ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АНДРОЇД ЗАСТОСУНКУ ДЛЯ ГЕНЕРАЦІЇ ВІЗИТІВОК

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістра

Виконавець роботи Кизименко Сергій Анатолійович
_____ «____»_____ 2025 р.
(підпис)

Науковий керівник к.пед.н., доцент, Кошова Оксана Петрівна
_____ «____»_____ 2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2023 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка програмного забезпечення андроїд застосунку для генерації візитівок**»**
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня магістр
Прізвище, ім'я, по батькові Кизименко Сергій Анатолійович
Затверджена наказом ректора № 206-Н від «27» жовтня 2023 р.
Термін подання студентом роботи «___»_____ 2024 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки андроїд застосунків.

Зміст пояснювальної записки (перелік питань, які потрібно розробити)

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд сучасних рішень для генерації візитівок
2.2. Порівняльний аналіз популярних застосунків для генерації візитівок
2.3. Переваги та недоліки існуючих рішень
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Опис проектних рішень, інструментів та підходів до розробки застосунку.
3.2. Опис методології створення програмного забезпечення та побудова діаграми послідовностей застосунку
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка алгоритму роботи застосунку для генерації візитівок
4.2. Опис та побудова діаграми класів розробленого застосунку.
4.3. Інтерфейс користувача та функціональні можливості застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Перелік графічного матеріалу: 1 аркуш діаграм, 15 ілюстрацій.

Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Кошова О.П.		
Інформаційний огляд	Кошова О.П.		
Теоретична частина	Кошова О.П.		
Практична частина	Кошова О.П.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2023 р.

Здобувач вищої освіти _____ Кизименко Сергій Анатолійович
(підпис)

Науковий керівник _____ к.пед.н., доц. Кошова О.П.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)

Результати захисту кваліфікаційної роботи

Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)

Протокол засідання ЕК № ____ від «____» _____ 2025 р.

Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2023 р.

Погоджено
Науковий керівник _____
к.пед.н., Оксана КОШОВА
«____» _____ 2023 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Розробка програмного забезпечення андроїд застосунку для генерації візитівок»
Прізвище, ім'я, по батькові Кизименко Сергій Анатолійович
ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Огляд сучасних рішень для генерації візитівок
2.2. Порівняльний аналіз популярних застосунків для генерації візитівок
2.3. Переваги та недоліки існуючих рішень
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Опис проектних рішень, інструментів та підходів до розробки застосунку.
3.2. Опис методології створення програмного забезпечення та побудова діаграми послідовностей застосунку
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Розробка алгоритму роботи застосунку для генерації візитівок
4.2. Опис та побудова діаграми класів розробленого застосунку.
4.3. Інтерфейс користувача та функціональні можливості застосунку
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А

Здобувач вищої освіти _____ С.А. Кизименко
(підпис)
«____» _____ 2023 р.

РЕФЕРАТ

Записка: 76 сторінок, 15 рисунків, 1 додаток, 15 літературних джерел.

Мета дипломної роботи є розробка програмного забезпечення андроїд застосунк для генерації візитівок.

Об'єкт розробки – андроїд застосунок для генерації візитівок.

Методи дослідження – проектування мобільних додатків, зокрема методики UX/UI дизайну. Також використано архітектурний шаблон MVVM (Model-View-ViewModel). Застосунок розроблений із використанням Clean Architecture. Залежності впроваджуються за допомогою фреймворку Koin, який реалізує Dependency Injection, що забезпечує більшу гнучкість та тестованість коду.; застосування Data та View Binding спрощує роботу з елементами інтерфейсу і даними. Використання Navigation Components дозволяє ефективно управляти навігацією між екранами додатку. Плагіни Kotlin і AndroidX надають додаткові можливості і оптимізації для роботи з Kotlin і Android Jetpack. Додаток розроблено у середовищі розробки Android Studio.

Визначено ключові вимоги до розробки програмного забезпечення андроїд застосунку для генерації візитівок; проведено аналіз переваг та обмежень аналогічних застосунків та джерел отримання візитівок; проаналізовано та описано підходи та інструменти, використані при розробці ПЗ застосунку; описано алгоритм, методолію створення та діаграму послідовностей розробленого андроїд застосунку; розроблено мобільний інтерфейс для взаємодії користувача з матеріалом та створено докладну інструкцію користувача; представлено діаграму класів, що ілюструє особливості програмного коду розробленого застосунку; розроблено та протестовано андроїд застосунок для генерації візитівок.

ЗМІСТ

ВСТУП

Актуальність. У сучасному світі цифрових технологій розвиток мобільних пристроїв та програмного забезпечення став невід’ємною частиною повсякденного життя. Одним із ключових напрямів є розробка мобільних застосунків, які спрощують виконання рутинних завдань і забезпечують зручний доступ до різноманітних послуг. Візитівки, як інструмент професійної комунікації, залишаються важливим елементом ділового етикету, проте їх фізичний формат поступово витісняється цифровими аналогами. Цифрові візитівки мають значні переваги, зокрема, легкість у створенні, оновленні та поширенні, що робить їх більш адаптованими до вимог сучасного ділового середовища.

Розробка Android-застосунку для генерації візитівок є актуальною через зростаючу популярність мобільних платформ, зокрема операційної системи Android, яка займає провідне місце на ринку мобільних пристроїв. Такий застосунок дозволить користувачам швидко створювати персоналізовані цифрові візитівки, які відповідають сучасним стандартам графічного дизайну та інформаційної структури. Крім того, автоматизація процесу генерації візитівок сприятиме мінімізації помилок і заощадженню часу користувачів, забезпечуючи простоту використання навіть для людей без спеціальних технічних знань.

Необхідність розробки такого програмного забезпечення зумовлена не лише технічними тенденціями, а й зміною підходів до комунікації в бізнес-середовищі. Зростання популярності безконтактних технологій і необхідність миттєвого обміну інформацією підсилюють актуальність впровадження інструментів, які забезпечують зручний спосіб обміну контактними даними. Таким чином, розробка Android-застосунку для генерації візитівок є важливою задачею, що відповідає сучасним вимогам мобільного програмного забезпечення та потребам користувачів у сфері професійної взаємодії.

Метою даної роботи є розробка програмного зпбезпечення андроїд застосунку для генерації візитівок. Для досягнення цієї мети передбачено вирішення наступних завдань: аналіз існуючих рішень та технологій у сфері автоматичної генерації документів та візитівок, розробка алгоритму роботи застосунку, створення програмного забезпечення для андроїд застосунку, його тестування та оптимізація.

Об'єктом дослідження є андроїд застосунок для генерації візитівок, а *предметом* –програмне забезпечення андроїд застосунку для генерації візитівок. Для досягнення поставлених цілей у роботі буде використано методи збору та аналізу даних, а також програмні засоби для розробки програмного забезпечення.

Перелік методів включає проектування мобільних додатків, зокрема методики UX/UI дизайну. Застосунок для генерації візитівок «Business Card Creator» розроблений для Android, використовуючи сучасні бібліотеки та інструменти. Мінімально дозволена версія Android для додатку - SDK 24, що відповідає Android 7.0, а цільова - SDK 34 (Android 14). Застосування Data та View Binding спрощує роботу з елементами інтерфейсу і даними. Використання Navigation Components дозволяє ефективно управляти навігацією між екранами додатку. Koin використовується для впровадження залежностей (Dependency Injection), що забезпечує більшу гнучкість та тестованість коду. Плагіни Kotlin і AndroidX надають додаткові можливості і оптимізації для роботи з Kotlin і Android Jetpack. Додаток розроблено у середовищі розробки Android Studio.

Додаток має дозволи для роботи з зовнішнім сховищем. Використання `FileProvider` забезпечує безпеку при роботі з файлами, обмежуючи доступ до файлів додатку для інших додатків.

Пояснювальна записка включає чотири основних розділи: постановка задачі, огляд сучасного стану проблеми, теоретична та практична частини. Структура роботи орієнтована на логічне представлення матеріалу та всебічне розкриття теми дослідження.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

Мета застосунку

Метою розробки Android-застосунку для генерації візитівок є надання користувачам простого та зручного інструменту для створення, редагування й поширення власних візитівок. Така цифрова альтернатива традиційним друкованим карткам дає змогу швидко ділитися контактними даними та особистою інформацією без використання паперу, тим самим долучаючись до сучасних екологічних ініціатив. Окрім того, можливість індивідуалізувати дизайн сприяє розвитку креативності та персонального бренду студента.

У процесі розробки застосунку ми зможемо застосувати свої знання з програмування, дизайну та проєктування інтерфейсів користувача (UI/UX). За допомогою вбудованих бібліотек для створення та обробки зображень користувачі матимуть змогу обирати шаблони, змінювати кольорову гаму, додавати логотипи чи фото, а також коригувати текстові блоки із власними даними. Використання Firebase чи інших хмарних сервісів дозволить надійно зберігати створені візитівки та ділитися ними через різноманітні канали комунікації, від електронної пошти до соціальних мереж.

Додатково, застосунок можна в майбутньому інтегрувати із системами розпізнавання QR-кодів, що спростить обмін контактами: достатньо відсканувати код, аби миттєво завантажити інформацію до телефонної книги. Це стане в пригоді на виставках, конференціях чи при спілкуванні з потенційними роботодавцями, допомагаючи студенту заявити про себе та демонструючи його технологічну обізнаність.

Таким чином, Android-застосунок для генерації візитівок вирішує одразу декілька завдань: він допомагає закріпити теоретичні знання з програмування, розробки дизайну та UI/UX, підсилює професійне резюме розробника, а також робить процес налагодження контактів максимально зручним та сучасним.

Основні функції застосунку

Розробка Android-застосунку для генерації візитівок вимагає впровадження широкого спектра функціональних можливостей, що забезпечать зручність використання, гнучкість налаштування та відповідність сучасним вимогам мобільного програмного забезпечення. Основним завданням додатку є надання користувачам інструментів для створення та редагування цифрових візитівок. Ця функція передбачає можливість введення персональних даних, таких як ім’я, посада, контактна інформація та посилання на соціальні мережі. Інтерфейс додатку повинен підтримувати вибір із низки шаблонів, які автоматично адаптуються до введених даних, забезпечуючи індивідуальність та професійний вигляд візитівки.

Критичною функцією є динамічний попередній перегляд, що дозволяє в режимі реального часу оцінювати результати внесених змін. Завдяки використанню технологій Data та View Binding інтеграція елементів інтерфейсу з даними значно спрощується, забезпечуючи стабільність та ефективність роботи застосунку. Для забезпечення збереження результатів необхідно реалізувати можливість зберігання візитівок у популярних форматах, таких як PDF чи зображення, з подальшою їх інтеграцією у внутрішнє або зовнішнє сховище. Використання механізму FileProvider гарантує безпечний доступ до створених файлів, мінімізуючи ризики несанкціонованого використання.

Функціонал обміну візитівками є ще одним важливим елементом застосунку. Він повинен включати можливість передачі файлів через різноманітні канали, такі як електронна пошта, месенджери чи QR-коди. Останній підхід значно прискорює процес обміну контактною інформацією, відповідаючи сучасним трендам ділової комунікації.

Ефективність навігації між екранами додатку забезпечується за допомогою Navigation Components, що дозволяє створити інтуїтивно зрозумілий інтерфейс із підтримкою стеку навігації. Особливу увагу варто

приділити локалізації, оскільки підтримка багатомовного інтерфейсу підвищує доступність застосунку для користувачів з різних країн. Налаштування програми повинні передбачати гнучкий вибір параметрів збереження файлів, форматів візитівок, а також кастомізації дизайну, що відповідає вимогам індивідуальних користувачів.

Розробка застосунку спирається на використання сучасних інструментів, таких як Koin, що дозволяє ефективно впроваджувати залежності та забезпечує високу тестованість коду. Застосування AndroidX і Kotlin сприяє оптимізації роботи з функціоналом Android Jetpack та підвищує продуктивність розробки. Врахування мінімальних вимог до операційної системи Android 7.0 (SDK 24) та підтримка актуальних версій до Android 14 (SDK 34) гарантує сумісність з широким спектром пристроїв.

Отже, застосунок "Business Card Creator" буде розроблено для Android, використовуючи сучасні бібліотеки та інструменти. Мінімально дозволена версія Android для додатку - SDK 24, що відповідає Android 7.0, а цільова - SDK 34 (Android 14). Застосування Data та View Binding спрощує роботу з елементами інтерфейсу і даними. Використання Navigation Components дозволяє ефективно управляти навігацією між екранами додатку. Koin буде використовуватися для впровадження залежностей (Dependency Injection), що забезпечує більшу гнучкість та тестованість коду. Плагіни Kotlin і AndroidX нададуть додаткові можливості і оптимізації для роботи з Kotlin і Android Jetpack.

Додаток буде мати дозволи для роботи з зовнішнім сховищем. Використання `FileProvider` забезпечує безпеку при роботі з файлами, обмежуючи доступ до файлів додатку для інших додатків.

Таким чином, впровадження вищезазначених функцій та технологій дозволяє створити інноваційний інструмент, що відповідає сучасним стандартам мобільного програмного забезпечення та забезпечує користувачам зручність і гнучкість у створенні цифрових візитівок.

РОЗДІЛ 2. ОГЛЯД СИСТЕМ АНАЛОГІЧНОГО ПРИЗНАЧЕННЯ

2.1. Огляд сучасних сучасних рішень для генерації візитівок

Сучасні рішення для генерації електронних візитівок та бізнес-карток пропонують широкий спектр функціональності, орієнтованої на автоматизацію, зручність використання, індивідуалізацію та інтеграцію з іншими цифровими платформами. Ці застосунки спрямовані на створення професійно оформлених візитівок із можливістю подальшого поширення у фізичному або електронному форматі. Серед популярних та інноваційних рішень можна виділити наступні категорії застосунків.

Десктопні рішення.

Десктопні програми для створення бізнес-карток, такі як Adobe Illustrator, CorelDRAW та Canva, є потужними інструментами для створення високоякісного дизайну з багатими можливостями налаштувань. Вони дозволяють користувачам створювати візитівки з використанням професійних шаблонів, власних зображень та логотипів. Особливою перевагою таких програм є їхня здатність підтримувати складні векторні дизайни, які можна друкувати у високій якості.

Онлайн-платформи.

Онлайн-сервіси, такі як Visme, Vistaprint, Zazzle, Business Card Maker, та Crello, надають інструменти для швидкого створення візитівок без необхідності встановлення додаткового програмного забезпечення. Вони часто пропонують бібліотеки готових шаблонів, які можна налаштовувати під конкретні потреби. Такі сервіси також інтегруються з друкарськими послугами, дозволяючи замовляти фізичні копії створених карток.

Мобільні застосунки.

Мобільні рішення, наприклад, CamCard, Haystack, Blinq та HiHello, дозволяють створювати та обмінюватися електронними візитівками

безпосередньо через смартфони. Ці застосунки надають можливість сканування фізичних карток із автоматичним перетворенням їх у цифровий формат, інтеграції з контактами телефону та соціальними мережами. Крім того, вони забезпечують миттєвий обмін картками через QR-коди, посилання або NFC.

Інтегровані системи.

Деякі компанії пропонують рішення, інтегровані в ширші екосистеми. Наприклад, Microsoft Outlook та Google Contacts дозволяють зберігати контактні дані у цифровому форматі, який можна використовувати як бізнес-картку. LinkedIn також має функціонал створення професійних профілів, які фактично замінюють візитівки в корпоративному середовищі.

Інноваційні рішення.

Новітні технології сприяють появі унікальних рішень, таких як інтерактивні бізнес-картки на основі NFC-чіпів або AR-технологій (доповнена реальність). Ці інструменти дозволяють користувачам передавати не лише текстову інформацію, але й мультимедійні дані: відео, презентації або посилання на веб-ресурси.

Сучасні рішення для створення бізнес-карток відрізняються високою функціональністю та орієнтовані на різні категорії користувачів: від дизайнерів до бізнес-професіоналів. Вони дозволяють інтегрувати традиційний концепт візитівок із сучасними цифровими інструментами, забезпечуючи зручність, ефективність та можливість персоналізації. Вибір конкретного рішення залежить від індивідуальних потреб, бюджету та рівня технічної підготовки користувача.

2.2. Порівняльний аналіз популярних застосунків для генерації візитівок

Десктопні застосунки для створення електронних бізнес-карток

становлять важливий сегмент ринку графічного програмного забезпечення, орієнтованого на професійних дизайнерів та бізнес-користувачів. Такі програми відрізняються високою функціональністю, розширеними можливостями для роботи з графікою та текстом, а також інтеграцією з сучасними інструментами друку. Основними представниками цього сегмента є Adobe Illustrator, CorelDRAW та Canva, кожен із яких розроблений за допомогою передових інструментів програмної інженерії та графічної обробки.

Adobe Illustrator є векторним графічним редактором, створеним компанією Adobe Systems. Це програмне забезпечення було розроблено на основі мови програмування C++, що забезпечує високу продуктивність та ефективність роботи з великими обсягами графічних даних. Illustrator пропонує потужні інструменти для створення складних дизайнів візитівок, таких як:

Pen Tool для точного малювання кривих і геометричних форм.

Typography Tools, які дозволяють працювати з текстовими елементами, включаючи шрифти, міжрядковий інтервал і вирівнювання.

Color Management System для налаштування кольорових профілів, що гарантує відповідність палітри між цифровими й друкованими версіями.

Illustrator інтегрується з іншими продуктами Adobe, такими як Photoshop, що дозволяє імпортувати растрові зображення та комбінувати їх із векторною графікою для створення унікальних бізнес-карток.

CorelDRAW є ще одним потужним інструментом для створення бізнес-карток. Його розробка базується на поєднанні C++ та бібліотек графічного інтерфейсу, таких як MFC (Microsoft Foundation Classes). Це програмне забезпечення було створене для Windows і підтримує функції роботи з векторною графікою, яка є основою для створення високоякісного дизайну.

Особливості CorelDRAW включають:

Shape Tools для роботи з геометричними об'єктами.

Gradient Fills and Effects для створення плавних переходів кольорів.

Print Layout Features, що дозволяють оптимізувати макет бізнес-картки

для друку.

CorelDRAW часто використовується дизайнерами, оскільки надає інтуїтивно зрозумілий інтерфейс і широкий спектр функціоналу для роботи з графікою та текстом.

На відміну від Adobe Illustrator і CorelDRAW, Canva спочатку розроблялася як онлайн-платформа для створення графічного контенту, проте сьогодні вона також доступна як десктопний застосунок. Її основа - фреймворк React для веб-інтерфейсу та мікросервісна архітектура з використанням Node.js та Python. Canva розроблена для широкої аудиторії, включаючи користувачів без попереднього досвіду в дизайні.

Серед її функціоналу: Бібліотека готових до використання шаблонів бізнес-карток. Drag-and-Drop Interface - простота у створенні дизайнів за допомогою перетягування елементів.

Інтеграція з хмарними сховищами, такими як Google Drive і Dropbox, для збереження проєктів.

Для створення таких потужних програмних продуктів, як Adobe Illustrator та CorelDRAW, використовуються кілька рівнів інженерних підходів:

Основу коду становлять C++ і C, які забезпечують високопродуктивне обчислення та ефективне управління пам'яттю.

Бібліотеки для роботи з графікою - використовуються API, такі як Direct2D для Windows або Quartz для macOS, що дозволяють відтворювати складні графічні об'єкти.

Технології інтеграції: Застосовуються методи інтеграції з хмарними платформами, друкарськими пристроями та веб-сервісами для підвищення функціональності.

Десктопні рішення для створення бізнес-карток представляють складні програмні системи, що забезпечують професійний рівень роботи з графікою та текстом. Вони є результатом використання сучасних технологій програмування, графічних бібліотек та інтеграційних можливостей, які

дозволяють дизайнерам створювати високоякісні та креативні бізнес-картки. Вибір конкретного інструменту залежить від рівня досвіду користувача, специфічних потреб та обсягу роботи.

Онлайн-платформи для створення бізнес-карток є популярним інструментом завдяки їх доступності, простоті використання та можливості швидкого створення професійних дизайнів. Вони орієнтовані на широке коло користувачів - від професійних дизайнерів до малих підприємців, які не мають спеціалізованих знань у галузі графічного дизайну. Важливим аспектом таких платформ є їх веб-орієнтованість, що дозволяє користувачам працювати через браузер без необхідності встановлення додаткового програмного забезпечення.

Платформи, такі як Canva, Vistaprint, Crello та Zazzle, надають інтуїтивно зрозумілий інтерфейс для створення бізнес-карток. Вони побудовані на основі сучасних веб-технологій, включаючи HTML5, CSS3 та JavaScript, що забезпечують інтерактивність та адаптивність інтерфейсу. Використання фреймворків, таких як React або Vue.js, дозволяє реалізувати складні функції редагування в реальному часі.

Серед ключових можливостей онлайн-платформ варто відзначити:

Бібліотеки готових шаблонів, розроблених професійними дизайнерами, які можна легко налаштувати.

Платформи забезпечують прямий зв’язок із друкарськими сервісами, що дозволяє користувачам отримувати фізичні копії своїх бізнес-карток.

Проекти автоматично зберігаються у хмарі, що дає змогу легко повернутися до них пізніше.

Платформи адаптовані для користувачів із різних країн, забезпечуючи підтримку кількох мов.

Більшість онлайн-платформ створені з використанням мікросервісної архітектури, що забезпечує масштабованість та надійність. Для обробки графіки застосовуються бібліотеки, такі як Fabric.js або Konva.js, що дозволяють створювати інтерактивні елементи на полотні. Серверна частина

платформ, як правило, реалізується на основі Node.js, Python (Django) або Ruby on Rails, що забезпечує обробку запитів і взаємодію з базами даних.

Платформи, такі як Vistaprint і Zazzle, поєднують можливість створення дизайнів із друком на замовлення. Це дозволяє користувачам не тільки створювати бізнес-картки, але й замовляти їх доставку до офісу чи дому.

Мобільні застосунки для створення бізнес-карток є важливою частиною сучасного цифрового середовища, забезпечуючи користувачам можливість створювати, редагувати та обмінюватися бізнес-картками безпосередньо зі своїх смартфонів. Вони орієнтовані на користувачів, які цінують мобільність і швидкість, та забезпечують інтеграцію з іншими цифровими платформами.

Мобільні застосунки, такі як CamCard, HiHello, Blinq та Haystack, пропонують користувачам зручні функції, включаючи:

Використання технологій оптичного розпізнавання символів (OCR) для перетворення фізичних візитівок у цифровий формат.

Використання QR-кодів або NFC для обміну картками з іншими користувачами.

Збереження створених бізнес-карток у телефонній книзі або синхронізація з корпоративними сервісами, такими як Google Contacts або Microsoft Outlook.

Додаткові мультимедійні елементи: Деякі застосунки дозволяють додавати до бізнес-картки інтерактивні елементи, такі як посилання на веб-сайти, відеопрезентації або соціальні мережі.

Мобільні застосунки розробляються за допомогою платформ, таких як React Native, Flutter або Swift для iOS та Kotlin для Android. Для обробки зображень і OCR застосовуються бібліотеки, такі як Tesseract. Хмарні сервіси, такі як Firebase, використовуються для зберігання даних, що забезпечує синхронізацію між пристроями.

Деякі сучасні мобільні застосунки, наприклад, HiHello, підтримують створення інтерактивних карток з можливістю додавання відео або анімацій.

Blinq використовує технологію NFC, що дозволяє передавати дані безконтактно, створюючи сучасну альтернативу традиційним паперовим карткам.

Онлайн-платформи та мобільні застосунки для створення бізнес-карток є важливими інструментами для автоматизації та персоналізації процесу обміну контактною інформацією. Вони використовують передові технології для забезпечення функціональності, мобільності та інтеграції, що робить їх незамінними у сучасному цифровому середовищі. Вибір конкретного рішення залежить від потреб користувача, технічних можливостей і переваг у роботі з різними платформами.

Інтегровані системи для створення бізнес-карток є ключовим компонентом сучасних корпоративних екосистем, що дозволяють автоматизувати управління контактною інформацією та полегшують її обмін у бізнес-середовищі. Ці рішення орієнтовані на потреби великих організацій і забезпечують інтеграцію з іншими інструментами, такими як системи електронної пошти, CRM-системи та платформи управління контактами. Серед найбільш популярних рішень виділяються Microsoft Outlook, Google Contacts і LinkedIn, які надають можливості для створення та збереження електронних візитівок у рамках своїх екосистем. Важливим аспектом цих систем є синхронізація контактів між різними пристроями та інтеграція з іншими додатками, що спрощує управління даними та забезпечує їх актуальність.

Технологічно інтегровані системи створюються на основі багатокомпонентних архітектур. Серверна частина, зазвичай, розробляється з використанням платформ, таких як .NET для Microsoft Outlook або Google Cloud Platform для Google Contacts, що забезпечує масштабованість і захист даних. API, зокрема REST або GraphQL, забезпечують інтеграцію з зовнішніми системами, дозволяючи отримувати і оновлювати контактну інформацію в реальному часі. Використання хмарних баз даних, таких як Azure SQL Database або Google Cloud Datastore, гарантує збереження великих обсягів контактів і

доступ до них з будь-якої точки світу.

Інноваційні рішення для створення бізнес-карток зосереджуються на впровадженні новітніх технологій, таких як NFC (Near Field Communication) і доповнена реальність (AR). Вони дозволяють створювати інтерактивні візитівки, які значно розширюють можливості традиційного обміну контактною інформацією. Інтерактивні картки, обладнані NFC-чіпами, можуть миттєво передавати дані на пристрій користувача безконтактно, що особливо зручно в умовах швидкої комунікації. Подібні рішення, наприклад Popl або Tapp y, дозволяють інтегрувати в бізнес-картку не лише текстові дані, а й посилання на соціальні мережі чи відеоматеріали.

Доповнена реальність відкриває нові можливості у дизайні бізнес-карток. Використовуючи платформи ARKit або ARCore, компанії створюють інтерактивний контент, який активується через сканування картки. Це можуть бути 3D-моделі продуктів, анімовані логотипи чи відеопрезентації, що значно підвищують залученість клієнтів. Подібні технології інтегруються у застосунки, як-от HiHello чи Blinq, які також підтримують використання NFC для передачі даних та синхронізацію з CRM-системами.

Розробка інноваційних рішень базується на сучасних фреймворках та технологіях. Використання SDK для роботи з NFC, таких як NFC Tools, та фреймворків для AR, наприклад, Unity 3D або Vuforia, дозволяє створювати інтерактивний і мультимедійний контент. Дані зберігаються та обробляються у хмарних сервісах, таких як AWS Lambda або Google Firebase, що забезпечує швидкість і зручність доступу до інформації.

Таким чином, інтегровані системи та інноваційні рішення для створення бізнес-карток значно підвищують ефективність бізнес-комунікацій, розширюючи функціональність традиційних підходів. Вони впроваджують зручність, інтерактивність і персоналізацію, що робить їх важливими інструментами в умовах цифрової трансформації бізнесу.

2.3. Переваги та недоліки розробки існуючих рішень

Сучасні рішення для створення візитівок або бізнес-карток, включаючи десктопні програми, онлайн-платформи, мобільні додатки, інтегровані системи та інноваційні рішення, пропонують численні переваги для користувачів і бізнесів. Водночас процес їх розробки є технічно складним і вимагає значних ресурсів.

Однією з ключових переваг цих рішень є автоматизація створення візитівок, що дозволяє зменшити час і зусилля користувачів на виконання рутинних завдань. Наприклад, онлайн-платформи забезпечують доступність з будь-якого пристрою, дозволяючи користувачам створювати професійні дизайни без спеціалізованих навичок. Мобільні додатки та інтегровані системи забезпечують швидкий обмін контактною інформацією через QR-коди, NFC або хмарні сервіси, підвищуючи зручність і ефективність комунікації. Інноваційні рішення, такі як доповнена реальність і інтерактивні картки, відкривають нові можливості для персоналізації, підвищуючи залученість клієнтів.

Десктопні програми, такі як Adobe Illustrator та CorelDRAW, надають широкий спектр інструментів для створення високоякісних графічних дизайнів, що робить їх вибором професійних дизайнерів. Інтегровані системи, наприклад, Microsoft Outlook або Google Contacts, дозволяють легко керувати контактами, забезпечуючи їх синхронізацію та інтеграцію з іншими сервісами, такими як CRM.

Використання таких рішень також має свої недоліки. Десктопні програми можуть бути складними для освоєння новачками, а їхня вартість може бути недоступною для малого бізнесу або індивідуальних користувачів. Онлайн-платформи і мобільні додатки часто обмежені у функціональності порівняно з професійним програмним забезпеченням і можуть потребувати стабільного підключення до Інтернету. Інтегровані системи іноді мають проблеми з сумісністю між різними платформами або вимагають значних зусиль для налаштування. Інноваційні рішення, такі як NFC-картки, можуть не працювати

з усіма пристроями, особливо зі старими моделями смартфонів, а використання AR може вимагати значних обчислювальних ресурсів.

Процес розробки програмного забезпечення для таких рішень є технічно складним і вимагає участі висококваліфікованих спеціалістів. Однією з основних складностей є забезпечення кросплатформенності, особливо для мобільних додатків та інтегрованих систем. Використання таких технологій, як React Native чи Flutter, може значно скоротити час розробки, але водночас вимагає компромісів у продуктивності.

Розробка інноваційних рішень, таких як NFC-картки або AR-додатки, передбачає інтеграцію спеціалізованих апаратних і програмних компонентів. Наприклад, розробникам необхідно враховувати сумісність NFC-чіпів з різними смартфонами, налаштовувати AR-контент для коректного відображення на пристроях із різними характеристиками дисплеїв і камери. Це ускладнюється необхідністю тестування на численних пристроях, щоб уникнути проблем із сумісністю.

Ще однією важливою складністю є забезпечення безпеки даних, особливо для інтегрованих систем та хмарних сервісів. Дані контактів часто мають конфіденційний характер, тому розробники повинні впроваджувати механізми шифрування та автентифікації, а також дотримуватися міжнародних стандартів захисту даних, таких як GDPR.

Крім того, розробка програмного забезпечення для онлайн-платформ потребує значних зусиль для забезпечення масштабованості та низької затримки навіть за умови високих навантажень. Це вимагає використання хмарних технологій, мікросервісної архітектури та оптимізації баз даних.

Попри складності у розробці, сучасні рішення для створення візитівок та бізнес-карток пропонують значні переваги для користувачів, такі як зручність, автоматизація та можливості персоналізації. Однак успішна розробка таких продуктів вимагає значних технічних ресурсів, високої кваліфікації розробників і врахування численних факторів, включаючи сумісність, безпеку

та масштабованість. Баланс між інноваціями та простотою використання є ключовим фактором успіху подібних рішень.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Опис проектних рішень, інструментів та підходів до розробки застосунку

Зважаючи на проведений аналіз наукових джерел, зокрема [15] та існуючих рішень для генерації візитівок для розробки програмного забезпечення андроїд застосунку було використано наступні технології:

- бібліотеки AndroidX та плагінів Kotlin для управління навігацією в додатку та взаємодія з UI оптимізовані завдяки використанню.
- мова програмування Kotlin та фреймворку Koin для управління зв’язками між різними частинами додатку;
- інструмент R8 для зменшення розміру APK;
- Navigation Component спрощення навігації між екранами;
- View Binding для інтеграції UI компонентів з кодом;
- Data Binding для зв’язування UI компонентів у макеті (layouts) безпосередньо з даними (наприклад, змінними в коді);
- Clean Architecture для розділення різних аспекти програми на окремі шари;
- середовище розробки Android Studio.

Набір для розробки програмного забезпечення AndroidX.

AndroidX – це бібліотека компонентів, що є частиною Android Jetpack, яка надає сучасніші, ефективніші та кращі для підтримки і розширення версії оригінальних Android Support Library [2]. AndroidX є наступником Support Library і була розроблена для спрощення процесу розробки додатків на платформі Android, забезпечуючи нові можливості та кращу підтримку зворотної сумісності. (Рисунок 3.1).

Рисунок 3.1. Логотип AndroidX.

Основна мета AndroidX — забезпечити стандартизацію, покращення якості коду та зручність використання бібліотек, що використовуються для розробки додатків на Android. AndroidX включає широкий спектр бібліотек, які охоплюють різні аспекти розробки, такі як підтримка різних версій Android, створення інтерфейсу користувача, обробка даних, робота з фрагментами, навігація, анімації, архітектура додатків та інші функціональні можливості.

Ключовими особливостями AndroidX є її модульність та незалежність від версії платформи Android. На відміну від старих бібліотек підтримки, AndroidX забезпечує швидке оновлення та випуск нових версій бібліотек, які можуть бути використані на різних версіях Android без потреби в оновленні операційної системи. Це дає змогу розробникам швидше отримувати доступ до нових функцій та виправлень без необхідності чекати на оновлення операційної системи від виробника пристрою.

Крім того, AndroidX використовує нову систему іменування пакетів, яка чітко відокремлює її від старих бібліотек підтримки. Це дозволяє уникнути конфліктів і забезпечує чіткішу організацію залежностей в проектах. [2] .

Мова програмування Kotlin.

Kotlin – це сучасна статично типізована мова програмування, розроблена

компанією JetBrains. Вона призначена для розробки програмного забезпечення на платформі Java Virtual Machine (JVM), а також для компіляції в JavaScript або нативний код. Kotlin повністю сумісний з Java, що дозволяє розробникам використовувати існуючі Java-бібліотеки та фреймворки в проектах на Kotlin. (Рисунок 3.2).

Рисунок 3.2. Логотип Kotlin.

Основною перевагою Kotlin є його повна сумісність з Java, що дозволяє розробникам використовувати наявні Java-бібліотеки та фреймворки, спрощуючи інтеграцію з існуючими проектами. Kotlin забезпечує високий рівень лаконічності та читабельності коду завдяки використанню сучасних синтаксичних конструкцій та функціональних підходів, що значно зменшує кількість коду, необхідного для досягнення певної функціональності.

Безпека типів є ще однією важливою характеристикою Kotlin, яка запобігає поширеним помилкам, таким як NullPointerException, завдяки використанню Nullable та Non-Nullable типів. Це робить Kotlin надійнішою мовою порівняно з традиційними підходами до управління пам’яттю і обробки виключень. Мова також підтримує функціональне програмування, надаючи можливість використовувати лямбда-вирази, функції вищого порядку та імутабельні колекції, що сприяє більш чистому та модульному коду.

Крім того, Kotlin підтримує мультиплатформену розробку, що дозволяє створювати спільний код для різних платформ, включаючи Android, iOS, веб та серверні додатки, завдяки функціональності Kotlin Multiplatform. Це робить

Kotlin універсальним інструментом для сучасної розробки програмного забезпечення, де мобільність і кросплатформеність є ключовими вимогами [2].

Враховуючи всі ці переваги, Kotlin став офіційно підтримуваною мовою програмування для розробки Android застосунків і набув широкої популярності серед розробників, які прагнуть створювати сучасні, надійні та ефективні додатки. [2, 6, 7] .

Фреймворк Koin.

Koin - це легковагий фреймворк для інверсії керування (Inversion of Control, IoC) та впровадження залежностей (Dependency Injection, DI) у додатках на Kotlin. Розроблений спеціально для мови Kotlin, Koin надає інтуїтивно зрозумілий синтаксис і не вимагає додаткової конфігурації, що робить його легким у використанні та швидким у налаштуванні.

Рисунок 3.3. Логотип Koin.

Однією з основних переваг Koin є його простота та мінімалізм. Фреймворк не використовує рефлексії чи складних метапрограмувальних підходів, які характерні для багатьох інших DI-фреймворків. Замість цього Koin застосовує функції Kotlin для визначення залежностей і їх впровадження, що робить процес розробки більш природним і легким для розуміння. Завдяки цьому Koin добре підходить для проектів будь-якого розміру, від невеликих додатків до великих корпоративних систем.

Koin підтримує декілька моделей впровадження залежностей, включаючи Singleton, Factory, та Scoping, що дозволяє розробникам гнучко налаштовувати

життєвий цикл залежностей залежно від потреб проекту. Крім того, Koin забезпечує просту інтеграцію з платформами Android та Ktor, що робить його популярним вибором серед розробників Android-додатків і серверних додатків на Kotlin.

Фреймворк також надає інструменти для тестування, що дозволяють легко замінювати залежності на мок-об'єкти, сприяючи поліпшенню покриття тестами та підвищенню надійності коду. Завдяки цьому Koin сприяє більш ефективному розробленню тестів і підтримці проектів у чистому стані.

Koin стає все більш популярним серед розробників Kotlin завдяки своїй простоті, легкості інтеграції, ефективному використанню ресурсів та здатності швидко впроваджувати залежності без необхідності вивчення складних конфігураційних файлів або XML. Це робить Koin ідеальним вибором для розробників, які шукають простий, але потужний інструмент для управління залежностями у своїх проектах [6, 7].

Інструмент R8.

R8 - це інструмент для зменшення розміру APK який виконує функції зменшення розміру байт-коду, видалення невикористаного коду та ресурсів, а також обфускації (заплутування) коду для підвищення його безпеки (процес відомий як «minification» та «resource shrinking»). Він допомагає зробити додаток легшим та швидшим для користувачів. R8 був представлений як заміна для попереднього інструменту ProGuard, що також використовувався для цих завдань, але забезпечує більш високу ефективність та спрощений процес налаштування [7].

Рисунок 3.4. Інструмент R8.

Однією з основних функцій R8 є зменшення розміру додатків шляхом видалення непотрібних класів, методів і полів, які не використовуються під час виконання програми. Це досягається шляхом аналізу залежностей у коді і визначення частин, які можуть бути видалені без впливу на функціональність додатка. Ця оптимізація не лише зменшує розмір APK-файлу, але й покращує продуктивність додатка, оскільки зменшується кількість коду, який потрібно завантажувати та інтерпретувати під час виконання.

R8 також забезпечує обфускацію коду, змінюючи імена класів, методів і полів на менш зрозумілі для зловмисників. Це робить зворотню інженерію додатка складнішою, тим самим підвищуючи його захист від несанкціонованого доступу та копіювання. Інструмент R8 виконує обфускацію більш ефективно, ніж ProGuard, завдяки інтеграції з компілятором D8, що забезпечує більш глибокий аналіз коду і кращу оптимізацію.

Ще однією важливою перевагою R8 є те, що він поєднує функціональність компіляції байт-коду (D8) і обфускації/мінімізації (ProGuard) в одному кроці. Це зменшує час збірки та полегшує процес налаштування, оскільки розробникам не потрібно налаштовувати кілька інструментів для досягнення оптимального результату.

Крім того, R8 підтримує спеціальні правила конфігурації, що дозволяють розробникам точно визначати, які класи, методи або ресурси повинні

залишитися в незмінному вигляді (наприклад, для використання у зовнішніх бібліотеках або фреймворках, що потребують рефлексії). Це забезпечує більшу гнучкість у налаштуванні процесу оптимізації і допомагає уникнути можливих проблем, пов’язаних з видаленням або обфускацією важливих елементів.

Загалом, R8 є потужним інструментом для оптимізації додатків на Android, що сприяє підвищенню продуктивності, зменшенню розміру додатків і покращенню безпеки коду. Його інтеграція з сучасними інструментами розробки Android робить його невід'ємною частиною процесу створення ефективних і безпечних мобільних додатків.

Navigation Component.

Navigation Component - це бібліотека Android Jetpack, призначена для полегшення управління навігацією у додатках. Вона надає комплексний набір інструментів для створення, налаштування та управління навігацією між екранами (фрагментами та активностями) в додатку, спрощуючи реалізацію навігаційних шаблонів і забезпечуючи узгодженість користувацького досвіду [11].

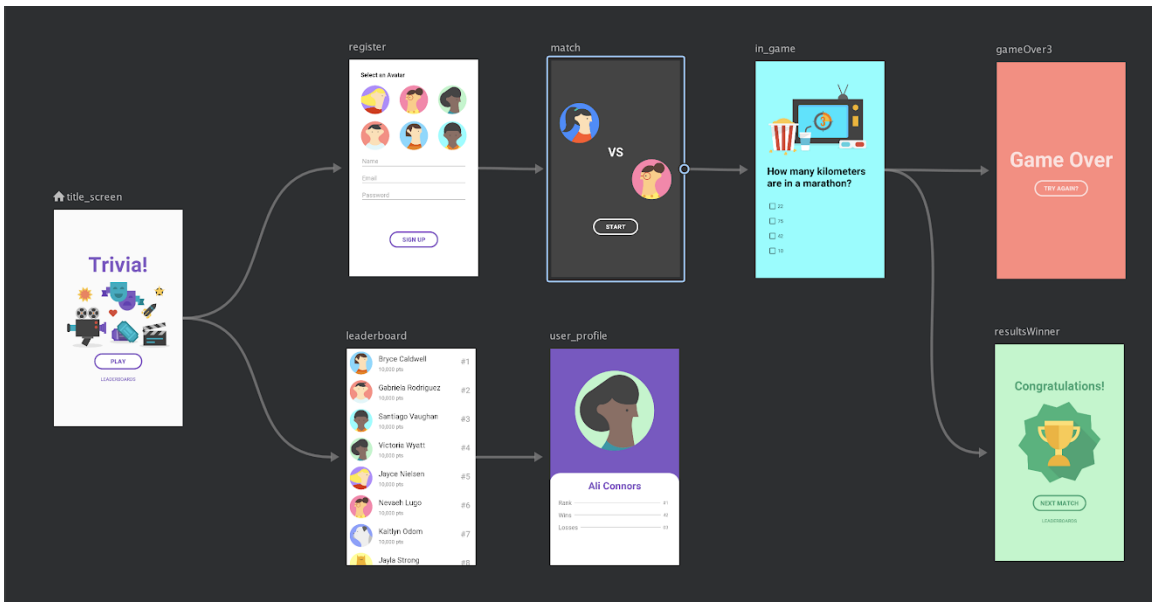

Рисунок 3.5. Візуалізація Navigation Component для Android застосунку.

Однією з головних переваг Navigation Component є можливість

декларативного визначення навігаційної логіки за допомогою графів навігації (navigation graphs). Ці графи дозволяють розробникам візуально відобразити всі можливі маршрути переходу між екранами додатку та налаштувати відповідні дії, а також вказати параметри, які можуть бути передані під час переходу. Цей підхід значно спрощує управління навігацією і забезпечує краще розуміння структури додатка, знижуючи ймовірність помилок, пов'язаних з некоректною навігацією.

Navigation Component також підтримує роботу з різними типами навігаційних елементів, такими як Bottom Navigation, Navigation Drawer та Toolbar. Це дозволяє легко створювати інтерфейси з багаторівневою навігацією, а також дотримуватись рекомендацій щодо дизайну матеріального дизайну (Material Design). Крім того, бібліотека підтримує навігацію через фрагменти, що є основним підходом до побудови інтерфейсу в сучасних Android-додатках, що забезпечує більш плавний і безперервний користувацький досвід.

Іншою важливою особливістю Navigation Component є підтримка Safe Args - інструменту, який генерує безпечні класи для передачі даних між екранами, усуваючи потребу в ручному створенні пакетів (Bundles) і зменшуючи ризик помилок типів даних. Safe Args автоматично генерує типобезпечні методи для передачі аргументів, що підвищує надійність коду і полегшує підтримку та розширення додатка.

Бібліотека також інтегрується з іншими компонентами Android Jetpack, такими як ViewModel та LiveData, що забезпечує просту та ефективну обробку навігаційних подій та збереження стану користувацького інтерфейсу під час навігації. Це дозволяє створювати додатки, що реагують на зміни стану та події, забезпечуючи більш інтерактивний і динамічний досвід для користувачів.

Navigation Component включає також підтримку глибоких посилань (deep linking), що дозволяє користувачам напряму переходити до конкретних екранів або функцій додатка з інших додатків або з веб-сайтів. Це розширює можливості інтеграції додатка з іншими сервісами та підвищує його зручність

для користувачів.

Таким чином, Navigation Component є потужним інструментом для управління навігацією в Android-додатках, який спрощує процес розробки, знижує ризик помилок і забезпечує узгоджений і плавний користувацький досвід. Завдяки широкому набору функцій та інтеграції з іншими компонентами Android Jetpack, Navigation Component є важливим інструментом для створення сучасних, надійних та зручних мобільних додатків.

View Binding

View Binding є частиною Android Jetpack і дозволяє легко інтегрувати UI-компоненти з кодом, забезпечуючи безпечне використання елементів інтерфейсу без необхідності ручного звернення за допомогою ідентифікаторів.

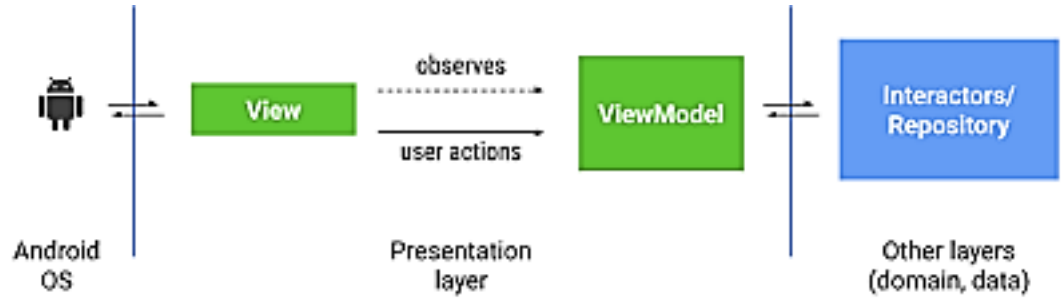

Рисунок 3.6. Процес отримання доступу до переглядів на Android

View Binding — це офіційний інструмент Android, що дозволяє безпосередньо взаємодіяти з елементами інтерфейсу користувача (View) у додатку на Android за допомогою згенерованих класів прив'язки (binding classes). Введена як альтернатива інструменту `findViewById()`, View Binding значно спрощує роботу з інтерфейсом користувача, забезпечуючи безпеку типів та зменшуючи кількість шаблонного коду, необхідного для доступу до елементів інтерфейсу [3].

Однією з основних переваг View Binding є автоматичне генерування класів прив'язки на основі XML-макетів інтерфейсу. Після увімкнення View Binding у проекті Android Studio автоматично генерує клас прив'язки для

кожного XML-файлу макета, який включає в себе властивості для кожного View з унікальним ідентифікатором. Це означає, що розробникам більше не потрібно викликати метод `findViewById()` або здійснювати небезпечні касти типів для доступу до елементів інтерфейсу. Замість цього вони можуть використовувати згенеровані класи прив’язки з типобезпечними властивостями, що робить код більш зрозумілим і надійним.

View Binding також покращує продуктивність розробки та зменшує ймовірність виникнення помилок під час компіляції або виконання, пов'язаних з доступом до невірних View або неправильним кастом типів. Інструмент генерує код на основі компіляції, що забезпечує перевірку наявності всіх елементів інтерфейсу на етапі компіляції, усуваючи помилки, які можуть виникнути під час виконання програми через спроби доступу до неіснуючих елементів.

Крім того, View Binding підтримує роботу з фрагментами, діалогами, RecyclerView та іншими компонентами Android, що забезпечує більш універсальний підхід до взаємодії з інтерфейсом користувача в різних сценаріях. Наприклад, у фрагментах View Binding дозволяє легко і безпечно зв’язувати макет фрагмента з його логікою, що спрощує управління життєвим циклом фрагмента і запобігає витокам пам'яті.

View Binding також добре інтегрується з іншими компонентами Android Jetpack, такими як Data Binding, забезпечуючи додаткову функціональність, наприклад, прив'язку даних до інтерфейсу користувача за допомогою спостерігачів (observers) та двосторонню прив'язку даних. Це робить View Binding гнучким і потужним інструментом, що дозволяє розробникам створювати більш динамічні та інтерактивні інтерфейси користувача.

На відміну від Data Binding, View Binding не додає жодних додаткових елементів до XML-файлів макетів і не потребує спеціальної обробки анотацій, що робить його легким і менш обтяжливим для розробників, які не потребують складних функцій прив'язки даних. Це робить View Binding ідеальним вибором

для тих, хто шукає простий і ефективний спосіб взаємодії з інтерфейсом користувача, зберігаючи при цьому безпеку типів і зменшуючи кількість шаблонного коду.

Таким чином, View Binding є важливим інструментом для розробників Android, що значно спрощує процес створення та підтримки додатків, підвищуючи безпеку коду, продуктивність розробки та якість кінцевого продукту.

Data Binding

Data Binding — це концепція, широко застосовувана у програмуванні для забезпечення зв'язку між джерелами даних та їх візуалізацією в інтерфейсі користувача. У сучасних програмних платформах, зокрема у таких, як Angular, React або WPF (Windows Presentation Foundation), цей підхід дозволяє розробникам автоматично синхронізувати зміни між моделлю даних і представленням, знижуючи необхідність у ручному управлінні оновленнями інтерфейсу [5].

Основною ідеєю Data Binding є абстрагування логіки, пов'язаної з оновленням елементів інтерфейсу в залежності від змін у моделі даних. Завдяки цьому забезпечується динамічне оновлення UI у реальному часі, що особливо важливо для інтерактивних застосунків. Умовно цей механізм можна поділити на одностороннє та двостороннє зв’язування. У першому випадку дані передаються від моделі до інтерфейсу, забезпечуючи візуалізацію, тоді як у другому — зміни у візуальних компонентах також синхронізуються з моделлю. Цей підхід зменшує обсяг коду та знижує ризик виникнення помилок, спричинених розходженням між станом даних і виглядом UI.

Імплементація Data Binding у різних платформах має певні особливості, проте всі вони базуються на фундаментальних принципах спостереження за змінами у даних. У JavaScript-фреймворках, таких як Angular, реалізація Data Binding ґрунтується на використанні двосторонніх зв’язків і реактивних

підходів, наприклад, через шаблонні вирази. У середовищі WPF Data Binding здійснюється через механізм залежних властивостей та спостереження, що дозволяє програмістам налаштовувати зв'язки між елементами даних і UI через XAML або програмний код.

Крім зручності, Data Binding також сприяє створенню гнучких і масштабованих архітектур, таких як MVVM (Model-View-ViewModel). У таких архітектурах модель даних і UI розділяються, що полегшує тестування, повторне використання компонентів та підтримку програмного забезпечення. Це особливо важливо для великих і складних проєктів, де відсутність прямої залежності між компонентами підвищує модульність системи [5].

Таким чином, Data Binding є ключовим інструментом сучасного розробника, що сприяє оптимізації процесу створення інтерфейсу користувача. Його впровадження дозволяє автоматизувати синхронізацію даних між моделью та візуальними компонентами, покращуючи продуктивність розробки, зручність підтримки і функціональність програмного забезпечення.

Clean Architecture

Clean Architecture - це підхід до проєктування програмного забезпечення, який фокусується на створенні систем із чіткою модульністю, високим рівнем гнучкості та незалежністю від конкретних технологій. Цей архітектурний стиль був запропонований Робертом С. Мартіном (відомим як Uncle Bob) і набув широкого застосування у розробці великих і складних систем. Основна мета Clean Architecture - забезпечити довговічність і простоту змін у програмному забезпеченні, що досягається через чітке розділення відповідальностей між компонентами.

Ключовим принципом Clean Architecture є відділення бізнес-логіки та правил домену від деталей реалізації, таких як інтерфейси користувача, бази даних чи сторонні бібліотеки. Це досягається через концентричну структуру, де ядро системи містить найбільш незалежні від зовнішнього середовища

компоненти, а зовнішні шари адаптуються до змінних технологій чи платформ. Найглибший рівень архітектури зазвичай складається з сутностей, які визначають бізнес-об’єкти та їхні правила. Ці сутності не залежать від інших компонентів системи, забезпечуючи високу стійкість до змін [12].

На наступному рівні знаходяться інтерфейси використання (use cases), які реалізують бізнес-логіку та координують взаємодію між користувачами та системою. Цей шар відповідає за забезпечення виконання правил домену, незалежно від деталей реалізації, що дозволяє легко змінювати або розширювати функціональність програми. Далі йдуть рівні інтерфейсів і інфраструктури, які включають адаптери для взаємодії з базами даних, API або інтерфейсами користувача. Ці шари можуть змінюватися чи замінюватися без впливу на внутрішню бізнес-логіку, що є суттєвою перевагою Clean Architecture.

Важливою особливістю цієї архітектури є залежність від абстракцій, а не від конкретних реалізацій. Це дозволяє зменшити ризики, пов’язані зі зміною технологій або платформ, і забезпечити модульність системи. Розробники можуть легко замінювати окремі компоненти або переносити систему на інші середовища без значних витрат на рефакторинг. Наприклад, зміна бази даних або фреймворку для роботи з інтерфейсом користувача вимагає лише оновлення відповідних адаптерів, тоді як ядро системи залишається незмінним [12].

Таким чином, Clean Architecture спрямована на створення програмного забезпечення, яке легко адаптується до змін, є простим у тестуванні та довговічним у використанні. Її принципи базуються на поділі відповідальностей, залежності від абстракцій і модульності, що робить цей підхід універсальним для багатьох типів проєктів, від невеликих застосунків до великих корпоративних систем.

Середовище розробки Android Studio.

Android Studio – це офіційне інтегроване середовище розробки (IDE) для створення додатків на платформі Android, розроблене і підтримуване компанією Google у співпраці з JetBrains [8]. Засноване на популярній IDE IntelliJ IDEA від JetBrains, Android Studio надає розробникам потужний набір інструментів для створення, тестування та оптимізації додатків для пристроїв Android, таких як смартфони, планшети, телевізори, годинники та автомобілі. (Рисунок 3.7).

Рисунок 3.7. Логотип Android Studio

Android Studio надає безліч інструментів і функцій, що значно полегшують процес розробки. Однією з ключових функцій є редактор коду, який підтримує підсвічування синтаксису, автозаповнення, рефакторинг, а також інтелектуальні підказки, що базуються на контексті коду. Ці можливості підвищують ефективність розробки, допомагають уникати синтаксичних помилок і знижують ризик помилок, пов'язаних із використанням API або бібліотек.

Дизайнер інтерфейсу користувача (UI Designer) в Android Studio дозволяє розробникам створювати та редагувати макети інтерфейсу користувача за допомогою візуального редактора, який забезпечує можливість перетягування (drag-and-drop) компонентів. Режим попереднього перегляду дозволяє бачити, як додаток виглядатиме на різних пристроях і версіях Android, що полегшує створення адаптивних і сумісних інтерфейсів. Крім того, дизайнер підтримує роботу з компонентами Material Design, що забезпечує дотримання сучасних рекомендацій Google щодо розробки користувацького інтерфейсу.

Android Studio також інтегрує потужний системний збірник Gradle, який автоматизує процес складання, тестування, розгортання та публікації додатків. Gradle підтримує створення багатомодульних проектів, налаштування залежностей, управління версіями і оптимізацію процесу збірки, що дозволяє розробникам легко керувати проектами будь-якої складності та масштабу.

Іншим важливим компонентом Android Studio є емулятор Android — вбудований інструмент для тестування додатків на різних конфігураціях пристроїв без потреби в реальних пристроях. Емулятор дозволяє швидко налаштовувати та змінювати різні параметри, такі як розмір екрану, версія Android, об'єм пам'яті, а також тестувати додатки в умовах різних мережевих станів, що допомагає розробникам виявляти помилки і забезпечувати сумісність додатків з різними версіями Android та типами пристроїв.

Android Studio підтримує інструменти для налагодження та профілювання, що включають Android Debug Bridge (ADB), Logcat, Android Profiler та інші засоби для діагностики та оптимізації додатків. Ці інструменти дозволяють розробникам відстежувати використання пам'яті, ЦП, батареї та мережевих ресурсів, знаходити вузькі місця в продуктивності та оптимізувати додатки для кращої роботи. Android Profiler, зокрема, надає детальну інформацію про роботу додатка в реальному часі, що дозволяє ідентифікувати та усувати проблеми продуктивності.

Інтеграція з іншими інструментами Android Jetpack і Firebase спрощує додавання функціональності, такої як навігація, обробка даних, управління життєвим циклом, аналітика, аутентифікація, хмарне зберігання даних та багато інших сервісів. Це забезпечує комплексне середовище розробки, яке охоплює всі аспекти створення сучасних мобільних додатків.

Android Studio також надає підтримку для спільної розробки, включаючи інтеграцію з системами контролю версій, такими як Git, SVN та Mercurial, що дозволяє командам розробників ефективно співпрацювати і керувати змінами в кодовій базі.

Таким чином, Android Studio є потужним і універсальним середовищем розробки, що забезпечує всі необхідні інструменти для створення, налагодження, тестування та розгортання високоякісних Android-додатків. Завдяки своїм можливостям автоматизації, інтеграції та підтримці сучасних стандартів розробки, Android Studio є невід'ємним інструментом для розробників Android, які прагнуть створювати надійні, продуктивні та зручні додатки для користувачів [8, 9].

3.2. Опис методології створення програмного забезпечення та побудова діаграми послідовностей застосунку

Водоспадна модель (Waterfall) є однією з традиційних методологій управління проектами, яка часто використовується у розробці програмного забезпечення. Ця модель базується на послідовному виконанні етапів, де кожен наступний починається лише після завершення попереднього. Вона вирізняється структурованістю, дозволяючи чітко визначити вимоги, спланувати та реалізувати проект. Використання підходу Waterfall для створення Android-додатка, призначеного для агрегації вакансій, може мати як переваги, так і недоліки, які слід оцінювати з огляду на особливості проекту [14].

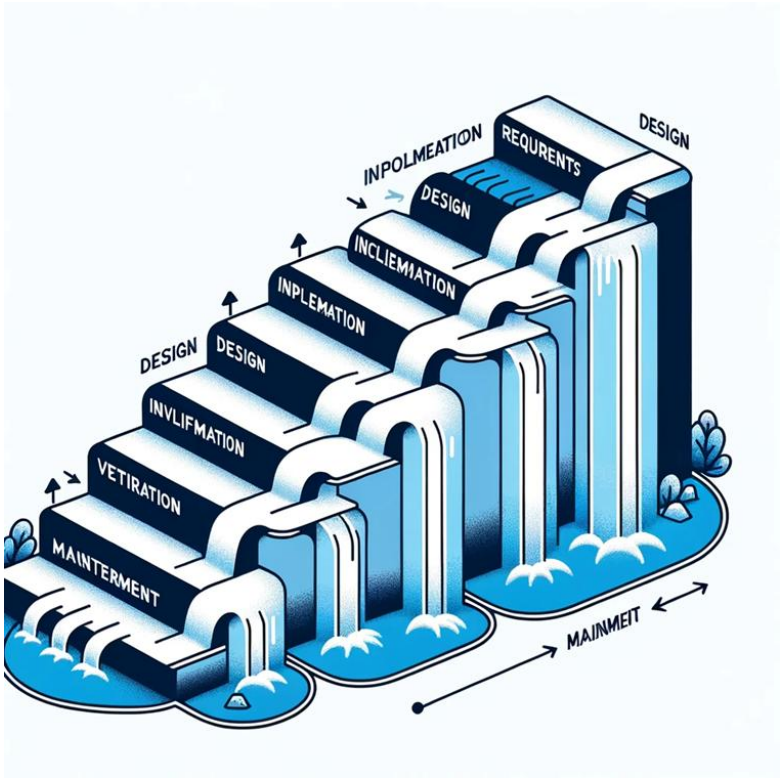

Рисунок. 3.8. Модель розробки застосунку

Розробка Android-застосунку для генерації візитівок із використанням методології «водоспаду» передбачає поетапне виконання всіх стадій створення програмного забезпечення, починаючи від аналізу вимог до впровадження і підтримки. Такий підхід забезпечує послідовність процесу та дозволяє ретельно контролювати реалізацію кожного етапу.

На першій стадії, етапі аналізу вимог, визначаються функціональні можливості та технічні обмеження додатку. Основними функціями застосунку є введення даних бізнес-картки, перевірка їх валідності, створення PDF-файлів, збереження цих файлів у файловій системі та можливість перегляду результатів. У коді ця стадія відображена через визначення моделі BusinessCardModel, яка містить ключові дані візитівки, такі як ім’я, прізвище, номер телефону та посада. Також на цьому етапі було враховано обробку дозволів для доступу до зовнішнього сховища, реалізовану в класі CreateFragment.

На етапі проєктування була обрана архітектурна модель MVVM (Model-View-ViewModel), яка забезпечує чіткий розподіл функціональності між інтерфейсом користувача (View), бізнес-логікою (ViewModel) та обробкою даних (Model). Навігація між екранами реалізована за допомогою компонента Navigation Component, що гарантує структуровану взаємодію між CreateFragment, PreviewFragment та ResultFragment. У коді це проєктне рішення реалізоване через MainActivity, яка використовує NavController для забезпечення переходів між фрагментами, та через MainViewModel, що відповідає за обробку даних і взаємодію з PDFManager.

На стадії реалізації здійснюється безпосереднє написання коду, створення графічного інтерфейсу та функціональності. У CreateFragment розроблено форми для введення даних із динамічною перевіркою їх валідності за допомогою методу checkValidation. У той час MainViewModel обробляє введені користувачем дані, передає їх до PDFManager, де виконується генерація PDF-документів через використання класу PdfDocument. Сформований документ зберігається у файловій системі пристрою, забезпечуючи можливість доступу до нього в подальшому.

На етапі тестування додаток проходить перевірку на коректність роботи основних функцій. Це включає модульне тестування методів, інтеграційне тестування взаємодії між компонентами, такими як MainViewModel і PDFManager, та системне тестування навігації між екранами. Для забезпечення зручного моніторингу змін у даних використовується LiveData, що також дозволяє проводити тестування в реальному часі.

Завершальним етапом є впровадження та підтримка. Додаток публікується в Google Play Store або інших платформах, забезпечуючи доступність для кінцевих користувачів. Після впровадження здійснюється моніторинг, виправлення помилок та додавання нових функцій на основі зворотного зв’язку. У коді передбачено обробку можливих змін у політиках доступу Android, наприклад, через механізм перевірки дозволів у

CreateFragment.

Застосування методології «водоспаду» у розробці цього Android-застосунку дозволило чітко структурувати процес створення програмного забезпечення, забезпечити відповідність функціональності до вимог користувачів і гарантувати якість кінцевого продукту. Це підкреслює важливість послідовного підходу до проєктування та реалізації мобільних застосунків у сучасному цифровому середовищі.

Основна перевага використання методології Waterfall для розробки Android-застосунку для генерації візитівок полягає в її структурованості і чіткості. Всі етапи процесу розробки чітко визначені і плануються на початку, що дозволяє забезпечити передбачуваність термінів і бюджету проекту. Такий підхід також забезпечує детальну документацію на кожному етапі, що може бути корисним для подальшої підтримки та оновлення застосунку.

Однак, основним недоліком Waterfall є її обмежена гнучкість. Оскільки модель передбачає послідовне проходження етапів, будь-які зміни в вимогах або виявлення помилок на пізніх етапах можуть вимагати значних витрат часу і ресурсів для їх виправлення. Це може бути особливо проблематичним для Android-застосунку, де ринкові умови можуть швидко змінюватися, а користувачі очікують швидких оновлень і нових функцій. Крім того, модель Waterfall не передбачає активного залучення користувачів у процес розробки, що може призвести до створення продукту, який не повністю відповідає їх потребам [14].

Діаграма послідовності розробленого андроїд застосунку для генерації візитівок (див. рисунок 3.9) описує основний сценарій взаємодії користувача з системою для створення бізнес-картки, включаючи ключові компоненти: User, CreateFragment, MainViewModel та PDFManager. Ця діаграма є відображенням логіки виконання основного функціоналу додатку і демонструє послідовність викликів між компонентами.

Опис сценарію.

У початковій точці сценарію користувач вводить дані бізнес-картки в інтерфейсі, представленому CreateFragment. Після цього фрагмент передає ці дані у MainViewModel для обробки. ViewModel викликає PDFManager, щоб створити PDF-документ, використовуючи надані дані. Після завершення створення документа менеджер повідомляє про успіх ViewModel, яка, у свою чергу, інформує фрагмент. Нарешті, CreateFragment надає користувачеві повідомлення про успішне завершення операції.

Взаємодія між CreateFragment і MainViewModel:

```
binding.createButton.setOnClickListener {
    mainViewModel.setNewCardModel(
        BusinessCardModel(
            binding.textFieldName.editText!!.text.toString(),
            binding.textFieldLastName.editText!!.text.toString(),
            binding.textFieldPhone.editText!!.text.toString(),
            binding.textFieldWorkPosition.editText!!.text.toString()
        )
    )
}
```

Цей фрагмент коду демонструє передачу даних від CreateFragment до MainViewModel. Об'єкт BusinessCardModel, що містить дані бізнес-картки, створюється та передається через метод setNewCardModel.

Виклик створення PDF-документа:

```
fun createPDFDocument(name: String, model: BusinessCardModel) {
    viewModelScope.launch {
        pdfManager.createPDF(name, model) {
            mutablePDFInfo.postValue(it)
        }
```

```
    }
}
```

У цьому коді MainViewModel викликає PDFManager для створення PDF-документа. Метод createPDFDocument забезпечує асинхронне виконання за допомогою корутин, а результат операції передається через LiveData.

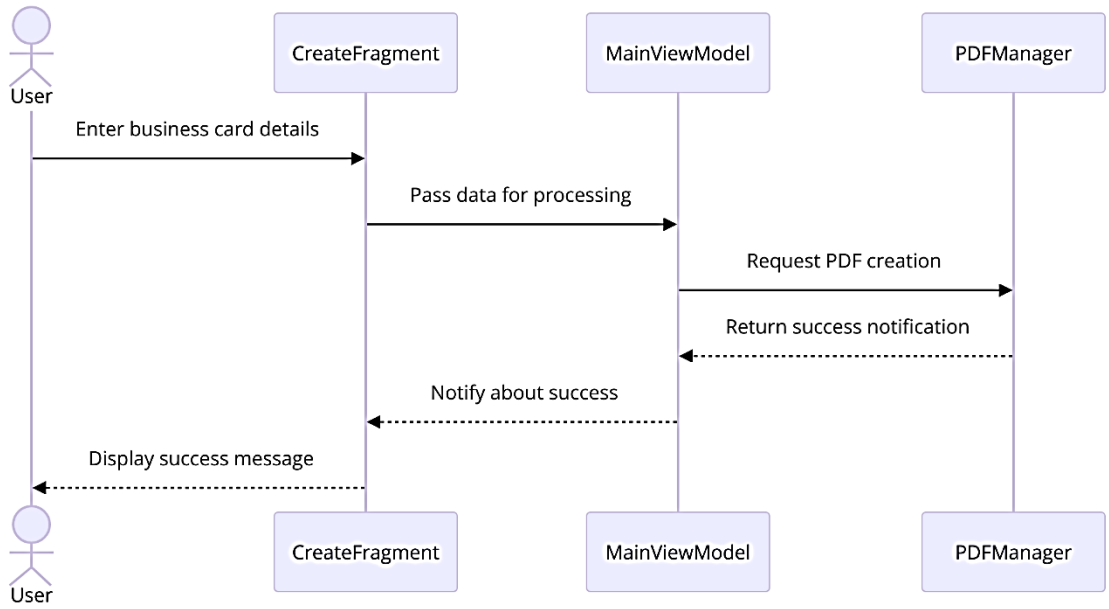

Рисунок 3. 9. Діаграма послідовностей ПЗ розробленого застосунку для генерації візитівок

Представлена діаграма послідовності є візуальним відображенням функціональності та взаємодії компонентів додатку. Вона демонструє розподіл обов’язків між частинами архітектури та спрощує розуміння процесу обробки даних. Використання модульного підходу, представленого у діаграмі, забезпечує масштабованість і підтримку системи.

Діаграма дозволяє ідентифікувати ключові точки взаємодії та слугує основою для подальшого вдосконалення та оптимізації процесу.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Розробка алгоритму роботи застосунку для генерації візитівок

Опишемо алгоритм роботи розробленого нами застосунку. Ось основні його кроки:

1. Ініціалізація додатку

Додаток запускається з класу App, який налаштовує середовище залежностей за допомогою Koin.

```
startKoin {
   androidLogger(level = Level.INFO)
   androidContext(this@App)
   modules(listOf(mainDI))
}
```

Цей код реєструє модулі залежностей (mainDI), забезпечуючи ініціалізацію компонентів, таких як ViewModel та менеджери.

2. Відображення головного інтерфейсу

Головна активність (MainActivity) ініціалізує навігацію між фрагментами.

```
val navHostFragment = supportFragmentManager
   .findFragmentById(R.id.nav_host_fragment) as NavHostFragment
navController = navHostFragment.navController
```

Пояснення: Клас NavController забезпечує навігацію між фрагментами, включаючи CreateFragment, PreviewFragment, і ResultFragment.

3. Введення даних у CreateFragment

Користувач заповнює форму з інформацією про бізнес-картку: ім’я, прізвище, телефон і посаду.

```
binding.textFieldName.editText?.addTextChangedListener(this)
```

```
binding.textFieldLastName.editText?.addTextChangedListener(this)
binding.textFieldPhone.editText?.addTextChangedListener(this)
binding.textFieldWorkPosition.editText?.addTextChangedListener(this)
```

Пояснення: Поля введення підключені до спостерігачів, які перевіряють валідність даних.

4. Перевірка валідності введених даних

Користувач натискає кнопку створення, і система перевіряє валідність кожного поля.

```
private fun checkValidation(inputLayout: TextInputLayout): Boolean {
    return if (!inputLayout.editText!!.text.isNullOrEmpty() &&
            inputLayout.editText!!.text.length <= 15) {
        inputLayout.helperText = null
        true
    } else {
        inputLayout.helperText = "must not be empty and not over 16 words"
        false
    }
}
```

Метод checkValidation переконується, що кожне поле заповнено коректно.

5. Передача даних у ViewModel

Після натискання кнопки створення дані передаються до MainViewModel.

```
mainViewModel.setNewCardModel(
    BusinessCardModel(
        binding.textFieldName.editText!!.text.toString(),
        binding.textFieldLastName.editText!!.text.toString(),
        binding.textFieldPhone.editText!!.text.toString(),
```

```
            binding.textFieldWorkPosition.editText!!.text.toString()
        )
    )
```

Об'єкт BusinessCardModel, який містить дані візитівки, створюється та передається до ViewModel.

6. Створення PDF-документа

ViewModel викликає PDFManager для створення PDF-файлу.

```
fun createPDFDocument(name: String, model: BusinessCardModel) {
    viewModelScope.launch {
        pdfManager.createPDF(name, model) {
            mutablePDFInfo.postValue(it)
        }
    }
}
```

Метод createPDFDocument запускає асинхронний процес створення PDF, передаючи назву файлу та дані бізнес-картки до PDFManager.

7. Генерація PDF у PDFManager

PDFManager створює PDF-файл і зберігає його у зовнішньому сховищі.

```
val title = Paint()
val myPage = doc.startPage(pageConsist)
val canvas: Canvas = myPage.canvas

title.typeface = Typeface.create(Typeface.SANS_SERIF, Typeface.NORMAL)
title.textSize = 14f
canvas.drawText("${modelCard.name}, ${modelCard.lastName}", 20f, 30f, title)
doc.finishPage(myPage)
```

```
withContext(Dispatchers.IO) {
    doc.writeTo(FileOutputStream(file))
}
```

Текст візитівки малюється на полотні Canvas, а документ зберігається у форматі PDF.

8. Повернення результату користувачеві

Після завершення процесу створення користувач отримує повідомлення про успіх.

```
mainViewModel.pdfInfo.observe(viewLifecycleOwner) {
    findNavController().navigate(
        R.id.action_createFragment_to_resultFragment,
        bundleOf("namePDF" to it)
    )
}
```

LiveData в ViewModel сповіщає про успішне створення PDF, і фрагмент навігації відображає результат.

Цей алгоритм демонструє, як компоненти архітектури взаємодіють для досягнення основної функціональності застосунку, забезпечуючи чіткий і ефективний процес створення бізнес-карток.

4.2. Опис та побудова діаграми класів розробленого застосунку

UML-діаграма класів відображає архітектуру програмного забезпечення для створення електронних візитівок (див. рисунок 4.1.). Вона відображає структуру основних класів, інтерфейсів та їх взаємодію в розробленому нами

Android-застосунку.

Основні класи:

App:

Базовий клас додатку, відповідальний за ініціалізацію середовища залежностей Koin та завантаження необхідних модулів. Він виконує роль стартової точки додатку.

MainActivity:

Основний клас активності, який забезпечує навігацію між фрагментами інтерфейсу користувача.

Має методи для переходу між екранами (transitionFragment) та обробки подій повернення (backTransition).

Фрагменти:

CreateFragment: Забезпечує інтерфейс для введення даних бізнес-картки. Основні функції включають перевірку коректності введених даних (checkValidation) та обробку події створення PDF-документа (createButtonAction).

PreviewFragment: Дозволяє користувачам попередньо переглядати вміст перед його збереженням. Забезпечує навігацію до інших частин додатку.

ResultFragment: Надає можливість перегляду створених PDF-документів, зокрема за допомогою інтеграції з файловою системою.

MainViewModel:

Реалізує бізнес-логіку програми. Зберігає стан та дані бізнес-карток у вигляді об’єкта LiveData.

Основні функції включають створення PDF-документів (createPDFDocument) та їх відкриття (openPDFDocument).

PDFManager:

Реалізує функціонал створення та перегляду PDF-документів. Він взаємодіє з файловою системою для збереження документів у зовнішньому сховищі.

BusinessCardModel:

Представляє модель даних бізнес-картки, що включає ім'я, прізвище, номер телефону та посаду. Цей клас є центральним елементом для обробки даних.

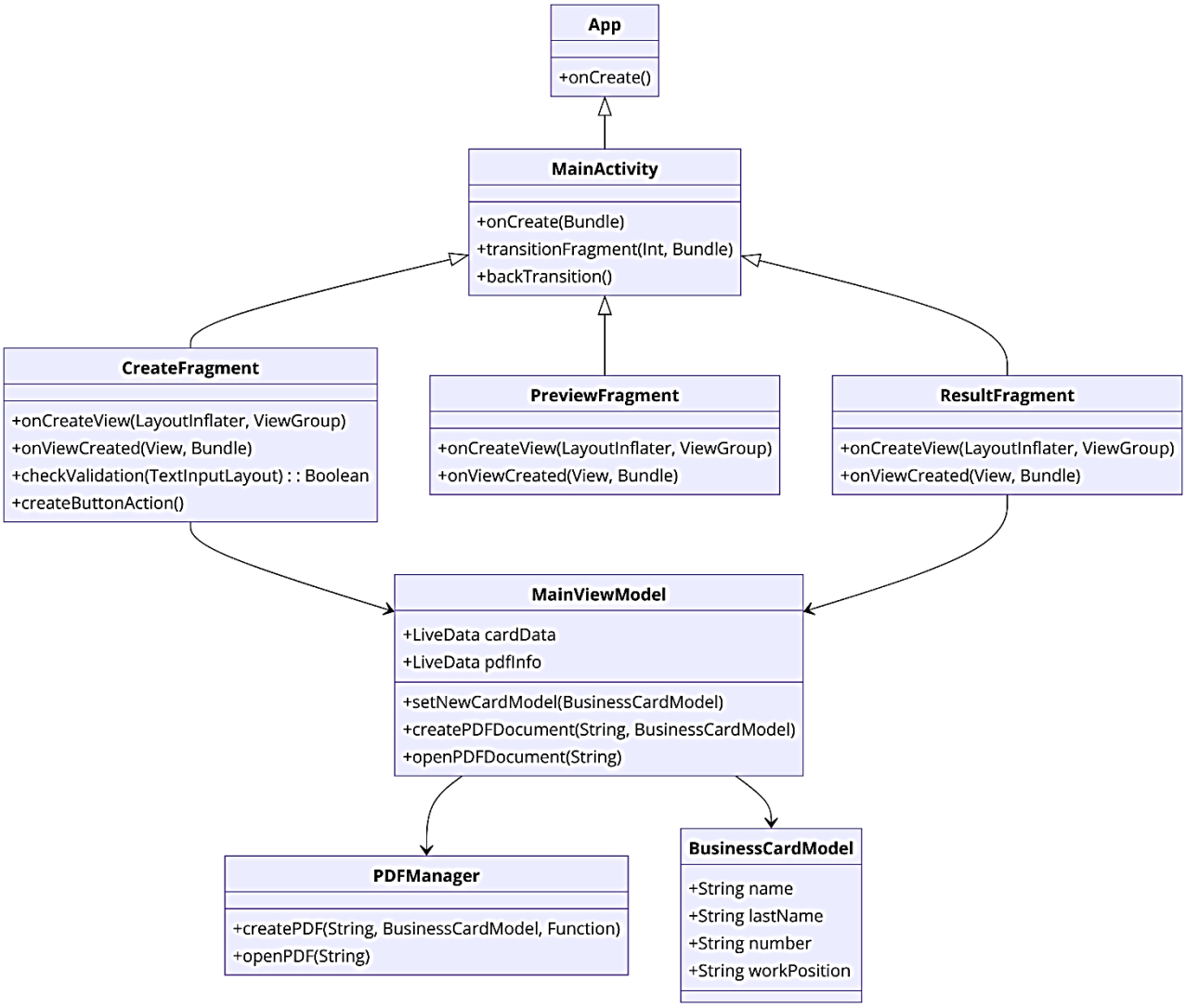

Рисунок 4.1. Діаграма класів андроїд застосунку для генерації візитівок

Зв’язки між класами:

App ініціалізує MainActivity, яка є основним модулем взаємодії з користувачем.

MainActivity здійснює навігацію між фрагментами інтерфейсу користувача: CreateFragment, PreviewFragment та ResultFragment.

CreateFragment та ResultFragment взаємодіють із MainViewModel для отримання та передачі даних.

MainViewModel використовує PDFManager для створення та перегляду PDF-документів, а також працює з об'єктами класу BusinessCardModel.

Представлена діаграма класів відображає чітко організовану архітектуру з розподілом обов’язків між компонентами. Вона забезпечує модульність, спрощує підтримку і розвиток додатку, а також забезпечує легкість тестування окремих компонентів.

4.3. Інтерфейс користувача та функціональні можливості застосунку

На рисунку (Рисунок 4.2.) зображено інтерфейс мобільного додатку з кнопкою в центрі. Кнопка має напис «Create PDF business card», що перекладається як «Створити візитівку у форматі PDF».

Фон скріншоту білий з абстрактною формою внизу кольору бордо. Дизайн мінімалістичний та сучасний.

Ця кнопка веде до функції створення візитки, яку можна зберегти у форматі PDF. Цей екран може бути частиною процесу, в якому користувач може ввести свої особисті дані або вибрати шаблон для створення візитки.

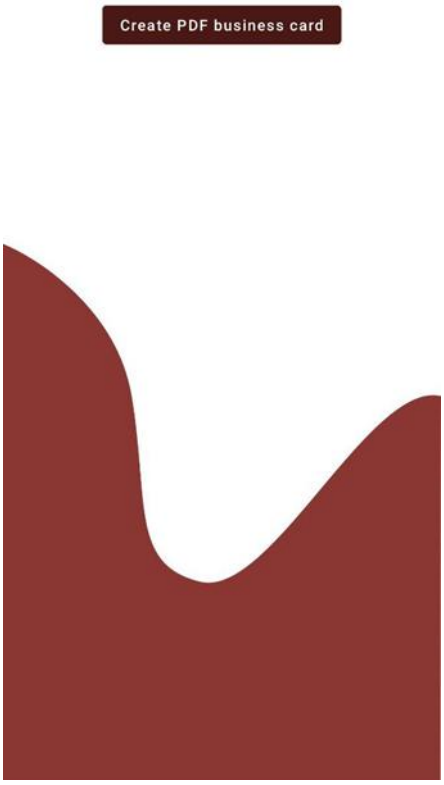

Рис. 4.2. Основний екран застосунку

На цьому скріншоті ми бачимо форму для введення даних у додатку «Business Card Creator» (Рисунок 4.3).

Вона містить чотири текстові поля для введення особистої інформації: ім'я («name»), прізвище («last name»), номер телефону («phone») та посаду («work position»).

Внизу екрана розташована кнопка «CREATE» («СТВОРИТИ») бордового кольору, яка використовується для генерації візитної картки після заповнення усіх полів. У верхньому лівому куті розташована стрілка для повернення на попередній екран. Загальний дизайн чистий і сучасний, що сприяє легкому орієнтуванню та використанню додатку користувачами.

Рисунок 4.3. Форма для введення даних

На наступному рисунку (Рисунок 4.4.) зображено форму заповнення даних в додатку для створення візиток.

У формі вже введені деякі дані: ім'я «Ілон», прізвище «Маск», номер телефону «+33 000 000 000», та посада «owner Space X» (власник Space X).

Внизу екрана активна клавіатура, що свідчить про те, що користувач наразі редагує дані. У верхній частині екрану зліва є стрілка для повернення на попередній екран. Кнопка «CREATE» розташована внизу, під полями для введення, готова для створення візитки на основі введених даних. Форма має чистий дизайн, який полегшує введення інформації

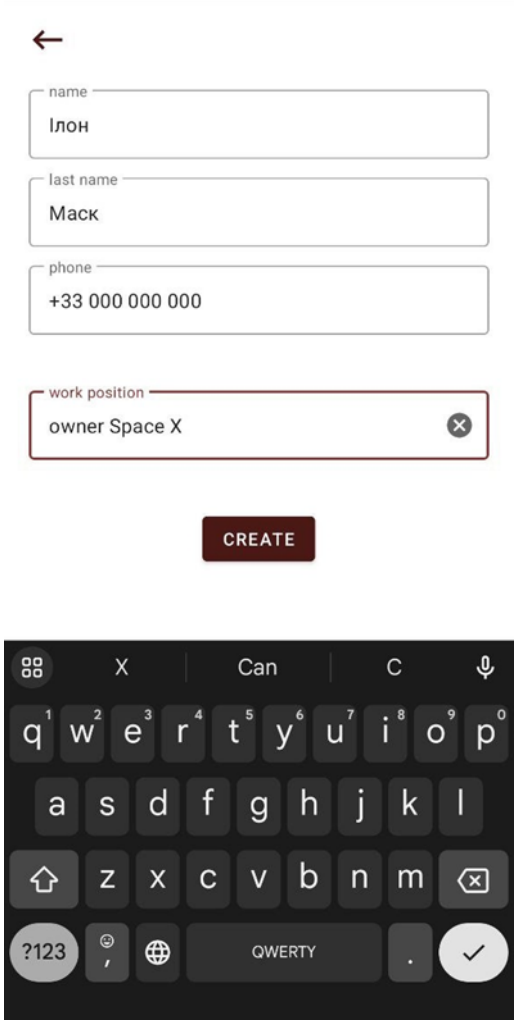

Рисунок 4.4. Приклад заповнення даних для генерації візитівки

На наступному рисунку (Рисунок 4.5.) зображено інтерфейс користувача додатку для створення візиток, де вгорі екрана знаходиться стрілка, що вказує вліво, що є кнопкою для повернення на перший екран. Нижче розміщено текст «Just have saved», який означає, що дія збереження щойно була виконана. Під цим текстом знаходиться кнопка «View PDF», що пропонує переглянути щойно збережений PDF-документ. Фон екрана білий з абстрактним декоративним елементом бордового кольору. Цілісний дизайн відповідає мінімалістичному стилю, який був присутній на попередніх етапах створення візитівки в додатку.

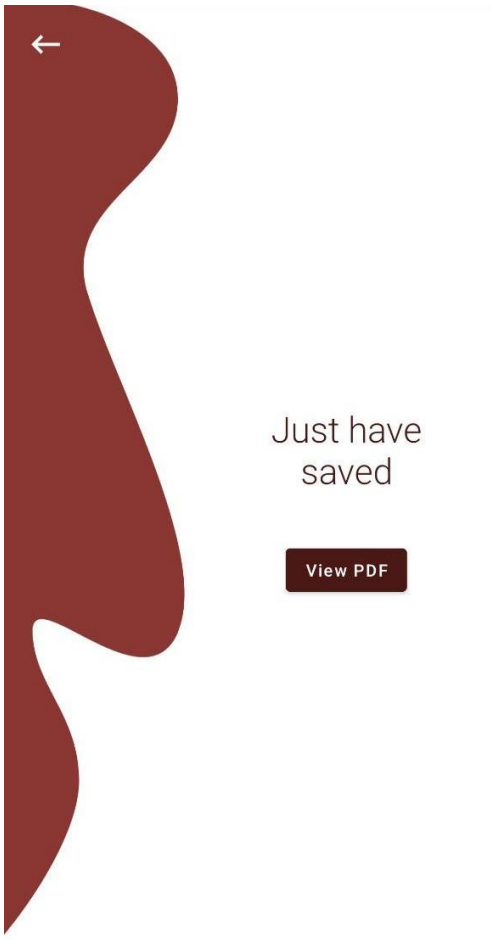

Рисунок 4.5. Етап збереження введених даних

На рисунку 4.6. зображено зразок візитівки, створеної у додатку «Business Card Creator».

Рисунок 4.6. Приклад згенерованої візитівки

На картці зазначені такі деталі:

Ім'я: Ілон

Прізвище: Маск

Номер телефону: +33 000 000 000

Посада: owner Space X

Картка має простий та чистий дизайн з текстом, розташованим на білому фоні. Ім'я та прізвище вказані великими літерами, нижче розміщено номер телефону, а ще нижче - назву посади. Зображення надає візуальне уявлення про кінцевий продукт, який користувач може створити за допомогою цього додатку.

Кожна картка зберігається на пристрої під унікальним ім’ям зза допомогою UUID, щоб уникнути помилок при збереженні

Розроблений нами андроїд застосунок для генерації візитівок є спеціалізованим програмним забезпеченням для створення електронних бізнес-карток, що забезпечує широкий набір функціональних можливостей, спрямованих на зручність користувача та автоматизацію ключових процесів. Його функціональність охоплює всі етапи створення, перевірки, збереження та перегляду бізнес-карток у цифровому форматі.

Основною функцією програми є надання користувачам інтерфейсу для введення персоналізованих даних, таких як ім’я, прізвище, номер телефону та посада. Завдяки інтерактивній формі з полями введення, користувачі можуть швидко і зручно вносити необхідну інформацію. Реалізована перевірка валідності введених даних забезпечує точність та коректність інформації, попереджаючи помилки через некоректні або порожні значення.

Програма автоматично обробляє введені дані та передає їх до бізнес-логіки, реалізованої у вигляді архітектури Model-View-ViewModel (MVVM). Ця структура дозволяє відокремити інтерфейс користувача від обробки даних, забезпечуючи модульність і спрощуючи підтримку коду. Дані користувача обробляються ViewModel, яка викликає функціональність менеджера PDF-документів для створення цифрового документа. PDF-документ генерується автоматично із заданими параметрами, включаючи структуроване розташування текстових даних на полотні.

Програма також включає функції для збереження створених документів у зовнішньому сховищі пристрою, що полегшує доступ до них для подальшого використання. Менеджер PDF-документів підтримує можливість перегляду файлів безпосередньо у додатку або через інші зовнішні програми, інтегровані з операційною системою. Це дозволяє користувачам перевіряти результат роботи програми та використовувати створені файли для професійних або особистих потреб.

Додаток підтримує багатоступеневу навігацію, яка дозволяє користувачам переходити між різними етапами роботи, такими як введення даних, попередній перегляд та перегляд результатів. Завдяки інтеграції з бібліотекою навігації Android, досягається плавність переходів та зручність у користуванні. Крім того, забезпечено обробку дозволів на доступ до зовнішнього сховища, що є важливою частиною для забезпечення сумісності з сучасними версіями операційної системи Android.

Таким чином, цей додаток поєднує зручний користувацький інтерфейс, модульну архітектуру та потужні інструменти для автоматизації створення електронних бізнес-карток, забезпечуючи гнучкість та надійність для широкого спектра користувачів.

ВИСНОВКИ

Актуальність розробки Android-застосунку для генерації візитівок зумовлена швидкою цифровізацією та зростанням попиту на мобільні технології у бізнес-середовищі. Візитівки залишаються важливим інструментом професійної комунікації, однак перехід до електронного формату дозволяє значно розширити їх функціональність. Android, як одна з найпопулярніших мобільних платформ, забезпечує широке охоплення користувачів завдяки доступності, інтерактивності та можливості інтеграції з іншими цифровими інструментами. Розробка таких застосунків дозволяє створювати персоналізовані електронні картки, інтегровані з соціальними мережами, CRM-системами та хмарними сервісами, що сприяє підвищенню ефективності обміну інформацією.

Цифрові візитівки також мають екологічний вимір, сприяючи зменшенню використання паперу та ресурсів. Використання таких технологій, як NFC та QR-коди, дозволяє миттєво обмінюватися контактами, відповідаючи сучасним вимогам до швидкості та зручності комунікацій. Завдяки доступності розробницьких інструментів і широким технічним можливостям Android, такі рішення є досяжними навіть для малого та середнього бізнесу, забезпечуючи їм можливість автоматизації та інновацій у сфері бізнес-комунікацій.

Отже, у результати виконання дипломного проєкту нами було повністю виконано поставлені завдання дипломної роботи, а саме:

- визначено ключові вимоги до розробки програмного забезпечення андроїд застосунку для генерації візитівок;
- проведено аналіз переваг та обмежень аналогічних застосунків та джерел отримання візитівок;
- проаналізовано та описано підходи та інструменти, використані при розробці ПЗ застосунку;
- описано алгоритм, методолію створення та діаграму послідовностей

розробленого андроїд застосунку;

- розроблено мобільний інтерфейс для взаємодії користувача з матеріалом та створено докладну інструкцію користувача;

- представлено діаграму класів, що ілюструє особливості програмного коду розробленого застосунку;

- розроблено та протестовано андроїд застосунок для генерації візитівок.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Черненко О. О. Методичні рекомендації щодо виконання кваліфікаційної роботи студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня магістра / С. В. Гаркуша, О. В. Ольховська, О. О. Черненко. – Полтава : ПУЕТ, 2023. – 68 с. Режим доступу: URL: http://elib.puet.edu.ua/action.php?kt_path_info=ktcore.SecViewPlugin.actions.document&fDocumentId=824868 - Назва з екрану

2. AndroidX with Xamarin. Режим доступу: URL: https://learn.microsoft.com/en-us/previous-versions/xamarin/android/platform/androidx - Назва з екрану

3. Antonio Leiva.View Binding: The Definitive way to access views on Android. Режим доступу: URL: https://antonioleiva.com/view-binding-android/ - Назва з екрану

4. Developer workflow basics. Режим доступу: URL: https://developer.android.com/studio/workflow - Назва з екрану

5. Data Binding Library Режим доступу: URL: https://developer.android.com/topic/libraries/data-binding/?utm_source=chatgpt.com - Назва з екрану

6. Kotlin for Android. Режим доступу: URL: https://kotlinlang.org/docs/android-overview.html - Назва з екрану

7. Kotlin home page. Режим доступу: URL: https://kotlinlang.org/ - Назва з екрану

8. Meet Android Studio. Режим доступу: URL: https://developer.android.com/studio/intro - Назва з екрану

9. Model-View-ViewModel (MVVM). Режим доступу: URL: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm - Назва з екрану

10. MVVM Clean Architecture Pattern in Android with Use Cases. Режим доступу: URL:https://medium.com/@ami0275/mvvm-clean-architecture-pattern-in-android-with-use-cases-eff7edc2ef76 - Назва з екрану

11. Navigation. Режим доступу: URL: https://developer.android.com/guide/navigation - Назва з екрану

12. A detailed summary of Clean Architecture by Robert C Martin. Режим доступу: URL: https://github.com/serodriguez68/clean-architecture?utm_source=chatgpt.com - Назва з екрану

13. Welcome to our tour of Kotlin. Режим доступу: URL: https://kotlinlang.org/docs/kotlin-tour-welcome.html - Назва з екрану

14. Waterfall Model for Software Development and Testing Teams. Режим доступу: URL: https://qatestlab.com/resources/knowledge-center/waterfall-process/ - Назва з екрану

15. КОШОВА, О., ЧЕРНЕНКО, О., ОРІХІВСЬКА, О., ТУР, В., & ЯНКО, О. (2024). РОЗРОБКА НАВЧАЛЬНОГО АНДРОЇД-ЗАСТОСУНКУ З ТЕМИ «СОРТУВАННЯ ВСТАВКАМИ» ДИСТАНЦІЙНОГО НАВЧАЛЬНОГО КУРСУ «АЛГОРИТМИ І СТРУКТУРИ ДАНИХ». *Інформаційні технології та суспільство*, (5 (11), 34-42. https://doi.org/10.32689/maup.it.2023.5.5

ДОДАТОК А
Код програми

```
package com.crecardbissnes.ebissnescard

import android.app.Application
import com.crecardbissnes.ebissnescard.di.mainDI
import org.koin.android.ext.koin.androidContext
import org.koin.android.ext.koin.androidLogger
import org.koin.core.context.startKoin
import org.koin.core.logger.Level

class App : Application() {

    override fun onCreate() {
        super.onCreate()

        startKoin {
            androidLogger(level = Level.INFO)
            androidContext(this@App)
            modules(listOf(mainDI))
        }

    }
}



package com.crecardbissnes.ebissnescard.presenter.ui

import android.os.Bundle
import android.view.View
import androidx.appcompat.app.AppCompatActivity
import androidx.navigation.NavController
import androidx.navigation.fragment.NavHostFragment
```

```
import com.crecardbissnes.ebissnescard.R
import com.crecardbissnes.ebissnescard.databinding.ActivityMainBinding
import com.crecardbissnes.ebissnescard.domain.MainNavigateTransition


class MainActivity : AppCompatActivity(), MainNavigateTransition {

    private lateinit var binding: ActivityMainBinding
    private var navController: NavController? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        val navHostFragment = supportFragmentManager
            .findFragmentById(R.id.nav_host_fragment) as NavHostFragment

        navController = navHostFragment.navController

        binding.mainTransition = this

        navController?.addOnDestinationChangedListener { _, destination, _ ->
            when (destination.label) {
                "Viewer" -> {
                    binding.backButton.setImageResource(R.drawable.ic_back_white)
                    binding.backButton.visibility = View.VISIBLE
                }
                "Creating" -> {
                    binding.backButton.setImageResource(R.drawable.ic_back_red)
                    binding.backButton.visibility = View.VISIBLE
                }

                else -> {
```

```
                binding.backButton.visibility = View.INVISIBLE
            }
        }

    }

}

override fun transitionFragment(nav: Int, bundle: Bundle?) {
    navController?.navigate(nav, bundle)
}

override fun backTransition() {
    this@MainActivity.onBackPressed()
}



}

package com.crecardbissnes.ebissnescard.presenter.ui.fragment


import android.Manifest
import android.content.Intent
import android.content.pm.PackageManager
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.os.Environment
import
android.provider.Settings.ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION
import android.text.Editable
import android.text.TextWatcher
```

```
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.content.ContextCompat
import androidx.core.os.bundleOf
import androidx.fragment.app.Fragment
import androidx.navigation.fragment.findNavController
import com.crecardbissnes.ebissnescard.BuildConfig.APPLICATION_ID
import com.crecardbissnes.ebissnescard.R
import com.crecardbissnes.ebissnescard.databinding.FragmentCreateBinding
import com.crecardbissnes.ebissnescard.domain.MainNavigateTransition
import com.crecardbissnes.ebissnescard.domain.models.BusinessCardModel
import com.crecardbissnes.ebissnescard.presenter.viewmodel.MainViewModel
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.google.android.material.textfield.TextInputLayout
import org.koin.androidx.viewmodel.ext.android.viewModel
import java.util.*


class CreateFragment : Fragment(), TextWatcher {

    private lateinit var binding: FragmentCreateBinding
    private val mainViewModel by viewModel<MainViewModel>()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentCreateBinding.inflate(inflater)
        return binding.root
    }
```

```
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.nav = R.id.action_createFragment_to_resultFragment
        binding.bundle = null
        binding.mainTransition = requireActivity() as MainNavigateTransition


        mainViewModel.pdfInfo.observe(viewLifecycleOwner) {
            findNavController().navigate(
                R.id.action_createFragment_to_resultFragment,
                bundleOf(
                    "namePDF" to it
                )
            )
        }

        mainViewModel.cardData.observe(viewLifecycleOwner) {
            val name = "business_card_${UUID.randomUUID()}.pdf"
            mainViewModel.createPDFDocument(name, it)
        }


        binding.createButton.setOnClickListener {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
                if (Environment.isExternalStorageManager()) createButtonAction()
                else
requestPermissionLauncher.launch(Manifest.permission.MANAGE_EXTERNAL_STORAGE)
            } else {

requestPermissionLauncher.launch(Manifest.permission.WRITE_EXTERNAL_STORAGE)
            }
        }
```

```
        binding.textFieldName.editText?.addTextChangedListener(this)
        binding.textFieldLastName.editText?.addTextChangedListener(this)
        binding.textFieldPhone.editText?.addTextChangedListener(this)
        binding.textFieldWorkPosition.editText?.addTextChangedListener(this)

    }


    override fun beforeTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
    }

    override fun onTextChanged(p0: CharSequence?, p1: Int, p2: Int, p3: Int) {
        binding.createButton.isEnabled = checkValidation(binding.textFieldName) &&
                checkValidation(binding.textFieldLastName) &&
                checkValidation(binding.textFieldPhone) &&
                checkValidation(binding.textFieldWorkPosition)
    }

    override fun afterTextChanged(p0: Editable?) {
    }

    private fun checkValidation(inputLayout: TextInputLayout): Boolean {
        return if (!inputLayout.editText!!.text.isNullOrEmpty() && inputLayout.editText!!.text.length
<= 15) {
            inputLayout.helperText = null
            true
        } else {
            inputLayout.helperText = "must not be empty and not over " +
                    "${if (inputLayout == binding.textFieldWorkPosition) "26" else "16"} words"
            false
        }
    }

    private val dialogPermissions by lazy {
```

```
        MaterialAlertDialogBuilder(requireContext())
            .setTitle("Permission Dialog")
            .setMessage("To continue, you need to grant permission (read smartphone memory) in the app settings")
            .setPositiveButton("Ok") { dialog, _ ->
                dialog.dismiss()
                requestPermissions()
            }
    }

    private val requestPermissionLauncher =
        registerForActivityResult(
            ActivityResultContracts.RequestPermission()
        ) { isGranted: Boolean ->
            if (isGranted) {
                createButtonAction()
            } else {
                dialogPermissions.show()
            }
        }

    private fun requestPermissions() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.R) {
            if (Environment.isExternalStorageManager()) createButtonAction()
            else {
                val intent = Intent(
                    ACTION_MANAGE_APP_ALL_FILES_ACCESS_PERMISSION,
                    Uri.parse("package:$APPLICATION_ID")
                )
                startActivity(intent)
            }

        } else when (PackageManager.PERMISSION_GRANTED) {
            ContextCompat.checkSelfPermission(
```

```
                requireContext(),
                Manifest.permission.WRITE_EXTERNAL_STORAGE
            ), ContextCompat.checkSelfPermission(
                requireContext(),
                Manifest.permission.MANAGE_EXTERNAL_STORAGE
            ) -> createButtonAction()

        }
    }

    private fun createButtonAction() {
        mainViewModel.setNewCardModel(
            BusinessCardModel(
                binding.textFieldName.editText!!.text.toString(),
                binding.textFieldLastName.editText!!.text.toString(),
                binding.textFieldPhone.editText!!.text.toString(),
                binding.textFieldWorkPosition.editText!!.text.toString()
            )
        )
    }


}

package com.crecardbissnes.ebissnescard.presenter.ui.fragment

import android.Manifest
import android.content.Intent
import android.content.pm.PackageManager
import android.location.LocationManager
import android.os.Bundle
import android.provider.Settings
import android.util.Log
import android.view.LayoutInflater
```

```
import android.view.View
import android.view.ViewGroup
import androidx.activity.result.contract.ActivityResultContracts
import androidx.core.content.ContextCompat
import androidx.fragment.app.Fragment
import androidx.navigation.fragment.findNavController
import com.crecardbissnes.ebissnescard.R
import com.crecardbissnes.ebissnescard.databinding.FragmentPreviewBinding
import com.crecardbissnes.ebissnescard.domain.MainNavigateTransition

class PreviewFragment : Fragment() {

    private lateinit var binding: FragmentPreviewBinding

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentPreviewBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.bundle = null
        binding.nav = R.id.action_previewFragment_to_createFragment
        binding.mainTransition = requireActivity() as MainNavigateTransition

    }


}
```

```
package com.crecardbissnes.ebissnescard.presenter.ui.fragment

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import com.crecardbissnes.ebissnescard.databinding.FragmentResultBinding
import com.crecardbissnes.ebissnescard.presenter.viewmodel.MainViewModel
import org.koin.androidx.viewmodel.ext.android.viewModel

class ResultFragment : Fragment() {

    private lateinit var binding: FragmentResultBinding
    private val mainViewModel by viewModel<MainViewModel>()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentResultBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.viewButton.setOnClickListener {
            arguments?.let {
                it.getString("namePDF")?.let { fileName ->
                    mainViewModel.openPDFDocument(fileName)
                }
```

```
            }
        }

    }

}

package com.crecardbissnes.ebissnescard.presenter.viewmodel

import android.util.Log
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.crecardbissnes.ebissnescard.domain.PDFManager
import com.crecardbissnes.ebissnescard.domain.models.BusinessCardModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class MainViewModel(private val pdfManager: PDFManager) : ViewModel() {

    private val mutableCardData = MutableLiveData<BusinessCardModel>()
    val cardData: LiveData<BusinessCardModel>
        get() = mutableCardData

    fun setNewCardModel(newModel: BusinessCardModel) {
        mutableCardData.value = newModel
    }

    private val mutablePDFInfo = MutableLiveData<String>()
    val pdfInfo: LiveData<String>
        get() = mutablePDFInfo

    fun createPDFDocument(name: String, model: BusinessCardModel) {
```

```
        viewModelScope.launch {
            pdfManager.createPDF(name, model) {
                mutablePDFInfo.postValue(it)
            }
        }
    }

    fun openPDFDocument(name: String) {
        pdfManager.openPDF(name)
    }

}

package com.crecardbissnes.ebissnescard.domain

import android.content.ActivityNotFoundException
import android.content.Context
import android.content.Intent
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.Typeface
import android.graphics.pdf.PdfDocument
import android.net.Uri
import android.os.Environment
import android.util.Log
import androidx.core.content.ContextCompat
import androidx.core.content.ContextCompat.startActivity
import androidx.core.content.FileProvider
import com.crecardbissnes.ebissnescard.R
import com.crecardbissnes.ebissnescard.domain.models.BusinessCardModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import java.io.File
import java.io.FileOutputStream
```

```
import java.io.IOException

class PDFManager(private val context: Context) {

    suspend fun createPDF(name: String, modelCard: BusinessCardModel, end: (String) -> Unit) {
        val doc = PdfDocument()

        val pageConsist = PdfDocument.PageInfo.Builder(200, 100, 1).create()

        val path =

Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS).pat
h
        val dir = File(path)
        val file = File(dir, name)



        val title = Paint()

        val myPage = doc.startPage(pageConsist)

        val canvas: Canvas = myPage.canvas

        title.typeface = Typeface.create(Typeface.SANS_SERIF, Typeface.NORMAL)
        title.textSize = 14f
        title.color = ContextCompat.getColor(context, R.color.black)
        canvas.drawText("${modelCard.name}, ${modelCard.lastName}", 20f, 30f, title)
        title.textSize = 9f
        canvas.drawText(modelCard.number, 20f, 45f, title)
        title.textSize = 10f
        canvas.drawText(modelCard.workPosition, 20f, 75f, title)

        doc.finishPage(myPage)
```

```
        try {

            withContext(Dispatchers.IO) {
                doc.writeTo(FileOutputStream(file))
            }

        } catch (e: IOException) {
            Log.i("myLog create", e.toString())
        } finally {
            doc.close()
            end(name)
        }

    }

    fun openPDF(name: String) {
        val file = File(

Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS),
            name
        )
        val excelPath: Uri =
            FileProvider.getUriForFile(
                context,
                "com.crecardbissnes.ebissnescard.provider",
                file
            )

        val pdfIntent = Intent(Intent.ACTION_VIEW)
        pdfIntent.setDataAndType(excelPath, "application/pdf")

        pdfIntent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
        pdfIntent.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY)
        pdfIntent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP)
```

```
        pdfIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)

        try {
            context.startActivity(pdfIntent)
        } catch (e: ActivityNotFoundException) {
            Log.i("myLog result", e.toString())
        }
    }


}

package com.crecardbissnes.ebissnescard.domain.models

data class BusinessCardModel(
    val name: String,
    val lastName: String,
    val number: String,
    val workPosition: String
)

package com.crecardbissnes.ebissnescard.di

import com.crecardbissnes.ebissnescard.domain.PDFManager
import com.crecardbissnes.ebissnescard.presenter.viewmodel.MainViewModel
import org.koin.androidx.viewmodel.dsl.viewModel
import org.koin.dsl.module

val mainDI = module {

    factory {
        PDFManager(get())
    }
```

```
    viewModel {
        MainViewModel(get())
    }

}
```