataSnapshot.getValue(Candidate.class);
                    candidatesList.add(candidate);
                }
                Log.d("CANDIDATE DAO", "Read " + candidatesList.size() + "
candidates");
                DatabaseInitializer.inc();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                Log.d("CANDIDATE DAO", "Cancelled: " + error.getMessage());
            }
        });

        Log.d("CANDIDATE DAO", "Loaded " + candidatesList.size() + "
candidates");
    }
}
```

Файл VacanciesDAO.java
Реалізація функціональної вимоги системи роботи з вакансіями.

```
public class VacanciesDAO {
    private static VacanciesDAO instance;
    private final FirebaseDatabase database;

    private static ArrayList<Vacancy> vacanciesList;

    private RecommendedVacanciesRecyclerAdapter
recommendedVacanciesRecyclerAdapter;
    private BandVacanciesRecyclerAdapter bandVacanciesRecyclerAdapter;

    private VacanciesDAO() {
        vacanciesList = new ArrayList<>();

        database = DatabaseConnectionProvider.getInstance().getDatabase();
        loadVacancies();
    }

    public static VacanciesDAO getInstance() {
        if (instance == null) {
            instance = new VacanciesDAO();
        }
        return instance;
    }

    public void createVacancy(Vacancy vacancy) {
        String key = database.getReference("vacancies").push().getKey();

        if (key != null) {
            vacancy.setVacancyUID(key);
            database.getReference("vacancies").child(key).setValue(vacancy);
            Log.d("VACANCY DAO", "Adding vacancy:\n" + vacancy);
        } else {
            Log.d("VACANCY DAO", "Key is null");
        }
    }

    public Vacancy readVacancy(String vacancyUID) {
        Log.d("VACANCY DAO", "Reading vacancy: " + vacancyUID);
        if (vacanciesList.size() != 0) {
            for (Vacancy vacancy : vacanciesList) {
                if (vacancy.getVacancyUID().equals(vacancyUID)) {
                    return vacancy;
                }
            }
        }
        return null;
    }

    public void updateVacancy(Vacancy vacancy) {

database.getReference("vacancies").child(vacancy.getVacancyUID()).setValue(va
cancy);

        for (int i = 0; i < vacanciesList.size(); i++) {
            if
(vacanciesList.get(i).getVacancyUID().equals(vacancy.getVacancyUID())) {
                vacanciesList.set(i, vacancy);
                break;
            }
        }
```

```
        Log.d("VACANCY DAO", "Updating vacancy:\n" + vacancy);
    }

    public void deleteVacancy(String vacancyUID) {
        database.getReference("vacancies").child(vacancyUID).removeValue();

        for (Vacancy vacancy : vacanciesList) {
            if (vacancy.getVacancyUID().equals(vacancyUID)) {
                vacanciesList.remove(vacancy);
                break;
            }
        }

        Log.d("VACANCY DAO", "Deleting vacancy: " + vacancyUID);
        Log.d("VACANCY DAO", vacanciesList.toString());
    }

    public void loadVacancies() {
        DatabaseReference vacanciesReference =
database.getReference("vacancies");

        vacanciesReference.addValueEventListener(new ValueEventListener() {
            @SuppressLint("NotifyDataSetChanged")
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                vacanciesList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Vacancy vacancy = dataSnapshot.getValue(Vacancy.class);
                    vacanciesList.add(vacancy);
                }
                if (recommendedVacanciesRecyclerAdapter != null) {

recommendedVacanciesRecyclerAdapter.setArrayList(getRecommendedVacancies(Auth
UID.getUID()));

recommendedVacanciesRecyclerAdapter.notifyDataSetChanged();
                }
                if (bandVacanciesRecyclerAdapter != null) {

bandVacanciesRecyclerAdapter.setArrayList(getBandVacancies(AuthUID.getUID()))
;
                    bandVacanciesRecyclerAdapter.notifyDataSetChanged();
                }
                Log.d("VACANCY DAO", "Read " + vacanciesList.size() + "
vacancies");
                DatabaseInitializer.inc();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        Log.d("VACANCY DAO", "Loaded " + vacanciesList.size() + "
vacancies");
    }

    public ArrayList<Vacancy> getRecommendedVacancies(String candidateUID) {
        ResumesDAO resumesDAO = ResumesDAO.getInstance();
        ArrayList<Vacancy> recommendedVacancies = new ArrayList<>();
        for (Vacancy vacancy : vacanciesList) {
            if(!resumesDAO.isSentResume(vacancy.getVacancyUID())) {
                if (isRecommendedVacancy(vacancy, candidateUID)) {
```

```
                recommendedVacancies.add(vacancy);
                }
            }
        }
        return recommendedVacancies;
    }

    private boolean isRecommendedVacancy(Vacancy vacancy, String
candidateUID) {
        Candidate candidate =
CandidatesDAO.getInstance().readCandidate(candidateUID);
        String bandCity =
BandsDAO.getInstance().readBand(vacancy.getBandUID()).getCity();
        return candidate.getCity().equals(bandCity) &&
StringHelper.inString(vacancy.getRole(), candidate.getRole());
    }

    public void
setRecommendedVacanciesRecyclerAdapter(RecommendedVacanciesRecyclerAdapter
adapter) {
        this.recommendedVacanciesRecyclerAdapter = adapter;
    }

    public ArrayList<Vacancy> getBandVacancies(String bandUID) {
        ArrayList<Vacancy> bandVacancies = new ArrayList<>();
        for (Vacancy vacancy : vacanciesList) {
            if (vacancy.getBandUID().equals(bandUID)) {
                bandVacancies.add(vacancy);
            }
        }
        return bandVacancies;
    }

    public void setBandVacanciesRecyclerAdapter(BandVacanciesRecyclerAdapter
adapter) {
        this.bandVacanciesRecyclerAdapter = adapter;
    }
}
```

Файл ResumesDAO.java

Реалізація функціональної вимоги системи обробки резюме.

```
public class ResumesDAO {
    private static ResumesDAO instance;
    private final FirebaseDatabase database;

    private static ArrayList<Resume> resumesList;

    private ReceivedResumesRecyclerAdapter receivedResumesRecyclerAdapter;
    private ResumeHistoryRecyclerAdapter resumeHistoryRecyclerAdapter;
    private ActiveResumeRecyclerAdapter activeResumeRecyclerAdapter;

    private ResumesDAO() {
        resumesList = new ArrayList<>();

        database = DatabaseConnectionProvider.getInstance().getDatabase();
        loadResumes();
    }

    public static ResumesDAO getInstance() {
        if (instance == null) {
            instance = new ResumesDAO();
```

```
        }
        return instance;
    }

    public void createResume(Resume resume) {
        String key = database.getReference("resumes").push().getKey();

        if (key != null) {
            resume.setResumeUID(key);
            database.getReference("resumes").child(key).setValue(resume);
            Log.d("RESUME DAO", "Adding resume:\n" + resume);
        } else {
            Log.d("RESUME DAO", "Key is null");
        }
    }

    public Resume readResume(String resumeUID) {
        Log.d("RESUME DAO", "Reading resume: " + resumeUID);
        if (resumesList.size() != 0) {
            for (Resume resume : resumesList) {
                if (resume.getResumeUID().equals(resumeUID)) {
                    return resume;
                }
            }
        }
        return null;
    }

    @SuppressLint("NotifyDataSetChanged")
    public void updateResume(Resume resume) {

database.getReference("resumes").child(resume.getResumeUID()).setValue(resume
);

        for (int i = 0; i < resumesList.size(); i++) {
            if
(resumesList.get(i).getResumeUID().equals(resume.getResumeUID())) {
                resumesList.set(i, resume);
                break;
            }
        }

        if (receivedResumesRecyclerAdapter != null) {

receivedResumesRecyclerAdapter.setArrayList(getReceivedResumes(AuthUID.getUID
()));
            receivedResumesRecyclerAdapter.notifyDataSetChanged();
        }
        if (resumeHistoryRecyclerAdapter != null) {

resumeHistoryRecyclerAdapter.setArrayList(getResumeHistory(AuthUID.getUID()))
;
            resumeHistoryRecyclerAdapter.notifyDataSetChanged();
        }
        if (activeResumeRecyclerAdapter != null) {

activeResumeRecyclerAdapter.setArrayList(getActiveResumes(AuthUID.getUID()));
            activeResumeRecyclerAdapter.notifyDataSetChanged();
        }

        Log.d("RESUME DAO", "Updating resume:\n" + resume);
    }

    public void deleteResume(String resumeUID) {
```

```
        database.getReference("resumes").child(resumeUID).removeValue();

        for (Resume resume : resumesList) {
            if (resume.getResumeUID().equals(resumeUID)) {
                resumesList.remove(resume);
                break;
            }
        }

        Log.d("RESUME DAO", "Deleting resume: " + resumeUID);
    }

    public void loadResumes() {
        DatabaseReference resumesReference =
database.getReference("resumes");

        resumesReference.addValueEventListener(new ValueEventListener() {
            @SuppressLint("NotifyDataSetChanged")
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                resumesList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Resume resume = dataSnapshot.getValue(Resume.class);
                    resumesList.add(resume);
                }
                if (receivedResumesRecyclerAdapter != null) {

receivedResumesRecyclerAdapter.setArrayList(getReceivedResumes(AuthUID.getUID
()));
                    receivedResumesRecyclerAdapter.notifyDataSetChanged();
                }
                if (resumeHistoryRecyclerAdapter != null) {

resumeHistoryRecyclerAdapter.setArrayList(getResumeHistory(AuthUID.getUID()))
;
                    resumeHistoryRecyclerAdapter.notifyDataSetChanged();
                }
                if (activeResumeRecyclerAdapter != null) {

activeResumeRecyclerAdapter.setArrayList(getActiveResumes(AuthUID.getUID()));
                    activeResumeRecyclerAdapter.notifyDataSetChanged();
                }
                Log.d("RESUME DAO", "Read " + resumesList.size() + "
resumes");
                DatabaseInitializer.inc();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        Log.d("RESUME DAO", "Loaded " + resumesList.size() + " resumes");
    }

    public ArrayList<Resume> getReceivedResumes(String bandUID) {
        ArrayList<Resume> receivedResumes = new ArrayList<>();
        VacanciesDAO vacanciesDAO = VacanciesDAO.getInstance();
        for (Resume resume : resumesList) {
            if (resume.getStatus().equals(ResumeStatus.NEW.toString())) {
                String vacancyBandUID =
vacanciesDAO.readVacancy(resume.getVacancyUID()).getBandUID();
                if (vacancyBandUID.equals(bandUID)) {
```

```
                    receivedResumes.add(resume);
                }
            }
        }
        return receivedResumes;
    }

    public void
setReceivedResumesRecyclerAdapter(ReceivedResumesRecyclerAdapter adapter) {
        this.receivedResumesRecyclerAdapter = adapter;
    }

    public void markAllResumesDeclinedForVacancy(String vacancyUID) {
        for (Resume resume : resumesList) {
            if (resume.getVacancyUID().equals(vacancyUID)) {
                resume.setStatus(ResumeStatus.DECLINED.toString());
                updateResume(resume);
            }
        }
    }

    public boolean isSentResume(String vacancyUID) {
        for (Resume resume : resumesList) {
            if (resume.getVacancyUID().equals(vacancyUID) &&
resume.getCandidateUID().equals(AuthUID.getUID())) {
                return true;
            }
        }
        return false;
    }

    public ArrayList<Resume> getResumeHistory(String uid) {
        ArrayList<Resume> resumeHistory = new ArrayList<>();
        for (Resume resume : resumesList) {
            if (resume.getCandidateUID().equals(uid) &&
!resume.getStatus().equals(ResumeStatus.NEW.toString())) {
                resumeHistory.add(resume);
            }
        }
        return resumeHistory;
    }

    public void setResumeHistoryRecyclerAdapter(ResumeHistoryRecyclerAdapter
adapter) {
        this.resumeHistoryRecyclerAdapter = adapter;
    }

    public ArrayList<Resume> getActiveResumes(String uid) {
        ArrayList<Resume> activeResumes = new ArrayList<>();
        for (Resume resume : resumesList) {
            if (resume.getCandidateUID().equals(uid) &&
resume.getStatus().equals(ResumeStatus.NEW.toString())) {
                activeResumes.add(resume);
            }
        }
        return activeResumes;
    }

    public void setActiveResumesRecyclerAdapter(ActiveResumeRecyclerAdapter
adapter) {
        this.activeResumeRecyclerAdapter = adapter;
    }
}
```

Файл ChatsDAO.java
Реалізація функціональної вимоги системи чату.

```
public class ChatsDAO {
    private static ChatsDAO instance;
    private final FirebaseDatabase database;

    private static ArrayList<Chat> chatsList;
    private ChatsRecyclerAdapter chatsRecyclerAdapter;

    private ChatsDAO() {
        chatsList = new ArrayList<>();

        database = DatabaseConnectionProvider.getInstance().getDatabase();
        loadChats();
    }

    public static ChatsDAO getInstance() {
        if (instance == null) {
            instance = new ChatsDAO();
        }
        return instance;
    }

    public void createChat(Chat chat) {
        String key = database.getReference("chats").push().getKey();

        if (key != null) {
            chat.setChatUID(key);
            database.getReference("chats").child(key).setValue(chat);
            Log.d("CHAT DAO", "Adding chat:\n" + chat);
        } else {
            Log.d("CHAT DAO", "Key is null");
        }
    }

    public Chat readChat(String chatUID) {
        Log.d("CHAT DAO", "Reading chat: " + chatUID);
        if (chatsList.size()!= 0) {
            for (Chat chat : chatsList) {
                if (chat.getChatUID().equals(chatUID)) {
                    return chat;
                }
            }
        }
        return null;
    }

    public void updateChat(Chat chat) {

database.getReference("chats").child(chat.getChatUID()).setValue(chat);

        for (int i = 0; i < chatsList.size(); i++) {
            if (chatsList.get(i).getChatUID().equals(chat.getChatUID())) {
                chatsList.set(i, chat);
                break;
            }
        }

        if (chatsRecyclerAdapter != null) {
            chatsRecyclerAdapter.setArrayList(chatsList);
            chatsRecyclerAdapter.notifyDataSetChanged();
        }
```

```
        Log.d("CHAT DAO", "Updating chat:\n" + chat);
    }

    public void deleteChat(String chatUID) {
        database.getReference("chats").child(chatUID).removeValue();

        for (Chat chat : chatsList) {
            if (chat.getChatUID().equals(chatUID)) {
                chatsList.remove(chat);
                break;
            }
        }

        if (chatsRecyclerAdapter!= null) {
            chatsRecyclerAdapter.setArrayList(chatsList);
            chatsRecyclerAdapter.notifyDataSetChanged();
        }

        Log.d("CHAT DAO", "Deleting chat:\n" + chatUID);
    }

    public void loadChats() {
        DatabaseReference chatsReference = database.getReference("chats");

        chatsReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                chatsList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Chat chat = dataSnapshot.getValue(Chat.class);
                    chatsList.add(chat);
                }
                if (chatsRecyclerAdapter != null) {
                    chatsRecyclerAdapter.setArrayList(chatsList);
                    chatsRecyclerAdapter.notifyDataSetChanged();
                }

                Log.d("CHAT DAO", "Read " + chatsList.size() + " chats");
                DatabaseInitializer.inc();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        Log.d("CHAT DAO", "Loaded " + chatsList.size() + " chats");
    }

    public void setChatsRecyclerAdapter(ChatsRecyclerAdapter
chatsRecyclerAdapter) {
        this.chatsRecyclerAdapter = chatsRecyclerAdapter;
    }

    public ArrayList<Chat> getChats(String userUID, String userType) {
        ArrayList<Chat> chatList = new ArrayList<>();

        if (userType.equals(UserType.CANDIDATE.toString())) {
            for (Chat chat : chatsList) {
                if (chat.getCandidateMemberUID().equals(userUID)) {
                    chatList.add(chat);
                }
            }
```

```
        } else {
            for (Chat chat : chatsList) {
                if (chat.getBandMemberUID().equals(userUID)) {
                    chatList.add(chat);
                }
            }
        }

        return chatList;
    }

    public Chat find(String candidateUID, String bandUID) {
        for (Chat chat : chatsList) {
            if (chat.getCandidateMemberUID().equals(candidateUID) &&
chat.getBandMemberUID().equals(bandUID)) {
                return chat;
            }
        }
        return null;
    }
}
```

Файл MessagesDAO.java

Реалізація функціональної вимоги відображення історії чату.

```
public class MessagesDAO {
    private static MessagesDAO instance;
    private final FirebaseDatabase database;

    private final ArrayList<Message> messagesList;

    private UsersDAO usersDAO;

    private MessagesDAO() {
        messagesList = new ArrayList<>();

        database = DatabaseConnectionProvider.getInstance().getDatabase();
        usersDAO = UsersDAO.getInstance();

        loadMessages();
    }

    public static MessagesDAO getInstance() {
        if (instance == null) {
            instance = new MessagesDAO();
        }
        return instance;
    }

    public void createMessage(Message message) {
        String key = database.getReference("messages").push().getKey();

        if (key != null) {
            message.setMessageUID(key);
            database.getReference("messages").child(key).setValue(message);
            Log.d("MESSAGE DAO", "Adding message:\n" + message);
        } else {
            Log.d("MESSAGE DAO", "Key is null");
        }
    }

    public Message readMessage(String messageUID) {
        Log.d("MESSAGE DAO", "Reading message: " + messageUID);
```

```
        if (messagesList.size() != 0) {
            for (Message message : messagesList) {
                if (message.getMessageUID().equals(messageUID)) {
                    return message;
                }
            }
        }
        return null;
    }

    public void updateMessage(Message message) {

database.getReference("messages").child(message.getMessageUID()).setValue(mes
sage);

        for (int i = 0; i < messagesList.size(); i++) {
            if
(messagesList.get(i).getMessageUID().equals(message.getMessageUID())) {
                messagesList.set(i, message);
                break;
            }
        }

        Log.d("MESSAGE DAO", "Updating message:\n" + message);
    }

    public void loadMessages() {
        DatabaseReference messagesReference =
database.getReference("messages");

        messagesReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                messagesList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Message message = dataSnapshot.getValue(Message.class);
                    messagesList.add(message);
                }
                Log.d("MESSAGE DAO", "Read " + messagesList.size() + "
messages");
                DatabaseInitializer.inc();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        Log.d("MESSAGE DAO", "Loaded " + messagesList.size() + " messages");
    }

    public Message getLastMessage(String chatUID) {
        Message res = null;
        for (Message message : messagesList) {
            if (res == null)
                res = message;
            else if (message.getChatUID().equals(chatUID) &&
res.getDatetime().compareTo(message.getDatetime()) <= 0) {
                res = message;
            }
        }
        return res;
    }
```

```
    public int getNewMessagesCount(String chatUID) {
        String myType = usersDAO.readUser(AuthUID.getUID()).getType();
        int count = 0;
        for (Message message : messagesList) {
            if (message.getChatUID().equals(chatUID) &&
message.getStatus().equals(MessageStatus.SENT.toString()) &&
!message.getSenderType().equals(myType)) {
                count++;
            }
        }
        return count;
    }
}
```

Файл NotificationsDAO.java

Реалізація функціональної вимоги отримання сповіщень.

```
public class NotificationsDAO {
    private static NotificationsDAO instance;
    private final FirebaseDatabase database;

    private static ArrayList<Notification> notificationsList;
    private NotificationRecyclerAdapter notificationRecyclerAdapter;


    private NotificationsDAO() {
        notificationsList = new ArrayList<>();

        database = DatabaseConnectionProvider.getInstance().getDatabase();
        loadNotifications();
    }

    public static NotificationsDAO getInstance() {
        if (instance == null) {
            instance = new NotificationsDAO();
        }
        return instance;
    }

    public void createNotification(Notification notification) {
        String key = database.getReference("notifications").push().getKey();

        if (key != null) {
            notification.setNotificationUID(key);

database.getReference("notifications").child(key).setValue(notification);
            Log.d("NOTIFICATION DAO", "Adding notification:\n" +
notification);
        } else {
            Log.d("NOTIFICATION DAO", "Key is null");
        }
    }

    public Notification readNotification(String notificationUID) {
        Log.d("NOTIFICATION DAO", "Reading notification: " +
notificationUID);
        if (notificationsList.size() != 0) {
            for (Notification notification : notificationsList) {
                if
(notification.getNotificationUID().equals(notificationUID)) {
                    return notification;
                }
```

```
            }
        }
        return null;
    }

    @SuppressLint("NotifyDataSetChanged")
    public void updateNotification(Notification notification) {

database.getReference("notifications").child(notification.getNotificationUID(
)).setValue(notification);

        for (int i = 0; i < notificationsList.size(); i++) {
            if
(notificationsList.get(i).getNotificationUID().equals(notification.getNotific
ationUID())) {
                notificationsList.set(i, notification);
                break;
            }
        }

        if (notificationRecyclerAdapter != null) {
            notificationRecyclerAdapter.setArrayList(notificationsList);
            notificationRecyclerAdapter.notifyDataSetChanged();
        }

        Log.d("NOTIFICATION DAO", "Updating notification:\n" + notification);
    }

    @SuppressLint("NotifyDataSetChanged")
    public void deleteNotification(String notificationUID) {

database.getReference("notifications").child(notificationUID).removeValue();

        for (Notification notification : notificationsList) {
            if (notification.getNotificationUID().equals(notificationUID)) {
                notificationsList.remove(notification);
                break;
            }
        }

        if (notificationRecyclerAdapter!= null) {
            notificationRecyclerAdapter.setArrayList(notificationsList);
            notificationRecyclerAdapter.notifyDataSetChanged();
        }

        Log.d("NOTIFICATION DAO", "Deleting notification: " +
notificationUID);
    }

    public void loadNotifications() {
        DatabaseReference notificationsReference =
database.getReference("notifications");

        notificationsReference.addValueEventListener(new ValueEventListener()
{
            @SuppressLint("NotifyDataSetChanged")
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                notificationsList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Notification notification =
dataSnapshot.getValue(Notification.class);
                    notificationsList.add(notification);
                }
```

```
                if (notificationRecyclerAdapter != null) {

notificationRecyclerAdapter.setArrayList(notificationsList);
                    notificationRecyclerAdapter.notifyDataSetChanged();
                }

                Log.d("NOTIFICATION DAO", "Read " + notificationsList.size()
+ " notifications");
                DatabaseInitializer.inc();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });

        Log.d("NOTIFICATION DAO", "Loaded " + notificationsList.size() + "
notifications");
    }

    public ArrayList<Notification> getNotifications(String userUID) {
        ArrayList<Notification> userNotifications = new ArrayList<>();

        for (int i = notificationsList.size() - 1; i >= 0; i--) {
            if (notificationsList.get(i).getReceiverUID().equals(userUID)) {
                userNotifications.add(notificationsList.get(i));
            }
        }

        return userNotifications;
    }

    public void setNotificationRecyclerAdapter(NotificationRecyclerAdapter
notificationRecyclerAdapter) {
        this.notificationRecyclerAdapter = notificationRecyclerAdapter;
    }

    @SuppressLint("NotifyDataSetChanged")
    public void setNotificationsRead(String uid) {
        for (Notification notification : notificationsList) {
            if (notification.getReceiverUID().equals(uid) &&
notification.getStatus().equals(NotificationStatus.NEW.toString())) {
                notification.setStatus(NotificationStatus.READ.toString());
                updateNotification(notification);
            }
        }

        if (notificationRecyclerAdapter != null) {
            notificationRecyclerAdapter.setArrayList(notificationsList);
            notificationRecyclerAdapter.notifyDataSetChanged();
        }

        Log.d("NOTIFICATION DAO", "Marking notifications as read");
    }
}
```

Файл ImageStorageProvider.java

Реалізація функціональної вимоги встановлення аватару користувача.

```
public class ImageStorageProvider {
    private static final String bucketUrl = "bander-63922.appspot.com";
    private static ImageStorageProvider instance;
```

```
    private static FirebaseStorage storage;

    private ImageStorageProvider() {
        storage = FirebaseStorage.getInstance("gs://" + bucketUrl);
        Log.d("STORAGE", "Image Storage Provider initialized");
    }

    public static ImageStorageProvider getInstance() {
        if (instance == null) {
            instance = new ImageStorageProvider();
        }
        return instance;
    }

    public void uploadImage(Context context, String userUid, Uri imageUri) {
        StorageReference filepath =
storage.getReference().child("profileImages").child(userUid);
        Log.d("STORAGE", "Filepath reference: " + filepath);
        try {
            Bitmap bitmap =
ImageHelper.handleSamplingAndRotationBitmap(context, imageUri);

            ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
            bitmap.compress(Bitmap.CompressFormat.JPEG, 20,
byteArrayOutputStream);

            byte[] imageData = byteArrayOutputStream.toByteArray();
            UploadTask uploadTask = filepath.putBytes(imageData);

            uploadTask.addOnFailureListener(e -> {
                Log.e("STORAGE", "Can't upload the image for user " +
userUid);
            });
        } catch (IOException e) {
            Log.e("STORAGE", "Can't upload the image for user " + userUid);
            Log.e("STORAGE", e.getMessage());
        }
    }

    public Uri downloadImageUri(String userUid) {
        Uri imageUri =
Uri.parse("https://firebasestorage.googleapis.com/v0/b/" + bucketUrl +
"/o/profileImages%2F" + userUid + "?alt=media");
        Log.d("STORAGE", "Download Image Uri: " + imageUri);
        return imageUri;
    }
}
```

Файл LoginRegisterActivity.java

Реалізація функціональної вимоги входу в обліковий запис.

```
public class LoginRegisterActivity extends AppCompatActivity {
    private EditText mEmail, mPassword;

    private AuthProvider authProvider;
    private static FirebaseAuth.AuthStateListener firebaseAuthStateListener;

    public static void removeCover() {

AuthProvider.getInstance().addAuthStateListener(firebaseAuthStateListener);
    }
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login_register);

        Window window = getWindow();

window.addFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS)
;

window.clearFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS);
        window.setStatusBarColor(ContextCompat.getColor(this,
R.color.primary));

        DatabaseInitializer.init();
        authProvider = AuthProvider.getInstance();
        firebaseAuthStateListener = firebaseAuth -> {
            final FirebaseUser user =
FirebaseAuth.getInstance().getCurrentUser();
            if (user != null) {
                Intent intent = new Intent(LoginRegisterActivity.this,
MainActivity.class);
                startActivity(intent);
                finish();
            } else {
                ImageView mCoverImage = findViewById(R.id.coverImage);
                mCoverImage.setVisibility(View.GONE);
                Button mLoginButton = findViewById(R.id.login_button);
                mLoginButton.setVisibility(View.VISIBLE);
            }
        };

        mEmail = findViewById(R.id.emailEditText);
        mPassword = findViewById(R.id.passwordEditText);

        if (DatabaseInitializer.isInitialized()) {
            removeCover();
        }
    }

    public void login(View view) {
        final String email = mEmail.getText().toString();
        final String password = mPassword.getText().toString();
        authProvider.login(email,
password).addOnCompleteListener(LoginRegisterActivity.this, task -> {
            if (task.isSuccessful()) {
                Log.d("Auth", "LOG IN successfully");
            }
            else {
                Toast.makeText(LoginRegisterActivity.this, "Log in error",
Toast.LENGTH_SHORT).show();
            }
        });
    }

    public void openRegistrationActivity(View view) {
        Intent intent = new Intent(LoginRegisterActivity.this,
RegistrationActivity.class);
        startActivity(intent);
        finish();
    }

    @Override
```

```
    protected void onStart() {
        super.onStart();
    }

    @Override
    protected void onStop() {
        super.onStop();
        authProvider.removeAuthStateListener(firebaseAuthStateListener);
    }

    public void openBandRegistrationActivity(View view) {
        Intent intent = new Intent(LoginRegisterActivity.this,
BandRegistrationActivity.class);
        startActivity(intent);
        finish();
    }
}
```

Файл BandRegistrationActivity.java

Реалізація функціональної вимоги реєстрації облікового запису.

```
public class BandRegistrationActivity extends AppCompatActivity {
    private EditText mEmail, mPassword, mConfirmPassword;
    private EditText mName, mCity;

    private AuthProvider authProvider;
    private FirebaseAuth.AuthStateListener firebaseAuthStateListener;

    private BandsDAO bandsDAO;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_band_registration);

        Window window = getWindow();

window.addFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS)
;

window.clearFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS);
        window.setStatusBarColor(ContextCompat.getColor(this,
R.color.primary));

        authProvider = AuthProvider.getInstance();
        firebaseAuthStateListener = firebaseAuth -> {
            final FirebaseUser user =
FirebaseAuth.getInstance().getCurrentUser();
            if (user != null) {
                Intent intent = new Intent(BandRegistrationActivity.this,
MainActivity.class);
                startActivity(intent);
                finish();
            }
        };

        bandsDAO = BandsDAO.getInstance();

        mEmail = findViewById(R.id.emailEditText);
        mPassword = findViewById(R.id.passwordEditText);
        mConfirmPassword = findViewById(R.id.confirmPasswordEditText);
        mName = findViewById(R.id.nameEditText);
        mCity = findViewById(R.id.cityEditText);
```

```
    }

    public void registerBand(View view) {
        final String email = mEmail.getText().toString();
        final String password = mPassword.getText().toString();
        final String confirmPassword = mConfirmPassword.getText().toString();
        final String name = mName.getText().toString();
        final String city = mCity.getText().toString();

        if (name.isEmpty() || city.isEmpty()) {
            Toast.makeText(BandRegistrationActivity.this, "Fill all fields",
Toast.LENGTH_SHORT).show();
            Log.d("Auth", "Some fields are empty");
            return;
        }

        authProvider.register(email, password,
confirmPassword).addOnCompleteListener(BandRegistrationActivity.this, task ->
{
            if (task.isSuccessful()) {
                Band band = new Band(authProvider.getUid(), email, name,
city, "", "", "", new ArrayList<>());
                bandsDAO.createBand(band);

                Log.d("Auth", "Created new band: " + band);
                Log.d("Auth", "SIGN UP successfully");
            }
            else {
                if (task.getException() instanceof
FirebaseAuthUserCollisionException) {
                    Toast.makeText(BandRegistrationActivity.this, "User with
this email already exists", Toast.LENGTH_SHORT).show();
                    Log.d("Auth", "User with this email already exists");
                } else {
                    Toast.makeText(BandRegistrationActivity.this,
Objects.requireNonNull(task.getException()).getMessage(),
Toast.LENGTH_SHORT).show();
                    Log.d("Auth",
Objects.requireNonNull(task.getException().getMessage()));
                }
            }
        });
    }

    public void openLoginActivity(View view) {
        Intent intent = new Intent(BandRegistrationActivity.this,
LoginRegisterActivity.class);
        startActivity(intent);
        finish();
    }

    @Override
    protected void onStart() {
        super.onStart();
        authProvider.addAuthStateListener(firebaseAuthStateListener);
    }

    @Override
    protected void onStop() {
        super.onStop();
        authProvider.removeAuthStateListener(firebaseAuthStateListener);
    }
}
```

Файл RegistrationActivity.java

Реалізація функціональної вимоги реєстрації облікового запису.

```
public class RegistrationActivity extends AppCompatActivity {
    private EditText mEmail, mPassword, mConfirmPassword;
    private EditText mName, mSurname, mBirthday, mCity;

    private AuthProvider authProvider;
    private FirebaseAuth.AuthStateListener firebaseAuthStateListener;

    private CandidatesDAO candidatesDAO;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_registration);

        Window window = getWindow();

window.addFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS)
;

window.clearFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS);
        window.setStatusBarColor(ContextCompat.getColor(this,
R.color.primary));

        init();
    }

    private void init() {
        authProvider = AuthProvider.getInstance();
        firebaseAuthStateListener = firebaseAuth -> {
            final FirebaseUser user =
FirebaseAuth.getInstance().getCurrentUser();
            if (user != null) {
                Intent intent = new Intent(RegistrationActivity.this,
MainActivity.class);
                startActivity(intent);
                finish();
            }
        };

        candidatesDAO = CandidatesDAO.getInstance();

        mEmail = findViewById(R.id.emailEditText);
        mPassword = findViewById(R.id.passwordEditText);
        mConfirmPassword = findViewById(R.id.confirmPasswordEditText);
        mName = findViewById(R.id.nameEditText);
        mSurname = findViewById(R.id.surnameEditText);
        mBirthday = findViewById(R.id.dateEditText);
        mCity = findViewById(R.id.cityEditText);

        Date currentDate = new Date();
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(currentDate);
        calendar.add(Calendar.YEAR, -12);
        transformIntoDatePicker(mBirthday, this, calendar.getTime());
    }

    public void registerCandidate(View view) {
        final String email = mEmail.getText().toString();
        final String password = mPassword.getText().toString();
        final String confirmPassword = mConfirmPassword.getText().toString();
        final String name = mName.getText().toString();
```

```
        final String surname = mSurname.getText().toString();
        final String birthday = mBirthday.getText().toString();
        final String city = mCity.getText().toString();

        if (name.isEmpty() || surname.isEmpty() || birthday.isEmpty() ||
city.isEmpty()) {
            Toast.makeText(RegistrationActivity.this, "Fill all fields",
Toast.LENGTH_SHORT).show();
            Log.d("Auth", "Some fields are empty");
            return;
        }

        authProvider.register(email, password,
confirmPassword).addOnCompleteListener(RegistrationActivity.this, task -> {
            if (task.isSuccessful()) {
                Candidate candidate = new Candidate(authProvider.getUid(),
email, name, surname, birthday, city, "", "", "", "", "");
                candidatesDAO.createCandidate(candidate);

                Log.d("Auth", "Created new candidate: " + candidate);
                Log.d("Auth", "SIGN UP successfully");
            }
            else {
                if (task.getException() instanceof
FirebaseAuthUserCollisionException) {
                    Toast.makeText(RegistrationActivity.this, "User with this
email already exists", Toast.LENGTH_SHORT).show();
                    Log.d("Auth", "User with this email already exists");
                } else {
                    Toast.makeText(RegistrationActivity.this,
Objects.requireNonNull(task.getException()).getMessage(),
Toast.LENGTH_SHORT).show();
                    Log.d("Auth",
Objects.requireNonNull(task.getException().getMessage()));
                }
            }
        });
    }

    public void openLoginActivity(View view) {
        Intent intent = new Intent(RegistrationActivity.this,
LoginRegisterActivity.class);
        startActivity(intent);
        finish();
    }

    @Override
    protected void onStart() {
        super.onStart();
        authProvider.addAuthStateListener(firebaseAuthStateListener);
    }

    @Override
    protected void onStop() {
        super.onStop();
        authProvider.removeAuthStateListener(firebaseAuthStateListener);
    }

    private static void transformIntoDatePicker(EditText editText, Context
context, Date maxDate) {
        editText.setFocusableInTouchMode(false);
        editText.setClickable(true);
        editText.setFocusable(false);
```

```
        final Calendar myCalendar = Calendar.getInstance();
        final DatePickerDialog.OnDateSetListener datePickerOnDataSetListener
=
                (view, year, monthOfYear, dayOfMonth) -> {
                    myCalendar.set(Calendar.YEAR, year);
                    myCalendar.set(Calendar.MONTH, monthOfYear);
                    myCalendar.set(Calendar.DAY_OF_MONTH, dayOfMonth);
                    SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy",
Locale.UK);
                    editText.setText(sdf.format(myCalendar.getTime()));
                    Log.d("Auth", "Date picked: " +
sdf.format(myCalendar.getTime()));
                };

        editText.setOnClickListener(v -> {
            DatePickerDialog datePickerDialog = new DatePickerDialog(
                    context, datePickerOnDataSetListener, myCalendar
                    .get(Calendar.YEAR), myCalendar.get(Calendar.MONTH),
                    myCalendar.get(Calendar.DAY_OF_MONTH)
            );
            if (maxDate != null) {

datePickerDialog.getDatePicker().setMaxDate(maxDate.getTime());
            }
            datePickerDialog.show();
        });
    }
}
```

Файл ChatActivity.java

Реалізація функціональної вимоги відображення історії чату, надсилання текстового повідомлення.

```
public class ChatActivity extends AppCompatActivity {
    private RecyclerView messageRecyclerView;
    private ArrayList<Message> messagesList;
    private MessagesRecyclerAdapter messagesRecyclerAdapter;
    private DatabaseReference messageReference;
    private UsersDAO usersDAO;

    private ImageView chatImage;
    private TextView chatTitle;
    private EditText messageText;

    private Chat chat;
    private String myType;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_chat);

        Window window = getWindow();

window.addFlags(WindowManager.LayoutParams.FLAG_DRAWS_SYSTEM_BAR_BACKGROUNDS)
;

window.clearFlags(WindowManager.LayoutParams.FLAG_TRANSLUCENT_STATUS);
        window.setStatusBarColor(ContextCompat.getColor(this,
R.color.primary));

        usersDAO = UsersDAO.getInstance();
```

```
        chat = (Chat) getIntent().getSerializableExtra("chat");
        myType = usersDAO.readUser(AuthUID.getUID()).getType();

        init();
        loadData();
        loadMessages();
    }

    private void init() {
        messageReference =
DatabaseConnectionProvider.getInstance().getDatabase().getReference("messages
");
        messagesList = new ArrayList<>();

        chatImage = findViewById(R.id.messageChatImage);
        chatTitle = findViewById(R.id.messageChatTitle);
        messageText = findViewById((R.id.messageWriteEditText));

        messageRecyclerView = findViewById(R.id.messageRecyclerView);
        messageRecyclerView.setLayoutManager(new LinearLayoutManager(this));
    }

    private void loadData() {
        String chatMemberUid =
chat.getCandidateMemberUID().equals(AuthUID.getUID()) ?
chat.getBandMemberUID() : chat.getCandidateMemberUID();
        Log.d("CHAT", "Chat member UID: " + chatMemberUid);

        ImageStorageProvider imageStorageProvider =
ImageStorageProvider.getInstance();
        Uri imageUri = imageStorageProvider.downloadImageUri(chatMemberUid);
        Glide
                .with(this)
                .load(imageUri)
                .apply(ImageOptions.imageOptions())
                .into(chatImage);


chatTitle.setText(UsersDAO.getInstance().readUser(chatMemberUid).getName());
    }

    private void loadMessages() {
        messageReference.addValueEventListener(new ValueEventListener() {

            @SuppressLint("NotifyDataSetChanged")
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                messagesList.clear();
                for (DataSnapshot dataSnapshot : snapshot.getChildren()) {
                    Message message = dataSnapshot.getValue(Message.class);

                    if (message != null &&
message.getChatUID().equals(chat.getChatUID())) {
                        messagesList.add(message);
                    }
                }
                messagesRecyclerAdapter = new
MessagesRecyclerAdapter(ChatActivity.this, messagesList);
                messageRecyclerView.setAdapter(messagesRecyclerAdapter);
                messagesRecyclerAdapter.notifyDataSetChanged();

messageRecyclerView.scrollToPosition(messagesRecyclerAdapter.getItemCount() -
1);
```

```
                Log.d("CHAT", "Load " + messagesList.size() + " messages");
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });
    }

    public void sendMessage(View view) {
        String message = messageText.getText().toString();

        if (!message.isEmpty()) {
            String key = messageReference.push().getKey();

            if (key != null) {
                LocalDateTime now = LocalDateTime.now();
                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm");
                String datetime = now.format(formatter);

                Message newMessage = new Message(
                        key,
                        chat.getChatUID(),
                        myType,
                        message,
                        datetime,
                        MessageStatus.SENT.toString()
                );

                messageReference.child(key).setValue(newMessage);
                messageText.setText("");

                Log.d("CHAT", "Message sent: " + newMessage);
            } else {
                Log.d("CHAT", "Key is null");
            }
        }
    }

    public void back(View view) {
        finish();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();

        for (Message message : messagesList) {
            if (!message.getSenderType().equals(myType) &&
message.getStatus().equals(MessageStatus.SENT.toString())) {

messageReference.child(message.getMessageUID()).child("status").setValue(Mess
ageStatus.READ.toString());
            }
        }
    }

    public void scrollDown(View view) {
        new Handler().postDelayed(() -> {

messageRecyclerView.scrollToPosition(messagesRecyclerAdapter.getItemCount() -
1);
```

```
        }, 300);
    }
}
```

Файл ChatsFragment.java

Реалізація функціональної вимоги системи чатів.

```
public class ChatsFragment extends Fragment {
    private ChatsViewModel chatsViewModel;
    private RecyclerView chatsRecyclerView;
    private ArrayList<Chat> chatsList;
    private ChatsRecyclerAdapter chatsRecyclerAdapter;
    private TextView mEmptyText;

    private FragmentChatsBinding binding;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState)
{
        chatsViewModel = new
ViewModelProvider(this).get(ChatsViewModel.class);

        binding = FragmentChatsBinding.inflate(inflater, container, false);
        View root = binding.getRoot();

        init();

        return root;
    }

    private void init() {
        mEmptyText = binding.chatsEmptyText;
        chatsRecyclerView = binding.chatsRecyclerView;
        chatsRecyclerView.setLayoutManager(new
LinearLayoutManager(requireActivity()));

        chatsList = chatsViewModel.getChatsList(AuthUID.getUID());
        if (chatsList.size() == 0) {
            mEmptyText.setVisibility(View.VISIBLE);
            chatsRecyclerView.setVisibility(View.INVISIBLE);
        } else {
            mEmptyText.setVisibility(View.INVISIBLE);
            chatsRecyclerView.setVisibility(View.VISIBLE);

            chatsRecyclerAdapter = new
ChatsRecyclerAdapter(requireActivity(), chatsList);
            chatsRecyclerView.setAdapter(chatsRecyclerAdapter);
            chatsViewModel.setChatsAdapter(chatsRecyclerAdapter);
        }
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        binding = null;
    }
}
```

Файл NotificationsFragment.java

Реалізація функціональної вимоги отримання сповіщень.

```
public class NotificationsFragment extends Fragment {
```

```
    private NotificationsViewModel notificationsViewModel;
    private RecyclerView notificationRecyclerView;
    private ArrayList<Notification> notificationsList;
    private NotificationRecyclerAdapter notificationRecyclerAdapter;
    private TextView mEmptyText;

    private FragmentNotificationsBinding binding;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState)
{
        notificationsViewModel = new
ViewModelProvider(this).get(NotificationsViewModel.class);

        binding = FragmentNotificationsBinding.inflate(inflater, container,
false);
        View root = binding.getRoot();

        init();

        return root;
    }

    private void init() {
        mEmptyText = binding.notificationEmptyText;
        notificationRecyclerView = binding.notificationRecyclerView;
        notificationRecyclerView.setLayoutManager(new
LinearLayoutManager(requireActivity()));

        notificationsList =
notificationsViewModel.getNotificationList(AuthUID.getUID());
        if (notificationsList.size() == 0) {
            mEmptyText.setVisibility(View.VISIBLE);
            notificationRecyclerView.setVisibility(View.INVISIBLE);
        } else {
            mEmptyText.setVisibility(View.INVISIBLE);
            notificationRecyclerView.setVisibility(View.VISIBLE);

            notificationRecyclerAdapter = new
NotificationRecyclerAdapter(requireActivity(), notificationsList);
            notificationRecyclerView.setAdapter(notificationRecyclerAdapter);

notificationsViewModel.setNotificationAdapter(notificationRecyclerAdapter);
;        }
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        notificationsViewModel.setNotificationsRead(AuthUID.getUID());
        binding = null;
    }
}
```

Файл ProfileEditActivity.java

Реалізація функціональної вимоги налаштування профілю користувача.

```
public class ProfileEditActivity extends AppCompatActivity {
    private EditText mName, mSurname, mBirthday, mCity;
    private EditText mExperience, mAbout, mRoles, mPreferredGenres,
mVideoLinks;
    private ImageView mProfileImage;
    private ActivityResultLauncher<Intent> resultLauncher;
```

```
    private Uri profileImageUri;

    private ImageStorageProvider imageStorageProvider;
    private CandidatesDAO candidatesDAO;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_profile_edit);

        imageStorageProvider = ImageStorageProvider.getInstance();
        candidatesDAO = CandidatesDAO.getInstance();
        init();
        loadData();
    }

    private void init() {
        mName = findViewById(R.id.profileEditTextName);
        mSurname = findViewById(R.id.profileEditTextSurname);
        mBirthday = findViewById(R.id.profileEditTextBirthday);
        mCity = findViewById(R.id.profileEditTextCity);
        mExperience = findViewById(R.id.profileEditTextExperience);
        mAbout = findViewById(R.id.profileEditTextAbout);
        mRoles = findViewById(R.id.profileEditTextRoles);
        mPreferredGenres = findViewById(R.id.profileEditTextGenres);
        mVideoLinks = findViewById(R.id.profileEditTextLinks);

        mProfileImage = findViewById(R.id.profileImageButton);

        mProfileImage.setOnClickListener(v -> {
            Intent intent = new Intent();
            intent.setType("image/*");
            intent.setAction(Intent.ACTION_GET_CONTENT);
            resultLauncher.launch(intent);
        });

        resultLauncher = registerForActivityResult(
                new ActivityResultContracts.StartActivityForResult(),
                activityResult -> {
                    try {
                        Uri imageUri =
Objects.requireNonNull(activityResult.getData()).getData();
                        mProfileImage.setImageURI(imageUri);
                        profileImageUri = imageUri;
                    } catch (Exception e) {
                        Log.e("PROFILE EDIT", "Can't to pick the image");
                    }
                }
        );
    }

    private void loadData() {
        Candidate candidate =
candidatesDAO.readCandidate(AuthProvider.getInstance().getUid());

        if (candidate != null) {
            if (!candidate.getName().isEmpty())
                mName.setText(candidate.getName());
            if (!candidate.getSurname().isEmpty())
                mSurname.setText(candidate.getSurname());
            if (!candidate.getBirthday().isEmpty())
                mBirthday.setText(candidate.getBirthday());
            if (!candidate.getCity().isEmpty())
                mCity.setText(candidate.getCity());
```

```
            if (!candidate.getExperience().isEmpty())
                mExperience.setText(candidate.getExperience());
            if (!candidate.getAbout().isEmpty())
                mAbout.setText(candidate.getAbout());
            if (!candidate.getRole().isEmpty())
                mRoles.setText(candidate.getRole());
            if (!candidate.getPreferredGenres().isEmpty())
                mPreferredGenres.setText(candidate.getPreferredGenres());
            if (!candidate.getVideoLinks().isEmpty())
                mVideoLinks.setText(candidate.getVideoLinks());
        }

        Uri profileImage =
imageStorageProvider.downloadImageUri(AuthUID.getUID());
        Glide
                .with(this)
                .load(profileImage)
                .apply(ImageOptions.imageOptions())
                .into(mProfileImage);

        Date currentDate = new Date();
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(currentDate);
        calendar.add(Calendar.YEAR, -12);
        transformIntoDatePicker(mBirthday, this, calendar.getTime());
    }

    private static void transformIntoDatePicker(EditText editText, Context
context, Date maxDate) {
        editText.setFocusableInTouchMode(false);
        editText.setClickable(true);
        editText.setFocusable(false);

        final Calendar myCalendar = Calendar.getInstance();
        final DatePickerDialog.OnDateSetListener datePickerOnDataSetListener
=
                (view, year, monthOfYear, dayOfMonth) -> {
                    myCalendar.set(Calendar.YEAR, year);
                    myCalendar.set(Calendar.MONTH, monthOfYear);
                    myCalendar.set(Calendar.DAY_OF_MONTH, dayOfMonth);
                    SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy",
Locale.UK);
                    editText.setText(sdf.format(myCalendar.getTime()));
                    Log.d("Auth", "Date picked: " +
sdf.format(myCalendar.getTime()));
                };

        editText.setOnClickListener(v -> {
            DatePickerDialog datePickerDialog = new DatePickerDialog(
                    context, datePickerOnDataSetListener, myCalendar
                    .get(Calendar.YEAR), myCalendar.get(Calendar.MONTH),
                    myCalendar.get(Calendar.DAY_OF_MONTH)
            );
            if (maxDate != null) {

datePickerDialog.getDatePicker().setMaxDate(maxDate.getTime());
            }
            datePickerDialog.show();
        });
    }

    public void save(View view) {
        AlertDialog.Builder builder = new
AlertDialog.Builder(ProfileEditActivity.this);
```

```
        builder
                .setTitle(R.string.text_to_sign_out_title)
                .setMessage(R.string.text_to_save_message)
                .setPositiveButton(R.string.yes, (dialog, which) -> {
                    String name = mName.getText().toString();
                    String surname = mSurname.getText().toString();
                    String birthday = mBirthday.getText().toString();
                    String city = mCity.getText().toString();
                    String experience = mExperience.getText().toString();
                    String about = mAbout.getText().toString();
                    String roles = mRoles.getText().toString();
                    String preferredGenres =
mPreferredGenres.getText().toString();
                    String videoLinks = mVideoLinks.getText().toString();

                    if (name.isEmpty() || surname.isEmpty() ||
birthday.isEmpty() || city.isEmpty()) {
                        Toast.makeText(ProfileEditActivity.this, "Fill all
required fields", Toast.LENGTH_SHORT).show();
                        Log.d("PROFILE EDIT", "Some fields are empty");
                        return;
                    }

                    Candidate oldCandidate =
candidatesDAO.readCandidate(AuthProvider.getInstance().getUid());
                    Candidate newCandidate = new Candidate(
                            oldCandidate.getCandidateUID(),
                            oldCandidate.getEmail(),
                            name,
                            surname,
                            birthday,
                            city,
                            about,
                            roles,
                            preferredGenres,
                            experience,
                            videoLinks);
                    candidatesDAO.updateCandidate(newCandidate);

                    if (profileImageUri != null)
                        imageStorageProvider.uploadImage(this,
AuthUID.getUID(), profileImageUri);

                    Intent intent = new Intent(ProfileEditActivity.this,
MainActivity.class);
                    startActivity(intent);
                    finish();
                })
                .setNegativeButton(R.string.no, (dialog, which) -> {})
                .show();
    }

    public void cancel(View view) {
        profileImageUri = null;
        finish();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        loadData();
    }
}
```

Файл BandEditActivity.java

Реалізація функціональної вимоги налаштування профілю користувача.

```
public class BandEditActivity extends AppCompatActivity {
    EditText mName, mCity, mGenres;
    EditText mAbout, mVideoLinks;
    private ImageView mProfileImage;
    private ActivityResultLauncher<Intent> resultLauncher;
    private Uri profileImageUri;

    private ImageStorageProvider imageStorageProvider;
    private BandsDAO bandsDAO;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_band_edit);

        imageStorageProvider = ImageStorageProvider.getInstance();
        bandsDAO = BandsDAO.getInstance();
        init();
        loadData();
    }

    private void init() {
        mName = findViewById(R.id.bandEditTextName);
        mCity = findViewById(R.id.bandEditTextCity);
        mGenres = findViewById(R.id.bandEditTextGenres);
        mAbout = findViewById(R.id.bandEditTextAbout);
        mVideoLinks = findViewById(R.id.bandEditTextLinks);

        mProfileImage = findViewById(R.id.bandImageButton);

        mProfileImage.setOnClickListener(v -> {
            Intent intent = new Intent();
            intent.setType("image/*");
            intent.setAction(Intent.ACTION_GET_CONTENT);
            resultLauncher.launch(intent);
        });

        resultLauncher = registerForActivityResult(
                new ActivityResultContracts.StartActivityForResult(),
                activityResult -> {
                    try {
                        Uri imageUri =
Objects.requireNonNull(activityResult.getData()).getData();
                        mProfileImage.setImageURI(imageUri);
                        profileImageUri = imageUri;
                    } catch (Exception e) {
                        Log.e("BAND EDIT", "Can't to pick the image");
                    }
                }
        );
    }

    private void loadData() {
        Band band = bandsDAO.readBand(AuthProvider.getInstance().getUid());

        if (band != null) {
            if (!band.getName().isEmpty())
                mName.setText(band.getName());
            if (!band.getCity().isEmpty())
                mCity.setText(band.getCity());
            if (!band.getGenres().isEmpty())
```

```
                mGenres.setText(band.getGenres());
            if (!band.getAbout().isEmpty())
                mAbout.setText(band.getAbout());
            if (!band.getVideoLinks().isEmpty())
                mVideoLinks.setText(band.getVideoLinks());
        }

        Uri profileImage =
imageStorageProvider.downloadImageUri(AuthUID.getUID());
        Glide
                .with(this)
                .load(profileImage)
                .apply(ImageOptions.imageOptions())
                .into(mProfileImage);
    }

    public void save(View view) {
        AlertDialog.Builder builder = new
AlertDialog.Builder(BandEditActivity.this);
        builder
                .setTitle(R.string.text_to_sign_out_title)
                .setMessage(R.string.text_to_save_message)
                .setPositiveButton(R.string.yes, (dialog, which) -> {
                    String name = mName.getText().toString();
                    String city = mCity.getText().toString();
                    String genres = mGenres.getText().toString();
                    String about = mAbout.getText().toString();
                    String videoLinks = mVideoLinks.getText().toString();

                    if (name.isEmpty() || city.isEmpty()) {
                        Toast.makeText(BandEditActivity.this, "Fill all
required fields", Toast.LENGTH_SHORT).show();
                        Log.d("BAND EDIT", "Some fields are empty");
                        return;
                    }

                    Band oldBand =
bandsDAO.readBand(AuthProvider.getInstance().getUid());
                    Band newBand = new Band(
                            oldBand.getBandUID(),
                            oldBand.getEmail(),
                            name,
                            city,
                            genres,
                            about,
                            videoLinks,
                            oldBand.getMembersID());
                    bandsDAO.updateBand(newBand);

                    if (profileImageUri != null)
                        imageStorageProvider.uploadImage(this,
AuthUID.getUID(), profileImageUri);

                    Intent intent = new Intent(BandEditActivity.this,
MainActivity.class);
                    startActivity(intent);
                    finish();
                })
                .setNegativeButton(R.string.no, (dialog, which) -> {})
                .show();
    }

    public void cancel(View view) {
        profileImageUri = null;
```

```
        finish();
    }
}
```

Файл SearchFragment.java

Реалізація функціональної вимоги системи роботи з вакансіями та системи обробки резюме.

```
public class SearchFragment extends Fragment {
    private SearchViewModel searchViewModel;
    private RecyclerView recommendedVacanciesRecyclerView;
    private RecyclerView receivedResumesRecyclerView;
    private ArrayList<Vacancy> recommendedVacanciesList;
    private ArrayList<Resume> receivedResumesList;
    private RecommendedVacanciesRecyclerAdapter recommendedVacanciesAdapter;
    private ReceivedResumesRecyclerAdapter receivedResumesAdapter;
    private TextView mEmptyText, mBandEmptyText;

    private ConstraintLayout candidateLayout;
    private ConstraintLayout bandLayout;



    private FragmentSearchBinding binding;

    public View onCreateView(@NonNull LayoutInflater inflater,
                             ViewGroup container, Bundle savedInstanceState)
{
        searchViewModel = new
ViewModelProvider(this).get(SearchViewModel.class);

        binding = FragmentSearchBinding.inflate(inflater, container, false);
        View root = binding.getRoot();

        init();
        loadData();

        return root;
    }

    @SuppressLint("NotifyDataSetChanged")
    private void init() {
        candidateLayout = binding.candidateSearchLayout;
        bandLayout = binding.bandSearchLayout;
        String userType = searchViewModel.getUserType();

        if (userType.equals(UserType.CANDIDATE.toString())) {
            mEmptyText = binding.searchCandidateEmptyText;
            recommendedVacanciesRecyclerView =
binding.searchCandidateRecyclerView;
            recommendedVacanciesRecyclerView.setLayoutManager(new
LinearLayoutManager(requireActivity()));

            recommendedVacanciesList =
searchViewModel.getRecommendedVacancies(AuthUID.getUID());
            if (recommendedVacanciesList.size() == 0) {
                mEmptyText.setVisibility(View.VISIBLE);

recommendedVacanciesRecyclerView.setVisibility(View.INVISIBLE);
            } else {
                mEmptyText.setVisibility(View.INVISIBLE);
                recommendedVacanciesRecyclerView.setVisibility(View.VISIBLE);
```

```
                recommendedVacanciesAdapter = new
RecommendedVacanciesRecyclerAdapter(requireActivity(),
recommendedVacanciesList);


recommendedVacanciesRecyclerView.setAdapter(recommendedVacanciesAdapter);

searchViewModel.setRecommendedAdapter(recommendedVacanciesAdapter);
            }

            Button candidateHistoryButton =
binding.searchCandidateHistoryButton;
            candidateHistoryButton.setOnClickListener(v -> {
                Intent intent = new Intent(requireActivity(),
ResumeHistoryActivity.class);
                startActivity(intent);
            });

            Button activeResumesButton = binding.searchCandidateSendButton;
            activeResumesButton.setOnClickListener(v -> {
                Intent intent = new Intent(requireActivity(),
ActiveResumeActivity.class);
                startActivity(intent);
            });
        } else if (userType.equals(UserType.BAND.toString())) {
            mBandEmptyText = binding.searchBandEmptyText;
            receivedResumesRecyclerView = binding.searchBandRecyclerView;
            receivedResumesRecyclerView.setLayoutManager(new
LinearLayoutManager(requireActivity()));

            receivedResumesList =
searchViewModel.getReceivedResumesList(AuthUID.getUID());
            if (receivedResumesList.size() == 0) {
                mBandEmptyText.setVisibility(View.VISIBLE);
                receivedResumesRecyclerView.setVisibility(View.INVISIBLE);
            } else {
                mBandEmptyText.setVisibility(View.INVISIBLE);
                receivedResumesRecyclerView.setVisibility(View.VISIBLE);

                receivedResumesAdapter = new
ReceivedResumesRecyclerAdapter(requireActivity(), receivedResumesList);

receivedResumesRecyclerView.setAdapter(receivedResumesAdapter);
                searchViewModel.setReceivedAdapter(receivedResumesAdapter);
            }

            Button createVacancyButton =
binding.searchBandCreateVacancyButton;
            createVacancyButton.setOnClickListener(v -> {
                Intent intent = new Intent(requireActivity(),
CreateVacancyActivity.class);
                startActivity(intent);
            });

            Button openBandVacancyButton = binding.searchBandVacanciesButton;
            openBandVacancyButton.setOnClickListener(v -> {
                Intent intent = new Intent(requireActivity(),
BandVacanciesActivity.class);
                startActivity(intent);
            });
        }
    }
    private void loadData() {
        String userType = searchViewModel.getUserType();
```

```
        if (userType.equals(UserType.CANDIDATE.toString())) {
            candidateLayout.setVisibility(View.VISIBLE);
            bandLayout.setVisibility(View.INVISIBLE);

            if (recommendedVacanciesList.size() == 0) {
                mEmptyText.setVisibility(View.VISIBLE);

recommendedVacanciesRecyclerView.setVisibility(View.INVISIBLE);
            } else {
                mEmptyText.setVisibility(View.INVISIBLE);
                recommendedVacanciesRecyclerView.setVisibility(View.VISIBLE);
            }
        } else if (userType.equals(UserType.BAND.toString())) {
            candidateLayout.setVisibility(View.INVISIBLE);
            bandLayout.setVisibility(View.VISIBLE);

            if (receivedResumesList.size() == 0) {
                mBandEmptyText.setVisibility(View.VISIBLE);
                receivedResumesRecyclerView.setVisibility(View.INVISIBLE);
            } else {
                mBandEmptyText.setVisibility(View.INVISIBLE);
                receivedResumesRecyclerView.setVisibility(View.VISIBLE);
            }
        }
    }

    @Override
    public void onDestroyView() {
        super.onDestroyView();
        binding = null;
    }

    @Override
    public void onResume() {
        super.onResume();

        loadData();
    }
}
```

Файл BandVacanciesActivity.java

Реалізація функціональної вимоги отримання списку вакансій для власника гурту.

```
public class BandVacanciesActivity extends AppCompatActivity {
    private RecyclerView bandVacanciesRecyclerView;
    private ArrayList<Vacancy> bandVacanciesList;
    private BandVacanciesRecyclerAdapter bandVacanciesRecyclerAdapter;
    private TextView mEmptyText;
    private VacanciesDAO vacanciesDAO;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_vacancy_band);

        init();
    }

    private void init() {
        vacanciesDAO = VacanciesDAO.getInstance();
        mEmptyText = findViewById(R.id.bandVacanciesEmptyText);
```

```
        bandVacanciesRecyclerView =
findViewById(R.id.bandVacanciesRecyclerView);
        bandVacanciesRecyclerView.setLayoutManager(new
LinearLayoutManager(BandVacanciesActivity.this));

        bandVacanciesList = vacanciesDAO.getBandVacancies(AuthUID.getUID());
        if (bandVacanciesList.size() == 0) {
            mEmptyText.setVisibility(View.VISIBLE);
            bandVacanciesRecyclerView.setVisibility(View.INVISIBLE);
        } else {
            mEmptyText.setVisibility(View.INVISIBLE);
            bandVacanciesRecyclerView.setVisibility(View.VISIBLE);

            bandVacanciesRecyclerAdapter = new
BandVacanciesRecyclerAdapter(BandVacanciesActivity.this, bandVacanciesList);

bandVacanciesRecyclerView.setAdapter(bandVacanciesRecyclerAdapter);

vacanciesDAO.setBandVacanciesRecyclerAdapter(bandVacanciesRecyclerAdapter);
        }
    }
}
```

Файл VacancyInfoActivity.java

Реалізація функціональної вимоги перегляду вакансії, видалення вакансії.

```
public class VacancyInfoActivity extends AppCompatActivity {
    private BandsDAO bandsDAO;
    private Vacancy vacancy;

    private TextView mTitle, mBand, mGenres, mSalary, mCity;
    private TextView mText, mAbout, mLinks, mDatetime;
    private ImageView mBandImage;

    private ImageStorageProvider imageStorageProvider;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_vacancy_info);

        bandsDAO = BandsDAO.getInstance();
        vacancy = (Vacancy) getIntent().getSerializableExtra("vacancy");

        imageStorageProvider = ImageStorageProvider.getInstance();

        init();
        loadData();
    }

    private void init() {
        mTitle = findViewById(R.id.vacancyInfoTitle);
        mBand = findViewById(R.id.vacancyInfoBand);
        mGenres = findViewById(R.id.vacancyInfoGenres);
        mSalary = findViewById(R.id.vacancyInfoSalary);
        mCity = findViewById(R.id.vacancyInfoCity);
        mText = findViewById(R.id.vacancyInfoText);
        mAbout = findViewById(R.id.vacancyInfoAboutBand);
        mLinks = findViewById(R.id.vacancyInfoBandLinks);
        mDatetime = findViewById(R.id.vacancyInfoDatetime);

        mBandImage = findViewById(R.id.bandImageView);
```

```
        Button mSend = findViewById(R.id.sendResumeButton);
        Button mDelete = findViewById(R.id.deleteVacancyButton);
        if
(UsersDAO.getInstance().readUser(AuthUID.getUID()).getType().equals(String.va
lueOf(UserType.CANDIDATE))) {
            mSend.setVisibility(View.VISIBLE);
            mDelete.setVisibility(View.INVISIBLE);
        } else {
            mSend.setVisibility(View.INVISIBLE);
            mDelete.setVisibility(View.VISIBLE);
        }
    }

    private void loadData() {
        Band band = bandsDAO.readBand(vacancy.getBandUID());

        mTitle.setText(vacancy.getRole());
        mBand.setText(band.getName());
        mGenres.setText(band.getGenres());
        mSalary.setText(vacancy.getSalary());
        mCity.setText(band.getCity());
        mText.setText(vacancy.getText());
        mAbout.setText(band.getAbout());
        mLinks.setText(band.getVideoLinks());
        mDatetime.setText(vacancy.getDatetime());

        Uri imageUri =
imageStorageProvider.downloadImageUri(vacancy.getBandUID());
        Glide
                .with(this)
                .load(imageUri)
                .apply(ImageOptions.imageOptions())
                .into(mBandImage);
    }

    public void back(View view) {
        finish();
    }

    public void sendResume(View view) {
        Intent intent = new Intent(VacancyInfoActivity.this,
SendResumeActivity.class);
        intent.putExtra("vacancy", vacancy);
        startActivity(intent);
    }

    public void deleteVacancy(View view) {
        AlertDialog.Builder builder = new
AlertDialog.Builder(VacancyInfoActivity.this);
        builder
                .setTitle(R.string.text_to_sign_out_title)
                .setMessage(R.string.text_delete_vacancy_warning)
                .setPositiveButton(R.string.yes, (dialog, which) -> {
                    VacanciesDAO vacanciesDAO = VacanciesDAO.getInstance();
                    vacanciesDAO.deleteVacancy(vacancy.getVacancyUID());
                    Toast.makeText(VacancyInfoActivity.this, "Vacancy
deleted", Toast.LENGTH_SHORT).show();
                    ResumesDAO resumesDAO = ResumesDAO.getInstance();

resumesDAO.markAllResumesDeclinedForVacancy(vacancy.getVacancyUID());
                    finish();
                })
                .setNegativeButton(R.string.no, (dialog, which) -> {})
                .show();
```

```
    }
}
```

Файл CreateVacancyActivity.java

Реалізація функціональної вимоги створення вакансії.

```
public class CreateVacancyActivity extends AppCompatActivity {
    VacanciesDAO vacanciesDAO;
    private Spinner moneySpinner;
    private ArrayAdapter<String> moneyAdapter;

    private EditText mRole, mText, mSalary;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_vacancy_create);

        init();
    }

    private void init() {
        vacanciesDAO = VacanciesDAO.getInstance();

        moneySpinner = findViewById(R.id.moneySpinner);
        String[] moneys = {
                getResources().getString(R.string.dollars),
                getResources().getString(R.string.hryvnya)
        };
        moneyAdapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_dropdown_item, moneys);
        moneySpinner.setAdapter(moneyAdapter);

        mRole = findViewById(R.id.vacancyRoleEditText);
        mText = findViewById(R.id.vacancyTextEditText);
        mSalary = findViewById(R.id.vacancySalaryEditText);
    }

    public void createVacancy(View view) {
        String role = mRole.getText().toString();
        String text = mText.getText().toString();
        String salary = mSalary.getText().toString();
        String money =
moneyAdapter.getItem(moneySpinner.getSelectedItemPosition());

        if (role.isEmpty() || text.isEmpty()) {
            Toast.makeText(CreateVacancyActivity.this, "Please, fill role and
vacancy description", Toast.LENGTH_SHORT).show();
        } else {
            role = role.substring(0,1).toUpperCase() +
role.substring(1).toLowerCase();
            text = text.substring(0,1).toUpperCase() +
text.substring(1).toLowerCase();

            String salaryText = "\uD83D\uDCB0 -";
            if (!salary.isEmpty())
                salaryText = "\uD83D\uDCB0 " + salary + " " + money;
            LocalDateTime now = LocalDateTime.now();
            DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm");
            String datetime = now.format(formatter);

            Vacancy vacancy = new Vacancy(
```

```
                    "",
                    AuthUID.getUID(),
                    role,
                    text,
                    salaryText,
                    datetime
            );
            vacanciesDAO.createVacancy(vacancy);
            Toast.makeText(CreateVacancyActivity.this, "Vacancy created",
Toast.LENGTH_SHORT).show();
            finish();
        }
    }

    public void cancel(View view) {
        finish();
    }
}
```

Файл SendResumeActivity.java

Реалізація функціональної вимоги надсилання резюме на вакансію.

```
public class SendResumeActivity extends AppCompatActivity {
    private ResumesDAO resumesDAO;
    private CandidatesDAO candidatesDAO;
    private NotificationsDAO notificationsDAO;
    private Vacancy vacancy;
    private TextView mText;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_resume_send);

        resumesDAO = ResumesDAO.getInstance();
        candidatesDAO = CandidatesDAO.getInstance();
        notificationsDAO = NotificationsDAO.getInstance();
        vacancy = (Vacancy) getIntent().getSerializableExtra("vacancy");

        mText = findViewById(R.id.resumeTextEditText);
    }

    public void sendResume(View view) {
        String resumeText = mText.getText().toString();
        Candidate candidate = candidatesDAO.readCandidate(AuthUID.getUID());

        if (candidate.getRole().isEmpty()
                || candidate.getPreferredGenres().isEmpty()
                || candidate.getExperience().isEmpty()) {
            AlertDialog.Builder builder = new
AlertDialog.Builder(SendResumeActivity.this);
            builder
                    .setTitle(R.string.text_to_sign_out_title)
                    .setMessage(R.string.text_send_empty_resume_message)
                    .setPositiveButton(R.string.yes, (dialog, which) -> {
                        send(resumeText, candidate.getName() + " " +
candidate.getSurname());
                    })
                    .setNegativeButton(R.string.no, (dialog, which) -> {})
                    .show();
        } else {
            send(resumeText, candidate.getName() + " " +
candidate.getSurname());
```

```
        }

    }

    private void send(String resumeText, String candidateName) {
        LocalDateTime now = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm");
        String datetime = now.format(formatter);

        Resume resume = new Resume(
                "",
                AuthUID.getUID(),
                vacancy.getVacancyUID(),
                resumeText,
                datetime,
                ResumeStatus.NEW.toString()
        );
        resumesDAO.createResume(resume);

        Notification notification = new Notification(
                "",
                vacancy.getBandUID(),

getResources().getString(R.string.text_notification_new_resume_title),

getResources().getString(R.string.text_notification_new_resume) + " " +
candidateName,
                datetime,
                NotificationStatus.NEW.toString()
        );
        notificationsDAO.createNotification(notification);

        Toast.makeText(SendResumeActivity.this, "Resume sent",
Toast.LENGTH_SHORT).show();
        Intent intent = new Intent(SendResumeActivity.this,
MainActivity.class);
        startActivity(intent);
        finish();
    }

    public void cancel(View view) {
        finish();
    }
}
```

Файл ResumeInfoActivity.java

Реалізація функціональної вимоги перегляду резюме, відповіді на резюме, надсилання повідомлень кандидату із результатом.

```
public class ResumeInfoActivity extends AppCompatActivity {
    private CandidatesDAO candidatesDAO;
    private ResumesDAO resumesDAO;
    private NotificationsDAO notificationsDAO;
    private ChatsDAO chatsDAO;
    private MessagesDAO messagesDAO;
    private Resume resume;

    private TextView mName, mSurname, mRole, mExperience;
    private TextView mBirthday, mCity, mText;
    private TextView mAbout, mGenres, mLinks, mDatetime;
    private ImageView mResumeImage;
```

```
    private ImageStorageProvider imageStorageProvider;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_resume_info);

        candidatesDAO = CandidatesDAO.getInstance();
        resumesDAO = ResumesDAO.getInstance();
        notificationsDAO = NotificationsDAO.getInstance();
        chatsDAO = ChatsDAO.getInstance();
        messagesDAO = MessagesDAO.getInstance();
        resume = (Resume) getIntent().getSerializableExtra("resume");

        imageStorageProvider = ImageStorageProvider.getInstance();

        init();
        loadData();
    }

    private void init() {
        mName = findViewById(R.id.resumeInfoCandidateName);
        mSurname = findViewById(R.id.resumeInfoCandidateSurname);
        mRole = findViewById(R.id.resumeInfoRole);
        mExperience = findViewById(R.id.resumeInfoExperience);
        mBirthday = findViewById(R.id.resumeInfoBirthday);
        mCity = findViewById(R.id.resumeInfoCity);
        mText = findViewById(R.id.resumeInfoText);
        mAbout = findViewById(R.id.resumeInfoAbout);
        mGenres = findViewById(R.id.resumeInfoGenres);
        mLinks = findViewById(R.id.resumeInfoLinks);
        mDatetime = findViewById(R.id.resumeInfoDatetime);

        mResumeImage = findViewById(R.id.candidateImageView);
    }

    private void loadData() {
        Candidate candidate =
candidatesDAO.readCandidate(resume.getCandidateUID());

        mName.setText(candidate.getName());
        mSurname.setText(candidate.getSurname());
        mRole.setText(candidate.getRole());
        mExperience.setText(candidate.getExperience());
        mBirthday.setText(candidate.getBirthday());
        mCity.setText(candidate.getCity());
        mText.setText(resume.getText());
        mAbout.setText(candidate.getAbout());
        mGenres.setText(candidate.getPreferredGenres());
        mLinks.setText(candidate.getVideoLinks());
        mDatetime.setText(resume.getDatetime());

        Uri imageUti =
imageStorageProvider.downloadImageUri(resume.getCandidateUID());
        Glide
                .with(this)
                .load(imageUti)
                .apply(ImageOptions.imageOptions())
                .into(mResumeImage);
    }

    public void acceptResume(View view) {
        resume.setStatus(ResumeStatus.ACCEPTED.toString());
        resumesDAO.updateResume(resume);
```

```
        Toast.makeText(ResumeInfoActivity.this, "Resume accepted",
Toast.LENGTH_SHORT).show();

        LocalDateTime now = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm");
        String datetime = now.format(formatter);

        Notification notification = new Notification(
                "",
                resume.getCandidateUID(),

getResources().getString(R.string.text_notification_accept_resume_title),
                BandsDAO.getInstance().readBand(AuthUID.getUID()).getName() +
" " + getResources().getString(R.string.text_notification_accept_resume),
                datetime,
                NotificationStatus.NEW.toString()
        );
        notificationsDAO.createNotification(notification);

        Chat chatFind = chatsDAO.find(resume.getCandidateUID(),
AuthUID.getUID());
        if (chatFind != null) {
            Message message = new Message(
                    "",
                    chatFind.getChatUID(),
                    SenderType.BAND.toString(),
                    getResources().getString(R.string.text_hello_message),
                    datetime,
                    MessageStatus.SENT.toString()
            );
            messagesDAO.createMessage(message);
        } else {
            Chat chat = new Chat(
                    "",
                    resume.getCandidateUID(),
                    AuthUID.getUID()
            );
            chatsDAO.createChat(chat);

            Message message = new Message(
                    "",
                    chat.getChatUID(),
                    SenderType.BAND.toString(),
                    getResources().getString(R.string.text_hello_message),
                    datetime,
                    MessageStatus.SENT.toString()
            );
            messagesDAO.createMessage(message);
        }

        finish();
    }

    public void declineResume(View view) {
        resume.setStatus(ResumeStatus.DECLINED.toString());
        resumesDAO.updateResume(resume);

        Toast.makeText(ResumeInfoActivity.this, "Resume declined",
Toast.LENGTH_SHORT).show();

        LocalDateTime now = LocalDateTime.now();
```

```
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm");
        String datetime = now.format(formatter);

        Notification notification = new Notification(
                "",
                resume.getCandidateUID(),

getResources().getString(R.string.text_notification_decline_resume_title),
                BandsDAO.getInstance().readBand(AuthUID.getUID()).getName() +
" " + getResources().getString(R.string.text_notification_decline_resume),
                datetime,
                NotificationStatus.NEW.toString()
        );
        notificationsDAO.createNotification(notification);

        finish();
    }
}
```

Файл ResumeHistoryActivity.java

Реалізація функціональної вимоги перегляду списку надісланих резюме.

```
public class ResumeHistoryActivity extends AppCompatActivity {
    private RecyclerView resumeHistoryRecyclerView;
    private ArrayList<Resume> resumeHistoryList;
    private ResumeHistoryRecyclerAdapter resumeHistoryRecyclerAdapter;
    private TextView mEmptyText;
    private ResumesDAO resumesDAO;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_resume_history);

        init();
    }

    private void init() {
        resumesDAO = ResumesDAO.getInstance();
        mEmptyText = findViewById(R.id.resumeHistoryEmptyText);
        resumeHistoryRecyclerView =
findViewById(R.id.resumeHistoryRecyclerView);
        resumeHistoryRecyclerView.setLayoutManager(new
LinearLayoutManager(ResumeHistoryActivity.this));

        resumeHistoryList = resumesDAO.getResumeHistory(AuthUID.getUID());
        if (resumeHistoryList.size() == 0) {
            mEmptyText.setVisibility(View.VISIBLE);
            resumeHistoryRecyclerView.setVisibility(View.INVISIBLE);
        } else {
            mEmptyText.setVisibility(View.INVISIBLE);
            resumeHistoryRecyclerView.setVisibility(View.VISIBLE);

            resumeHistoryRecyclerAdapter = new
ResumeHistoryRecyclerAdapter(ResumeHistoryActivity.this, resumeHistoryList);

resumeHistoryRecyclerView.setAdapter(resumeHistoryRecyclerAdapter);

resumesDAO.setResumeHistoryRecyclerAdapter(resumeHistoryRecyclerAdapter);
        }
    }
}
```

Файл ActiveResumeActivity.java

Реалізація функціональної вимоги перегляду списку надісланих резюме.

```
public class ActiveResumeActivity extends AppCompatActivity {
    private RecyclerView activeResumeRecyclerView;
    private ArrayList<Resume> activeResumeList;
    private ActiveResumeRecyclerAdapter activeResumeRecyclerAdapter;;
    private TextView mEmptyText;
    private ResumesDAO resumesDAO;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_resume_active);

        init();
    }

    private void init() {
        resumesDAO = ResumesDAO.getInstance();
        mEmptyText = findViewById(R.id.activeResumeEmptyText);
        activeResumeRecyclerView =
findViewById(R.id.activeResumesRecyclerView);
        activeResumeRecyclerView.setLayoutManager(new
LinearLayoutManager(ActiveResumeActivity.this));

        activeResumeList = resumesDAO.getActiveResumes(AuthUID.getUID());
        if (activeResumeList.size() == 0) {
            mEmptyText.setVisibility(View.VISIBLE);
            activeResumeRecyclerView.setVisibility(View.INVISIBLE);
        } else {
            mEmptyText.setVisibility(View.INVISIBLE);
            activeResumeRecyclerView.setVisibility(View.VISIBLE);

            activeResumeRecyclerAdapter = new
ActiveResumeRecyclerAdapter(ActiveResumeActivity.this, activeResumeList);
            activeResumeRecyclerView.setAdapter(activeResumeRecyclerAdapter);

resumesDAO.setActiveResumesRecyclerAdapter(activeResumeRecyclerAdapter);
        }
    }
}
```

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПОШУКУ МУЗИЧНОГО ГУРТУ

Програма та методика тестування

КПІ.ІХ-0222.045440.04.51

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Валентин СЕРЕДЮК

Київ – 2024

ЗМІСТ

1 ОБ’ЄКТ ВИПРОБУВАНЬ

Об’єктом випробування є мобільний застосунок для пошуку музичного гурту, який розроблений під операційну систему Android.

2 МЕТА ТЕСТУВАННЯ

Метою тестування є наступне:

– перевірка правильності роботи програмного забезпечення відповідно до функціональних вимог;

– перевірка збереження даних;

– перевірка сумісності застосунку з різними версіями операційної системи Android;

– знаходження проблем, помилок і недоліків з метою їх усунення;

– перевірка зручності графічного інтерфейсу.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються такі методи:

– мануальне тестування – тестування без використання автоматизації, тест-кейси пише особа, що тестує програмне забезпечення;

– динамічне тестування – застосовується в процесі виконання програми. Коректність програмного засобу перевіряється на певній кількості тестів. При прогоні кожного з них збираються та аналізуються дані про проблеми та помилки в роботі програми;

– функціональне тестування – полягає у перевірці відповідності реальної поведінки програмного забезпечення очікуваній;

– системне тестування – перевіряється усе програмне забезпечення в цілому;

– тестування «чорної скриньки» – об’єктом тестування тут є функції присутні у програмі. Перевіряється коректність вихідних даних при заданих вхідних;

– тестування «сірої скриньки» – об’єктом тестування тут є деякі особливості внутрішньої поведінки програми. Перевіряється коректність вихідних даних при заданих вхідних, застосовується для тестування окремих алгоритмів (функцій).

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується мануально з метою знаходження помилок та недоліків як у функціональній частині програмного забезпечення так і в зручності користування. Для того, щоб перевірити працездатність та відмовостійкість застосунку, необхідно провести наступні тестування:

– динамічне тестування на відповідність функціональним вимогам;

– тестування на мобільних пристроях з різною роздільною здатністю екрану;

– тестування на мобільних пристроях з різною версією операційної системи Android від 9 до 14;

– тестування на виведення повідомлень про помилку, коли це необхідно;

– тестування валідності перевірки паролів;

– тестування працездатності програми у випадку відсутності з’єднання до мережі;

– тестування інтерфейсу користувача;

– тестування зміни мови для країн Україна та всіх інших.

– тестування зручності використання;

– тестування при одночасному використанні застосунку декількома користувачами.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПОШУКУ МУЗИЧНОГО ГУРТУ

Керівництво користувача

КПІ.ІХ-0222.045440.05.34

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Валентин СЕРЕДЮК

Київ – 2024

ЗМІСТ

1 ПРИЗНАЧЕННЯ ПРОГРАМИ

«Bander» - це мобільний застосунок для пошуку музичного гурту. Його головною функцією є забезпечення знаходження нових знайомств у музичній сфері, за рахунок обробки вакансій та резюме, а також системи чатів для обміну повідомлень.

2 ПІДГОТОВКА ДО РОБОТИ З ПРОГРАМНИМ ЗАБЕЗПЕЧЕННЯМ

2.1 Системні вимоги для коректної роботи

Для успішної роботи даного застосунку необхідне виконання наступних вимог:

– наявність мобільного пристрою з операційною системою на базі Android з версією 9 або вище;

– діагональ екрану від 4 дюймів;

– частота процесору від 1 ГГц;

– об’єм ОЗП від 3 Гб

– наявність доступу до Інтернету зі швидкістю від 8 мегабіт;

– для встановлення додатку на мобільному пристрої повинно бути не менше 28 МБ вільної пам’яті.

Рекомендовані характеристики мобільного пристрою:

– операційна система на базі Android з версією 9 або вище;

– діагональ екрану 6 дюймів;

– частота процесору 3 ГГц;

– об’єм ОЗП 6 Гб

– наявність доступу до Інтернету зі швидкістю від 100 мегабіт;

– для встановлення додатку на мобільному пристрої повинно бути не менше 28 МБ вільної пам’яті.

2.2 Завантаження застосунку

На даний момент одним із варіантів встановлення є встановлення власноруч застосунку, використовуючи відповідний apk-файл із наданого в тексті програми GitHub репозиторію. Для цього спершу необхідно завантажити його на мобільний пристрій, а потім, використовуючи який-небудь інсталятор, виконати встановлення даного додатку.

Також, для більш зручного завантаження, можна скористатись застосунком Amazon Appstore, у якому завантажити розроблений застосунок через маркетплейс.

2.3 Перевірка коректної роботи

По завершенню встановлення додатка на робочому столі мобільного пристрою повинна відобразитись іконка даного застосунку. У разі, якщо дана іконка не з’явилась, то встановлення відбулось не успішно. Інакше користувач має змогу запустити додаток, клацнувши на його іконку. Після натискання повинна відобразитись початкова сторінка застосунку.

3 ВИКОНАННЯ ПРОГРАМИ

При першому запуску програмного застосунку користувачу буде відображено сторінку авторизації (рис. 3.1).

Рисунок 3.1 – Початкова сторінка додатку (авторизації)

Користувач має змогу ввести електронну адресу та відповідний їй пароль, щоб увійти у свій профіль. У випадку введення не зареєстрованих або неправильних даних, користувачу відобразиться відповідне повідомлення. Також є можливість зареєструватись як кандидат та як власник гурту, кнопки яких переносять користувача на відповідні сторінки (рис. 3.2). Якщо усе введено правильно, то користувач переходить на головну сторінку застосунку (рис. 3.3)

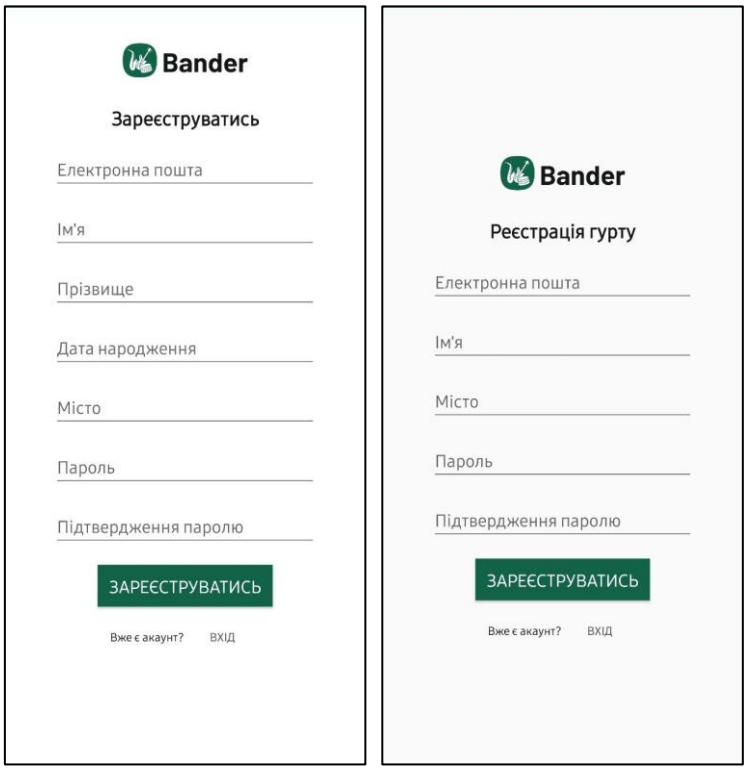


Рисунок 3.2 – Сторінки реєстрацій для кандидата та власника гурту

Користувач має змогу створити потрібний йому обліковий запис із необхідними початковими даними. Користувач може повернути на сторінку авторизації, за потреби.

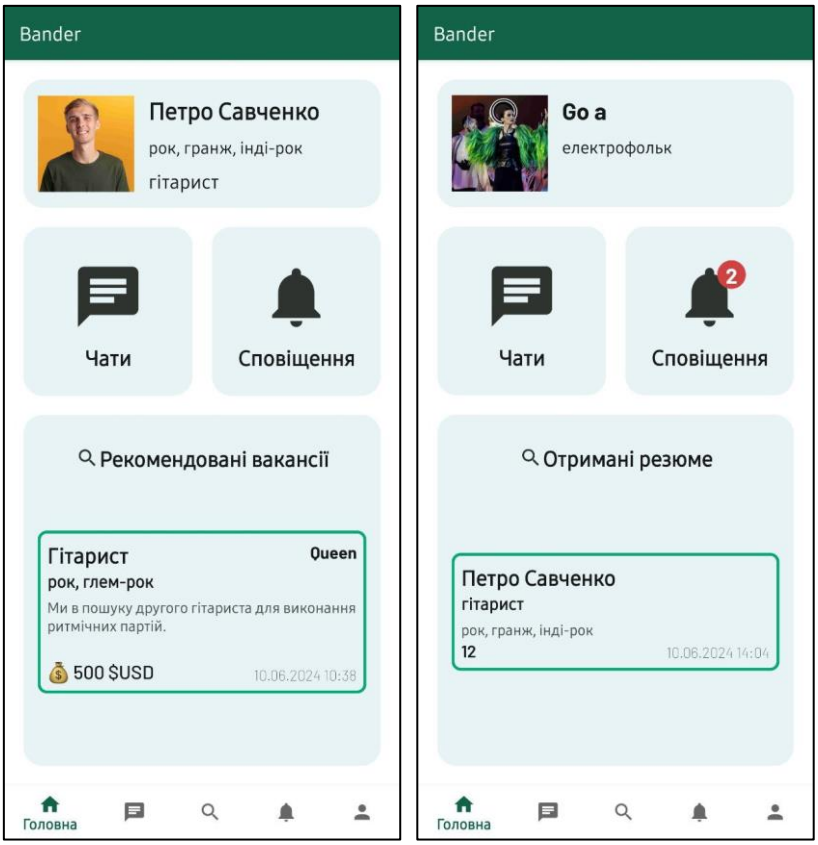


Рисунок 3.3 – Головна сторінка для кандидата та власника гурту

Користувач має змогу здійснити навігацію по пунктам меню, або ж по плиткам на головній сторінці, такі як профіль (рис. 3.4), чати (рис. 3.6), сповіщення (рис. 3.11) та пошук (рис. 3.12). На відповідних плитках чатів та сповіщень будуть відображатись кількість нових повідомлень та сповіщень. У залежності від типу облікового запису, відображається різна плитка пошуку як рекомендовані вакансії, так і отримані резюме.

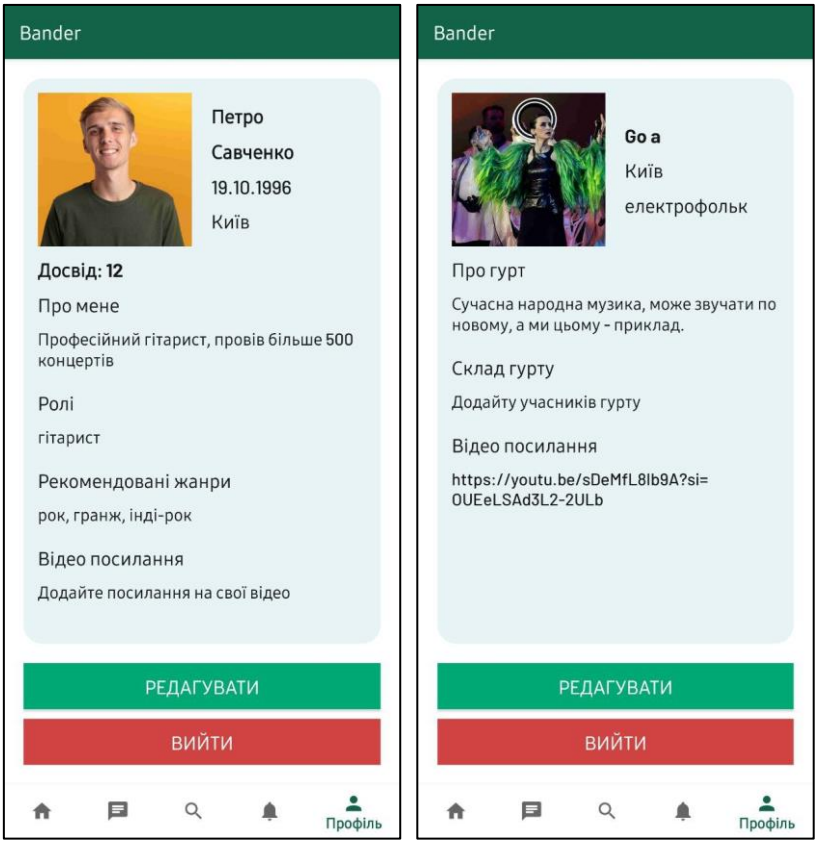


Рисунок 3.4 – Сторінки профілей кандидата та власника гурту

Користувач може переглянути інформацію про свій обліковий запис. За бажання, він може відредагувати свої дані (рис. 3.5) або ж просто вийти із свого облікового запису.

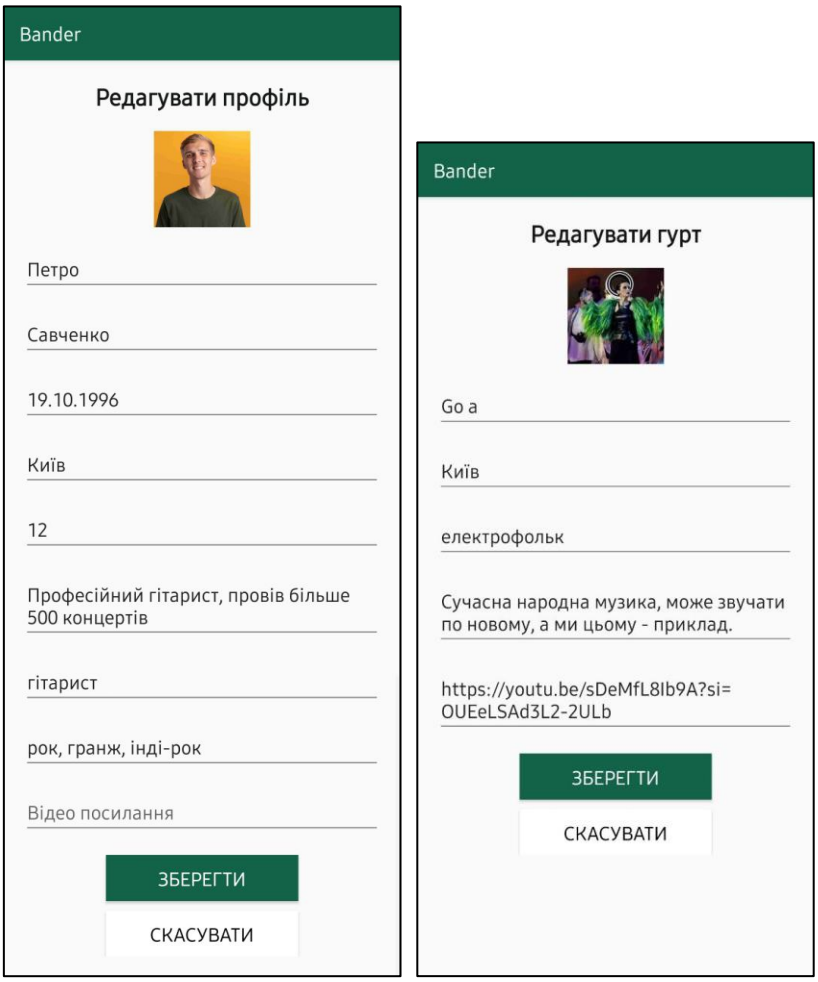


Рисунок 3.5 – Сторінки редагування профілів

Користувач може змінити фото облікового запису, а також більше кількість інформації, окрім електронної пошти. Також, користувач може зберегти зміни, які він ввів, або ж скасувати їх.

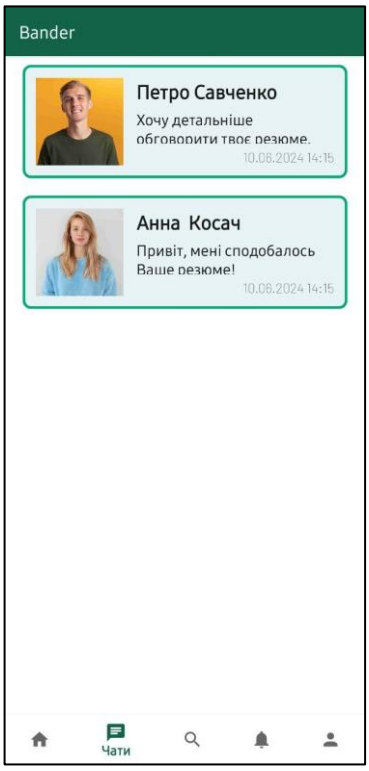


Рисунок 3.6 – Сторінка списку чатів

Користувач має змогу переглянути усі його чати з іншими користувачами та одразу дізнатись час останнього повідомлення та кількість нових повідомлень у конкретному чату. При необхідності, користувач може перейти у час із конкретним користувачем (рис. 3.7).

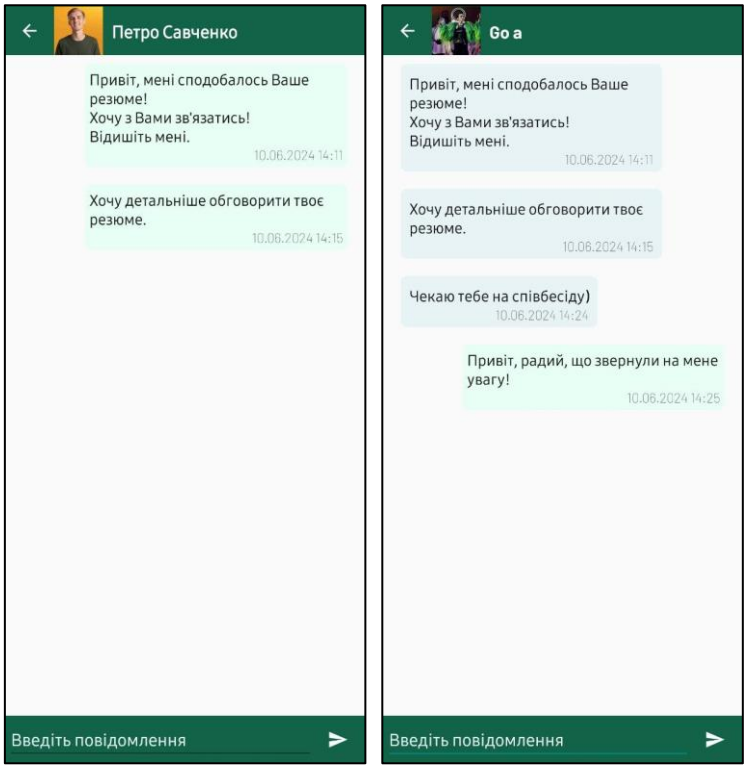
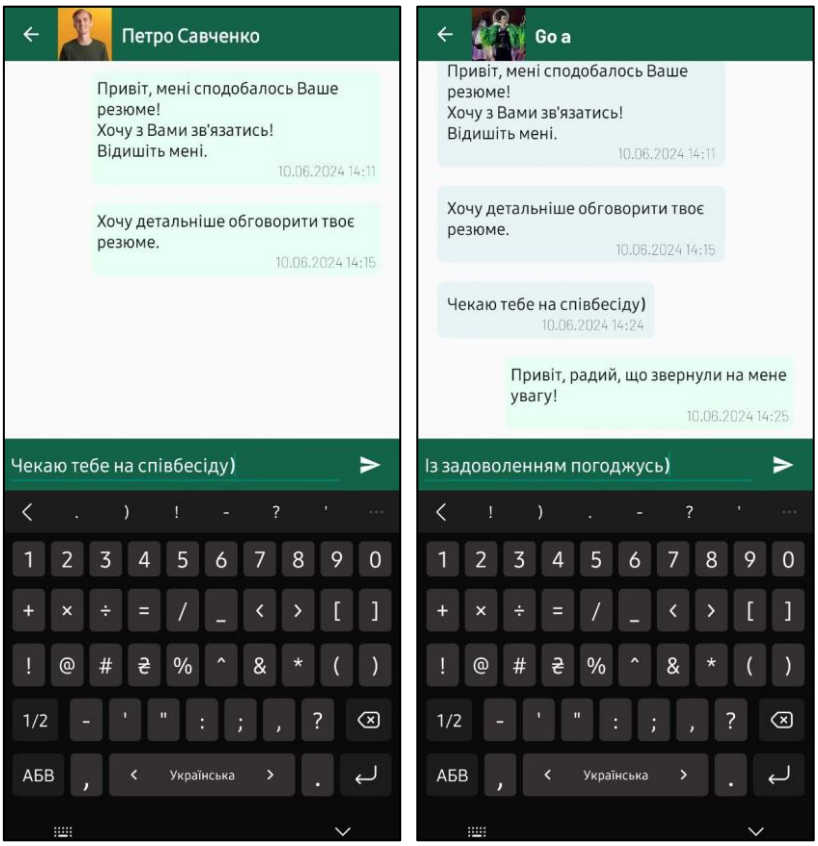


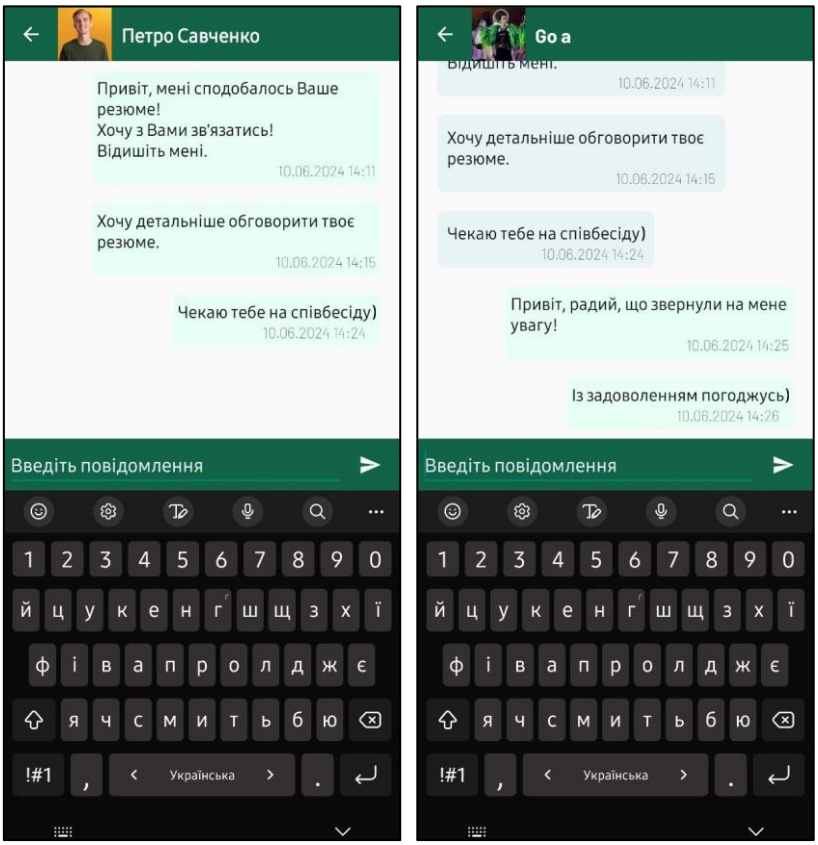
Рисунок 3.7 – Відкриті чати із іншими користувачами

Користувач може переглядати історію повідомлень у порядку їх надсилання. Також він може повернутись назад, або ж почати вводити нове повідомлення (рис. 3.8).



Рисуонк 3.8 – Введення тексту для повідомлень

Якщо користувач завершив вводити повідомлення і бажає його надіслати, то він тисне кнопку надсилання праворуч від введеного тексту, після чого повідомлення додається до історії чату (рис. 3.9).



Рисуонк 3.9 – Додавання нового повідомлення у чатування

Користувач може покинути чат у будь-який момент. Нові ж повідомлення завжди позначаються фіолетовим фоном (рис. 3.10).

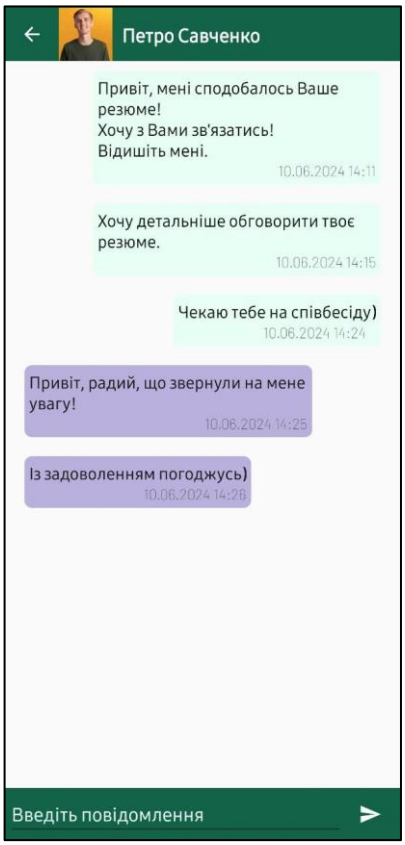


Рисунок 3.10 – Відображення нових повідомлень

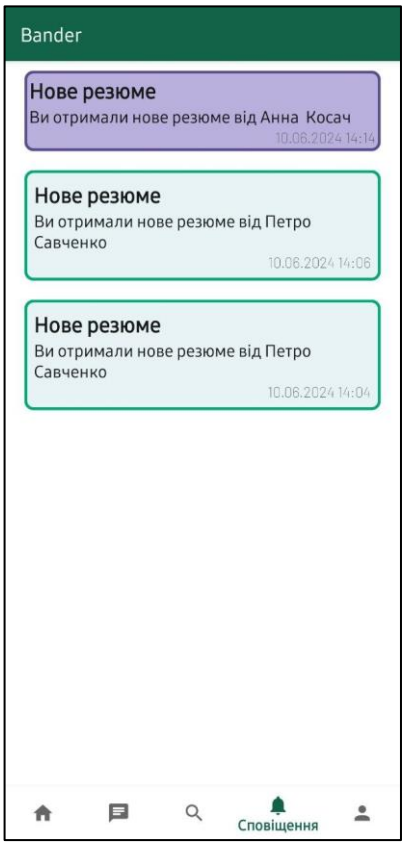


Рисунок 3.11 – Сторінка сповіщень

Користувач може переглянути усі сповіщення, які прийшли йому. Нові сповіщення відображаються фіолетовим кольором.

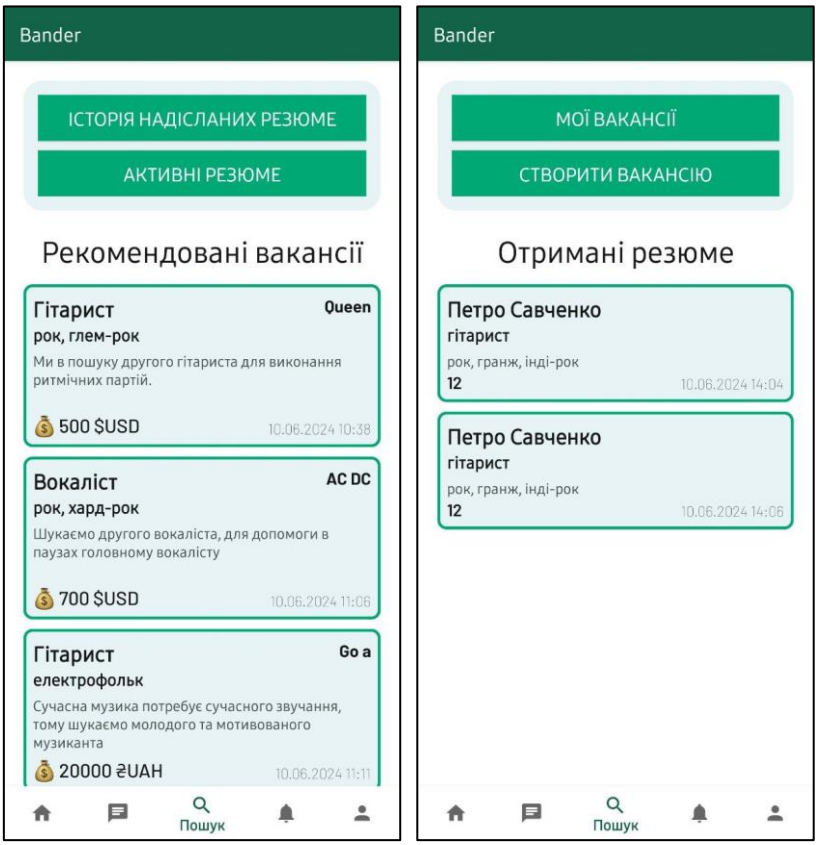


Рисунок 3.12 – Сторінки пошуку для кандидата та власника гурту

Користувач в залежності від типу має різні сторінки пошуку. Кандидат має змогу переглянути результати відповідей на його резюме і резюме, які ще очікують на відповідь (рис. 3.13) Також кандидат може переглядати рекомендовані йому вакансії, відкриваючи необхідні йому (рис. 3.14).

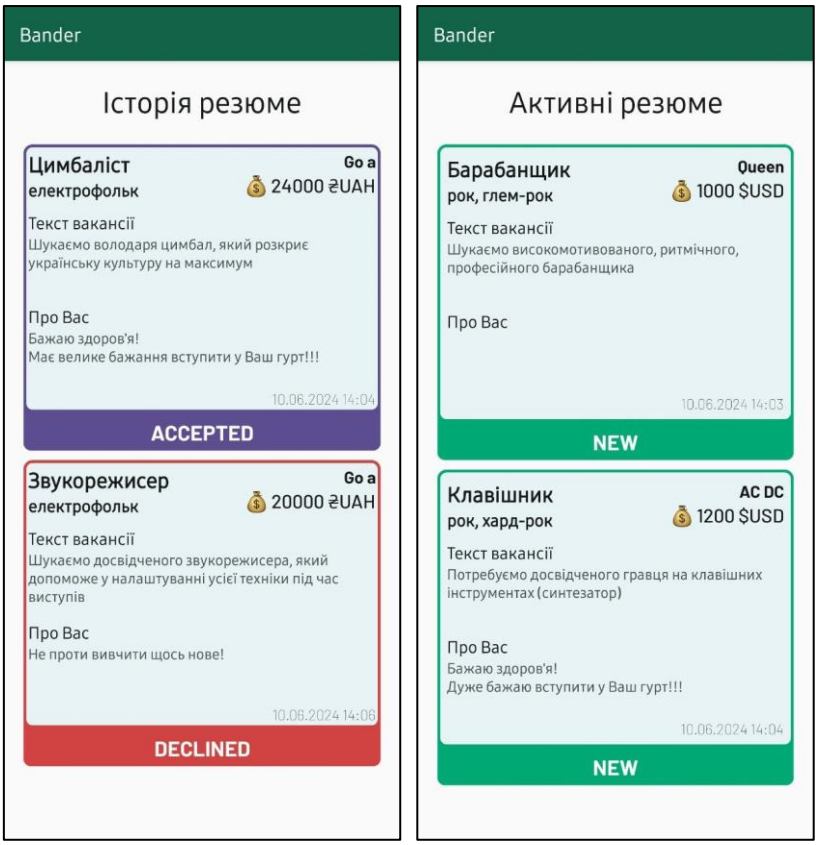


Рисунок 3.13 – Сторінки перегляду результатів резюме та ще активних резюме

Користувач може детально дізнатись про резюме, які він відправляв, а також їх статус на даний момент часу.

Рисунок 3.14 – Сторінка відкриття певної вакансії з боку кандидата

Користувач може переглянути детальну інформацію про гурт та вакансію у даний гурт. Також перейти за посилання на відеоматеріал з прикладом виступів за наявності. Основною ж можливість користувача є відреагувати на вакансію, а саме надіслати на неї резюме (рис. 3.15).

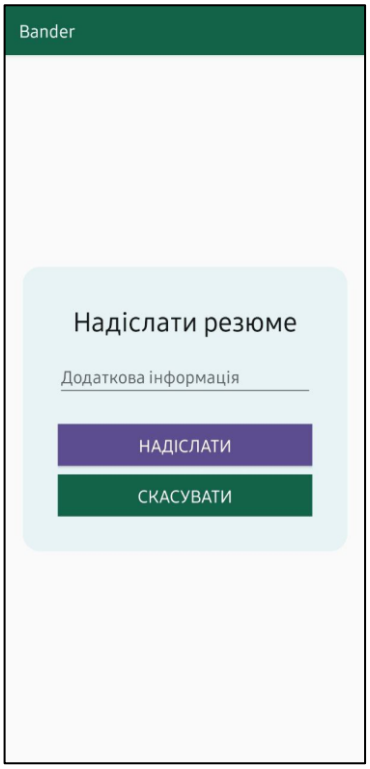


Рисунок 3.15 – Сторінка надсилання резюме

Користувач може ввести додаткову інформацію, яку він бажає розповісти власнику гурту вакансії, та надіслати йому своє резюме. За потреби, користувач може в будь-який момент скасувати надсилання.

Повертаючись до сторінки пошуку, розглянемо можливості власника гурту. Власник може переглянути усі виставлені ним вакансії (рис. 3.16), створити нову вакансію у свій гурт (рис. 3.18) та переглянути резюме, що надійшли йому (рис. 3.19).

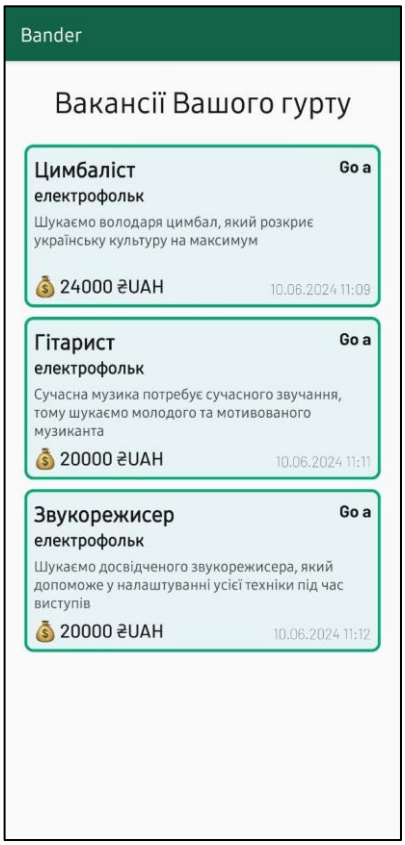


Рисунок 3.16 – Сторінка перегляду усіх вакансій гурту

Користувач може переглянути усі його вакансії та перейти на необхідну йому вакансію для детальнішої інформації (рис. 3.17).

Рисунок 3.17 – Сторінка перегляду детальної інформації вакансії для власника гурту

Користувач може переглянути усю інформацію про вакансію, а також видалити її, якщо вона йому більше не потрібна. За потреби, користувач може просто повернутись назад.

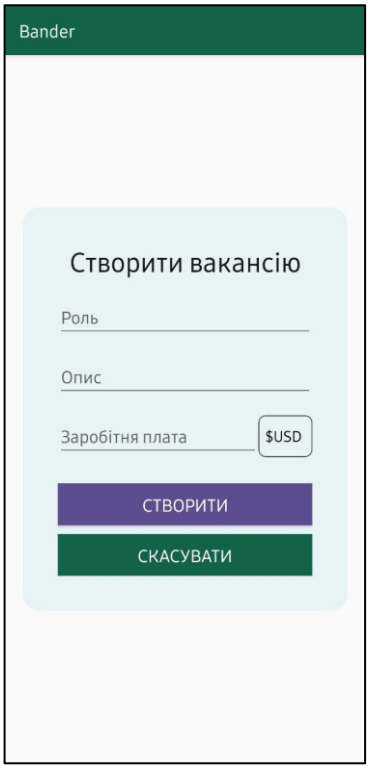


Рисунок 3.18 – Сторінка створення вакансії

Користувач має внести дані у відповідні поля. Поле заробітної плати не є обов’язковим, але при вказанні суми, потрібно обрати валюту (USD по стандарту). Після натиснення створення, вакансія автоматично створиться. Також користувач може просто повернутись назад.

Рисунок 3.19 – Сторінка перегляду резюме

Користувач може переглянути резюме, яке йому надіслав кандидат з усією необхідною інформацією. За наявності, можна перейти за посилання та переглянути відео-приклад робіт кандидата. Власник гурту повинен обрати, чи підходять йому дані цього кандидати та відповісти прийняттям або відхилення резюме. Обидві опцію повернуть власника назад, але прийняття ще створить новий чат із кандидатом для подальшого спілкування.

Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”
Завідувач кафедри
_____ Едуард ЖАРІКОВ
“___” _____ 2024 р.

МОБІЛЬНИЙ ЗАСТОСУНОК ДЛЯ ПОШУКУ МУЗИЧНОГО ГУРТУ

Графічний матеріал

КПІ.ІХ-0222.045440.06.99

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Любов ІВАНОВА

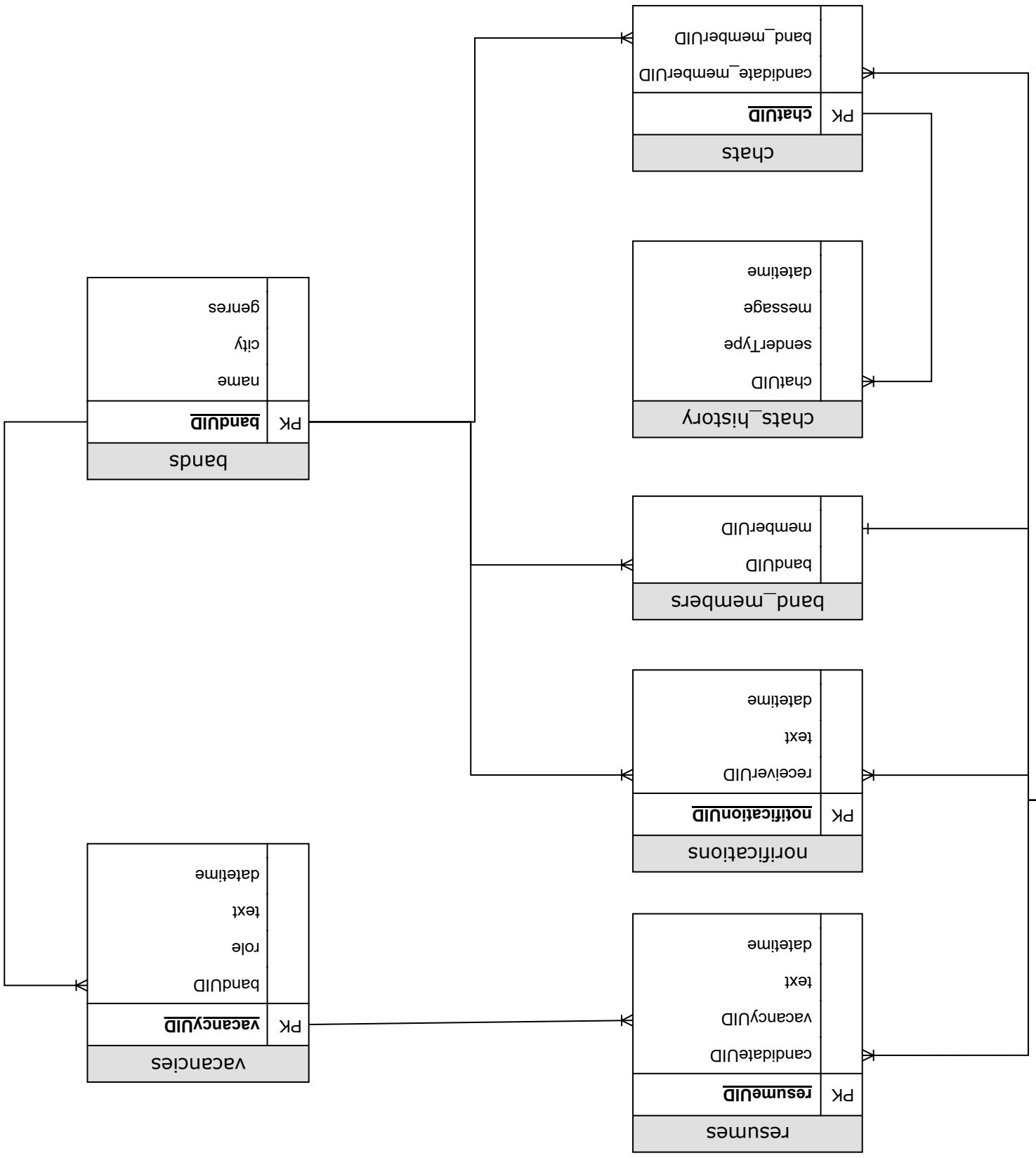
Нормоконтроль:
_____ Тетяна ШУЛЬКЕВИЧ

Виконавець:
_____ Валентин СЕРЕДЮК

Київ – 2024

КПІ.ІП-0222.045440.06.99.ССВ

					КПІ.ІП-0222.045440.06.99.ССВ
Зм.	Арк.	№ докум.	Підп.	Дата	Схема структурна варіантів використань
Розробив		Середюк В.В.			Лит. / Маса / Масштаб
Перевірив		Іванова Л.М.			Аркуш 1 / Аркушів 3
Т. контр.					Мобільний застосунок для пошуку музичного гурту
Н. контр.		Шулькевич Т.В.			КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-02
Затвердив		Жаріков Е.В.			



Зм.	Арк.	№ докум.	Підп.	Дата
Розробив		Середюк В.В.		
Перевірив		Іванова Л.М.		
Т. контр.				
Н. контр.		Шулькевич Т.В.		
Затвердив		Жарікова Е.В.		

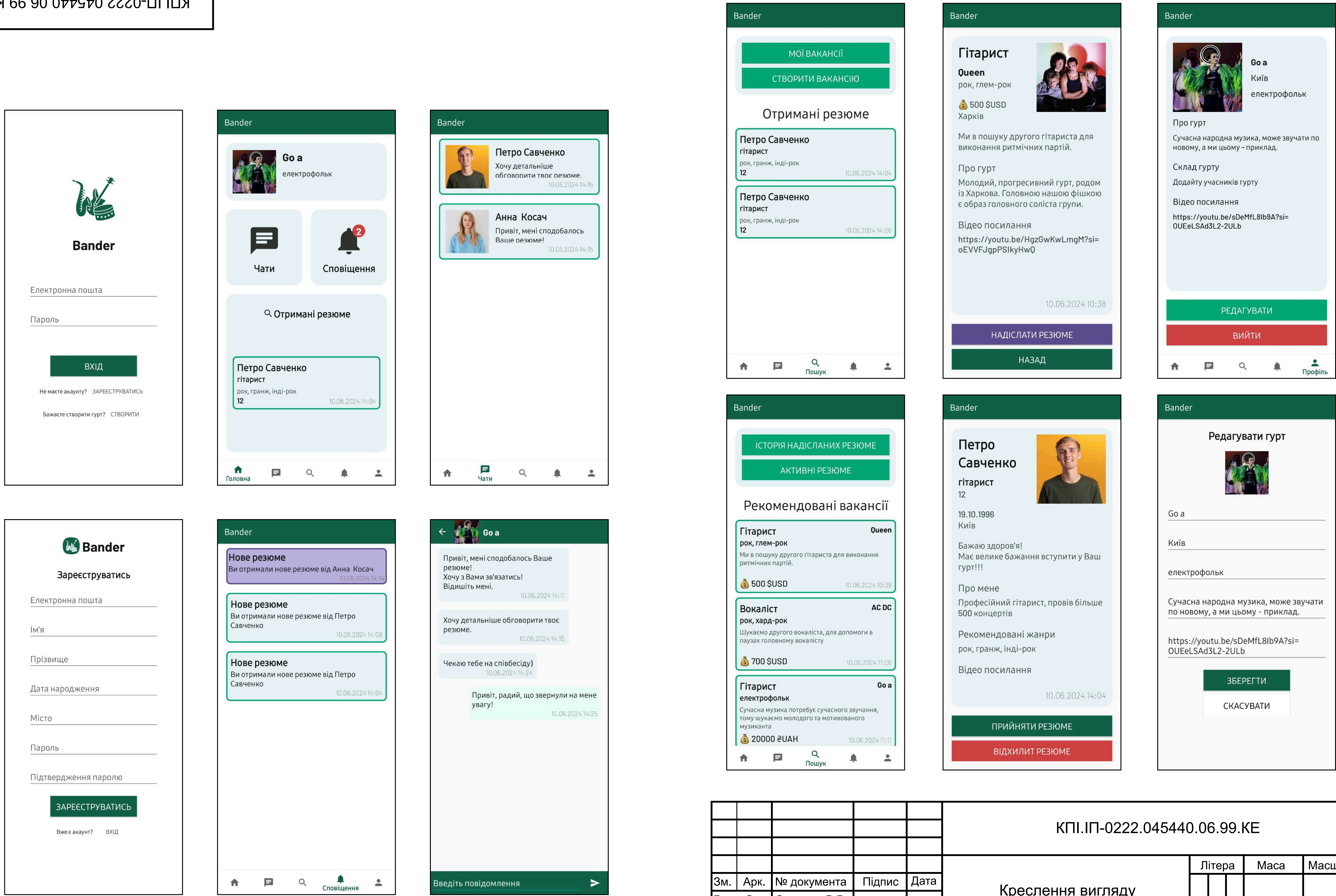
КПІ.ІП-0222.045440.06.99.СБД

		Схема бази даних	
Лит.	Маса		Масштаб
Аркуш 2			Аркушів 3

Мобільний застосунок для пошуку музичного гурту

КПІ ім.Ігоря Сікорського
Кафедра ІПІ
гр. ІП-02

КПІ.ІП-0222.045440.06.99.КЕ



					КПІ.ІП-0222.045440.06.99.КЕ			
Зм.	Арк.	№ документа	Підпис	Дата		Літера	Маса	Масштаб
Розробив		Середюк В.В.			Креслення вигляду екранних форм			
Перевірив		Іванова Л.М.						
Т. контр.						Аркуш 3		Аркушів 3
Н. контр.		Шулькевич Т.В			Мобільний застосунок для пошуку музичного гурту	КПІ ім.Ігоря Сікорського Кафедра ІПІ гр. ІП-02		
Затвердив		Жаріков Е.В.						