МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ПРИВАТНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ДО ЗАХИСТУ ДОПУЩЕНА

Зав. кафедрою _____
(підпис)
д.е.н., доцент Левицький С.І.

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

РОЗРОБКА МЕСЕНДЖЕРУ НА ANDROID ДЛЯ КОРПОРАТИВНИХ КЛІЄНТІВ БІЗНЕСУ

Виконав
ст. гр. КІ-112/м _____ Д.В. Скляр

Науковий керівник _____ Д.Г. Медведєв
к.т.н. (підпис)

Запоріжжя
2024

ПРАТ «ПВНЗ «ЗАПОРІЗЬКИЙ ІНСТИТУТ ЕКОНОМІКИ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ»

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ

Зав. кафедри _____
(підпис)
д.е.н., доцент Левицький С.І.

__.__.____ р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ

Студенту гр. КІ-112м , спеціальності *«Комп'ютерна інженерія*

Скляру Дмитру Вячеславовичу

1.Тема: Розробка месенджеру на Android для корпоративних клієнтів бізнесу

затверджена наказом по інституту № від __.09.2023 р.

2. Термін здачі студентом закінченої роботи: __.__.2023 р.

3. Перелік питань, що підлягають розробці:

- постановка завдання;
- інформаційний огляд;
- опис основних технологій реалізації;
- планування функціоналу додатку;
- створення функціональних UML діаграм для наочного зображення взаємодії міжкористувачами та доступних їм функцій;
- створення алгоритмів виконання завдання;
- розробка серверної архітектури у інфраструктурі SQLite;
- створення клієнтського додатку для більш зручної взаємодії із системою та отримання найбільш актуальної інформації;
- оформити звіт за результатами роботи.

ЗАТВЕРДЖУЮ
Зав.кафедрою _____

К А Л Е Н Д А Р Н И Й Г Р А Ф І К

підготовки магістерської дипломної роботи

студентом інституту ЗІЕІТ *денної* **форми навчання**

гр. _КІ-112м_ ____Скляру Дмитру Вячеславовичу _____

на 2023-2024 навчальний рік

№ етапу	Зміст	Терміни виконання	Готовність по графіку %, підпис керівника	Підпис керівника про повну готовність етапу, дата
1.	Корегування теми магістерської дипломної роботи, збір практичного матеріалу за темою магістерської дипломної роботи	**04.09.23-17.10.23**		
2.	**І атестація** І розділ магістерської дипломної роботи	**23.10.23-28.10.23**		
3.	**ІІ атестація** ІІ розділ магістерської дипломної роботи	**20.11.23-25.11.23**		
4.	**ІІІ атестація** ІІІ розділ магістерської дипломної роботи, висновки та рекомендації, додатки, реферат, перевірка програмою «Антиплагіат»	**18.12.23-23.12.23**		
5.	Доопрацювання магістерської дипломної роботи, підготовка презентації, отримання відгуку керівника і рецензії	**25.12.23-06.01.24**		
6.	**Попередній захист магістерської дипломної роботи**	**08.01.24-13.01.24**		
7.	Подача магістерської дипломної роботи на кафедру	**за 3 дні до захисту**		
8.	**Захист магістерської дипломної роботи**	**15.01.24-20.01.24**		

Дата видачі завдання: .09.2023 р.

Керівник кваліфікаційної
магістерської роботи _____ _Медведєв Д.Г._
(підпис) (прізвище та ініціали)

Завдання отримав до виконання _____ _Скляр Д.В.__

(підпис студента) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна магістерська робота містить 80 сторінок, 11 таблиць, 21 рисунків, 37 бібліографічних посилань, 7 додатксd.

Метою роботи є розробка месенджера на базі операційної системи Android.

Об’єктом дослідження є процес розробки месенджера на базі Android. Предметом дослідження виступають технології, що використовуються для розробки месенджера на базі Android

Здійснено детальний огляд предметної області та сучасних аналогів. У процесі дослідження використовувалися: технології розробки мобільних додатків під операційну систему Android, методи роботи з базою даних SQLite, методи передачі даних по HTTP протоколах при синхронізації локальної і серверної баз даних.

Здійснено проектування моделі предметної області, програмування сутностей та алгоритмів на базі клієнт-серверної архітектури. Серверна частина розроблялася з використанням Firebase, клієнтська − OAuth, Retrofit2 та Google Cloud Storage; база даних − SQLite.

У магістерській дипломній роботі удосконалені теоретико-методичні підходи до створення проєктів, доповнені підходи до створення проєктів та використання їх на різних операційних системах та у різних браузерах.

В роботі запропоновано концептуально новий підхід до створення, планування та використання проєктів, створення нового програмного забезпечення яке дозволить користувачам отримувати всі переваги користування сучасним месенджером.

БАЗА ДАНИХ, МЕСЕНДЖЕР, FIREBASE, GETSTREAM, PICASSO, ZXING QR SCANNER LIBRARY, SQLITE, NAVIGATION ARCHITECTURE COMPONENT, RXJAVA, MVVM, ROOM, OAUTH 2.0, TOOLBAR

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скорочення	**Повна назва**	**Пояснення/переклад**
БД	База даних	Сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами.
ООП	Об'єктно-орієнтоване програмування	
СУБД	Система управління базами даних	
AJAX	Asynchronous Javascript and XML	Технологія взаємодії з сервером без перезавантаження сторінки
API	Application Programming Interface	Прикладний програмний інтерфейс
App	Application	Додаток
BaaS	Backend-as-a-Service	Бекенд як сервіс
CORBA	Common Object Request Broker Architecture	Технологічний стандарт написання розподілених додатків
CRUD	Create, Read, Update і Delete	Створення, зчитування, оновлення та видалення
CSS	Cascading Style Sheets	Каскадні таблиці стилів
DOM	Document Object Model	Об'єктна модель документа
ES(-2015/-6)	ECMAScript	Стандарт мови програмування, затверджений міжнародною організацією ECMA згідно зі специфікацією ECMA-262
FaaS	Firebase-as-a-Service	Firebase як сервіс
HTML	HyperText Markup Language	Мова розмітки гіпертексту

ВСТУП

Багатьом організаціям необхідно покращити залучення співробітників і спілкування на робочому місці, особливо враховуючи нову тенденцію віддаленої роботи. Ефективна комунікація з клієнтами також важлива для успіху бізнесу. Особливо для компаній, які належать до ліги постачальників послуг цілодобової підтримки.

Загальним рішенням для полегшення спілкування між співробітниками та клієнтами є розробка функціональності чату в реальному часі, яка дозволяє користувачам отримувати потрібну інформацію за лічені секунди. Ви можете вбудувати функцію чату в існуючі програми (наприклад, програму для мобільної торгівлі) або створити окрему програму, наприклад Slack, для спілкування на робочому місці.

Існуючі програми обміну повідомленнями оснащені багатьма корисними функціями та розширеннями, але можуть коштувати чималі копійки. Наприклад, Slack стягує 8 доларів на місяць за активного користувача. Крім того, деякі галузі (наприклад, фінанси) потребують сильного контролю конфіденційності, тому компаніям потрібно зберігати дані на власних серверах. Це створює потребу створити спеціальний месенджер або інтегрувати функції чату в існуючі програми.

З огляду на більшу розповсюдженість мобільних пристроїв з операційною системою Android то метою даної роботи є реалізація мобільного додатку на базі цієї операційної системи.

Основні задачі дипломної роботи:

- планування функціоналу додатку;
- створення функціональних UML діаграм для наочного зображення взаємодії міжкористувачами та доступних їм функцій;
- створення алгоритмів виконання завдання;
- розробка серверної архітектури у інфраструктурі SQLite;
- створення клієнтського додатку для більш зручної взаємодії із

системою та отримання найбільш актуальної інформації.

Об’єктом досліджень даної роботи є процес розробки месенджера на базі Android.

Предметом дослідження виступають технології, що використовуються для розробки месенджера на базі Android.

Новизна полягає у розробці месенджера за допомогою власного методу, який найбільше мінімізує час розробника та найкраще задовольняє потреби користувачів. Практична значимість роботи полягає у можливості створення месенджера за допомогою найкращих технологій. Практична цінність програмної системи, що розроблятиметься, визначається її універсальністю, можливістю її використання на будь-якій операційній системі, цінністю яку вона буде приносити користувачам, які її використовують та кількістю активних користувачів у системі.

У даній роботі важливе місце займає мова програмування, за допомогою якої буде створюватись додаток, а також допоміжні їй технології. У даному випадку найкращим варіантом буде мова програмування PHP. Завдяки використанню інфраструктури SQLite та бази даних Firebase розробка спрощується у багато разів.

Також варто зазначити, що у роботі слід використовувати OAuth 2.0 для обміну користувачами певними даними із програмою, зберігаючи приватні імена користувачів, паролі та іншу інформацію. У поєднанні з бібліотеками Picasso та ZXing QR scanner library процес користування додатком стає найкращим для користувачів.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ПРОЄКТУ. ПОСТАНОВКА ЗАДАЧІ

1.1. Характеристика задачі

Ідея створення власного або корпоративного месенджера виникає, коли у компанії вже є сотні, а то й тисячі клієнтів, з якими вони постійно спілкуються. Часом в цей час у них виникає необхідність дати клієнтам можливість спілкуватися між собою. Та й не просто писати текстові повідомлення, а створювати групи, відправляти фото, відео та файли. Загалом, робити всі ті речі, які ми робимо щодня в Skype, Viber, Telegram, WhatsApp.

Якщо компанії потрібно дати клієнтам можливість спілкуватися між собою в рамках однієї закритої площадки або створити простір для безпечної передачі конфіденційної інформації, використовувати публічні месенджери не вийде. Досить важко одночасно адмініструвати необмежену кількість груп в публічних месенджерах і контролювати склад учасників і інформації, що передається в групах.

У такому випадку проблему може вирішити створення власного або корпоративного месенджера.

Microfocus повідомляє, що щохвилини у всьому світі надсилається понад 100 мільйонів повідомлень. Не дивно, що ринок додатків для обміну повідомленнями стрімко розвивається.

При розробці дипломного проєкту важливо створити інфраструктуру, за допомогою якої користувачі зможуть спілкуватися між собою, ділитися важливими фото та відео, відправляти реакцію на повідомлення.

Для спрощення взаємодії користувача з системою перш за все необхідно розробити прототип для майбутнього додатку за допомогою онлайн-сервісу Figma. Далі переходимо до розробки безпосередньо серверної та клієнтської частин додатку.

Для запобігання проблемам безпеки даних, які можуть виникнути за умови надання API доступу до персональних даних зареєстрованих користувачів, необхідно дотримуватись оригінальної структури запитів.

Для забезпечення стабільності та масштабування серверну частину складають скрипти, написані мовою програмування PHP. У роботі було використано бази даних firebase database, firebase storage, firebase authentication.

База даних Firebase дозволяє створювати багатофункціональні програми для спільної роботи, забезпечуючи безпечний доступ до бази даних безпосередньо з клієнтського коду. Дані зберігаються локально, і навіть в автономному режимі події в реальному часі продовжують запускатися, що дає кінцевому користувачу можливість швидко реагувати. Коли пристрій відновлює з'єднання, база даних реального часу синхронізує зміни локальних даних з віддаленими оновленнями, які відбулися, коли клієнт був в автономному режимі, автоматично об'єднуючи будь-які конфлікти.

Ручне введення залежностей або локаторів служб у додатку для Android можуть бути проблематичними залежно від розміру проєкту. Тому для обмеження складності проєкту було використано Dagger для управління залежностями.

У даному проєкті буде використовуватись реляційна база даних SQLite.

Room використовується для зберігання даних додатків в Android. Він забезпечує абстракційний рівень над SQLite, щоб забезпечити вільний доступ до бази даних, одночасно використовуючи всю потужність SQLite.

Окрім серверної частини також існує і клієнтський додаток, який також необхідно створити для відбиття інформації для користувача у якомога зручнішому варіанті представлення даних.

Додаток «EXPO» функціонує на основі клієнт-серверної взаємодії. Для отримання інформації з бібліотеки Firebase, яка виконує роль сервера, та сторонніх API використано HTTP клієнт Retrofit2. Дані на Firebase зберігаються в Google Cloud Storage.

1.2. Огляд чинних програмних комплексів

Мобільний Інтернет, який налічує 3,7 мільярда користувачів, стає надзвичайно популярним, а месенджери є одними з найбільш часто використовуваних програм. Вони дають можливість спілкуватися в чаті і миттєво відправляти різні файли, економлячи багато часу.

У 2022 році приголомшливі 3,09 мільярда осіб були постійними користувачами програм для чату. Це число продовжує зростати і, за прогнозами, досягне 3,51 мільярда до 2025 року. Лідирує в списку WhatsApp, найбільш використовуваний у світі додаток для обміну повідомленнями, який має понад 2 мільярди активних користувачів щомісяця. Примітно, що його база користувачів і охоплення особливо міцні в країнах за межами Сполучених Штатів, зміцнюючи його позиції як одного з найбільш завантажуваних додатків соціальних мереж на мобільних пристроях у всьому світі.

Провідними програмами для обміну повідомленнями у світі є What’s App, Viber, Line і WeChat (рич 1.1).

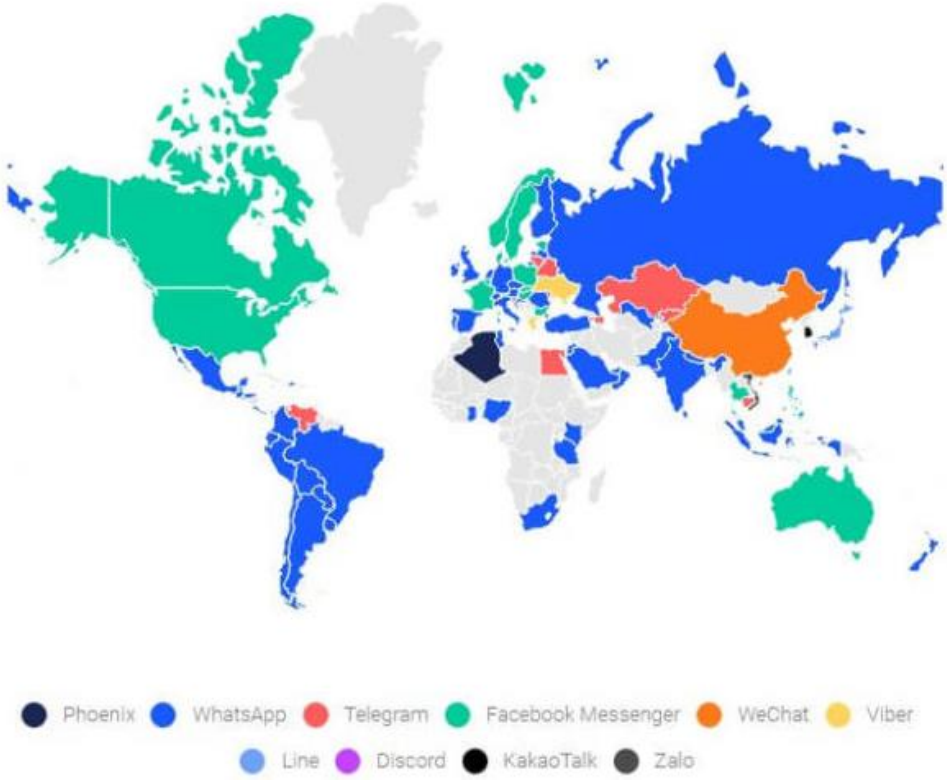

Рисунок 1.1 − Кількість активних користувачів у найпопулярніших месенджерах [37]

У кожному месенджері є свої переваги та недоліки. Для створення успышного додатку для обміну повідомленнями слід розглянути, які можливості вже надають користувачам гіганти індустрії обміну повідомленнями.

WhatsApp − популярна безкоштовна система миттєвого обміну текстовими повідомленнями для мобільних й інших платформ з підтримкою голосового й відеозв'язку. Вона є наймасовішим месенджером. Лише в Сполучених Штатах у 2019 році WhatsApp набрав 68,1 мільйона активних користувачів. Очікується, що ця цифра зросте до 85,8 мільйонів користувачів у США у 2023 році, але найцікавіше те, що цей додаток для обміну повідомленнями особливо популярний на ринках за межами США, але стикається з сильним конкуренція з боку азійських соціальних месенджерів.

У той час як WhatsApp стикається з сильною конкуренцією з боку таких гігантів соціальних медіа, як Facebook і Twitter у Сполучених Штатах, програма підтримує значну базу користувачів, яка може похвалитися понад 68 мільйонами активних користувачів.

У 2009 році WhatsApp було випущено як мобільний додаток для обміну повідомленнями для запуску, і з того часу він міцно зайняв позицію найпопулярнішого месенджера у світі, доступного майже для всіх мобільних операційних систем. WhatsApp є дешевою альтернативою SMS-повідомленням, які оплачуються оператором, особливо тому, що це вигідно для міжнародних або групових повідомлень. Популярність WhatsApp на мобільних ринках також ґрунтується на його ключових функціях:

- Швидка реєстрація за номером мобільного телефону;
- Події в реальному часі;
- Обмін файлами та зображеннями;
- Запис голосової пошти;
- Голосові дзвінки.

Крім того, у WhatsApp ви можете завантажувати файли зі сторонніх програм, таких як Google Drive, OneDrive та iCloud, у повідомлення. Для iOS він інтегрований із Siri, яка може надсилати повідомлення або здійснювати

Telegram, хмарна служба обміну миттєвими повідомленнями, з'явилася в 2013 році та підірвала ринок, заявивши, що є найбезпечнішим додатком для обміну повідомленнями.

Авторами є Миколай та Павло Дурови. Програма написана мовою високого рівня C++. Для нього створили протокол MTProto, який використовує декілька протоколів шифрування. При авторизації користувача застосовуються алгоритми RSA-2048, DH-2048 для шифрування. Також використовуються криптографічні хеш-алгоритми SHA-1 та MD5.

Telegram зробив безпеку своєю відмінною рисою та стверджує, що всі його дії: чати, групи та медіа зашифровані. Telegram навіть кілька разів змінював юрисдикцію штаб-квартири, щоб уникнути проблемних правил, тепер це Дубай.

Інша перевага рішення полягає в тому, що його можна використовувати на кількох пристроях (смартфонах, планшетах, ноутбуках і настільних комп'ютерах) і платформах (iOS, Android, Windows Phone, Windows NT, macOS, Linux).

Ключові особливості месенджера:

- мультимедійні повідомлення,
- заплановані та тихі повідомлення,
- голосові та відеодзвінки,
- редагування фото і відео перед відправкою,
- самознищення повідомлень у секретних чатах,
- необмежені можливості налаштування,
- безкоштовне необмежене онлайн-сховище,

- обмін геоданими в реальному часі,
- додавання контактів на основі місцезнаходження користувача,
- боти,
- повільний режим (може активуватися адміном групи),
- опитування,
- створення каналів для трансляції повідомлень широким аудиторіям,
- надсилання нестисненого відео та зображень,
- можливість блокування розмов,
- завершення активних сесій.

Більше того, Telegram запустив власну цифрову валюту Gram, стверджуючи, що її метою є створення повсякденної криптовалюти для масового ринку.

Facebook Messenger.

У 2011 році Facebook вирішила створити програму для чату — Messenger, яка призначена виключно для спілкування в чаті, вона така ж відома, як WhatsApp, коли справа стосується обміну повідомленнями. Messenger має зрозумілий і легкий інтерфейс, яким легко користуватися.

Окрім обміну текстовими повідомленнями, смайликів і фотографій, у ньому дуже багато функцій:

- Голосовий та відеодзвінок;
- Використання реакцій у чаті, GIF-файлів і спільного використання стікерів;
- Дійсно величезна кількість ботів;
- Можливість додавати людей до Messenger, натиснувши вкладку «Сканувати код», щоб відсканувати код профілю користувача;
- Багата кількість ігор всередині месенджера;

- Функція налаштування нагадувань для планів;
- Завдяки останньому оновленню користувач також може ділитися своїм/її місцем розташування з друзями.

Опитування під назвою «Найбільш використовуваний месенджер за брендом» визначило «Facebook Messenger» як провідну платформу для обміну повідомленнями, а «Viber» займає нижню частину спектру. Ці висновки випливають із всебічного онлайн-опитування, проведеного в 2023 році, яке охопило 5504 респондентів у Сполучених Штатах. Месенджер імпортує контакти Facebook і простий у використанні як на настільному комп'ютері, так і на мобільних пристроях.

Viber.

Viber був запущений десять років тому як додаток для обміну повідомленнями, і станом на березень 2020 року кількість його унікальних ідентифікаторів користувачів наблизилася до 1,2 мільярда в майже 200 країнах. Як і інші рішення для обміну повідомленнями, Viber дозволяє своїм користувачам залишатися на зв'язку, і не тільки, але й:

- Установка програми на кількох платформах і пристроях,
- Синхронізація історії та переадресація викликів,
- Здійснювати аудіо- та відеодзвінки
- Обмінюйтеся стікерами, GIF-файлами та смайлами, які дозволяють більше налаштовувати
- Записуйте та надсилайте негайні аудіо- та відеоповідомлення,
- Діліться зображеннями, музикою, відео та файлами,
- Видалити прочитані повідомлення та розмови,
- Створюйте спільноти.

Viber Out також дозволяє здійснювати якісні міжнародні дзвінки на будь-який стаціонарний або мобільний телефон у всьому світі, але для цього у вас повинен бути позитивний баланс.

Крім того, розширення чату дозволяють користувачам шукати та ділитися ресторанами, подіями, готелями тощо, навіть не виходячи з чату.

Slack − висхідна зірка корпоративних комунікацій

Програми спілкування на робочому місці стають все більш популярними. З моменту заснування Slack наприкінці 2013 року він став одним із найкращих корпоративних месенджерів.

За оцінками, станом на 2023 рік щоденна кількість активних користувачів Slack досягла 32,3 мільйона, що свідчить про збільшення в порівнянні з 25,7 мільйонами користувачів, про які повідомлялося в 2022 році. Заглядаючи наперед, прогнози показують, що кількість щоденних активних користувачів Slack, як очікується, зросте до 38,8 мільйонів у 2024 році і далі до 47,2 мільйонів у 2025 році.

Користувачі Slack можуть надсилати приватні повідомлення один одному та використовувати широкий спектр функцій для взаємодії та розваг:

- Можливість створювати багато різних каналів для спеціалізованих розмов;
- Slackbot, який працює як віртуальний блокнот для запису ідей, за якими ви хочете стежити;
- Повідомлення можна шукати за ключовими словами, також користувачі можуть налаштувати сповіщення, щоб бути в курсі бесід за ключовими словами;
- Інтеграція зі сторонніми програмами, такими як Asana, Dropbox і Google Hangouts, допомагає отримувати інформацію з різних джерел;
- Slack інтегровано з Giphy (додаток для пошуку GIF-файлів в Інтернеті), користувач отримує доступ до нього, просто ввівши " /giphy" і ключове слово на зразок «Hello world!»;

Мінус Slack полягає в тому, що він не зберігає історію повідомлень тривалий час. Але, не дивлячись на динаміку, користувачів це не дуже заважає.

Розглянувши основні функції та задачі роботи месенджерів можемо привести порівняльну таблицю (табл. 1.1).

Таблиця 1.1 – Основні характеристики месенджерів

Назва	Facebook Messenger	Viber	Telegram
Алгоритм шифрування	AES-256, HMAC-SHA256, 3-DH	SHA-256RSA	RSA-2048, DH-2048, AES-256, SHA-256
Додаткові методи захисту	HTTPS, MQTT	SSL	HTTPS
Секретні чати	+	+	+
Аудіо-дзвінки	+	+	+
Відео-дзвінки	+	+	+

Ознайомившись з найкращими компаніями обміну повідомленнями, ми можемо зробити висновок, що найголовніше – знайти свою цільову аудиторію та задовольнити її запити, надаючи багатий функціонал. Наприклад, Slack зайняв свій ринок, надаючи рішення для корпоративних потреб. WhatsApp і Facebook Messenger демонструють, що на ринку можуть успішно співіснувати два продукти зі схожим функціоналом і навіть зі схожими інтерфейсами.

WeChat популярний завдяки розширеній функціональності, а також Facebook заблоковано в деяких азіатських регіонах. Telegram завоював свою аудиторію, заявивши, що вони отримали високий рівень безпеки та конфіденційності з додатком.

Через екстремальне зростання ринку мобільних додатків і великої кількості смартфонів недорогі або безкоштовні програми обміну миттєвими

повідомленнями стали дешевою альтернативою обміну повідомленнями через SMS або MMS.

1.2. Вхідна інформація

Основою дипломної роботи є створення месенджера для Android платформи. Говорячи про месенджер, основною вхідною інформацією є дані про відвідувачів, смс-повідомлення, фото та відео, які вони будуть відправляти між собою.

Месенджер буде написано за допомогою GetStream, який являє собою широку інфраструктуру, що дозволяє за декілька годин розгортати масштабовані канали та обмін повідомленнями в чаті. GetStream використовується понад 500 компаніями та 200 мільйонами кінцевими користувачами [1].

GetStream можна використовувати як з Java, так і з Kotlin [2].

Підбираємо засоби програмування з точки зору стабільності програми для месенджера та всього програмного рішення. Для розробки будемо використовувати ці інструменти, коригуючи їх відповідно до архітектури.

Спільного між цими мовами справді не так вже й багато. Попри того, що Kotlin є поліпшеним Java, відмінностей між ними набагато більше ніж чогось спільного.

Kotlin і Java — це в першу чергу мови, що доповнюють один одного. Безумовно, деякі функції краще реалізовувати в Kotlin, а деякі практичніше було б використовувати в Java. Попри цього обидві мови повністю сумісні й здатні компілюватиметься в байт-код. Kotlin в такому випадку повністю використовує JVM екосистему і бібліотеку, що дозволяє гранично просто користуватися Java методами й класами.

Те, що вони сумісні між собою, допомагає при початку роботи з Kotlin, дозволяючи періодично впроваджувати код Kotlin в програми написані на Java.

Саме в JetBrains був створений Kotlin і представлений широкій публіці на Google I/O. Тоді його представили як другу після Java офіційну мову розробки Android-додатків.

Важливо також додати, що Kotlin краще Java тільки в Android розробці. Якщо порівнювати backend, то тут Java немає рівних.
Деякі розробники кажуть, що Kotlin — це своєрідна відповідь Android на Swift в iOS.Переваги Kotlin над Java:

- Додаткові потоки;
- Просте створення та розширення класів даних;
- Лямбда-вирази;
- Делегування;
- Вбудовані функції вищого порядку;
- Функція розумного приведення;
- Підтримка одного і більше конструкторів;
- Відсутність необхідності виявлення винятків;
- Простий код;
- Nullsafe;
- Kotlin Native;
- Функції вищого порядку [3].

Picasso — бібліотека Android з відкритим вихідним кодом для завантаження зображень в додаток для Android. Вона використовує OkHttp (мережеву бібліотеку від одного і того ж розробника) під капотом для завантаження зображень через Інтернет [4].

ZXing QR scanner library — це бібліотека для обробки зображень зі штрих-кодом, реалізована на Java, з портами на інші мови [5].

QR-код або код швидкого реагування — це, в основному, двовимірний штрих-код, який декодує вміст із більшою швидкістю. Перша система QR-коду була винайдена японською компанією Dense-Wave в 1994 році.

І сьогодні навіть смартфон, що має додаток для зчитування штрих-коду, може сканувати та декодувати будь-який QR-код [6].

Вхідною інформацію також можна вважати бібліотеку Retrofit2, що використовується для мережевої взаємодії месенджера. За його допомогою REST API перетворюється в інтерфейс Java. Він використовує анотації для опису HTTP-запитів, за замовчуванням замінює параметр URL і підтримку параметрів запиту. Крім того, він забезпечує функціональні можливості для копіювання багатосторінкового запиту і завантаження файлів [7].

Як обгортку над стандартними механізмами, яка покликана впорядкувати й спитати прості та складні шаблони навігації в додатку виступає Navigation Architecture Component. Компонент навігації також забезпечує узгоджену і передбачувану взаємодію з користувачем, дотримуючись встановленого набору принципів.

Бібліотека надає ряд переваг, у тому числі:

- Автоматична обробка транзакцій фрагментів;
- Коректна обробка кнопок «Вверх» і « Назад» за замовчуванням;
- Поведінка за замовчуванням для анімації і переходів;
- Глибокі зв'язки як першокласні операції;
- Реалізація шаблонів навігації для користувача інтерфейсу (таких як navigation drawer і bottom navigation) з невеликою додатковою роботою;
- Безпека типів при передачі інформації під час навігації;
- Інструменти Android Studio для візуалізації і редагування navigation flow додатка [8].

Важливим елементом внутрішньої інформації месенджера також є Firebase. Це сервіс від Google, який дозволяє поширювати інформацію з бази даних додатків на будь-яких платформах і що зручно − все в реальному часі [9].

Ручне введення залежностей або локаторів служб у додатку для Android можуть бути проблематичними залежно від розміру проєкту. Тому

для обмеження складності проєкту було використано Dagger для управління залежностями.

Dagger автоматично генерує код. Оскільки код генерується під час компіляції, він простежується і є більш продуктивним, ніж інші рішення на основі відбиття, такі як Guice.

Dagger звільняє від написання коду шляхом:

- Створення AppContainer коду (графіку програми), який зазвичай вручну застосовують в DI section;
- Створення факторій для класів, доступних на графіку додатків. Це те, як залежності задовольняються внутрішньо;
- Вирішення питання про повторне використання залежності чи створення нового екземпляра за допомогою області видимості;
- Створення контейнерів для конкретних потоків. Це покращує продуктивність програми, вивільняючи об’єкти в пам’ять, коли вони більше не потрібні.

Dagger автоматично робить все це під час побудови, поки ми оголошуємо залежність класу та вказуємо, як їх задовольнити, використовуючи анотації. Dagger генерує код та внутрішньо створює графік об’єктів, на які він може посилатися, щоб знайти спосіб надати екземпляр класу. Для кожного класу в графіку Dagger генерує клас factory-type, який він використовує внутрішньо для отримання екземплярів цього типу.

Під час збірки Dagger проходить через код і:

- Створює та перевіряє графіки залежностей, гарантуючи, що:
- Залежності кожного об’єкта можуть бути задоволені, тому не існує винятків під час виконання;
- Циклів залежностей не існує, тому не існує нескінченних циклів.
- Генерує класи, які використовуються під час виконання для створення фактичних об'єктів та їх залежностей [10].

Ще однією вхідною інформацією є бібліотека RxJava. Rx — це потужний інструмент, який дозволяє вирішувати проблеми в елегантному декларативному стилі, притаманному функціональному програмуванню.

Rx має наступні переваги:

- інтуїтивність. Дії в Rx описуються в такому ж стилі, як і в інших бібліотеках натхнених функціональним програмуванням, наприклад, Java Streams. Rx дає можливість використовувати функціональні трансформації над потоками подій;
- Можливість розширення. RxJava може бути розширена для користувача операторами. І хоча Java не дозволяє зробити це елегантним чином, RxJava пропонує всю розширюваність доступну в реалізаціях Rx будь-якою іншою мовою.
- Декларативність. Функціональні трансформації оголошені декларативно.
- Скомпонованість. Оператори в Rx легко компонуються, щоб проводити складні операції.
- Перетворюваність. Оператори в Rx можуть трансформувати типи даних, фільтруючи, обробляючи і розширюючи потоки даних при необхідності [11].

У розробці клієнтської частини додатку важливе значення приймає MVVM (Model-View-ViewModel). Це шаблон архітектури клієнтських додатків, який був запропонований Джоном Госсманом (John Gossman) як альтернатива шаблонами MVC і MVP при використанні технології зв'язування даних (Data Binding). Його концепція полягає в відділенні логіки представлення даних від бізнес-логіки шляхом винесення її в окремий клас для чіткішого розмежування.

Елемент Toolbar призначений для швидкого і зручного доступу користувача до часто використовуваних функцій.

Ще одним необхідним документом є Google OAuth 2.0 для доступу до API Google.

OAuth 2.0 дозволяє користувачам обмінюватися певними даними із програмою, зберігаючи приватні свої імена користувачів, паролі та іншу інформацію. Наприклад, програма може використовувати OAuth 2.0 для отримання дозволу від користувачів зберігати файли на своїх Дисках Google.

Цей потік OAuth 2.0 спеціально призначений для авторизації користувачів. Він призначений для програм, які можуть зберігати конфіденційну інформацію та підтримувати стан. Правильно авторизована програма веб-сервера може отримати доступ до API, поки користувач взаємодіє з програмою або після того, як користувач залишив програму [35].

Розглянемо наостанок важливу вхідну інформацію Room, який використовується для зберігання даних додатків в Android. Це частина нової Android Architecture, група бібліотек від Google, яка підтримує доречну архітектуру додатків.

1.4. Вихідна інформація

Вихідну інформацію користувач може отримувати також за допомогою клієнтського додатка. Серед основної вихідної інформації є користувальницький інтерфейс додатку, який включає до себе наступні можливості:

- створення та редагування акаунта;
- додавання номера телефону, паролю та нікнейму;
- додавання, редагування та пошук контакту;
- створення нового чату;
- текстові повідомлення, фото, відео та смайлики;
- відправка реакцій та відповідей на повідомлення;
- редагування відправлених повідомлень.

Починаючи роботу з месенджером, користувач бачить привітальне зображення (рис. А.1) та особливості месенджера (рис. Б.1, рис. Б.2, рис. Б.3).

Основними особливостями месенджера, які створюють довірю користувача, є:

- Інтуїтивно зрозумілий інтерфейс;
- «Scan-to-Chat» для додавання нового контакту;
- Конфіденційність.

Було проведено ретельне дослідження ринку, щоб створити месенджер, відповідний до найкращих практик користувальницького інтерфейсу / UX цієї епохи. Завдяки цьому було забезпечено досконаліть користування месенджером.

Після погодження з особливостями месенджера користувачу необхідно зареєструватися або авторизуватися (рис. В.1). Розглянемо більш детально обидва процеси користувальницького досвіду:

- Реєстрація, яка включає до себе наступні етапи:
    1) Генерація користувачем персонального ID, яка дозволяє об’єднати в один ланцюжок дії, зроблені однією людиною різних пристроях (рис. Г.1, рис. Г.2, рис. Г.3, рис. Г.4);
    2) Створення паролю, який не може включати до себе менш ніж 8 символів (рис. Д.1, рис. Д.2, рис. Д.3, рис. Д.4);
    3) Створення нікнейму, який повинен складатися з латинських літер та включати до себе не менш 5 символів (рис. Е.1, рис. Е.2, рис. Е.3, рис. Е.4);
    4) Додавання номеру телефона (за бажанням). На відміну від інших месенджерів, EXPO дозволяє користуватися додатком без прив’язки до номера телефона (рис. Ж.1, рис. Ж.2);
    5) Додавання зображення в профіль, яке можна зробити за допомогою камери при реєстрації, або завантажити з Photo Library / файлів телефону (рис. И.1, рис. И.2).

- Авторизація у месенджері за допомогою ID та паролю від акаунта (рис. К.1, рис. К.2, рис. К.3, рис. К.4).

Після реєстрації або авторизації в месенджері перед користувачем відчиняються можливості:

- Редагування профілю: зміна власного фото, ID та нікнейму (рис. Л.1, рис. Л.2);
- Налаштування: повідомлення, конфіденційність та статус (рис. М.1, рис. М.2);
- Перегляд та видалення наявних контактів (рис. Н.1, рис. Н.2) та додавання нових за допомогою клавіатури або сканування ID (рис. Н.3, рис. Н.4, рис. Н.5);
- Вибір співрозмовника (рис. П.1);
- Отримання повідомлень (рис. П.2);
- Відправка, видалення та копіювання повідомлення (рис. П.3);
- Редагування повідомлення (рис. П.4);
- Текстова відповідь на повідомлення співрозмовника (рис. П.5);
- Відправка реакції на повідомлення (рис. П.6);
- Відправка фото та відео співрозмовнику (рис. П.7).

Висновки до розділу І

У цьому розділі була поставлена задача проаналізувати чинні програмні комплекси, схарактеризувати цілеспрямованість створення власного або корпоративного месенджера та схарактеризувати вхідну й вихідну інформацію щодо проєкту даної роботи.

Комплексний аналіз множин схожих між собою месенджерів допоміг зробити висновок, що у своєї більшості вони мають однакові між собою можливості та функції для користувачів. Виявивши плюси та мінуси усіх наявних у мережі Інтернет месенджерів, можна побачити, що всі вони також мають й щось особливе, чого нема в інших. Цей аналіз допоміг зробити

композицію всіх найкращих функцій та відокремити відповідні недоліки, яких треба уникати.

Як уже було зазначено, було детально описано вхідну та вихідну інформацію додатку, на які буде опиратись проєктування функціональних діаграм за допомогою UML та серверна архітектура у наступному розділі.

До вхідної інформації проєкту відносяться бібліотека GetStream, яка використовується для власної імплементації месенджера, мова програмування Kotlin, що буде використовуватись для створення клієнтської та серверної частини додатку, та бібліотека Retrofit2, що використовується для мережевої взаємодії месенджера. Як обгортку над стандартними механізмами, яка покликана впорядкувати й спростити прості та складні шаблони навігації в додатку, обрано Navigation Architecture Component. Важливими елементами внутрішньої інформації месенджера також було зазначено Firebase, Dagger, RxJava та Room.

Було проведено ретельне дослідження ринку, щоб створити месенджер, відповідний до найкращих практик користувальницького інтерфейсу / UX цієї епохи. Завдяки цьому було забезпечено досконаліть користування месенджером

РОЗДІЛ ІІ
РОЗРОБКА АЛГОРИТМУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1. Створення циклу розробки продукту

Життєвий цикл розробки програмного забезпечення — це концептуальна модель, яка описує етапи проєкту та розвитку системи інформаційних систем, а також це процес, за допомогою якого виконується процес створення програмного забезпечення, де основною ціллю є створення продукту з найвищою якістю та найнижчою вартістю за найкоротший час [3 р. 83]. Приклад вимог, де якість є центральною рисою, до якої необхідно досягнути при розробці програмного забезпечення наведено на рис. 2.1.

Рисунок 2.1 − Приклад трикутнику вимог,
Джерело: сформовано на основі узагальнення досліджень

Згідно з каскадною моделлю життєвого циклу програмні системи проходять у своєму розвитку такі фази:

- розроблення;
- супровід.

Фази розбиваються на ряд етапів.

Розглянемо фази життєвого циклу розробки програмного забезпечення більш детально:

1) Специфікація (Requirements Specification), або планування — формулювання завдання і визначення вимог. На етапі формулювання завдання необхідно:
   - Визначити проблеми, цілі та ресурси;
   - Вивчити можливості альтернативних рішень шляхом зустрічей з клієнтами, постачальниками та консультантами;
   - Вивчити продукти конкурентів та фактори, які допоможуть зробити продукт краще, ніж у конкурентів.

Завдання аналізу полягає в тому, щоб вибудувати опис програми у вигляді логічної системи.

Цей етап несе у собі найбільшу цінність і не може бути замінений або виключений зі списку етапів життєвого циклу розробки програмного забезпечення. У результаті специфікації обов'язково формується технічне завдання на розроблення програмного забезпечення.

2) Проєктування (Design). На цьому етапі необхідно створити інтуїтивно зрозумілий інтерфейс, не жертвуючи функціональністю. При такій жорсткій конкуренції на цьому етапі потрібно продумати і відшліфувати всі найменші деталі. Також узгоджувати дизайн будь-якого виробу із замовником. Результати виконання цього етапу оформлюються як технічний проєкт, вимоги до документів якого встановлені стандартом ДСТУ 34.201-89 [16].
3) Реалізація (Construction), або імплементація передбачає розробку необхідного функціоналу.
4) Тестування (Testing and debugging) передбачає комплексний пошук помилок прописаного функціоналу. Якщо на цьому етапі були виявлені помилки, продукт відправляється на їх виправлення. Етапи 3 і 4 можуть виконуватись паралельно або у циклі в залежності від

того, як цей процес встановили на етапі специфікації.

5) Експлуатація (Installation) та супроводів охоплюють всю діяльність щодо забезпечення нормального функціонування програмних систем, підвищення експлуатаційних характеристик системи, адаптацію системи до довкілля. Фактично відбувається еволюція системи.
6) Створення документації. Цей процес є фінальним і необхідний на випадок оновлення вимог до продукту та його подальшої підтримки іншими розробниками та тестувальниками.

Створення та робота додатку за визначеним процесом має плюси як для розробників, для яких від проєкту до проєкту змінюються тільки функціонал, який необхідно створити, так і для замовника програмного забезпечення, який через процеси може відстежувати роботу, яка була зроблена, робиться зараз та буде зроблена у майбутньому [17 с. 211].

2.2. Розробка UML діаграм функцій додатку

Відштовхуючись від поданого вище плану створення додатку, можна зрозуміти, що першим етапом повинно стати планування функцій, які необхідно створити у додатку. Важливим елементом цього етапу є проєктування необхідного функціоналу за допомогою уніфікованої мови об'єктноорієнтованого моделювання (Unified Modeling Language — UML).

UML — це графічна мова для візуалізації, конкретизації, побудови та документування складових програмно-орієнтованих систем. UML надає стандартний спосіб створення візуальних описів системи, включаючи такі концептуальні речі, як опис бізнес-процесів, принципів функціонування, а також більш конкретні речі, такі як опис фрагментів коду мови програмування, схем баз даних та компонентів програмного забезпечення.

До переваг мови UML можна віднести різноманітні інструментальні засоби, які як підтримують життєвий цикл продукту, так і дозволяють налаштувати й зобразити специфіку діяльності розробників різних елементів проєкту.

Основними характеристиками об'єктноорієнтованої мови моделювання UML є:

- Організація взаємодії замовника і розробника (груп розробників) продукту шляхом побудови репрезентативних візуальних моделей;
- Спеціалізація базових позначень для конкретної предметної області.

Повністю виконуваний UML може бути розгорнутий на декількох платформах з використанням різних технологій з усіма процесами протягом усього циклу розробки ПЗ.

Використовуючи діаграму, можна досягти наступного:

- Чітко відокремити систему від її оточення;
- Визначити дійових осіб, їх взаємодію з системою і очікувану функціональність системи;
- Визначити основні поняття, які стосуються детального опису функціональності системи.

Починати роботу над діаграмою можна з текстового опису, отриманого при роботі з замовником. При цьому опускаються нефункціональні вимоги, до яких відноситься конкретна мова програмування. Для таких вимог складається інший документ.

Перш за все необхідно скласти основну діаграму використання додатку, яка зобразить функціональність додатку. Більшу деталізацію кожної

функції користувачі зможуть побачити нижче. На рис. 2.3 наведено діаграму використання додатку і взаємодію з сутністю «Месенджер». На діаграмі наведено роль користувача, яку він виконує під час користування месенджером. Також наведений прецедент «Аутентифікація», бо без нього неможливі варіанти реєстрації та авторизації. Але через те, що користувач напряму не взаємодіє з ним, його винесено окремо й пов'язано include тільки з двома варіантами.

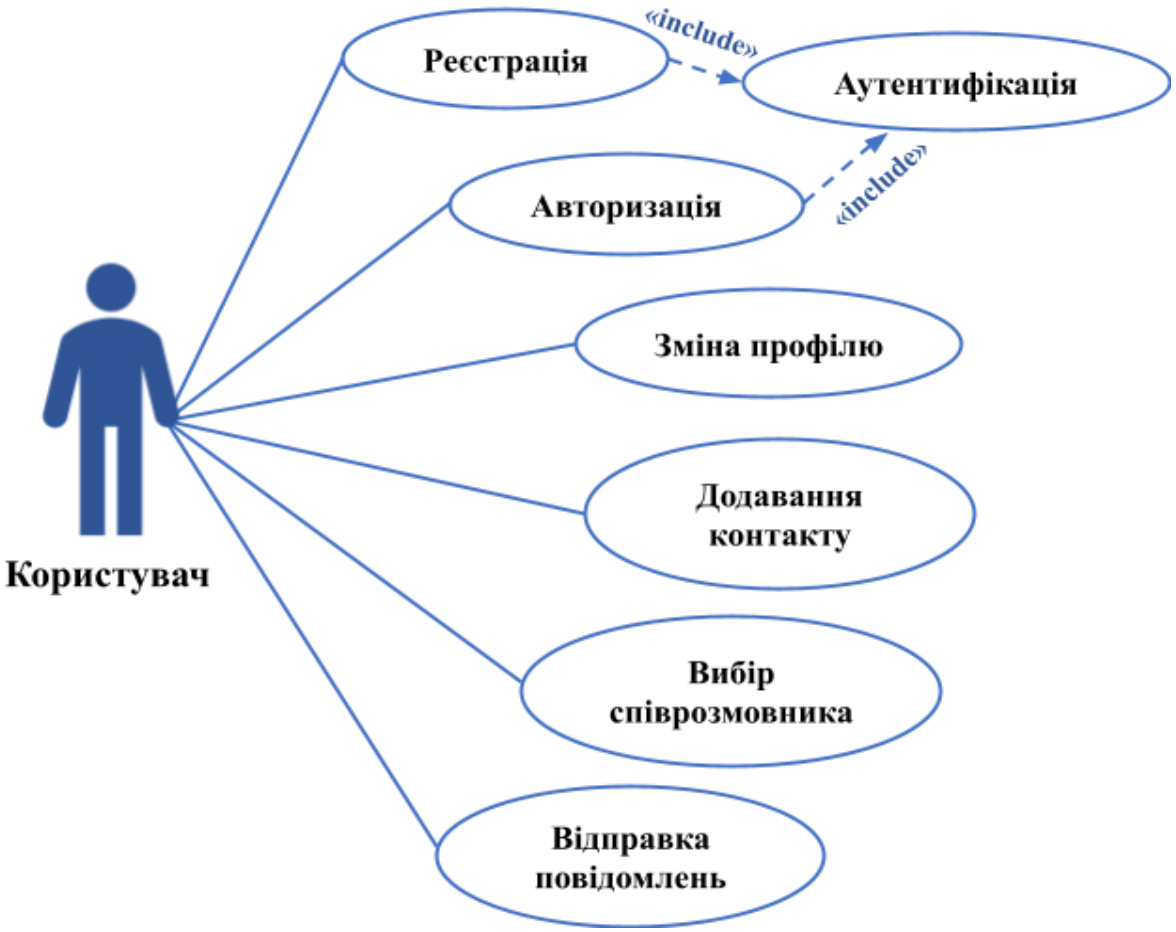

Рис. 2.3. Діаграма з основним функціоналом для користувача месенджера EXPO
Джерело: особистий доробок автора

Роль «Користувач» має наступні можливості:

1) Реєстрація у додатку, яка складається з 5 кроків:
   – Генерація користувачем персонального ID;

– Створення паролю;
– Створення нікнейму;
– Додавання номеру телефона (за бажанням);
– Додавання зображення до профілю.

2) Авторизація, для якої необхідно ввести персональні ID та пароль від акаунту;
3) Зміна профілю, що передбачає можливість змінити зображення профілю та редагуванняналаштувань;
4) Додавання контактів за допомогою клавіатури або сканування ID;
5) Вибір співрозмовника у розділі «Contacts» або серед чинних повідомлень у розділі «Messages»;
6) Відправка текстових повідомлень, фото та відео до свого співрозмовника.

2.3. Розробка архітектури додатку

Додаток, який необхідно створити, можна представити у вигляду 3-рівневої архітектури, як показано на рис. 2.4.

При цій побудові пропозиції сервер розділяється на дві частини — шар доступу до даних і шар логіки. У першому випадку код містить дії з базою даних: створення, видалення, оновлення та ін. У другому — функції які необхідні для клієнта.

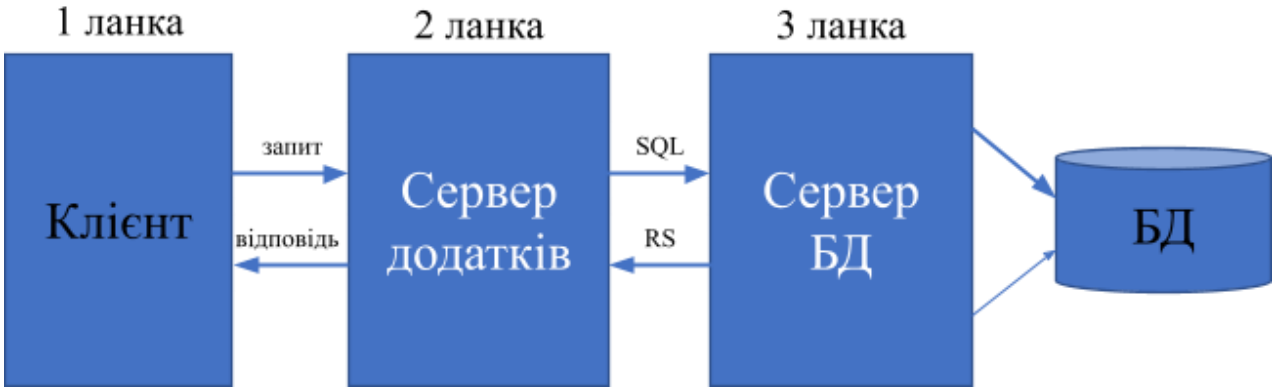

Рисунок 2.4 – 3-рівнева система архітектури додатку

Джерело: сформовано на основі узагальнених досліджень

Даний архітектурний підхід можна характеризувати наступними термінами:

– Клієнт — інтерфейс користувача., зазвичай графічний компонент, який представляє перший рівень, власне застосунок для кінцевого користувача. Перший рівень не повинен мати прямих зв'язків з базою даних для забезпечення безпеки всього додатку, та не повинен бути навантаженим основною бізнес-логікою і зберігати стан програми. На перший рівень виноситься найпростіша бізнес-логіка: інтерфейс авторизації, алгоритми шифрування, перевірка значень, що вводяться, на допустимість і відповідність формату, нескладні операції, наприклад сортування, групування, підрахунок значень.

У трирівневій архітектурі клієнт зазвичай не перевантажений функціями обробки даних, а виконує свою основну роль системи подання інформації, що надходить з сервера додатків. Це зменшує обсяг даних, переданих між клієнтом і сервером додатків. Крім того, клієнтська частина може бути настільки простою, що в більшості випадків її реалізують за допомогою універсального браузера. Але якщо міняти її все-таки доведеться, то цю процедуру можна здійснити швидко і безболісно.

Сервер додатків розташовується на другому рівні. На другому рівні зосереджена велика частина бізнес-логіки. Поза ним залишаються фрагменти, що експортуються у клієнтський додаток.

Сервер бази даних забезпечує зберігання даних і виноситься на третій рівень. Зазвичай це стандартна реляційна або об'єктноорієнтована СУБД. Якщо третій рівень являє собою базу даних разом зі збереженими процедурами, тригерами та схемою, яка описує додатокв термінах реляційної моделі, то другий рівень будується як програмний інтерфейс, що зв'язує клієнтські компоненти з прикладною логікою бази даних.

У простій конфігурації фізично сервер додатків може бути поєднаний з сервером бази даних на одному комп'ютері, до якого через мережу підключається один або кілька терміналів.

У «правильної» (з точки зору безпеки, надійності, масштабування) конфігурації сервер бази даних знаходиться на виділеному комп'ютері (або кластері), до якого через мережу підключені один або кілька серверів додатків, до яких, своєю чергою, через мережу підключаються термінали [18 с. 103].

Переваги 3-рівневої архітектури:

- Розвантаження сервера баз даних від виконання частини операцій, перенесених на сервердодатків;
- Зменшення розміру клієнтських додатків ккоштом розвантаження їх

від зайвого коду;

- Спрощення налаштування клієнтів — при зміні загального коду сервера додатків автоматично змінюється поведінка додатків клієнтів, тобто забезпечується єдина поведінка всіх клієнтів;
- Ще більше збалансоване навантаження на мережу.

З ростом користувачів систем «клієнт-сервер» необхідність 3-рівневої організації стає всебільш очевидною [19].

Недоліки випливають з переваг. У порівнянні з клієнт-серверною або файл-серверною архітектурою можна виділити наступні недоліки 3-рівневої архітектури:

- Вища складність створення додатків;
- Складніше в розгортанні й адмініструванні;
- Високі вимоги до продуктивності серверів додатків і сервера бази даних, а, значить, і висока вартість серверного обладнання;
- Високі вимоги до швидкості каналу (мережі) між сервером бази даних і серверами додатків.

Програмний додаток для ОС Android складається з набору активностей, кожній з яких відповідає вікні керування. Кожна активність представлена в проєкті класом, реалізованому на мовіKotlin, що зберігається в однойменному файлі з розширенням .kotlin. Кожній активності відповідаєxml файл-опис. В xml-файлі описано у вигляді xml-коду розташування візуалізації об'єктів. При запуску активності система Android автоматично розпізнає розмір екрану мобільного пристрою і призводить виведений контент у відповідність з розміткою, описаної в xml-файлі. Таким чином, одна і та ж активність буде виглядати однаково незалежно від діагоналі використовуваного пристрою. Також, для кожної програми Android повинен існувати xml-файл, в якому у вигляді xml-коду будуть прописані мінімальні вимоги до системи, а також активність, яка викликається призапуску додатку [20]

Пропонується наступна логіка роботи програми: клієнт звертається до сховища даних на сервері, відправляючи дані для зберігання в базі даних, а

також запрошуючи необхідні дані з неї. Сервер отримує запити від клієнтів, зберігає і повертає запитані дані клієнтам.

На рис. 2.5 представлена діаграма роботи фрагмента реалізованого додатку, що демонструє посилання повідомлення від клієнта серверу

Рис. 2.5. Покрокова схема роботи фрагмента додатку

Джерело: сформовано на основі узагальнених досліджень

Додаток працює з вбудовуваної реляційної бази даних SQLite. SQLite не використовує парадигму клієнт-сервер, тобто движок SQLite не є окремим процесом, з яким взаємодіє програма, а надає бібліотеку, з якої програма компонується і движок стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси й дані) в єдиному стандартному файлі на тому пристрої, на якому виповнюється програма [21].

2.4. Створення процесів для розробки серверної частини

Серверну частину складають скрипти, написані мовою програмування PHP.

Мова програмування PHP успішно використовується в багатьох великих проектах, включаючи What’s App, Facebook, Stack Overflow. За продуктивністю мова не гірша за інші мови, але її переваги:

- здатність витримувати інтенсивні навантаження;
- звільнення всіх ресурсів обчислювального сервера після виконання завдання;

– доступність розвитку.

Ми використовуємо мову PHP, оскільки вона дозволяє нам ефективно створювати систему обміну повідомленнями без шкоди для її продуктивності.

Node JS. Платформа заснована на популярній і зручній для кодування мові JavaScript, не блокує, потоково передає дані та динамічна. Він добре побудований, що дозволяє створювати високонавантажені системи. Node JS підтримує до 10 000 з’єднань одночасно, тому його рекомендують для сокет-з’єднань, наприклад, для чатів.

У роботі було використано бази даних firebase database, firebase storage, firebase authentication.

База даних Firebase дозволяє створювати багатофункціональні програми для спільної роботи,забезпечуючи безпечний доступ до бази даних безпосередньо з клієнтського коду. Дані зберігаються локально, і навіть в автономному режимі події в реальному часі продовжують запускатися, що дає кінцевому користувачу можливість швидко реагувати. Коли пристрій відновлює з'єднання, база даних реального часу синхронізує зміни локальних даних з віддаленими оновленнями, які відбулися, коли клієнт був в автономному режимі, автоматично об'єднуючи будь-які конфлікти.

Дані Firebase витягуються шляхом приєднання асинхронного прослухування firebase.database.Reference. Слухач запускається один раз для початкового стану даних і знову при кожній зміні даних.

Подивимось, як Firebase виявляється найкращим MBaaS (мобільним бекендом як послуги) у порівнянні з усіма іншими популярними у табл. 2.1.

Таблиця 2.1 − Чому вибрати Firebase як BAAS для мобільного додатка?

Особливості	Firebase	AWS	Кумулос	Метеор	Кінві
Платформа	Забезпечення платформи баз даних у реальному	Платформа хмарних сервісів	Хмарний бекендяк платформа обслуговування	Надання фреймворку веб-додатків	Інтегрован ісерверні сервіси

	часі				
База даних	Забезпечує власний API управління базами даниху режимі реального часу	Служба реляційних баз даних Amazon	База даних SQLite (підключення)	Забезпечує інноваційний API Mini-Mongo	Зберігання даних NoSQL
Ціноутворення	Простіша комплектація	платіжний підхід	фіксована щомісячна плата	На основі масштабу та підтримки	На основі ресурсів, функцій та рівнів
Фронт-енд	Angular, Ember, Vue.jsi React	AWS CodeCommit, CodeDeploy, CodePipeline Node.js, Go,	Інтерфейс програми	Пожежа	Backbone.js
Сторона сервера	Користувачі, лише безпека	Шифрування на стороні сервера	mBaaS забезпечує легшу масштабованість	Що завгодно	бізнес-логіка з меншим кодом
Хостинг	Інтернет, CDN, Користувачі, БД	Amazon EC2, S3, RDS, SQS, DevPay, CloudFront	Хмара Кумулос, підключення	Лише додаток	Бекенд-як-послуга (BAAS)
Організація	Google	AWS (веб-служби Amazon)	ТОВ "Кумулос"	ЦРТ (Група розвитку метеоритів)	Кінві

База даних Firebase надає гнучкий, заснований на висловлюваннях мова правил, званий Firebase Realtime Database Security Rules, для визначення того, як дані повинні бути структуровані й коли вони можуть бути прочитані або записані. При інтеграції з Firebase Authentication розробники можуть визначати, хто має доступ до яких даними і як вони можуть отримати до них доступ. Існує два способи додати Firebase до будь-якої програми Android:

– Використання Firebase Assistant;

– Додавання Firebase вручну.

У цьому способі кроки передбачають:

1) Створіння проєкту Firebase;
   – Створіння проєкту, натиснувши на «Створити проєкт» на консолі Firebase;
   – Заповнення вікна, що спливає, необхідними даними про проєкт;
   – Натискання на «Створити проєкт» для остаточного його створення.
2) Додавання проєкту до програми для Android;
   – Клацання на опцію «Додати Firebase до програми для Android» у вікні запуску;
   – Тепер програма підключена до Firebase, та всі хмарні й серверні сервіси можналегко використовувати в додатку.
3) Щоб отримати сертифікат SHA1 програми, необхідно перейти до Android Studio Project та виконати наступні дії: gradle → коренева папка → Завдання → Android → підпис звіту→ копіювання вставки SHA1 з консолі.
4) Тепер необхідно завантажити файл google-service-json і помістити його в кореневий каталог програми для Android (рис. 2.6)

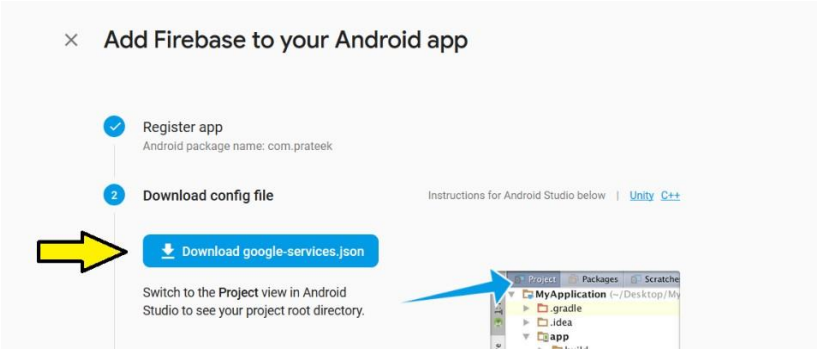

Рисунок 2.6 – Завантаження google-service-json у кореневий каталог програми для Android [22]

5) Додавання в проект наступного:
   – Додавання sdk в проект;

– Додавання наступного коду до PROJECT-LEVELbuild.gradle програми:

```
buildscript {
dependencies {
    classpath 'com.google.gms:google-services:4.0.0'
}
}
```

6) Додавання наступного коду до APP-LEVEL build.gradle програми:

```
dependencies {
compile 'com.google.firebase:firebase-core:16.0.0'
}
...
// Add to the bottom of the file
apply plugin: 'com.google.gms.google-services'
```

7) Синхронізація gradle, клацнувши на «sync now»;
8) Після додавання вищевказаного коду запускаємо програму, щоб надіслати підтвердження на консоль Firebase.
9) Тепер Firebase успішно встановлено.

2.5. Створення процесів для розробки клієнтської частини

В ході розробки додатку необхідно вирішити ряд проблем, серед яких:

– Спрощення комплексних завдань і робочих процесів;
– Донесення складних для розуміння даних в зрозумілому для користувача вигляді;
– Адаптація інформації з урахуванням ролей, потреб та завдань користувача.

Важливими складовими успіху програмних продуктів на ринку програмного забезпечення є вдале проєктування користувацького досвіду (UX) та користувацького інтерфейсу (UI). З метою кращого розуміння слід детальніше розглянути поняття «User Experience» та «User Interface».

Дизайн користувацького досвіду (UX design) — це процес створення продуктів, систем або сервісів, що надають користувачам відповідний та релевантний досвід користування. UX охоплює розробку всього процесу взаємодії з додатком, включаючи такі аспекти як брендинг, дизайн, зручність використання та функціонал [23].

Дизайн користувацького досвіду взаємодії з продуктом дозволяє досягати користувачам їх цілей. Проте UX не зосереджений лише на створенні зручного інтерфейсу, але й охоплює такі аспекти роботи користувачів як:

- Задоволення;
- Ефективність;
- Настрій;
- Розваги.

Узагальнивши вищезазначене, вдалим UX-дизайном можна вважати такий досвід користувача що відповідає конкретним потребам представника цільової аудиторії в визначеному контексті. Іншими словами User Experience Design — це вивчення поведінки користувачів та розуміння їх мотивацій з метою створення кращого досвіду їх взаємодії з продуктом.

Дизайн інтерфейсу для мобільних пристроїв у порівнянні з персональними комп'ютерами має вирішувати ряд додаткових проблем. У цілому, мобільні додатки вдало поєднують переваги зручного формату зі швидким доступом до актуальної інформації.

При завантаженні додатку користувач бажає отримати унікальний досвід який буде виправдовувати витрати трафіку, часу та пам'яті пристрою, відмінний від практики використання мобільної версії сайту. Якщо мобільний додаток не задовольняє очікування, його з легкістю видаляють. За даними fortune.com близько 75% завантажених додатків відкриваються користувачем лише раз. Для досягнення максимально приємного досвіду користування програмним забезпеченням слід дотримуватись таких ключових принципів мобільного UX дизайну [24]:

- Слід уникати перевантаження інтерфейсу;
- Інтуїтивно зрозуміла навігація з врахуванням особливостей операційної системи;
- Єдиний користувацький досвід для різних типів пристроїв;
- Розміщення основних елементів користувацького інтерфейсу у зручних місцях для керування за допомогою пальців;
- Зручність сприйняття контенту, яка досягається шляхом використання вдалих комбінацій гарнітур шрифтів, розділень між текстом, тощо;
- Достатньо видимі елементи інтерфейсу з врахуванням рекомендацій W3C щодо контрастності;
- При проєктуванні основних елементів управління слід враховувати положення рук користувача;
- Мінімізація необхідності ручного введення інформації.

Перед тим, як розробляти додаток, дизайнером було нарисовано всі етапи користувальницького досвіду взаємодії з додатком за допомогою програмного засобу Figma.

Figma – безкоштовний онлайн-інструмент з підтримкою одночасного редагування відразу кількома редакторами. Серед особливостей інструменту доцільно виокремити зображення властивостей елементів макету в вигляді CSS стилів.

Інтерфейс мобільного додатку для операційної системи Android розроблено на основі концепцій дизайн системи Google Material Design 2.0 – стилю дизайну програмного забезпечення розробленого компанією Google. Використання методичних рекомендацій Material Design 2.0. дає змогу зробити дизайн інтерфейсу користувача додатку візуально привабливим, однорідним відносно оформлення операційної системи та врахувати патерни поведінки користувачів.

Окрім того, інтерфейс командної строки Vue-CLI має шаблон для створення прогресивних веб-додатків. Додаток виконано у формі гібридного мобільного додатку з метою економії ресурсів та кросплатформної розробки.

Додаток «EXPO» функціонує на основі клієнт-серверної взаємодії. Для отримання інформації з бібліотеки Firebase, яка виконує роль сервера, та сторонніх API використано HTTP клієнт Retrofit2. Дані на Firebase зберігаються в Google Cloud Storage. Зазвичай на початкових етапах роботи з додатком слід провести навчання для користувача або детальніше проінформувати про проєкт. На перших екранах додатку розмістити інформацію про функціонал та поради щодо швидкого налаштування (рис. Б.1, рис. Б.2, рис. Б.3, рис. В.1).

Варто зазначити, що порада щодо початку роботи з додатком виділена за допомогою акцентного кольору з метою фокусування уваги нових та користувачів, що повернулись. Окрім того, сама порада є активною та при натисканні відбувається перехід на сторінку з налаштуваннямидодатку.

Оскільки API надає доступ до персональних даних зареєстрованих користувачів, це потребує особливої структури запитів, щоб запобігти проблемам безпеки даних.

OAuth оперує системними та користувальницькими запитами. Системні запити — запити, потрібні для функціонування програми, в яку інтегрується OAuth. Такі запити повинні включати передачу отриманих раніше ключів доступу для ідентифікації програми в середовищі IBM.

Для початку роботи з OAuth необхідно:

1) Увімкнути API для проєкту;
2) Створити облікові дані авторизації.

Через те, що в роботі використовується протокол HTTP / REST, в нас нема потреби встановлювати будь-які бібліотеки, щоб мати можливість безпосередньо викликати кінцеві точки OAuth 2.0.

Наступні кроки показують, як наша програма взаємодіє із сервером Google OAuth 2.0 для отримання згоди користувача на виконання запиту API від імені користувача. Наша програма повинна мати цю згоду, перш ніж вона зможе виконати запит Google API, який вимагає авторизаціїкористувача.

У наведеному нижче списку коротко наведено ці кроки:

1) Програма визначає необхідні дозволи;
2) Програма переспрямовує користувача до Google разом зі списком запитуваних дозволів;
3) Користувач вирішує, чи надавати дозволи програмі;
4) Програма з’ясовує, що вирішив користувач;
5) Якщо користувач надав запитувані дозволи, програма отримує маркери, необхідні для надсилання запитів API від імені користувача.

Розглянемо приведені вище кроки більш детально:

Крок 1. Встановлення параметрів авторизації.

Необхідно створити запит на авторизацію. Цей запит встановлює параметри, які ідентифікують програму та визначають дозволи, які користувач повинен буде надати EXPO.

Кінцева точка Google OAuth 2.0 дорівнює https://accounts.google.com/o/oauth2/v2/auth. Ця кінцева точка доступна лише через HTTPS. Звичайні з'єднання HTTP відмовляються.

Крок 2. Перенаправлення на сервер Google OAuth 2.0

Щоб розпочати процес аутентифікації, необхідно перенаправити користувача на серверGoogle OAuth 2.0.

Нижче наведено приклад URL-адреси з розривами рядків та пробілами для читабельності.

```
https://accounts.google.com/o/oauth2/v2/auth?
scope=https%3A/www.googleapis.com/auth/drive.metadata.readonly &
 access_type = офлайн &
include_granted_scopes = true & response_type = код &
стан =state_parameter_passthrough_value& redirect_uri =https% 3A //
oauth2.example.com / code& client_id =client_id
```

Після створення URL-адреси запиту необхідно переспрямувати на нього користувача.

Сервер авторизації Google підтримує параметри рядка запитів для програм веб-сервера,представлені у таблиці 2.2.

Таблиця 2.2 − Параметри рядка запитів для програм веб-сервера [35]

Параметри	
client_id	Необхідно Ідентифікатор клієнта для програми. Це значення можна знайти насторінці облікових даних API Console .
redirect_uri	Необхідно Визначає, куди сервер API перенаправляє користувача після завершення користувачем процесу авторизації. Значення має точно відповідати один з авторизованої URI переадресації для клієнта OAuth 2.0, який був налаштований в API консолі клієнта сторінці облікового дані . Якщо це значення не відповідає дозволеному URI перенаправлення для наданого, client_id, ми отримаємо о redirect_uri_mismatch помилку.
response_type	Необхідно Визначає, чи повертає кінцева точка Google OAuth 2.0 код авторизації. Встановлює для параметра code веб-сервера значення параметра .
scopes	Необхідно Розділений пробілом перелік областей, які визначають ресурси, до яких програма могла отримати доступ від імені користувача. Ці значення повідомляють екран згоди, який Google відображає користувачеві.
access_type	Рекомендовано Вказує, чи може програма оновити маркери доступу, коли користувач відсутній у браузері. Дійсними значеннями параметрів є значення online, яке є значенням за замовчуванням, та offline.

Продовження табл. 2.2.

state	Рекомендовано

	Вказує будь-яке значення рядка, яке програма використовує для підтримання стану між запитом авторизації та відповіддю сервера авторизації. Сервер повертає точне значення, яке надсилається у name=value парі в компоненті запиту URL redirect_uri після того, як користувач погодився або заборонив запит на доступ програми.
include_granted_scopes	Необов'язково Дозволяє програмам використовувати додаткову авторизацію длязапиту доступу до додаткових областей у контексті. Якщо для параметра цього параметра встановлено значення true, а запит авторизації надано, то новий маркер доступу також охоплюватиме всі сфери, до яких користувач раніше надав доступ додатку.
login_hint	Необов'язково Якщо додаток знає, який користувач намагається аутентифікуватися, він може використовувати цей параметр, щоб надати підказку серверу аутентифікації Google. Сервер використовує підказку для спрощення потоку входу, попередньо заповнивши поле електронної пошти у формі входу або вибравши відповідний сеанс багаторазового входу.
prompt	Необов'язково Розділений пробілом, чутливий до регістру список підказок для представлення користувача. Якщо не вказати цей параметр, користувачеві буде запропоновано лише перший раз, коли проект запитує доступ.

Сервер Google OAuth 2.0 аутентифікує користувача та отримує згоду від користувача на нашу програму для доступу до запитуваних областей. Відповідь надсилається до нашої програмиза допомогою вказаної нами URL-адреси переадресації.

Крок 3. Google пропонує користувачеві дати згоду.

На цьому кроці користувач вирішує, чи надати нашій програмі запитуваний доступ. На цьому етапі Google відображає вікно згоди, яке відображає ім'я нашої програми та сервісів API Google, які запитують дозвіл на доступ за допомогою облікових даних авторизації користувача та короткий опис обсягу доступу, який потрібно надати. Потім користувач може

дати згоду на надання доступу до одного або декількох обсягів, запитаних нашою заявкою, або відхилити запит.

На цьому етапі нашій програмі не потрібно нічого робити, оскільки вона чекає відповіді від сервера OAuth 2.0 Google із зазначенням, чи був наданий доступ. Ця відповідь пояснюється на наступному кроці.

Крок 4. Оброблення відповіді сервера OAuth 2.0.

Сервер OAuth 2.0 відповідає на запит на доступ нашої програми, використовуючи URL-адресу, вказану в запиті.

Якщо користувач схвалює запит на доступ, відповідь містить код авторизації. Якщо користувач не схвалює запит, відповідь містить повідомлення про помилку. Код авторизації або повідомлення про помилку, яке повертається на веб-сервер, з'являється у рядку запиту, як показанонижче:

Відповідь на помилку:

```
https://oauth2.example.com/auth?error=access_denied
Відповідь коду авторизації:
https://oauth2.example.com/auth?code=4/P7q7W91a-
oMsCeLvIaQm6bTrgtp7
```

Крок 5. Обмін кодом авторизації для оновлення та доступу до маркерів.

Після отримання веб-сервером коду авторизації він може обміняти код авторизації на маркердоступу.

Наступний фрагмент показує зразок запиту:

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-Type: application/x-www-form-urlencoded
code=4/P7q7W91a-oMsCeLvIaQm6bTrgtp7&
client_id=your_client_id&
client_secret=your_client_secret&
redirect_uri=https%3A//oauth2.example.com/code&
grant_type=authorization_code
```

Щоб обміняти код авторизації на маркер доступу, необхідно зателефонувати https://oauth2.googleapis.com/tokenкінцевій точці та встановити параметри, які зображені у таблиці 2.3.

Таблиця 2.3 − Параметри для обміну коду авторизації на маркер доступу [35]

Поля	
client_id	Ідентифікатор клієнта, отриманий на сторінці облікових даних API Console.
client_secret	Клієнтський секрет, отриманий на сторінці облікових даних API Console.
code	Код авторизації, повернутий із початкового запиту.
grant_type	Для цього поля потрібно встановити значення .authorization_code.
redirect_uri	Один із перенаправлених URI, перелічених для нашого проекту на сторінці Консольних даних облікового запису API для даного .client_id.

Google відповідає на цей запит, повертаючи об’єкт JSON, який містить короткочасний маркер доступу та маркер оновлення.

Відповідь містить поля, зображені у таблиці 2.4.

Таблиця 2.4 − Поля відповіді Google на запит [35]

Поля	
access_token	Токен, який наш додаток надсилає для авторизації запиту Google API.
expires_in	Час життя маркера доступу, що залишився, у секундах.
refresh_token	Токен, який можна використовувати для отримання нового маркера доступу. Токени оновлення дійсні, поки користувач не скасує доступ.
scope	Сфери доступу, що надаються вираженими у вигляді списку рядків з обмеженими пробілами та регістром .access_token
token_type	Тип повернутого маркера. Зараз для цього поля завжди встановлено значення Bearer.

Наступний фрагмент показує зразок відповіді:

```
{ "access_token": "1 / fFAGRNJru1FTz70BzhT3Zg",
"expires_in" : 3920, "token_type" : "Носій", "сфера":
"https://www.googleapis.com/auth/drive.metadata.rea
donly", "refresh_token": " 1 //
xEoDL4iW3cxlI7yDbSRFYNG01kVKM2C-259HOF2aQbI" }
```

У цьому розділі була поставлена задача розробити життєвий цикл розробки програмного забезпечення, з’ясувати специфіку UML діаграм функцій додатку, розкрити сутність створення процесів для розробки серверної та клієнтської частин.

Отже, завдяки візуалізації процесу роботи над проєктом, було покращене розуміння продукту та показано взаємодію користувача з клієнтською частиною. Саме тому використання функціональних UML діаграм заслуговують уваги. Використовуючи їх, можна наочно та зрозуміло розповісти про проєкт для будь-кого із команди, яка займається розробкою продукту.

Було розглянуто та розроблено основний функціонал, який необхідно створити у клієнтському додатку за допомогою функціональних UML діаграм, і який результат отримає користувач по завершенню робіт над створенням додатку.

Наведено приклад традиційної 3-рівневої архітектури. Описано етапи створення серверної та клієнтської частини додатку. Зроблено висновки щодо розробки проєкту.

РОЗДІЛ 3
ОРГАНІЗАЦІЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Структура класів та інформаційних масивів

У найширшому визначенні база даних - це все, що збирає та впорядковує дані. Електронна таблиця з бронюваннями клієнтів є базою даних, а також простим текстовим файлом, що містить дані розкладу рейсів. Самі дані у форматі звичайного тексту можуть зберігатися в різних форматах, включаючи XML і CSV.

Однак професійно, коли йдеться про «базу даних», швидше за все, мається на увазі система керування реляційними базами даних (СУБД). Цей термін може здатися технічним і лякаючим, але СУБД - це просто тип бази даних, яка містить одну або кілька таблиць, які можуть мати відношення один до одного.

Якщо ви очікуєте, що десятки, сотні або тисячі користувачів і додатків будуть використовувати базу даних одночасно, легкі бази даних не збираються скорочувати її. Вам потрібна централізована база даних, яка працює на сервері та ефективно обробляє великий обсяг трафіку. Існує широкий спектр рішень централізованих баз даних на вибір, включаючи наступне:

- MySQL
- Microsoft SQL Server
- Оракул
- PostgreSQL
- Teradata
- IBM DB2
- MariaDB

Деякі з цих рішень можна інсталювати на будь-який комп'ютер і перетворити цей комп'ютер на сервер. Потім ви можете підключити комп'ютери користувачів (також відомі як клієнти) до сервера,

щоб вони могли отримати доступ до даних. Клієнт може відправити SQL-інструкцію із запитом конкретних даних, а сервер обробляє запит і повертає відповідь. Це класична клієнт–серверна настройка. Клієнт щось запитує, а сервер це дає.

Хоча ми можете перетворити будь-який MacBook або дешевий ПК на сервер MySQL, більші обсяги трафіку вимагають більш спеціалізованих комп'ютерів (так званих серверних *комп'ютерів*), оптимізованих для серверних завдань. Вони, як правило, підтримуються IT-відділом, члени якого адмініструють та контролюють бази даних, які формально вважаються критично важливими для бізнесу.

Якщо шукаємо просте рішення для одного користувача або невеликої кількості користувачів (наприклад, ваших колег), легка база даних — гарне місце для початку. Легкі бази даних практично не мають накладних витрат, тобто вони не мають серверів і дуже спритні. Бази даних зазвичай зберігаються у файлі, до якого можна надати спільний доступ іншим користувачам, хоча вони починають ламатися, коли кілька користувачів одночасно редагують файл. У разі виникнення цієї проблеми може знадобитися перехід до централізованої бази даних.

Двома найпоширенішими полегшеними базами даних є SQLite і Microsoft Access. SQLite — це те, що вони безкоштовні, легкі та інтуїтивно зрозумілі у використанні. Вони використовується в більшості пристроїв і їх можна знайти в смартфонах, супутниках, літаках та автомобільних системах. Вони практично не мають обмежень у розмірі і ідеально підходять для середовищ, де його не використовує більше однієї людини (або максимум кілька людей).

Microsoft Access існує вже деякий час і поступається SQLite з точки зору масштабованості та продуктивності. Але вона широко використовується в бізнес-середовищі. Вона має багато візуальних інструментів для написання запитів без використання SQL, а також конструктори візуальних форм та можливості макросів. Існує багато робочих місць, щоб стати власником баз

даних Microsoft Access і підтримувати їх, а також переносити їх на кращі платформи баз даних, такі як MySQL.

У даному проєкті буде використовуватись реляційна база даних SQLite.

Коли є багато місць для розміщення даних, але часто нам потрібне швидке та просте місце для розміщення даних без усіх клопотів із налаштуванням клієнт-серверу. Ми хочемо зберігати дані в простому файлі та редагувати їх так само легко, як документ Word. Це оптимальна ситуація для використання SQLite.

SQLite є найпоширенішою базою даних у світі. Вона ставиться на iPhone, iPad, пристрої Android, телефони Windows, термостати, автомобільні консолі, супутники та багато інших сучасних пристроїв, яким потрібно легко зберігати та отримувати дані. SQLite широко використовується в операційній системі Windows 10, а також у літаку Airbus A350 XWB. Вона перевершує там, де потрібна простота та низькі накладні витрати. Також відмінно підходить для створення прототипів бізнес-баз даних.

Але кожна технологія має компроміс. Оскільки вона не має сервера, який керує доступом до неї, вона зазнає невдачі в багатокористувацьких середовищах, де кілька людей можуть одночасно редагувати файл SQLite.

Для формування більш складних запитів, в мобільному додатку не тільки створені таблиці, але й встановлені зв`язки між ними за допомогою первинних ключів (primary keys).

Завжди потрібно прагнути наявності первинного ключа в будь-якій таблиці. Первинний ключ — це спеціальне поле (або комбінація полів), яке забезпечує унікальний ідентифікатор кожному запису. Первинний ключ часто визначає зв'язок і часто об'єднується. Первинним ключем у таблиці є поле, і так далі. Хоча нам не потрібно призначати поле як первинний ключ, щоб приєднатися до нього, це дозволяє програмному забезпеченню бази даних виконувати запити набагато ефективніше. Він також діє як обмеження для забезпечення цілісності даних. У первинному ключі не допускається

дублікатів, тобто не можна мати два записи з числом два. База даних заборонить це і видасть помилку.

Не варто плутати первинний ключ із зовнішнім. Первинний ключ існує у батьківській таблиці, а зовнішній ключ – у дочірній. Зовнішній ключ у дочірній таблиці вказує на первинний ключ у батьківській таблиці. Наприклад, у таблиці є первинним ключем, а в таблиці – зовнішнім ключем. Вони об'єднані разом для стосунків один до багатьох. На відміну від первинного ключа, зовнішній ключ не забезпечує унікальність, оскільки це «багато» у зв'язку «один-до-багатьох».

Первинний і зовнішній ключі не обов'язково мають однакове ім'я поля. У таблиці знаходиться зовнішній ключ, що вказує на батьківську таблицю. Ім'я поля може відрізнятися на зовнішньому ключі, щоб зробити його більш описовим щодо його використання. У цьому випадку є більш описовим, ніж просто. Семантика суб'єктивна, але все одно легітимна, якщо бізнес-формулювання зрозумілі.

3.2. Сутності SQLite схеми

Одним з основних принципів баз даних SQLite є схема: офіційне оголошення про те, як організована база даних. Схема відображається в операторах SQLite, які використовуються для створення бази даних. Також корисно створити супутній клас, відомий як клас контракту, який чітко вказує макет схеми систематичним та самодокументованим способом.

Клас контракту − це контейнер для констант, що визначають імена для URI, таблиць та стовпців. Клас контракту дозволяє використовувати однакові константи для всіх інших класів в одному пакеті. Це дозволяє змінити назву стовпця в одному місці і поширити його по всьому коду.

Хорошим способом організації класу контракту є розміщення визначень, які є загальними для всієї бази даних, на кореневому рівні класу.

Потім створюється внутрішній клас для кожної таблиці. Кожен внутрішній клас перераховує відповідні стовпці таблиці.

Наприклад, наступний FeedReaderContract визначає назву таблиці та назви стовпців дляоднієї таблиці, що представляє RSS-канал:

```
object FeedReaderContract {//Вміст таблиці згруповано в анонімному об'єкті. object FeedEntry: BaseColumns {const val TABLE_NAME = "entry" const val COLUMN_NAME_TITLE = "title" const val COLUMN _ NAME _ SUBTITLE = "subtitle" } }
```

Кожне значення, яке зберігається в базі даних SQLite, має один з представлених у табл. 3.1 класів елемент зберігання.

Таблиця 3.1 − Класи зберігання SQLite [26]

Назва	Опис
NULL	Значення — значення NULL.
INTEGER	Значення являє собою ціле число зі знаком,збережене в 1, 2, 3, 4, 6 або 8 байтах в залежності від величини значення.
REAL	Значення являє собою значення зплаваючою комою, яке зберігається як 8-байтовое число з плаваючою точкою IEEE.
TEXT	Значення являє собою текстовий рядок, що зберігається з використанням кодування бази даних (UTF-8, UTF-16BE або UTF-16LE)
BLOB	Значення являє собою блок даних, який зберігається точно так же, як він був введений.

У таблиці 3.2 перераховані імена типів даних, які було використано при створенні таблиць SQLite з відповідною застосовною спорідненістю.

Таблиця 3.2 – Подібність SQLite і імена типів [26]

Тип даних	Близькість
• ID	INTEGER
– expoID – token – username – password – StreamDataIntegration – apiKey	TEXT

SQLite підтримує концепцію affinity (близькість) типу до стовпців. Будь-який стовпець може зберігати дані будь-якого типу, але переважний клас зберігання для стовпця називається affinity. Кожному стовпцю таблиці у базі даних SQLite присвоюється один з представлених у таблиці 3.3 типів афінностей:

Таблиця 3.3 – Тип злиття SQLite [26]

Назва	Опис
TEXT	У цьому стовпці зберігаються всі дані з використанням класів зберігання NULL, TEXT або BLOB.
NUMERIC	Цей стовпець може містити значення,використовуючи всі п'ять класів зберігання.
INTEGER	Працює так само, як стовпець з NUMERICспорідненістю, з виключенням в вираженні CAST.
REAL	Поводиться як стовпець з NUMERIC спорідненістю, за винятком того, що він призводить цілі значення в уявлення з плаваючою комою.
NONE	Стовпець з афінністю NONE не любить один клас зберігання над іншим, і нема спроб примусити дані з одного класу зберігання до іншого.

SQLite не має окремого булевського класу зберігання. Замість цього булеві значення зберігаються як цілі числа 0 (брехня) і 1 (істина) (табл. 3.4).

Таблиця 3.4 − Тип даних дати і часу [26]

Назва	Опис
TEXT	Дата у форматі "РРРР-ММ-ДД ЧЧ: ММ: SS.SSS"
REAL	Число днів з полудня за Грінвічем 24листопада 4714 до н.е.
INTEGER	Кількість секунд з 1970-01-01 00:00:00 UTC

SQLite не має окремого класу зберігання для зберігання дат і/або часу, але SQLite здатний зберігати дати і час як значення TEXT, REAL або INTEGER.

3.3. Сутності бази даних SQLite

SQLite підтримує всі функції реляційної бази даних. Для того, щоб отримати доступ до цієї бази даних, не потрібно встановлювати будь-які з'єднання, такі як JDBC, ODBC тощо.

Основним пакетом є android.database.sqlite, який містить класи для управління власними базамиданих.

Для того, щоб створити базу даних, потрібно викликати цей метод openOrCreateDatabase із назвою та режимом бази даних як параметром. Він повертає екземпляр бази даних SQLite, який ми повинні отримати у власному об'єкті, його синтаксис наведено нижче.

```
SQLiteDatabase mydatabase = openOrCreateDatabase
("your database name",MODE_PRIVATE,null);
```

Окрім цього, у пакеті баз даних є інші функції, які виконують цю роботу (табл. 3.5).

Таблиця 3.5 − Функції пакету баз даних SQLite [27]

№	Метод та опис
1	`openDatabase(String path, SQLiteDatabase. CursorFactory factory, int flags,DatabaseErrorHandler errorHandler)` Цей метод відкриває лише існуючу базу даних із відповідним режимом прапора. Режим загальних прапорів може бути OPEN_READWRITE OPEN_READONLY.
2	`openDatabase(String path,SQLiteDatabase.CursorFactory factory, int flags)` Він подібний до вищезазначеного методу, оскільки також відкриває існуючу базуданих, але не визначає жодного обробника для обробки помилок баз даних.
3	`openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)` Він не тільки відкриває, але й створює базу даних, якщо її нема. Цей методеквівалентний методу openDatabase.
4	`openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)` Цей метод подібний до вищезазначеного, але він бере об'єкт File як шлях, а не якрядок. Це еквівалентно file.getPath().

Ми можемо створити таблицю або вставити дані в таблицю, використовуючи метод execSQL, визначений у класі SQLiteDatabase. Його синтаксис наведено нижче:

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS
TutorialsPoint(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialsPoint
VALUES('admin','admin');");
```

Це додасть деякі значення до нашої таблиці в базі даних.

Ми можемо отримати будь-що з бази даних, використовуючи об’єкт класу Cursor. Ми будемо викликати метод цього класу, який називається rawQuery, і він поверне набір результатів із курсором, що вказує на таблицю. Ми можемо рухати курсор вперед і отримувати дані:

```
Cursor resultSet = mydatbase.rawQuery("Select * from
TutorialsPoint",null); resultSet.moveToFirst();
String username = resultSet.getString(0); String
password = resultSet.getString(1);
```

У класі Cursor доступні інші функції, які дозволяють нам ефективно отримувати дані (табл. 3.6).

Для управління всіма операціями, пов'язаними з базою даних, був наданий допоміжний клас, який називається SQLiteOpenHelper. Він автоматично керує створенням та оновленням бази даних. Його синтаксис наведено нижче:

```
public class DBHelper extends SQLiteOpenHelper {
    public DBHelper(){
      super(context,DATABASE_NAME,null,1);
       }
         public void onCreate(SQLiteDatabase db) {}
          public void onUpgrade(SQLiteDatabase database,
        int oldVersion, int newVersion) {}
```

Таблиця 3.6 – Функції класу Cursor [27]

№	Метод та опис
1	`getColumnCount()` Цей метод повертає загальну кількість стовпців таблиці.
2	`getColumnIndex(String columnName)` Цей метод повертає номер індексу стовпця, вказуючи назву стовпця.
3	`getColumnName(int columnIndex)` Цей метод повертає ім'я стовпця, вказуючи індекс стовпця.
4	`getColumnNames()` Цей метод повертає масив усіх імен стовпців таблиці.
5	`getCount()` Цей метод повертає загальну кількість рядків у курсорі.
6	`getPosition()` Цей метод повертає поточне положення курсору в таблиці.
7	`isClosed()` Цей метод повертає значення true, якщо курсор закритий, а в протилежному випадкуповертає значення false.

В ході виконання роботи над створенням розділу ІІІ було визначено, що додаток потребує проектування та створення сутностей й схеми для коректної роботи, а також зберігання даних користувачів для комфортного їх користування додатком. Головну роль серед зазначених питаньграє реляційна база даних SQLite.

Було визначено класи зберігання SQLite, типи злиття та важливі в рамках проєкту подібності й імена типів. Крім того, було зроблено таблиці , які описують функції пакету баз даних та функції класу Cursor.

Отже, SQLite займає найважливіше місце у розробці проекту.

РОЗДІЛ 4
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРОЄКТУ

4.1. Конфігурація бази даних

Для обміну інформацією між користувачами необхідно зберігати дані кожного користувача та інформацію про його місцезнаходження. Здійснення запитів на сервер та обробка відповідей має спрощену структуру завдяки бібліотеці Retrofit2 для Android. Вона дозволяє відправляти GET та POST HTTP запити, інкапсулюючи в собі деталі реалізації цих методів, а самі запити можна поміщати у чергу для обробки.

HTTP − широко поширений протокол передачі даних, спочатку призначений для передачі гіпертекстових документів (тобто документів, які можуть містити посилання, що дозволяють організувати перехід до інших документів).

Абревіатура HTTP розшифровується як HyperText Transfer Protocol, «протокол передачі гіпертексту». Відповідно до специфікації OSI, HTTP є протоколом прикладного (верхнього, 7-го)рівня. Актуальна на даний момент версія протоколу, HTTP 1.1, описана в специфікації RFC 2616.Протокол HTTP припускає використання клієнт-серверної структури передачі даних.

Клієнтську програму формує запит і відправляє його на сервер, після чого серверне програмне забезпечення обробляє цей запит, формує відповідь і передає його назад клієнтові. Після цього клієнтську програму може продовжити відправляти інші запити, які будуть оброблені аналогічним чином.

Завдання, яке традиційно вирішується за допомогою протоколу HTTP — обмін даними між призначеним для користувача додатком, що здійснює доступ до веб-ресурсів (зазвичай це веб-браузер) і веб-серверу. На даний момент саме завдяки протоколу HTTP забезпечується робота Всесвітньої павутини.

Також HTTP часто використовується як протокол передачі інформації для інших протоколів прикладного рівня, таких як SOAP, XML-RPC і WebDAV. У такому випадку говорять, що протокол HTTP використовується як «транспорт». API багатьох програмних продуктів також має на увазі використання HTTP для передачі даних — самі дані при цьому можуть матибудь-який формат, наприклад, XML або JSON.

Як правило, передача даних по протоколу HTTP здійснюється через TCP / IP з'єднання. Серверне програмне забезпечення при цьому зазвичай використовує TCPпорт 80 (і, якщо порт не вказано явно, то зазвичай клієнтське програмне забезпечення за замовчуванням використовуєсаме 80-й порт для відкритих HTTP з'єднань), хоча може використовувати і будь-який інший.

Таким чином, HTTP-запити відправляються на веб-сервер Apache (на якому має бути встановлений PHP) обробляються PHP-скриптами, після чого сервером видається відповідь: в разі успіху — дані, в разі невдачі — код помилки.

Іншою важливою задачею є збереження та пошук інформації в базі даних SQLite. Оскільки в даній системі є такі одиниці даних, як «користувач», «подія», «ділянка» і т. д., і всі вони пов`язані між собою, було обрано систему керування базами даних SQLite, типом збереження даних InnoDB, що підтримує первинні ключі.

Primary Key (Первинний ключ) є полем в таблиці, яке однозначно ідентифікує кожен рядок / запис в таблиці бази даних. Первинні ключі повинні містити унікальні значення. Первинний ключ стовпець не може мати значення NULL. Таблиця може мати тільки один первинний ключ, який може складатися з одного або декількох полів. Коли кілька полів використовуються в якості первинного ключа, їх називають складовим ключем. Якщо таблиця має первинний ключ, визначений на будь-якому полі (ях), то ви не можете мати два записи, які мають однакове значення цього поля (їй).

Саме завдяки первинним ключам є можливість для пошуку по декільком таблицям із використанням відношення Primary Key — Foreign Key.

Створимо автономного REST клієнта. Відповіді генеруються Mock-сервером.Додаємо новий пакет в src/main/java з ім'ям com.vogella.retrofitgerrit.

Додаємо наступні залежності в файл build.gradle:

```
// retrofit
implementation 'com.squareup.retrofit2:retrofit:2.1.0'
implementation 'com.squareup.retrofit2:converter-gson:2.1.0'
// Junit
testImplementation("org.junit.jupiter:junit-jupiter-
api:5.0.0") testRuntime("org.junit.jupiter:junit-jupiter-
engine:5.0.0")
// to run JUnit 3/4 tests:
testImplementation("junit:junit:4.12")
testRuntime("org.junit.vintage:junit-vintage-engine:4.12.0")
```

У JSON відповіді від Gerrit нас цікавить тільки питання про зміни. Тому створимо наступний клас даних в раніше доданому пакеті за замовчуванням:

```
package com.vogella.java.retrofitgerrit;
public class Change {
   String subject;
   public String getSubject() {
      return subject;
       }
        public void setSubject(String subject) {
           this.subject = subject;
             }
                }
```

Визначимо REST API для Retrofit через наступний інтерфейс:

```
package com.vogella.java.retrofitgerrit;
 import java.util.List;
import  retrofit2.Call;
import retrofit2.http.GET;
```

```
import retrofit2.http.Query;
public interface GerritAPI {
     @GET("changes/")
     Call<List<Change>> loadChanges(@Query("q") String
status);
}
```

Створимо наступний клас контролера. Цей клас створює Retrofit клієнт, викликає GerritAPI і обробляє результат (виводить результат виклику в консоль).

```
package com.vogella.java.retrofitgerrit;
import java.util.List;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
public class Controller implements Callback<List<Change>> {
  static final String BASE_URL=https://git.eclipse.org/r/";
  public void start() {
    Gson gson = new GsonBuilder()
      .setLenient()
      .create();
    Retrofit retrofit = new Retrofit.Builder()
      .baseUrl(BASE_URL)
      addConverterFactory(GsonConverterFactory.create(gson))
      .build();
      GerritAPI gerritAPI= retrofit.create(GerritAPI.class);
     Call<List<Change>>call=gerritAPI.loadChanges("status:open");
     call.enqueue(this);
   }
```

```
  @Override
  public void onResponse(Call<List<Change>> call,
  Response<List<Change>> response) {
  if(response.isSuccessful()) {
      List<Change> changesList = response.body(;
      changesList.forEach(change->
       System.out.println(change.subject));
   } else {
      System.out.println(response.errorBody());
   }
 }
 @Override
 public void onFailure(Call<List<Change>> call,Throwable t){
  t.printStackTrace();
 }
}
```

Створимо клас з main-методом для запуску контролера.

```
package com.vogella.java.retrofitgerrit;
public class Main {
   public static void main(String[] args) {
      Controller controller = new Controller();
      controller.start();
  }
}
```

4.2.Конфігурація бібліотеки збереження Room

Room забезпечує абстракційний рівень над SQLite, щоб забезпечити вільний доступ до бази даних, одночасно використовуючи всю потужність SQLite.

Додатки, які обробляють нетривіальні обсяги структурованих даних, можуть отримати велику користь від збереження цих даних локально. Найбільш поширеним варіантом використання є кешування відповідних фрагментів даних. Таким чином, коли пристрій не може отримати доступ до мережі, користувач все ще може переглядати цей вміст, перебуваючи поза мережею. Потім будь-які зміни вмісту, ініційовані користувачем, синхронізуються із сервером після того, як пристрій знову в мережі.

Щоб використовувати Room у додатку, спершу потрібно належним чином налаштувати програму за допомогою набору інструментів Gradle. Перейдемо в Project > Scripts Gradle і відкриємо build.gradle (Module: App) (рис. 4.1)

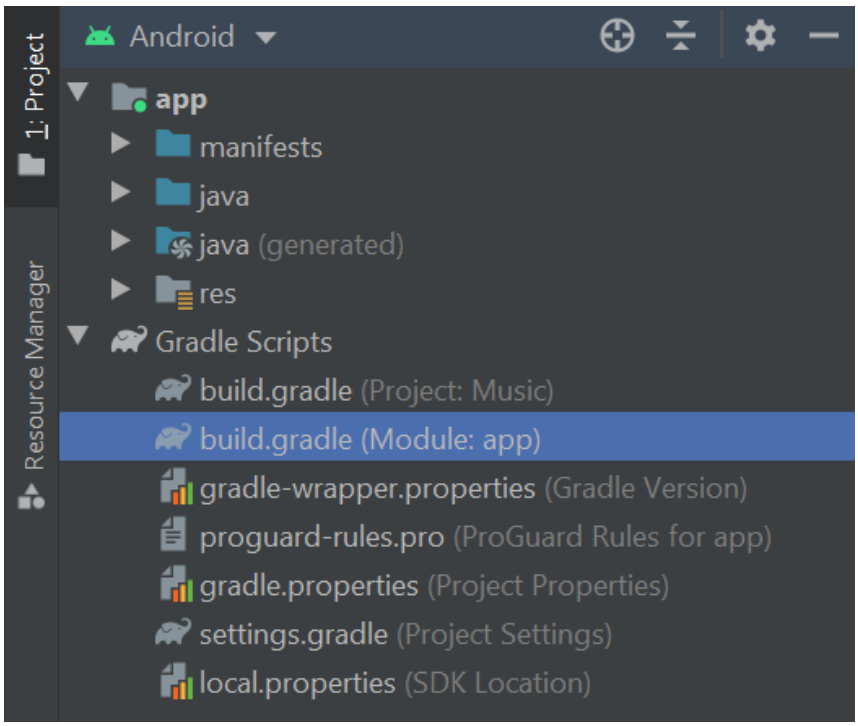

Рисунок 4.1 − Build.gradle (Module: App) [28]

Прокрутивши вниз до розділу залежностей, можна знайти перелік тверджень про «implementation» (серед іншого). Необхідно додати код, представлений на рисунку 4.2, для реалізації бібліотеки Room.

```
def room_version = "2.2.5"

implementation "androidx.room:room-runtime:$room_version"
kapt "androidx.room:room-compiler:$rootProject.roomVersion"

// optional - Kotlin Extensions and Coroutines support for Room
implementation "androidx.room:room-ktx:$room_version"

// optional - RxJava support for Room
implementation "androidx.room:room-rxjava2:$room_version"

// optional - Guava support for Room, including Optional and ListenableFuture
implementation "androidx.room:room-guava:$room_version"

// Test helpers
testImplementation "androidx.room:room-testing:$room_version"
```

Рисунок 4.2 − Код для реалізації бібліотеки Room [28]

Важливо не забути повторно синхронізувати проєкт (якщо буде запропоновано цезробити).

Після успішної синхронізації проєкту можна приступати до побудови бази даних [28].Room має 3 основні компоненти:

1) База даних: містить власника бази даних і служить основною точкою доступу длябазового підключення до постійних реляційних даних додатка. Клас, котрий коментується, @Database повинен відповідати наступним умовам:
    a) Бути абстрактним класом, який розширюється RoomDatabase;
    b) Включати до об’єкта список об’єктів, пов’язаних із базою даних;
    c) Містити абстрактний метод, який має 0 аргументів, і повертає клас, котрий анотовано @Dao. Під час виконання можна отримати екземпляр Database, зателефонувавши Room.databaseBuilder() або Room.inMemoryDatabaseBuilder().
2) Сутність: представляє таблицю в базі даних.
3) DAO: містить методи, що використовуються для доступу до бази даних [29].

Додаток використовує базу даних Room, щоб отримати об'єкти доступу до даних або DAO, пов'язані з цією базою даних. Потім додаток використовує кожен DAO для отримання сутностей із бази даних та збереження будь-яких змін до цих сутностей назад до бази даних. Нарешті,

програма використовує сутність для отримання та встановлення значень, що відповідають стовпцям таблиці в базі даних

У табл. 4.1 представлено фрагмент коду конфігурації бази даних з однією суттю та одним DAO.

Таблиця 4.1 – Фрагмент коду конфігурації бази даних з однією суттю та одним DAO [29]:

Тип даних	Фрагмент коду Kotlin

User	`@Entity`
	`data class`
	`User(`
	`@PrimaryKey`
	`val uid: Int,`
	`@ColumnInfo(name =`
	`"first_name") val firstName:`
	`String?, @ColumnInfo(name =`
	`"last_name") val lastName:`
	`String?`
	`)`

UserDao	`@Dao`
	`interface UserDao {`
	`@Query("SELECT *`
	`FROM user") fun`
	`getAll():`
	`List<User>`
	`@Query("SELECT * FROM user WHERE`
	`uid IN (:userIds)") fun`
	`loadAllByIds(userIds: IntArray):`
	`List<User>`
	`@Query("SELECT * FROM user WHERE`
	`first_name LIKE :first AND " +`
	`"last_name LIKE :last LIMIT 1")`
	`fun findByName(first:`
	`String, last: String):`
	`User @Insert`
	`fun`
	`insertAll(vararg`
	`users: User)`
	`@Delete`
	`fun delete(user: User)`
	`}`

AppDatabase	`@Database(entities =`
	`arrayOf(User::class), version`
	`= 1) abstract class`
	`AppDatabase : RoomDatabase() {`
	`abstract fun userDao(): UserDao`
	`}`

Після створення вищевказаних файлів ми отримуємо екземпляр створеної бази даних,використовуючи наступний код [29]:

```
val db = Room.databaseBuilder( applicationContext,
        AppDatabase::class.java, "database-name"
    ).build()
```

Існує безліч різних маршрутів, якими можна піти під час написання запитів у Room.

Важливо переконатися, що взаємодія додатка з базою даних Room є якомога швидшою та надійнішою. В ідеалі, вміст бази даних повинен бути легко доступний з будь-якої точки програми. Вихідні дані бази даних можуть бути доступні як поточні дані: тип власника даних, який можна динамічно оновлювати [28].

4.3. Конфігурація архітектури

Під час розробки проєкту було бажання виключити повторення написання коду. Для цього було вирішено написати DependenciesLoaderFragment (рис. 4.3).

Для завантаження залежностей у ViewModel у даній програмі ми використовуємо Dagger. Для цього необхідно створити абстрактний клас, від котрого ми зможемо спадкуватися, та в ньому вже буде лежати ViewModelFactory.

Абстрактні класи − це класи, які залишають деякі або всі елементи нереалізованими, щоб реалізації могли надаватися похідними класами [30]. В абстрактному класі також можна визначити поля і методи, але одночасно не можна створити об'єкт або екземпляр абстрактного класу. Абстрактні класи покликані надавати базовий функціонал для класів-спадкоємців. А похідні класи вже реалізують цей функціонал [31].

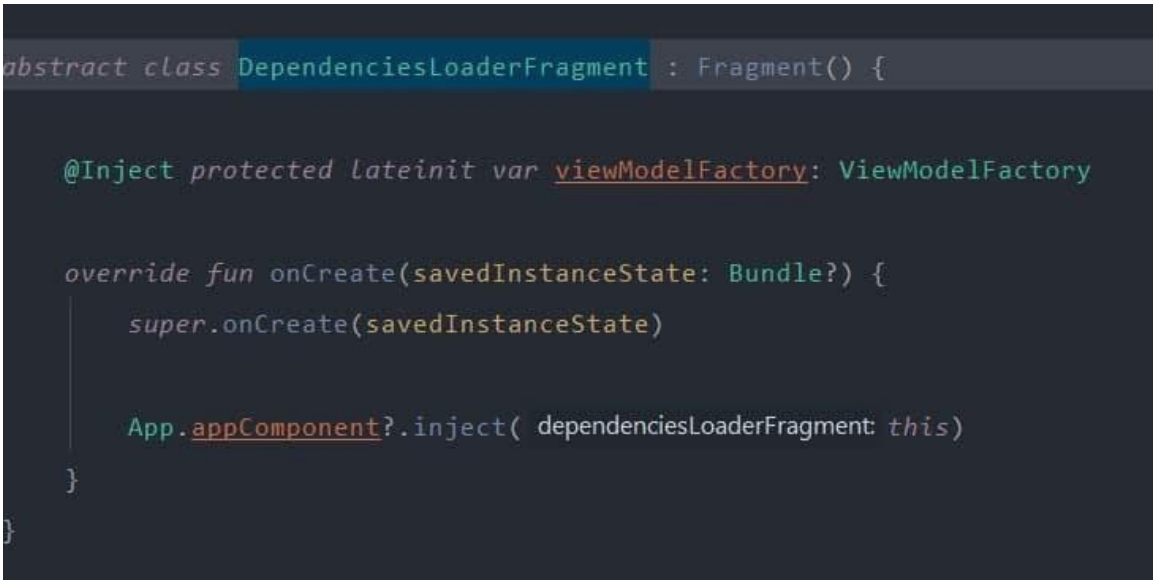

Рисунок 4.3 − DependenciesLoaderFragment
Джерело: особистий доробок автора

Компонент ViewModel складається з таких класів: ViewModel, AndroidViewModel, ViewModelProvider, ViewModelProviders, ViewModelStore, ViewModelStores [32].

Для ініціалізації ViewModel за допомогою Dagger необхідно написати ViewModelFactory (рис. 4.4). ViewModelFactory Клас використовується для створення екземпляра та повернення ViewModel об’єкта, який переживає зміни конфігурації.

Рисунок 4.4 – ViewModelFactory
Джерело: особистий доробок автора

Далі інтегруємо файл ViewBinding для кожного фрагменту (рис. 4.5, рис. 4.5). View Binding − це інструмент, який дозволяє простіше писати код для взаємодії з view. При включенні View Binding в певному модулі він генерує binding-класи для кожного файлу розмітки (layout) в модулі. Об'єкт згенерованого binding класу містить посилання на всі view з файлу розмітки, для яких вказано android:id.

Рисунок 4.5 − Інтеграція файлу ViewBinding для кожного фрагменту
Джерело: особистий доробок автора

Кожен згенерований binding-клас містить посилання на кореневий view розмітки (root) і посилання на всі view, які мають id. Ім'я генерується класу формується як «назва файлу розмітки», перекладене в camel case + «Binding».

Наприклад, для файлу розмітки result_profile.xml:

```
<LinearLayout ... >
    <TextView android:id="@+id/name" />
    <ImageView android:cropToPadding="true" />
    <Button  android:id="@+id/button"
        android:background="@drawable/rounded_button" />
</LinearLayout>
```

Згенерує клас `ResultProfileBinding`, що містить два поля: `TextView namei Button button`. Для `ImageView` нічого згенеровано не буде, бо воно не має id. Також в класі `ResultProfileBinding` буде метод `getRoot()`, який повертає кореневий LinearLayout.

Щоб створити об'єкт класу `ResultProfileBinding`, треба викликати статичний метод `inflate()`. Після цього можна використовувати кореневої view як content view в Activity:

```
private lateinit var binding: ResultProfileBinding
@Override
fun onCreate(savedInstanceState: Bundle) {
   super.onCreate(savedInstanceState)
   binding = ResultProfileBinding.inflate(layoutInflater)
   setContentView(binding.root)
}
```

Пізніше binding можна використовувати для отримання view:

```
binding.name.text = viewModel.name
binding.button.setOnClickListener { viewModel.userClicked()
```

```
override fun onResume() {
    super.onResume()

    ToolbarManager(builder(), activity?.actionBar!!, layoutInflater).prepareToolbar()
}

protected abstract fun builder() : FragmentToolbar

override fun onDestroyView() {
    super.onDestroyView()
    _binding = null
}
```

}

Рисунок 4.6 – Інтеграція файлу ViewBinding для кожного фрагменту

Джерело: особистий доробок автора

Для написання ViewModelFactory було використано базовий клас, котрий дозволяє спадкуватися від кожного, й вставляти в аргументи будь-який об’єкт ViewBinding (рис. 4.7).

```
override fun bind(): SearchContactFragmentBinding {
    return SearchContactFragmentBinding.inflate(layoutInflater)
}
```

Рисунок 4.7 − Базовий клас, котрий дозволяє спадкуватися від кожного, й вставляти в аргументи будь-який об’єкт ViewBinding [33]

Щоб поєднати залежності у ViewModel, BaseFragment спадкується від DependenciesLoaderFragment. Таким чином, ми одночасно завантажуємо Toolbar, довантажуємо ViewBinding й завантажуємо ViewModelFactory.

Важливо надати особливості BaseFragment:

1) Організувати інтерфейси делегування;
2) Впровадження інтерфейсів делегатів у фрагменті;
3) Реалізація делегатів Kotlin.

Зараз у нас є невеликий та потужний базовий фрагмент з безліччю служб та функцій, якими спадкоємці можуть користуватися безпосередньо, не записуючи назви властивостей [34].

4.4. Конфігурація Toolbar

Елемент Toolbar призначений для швидкого і зручного доступу користувача до часто використовуваних функцій. Створити його можна використовуючи як спрощений варіант, в якому багато про що вже подбали розробники системи Android, так і повністю керуючи усіма внутрішніми компонентами.

В головному вікні з детальною інформацією про користувача нам необхідно створити Toolbar, зображений на рис. 4.8.

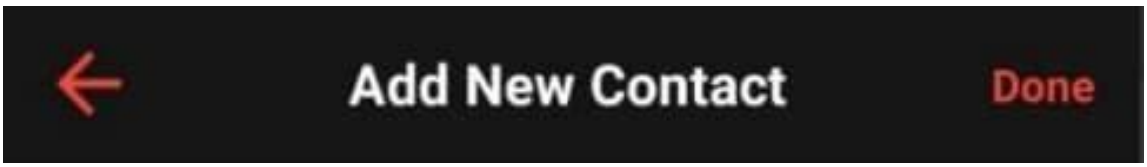

Рисунок 4.8 − Створення Toolbar

Тут у нас знаходиться тільки один елемент: кнопка пошуку, яка повинна перенаправляти нас на екран для пошуку інших користувачів.

У ранніх версіях Android використовувався елемент ActionBar, тепер же його функцію виконує Toolbar. Важливо, використовувати Toolbar з пакетjv android.support.v7.widget, щоб у нас була сумісність зі старими пристроями (версія Android нижче 5.0).

Тому спершу нам необхідно подбати про те, щоб наші екрани не містили елемент ActionBar за замовчуванням. Для цього нам потрібно успадкувати головний стиль додатку (знаходиться в файлі styles.xml) від необхідного нам Theme.AppCompat.Light.NoActionBar:

```
<resources>
 <!-- Base application theme. -->
 <style name="AppTheme"
 parent="Theme.AppCompat.Light.NoActionBar">
  <!-- Customize your theme here. -->
  <item name="colorPrimary">@color/colorPrimary</item>
  <item
  name="colorPrimaryDark">@color/colorPrimaryDark</item>
  <item name="colorAccent">@color/colorAccent</item>
 </style>
 <!-- Остальные элементы ниже не изменились -->
</resources>
```

Тепер необхідно додати елемент Toolbar в xml-файл activity_user_info.xml. Для цього додамо Toolbar над контейнером

RelativeLayout, в якому знаходиться вся інформація про користувача. Також додамо стиль для Toolbar, щоб перевикористати його на інших екранах.

Activity_user_info.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">
  <android.support.v7.widget.Toolbar
     android:id="@+id/toolbar" style="@style/Toolbar"/>
    <RelativeLayout>
      <!-- Весь контент информації про користувача -->
    </RelativeLayout>
</LinearLayout>
```

styles.xml:

```
<resources>
  <!-- Остальные элементы выше не изменились -->
  <style name="Toolbar">
    <item name="android:layout_width">match_parent</item>
    <item name="android:layout_height">?attr/actionBarSize</item>
    <!-- <item name="theme">@style/
    ThemeOverlay.AppCompat.Dark.ActionBar</item> -->
    <item name="android:background">@color/colorPrimary</item>
  </style>
  <!-- Остальные элементы ниже не изменились -->
</resources>
```

Для того, щоб кастомізувати Toolbar та для простоти змін було використано патерн Builder (рис. 4.9). Він дозволяє гнучко налаштувати Toolbar, додавши лише ті опції, які потрібні на даному екрані.

```
override fun builder(): FragmentToolbar {
    return FragmentToolbar.Builder()
        .setShown()
        .withTitle("Add New Contact")
        .showBackButton()
        .showActionButton()
        .setButtonsListener(this)
        .build()
}
```

Рис. 4.9. Патерн Builder для кастомізації Toolbar

Патерн Builder відокремлює алгоритм поетапного конструювання складного продукту (об'єкта) від його зовнішнього уявлення так, що за допомогою одного і того ж алгоритму можна отримувати різні уявлення цього продукту.

Поетапне створення продукту означає його побудову по частинах. Після того як побудована остання частина, продукт можна використовувати.

Для цього патерн Builder визначає алгоритм поетапного створення продукту в спеціальному класі Director (розпорядник), а відповідальність за координацію процесу складання окремих частин продукту покладає на ієрархію класів Builder. У цій ієрархії базовий клас Builder оголошує інтерфейси для побудови окремих частин продукту, а відповідні підкласи ConcreteBuilder їх реалізують відповідним чином, наприклад, створюють або отримують потрібні ресурси, зберігають проміжні результати, контролюють результати виконання операцій.

У цьому розділі було розглянуто створення основних фрагментів у базі даних SQLite, таких як:

- Retrofit2, що використовується для мережевої взаємодії месенджера. За його допомогою REST API перетворюється в інтерфейс Java. Він використовує анотації для опису HTTP-запитів, за замовчуванням замінює параметр URL і підтримку параметрів запиту. Крім того, він забезпечує функціональні можливості для копіювання багатосторінкового запиту і завантаження файлів
- Room, що забезпечує абстракційний рівень над SQLite для забезпечення вільного доступу до бази даних, одночасно використовуючи всю потужність SQLite.
- Елемент Toolbar призначений для швидкого і зручного доступу користувача до часто використовуваних функцій.

За допомогою розглянутих у розділі інструментів процес розробки додатку стане більш простим і надійним.

Розроблений додаток за допомогою SQLite додатку можна використовувати у командній розробці, оскільки усі необхідні процеси з налагодження клієнтського та серверного додатку впроваджені.

ВИСНОВКИ

В результаті виконання даної дипломної роботи було розроблено месенджер на базі операційної системи Android.

У першому розділі була поставлена задача проаналізувати чинні програмні комплекси, схарактеризувати цілеспрямованість створення власного або корпоративного месенджера та схарактеризувати вхідну й вихідну інформацію щодо проєкту даної роботи.

Комплексний аналіз множин схожих між собою месенджерів допоміг зробити висновок, що у своєї більшості вони мають однакові між собою можливості та функції для користувачів. Цей аналіз допоміг зробити композицію всіх найкращих функцій та відокремити відповідні недоліки, яких треба уникати.

Також було детально описано вхідну та вихідну інформацію додатку, на які буде опиратись проєктування функціональних діаграм за допомогою UML та серверна архітектура у наступному розділі.

До вхідної інформації проєкту відносяться бібліотека GetStream, яка використовується длявласної імплементації месенджера, мова програмування Kotlin, що буде використовуватись для створення клієнтської та серверної частини додатку, та бібліотека Retrofit2, що використовується для мережевої взаємодії месенджера. Як обгортку над стандартними механізмами, яка покликана впорядкувати й спростити прості та складні шаблони навігації в додатку, обрано Navigation Architecture Component. Важливими елементами внутрішньої інформації месенджера також було зазначено Firebase, Dagger, RxJava та Room.

У другому розділі була поставлена задача розробити життєвий цикл розробки програмного забезпечення, з’ясувати специфіку UML діаграм функцій додатку, розкрити сутність створення процесів для розробки серверної та клієнтської частин.

Завдяки візуалізації процесу роботи над проєктом, було покращене розуміння продукту та показано взаємодію користувача з клієнтською

частиною. Саме тому використання функціональних UML діаграм заслуговують уваги. Використовуючи їх, можна наочно та зрозуміло розповісти про проєкт для будь-кого із команди, яка займається розробкою продукту.

Було розглянуто та розроблено основний функціонал, який необхідно створити у клієнтському додатку за допомогою функціональних UML діаграм, і який результат отримає користувач по завершенню робіт над створенням додатку.

Наведено приклад традиційної трьохрівневої архітектури. Описано етапи створення серверної та клієнтської частини додатку. Зроблено висновки щодо розробки проєкту.

У третьому розділі було визначено, що додаток потребує проектування та створення сутностей й схеми для коректної роботи, а також зберігання даних користувачів для комфортного їх користування додатком. Головну роль серед зазначених питань грає реляційна база даних SQLite.

Було визначено класи зберігання SQLite, типи злиття та важливі в рамках проєкту подібності й імена типів. Крім того, було зроблено таблиці , які описують функції пакету баз даних та функції класу Cursor.

У четвертому розділі було розглянуто створення основних фрагментів у базі даних SQLite, серед яких є Retrofit2, Room та Toolbar. За допомогою розглянутих у розділі інструментів процес розробки додатку стане більш простим і надійним.

Розроблений додаток за допомогою SQLite додатку можна використовувати у командній розробці, оскільки усі необхідні процеси з налагодження клієнтського та серверного додатку впроваджені.

Отриманий досвід проєктування та розробки продукту повного циклу знадобиться у майбутній практиці як програміста.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційний сайт GetStream [Електронний ресурс].− Режим доступу: https://getstream.io.
2. Go testing at Stream [Електронний ресурс]. — Режим доступу: https://medium.com/@getstream_io.
3. Kotlin vs Java. В чем отличие и что лучше? [Електронний ресурс]. — Режим доступу: https://blog.mo-apps.com/razrabotka/kotlin-vs-java-v-chem-otlichie-i-chto-luchshe.
4. Код приложения для Android-приложений с Picasso [Електронний ресурс]. — Режимдоступу: https://code.tutsplus.com/ru/tutorials/code-an-image-gallery-android-app-with-picasso--cms-30966.
5. ZXing [Електронний ресурс]. — Режим доступу: https://opensource.google/projects/zxing.
6. How to Use Zxing, Android QR Code Scanner Library, to Read QR Code? [Електронний ресурс]. — Режим доступу: https://www.spaceotechnologies.com/qr-code-android-using-zxing-library/.
7. Используем Retrofit 2 в Android-приложении [Електронний ресурс]. — Режим доступу: https://devcolibri.com/getting-started-with-retrofit-in-android/.
8. Навигация в приложении. Начало работы с Navigation Architecture Component [Електронний ресурс]. — Режим доступу: https://www.fandroid.info/12-navigation-architecture-component/.
9. Предоставляем доступ к Google Firebase через свой сервер [Електронний ресурс]. — Режим доступу: https://cutt.ly/BhxXVjb.
10. Dagger basics [Електронний ресурс]. — Режим доступу: https://developer.android.com/training/dependency-injection/dagger-basics.
11. Введение в RxJava: Почему Rx? [Електронний ресурс]. — Режим доступу: https://habr.com/ru/post/269417/.

12. Room: Хранение данных на Android для всех и каждого [Електронний ресурс]. — Режим доступу: https://habr.com/ru/post/336196/.
13. Mayur Pandey LLVM Cookbook / Mayur Pandey, Suyog Sarda. - Mumbai, 2006. — 296 p.
14. Алексенко О. В. Технологія створення програмних продуктів [Електронний ресурс].Режим доступу: https://dl.sumdu.edu.ua/textbooks/109154/414628/index.html.
15. Автоматизированные системы. Стадии создания. ГОСТ 34.601-90. — [Чинний від 1991-01-01] — 10 с. — (Міждержавний стандарт).
16. Виды, комплектность и обозначение документов при создании автоматизированных систем. ГОСТ 34.201-89. — [Чинний від 1990-01-01] — 8 с. — (Міждержавний стандарт).
17. Зеленський О.С. Інструментальні засоби прикладного програмування: навчальний посібник. У двох частинах. Частина І / О.С. Зеленський, В.С. Лисенко, І.Є. Афанасьєв. Кривий Ріг: КЕІ, 2012. — 268 с.
18. Ананий В. Алгоритмы: введение в разработку и анализ / В. Ананий. — М.: «Вильямс»,2006. — 576 с.
19. Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности: учеб. пособие / Г.Н. Федорова [Електронний ресурс]. — Режим доступу: https://cutt.ly/jhnNtz4.
20. Варакін М.В. Разработка мобильных приложений под Android. УЦ «Спеціаліст» при МДТУ ім. Н. Е. Баумана, 2012.
21. John Wiley & Sons. Reto Meier Professional Android 4 Application Development. Wrox, 2012.
22. Adding Firebase to Android App [Електронний ресурс]. — Режим доступу: https://www.geeksforgeeks.org/adding-firebase-to-android-app/.
23. UI/UX Design services [Електронний ресурс] — Режим доступу: https://syndicode.com/uiux-design/.
24. Korkishko I. 8 key principles of mobile UX design [Електронний ресурс] /

Iryna Korkishko. — 2019. — Режим доступу: https://syndicode.com/2019/02/12/8-key-principles-of-mobile-ux-design/.

25. Організація баз даних: Методичні вказівки до самостійної роботи студентів за спеціальностями 6.050102/123 «Комп'ютерна інженерія», 125 «Кібербезпека» / уклад. В.В. Сидоренко, Л.В. Константинова— Кропивницький: ЦНТУ, 2017. — 88 с.
26. SQLite — Типы данных [Електронний ресурс]. — https://unetway.com/tutorial/sqlite-type-data.
27. Android — SQLite Database [Електронний ресурс]. — https://www.tutorialspoint.com/android/android_sqlite_database.htm.
28. How to store and manage data using an in-app Room SQLite database [Електронний ресурс] — Режим доступу: https://codersguidebook.com/how-to-create-an-android-app/store-data-room-sqlite-database-android.
29. Save data in a local database using Room [Електронний ресурс]. — Режим доступу: https://developer.android.com/training/data-storage/room.
30. Абстрактные классы [Електронний ресурс]. — Режим доступу: https://docs.microsoft.com/ru-ru/dotnet/fsharp/language-reference/abstract-classes.
31. Абстрактные классы. [Електронний ресурс]. — Режим доступу: https://metanit.com/java/tutorial/3.6.php.
32. Android Architecture Components. Часть 4. ViewModel [Електронний ресурс]. — Режимдоступу: https://habr.com/ru/post/334942/.
33. Долгожданный View Binding в Android [Електронний ресурс]. — Режим доступу:https://habr.com/ru/post/467295/.
34. Refactor your BaseFragment class [Електронний ресурс]. — Режим доступу: https://programmerr47.medium.com/refactor-your-basefragment-class-d6f721decc85.
35. Using OAuth 2.0 for Web Server Applications [Електронний ресурс]. — Режим доступу: https://developers.google.com/identity/ protocols/oauth2/

web-server#libraries.

36. Number of monthly active Telegram users worldwide from March 2014 to April 2020 (in millions) [Електронний ресурс]. — Режим доступу: https://www.statista.com/statistics/234038/telegram-messenger-mau-users/.
37. Most popular global mobile messenger apps as of April 2018, based on number of monthly active users (in millions) [Електронний ресурс]. — Режим доступу: https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/.