МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук,
управління та адміністрування
Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Створення Android-застосунку для оптимізації пошуку осіб за визначеними критеріями

Виконав студент 2 курсу групи МІС-21
спеціальності 122 Комп'ютерні науки
Дерін Кирило Олександрович

Керівник к.т.н., доцент
Фразе-Фразенко Олексій Олексійович

Рецензент к.т.н., начальник ЦІТ ОНЕУ
Домаскін О.М.

Одеса 2022

АНОТАЦІЯ

на магістерську кваліфікаційну роботу

«Створення Android-застосунку для оптимізації пошуку осіб за визначеними критеріями»

студента Деріна Кирила Олександровича

Актуальність магістерської роботи зумовлена швидким та повсюдним поширенням мобільних розумних пристроїв на споживчому ринку, яке змусило спільноту розробників програмного забезпечення швидко адаптувати підходи до розробки, усвідомлюючи нові можливості мобільних додатків. Поєднання обчислювальної потужності, доступу до нових вбудованих датчиків і простоти передачі додатків на ринок зробили мобільні пристрої новою обчислювальною платформою для компаній і незалежних розробників. Однак зростання цієї нової обчислювальної платформи випереджає роботу з розробки програмного забезпечення, спрямовану на розробку мобільних додатків.

Метою дослідження є отримання теоретичних та практичних знань у галузі розробки мобільних застосунків.

Об'єктом дослідження є процес розробки застосунків для операційної системи Android. Предметом дослідження є технічні та організаційні аспекти розробки застосунків для операційної системи Android.

В роботі було проведено дослідження методологій розробки мобільних додатків, вивчено документацію сучасних інструментів розробки та виконано програмну реалізацію програми для оптимізації пошуку осіб за визначеними критеріями на базі Android.

Ключові слова: ANDROID-ЗАСТОСУНОК, МОБІЛЬНА РОЗРОБКА, FIREBASE, ANDROID STUDIO, ДОШКА ОГОЛОШЕНЬ

Магістерська робота містить 82 сторінки, 39 рисунків, 30 посилань.

SUMMARY

for master's thesis

"Creating an Android application to optimize the search for people according to specified criteria"

of student Derin Kyrylo Oleksandrovych

The relevance of the master's thesis is due to the rapid and widespread spread of mobile smart devices in the consumer market, which forced the software development community to quickly adapt development approaches, realizing the new possibilities of mobile applications. The combination of computing power, access to new built-in sensors, and the ease of bringing applications to market have made mobile devices the new computing platform for companies and independent developers. However, the growth of this new computing platform is outpacing software development efforts aimed at mobile application development.

The purpose of the study is to obtain theoretical and practical knowledge in the field of mobile application development.

The object is the process of developing applications for the Android operating system. The subject of the study is the technical and organizational aspects of developing applications for the Android operating system.

In the work, a study of mobile application development methodologies was conducted, the documentation of modern development tools was studied, and the software implementation of the program was performed to optimize the search for people according to the specified criteria based on Android.

Keywords: ANDROID APPLICATION, MOBILE DEVELOPMENT, FIREBASE, ANDROID STUDIO, JOB BOARD

The master's thesis contains 82 pages, 39 figures, 30 references.

ЗМІСТ

ВСТУП

Швидке та повсюдне поширення мобільних розумних пристроїв на споживчому ринку змусило спільноту розробників програмного забезпечення швидко адаптувати підходи до розробки, усвідомлюючи нові можливості мобільних додатків.

Поєднання обчислювальної потужності, доступу до нових вбудованих датчиків і простоти передачі додатків на ринок зробили мобільні пристрої новою обчислювальною платформою для компаній і незалежних розробників. Однак зростання цієї нової обчислювальної платформи випереджає роботу з розробки програмного забезпечення, спрямовану на розробку мобільних додатків.

Метою дослідження є отримання теоретичних та практичних знань у галузі розробки мобільних застосунків.

Під час виконання кваліфікаційної роботи було поставлено такі завдання:

1. Визначитись з методологією розробки мобільних додатків.
2. Вивчити документацію сучасних інструментів розробки.
3. Застосувати отримані знання під час розробки програми.

Об'єктом дослідження є процес розробки застосунків для операційної системи Android. Предметом дослідження є технічні та організаційні аспекти розробки застосунків для операційної системи Android.

У застосунок буде інтегровано Firebase Authentication – сервіс для автентифікації користувача, та Firebase Realtime Database – сервіс для створення бази даних у реальному часі.

Для розробки Android-застосунку буде використано інтегроване середовище розробки Android Studio від Google.

За допомогою програмного забезпечення для створення макетів Figma буде реалізовано макети, які у подальшому буде перенесено у Android Studio.

1 АНАЛІЗ МЕТОДОЛОГІЇ РОЗРОБКИ ANDROID-ЗАСТОСУНКІВ

Три найпопулярніші методології розробки мобільних додатків – це Waterfall, Agile та SDLC.

Методологія Waterfall — це послідовний процес проектування, у якому розвиток неухильно йде вниз, як водоспад. Він починається від концепції до закриття проекту [1].

Методологія Agile працює за принципом швидкості та повторюваності. Завдання розбиті на короткі етапи роботи, часте оцінювання та адаптацію до планів.

Agile виявилася ефективною та корисною для середовища розробки мобільних додатків. Вона відповідним чином відповідає цим характеристикам, оскільки є більш гнучкою, у той час як традиційні методи мають дуже мало можливостей для змін.

Оскільки Agile є більш адаптивною методологією, вона допомагає створювати безперебійні, швидкі, невеликі за розміром застосунки, з якими легко працювати. Гнучка розробка робить застосунок більш стабільним, з меншою кількістю помилок, що підвищує якість [2].

Методологія SDLC(Software Development Life Cycle) — це спосіб вимірювання та вдосконалення процесу розробки. Методологія дозволяє детально аналізувати кожен етап процесу. Це, у свою чергу, допомагає розробникам максимізувати ефективність на кожному етапі. Зі збільшенням обчислювальної потужності зростає вимога до програмного забезпечення та розробників. SDLC допомагає досягти цих цілей, виявляючи неефективність і вищі витрати та виправляючи їх для безперебійної роботи.

SDLC визначає 8 кроків розробки програмного забезпечення – це:

1. Планування.
2. Вимоги.
3. Проектування.
4. Створення.

5. Документування.

6. Тестування.

7. Розгортання.

8. Обслуговування [3] .

Для створення Android-застосунку Infomage було використано методологію SLDC.

1.1 Проблеми та виклики при розробці застосунка для смартфону

Смартфони є обчислювальною платформою, що розвивається найшвидше, кількість користувачів мобільних пристроїв оцінюється в 1,6 мільярда.

Це стрімке поширення мобільних пристроїв за останні п’ять років різко змінило платформу, яка використовується для соціальних, бізнес-розваг, ігор, продуктивності та маркетингу за допомогою програмних додатків.

Мобільні пристрої, що містять датчики глобального позиціонування, бездротове підключення, можливості фото/відео, вбудовані веб-браузери, розпізнавання голосу та інші датчики, дозволили розробити мобільні застосунки, які можуть надавати багатий, високолокалізований, контекстно-залежний вміст для користувачів у кишенькових пристроях, оснащених такою ж обчислювальною потужністю, як і стандартний ПК.

Поєднання обчислювальної потужності, доступу до нових вбудованих датчиків і простоти монетизації додатків і їх передачі на ринок зробили мобільні додатки новою IT-платформою для розробки. Проте швидке поширення мобільних пристроїв і застосунків випередило підходи до розробки програмного забезпечення, призначені для розробки програмного забезпечення для мобільних застосунків.

Тим не менш, ті самі нові функції, знайдені в мобільних пристроях, створюють нові виклики та вимоги до розробників застосунків, які не зустрічаються у традиційних програмних додатках [4].

Традиційні підходи розробки програмного забезпечення можуть не застосовуватися безпосередньо в контексті мобільних пристроїв.

По-перше, користувацькі інтерфейси(UI) мобільних пристроїв забезпечують нову парадигму для нових послідовностей взаємодії між людиною та комп'ютером (наприклад, мультисенсорні інтерфейси, сканування QR-кодів, розпізнавання зображень, доповнена реальність тощо), які раніше не розглядалися в дослідженнях та для яких не існує жодних встановлених рекомендацій щодо UI.

По-друге, різні платформи та виробники мобільних телефонів і планшетів змусили розробників створити серію однакових застосунків, адаптованих для кожного типу пристроїв.

По-третє, новизна справді мобільної обчислювальної платформи надає як унікальні можливості, так і проблеми.

На основі трьох факторів, нових для розробки мобільних додатків, було окреслено наступні фундаментальні, унікальні виклики сучасній практиці розробки програмного забезпечення для мобільних застосунків [5].

1.1.1 Створення універсальних користувацьких інтерфейсів

Було проведено дослідження щодо створення універсального UI для мобільних пристроїв. Розробка UI для мобільних пристроїв має враховувати розмір і роздільну здатність екрана.

Наприклад, пристрої Apple обмежені двома розмірами залежно від розміру iPhone і iPad, а Windows та Android пропонують екрани різних розмірів і роздільної здатності. Як наслідок, дизайн UI є складним і розробники мобільних додатків повинні передбачити цільовий пристрій.

Дослідження, свідчать про те, що чотири вказівки легко застосовуватимуться на мобільних пристроях, зокрема:

- дозволити користувачам використовувати ярлики;
- пропонувати інформативний зворотний зв'язок;

- розробляти діалогові вікна для закриття;
- підтримувати внутрішній локус контролю.

Оскільки ці виклики продовжують розвиватися, подальші дослідження мають зосередитися на оптимізації зусиль щодо розробки додатків незалежно від мобільної платформи чи пристрою. Значні зусилля повинні бути спрямовані на передбачення різноманітності можливостей користувача, інтерфейсів і методів введення користувачами [6].

1.1.2. Повторне використання програмного забезпечення на мобільних платформах

Мобільні програми наразі охоплюють кілька різних операційних систем, різних виробників апаратного забезпеченн, способи використання обчислювальних платформ.

Кожен із цих параметрів необхідно враховувати під час розробки мобільних застосунків, оскільки вони безпосередньо впливають на вимоги до програмного забезпечення. Наразі компаніям потрібно прийняти бізнес-рішення та націлитися на одну платформу мобільного пристрою з багатьма функціями або витратити ресурси, необхідні для широкого націлювання на діапазон мобільних пристроїв із багатьма власними програмами.

Якщо компанії не будуть націлені на одну платформу, розробники можуть вирішити створити єдину програму для всіх платформ із ризиком деяких функціональних невідповідностей або натомість розглянути можливість створення кількох версій, націлених на кожну апаратну/обчислювальну платформу.

У цьому середовищі розробки багато компаній мають окремі команди розробників або окремий контракт на розробку для різних платформ (наприклад, iOS та Android), що фактично подвоює зусилля з розробки програмного забезпечення, необхідні для функціонально схожих мобільних застосунків.

Навіть коли розробка координується між командами розробників, орієнтованими на різні платформи, вона часто відбувається на основі спеціального плану без узгоджених зусиль, щоб скоротити час і вартість розробки за допомогою існуючих методологій інженерії програмного забезпечення, які використовують повторно.

Нещодавні спроби адаптувати HTML5 за допомогою таких інструментів, як PhoneGap, спрямовані на скорочення зусиль розробки для створення майже нативних програм на кількох платформах шляхом рендерингу нативних інтерфейсів програм через веб-перегляди.

Однак цей підхід не дозволяє використовувати багато функцій, які мають доступ до API мобільного пристрою, і є технологічним рішенням, а це є бажаним підходом до розробки програмного забезпечення для повторного використання ранніх ресурсів розробки програмного забезпечення [7].

1.1.3. Проектування контекстно-залежних мобільних застосунків

Мобільні пристрої представляють собою різкий відхід від традиційних обчислювальних платформ, оскільки вони більше не представляють «статичне поняття контексту, де зміни відсутні, незначні або передбачувані».

Швидше, мобільні пристрої дуже персоналізовані та повинні постійно контролювати своє оточення, таким чином роблячи застосунки за своєю суттю контекстно обізнаними. Мобільні програми тепер контекстуалізують близькість, місцезнаходження, погоду, час тощо, щоб надавати користувачам гіперспеціалізований, динамічний, насичений вміст за допомогою контекстно-залежних програм.

Раніше веб-програми часто надавали контекстуалізований вміст на основі часу, виявленого місцезнаходження та мови. Однак ступінь усвідомлення контексту, можливий на даний момент у мобільних застосунках, виходить за межі того, з чим стикаються підходи програмної інженерії за межами агентно-орієнтованої програмної інженерії.

Розгляд усвідомлення контексту як першокласної функції в розробці програмного забезпечення для мобільних додатків необхідний, щоб розробники приділяли необхідну увагу під час аналізу цих вимог, що призвело б до створення краще розроблених контекстно-залежних додатків. Баланс між гнучкістю та невизначеністю у вимогах.

У той час як більшість розробників мобільних додатків використовують гнучкий підхід або майже спеціальний підхід, зростаючий попит на контекстно-залежні додатки, конкуренція серед мобільних додатків і низька толерантність користувачів до нестабільних і/або невідповідних мобільних додатків (навіть безкоштовних) вимагає більшого напівформальний підхід. Це має бути інтегровано в гнучку інженерію, щоб визначити та проаналізувати вимоги до мобільних застосунків.

Динамічний, контекстний характер вмісту мобільних програм допускає ситуації, в яких поведінка програми може не повністю задовольнити визначені функціональні та нефункціональні вимоги, що вимагає, щоб програма була самоадаптивною. У цьому сценарії програмне забезпечення надаватиме менш насичений вміст, який задовольнятиме менш суворі вимоги.

Для деяких мобільних програм це може виникнути, якщо, як визначено у вимогах, краще, щоб програма працювала безперервно та, за потреби, автономно змінювала свою поведінку та забезпечувала обмежену функціональність, а не забезпечувала її повну відсутність.

Наприклад, у додатку на основі визначення місцезнаходження кілька факторів можуть впливати на деталізацію та актуальність її вмісту. У деяких програмах на основі розташування може бути краще надати старий вміст (тобто вміст, заснований на попередньому розташуванні), а не відображати повідомлення про помилку, інакше програма буде працювати повільно або взагалі не відповість.

У розробці програмного забезпечення для мобільних операційних систем потреба в самоадаптації програми залежно від контексту була створена за допомогою спеціальних підходів. Проте, оскільки мобільні додатки стають

більш обізнаними про контекст, самоадаптивні вимоги потрібно буде більш формально інтегрувати в гнучку розробку, щоб поведінці програми приділялося більше уваги, коли її повні вимоги не можуть бути задоволені динамічно, і як вона може самостійно адаптуватися, щоб частково задовольнити вимоги [8].

1.1.4. Використання сучасних підходів до розробки програмного забезпечення

Для підтримки зниження витрат на розробку функціонально подібних мобільних додатків на кількох платформах розробка програмного забезпечення для мобільних додатків повинна активно використовувати існуючі підходи до розробки програмного забезпечення, що свідомо повторного використання, як-от розробка лінійки програмних продуктів (SPLE).

SPLE підтримує повторне використання, розробляючи набір додатків, що мають загальний контрольований набір вимог, і є перевагою, оскільки використовує потенціал повторного використання в аналізі та розробці. Лінійка програмних продуктів — набір додатків, розроблених компанією, які мають загальний набір основних вимог, але відрізняються між собою набором змінних вимог.

Цей підхід може скоротити час і витрати, необхідні для розробки програмного забезпечення, і може розглядатися як найбільш успішний підхід до повторного використання програмного забезпечення всередині організації. Фаза розробки домену визначає вимоги для всієї лінійки продуктів, а фаза розробки додатків повторно використовує їх для розробки конкретних програм у межах лінійки продуктів.

Цей підхід може бути придатним для розробки програмного забезпечення для мобільних операційних систем, оскільки він заохочуватиме

розробників активно зосереджуватися на загальних вимогах, дизайні, ресурсах тощо.

Інтеграція SPLE у розробку програмного забезпечення для мобільних операційних систем спонукає розробників оцінювати вимоги до програми незалежно від платформи та зосереджуватися на тому, що може бути спільним для всіх версій програми. Це також перемістить процес розробки програмного забезпечення для мобільних додатків на розробку вимог до додатків наперед, а не доручить проектування та розробку різним, можливо, незалежним групам/контрактам розробників, які можуть/можуть не координувати свої зусилля. Дослідницькі зусилля мають бути спрямовані на те, як SPLE можна спеціально пристосувати для розробки програмного забезпечення мобільних додатків, щоб уникнути дублювання ранньої роботи з розробки програмного забезпечення та/або активів [9].

1.1.5. Нефункціональні вимоги є критичними для мобільних додатків

Деяким мобільним додаткам може знадобитися динамічна самоадаптація, щоб забезпечити обмежену функціональність. Для кращої підтримки динамізму в мобільних додатках в результаті усвідомлення контексту та дизайну для самоадаптації розробка програмного забезпечення для мобільних додатків повинна адаптувати існуючі підходи до специфікації вимог до самоадаптивних систем, такі як RELAX [10].

Для кожної варіантної вимоги процес RELAX документує, які зміни середовища можуть вплинути на вимогу та як вимогу можна частково задовольнити. Цей підхід розширює традиційний вираз вимоги до включення ключових слів, включаючи якомога раніше, якомога ближче до, зрештою, якомога більше тощо, щоб задокументувати невизначеність і те, як програма може адаптуватися в умовах невизначеності до забезпечити певну функціональність.

Адаптація RELAX до розробки програмного забезпечення для мобільних додатків спрямує розробників до розгляду того, як програма може адаптуватися, коли середовище або її поведінка є неоптимальними [11].

1.2 Проектування користувацького інтерфейсу та досвіду

На етапі планування було визначено, що необхідно проаналізувати методології проектування користувацького інтерфейсу/досвіду для того, щоб застосувати ці методології при розробці користувацького інтерфейсу та користувацького досвіду Android-застосунку.

1.2.1 Використання патернів проектування користувацького інтерфейсу

Патерни допомагають в проектуванні користувацького інтерфейсу. Вони необхідні для коректного відображення різноманітних даних – візуальних або текстових.

Виділяються наступні патерни:

- Data Tips (випливаючі дані) – визначає взаємодію користувача з елементами UI, які зберігають приховані дані (підказки);
- Data Spotlight (підсвічування даних) – визначає взаємодію користувача з областю, що його цікавить;
- Dynamic Queries (динамічні запити) – використовується, коли необхідно вивести великий багатовимірний набір даних будь-якої форми і будь-яким способом представлення.

Патерн Data Tips використовується при виведенні пристроєм загального подання даних практично в будь-якій формі.

Патерн пропонує такі дії під час реалізації підказок для користувача:

- обдумування розташування вікна;
- створення тимчасового вікна, яке буде виводитись після наведення на елемент UI;

- форматування даних за їхньою структурою.

Патерн Data Spotlight підсвічує певний зріз даних та затіняє решту.

Якщо уявлення містить так багато інформації, що навіть структура даних стає неочевидною, то користувачеві складно помітити взаємини та відстежити зв'язки між наборами даних.

Якщо сама структура даних дуже складна - вони можуть включати кілька незалежних змінних та заплутані зрізи залежних даних, таких як лінії, області, розкидані точки, системи зв'язків.

Підсвічування даних допомагає роз'єднати нитки даних. Це один із способів реалізації фокусу у контексті складної інфографіки.

Спосіб реалізації патерну Dynamic Queries залежить від представлення даних, від того, які запити будуть вводити користувача, а також від можливостей розробки.

Просторові методики Dynamic Queries:

- повзунки для вибору числа з певного діапазону;
- подвійні повзунки або пари повзунків для визначення підмножини: виведення точок даних більше першого значення, але менше другого;
- перемикачі або комбіновані поля для вибору однієї з кількох можливих значень;
- прапорці для вибору довільного підмножини значень, змінних або шарів даних;
- текстові поля для введення окремих значень (пошук).

Також є визначення для організації роботи патернів, яка містить 3 рівні: контекст, сценарій та використання [12].

1.2.2 Інформаційна архітектура як елемент користувацького досвіду

Інформаційна архітектура (ІА) відповідає за структуру контенту всередині цифрового продукту. Основний фокус ІА спрямовано на

організацію контенту та забезпечення його доступності через мітки, карту сайту, навігацію тощо.

Компоненти інформаційної архітектури:

- організація - те, як структурована та впорядкована інформація;
- маркування - те, як представлена інформація;
- система навігації - те, як користувачі переглядають або отримують доступ до інформації;
- пошук - те, як користувачі шукають інформацію.

Процес проектування інформаційної архітектури:

1. Збір та аналіз даних - отримання даних через дослідження, юзабіліті-тестів та застосування методів проектування, орієнтованих на користувача.

2. Зворотне проектування потоку навігації - структуризація контенту методо зворотного проектування: створення опорних точок для ключових елементів. Кожна частина сайту має бути ретельно продумана та представлена у вигляді логічного потоку навігації.

3. Тестування та доопрацювання - створення/доопрацювання макета, тестування взаємодії елементів.

Принципи інформаційної архітектури:

- об'єктування – визначення, з якими об'єктами працює розробник, а також як вони функціонують, коли користувач взаємодіє з ними;
- розкриття інформації - розробник повинен давати користувачам достатньо візуальних та текстових підказок, щоб користувачі могли навчатися у процесі взаємодії з програмою;
- зразки – показ того, що саме побачать користувачі, якщо виберуть той чи інший пункт чи категорію;
- вхідні двері – кожен екран має бути оформлений таким чином, щоб користувачі одразу розуміли, де вони знаходяться і куди їм потрібно йти;
- множинна класифікація - краще передбачити різні способи переміщення програмним забезпеченням та взаємодії з ним [13].

1.3 Можливості операційної системи Android

Android — це стек програмного забезпечення для мобільних пристроїв, який включає операційну систему, допоміжне програмне забезпечення та ключові програми.

Різні компоненти Android створені як стек, причому програми формують верхній рівень стеку, а ядро Linux утворює нижній рівень.

Android поставляється з набором основних програм, включаючи клієнт електронної пошти, програму для SMS, календар, карти, браузер, контакти та інші функції. Більшість програм написано на мові програмування Java.

Розробники мають повний доступ до тих самих інтерфейсів API, які використовуються основними додатками. Архітектура додатків розроблена для спрощення повторного використання компонентів.

Можливості будь-якої програми можуть бути опубліковані, а потім використані будь-якою іншою програмою (з урахуванням обмежень безпеки, встановлених інфраструктурою).

Цей самий механізм дозволяє користувачеві замінювати компоненти. Наприклад, якщо у користувача є невелика програма для створення нотаток на мобільному телефоні та він хоче знайти певне місце, адресу якого він щойно записав, він може скористатися програмою карт безпосередньо з програми для створення нотаток, а не перемикатися між програмами [14].

Android містить набір бібліотек C/C++, які використовуються різними компонентами системи Android. Ці можливості надаються розробникам через інфраструктуру програм Android.

Android SDK – це набір для розробки програмного забезпечення, що включає повний набір засобів розробки. До них відносяться налагоджувач, бібліотеки, емулятор телефону на основі QEMU, документація, зразок коду та навчальні посібники.

Наразі Android SDK включає у собі:

- Android SDK Platform Tools;

- Android Debug Bridge;
- Fastboot;
- Android NDK;
- Android Open Accessory Development Kit.

Платформерні інструменти Android SDK — це окремо завантажувана підмножина повного SDK, що складається з інструментів командного рядка, таких як adb і fastboot.

Android Debug Bridge (ADB) — це інструмент для запуску команд на підключеному пристрої Android. Утиліта adbd запускається на пристрої, а клієнт adb запускає фоновий сервер для мультиплексування команд, надісланих на пристрої. На додаток до інтерфейсу командного рядка існують численні графічні інтерфейси користувача для керування adb.

Fastboot — це протокол, і він містить інструмент із такою ж назвою, що входить до складу пакета Android SDK, який використовується переважно для зміни файлової системи флеш-пам'яті через USB-з'єднання з головного комп'ютера. Це вимагає, щоб пристрій запускався в режимі завантажувача або вторинного програмного завантажувача, у якому виконується лише найпростіша апаратна ініціалізація [15].

Після ввімкнення протоколу на самому пристрої він прийматиме певний набір команд, надісланих йому через USB за допомогою командного рядка. Деякі з найбільш часто використовуваних команд швидкого завантаження включають:

- flash – перезапис розділу із двійковим образом, що зберігається на головному комп'ютері;
- reboot – перезавантаження пристрою в основну операційну систему, розділ відновлення системи або назад у завантажувач;
- device – відображає список усіх пристроїв (із серійним номером), підключених до головного комп'ютера;
- format – форматування певного розділу, файлова система розділу повинна розпізнаватися пристроєм.

Код, написаний на C/C++, можна скомпілювати до власного коду ARM або x86 (або їх 64-розрядних варіантів) за допомогою Android Native Development Kit (NDK). NDK використовує компілятор Clang для компіляції C/C++.

Власні бібліотеки можна викликати з коду Java, що виконується в Android Runtime, за допомогою System.loadLibrary, частини стандартних класів Android Java. Інструменти командного рядка можна зібрати за допомогою NDK і встановити за допомогою adb.

Android використовує Bionic як бібліотеку C, а LLVM libc++ як стандартну бібліотеку C++. NDK також містить низку інших API: стиснення zlib, графіку OpenGL ES або Vulkan, аудіо OpenSL ES і різноманітні специфічні API для Android для таких речей, як журналювання, доступ до камер або прискорення нейронних мереж.

NDK включає підтримку CMake і власну збірку ndk (на основі GNU Make). Android Studio підтримує запуск будь-якого з них із Gradle. Інші інструменти сторонніх виробників дозволяють інтегрувати NDK в Eclipse і Visual Studio.

Коли пристрій на базі Android перебуває в режимі аксесуара, підключений аксесуар діє як USB-хост (живить шину та нумерує пристрої), а пристрій на базі Android діє як пристрій USB. USB-аксесуари Android спеціально розроблені для під’єднання до пристроїв на базі Android і дотримання простого протоколу (протокол аксесуарів Android), який дозволяє їм виявляти пристрої на базі Android, які підтримують режим аксесуарів [16].

1.3.1 Робота з даними та файлами у Android

Android використовує файлову систему, подібну до дискових файлових систем на інших платформах. Система надає кілька варіантів збереження даних програми:

1. Спеціальне сховище програми – зберігання файлів, які призначені лише для використання програмою, у спеціальних каталогах внутрішньої пам’яті або в інших виділених каталогах зовнішньої пам’яті. Використання каталогів внутрішньої пам’яті для збереження конфіденційної інформації, до якої інші програми не мають доступу.

2. Спільне сховище – зберігання файлів, якими програма має намір поділитися з іншими програмами, включно з медіафайлами, документами та іншими файлами.

3. Параметри – зберігання приватних простих даних у парах ключ-значення.

4. Бази даних – зберігання структурованих даних у приватній базі даних за допомогою бібліотеки збереження Room.

Внутрішнє сховище має обмежений простір для даних програми. Якщо розробнику потрібно зберегти значний обсяг даних, йому потрібно використовувати інші типи сховища.

Якщо базові функції програми потребують певних даних, наприклад, коли програма запускається, найкраще розмістити дані у каталозі внутрішньої пам’яті чи базі даних. Файли програми, які зберігаються у зовнішній пам’яті, не завжди доступні, оскільки деякі пристрої дозволяють користувачам видалити фізичний пристрій, який відповідає зовнішній пам’яті.

Android пропонує два типи фізичних сховищ: внутрішню та зовнішню пам’ять. На більшості пристроїв внутрішня пам’ять менша, ніж зовнішня. Однак внутрішня пам’ять завжди доступна на всіх пристроях, що робить її більш надійним місцем для зберігання даних, від яких залежить програма [17].

1.3.2 Реалізація дозволів у Android

Дозволи програми допомагають підтримувати конфіденційність користувачів, захищаючи доступ до:

- обмежених даних, таких як стан системи та контактна інформація користувачів;
- обмежених дій, таких як підключення до сполученого пристрою та запис аудіо.

Якщо програма пропонує функції, які можуть вимагати доступу до обмежених даних або обмежених дій, розробник повинен визначити, чи може користувач отримати інформацію або виконати дії без необхідності оголошувати дозволи.

Дозволи можуть відповідати багатьом випадкам використання в додатку, наприклад фотографування, відтворення медіафайлів і відображання релевантної реклами.

Якщо розробник вирішить, що програма повинна мати доступ до обмежених даних або виконувати обмежені дії для виконання сценарію використання, він повинен оголосити відповідні дозволи.

Деякі дозволи, відомі як дозволи під час встановлення, автоматично надаються під час встановлення програми. Інші дозволи, відомі як дозволи під час виконання, вимагають, щоб програма пішла далі й запитала дозвіл під час виконання.

Android класифікує дозволи на різні типи, зокрема дозволи під час встановлення, дозволи під час виконання та спеціальні дозволи. Кожен тип дозволу вказує на обсяг обмежених даних, до яких програма може отримати доступ, і на обсяг обмежених дій, які програма може виконувати, коли система надає програмі такий дозвіл.

Дозволи під час встановлення надають програмі обмежений доступ до обмежених даних або дозволяють програмі виконувати обмежені дії, які мінімально впливають на систему чи інші програми.

Коли в додатку оголошено дозволи під час встановлення, магазин додатків надає користувачеві повідомлення про дозвіл під час встановлення, коли він переглядає сторінку з інформацією про додаток [18].

Система автоматично надає програмі дозволи, коли користувач встановлює програму. Програми, які реалізують привілейовані служби, такі як автозаповнення або служби VPN, також використовують дозволи підпису. Ці програми вимагають дозволів підпису зв’язування служби, щоб лише система могла зв’язуватися зі службами.

Дозволи під час виконання, також відомі як небезпечні дозволи, надають програмі додатковий доступ до обмежених даних або дозволяють програмі виконувати обмежені дії, які більш суттєво впливають на систему та інші програми.

Тому розробнику потрібно запитати дозволи на час виконання в додатку, перш ніж він зможе отримати доступ до обмежених даних або виконувати обмежені дії. Ці дозволи не надавалися раніше — розробник має перевіряти їх і, якщо потрібно, запитувати їх перед кожним доступом. Коли програма запитує дозвіл на виконання, система показує запит на дозвіл на виконання. Дозволи програми базуються на функціях безпеки системи та допомагають Android підтримувати такі цілі, пов’язані з конфіденційністю користувачів:

- контроль - користувач має контроль над даними, якими він ділиться з програмами;
- прозорість - користувач розуміє, які дані використовує програма та чому програма отримує доступ до цих даних;
- мінімізація даних - програма отримує доступ і використовує лише ті дані, які потрібні для певного завдання, яке викликає користувач.

Багато дозволів виконання мають доступ до приватних даних користувача, особливого типу обмежених даних, які містять потенційно конфіденційну інформацію [19].

1.3.3 Реалізація та фільтрація намірів у Android

Намір — це об’єкт обміну повідомленнями, який можна використовувати для запиту дії від іншого компонента програми.

Хоча наміри полегшують зв’язок між компонентами декількома способами, існує три основних випадки використання:

- запуск діяльності;
- запуск служби;
- ведення трансляції.

Є два типи намірів:

1. Явні наміри - вказують, яка програма задовольнить намір, надаючи назву пакета цільової програми або повне ім’я класу компонента.

2. Неявні наміри - не називають конкретний компонент, а натомість оголошують загальну дію для виконання, яка дозволяє компоненту з іншої програми обробляти її [20].

Якщо намір відповідає фільтру намірів, система запускає цей компонент і доставляє йому об’єкт Intent. Якщо кілька фільтрів намірів сумісні, система відображає діалогове вікно, щоб користувач міг вибрати, яку програму використовувати.

Фільтр намірів — це вираз у файлі маніфесту програми, який визначає тип намірів, які компонент хоче отримати. Наприклад, оголошуючи фільтр намірів для дії, розробник дозволяє іншим програмам безпосередньо починати діяльність із певним типом намірів.

Об’єкт Intent містить інформацію, яку система Android використовує, щоб визначити, який компонент запускати, а також інформацію, яку компонент-одержувач використовує для належного виконання дії.

Неявний намір визначає дію, яка може викликати будь-яку програму на пристрої, здатну виконати дію. Використання неявного наміру корисно, коли програма не може виконати дію, але інші програми, ймовірно, можуть, і розробник хоче, щоб користувач обирав, яку програму використовувати.

Наприклад, якщо розробник має вміст, яким він хоче, щоб користувач поділився з іншими людьми, він створить намір за допомогою дії ACTION_SEND і додасть додаткові функції, які визначають вміст для спільного використання [21].

1.3.4 Реалізація служб у Android

Служба — це компонент програми, який може виконувати тривалі операції у фоновому режимі. Вона не забезпечує інтерфейс користувача. Після запуску служба може продовжувати працювати протягом деякого часу, навіть після того, як користувач перейде до іншої програми. Крім того, компонент може прив’язуватися до служби, щоб взаємодіяти з нею та навіть здійснювати міжпроцесний зв’язок.

Наприклад, служба може обробляти мережеві транзакції, відтворювати музику, виконувати введення/виведення файлів або взаємодіяти з постачальником контенту, і все це у фоновому режимі.

Є три різні види служб:

1. Передньопланова служба — виконує певну операцію, яка помітна для користувача. Наприклад, аудіопрограма використовуватиме службу переднього плану для відтворення звукової доріжки. Основні служби повинні відображати сповіщення. Основні служби продовжують працювати, навіть якщо користувач не взаємодіє з програмою.

2. Фонова служба — виконує операцію, яку користувач безпосередньо не помічає. Наприклад, якщо програма використовувала службу для ущільнення свого сховища, зазвичай це буде фонова служба.

3. Прив’язана служба — служба прив’язується, коли компонент програми зв’язується з нею шляхом виклику bindService(). Прив’язана служба пропонує інтерфейс клієнт-сервер, який дозволяє компонентам взаємодіяти зі службою, надсилати запити, отримувати результати та навіть робити це між процесами за допомогою міжпроцесного зв’язку. Прив’язана служба працює лише до тих пір, поки до неї прив’язано інший компонент програми. Кілька компонентів можуть прив’язуватися до служби одночасно, але коли всі вони від’єднуються, служба знищується.

Щоб створити службу, розробник повинен створити підклас служби або використати один із існуючих підкласів [22].

У реалізації розробник повинен перевизначити деякі методи зворотного виклику, які обробляють ключові аспекти життєвого циклу служби, і надати механізм, який дозволяє компонентам прив’язуватися до служби, якщо це необхідно. Це найважливіші методи зворотного виклику, які розробник повинен перевизначити:

1. onStartCommand() — система викликає цей метод, викликаючи startService(), коли інший компонент запитує запуск служби. Коли цей метод виконується, служба запускається і може працювати у фоновому режимі необмежений час. Щоб зупинити службу після завершення її роботи, необхідно викликати stopSelf() або stopService().

2. onBind() — система викликає цей метод, викликаючи bindService(), коли інший компонент хоче зв’язатися зі службою (наприклад, виконати RPC). У реалізації цього методу розробник повинен надати інтерфейс, який клієнти використовують для зв’язку зі службою, повертаючи IBinder.

3. onCreate() — система викликає цей метод для виконання одноразових процедур налаштування під час початкового створення служби. Якщо служба вже запущена, цей метод не викликається.

4. onDestroy() — система викликає цей метод, коли служба більше не використовується та знищується. Служба розробника повинна реалізувати це, щоб очистити будь-які ресурси, такі як потоки, зареєстровані слухачі або приймачі.

Якщо компонент запускає службу, викликаючи startService(), то служба продовжує працювати, доки не зупиниться сама за допомогою stopSelf() або інший компонент не зупинить її, викликавши stopService().

Якщо компонент викликає bindService() для створення служби, а onStartCommand() не викликається, служба працює лише до тих пір, поки компонент прив’язаний до неї.

Після того, як служба від’єднується від усіх своїх клієнтів, система знищує її [23].

Система Android зупиняє службу лише тоді, коли пам’яті мало, і вона повинна відновити системні ресурси для діяльності, на яку спрямований користувач.

Якщо послуга пов’язана з діяльністю, орієнтованою на користувача, менша ймовірність її знищення, якщо оголошено, що служба працює на передньому плані, вона рідко вимикається.

Якщо службу запущено та працює довго, система з часом знижує свою позицію в списку фонових завдань, і служба стає дуже сприйнятливою до знищення. Якщо службу запущено, необхідно спроектувати її так, щоб вона ефективно обробляла перезапуски за допомогою системи. Якщо система вимикає службу, вона перезапускає її, щойно ресурси стають доступними, але це також залежить від значення, яке повертається із onStartCommand().

Запущена служба — це служба, яку запускає інший компонент, викликаючи startService(), що призводить до виклику методу onStartCommand() служби.

Коли службу запускають, вона має життєвий цикл, який не залежить від компонента, який її запустив. Служба може працювати у фоновому режимі необмежений час, навіть якщо компонент, який її запустив, буде знищено. Таким чином, служба повинна зупинити себе, коли її робота завершена, викликавши stopSelf(), або інший компонент може зупинити її, викликавши stopService() [24].

Компонент програми, наприклад активність, може запустити службу, викликавши startService() і передавши Intent, який визначає службу та містить будь-які дані для використання службою. Служба отримує цей намір у методі onStartCommand().

Припустімо, що під час дії необхідно зберегти деякі дані в онлайн-базі даних. Активність може запустити супутню службу та надати їй дані для збереження, передавши намір у startService(). Служба отримує намір у onStartCommand(), підключається до Інтернету та виконує транзакцію бази даних. Після завершення транзакції служба припиняє роботу та знищується.

Клас Service є базовим класом для всіх послуг. Коли розробник розширює цей клас, важливо створити новий потік, у якому служба зможе завершити всю свою роботу. Служба за замовчуванням використовує основний потік програми, що може уповільнити продуктивність будь-якої активності, яку виконує програма.

Платформа Android також надає підклас служби IntentService, який використовує робочий потік для обробки всіх запитів на запуск по одному.

Служба запускається з діяльності або іншого компонента програми, передаючи Intent до startService() або startForegroundService(). Система Android викликає метод служби onStartCommand() і передає йому Intent, який визначає, яку службу запускати [25].

Наприклад, дія може запустити приклад служби в попередньому розділі, використовуючи явний намір за допомогою startService(). Це наведено нижче.

```
Intent(this, HelloService::class.java).also { intent ->
    startService(intent)
}
```

Метод startService() повертає негайно, і система Android викликає метод служби onStartCommand(). Якщо служба ще не запущена, система спочатку викликає onCreate(), а потім викликає onStartCommand().

Якщо служба також не забезпечує зв’язування, намір, який доставляється за допомогою startService(), є єдиним способом зв’язку між компонентом програми та службою.

Кілька запитів на запуск служби призводять до кількох відповідних викликів onStartCommand() служби. Однак для зупинки служби потрібен лише один запит.

Запущена служба повинна керувати власним життєвим циклом. Тобто система не зупиняє та не знищує службу, якщо їй не потрібно відновити системну пам’ять, і служба продовжує працювати після повернення onStartCommand().

Служба повинна зупинити себе, викликавши stopSelf(), або інший компонент може зупинити її, викликавши stopService().

Після запиту на зупинку за допомогою stopSelf() або stopService() система знищує службу якомога швидше [26].

Якщо служба обробляє кілька запитів до onStartCommand() одночасно, розробнику не слід зупиняти службу після завершення обробки запиту на запуск, оскільки він міг отримати новий запит на запуск.

Щоб уникнути цієї проблеми, розробник може використовувати stopSelf(int), щоб запит на зупинку служби завжди базувався на останньому запиті на запуск. Тобто, коли викликається stopSelf(int), він передає ідентифікатор початкового запиту (startId, наданий onStartCommand()), якому відповідає зупинковий запит.

Потім, якщо служба отримує новий запит на запуск до того, як можна буде викликати stopSelf(int), ідентифікатор не збігається, і служба не зупиняється [27].

1.4 Можливості Android-інструментів Firebase від Google

Для додатків Android використовуються сервери Oracle SQL, Microsoft SQL Server і MySQL, які підключені до сервера за допомогою файлів PHP. Потім з’явився Firebase для додатків Android, які використовують JSON для зберігання даних. Інші сервери використовують формат таблиці (рядки та стовпці) для зберігання даних. Firebase базується на NoSQL. Існує дуже мало доступних хмарних серверів, подібних до Firebase, як-от:

1. AWS Mobile Hub – це інтегрована консоль, яка допомагає створювати, будувати, тестувати та контролювати мобільні програми, які використовують служби AWS.

2. CloudKit – це платформа Apple, яка допомагає зберігати дані та активи, але схожа лише на iOS.

Firebase вважається платформою веб-додатків. Це допомагає розробникам створювати високоякісні програми. Він зберігає дані у форматі JavaScript Object Notation (JSON), який не використовує запити для вставки,

оновлення, видалення або додавання даних. Це серверна частина системи, яка використовується як база даних для зберігання даних.

Firebase містить кілька сервісів, які можна додати до програми, а саме:

1. Firebase Analytics – надає інформацію про використання програми. Це рішення для вимірювання платної програми, яке також забезпечує залучення користувачів. Ця функція дозволяє розробнику програми зрозуміти, як користувачі використовують програму. SDK має функцію самостійного захоплення подій і властивостей, а також дозволяє отримувати спеціальні дані.

2. Firebase Cloud Messaging (FCM) – раніше відомий як Google Clouds Messaging (GCM), FCM – це платна служба, яка є міжплатформним рішенням для повідомлень і сповіщень для Android, веб-програм та IOS.

3. Firebase Auth – підтримує соціальні провайдери, такі як Facebook, Google GitHub і Twitter. Це послуга, яка може автентифікувати користувачів лише за допомогою коду на стороні клієнта, і це платна послуга. Він також містить систему керування користувачами, за допомогою якої розробники можуть увімкнути автентифікацію користувачів за допомогою електронної пошти та пароля для входу, які зберігаються у Firebase.

4. Realtime Database – надає такі послуги, як база даних у реальному часі та серверна частина. Розробнику програми надається API, який дозволяє синхронізувати дані програми між клієнтами та зберігати їх у хмарі Firebase. Клієнтські бібліотеки надаються компанією, що забезпечує інтеграцію з Android, IOS і JavaScript.

5. Firebase Storage – полегшує передачу файлів незалежно від якості мережі для програм Firebase. Він підтримується Google Cloud Storage, яка є рентабельною службою зберігання об’єктів. Розробник може використовувати його для зберігання зображень, аудіо, відео чи іншого вмісту, створеного користувачами.

6. Firebase Test Lab for Android – забезпечує хмарну інфраструктуру для тестування програм Android. За допомогою однієї операції розробники можуть ініціювати тестування своїх програм на різноманітних пристроях і

конфігураціях пристроїв. Різні результати тестування, такі як знімки екрана, відео та журнали, доступні на консолі Firebase. Навіть якщо розробник не написав жодного тестового коду для своєї програми, Test Lab може перевірити програму автоматично, шукаючи збої.

7. Firebase Crash Reporting – докладні звіти про помилки, що створюються в додатку. Помилки групуються в кластери схожих трасувань стека та сортуються за серйозністю. Розробник може реєструвати власні події, щоб допомогти зафіксувати кроки, що призвели до збою [28].

1.4.1 Firebase Auth як інструмент автентифікації користувача

Firebase Auth надає серверні служби, прості у використанні SDK і готові бібліотеки інтерфейсу користувача для автентифікації користувачів у програмі.

Він підтримує автентифікацію за допомогою паролів, номерів телефонів, популярних федеративних постачальників ідентифікаційної інформації, таких як Google, Facebook і Twitter, тощо.

Firebase Auth тісно інтегрується з іншими службами Firebase і використовує такі галузеві стандарти, як OAuth 2.0 і OpenID Connect, тож її можна легко інтегрувати з настроюваною серверною частиною.

Додавання автентифікації Firebase до програми відбувається за допомогою Gradle-модуля. Необхідно додати залежність для бібліотеки Firebase Auth Android. Рекомендується використовувати Firebase Android BoM для керування версіями бібліотеки.

Код залежності наведено нижче.

```
dependencies {
    implementation platform('com.google.firebase:firebase-bom:31.1.0')
    implementation 'com.google.firebase:firebase-auth'
}
```

Щоб використовувати постачальника автентифікації, розробнику потрібно ввімкнути його у консолі Firebase. Це наведено на рисунку 1.1.

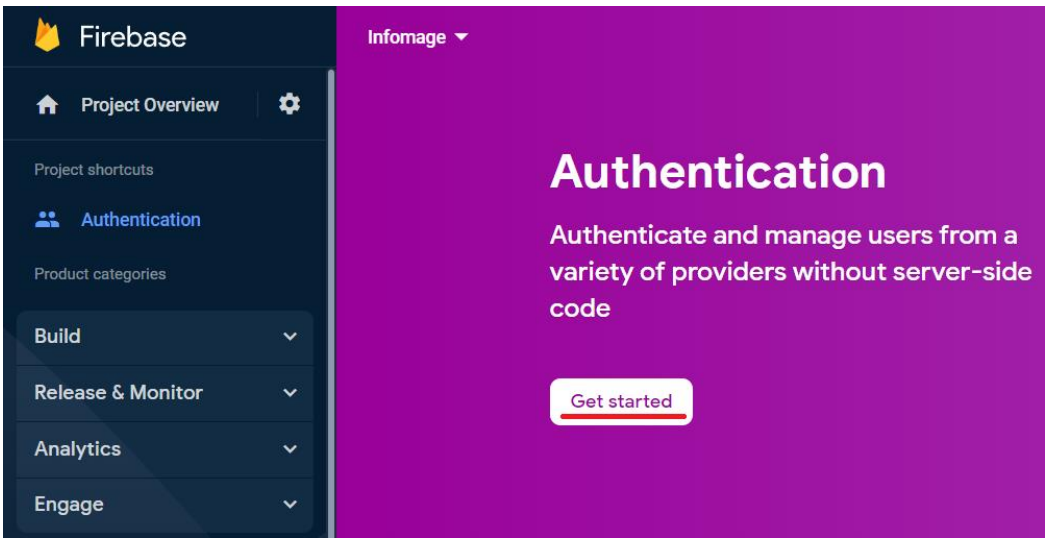

Рисунок 1.1 Додавання Firebase Authentication у консолі розробника

Якщо розробник обирає методи автентифікації та постачальників, випробовує різні моделі даних із загальнодоступними та приватними даними за допомогою автентифікації та правил безпеки Firebase або створює прототип дизайну інтерфейсу користувача для входу, можливість працювати локально без розгортання живих служб є перевагою.

Емулятор автентифікації є частиною набору локальних емуляторів, який дозволяє додатку взаємодіяти з емульованим вмістом і конфігурацією бази даних, а також з емульованими ресурсами проекту за бажанням.

Використання емулятора автентифікації включає лише кілька кроків:

1. Додавання рядка коду до тестової конфігурації програми для підключення до емулятора.

2. З кореня локального каталогу проекту, запуск емулятору за допомогою firebase: start.

3. Використання інтерфейсу користувача Local Emulator Suite для інтерактивного прототипування або REST API емулятора автентифікації для неінтерактивного тестування.

Щоб почати роботу з автентифікацією Firebase необхідно перш за все перевірити поточний стан автентифікації.

Для цього необхідно:

1. Оголосити екземпляр FirebaseAuth. Код для цього наведено нижче.

```
private FirebaseAuth mAuth;
```

2. У методі onCreate() ініціалізувати екземпляр FirebaseAuth. Код для цього наведено нижче.

```
mAuth = FirebaseAuth.getInstance();
```

3. Перевірити, чи користувач наразі ввійшов у систему під час ініціалізації діяльності. Код для цього наведено нижче.

```
@Override
public void onStart() {
    super.onStart();
    FirebaseUser currentUser = mAuth.getCurrentUser();
    if(currentUser != null){
        reload();
    }
}
```

Після цього необхідно реалізувати надсилання коду підтвердження на телефон користувача.

Щоб ініціювати вхід за номером телефону, розробник повинен надати користувачеві інтерфейс, який пропонує ввести номер телефону. Юридичні вимоги відрізняються, але, як найкраща практика, та щоб встановити очікування для користувачів, розробник повинен повідомити їх, що якщо вони використовують вхід із телефону, вони можуть отримати SMS-повідомлення для перевірки та після цього застосуються стандартні тарифи.

Потім необхідно передати його номер телефону в метод PhoneAuthProvider.verifyPhoneNumber, щоб запросити від Firebase підтвердження номеру телефону користувача.

Код для цього наведено нижче.

```
    PhoneAuthOptions options =
PhoneAuthOptions.newBuilder(mAuth)
    .setPhoneNumber(phoneNumber)
    .setTimeout(60L, TimeUnit.SECONDS)
    .setActivity(this)
    .setCallbacks(mCallbacks)
     OnVerificationStateChangedCallbacks
    .build();
PhoneAuthProvider.verifyPhoneNumber(options);
```

Метод verifyPhoneNumber є реентерабельним: якщо розробник викликає його кілька разів, наприклад, у методі onStart для дії, метод verifyPhoneNumber не надсилатиме друге SMS, якщо вихідний запит не минув.

Розробник може використати цю поведінку, щоб відновити процес входу за номером телефону, якщо програма закриється, перш ніж користувач зможе ввійти.

Після виклику verifyPhoneNumber необхідно встановити позначку, яка вказує на те, що перевірка триває. Потім потрібно зберегти прапорець у методі onSaveInstanceState своєї активності та відновити прапорець у onRestoreInstanceState [29].

1.4.2 Firebase Realtime Database як інструмент реалізації бази даних

Firebase Realtime Database — це база даних, розміщена в хмарі. Дані зберігаються як JSON і синхронізуються в реальному часі з кожним підключеним клієнтом.

Коли міжплатформні додатки створюються на платформах Apple, Android і JavaScript, усі клієнти спільно використовують один екземпляр Realtime Database і автоматично отримують оновлення з найновішими даними.

Щоб створити базу даних необхідно:

- перейти до розділу Realtime Database у консолі Firebase;
- обрати режим запуску для правил безпеки Firebase;
- обрати місце для бази даних.

Далі необхідно додати Firebase Realtime Database SDK у проект через залежності Gradle. Код залежностей наведено нижче.

```
dependencies {
    implementation platform('com.google.firebase:firebase-bom:31.1.0')
    implementation 'com.google.firebase:firebase-database'
}
```

База даних у реальному часі надає мову декларативних правил, яка дозволяє визначати, як дані повинні бути структуровані, як вони повинні бути індексовані, а також коли дані можна читати та записувати.

Щоб записати дані в базу даних, розробник повинен отримати екземпляр бази даних за допомогою getInstance() і вказати місце, куди його записати. Код для цього наведено нижче.

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");
```

Таким чином розробник може зберігати в базі даних ряд типів даних, включаючи об’єкти Java. Коли об’єкт зберігає відповіді від будь-яких геттерів, вони будуть збережені як дочірні для цього розташування.

Щоб оновлювати дані програми в реальному часі, необхідно додати ValueEventListener до щойно створеного посилання.

Метод onDataChange() у цьому класі запускається один раз, коли приєднується слухач, і знову кожного разу, коли змінюються дані, включно з дочірніми. Код для цього наведено нижче.

```
myRef.addValueEventListener(new ValueEventListener() {
@Override
public void onDataChange(DataSnapshot dataSnapshot) {
String value = dataSnapshot.getValue(String.class);
Log.d(TAG, "Значення: " + value);
}
@Override
public void onCancelled(DatabaseError error) {

Log.w(TAG, "Помилка зчитування", error.toException());
}
});
```

Увімкнувши поведінку збереження, будь-які дані, які клієнт Firebase Realtime Database синхронізував би в режимі онлайн, зберігаються на диску та доступні в автономному режимі, навіть коли користувач або операційна система перезапускає програму.

Це означає, що програма працює так само, як і в Інтернеті, використовуючи локальні дані, що зберігаються в кеші.

Зворотні виклики слухача продовжуватимуть запускати локальні оновлення.

Клієнт Firebase Realtime Database автоматично зберігає чергу всіх операцій запису, які виконуються, коли програма офлайн.

Якщо постійність увімкнено, ця черга також зберігається на диску, тому всі записи доступні, коли користувач або операційна система перезапускає програму.

Коли програма відновлює з'єднання, усі операції надсилаються на сервер бази даних у реальному часі Firebase.

Якщо програма використовує автентифікацію Firebase, клієнт Firebase Realtime Database зберігає маркер автентифікації користувача під час перезапуску програми.

Якщо термін дії маркера автентифікації закінчується, коли програма перебуває в автономному режимі, клієнт призупиняє операції запису, доки програма повторно не автентифікує користувача, інакше операції запису можуть бути невдалими через правила безпеки.

Firebase Realtime Database зберігає дані, отримані від запиту, для використання в режимі офлайн.

Для запитів, створених у автономному режимі, база даних Firebase продовжує працювати з раніше завантаженими даними. Якщо запитувані дані не завантажуються, Firebase Realtime Database завантажує дані з локального кешу.

Коли з'єднання з мережею знову стане доступним, дані завантажуються та відображатимуть запит.

Клієнти Firebase Database надають прості примітиви, які можна використовувати для запису в базу даних, коли клієнт від'єднується від серверів Firebase Database.

Ці оновлення відбуваються незалежно від того, чи від'єднується клієнт чи ні, тож розробник може покладатися на них для очищення даних, навіть якщо з'єднання розривається або клієнт виходить з ладу.

Усі операції запису, включаючи налаштування, оновлення та видалення, можна виконувати після відключення [30].

2 ТЕХНОЛОГІЇ РОЗРОБКИ МОБІЛЬНИХ ЗАСТОСУНКІВ

Якщо необхідно розробити Android або iOS програму без проблем із серверним програмуванням, то дуже важливо знайти кращу платформу для розробки програм.

У кваліфікаційній роботі вже було розглянуто платформу для створення серверної частини мобільного додатка Firebase від Google. У цьому розділі також буде проаналізовано схожі технології для серверного програмування Android-додатків, а саме:

- Back4App;
- Parse;
- AWS Amplify;
- Firehose;
- Deployd;
- Atmosphere;
- Hasura.

Технологію Firebase було обрано за велику кількість додаткових сервісів, інтеграцію з Google Analytics а також за легкіть використання документації для розробника, але перед тим, як обрати саме Firebase, було проаналізовано інші технології.

2.1 Аналіз технології розробки мобільних застосунків Back4App

Back4App — це платформа з відкритим вихідним кодом (на основі платформи Parse Platform), яка допомагає розробляти сучасні програми. У Back4App є наступний функціонал:

- Зберігання даних (реляційне);
- Функції Cloud Code;
- API (GraphQL і REST);
- Зберігання файлів;

- Аутентифікація;
- Push-сповіщення.

Для Android доступний Parse SDK, який дозволяє інтегрувати в програму різні інструменти і також спрощує розробку. Як і у випадку з Firebase, Parse SDK встановлюється за допомогою Gradle.

У Back4App розробка заснована на шаблонах. Є два шаблони для Android, один написаний на Java, а інший на Kotlin. Шаблони дозволяють керувати моделлю даних та виконувати такі операції: додавання, видалення, оновлення та читання даних.

Значення, які можуть зберігатися в базі даних Back4App, можуть бути типу String, Number, Bool, Array, Object, Date, File, Pointer, Relation і null.

Зберігання даних у Back4App побудовано навколо ParseObject. Кожен ParseObject містить пари ключ-значення JSON-сумісних даних. Ці дані є безсхемними, що означає, що не потрібно завчасно вказувати, які ключі існують для кожного ParseObject. Можна встановлювати будь-які пари ключ-значення, які потрібні, і бекенд зберігатиме їх.

Використовуючи Back4App, можна зберігати об’єкти даних, встановлюючи зв’язки між ними.

Для моделювання такої поведінки будь-який ParseObject можна використовувати як значення в іншому ParseObject. Внутрішньо структура Back4App зберігатиме об’єкт, на який посилається, лише в одному місці, щоб підтримувати узгодженість.

Це може дати додаткові можливості під час створення та виконання складних запитів. Існує три основних типи відносин:

- one-to-one, що встановлює безпосередні зв'язки між двома об'єктами і тільки ними;
- one-to-many, де один об'єкт може бути пов'язаний з багатьма іншими об'єктами;
- many-to-many, який може створювати багато складних зв'язків між багатьма об'єктами.

Існує два способи створити відношення one-to-many у Back4App:

По-перше, це використання вказівників у дочірньому класі, який є найшвидшим у часі створення та запиту.

По-друге, використання масивів вказівників у батьківському класі може призвести до сповільнення часу виконання запитів залежно від їх розміру.

Також у Back4App є запити, які можуть реалізовувати зв’язки між об’єктами та допомагати в управлінні даними. Роботу з запитами реалізовано через клас ParseQuery.

У Back4App існує поняття “живих запитів”. Живі запити призначені для використання в програмах у реальному часі, де просто використання традиційної парадигми запитів призведе до деяких проблем, такі як збільшення часу відповіді та високе використання мережі та сервера.

Живі запити слід використовувати у випадках, коли потрібно постійно оновлювати сторінку свіжими даними, що надходять із бази даних, що часто трапляється, але не обмежується ними: в онлайн-іграх, клієнтах обміну повідомленнями та спільних списках справ.

Back4App дозволяє реалізувати автентифікацію та авторизацію користувача: за допомогою Facebook, Twitter та електронної пошти.

Крім того, в Back4App існує окремий сервіс для роботи з повідомленнями в Android. Інструмент є достатньо гнучким для роботи з повідомленнями як за допомогою Dashboard, так і Cloud Code.

2.2 Аналіз технології розробки мобільних застосунків AWS Amplify

AWS Amplify допомагає розробляти та розгортати хмарні мобільні та веб-програми. Amplify зберігає багато сервісів, які можуть допомогти з різними завданнями в процесі розробки. Наприклад:

Amplify Authentication – увімкнення входу, реєстрації та виходу за допомогою попередньо створених компонентів інтерфейсу користувача та API автентифікації;

Amplify Storage – механізм керування вмістом користувачів у публічному, захищеному чи приватному сховищі;

GraphQL API – рішення для доступу до внутрішніх даних із підтримкою оновлень у реальному часі;

Amplify DataStore – синхронізація та збереження даних онлайн і офлайн у хмарі, а також на різних пристроях;

Amplify Geo – впровадження інтерактивних карт з маркерами розташування та пошуком місця розташування;

REST API – просте та безпечне рішення для надсилання запитів HTTP;

Amplify Analytics – прийняття обґрунтованих рішень за допомогою вбудованої аналітики для відстеження сеансів користувачів, налаштування атрибутів користувача та показників у програмі;

Amplify Push Notifications – стимулювання залучення клієнтів за допомогою сповіщень із аналітикою та націлюванням;

Amplify PubSub – з'єднання програми з проміжним програмним забезпеченням, орієнтованим на повідомлення, у хмарі.

Розглянемо Amplify Authentication та Storage.

Сервіс Amplify Auth надає інтерфейс для автентифікації користувача. За лаштунками він надає необхідний дозвіл для інших категорій Amplify. Він поставляється зі стандартною вбудованою підтримкою пулу користувачів Amazon Cognito та пулу ідентифікаційних даних. Інтерфейс командного рядка Amplify допомагає створити та налаштувати категорію автентифікації за допомогою автентифікації. Існує шість варіантів для автентифікації:

- ім'я користувача та пароль;
- використання веб-інтерфейсу;
- Facebook;
- Google;
- Amazon;
- Apple.

Сервіс Amplify Storage надає інтерфейс для керування користувацьким вмістом програми в загальнодоступних, захищених або приватних сховищах. Сервіс Storage має стандартну підтримку Amazon Simple Storage Service (S3). Amplify CLI допоможе створити та налаштувати сегменти зберігання для застосунку.

Щоб додати сервіси до процесу розробки необхідно додати залежності у Gradle.

Amplify Auth та Storage реалізуються за допомогою класу Amplify у Java або Kotlin.

2.3 Аналіз технології розробки мобільних застосунків PocketBase

PocketBase – це бекенд із відкритим вихідним кодом, що складається з вбудованої бази даних (SQLite) із підписками в реальному часі, вбудованим керуванням автентифікацією, зручним інтерфейсом інформаційної панелі та простим REST API.

Дані програми представлені колекціями. Колекція - це звичайна таблиця SQLite, яка автоматично генерується з назвою колекції та полями (такими як стовпцями). Окремий запис колекції називається записом або одним рядком у таблиці SQL.

Розробник може створювати колекції та записи з інтерфейсу адміністратора або Web API.

Base Collection є типом колекції за замовчуванням, і її можна використовувати для зберігання будь-яких даних програми (наприклад, статей, продуктів, публікацій тощо). Вона поставляється з 3 системними полями за замовчуванням, які завжди доступні та автоматично заповнюються: id, created, updated. Явно можна встановити лише id (рядок із 15 символів).

Auth Collection містить усе, що стосується базової колекції, але з деякими додатковими спеціальними полями, які допоможуть керувати користувачами програми, надаючи різні параметри автентифікації.

Кожна колекція Auth містить такі системні поля: id, created, updated, username, email, emailVisibility, verified.

Розробник може мати скільки завгодно колекцій автентифікацій (наприклад, користувачів, менеджерів, співробітників, учасників, клієнтів тощо), кожна з яких має власний набір полів, окремий логін (електронна адреса/ім'я користувача + пароль або OAuth2) і моделі керування кінцевими точками.

Розробник може створювати різноманітні елементи керування доступом:

Role – розробник може приєднати поле вибору "роль" до колекції Auth із такими параметрами: regularUser і superUser. І тоді в деяких інших колекціях розробник міг би визначити правило, яке дозволяє лише супер-користувачам керувати даними.

Relation – скажімо, у розробника є 2 колекції - базова колекція повідомлень і колекція авторизації користувачів. У колекції повідомлень розробник може створити поле зв'язку "автор", яке вказує на колекцію користувачів. Щоб дозволити доступ лише автору запису, розробник може використати правило, яке забороняє доступ для всіх, крім автора.

PocketBase дозволяє використовувати наступні способи автентифікації:

як адмін;

як користувач програми;

Web OAuth2.

Користувач може пройти автентифікацію як адміністратор за допомогою електронної пошти та пароля. Адміністратори мають доступ до всього, і правила API до них не застосовуються.

Найпростіший спосіб автентифікувати користувачів застосунку - це за допомогою імені користувача/електронної пошти та пароля.

Розробник також може автентифікувати користувачів за допомогою постачальника OAuth2 (Google, GitHub, Microsoft тощо).

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ANDROID-ЗАСТОСУНКУ

Для розробки користувальницького інтерфейсу(UI) було проаналізовано патерни UI. Також було проаналізовано реалізацію інформаційної архітектури та для подальшої реалізації користувацького досвіду. Окрім цього, було розроблено UML-діаграму, що представляє відносини між передньою частиною та серверною частиною застосунку. UML-діаграму показано на рисунку 3.1.

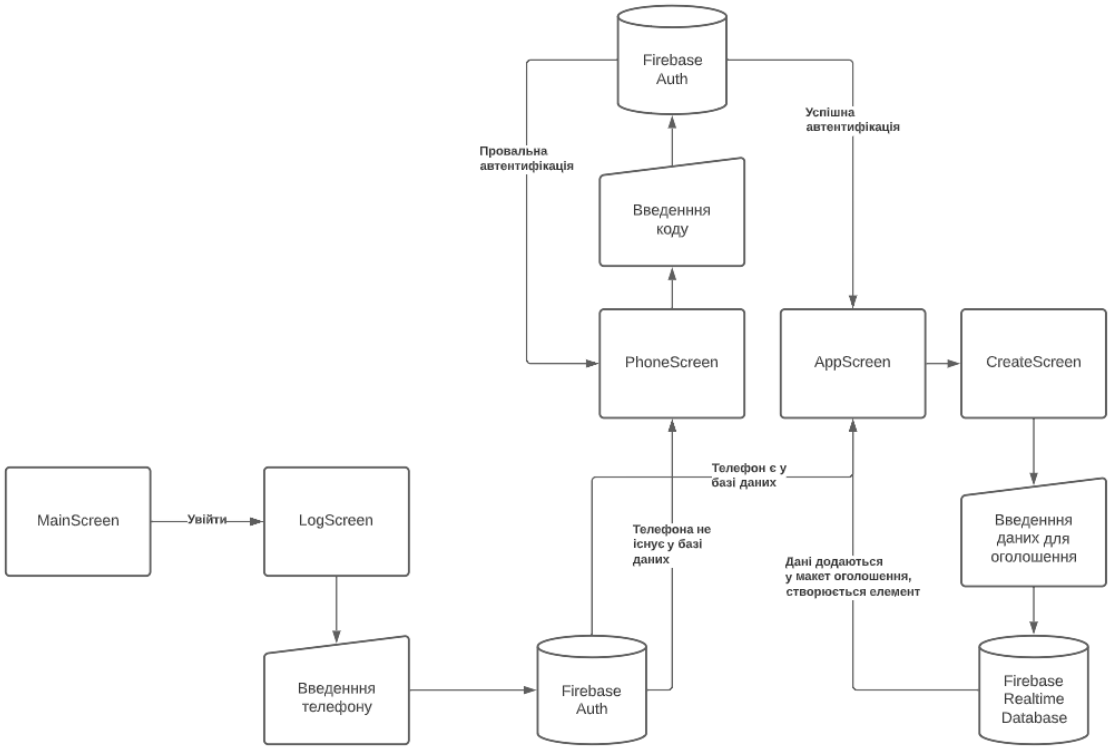

Рисунок 3.1 – UML-діаграма застосунку Infomage

Користувач починає роботу з сутності MainScreen, що є головним екраном. Далі після натискання на елемент управління “Увійти” він потрапляє на сутність LogScreen, що є екраном вводу телефону користувача.

Після введення телефону, він перевіряється за допомогою Firebase Auth, якщо телефону не існує у базі даних, то користувач потрапляє на PhoneScreen

і повинен автентифікувати телефон, якщо телефон вже є у базі, то користувач одразу переходить до екрану з оголошеннями AppScreen.

Процес автентифікації на екрані PhoneScreen відбувається наступним чином:

- користувач отримує код за допомогою СМС;
- якщо користувач правильно вводить код, то він переходить до екрану з оголошеннями AppScreen;
- якщо код було введено неправильно, то користувач залишається у PhoneScreen.

Після автентифікації користувач може перейти до екрану додавання оголошення CreateScreen.

Процесс додавання оголошень відбувається наступним чином:

- користувач вводить дані – ім’я, прізвище, телефон, предмет, роль;
- після натискання на “Створити” дані додаються до Firebase Realtime Database, де передаються у застосунок до нового макету оголошення і створюють новий елемент на екрані AppScreen;
- користувач повертається до екрану AppScreen.

3.1 Створення макету екранів та елементів керування Infomage

Для мобільного додатку Infomage було створено 6 макетів:

- головний екран;
- екран аутентифікації користувача за мобільним телефоном;
- екран введення коду для автентифікації;
- екран програми без оголошень;
- екран програми з оголошеннями;
- екран створення оголошення.

Головний екран додатку зображено на рисунку 3.1.

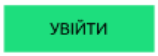

Рисунок 3.2 – Головний екран додатку Infomage

Головний екран містить кнопку входу в програму, яка призведе користувача до екрана автентифікації мобільного телефону. Екран автентифікації зображено на рисунку 3.3.

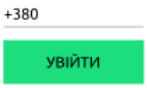

Рисунок 3.3 – Екран автентифікації користувача

Після введення мобільного телефону користувач переходить на екран вводу коду, який повинен прийти у СМС. Екран вводу коду зображено на рисунку 3.4.

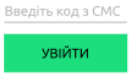

Рисунок 3.4 – Екран вводу коду, що прийде користувачу у СМС

Після автентифікації користувач переходить у головний екран без оголошень. Цей екран зображено на рисунку 3.5.

Рисунок 3.6 – Головний екран без доданих оголошень

Якщо користувач натисне на кнопку “ОНОВИТИ”, то головний екран буде оновлено. Якщо користувач натисне на “+”, то він перейде на екран створення нового оголошення. Якщо користувач натисне на “ВИЙТИ”, то йому буде запропоновано натиснути ще раз, якщо він хоче вийти зі свого акаунту. Екран створення оголошення зображено на рисунку 3.7.

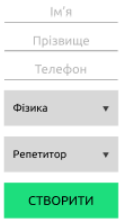

Рисунок 3.7 – Екран створення оголошення

Користувач повинен ввести своє ім’я, прізвище та телефон(телефон у подальшому підставляється той, з яким користувач автентифікувався у додаток).

Також користувачеві надається змога обрати предмет із списку предметів та свою роль – або репетитор(той, хто надає послуги) або студент(той, хто потребує послуг).

Після усіх введень користувач має натиснути на кнопку “СТВОРИТИ”, щоб на головному екрані створилося нове оголошення.

Після натискання на кнопку “СТВОРИТИ”, екран створення оголошення закривається та користувач переходить на головний екран вже з новим оголошенням.

Головний екран з оголошенням зображено на рисунку 3.8.

Рисунок 3.8 – Головний екран з новим оголошенням

Також окремо було створено макет самого оголошення, так як у майбутньому цей макет буде використано у Android Studio у якості нової розмітки у файлі ad_layout. Макет оголошення зображено на рисунку 2.7.

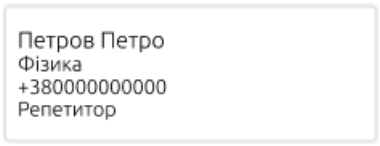

Рисунок 3.9 – Макет оголошення, що буде використано окремим файлом при розробці

Отже, за допомогою програмного забезпечення Figma було створено макети екранів, які у подальшому буде використано у Android-застосунку Infomage.

3.2. Реалізація користувацького інтерфейсу Android-застосунку

Перш за все, у Android Studio було створено новий проект. Проект має підтримку Android 5-ї версії та вище, API версії 21. Було додано новий ресурс – strings.xml. Він потрібен для того, щоб надавати текст для елементів користувацького інтерфейсу. Також було змінено файл colors.xml, який відповідає за вибір кольорів Android-застосунку.

3.2.1 Створення головного екрану додатку Infomage

У Android Studio було створенно новий Java-клас MainActivity та файл макету для екрану activity_main.xml.

Для екрану було створено елемент Button з текстом “УВІЙТИ”, для цього елементу було додано Constraint зліва, справа, зверху та знизу.

Constraint – це прив’язка елемента до іншого елемента, у цьому випадку – це до батьківського елементу – екрану. Екран, що було створено показано на рисунку 3.10.

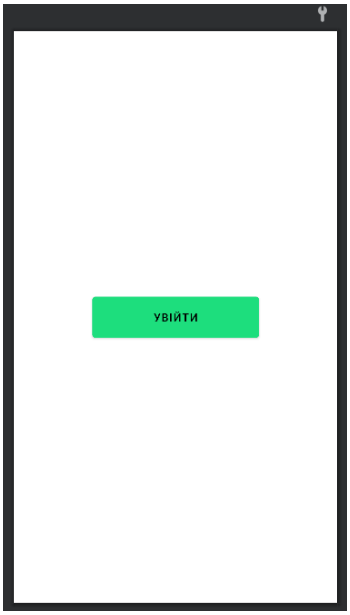

Рисунок 3.10 – Реалізація головного екрану у Android Studio

Для класу MainActivity було перевизначено метод onBackPressed(), що оброблює натискання кнопки “Назад” у Android. Після натискання цієї кнопки програма завершується. Код методу показано нижче.

```
@Override
    public void onBackPressed() {
        finish();
}
```

Метод finish(), що знаходиться у тілі метода onBackPressed() завершує роботу MainActivity.

3.2.2 Створення екрану автентифікації мобільного телефону

У Android Studio було створенно новий Java-клас LogActivity та файл макету для екрану activity_log.xml. Для екрану було створено елемент EditText для вводу телефону та елемент Button для початку автентифікації. Для цих елементів було додано Constraint: для EditText – зліва, справа та зверху до екрану, знизу до Button, для Button – зліва, справа та знизу до екрану, зверху до EditText. Також елементам було додано відступи зверху та знизу, щоб вони розташувалися по центру екрану. Екран автентифікації показано на рисунку 3.11.

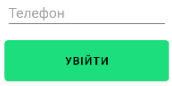

Рисунок 3.11 – Екран автентифікації мобільного телефону

Для класу LogActivity було також перевизначено метод onBackPressed(), що оброблює натискання кнопки “Назад” у Android. Після натискання цієї кнопки користувач переходить на головний екран. Перехід на головний екран реалізовано за допомогою класу Intent.

Цей клас визначає “наміри” класу перед іншим класом.

За допомогою цього класу у подальшому буде передано мобільний телефон на інший екран. Код методу показано нижче.

```
@Override
    public void onBackPressed() {
        Intent i = new Intent(LogActivity.this, MainActivity.class);
        startActivity(i);
        finish();
}
```

Також було пов’язано головний екран з екраном автентифікації за допомогою методу створеного onClickLogIn(). Цей метод прив’язується к кнопці “УВІЙТИ” через XML. Код методу наведено нижче.

```
public void onClickLogIn(View view){
        Intent i = new Intent(MainActivity.this, LogActivity.class);
        startActivity(i);
        finish();
}
```

3.2.3 Створення екрану для вводу коду з СМС

У Android Studio було створенно новий Java-клас PhoneActivity та файл макету для екрану activity_phone.xml.

Для екрану було створено елемент EditText для вводу коду та елемент Button для підтвердження коду.

Для цих елементів було додано ConstraintТакож елементам було додано відступи зверху та знизу, щоб вони розташувалися по центру екрану.

Екран автентифікації показано на рисунку 3.12.

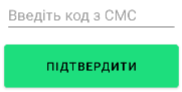

Рисунок 3.12 – Екран вводу коду з СМС

Для класу LogActivity було також перевизначено метод onBackPressed(), що оброблює натискання кнопки “Назад” у Android. Після натискання цієї кнопки користувач переходить на головний екран мобільного додатку. Код методу наведено нижче.

```
@Override
    public void onBackPressed() {
        Intent i = new Intent(PhoneActivity.this, MainActivity.class);
        startActivity(i);
        finish();
}
```

Також було додано метод onClickLog, який закриває LogActivity та відкриває PhoneActivity.

Це зроблено доки немає реалізації автентифікації мобільного телефону користувача.

Метод прив’язується до кнопки “УВІЙТИ” через XML. Код методу наведено нижче.

```
public void onClickLog(View view){
        Intent i = new Intent(LogActivity.this, PhoneActivity.class);
        startActivity(i);
        finish();
}
```

3.2.4 Створення екрану з оголошеннями

У Android Studio було створенно новий Java-клас AppActivity та файл макету для екрану activity_app.xml. Для екрану було створено елемент RecyclerView – представлення для оголошень та LinearLayout – макет, що містить у собі елементи у горизонтальному напрямку. У LinearLayout було додано кнопки “ОНОВИТИ”, “+” та “ВИЙТИ”.

Елемент RecyclerView розташовується з початку екрану та до LinearLayout. LinearLayout розташовується знизу екрану. Для RecyclerView та LinearLayout було додано Constraint:

- для RecyclerView – зліва, зверху та справа до батьківського елементу, знизу – до LinearLayout;
- для LinearLayout – зліва, знизу та справа до батьківського елементу, зверху – до RecyclerView.

Для RecyclerView та LinearLayout було додано відступ зліва та справа у розмірі 8dp. dp – це одиниця вимірювання екрану у Android Studio. Екран з оголошеннями показано на рисунку 3.13.

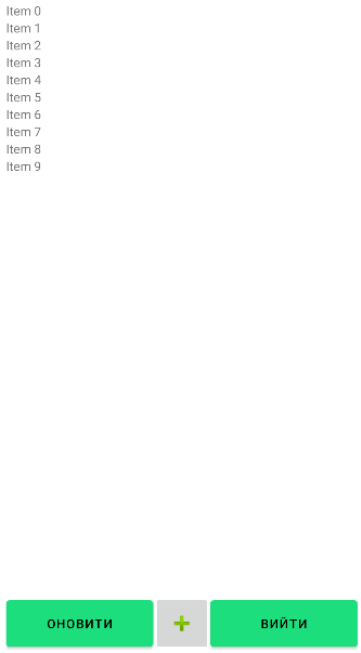

Рисунок 3.13 – Екран з оголошеннями

У коді для кнопки “ОНОВИТИ” було додано Intent для відкриття AppActivity. Код наведено нижче.

```
public void onClickReload(View view) {
        Intent i = new Intent(AppActivity.this, AppActivity.class);
        startActivity(i);
        finish();
}
```

У коді для кнопки “+” було додано Intent для відкриття екрану зі створенням оголошень. Код наведено нижче.

```
public void onClickCreate(View view){
        Intent i = new Intent(AppActivity.this, CreateActivity.class);
        startActivity(i);
        finish();
}
```

У коді для кнопки “ВИЙТИ” було додано Intent для виходу користувача з акаунту та відкриття MainActivity. Код наведено нижче.

```
public void onClick(View view) {
      startActivity(new Intent(AppActivity.this, MainActivity.class));
      finish();
}
```

3.2.5 Створення екрану для додавання оголошень

У Android Studio було створенно новий Java-клас CreateActivity та файл макету для екрану activity_create.xml. Для екрану було створено EditText-елементи для імені, прізвища та телефону користувача.

Було додано Spinner-елементи, які є спадними списками для предмету, що потрібен користувачу та ролі(репетитор або студент).

Також було додано Button-елемент “СТВОРИТИ” для додавання оголошення на екран AppActivity.

Кожен елемент має Constraint-прив’язку та відступ, щоб вони розташовувалися по центру екрану.

Екран наведено на рисунку 3.14.

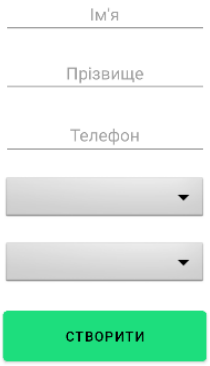

Рисунок 3.14 – Екран створення оголошення

Для кнопки “СТВОРИТИ” було додано Intent для переходу на AppActivity. Код для цього аналогічний попереднім реалізаціям Intent.

3.3 Реалізація розмітки користувацького інтерфейсу

Для головного екрану було змінено файл activity_main.xml. XML у Android дозволяє вказати властивості для елементів на екрані. Код для activity_main.xml вказано нижче.

```
      <androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:app="http://schemas.android.com/apk/res-auto"
          xmlns:tools="http://schemas.android.com/tools"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          tools:context=".MainActivity">
          <Button
              android:id="@+id/logInBtn"
              android:layout_width="212dp"
              android:layout_height="64dp"
              android:onClick="onClickLogIn"
              android:text="@string/logIn"
              android:textColor="@color/black"
              app:layout_constraintBottom_toBottomOf="parent"
              app:layout_constraintEnd_toEndOf="parent"
              app:layout_constraintStart_toStartOf="parent"
              app:layout_constraintTop_toTopOf="parent" />
      </androidx.constraintlayout.widget.ConstraintLayout>
```

<Button> вказує на те, що цей елемент є кнопкою. Для нього було створено ідентифікатор. Також встановлено ширину та довжину у одиницях dp. В якості події onClick вказано метод onClickLogIn, який буде створено у наступному підрозділі. Також було встановлено колір тексту та Constraint.

Для екрану автентифікації мобільного телефону було створено макет activity_log.xml. Код макету наведено нижче.

```
<EditText
        android:id="@+id/enterPhone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="300dp"
        android:ems="10"
        android:hint="@string/enterPhone"
        android:inputType="phone"
        app:layout_constraintBottom_toTopOf="@+id/logIn"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/logIn"
        android:layout_width="212dp"
        android:layout_height="64dp"
        android:layout_marginBottom="300dp"
        android:onClick="onClickLog"
        android:text="@string/loginBtnLog"
        android:textColor="@color/black"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/enterPhone" />
```

<EditText> вказує на те, що цей елемент є полем для вводу. Для нього було створено ідентифікатор. Також встановлено ширину та довжину через обернення. Було додано верхній відступ, підказку, яка бере текст зі strings.xml, тип вводу – телефон та Constraint.

<Button> вказує на те, що цей елемент є кнопкою. Для нього було створено ідентифікатор. Також встановлено ширину та довжину у одиницях dp. В якості події onClick вказано метод onClickLog, який буде створено у наступному підрозділі. Також було встановлено колір тексту та Constraint.

Для екрану вводу коду було створено макет activity_phone.xml. Код макету наведено нижче.

```
<EditText
        android:id="@+id/enterPhone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="300dp"
        android:ems="10"
```

```
        android:hint="@string/enterCode"
        android:inputType="numberDecimal"
        app:layout_constraintBottom_toTopOf="@+id/logIn"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/logIn"
        android:layout_width="212dp"
        android:layout_height="64dp"
        android:layout_marginBottom="300dp"
        android:onClick="onClickVerify"
        android:text="@string/verifyCode"
        android:textColor="@color/black"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/enterPhone" />
```

<EditText> вказує на те, що цей елемент є полем для вводу. Для нього було створено ідентифікатор. Також встановлено ширину та довжину через обернення. Було додано верхній відступ, підказку, яка бере текст зі strings.xml, тип вводу – десятичне число та Constraint.

<Button> вказує на те, що цей елемент є кнопкою. Для нього було створено ідентифікатор. Також встановлено ширину та довжину у одиницях dp. В якості події onClick вказано метод onClickVerify, який буде створено у наступному підрозділі. Також було встановлено колір тексту та Constraint.

Для екрану з оголошеннями було створено макет activity_app.xml. Код макету наведено нижче.

```
<androidx.recyclerview.widget.RecyclerView
        android:id="@+id/itemsLayout"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        app:layout_constraintBottom_toTopOf="@id/buttonsLayout"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <LinearLayout
        android:id="@+id/buttonsLayout"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:orientation="horizontal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/itemsLayout">
        <Button
            android:id="@+id/filterBtn"
            android:layout_width="212dp"
            android:layout_height="64dp"
            android:layout_weight="1"
            android:onClick="onClickReload"
```

```
            android:text="@string/reloadBtn"
            android:textColor="@color/black" />
        <ImageButton
            android:id="@+id/profileBtn"
            android:layout_width="64dp"
            android:layout_height="64dp"
            android:onClick="onClickCreate"
            app:srcCompat="@android:drawable/ic_input_add" />
        <Button
            android:id="@+id/logOutBtn"
            android:layout_width="212dp"
            android:layout_height="64dp"
            android:layout_weight="1"
            android:onClick="onClickLogOut"
            android:text="@string/logOutBtn"
            android:textColor="@color/black" />
    </LinearLayout>
```

Усі кнопки мають ідентифікатор, ширину/довжину та власні методи, які будуть викликатися при натисканні на кнопку(також присутній ImageButton).

Елемент <RecyclerView> дозволяє відображати множину елементів. Тут він відображає елементи вертикально. Він має ідентифікатор, ширину/довжину, відступи та Constraint. У ньому знаходиться елемент <LinearLayout>, який грає роль контейнера одного елемента.

Він теж має ідентифікатор, ширину/довжину, відступи та Constraint. Також в ньому є властивість android:orientation зі значенням “horizontal”, що означає горизонтальне розташування елементів у ньому.

Для екрану зі створенням оголошеннями було реалізовано макет activity_create.xml. Код макету наведено нижче.

```
<EditText
        android:id="@+id/enterFirstName"
        android:layout_width="212dp"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:hint="@string/enterFirstName"
        app:layout_constraintBottom_toTopOf="@+id/enterLastName"/>

    <EditText
        android:id="@+id/enterLastName"
        android:layout_width="212dp"
        android:layout_height="42dp"
        android:gravity="center"
        android:hint="@string/enterLastName"
        app:layout_constraintBottom_toTopOf="@+id/enterPhone"
        app:layout_constraintTop_toBottomOf="@+id/enterFirstName" />

    <Spinner
        android:id="@+id/enterSubjectApp"
        android:layout_width="212dp"
        android:layout_height="wrap_content"
        android:background="@android:drawable/btn_dropdown"
        android:hint="@string/enterSubjectTxt"
        android:spinnerMode="dropdown"
        app:layout_constraintBottom_toTopOf="@+id/enterRole"
        app:layout_constraintTop_toBottomOf="@+id/enterPhone">
    </Spinner>
```

```
    <EditText
        android:id="@+id/enterPhone"
        android:layout_width="212dp"
        android:layout_height="wrap_content"
        android:hint="@string/enterPhone"
        android:inputType="phone"
        app:layout_constraintBottom_toTopOf="@+id/enterSubjectApp"
        app:layout_constraintTop_toBottomOf="@+id/enterLastName" />

    <Spinner
        android:id="@+id/enterRole"
        android:layout_width="212dp"
        android:layout_height="wrap_content"
        android:background="@android:drawable/btn_dropdown"
        android:spinnerMode="dropdown"
        android:textOff="@string/studentText"
        android:textOn="@string/tutorText"
        app:layout_constraintBottom_toTopOf="@+id/saveProfileBtn"
        app:layout_constraintTop_toBottomOf="@+id/enterSubjectApp">
    </Spinner>

    <Button
        android:id="@+id/saveProfileBtn"
        android:layout_width="212dp"
        android:layout_height="64dp"
        android:onClick="onClickCreate"
        android:text="@string/saveBtn"
        android:textColor="@color/black"
        app:layout_constraintTop_toBottomOf="@+id/enterRole" />
```

Макет має 3 текстових поля <EditText>, 2 елементи <Spinner> та кнопку.

3.4 Реалізація логіки для Android-застосунку

У Android Studio було створено декілька Activity-класів, а саме:

- MainActivity – головний екран;
- LogActivity – екран автентифікаціїї мобільного телефону;
- PhoneActivity – екран вводу коду з СМС;
- AppActivity – екран з оголошеннями;
- CreateActivity – екран для створення оголошень.

3.4.1 Створення логіки для MainActivity

Окрім Intent для переходу на інші екрани, головний екран має:

1. Функцію init, що ініціалізує кнопку та екземпляр FirebaseAuth. Код наведено нижче.

```
public void init(){
        logInBtn = findViewById(R.id.logInBtn);
        myAuth = FirebaseAuth.getInstance();
}
```

2. Перевірку на те, що користувач вже автентифікував телефон. Якщо так, то при вході у додаток відкривається екран AppActivity. Ця перевірка знаходиться в методі onCreate(). Код наведено нижче.

```
      if(myAuth.getCurrentUser()!=null){
                  Intent i = new Intent(MainActivity.this,
AppActivity.class);
                  startActivity(i);
                  finish();
      }
```

3.4.2 Створення логіки для LogActivity

Окрім Intent для переходу на інші екрани та функції init, екран автентифікації має:

1. Метод onClickLog, що оброблює натискання на кнопку та перевіряє чи не пустий ввод телефону.

Якшо пустий, то користувач отримує повідомлення, що телефон не може бути пустим. Якщо ввод не пустий, то користувач переходить на екран вводу коду з СМС та за допомогою методу putExtra номер з вводу передається на PhoneActivity. Код методу наведено нижче.

```
      public void onClickLog(View view){
              String number = enterPhone.getText().toString();
              if (!TextUtils.isEmpty(enterPhone.getText().toString())) {
                  Toast.makeText(LogActivity.this, "Будь ласка зачекайте",
Toast.LENGTH_SHORT).show();
                  Intent i = new Intent(LogActivity.this,
PhoneActivity.class);
                  i.putExtra("number", number);
                  startActivity(i);
              } else {
                  Toast.makeText(LogActivity.this, "Телефон не може бути
пустим", Toast.LENGTH_SHORT).show();
              }
          }
```

2. Слухач setOnFocusChangeListener, який оброблює фокус на ввод телефону. Якщо ввод у фокусі, то до вводу додається +380, якщо ні, то ввод стає пустим. Код наведено нижче.

```
      enterPhone.setOnFocusChangeListener(new View.OnFocusChangeListener() {
                  @Override
                  public void onFocusChange(View view, boolean hasFocus) {
                      if(hasFocus){
                          enterPhone.setText(R.string.enterPhoneUkraine);
```

```
                }
                else{
                    enterPhone.setText("");
                }
            }
        });
```

3.4.3 Створення логіки для PhoneActivity

Окрім Intent для переходу на інші екрани та функції init, екран вводу коду з СМС має:

1. Отримання Intent від LogActivity для телефону. Код для отримання наведено нижче.

```
Bundle extras = getIntent().getExtras();
        if (extras != null) {
            String number = extras.getString("number");
            sendCode(number);
}
```

2. Метод sendCode, який в якості аргументу отримує номер телефону. Тіло методу містить код з посібника Firebase Authentication для реалізації підключення провайдеру, що відправляє код на номер користувача. Код методу наведено нижче.

```
private void sendCode(String userPhone) {
        PhoneAuthOptions options =
                PhoneAuthOptions.newBuilder(myAuth)
                        .setPhoneNumber(userPhone)
                        .setTimeout(60L, TimeUnit.SECONDS)
                        .setActivity(this)
                        .setCallbacks(mCallbacks)
                        .build();
        PhoneAuthProvider.verifyPhoneNumber(options);
}
```

3. Об’єкт mCallbacks, який перезаписує різні методи PhoneAuthProvider, тобто провайдера автентифікації. Цей об’єкт перезаписує методи:

- onVerificationCompleted() – метод, що виконує деякий код, якщо веріфікацію було пройдено;
- onVerificationFailed() – метод, що виконує деякий код, якщо веріфікацію не було пройдено;
- onCodeSent() – метод, що виконує деякий код, якщо було відправлено код у СМС.

Код об’єкту наведено нижче.

```
      private PhoneAuthProvider.OnVerificationStateChangedCallbacks
mCallbacks = new PhoneAuthProvider.OnVerificationStateChangedCallbacks() {

              @Override
              public void onVerificationCompleted(@NonNull
PhoneAuthCredential credential) {
                  final String code = credential.getSmsCode();
                  if (code != null) {
                      verifyCode(code);
                  }
              }
              @Override
              public void onVerificationFailed(@NonNull FirebaseException e)
{
                  Toast.makeText(PhoneActivity.this, "Верифікація не
пройшла", Toast.LENGTH_SHORT).show();
              }

              @Override
              public void onCodeSent(@NonNull String s,
                                     @NonNull
PhoneAuthProvider.ForceResendingToken token) {
                  super.onCodeSent(s, token);
                  verifyID = s;}
          };
```

4. Метод onClickVerify, який оброблює натискання на кнопку перевірки коду. Якщо ввод коду не був пустим, то викликається метод verifyCode, якщо пустим, то користувач отримує повідомлення, що ввод не може бути пустим. Код методу наведено нижче.

```
      public void onClickVerify(View view){
              if (!TextUtils.isEmpty(enterCode.getText().toString())) {
                  Toast.makeText(PhoneActivity.this, "Зачекайте",
Toast.LENGTH_SHORT).show();
                  verifyCode(enterCode.getText().toString());
              } else {
                  Toast.makeText(PhoneActivity.this, "Неправильний код",
Toast.LENGTH_SHORT).show();
              }

          }
```

5. Метод verifyCode, який створює новий об’єкт типу PhoneAuthCredential, що отримує дані – ID для верифікації та сам код з PhoneAuthProvider. Також метод викликає метод signinByCredentials, який у подальшому буде виконувати вхід користувача до акаунту. Код методу наведено нижче.

```
      private void verifyCode(String code) {
              PhoneAuthCredential credential =
PhoneAuthProvider.getCredential(verifyID, code);
              signinByCredentials(credential);

          }
```

6. Метод signinByCredentials, який отримує об’єкт типу PhoneAuthCredential, ініціалізує FirebaseAuth та виконує вхід до акаунту. Код методу наведено нижче.

```
      private void signinByCredentials(PhoneAuthCredential credential) {
              FirebaseAuth fbAuth = FirebaseAuth.getInstance();
              fbAuth.signInWithCredential(credential)
                      .addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
                          @Override
                          public void onComplete(@NonNull Task<AuthResult>
task) {
                              Toast.makeText(PhoneActivity.this, "Успішна
авторизація", Toast.LENGTH_SHORT).show();
                              startActivity(new Intent(PhoneActivity.this,
AppActivity.class));
                              finish();
                          }

                      });
      }
```

3.4.4 Створення логіки для AppActivity

Окрім Intent для переходу на інші екрани та функції init, екран з оголошеннями має:

1. Метод getData(), який у подальшому передає дані у класс AdModel та за допомогою цього класу відображає дані на RecyclerView(макет, який зберігає оголошення та за допомогою макету ad_layout створює нові елементи зверху вниз). Код методу наведено нижче.

```
      private void getData(){
              ValueEventListener v_listener = new ValueEventListener() {
                  @Override
                  public void onDataChange(@NonNull DataSnapshot snapshot) {
                      if (ad_data.size() > 0) ad_data.clear();
                      for(DataSnapshot ds : snapshot.getChildren()){
                          Ad ad = ds.getValue(Ad.class);
                          assert ad != null;
                          adModelArrayList.add(new AdModel(ad.firstName,
ad.lastName, ad.subject, ad.phoneNumber, ad.userRole));
                      }
                      adAdapter.notifyDataSetChanged();
                  }
                  @Override
                  public void onCancelled(@NonNull DatabaseError error) {

           }
              };
              myDatabase.addValueEventListener(v_listener);
          }
```

2. Метод onClickLogout, що оброблює натискання на кнопку виходу з акаунту. Якщо користувач натисне на цю кнопку, то викликається метод signOut() через об’єкт myAuth типу FirebaseAuth. Код методу наведено нижче.

```
      public void onClickLogOut(View view){
              Toast.makeText(AppActivity.this, "Натисніть ще раз, щоб вийти",
Toast.LENGTH_SHORT).show();
              logoutBtn.setOnClickListener(new View.OnClickListener() {
                  @Override
                  public void onClick(View view) {
                      myAuth.signOut();
                      startActivity(new Intent(AppActivity.this,
MainActivity.class));
                      finish();
                  }
              });
          }
```

3.4.5 Створення логіки для CreateActivity

Окрім Intent для переходу на інші екрани та функції init для ініціалізації елементів користувацького інтерфейсу, екран з додаванням оголошень має:

1. Метод onClickCreate, який оброблює натискання на створення оголошення. При натисканні створюються об’єкти id, firstName, lastName, subject, phoneNumber та userRole, які отримують дані із вводу користувача. Також створюється новий об’єкт типу Ad для запису цих даних у базу даних та подальшій передачі цих даних до AppActivity. Код методу наведено нижче.

```
      public void onClickCreate(View view){
              String id = UUID.randomUUID().toString();
              String firstName = enterFirstName.getText().toString();
              String lastName = enterLastName.getText().toString();
              String subject = enterSubject.getSelectedItem().toString();
              String phoneNumber = enterPhone.getText().toString();
              String userRole = enterRole.getSelectedItem().toString();
              Ad newAd = new Ad(id, firstName, lastName, subject,
phoneNumber, userRole);
              if(!TextUtils.isEmpty(firstName) &&
!TextUtils.isEmpty(lastName) && !TextUtils.isEmpty(subject) &&
!TextUtils.isEmpty(phoneNumber)){
                  myDatabase.push().setValue(newAd);
                  Intent i = new Intent(CreateActivity.this,
AppActivity.class);
                  i.putExtra(Constant.FIRSTNAME, firstName);
                  i.putExtra(Constant.LASTNAME, lastName);
                  i.putExtra(Constant.SUBJECT, subject);
                  i.putExtra(Constant.PHONE, phoneNumber);
                  i.putExtra(Constant.ROLE, userRole);
                  startActivity(i);
                  finish();
              }

              else{
                  Toast.makeText(CreateActivity.this, "Заповніть всі поля",
Toast.LENGTH_SHORT).show();}}
```

4 ТЕСТУВАННЯ РОБОТИ ДОДАТКУ

Додаток було протестовано на декілька пристроях. На усіх пристроях Infomage працював без збоїв. Нижче наведено тестування роботи додатку.

4.1 Тестування Android-застосунку на смартфоні Meizu X8

1. Було відкрито сам додаток. Так як мобільний телефон ще не було автентифіковано - завантажився головний екран. Це наведено на рисунку 4.1.

Рисунок 4.1 – Головний екран додатку Infomage

2. Користувач натиснув на “УВІЙТИ”. Відкрився екран автентифікації. У поле вводу було введено мобільний телефон. Це наведено на рисунку 4.2.

Рисунок 4.2 – Введення телефону на екрані автентифікації користувача

3. Після введення користувач потрапляє на екран вводу коду з СМС. Це наведено на рисунку 4.3.

Рисунок 4.3 – Екран вводу коду з СМС

4. Через декілька секунд відкривається екран з підтвердженням, що користувач – не робот за допомогою Captcha. Це наведено на рисунку 4.4.

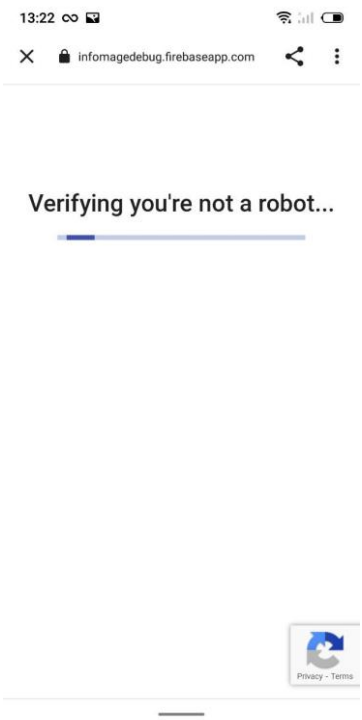

Рисунок 4.4 – Екран підтвердження, що користувач - не робот

5. Після підтвердження користувач отримує код у СМС, який він у подальшому може вписати у поле ввода. Відправку наведено на рисунку 4.5.

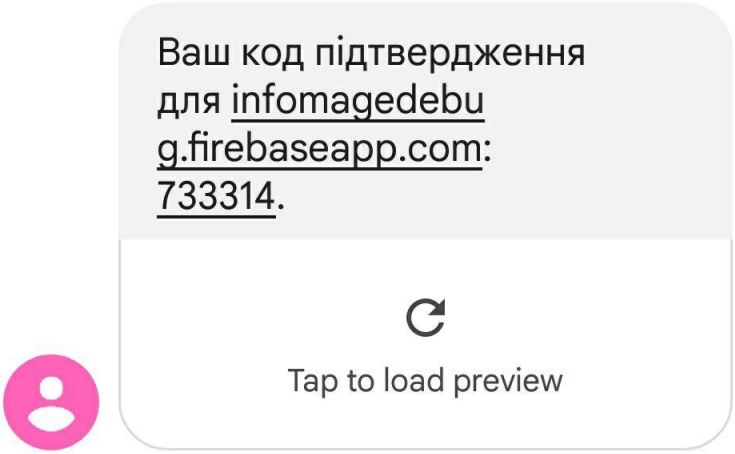

Рисунок 4.5 – Код було успішно відправлено через СМС

6. Тепер користувач може вписати код у поле вводу. Це наведено на рисунку 4.6.

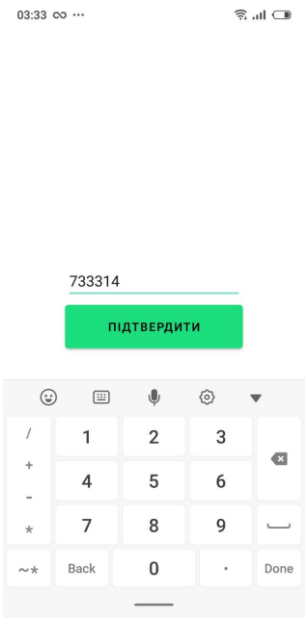

Рисунок 4.6 – Введення коду у поле

7. Після натискання на “ПІДТВЕРДИТИ”, якщо код правильний, то користувач переходить на екран з оголошеннями. Це наведено на рисунку 4.7.

Рисунок 4.7 – Екран з оголошеннями

8. Оголошень ще немає, тож спробуємо додати одне. Натиснувши на “+”, користувач переходить на екран з додаванням оголошень. Це наведено на рисунку 4.8.

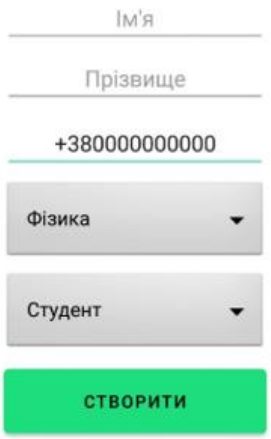

Рисунок 4.8 – Екран додавання оголошень

9. Введемо ім’я, прізвище. Телефон вже введено той, що ми автентифікували, але ми можемо його змінити. Також оберемо предмет та роль – студент або репетитор. Створення тестового оголошення наведено на рисунку 4.9.

Рисунок 4.9 – Створення тестового оголошення

10. Після натискання на “СТВОРИТИ”, користувач переходить на екран з оголошеннями та отримує новий макет оголошення з даними із Firebase Realtime Database. Це наведено на рисунку 4.10.

Рисунок 4.10 – Екран з оголошеннями після створення оголошення

11. Оглянемо також базу даних Firebase. Було створено новий дочірній елемент типу Ad із випадковим ідентифікатором. Це наведено на рисунку 4.11.

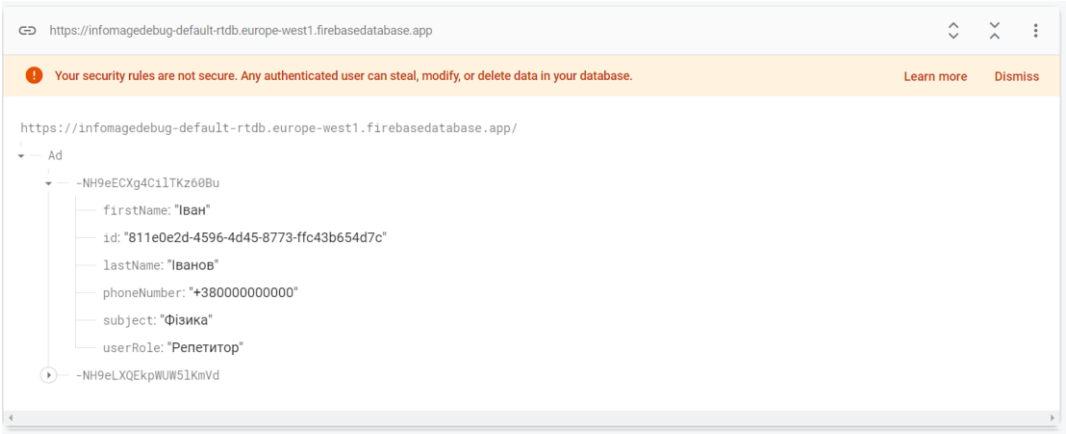

Рисунок 4.11 – Відображення даних нового оголошення у Firebase Realtime Database

12. Оглянемо також список автентифікованих мобільних телефонів. Кожен телефон – це окремий акаунт користувача. Список наведено на рисунку 4.12.

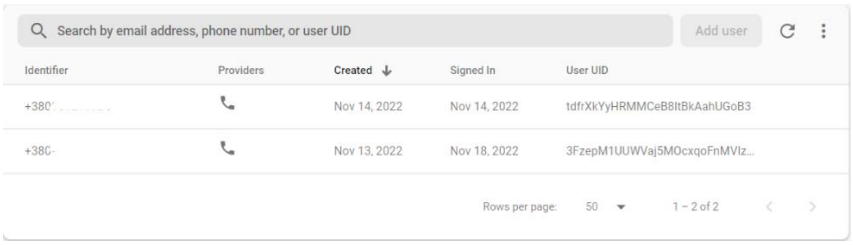

Рисунок 4.12 – Список автентифікованих користувачів у Firebase

Отже, було протестовано мобільний застосунок Infomage, який має автентифікацію мобільного телефону та додавання даних у реальному часі на смартфоні Meizu X8.

4.2 Тестування Android-застосунку на смартфоні OnePlus 7T

1. Так як мобільний телефон ще не було автентифіковано - завантажився головний екран. Це наведено на рисунку 4.13.

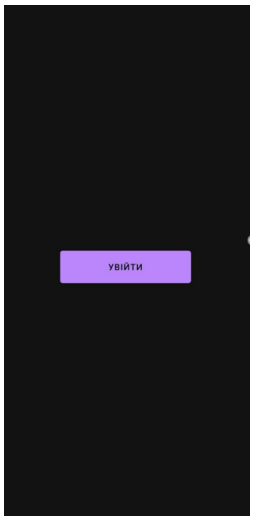

Рисунок 4.13 – Головний екран додатку Infomage

2. Користувач натиснув на “УВІЙТИ”. Відкрився екран автентифікації. У поле вводу було введено мобільний телефон. Це наведено на рисунку 4.14.

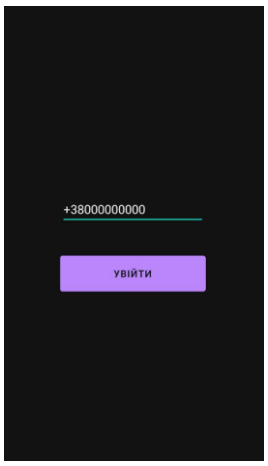

Рисунок 4.14 – Введення телефону на екрані автентифікації

3. Користувач пройшов перевірку, що він не робот. Це показано на рисунку 4.15.

Рисунок 4.15 – Перевірка на бота

5. Після підтвердження користувач отримує код у СМС, який він у подальшому може вписати у поле ввода. Відправку наведено на рисунку 4.16.

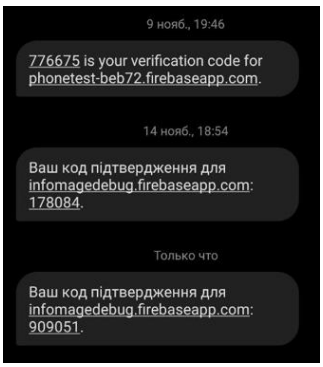

Рисунок 4.16 – Відправка коду у СМС

6. Після натискання на “ПІДТВЕРДИТИ”, якщо код правильний, то користувач переходить на екран з оголошеннями. Це наведено на рисунку 4.17.

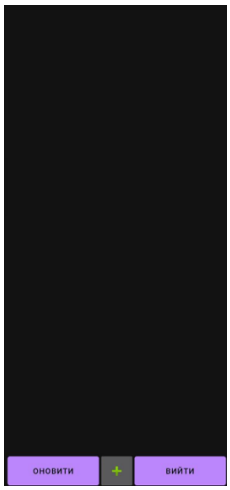

Рисунок 4.17 – Користувач перейшов на екран з оголошеннями

7. Оголошень ще немає, тож спробуємо додати одне. Натиснувши на “+”, користувач переходить на екран з додаванням оголошень. Це наведено на рисунку 4.18.

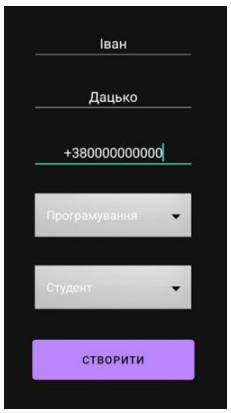

Рисунок 4.18 – Користувач вводить дані для нового оголошення

8. Після натискання на “СТВОРИТИ” користувач переходить на попередній екран і бачить нове оголошення. Це показано на рисунку 4.19.

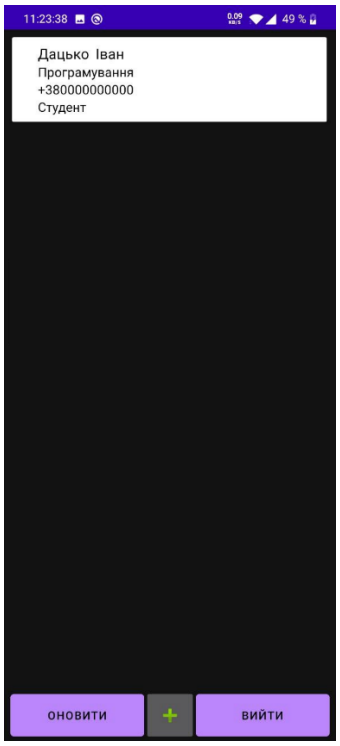

Рисунок 4.19 – Нове оголошення було створено

Отже, було протестовано мобільний застосунок Infomage, який має автентифікацію мобільного телефону та додавання даних у реальному часі на смартфоні OnePlus 7T.

Для тестування мобільного застосунку було використано методику функціонального тестування.

Функціональне тестування мобільних додатків – це процес тестування функціональних можливостей мобільних додатків, таких як взаємодія користувачів, а також тестування транзакцій, які можуть виконувати користувачі. Основною метою функціонального тестування мобільних додатків є забезпечення якості, відповідність заданим очікуванням, зниження ризику або помилок і задоволення клієнтів.

Різноманітні фактори, які мають значення для функціонального тестування:

- тип програми на основі використання бізнес-функцій (банківська, ігрова, соціальна чи бізнес)
- тип цільової аудиторії (споживач, підприємство, освіта)
- канал розповсюдження, який використовується для поширення програми (наприклад, Apple App Store, Google play, пряме розповсюдження)

Найбільш фундаментальними тестовими сценаріями у функціональному тестуванні можна вважати:

- перевірка, чи всі необхідні обов’язкові поля працюють належним чином.
- перевірка, що обов’язкові поля відображаються на екрані в інший спосіб, ніж необов’язкові поля.
- перевірка, чи програма працює відповідно до вимог кожного разу, коли програма запускається/зупиняється.
- перевірка, що пристрій здатний виконувати необхідні багатозадачні вимоги, коли це необхідно.
- перевірка, що навігація між відповідними модулями в додатку відповідає вимогам.

За приведеними тестовими сценаріями застосунок Infomage пройшов перевірку.

ВИСНОВКИ

З моменту свого першого комерційного випуску у вересні 2008 року мобільна операційна система Android стала свідком стабільного впровадження виробничою промисловістю, користувачами мобільних пристроїв і спільнотою розробників програмного забезпечення.

Оскільки додатки для Android зараз пронизують усі дії користувачів, погано розроблені та шкідливі додатки стали великою загрозою, яка може призвести до збитків різного ступеня тяжкості.

Таким чином, незважаючи на значні зусилля спільноти, найсучасніші інструменти все ще стикаються з проблемами через відсутність підтримки деяких функцій аналізу.

Отже, внаслідок роботи над кваліфікаційною роботою було:

1. Визначено методологією розробки мобільних додатків.

2. Вивчено документацію сучасних інструментів розробки.

3. Застосовано отримані знання під час розробки програми.

У застосунок було інтегровано Firebase Authentication – сервіс для автентифікації користувача, та Firebase Realtime Database – сервіс для створення бази даних у реальному часі.

Для розробки Android-застосунку було використано інтегроване середовище розробки Android Studio від Google.

За допомогою програмного забезпечення для створення макетів Figma було реалізовано макети, які у подальшому було перенесено у Android Studio.

Мету та завдання кваліфікаційної роботи було виконано.

Результатом виконання роботи є реалізований застосунок для операційної системи Android, який можна використовувати в якості методичних вказівок для вивчення об’єктно-орієнтованого програмування, проектування програмного забезпечення для мобільних пристроїв, а також алгоритмізації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Designing Mobile Alumni Tracer Study System Using Waterfall Method: an Android Based [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://cutt.ly/v1hFuvB

2. Kaleel S. Applying Agile Methodology in Mobile Software Engineering: Android Application Development and its Challenges / Shakira Banu Kaleel., 2013. – 11 с. – (Digital Commons).

3. Radack S. THE SYSTEM DEVELOPMENT LIFE CYCLE [Електронний ресурс] / Shirley Radack – Режим доступу до ресурсу: https://cutt.ly/Z1hGey5

4. Dehlinger J. Mobile Application Software Engineering: Challenges and Research Directions [Електронний ресурс] / J. Dehlinger, J. Dixon // ACADEMIA – Режим доступу до ресурсу: https://cutt.ly/b1hGvJt

5. Wasserman A. Software engineering issues for mobile application development [Електронний ресурс] / Anthony Wasserman. – 2010. – Режим доступу до ресурсу: https://cutt.ly/g1hG9L0

6. Salmre I. Writing Mobile Code: Essential Software Engineering for Building Mobile Applications [Електронний ресурс] / Ivo Salmre. – 2005. – Режим доступу до ресурсу: https://cutt.ly/u1hHfmY

7. Software Engineering Challenges in Multi Platform Mobile Application Development [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://cutt.ly/51hHvOb

8. Mobile software engineering in mobile computing curriculum [Електронний ресурс] // IEEE. – 2013. – Режим доступу до ресурсу: https://cutt.ly/Q1hHBeT

9. Future Trends in Software Engineering Research for Mobile Apps [Електронний ресурс] // IEEE. – 2016. – Режим доступу до ресурсу: https://cutt.ly/t1hJewq

10. Balagtas-Fernandez F. A Methodology and Framework to Simplify Usability Analysis of Mobile Applications [Електронний ресурс] / F. Balagtas-

Fernandez, H. Hussmann // IEEE. – 2009. – Режим доступу до ресурсу: https://cutt.ly/T1hJHmB

11. Kangas E. Applying user-centered design to mobile application development [Електронний ресурс] / Eeva Kangas // ACM. – 2005. – Режим доступу до ресурсу: https://cutt.ly/u1hJ7RY

12. The iMPAcT Tool: Testing UI Patterns on Mobile Applications [Електронний ресурс] // IEEE. – 2015. – Режим доступу до ресурсу: https://cutt.ly/F1hKsAP

13. Testing Approach for Mobile Applications through Reverse Engineering of UI Patterns [Електронний ресурс] // IEEE. – 2015. – Режим доступу до ресурсу: https://cutt.ly/a1hKcVT

14. Baldini I. Cloud-native, event-based programming for mobile applications [Електронний ресурс] / I. Baldini, P. Castro, P. Cheng // ACM. – 2016. – Режим доступу до ресурсу: https://cutt.ly/M1hKKvc

15. Approaches to mobile application development: Comparative performance analysis [Електронний ресурс] // IEEE. – 2017. – Режим доступу до ресурсу: https://cutt.ly/E1hLqdB

16. Understanding Android Security [Електронний ресурс] // IEEE. – 2009. – Режим доступу до ресурсу: https://cutt.ly/N1hLVqY

17. Research on Development of Android Applications [Електронний ресурс] // IEEE. – 2011. – Режим доступу до ресурсу: https://cutt.ly/N1hZycw

18. Analysis of the Android Architecture [Електронний ресурс] // KIT. – 2010. – Режим доступу до ресурсу: https://cutt.ly/D1hZm2a

19. Android: Changing the Mobile Landscape [Електронний ресурс] // IEEE. – 2010. – Режим доступу до ресурсу: https://cutt.ly/w1hZO9P

20. Comparative analysis of Android and iOS from security viewpoint [Електронний ресурс] // ScienceDirect. – 2021. – Режим доступу до ресурсу: https://cutt.ly/q1hZ0gE

21. Statistical Deobfuscation of Android Applications [Електронний ресурс] // ACM. – 2016. – Режим доступу до ресурсу: https://cutt.ly/11hXcqd

22. Improving Acknowledgement in Android Application [Електронний ресурс] // ingenta. – 2019. – Режим доступу до ресурсу: https://cutt.ly/n1hXTsm

23. Nadkarni B. Practical DIFC Enforcement on Android [Електронний ресурс] / B. Nadkarni, A. Andow, W. Enck // USENIX – Режим доступу до ресурсу: https://cutt.ly/v1hXGtG

24. The Android Platform Security Model [Електронний ресурс] // ACM. – 2021. – Режим доступу до ресурсу: https://cutt.ly/i1hConN

25. Introducción a Android [Електронний ресурс] // 2008 – Режим доступу до ресурсу: https://cutt.ly/21hCRLA

26. An Analysis of Pre-installed Android Software [Електронний ресурс] // IEEE. – 2020. – Режим доступу до ресурсу: https://cutt.ly/j1hCC7F

27. Development Of An Android Application In The Form Of A Simulation Lab As Learning Media for Senior High School Students [Електронний ресурс] // MODESTUM. – 2015. – Режим доступу до ресурсу: https://cutt.ly/41hC2iQ

28. Android Application Development: A Brief Overview of Android Platforms and Evolution of Security Systems [Електронний ресурс] // IEEE. – 2019. – Режим доступу до ресурсу: https://cutt.ly/S1hVeuQ

29. Firebase Authentication [Електронний ресурс] // Google – Режим доступу до ресурсу: https://cutt.ly/31hVjEV

30. Firebase Realtime Database [Електронний ресурс] // Google – Режим доступу до ресурсу: https://cutt.ly/I1hVWLT