

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРИЛОЖЕНИЕ ПО ФОРМИРОВАНИЮ
ГАРДЕРОБА ПУТЕШЕСТВЕННИКА

Студент

Сорокина 03.06.2025 Сорокина Елена Андреевна
(подпись, дата) (ФИО полностью)

Руководитель

Е 03.06.2025 Гурин Е.И.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

Бычков 03.06.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой

М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

6 июня 2025 г. № 12)

Секретарь ГЭК

Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 06 » мая 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Сорокина Елена Андреевна гр. 21ВВП1 факультета вычислительной техники

2. Тема работы Приложение по формированию гардероба путешественника.

Тема утверждена приказом ПГУ № 601/о от "6" мая 2025г.

3. Руководитель бакалаврской работы профессор кафедры ВТ ПГУ Гурин Е.И.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать Android-приложение с использованием архитектуры MVVM, обладающее интуитивно понятным интерфейсом и комплексным функционалом для работы с изображениями и мониторинга погоды.

Приложение должно реализовывать следующие функции: управление медиафайлами, включая добавление, удаление фотографий, создание и редактирование каталогов для сортировки изображений, отображение текущей погоды и прогноза с использованием сторонних API, возможность выбора локации для мониторинга. Программа должна обладать интуитивно понятным интерфейсом в приложении и использовать следующие технологии:

1. Язык программирования Kotlin;
2. Облачная СУБД Firebase;
3. API OpenWeatherApi.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги приложений по формированию виртуального гардероба, определить функционал реализуемого приложения и требования к нему, выбрать стек технологий. Описать архитектуру приложения, реализовать базовый функционал приложения включая связь с API погоды.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	4%	06.05.2025	<i>Е</i>
2	Анализ требований	8%	07.05.2025- 08.05.2025	<i>Е</i>
3	Выбор технологий, применяемых при проектировании и разработке	8%	09.05.2025- 10.05.2025	<i>Е</i>
4	Проектирование архитектуры	12%	11.05.2025- 13.05.2025	<i>Е</i>
5	Проектирование и Разработка ПО	44%	14.05.2025- 24.05.2025	<i>Е</i>
6	Тестирование и отладка	12%	25.05.2025- 27.05.2025	<i>Е</i>
7	Оформление выпускной работы	12%	28.05.2025- 30.05.2025	<i>Е</i>

Дата выдачи задания "6" мая 2025 г.

Руководитель бакалаврской работы *Е* 06.05.2025 Гурин Е.И.
(подпись, дата)

Задание к исполнению принял студент *Сорокина* 06.05.25 Сорокина Е.А.
(подпись, дата)

Работу к защите допустить
Декан факультета *Л.Р.* 05.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 77 листов, 40 рисунков, 1 таблицу, 7 источников, 3 приложения.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ГАРДЕРОБ, ПУТЕШЕСТВИЯ, FIREBASE AUTHENTICATION, KOTLIN, ANDROID STUDIO

Цель работы – разработать мобильное приложение для виртуальной организации гардероба путешественника.

Реализуемое приложение позволяет пользователям создать виртуальную версию своего гардероба, а именно: добавлять и удалять собственные фотографии вещей, создавать каталоги, в которые можно копировать сделанные изображения, а производить мониторинг погоды в произвольные даты в нужной локации. Для удобства использования функционала приложения необходимо обеспечить регистрацию и авторизацию.

В результате ВКР разработано мобильное приложение, которое удовлетворяет поставленным задачам.

В ВКР использован следующий стек технологий: ОС Android, IDE Android Studio, язык программирования Kotlin, СУБД Firebase.

					ПГУ1.090301.20.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Приложение по формированию гардероба путешественника	Лит.	Лист	Листов	
Разраб.		Сорокина Е.А.		23.06.25					
Провер.		Гурин Е.И.		23.06.25			2	77	
Н. Контр		Бычков А.С.		23.06.25		ФВТ, каф. ВТ гр. 21ВВП1			
Утверд.		Митрохин М.А.		23.06.25					

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Анализ предметной области	8
1.1 Анализ актуальности темы	8
1.2 Анализ существующих аналогов.....	9
1.2.1 Acloset	9
1.2.2 GetWardrobe.....	10
1.2.3 PackPoint	11
1.2.4 Вывод	12
1.3 Постановка задачи	12
1.4 Вывод	14
2 Выбор инструментов разработки	15
2.1 Выбор языка программирования.....	15
2.1.1 Java	15
2.1.2 C++	15
2.1.3 Kotlin	15
2.1.4 Вывод	16
2.2 Выбор среды разработки	16
2.2.3 Android Studio.....	16
2.2.4 Eclipse.....	17
2.2.5 Вывод	18
3 Архитектура приложения.....	19
4 Реализация приложения	23
4.1 Описание функционала	23
4.2 Описание работы приложения.....	24
4.3 Описание компонентов приложения	26
4.4 Описание интерфейса приложения	28
4.5 Тестирование приложения	31
4.5.1 Регистрация	31
4.5.2 Авторизация	34

4.5.3	Возможность делать фото.....	34
4.5.4	Создание каталога.....	36
4.5.5	Просмотр погоды.....	38
4.5.6	Итоги тестирования.....	40
4.6	Вывод	41
5	Безопасность жизнедеятельности.....	42
5.1	Правила и техника безопасности при работе с ПК	42
5.2	Правила безопасности при работе со смартфоном.....	43
	Заключение	44
	Список использованных источников	45
	Приложение А – UML-диаграммы.....	46
	Приложение Б – Листинг программы	52
	Приложение В – Презентация.....	72

Перечень сокращений и обозначений

В данной работе применяют следующие сокращения и обозначения

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

DAO – (Data Access Object) – объект доступа к данным

IDE – (Integrated Development Environment) – интегрированная среда разработки

MVVM – (Model-View-ViewModel) – модель-представление-модель представления

UI – (User Interface) – пользовательский интерфейс

UML – (Unified Modeling Language) – унифицированный язык моделирования

UX – (User Experience) – это совокупность всех аспектов взаимодействия пользователя с продуктом, системой или сервисом

					ПГУ1.090301.20.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В настоящее время невозможно представить жизнь человека без использования смартфона и мобильных приложений. Они могут существенно облегчать выполнение ежедневных задач, а также помогают в освоении нового. Один из самых распространенных видов отдыха сейчас – путешествия, так что разработка приложения, которое может помочь туристам в очередной поездке, является актуальной и нужной задачей.

Выпускная квалификационная работа посвящена разработке Android-приложения для рекомендаций по формированию гардероба путешественника. Цель реализуемого приложения – предоставление людям, которые любят путешествовать, понятного и удобного сервиса для составления и организации собственного гардероба, а также планирования поездки в зависимости от погоды в нужном городе.

Приложение предоставляет следующий список возможностей: добавление фото одежды, создание собственных образов, мониторинг погоды в выбранные даты в определенной локации.

В связи с повсеместной цифровизацией данное приложение может помочь пользователям составить свой виртуальный гардероб, все вещи которого будут необходимы в поездке. Это позволяет существенно сэкономить время на сборе чемодана для поездки и на покупке билетов эконом класса с минимальным платежом за багаж.

Технической целью ВКР является разработка интуитивно понятного и удобного Android-приложения, обладающего всем необходимым функционалом. Реализуемая программа предоставляет пользователям возможность организовать свой гардероб как в зависимости от погодных условий, так и по собственным критериям. В процессе ВКР применены навыки написания программ для мобильных устройств, с использованием сторонних API, а также организации надежной аутентификации и авторизации пользователей.

Результатом ВКР будет полезное для путешественников свободно распространяемое MVP.

					ПГУ1.090301.20.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Анализ предметной области

1.1 Анализ актуальности темы

В наше время люди часто совершают различные поездки, и для каждого повода необходимо собирать разный гардероб. Так как все поводы разные, нужно различать, для чего именно и какие вещи надо собирать, например:

- для рабочей командировки;
- для отдыха с семьей;
- для одиночной поездки.

В условиях повсеместной цифровизации и необходимостью использовать телефон в повседневных делах приложение по формированию гардероба путешественника может помочь сэкономить время. Не придется осматривать все вещи в шкафах и составлять образы из раза в раз, так как это можно будет сделать всего один раз и обновлять списки в приложении. Для путешественников предоставляется возможность собрать капсульный гардероб, который можно легко собирать для каждой поездки. Более того, в самом приложении можно посмотреть погоду в нужном городе, чтобы сразу определиться со своими планами.

Приложение так же может помочь людям определенных профессий, например, стилистам. Специалист может зайти на профиль клиента и по вещам в аккаунте помочь ему составить образы.

Приложение по формированию гардероба путешественника – сервис, в котором можно составлять образы под определенные случаи и поездки. Тема разработки такой программы актуальна, так как в настоящее время люди всё чаще и чаще стремятся путешествовать, не важно, внутренний это туризм или внешний, и такое приложение может существенно облегчить планирование поездки.

Учитывая, что на данный момент самой популярной ОС для телефонов является Android, так как её использует более 70% устройств, выбор пал на

разработку Android-приложения, ведь таким образом наибольшее количество людей сможет комфортно пользоваться программой [1].

1.2 Анализ существующих аналогов

Здесь рассматриваются существующие приложения, которые предоставляют возможности создания виртуальной версии своего гардероба. Для более детального анализа были выявлены достоинства и недостатки каждого из них.

1.2.1 Acloset

Одно из самых популярных приложений для создания электронной версии своего гардероба. Обладает всем необходимым функционалом – добавить фото, создать новый каталог, просмотр погоды. Так же присутствует искусственный интеллект, который может помочь составить образ. Однако приложение ограничивает платная подписка, без которой возможности сильно ограничены.

На рисунке 1 представлены страницы приложения «Acloset».

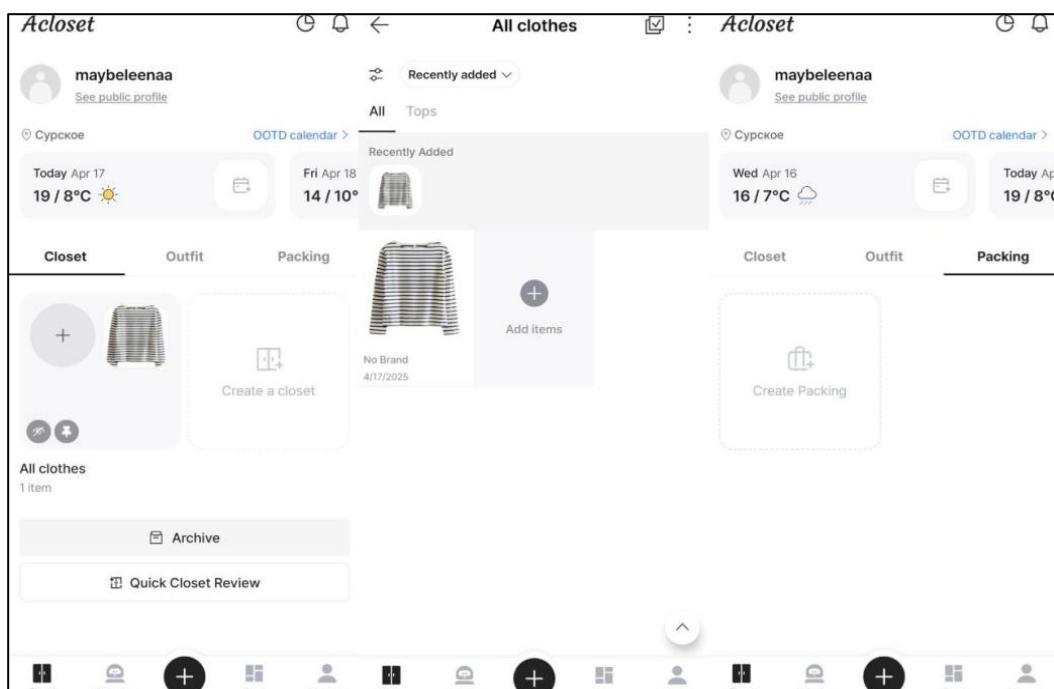


Рисунок 1 – Приложение Acloset

Достоинства приложения:

- возможность добавления собственных вещей;
- возможность составления образов;
- показ погоды по местоположению;
- наличие искусственного интеллекта.

Недостатки приложения:

- отсутствие русификации;
- ограниченный функционал без оформления платной подписки;

1.2.2 GetWardrobe

Программа, в которую так же можно загружать свои вещи, составлять различные комбинации из них и организовывать свой гардероб в одном месте. Присутствует меню с ответами на часто задаваемые вопросы для упрощения навигации по приложению. К недостаткам можно отнести то, что сервис так же ограничен своим функционалом для пользователей без платной подписки.

На рисунке 2 представлен интерфейс приложения «GetWardrobe».

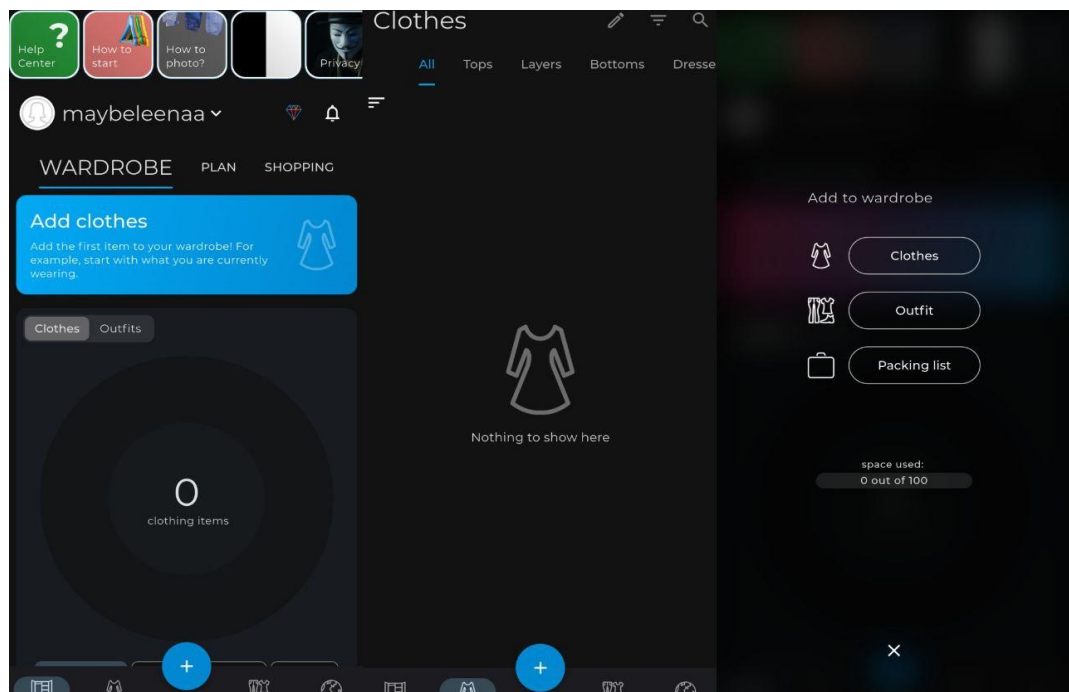


Рисунок 2 – Главная страница «GetWardrobe»

Достоинства:

- возможность добавления собственных вещей;
- возможность составления образов;
- показ погоды по местоположению.

Недостатки:

- отсутствие русификации;
- ограниченный функционал приложения без оформления платной подписки;
- интерфейс может быть не очень понятен при использовании.

1.2.3 PackPoint

Приложение позволяет быстро составить список необходимых вещей для поездки в зависимости от активностей, которые планируются. При вводе всех нужных данных просчитывается, показываются рекомендации, какие вещи лучше взять и в каком количестве.

На рисунке 3 представлено приложение PackPoint.

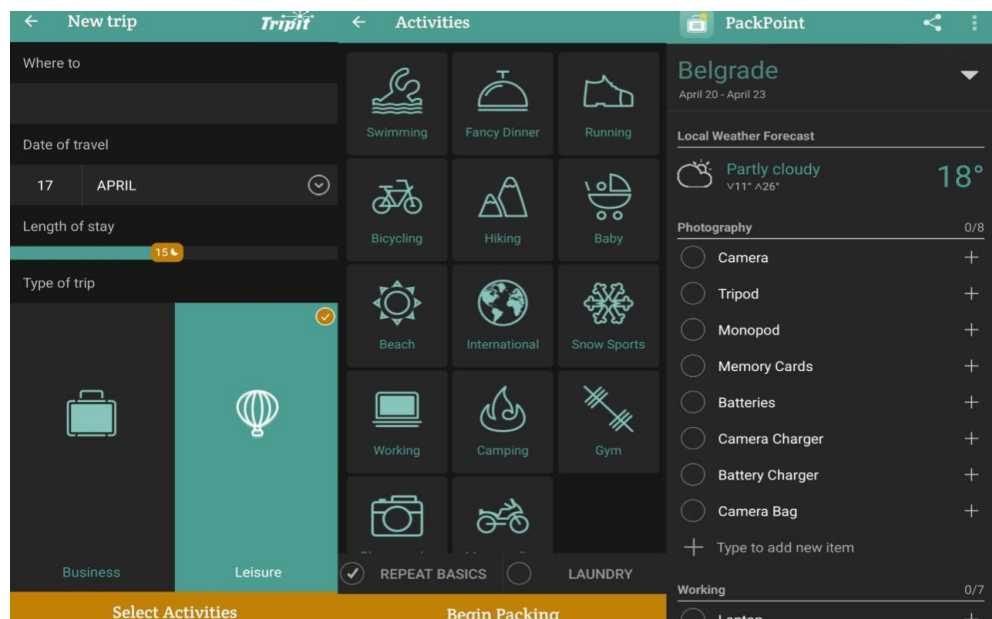


Рисунок 3 – Приложение PackPoint

Достоинства:

- возможность добавления направления поездки по всему миру;
- автоматическое составление списка необходимых вещей для поездки;

- интуитивно понятный интерфейс.

Недостатки:

- отсутствие русификации;
- ограниченный функционал приложения без оформления платной подписки;
- нет возможности загрузки собственных фото с вещами и составления образов.

1.2.4 Вывод

В процессе анализа существующих аналогов было выявлено, что у всех из них отсутствуют какие-либо базовые функции, необходимые для комфортного пользования.

В большинстве своем приложение по формированию гардероба либо не предоставляют всех возможностей без платной подписки, либо не обладают нужным функционалом, чтобы иметь представление обо всех своих вещах. Вследствие чего было принято решение создать приложение по формированию гардероба путешественника с наличием русского интерфейса, добавлением собственных фотографий, созданием образов и просмотром погоды

1.3 Постановка задачи

Необходимо разработать приложение на базе ОС Android, позволяющее добавлять вещи своего гардероба, составлять образы, сортировать одежду по различным параметрам при необходимости и просматривать погоду в определенном городе на выбранные даты.

В приложении должны быть реализованы:

- регистрация пользователя;
- добавление фотографий одежды;
- составление образов;
- возможность редактирования образов;

					<i>ПГУ1.090301.20.001 ПЗ</i>	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

- просмотр погоды в нужном городе на необходимые даты.

На главном экране приложения отображается основная информация о пользователе, а также реализована навигация, по которой можно перейти на каждую из вкладок, содержащих свой функционал, например отображение фотографий и образов, сделанных пользователем, или показ погоды по запросу.

На рисунке 4 представлена диаграмма вариантов использования.

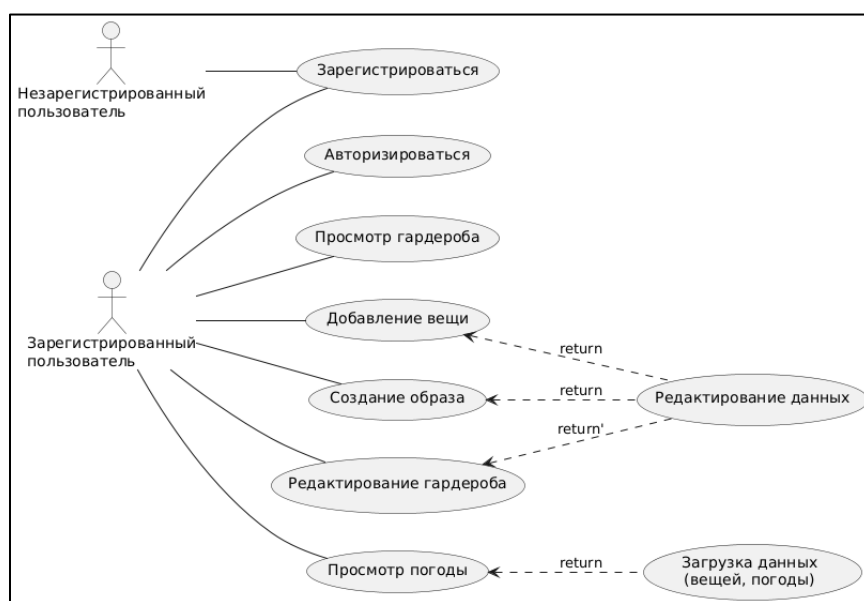


Рисунок 4 – UML-диаграмма вариантов использования

Все функции приложения доступны только авторизованным пользователям. Незарегистрированные пользователи могут зарегистрироваться, тем самым открыв все возможности для себя.

Минимальные системные требования:

- операционная система Android версии 10 или выше;
- наличие камеры на устройстве;
- подключение к сети Интернет.

1.4 Вывод

В результате проведённого анализа была определена актуальность выбранной предметной области и рассмотрены недостатки существующих аналогов. Так же была поставлена задача проекта, которая определяет всю последующую работу над приложением и его разработку.

					ПГУ1.090301.20.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

2 Выбор инструментов разработки

В этом разделе проведен анализ возможных средств, которые могут использоваться для разработки Android-приложений. В ходе исследования были выбраны наиболее удобные инструменты разработки.

2.1 Выбор языка программирования

На подавляющем большинстве телефонов стоит ОС Android, вследствие чего необходимо выбрать тот язык программирования, использование которого максимально упростит разработку и поможет сделать.

Существует множество языков программирования, на которых можно написать мобильные приложения. Рассмотрим одни из наиболее распространенных, а именно Java, C++ и Kotlin.

2.1.1 Java

Java – официальный язык программирования для разработки программного обеспечения под Android. Является основным языком в Android Studio, имеет наибольшее количество вспомогательных элементов разработки в виде фреймворков и библиотек. Очень распространен среди разработчиков, что делает его хорошим вариантом для выбора в качестве инструмента разработки.

2.1.2 C++

C++ довольно распространенный язык программирования среди многих разработчиков благодаря своей высокой производительности и другим особенностям. В большинстве своем он используется для написания игр для ОС Android. Однако, необходимость дополнительно устанавливать Android NDK (Native Development Kit) и сложности языка затрудняют разработку программ на нем.

2.1.3 Kotlin

Kotlin является высокоуровневым языком программирования. Он поддерживается компанией Google, что делает его официальным языком для

					<i>ПГУ1.090301.20.001 ПЗ</i>	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

разработки ПО для мобильных устройств. Этот язык представляет собой надстройку над Java, но имеет лучшую типизацию [2]. Так же в нем отсутствует необходимость написания большого количества лишнего кода. Kotlin поддерживает функциональное программирование, что сильно выделяет его на фоне остальных языков [3].

2.1.4 Вывод

Исходя из всех вариантов, наиболее удачным выбором языка программирования для разработки приложения будет Kotlin. Благодаря возможностям и функциям, которые можно реализовать, написав приложение именно на нем, получится создать программу, которая будет удовлетворять все поставленные задачи.

2.2 Выбор среды разработки

В настоящее время существует несколько вариантов среды разработки ПО под ОС Android на языке программирования Kotlin. Наиболее распространенными являются:

- Android Studio;
- Eclipse.

Рассмотрим их подробнее.

2.2.3 Android Studio

Среди разработчиков мобильных приложений на языке Kotlin наиболее распространена IDE Android Studio, разработанная компания Google. Ее можно установить на все распространенные операционные системы (Windows, MacOS, Linux). Android Studio обладает эмулятором, что позволяет писать программы даже если нет возможности запустить ее на реальном телефоне с ОС Android. Также присутствует интеграция с Git, которая дает много возможностей разработчикам, и функция завершения кода. IDE Поддерживает работу не только с Kotlin, так же можно писать программы на C/C++, Java, есть

поддержка XML и других форматов [4]. На рисунках 5 и 6 представлены логотип и пример рабочего окна в Android Studio.



Рисунок 5 - Текущий логотип Android Studio

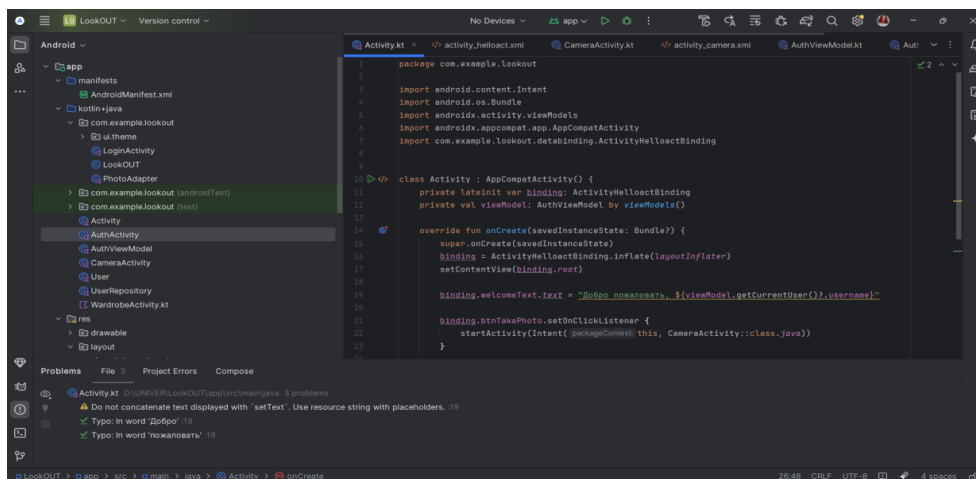


Рисунок 6 - Рабочее окно Android Studio

2.2.4 Eclipse

Эта IDE изначально была инструментом для Java-разработчиков, однако со временем в нее добавили много новых функций. Так же распространена для разработки мобильных приложений. Поддерживает многие языки программирования, например, C/C++, Java, Python, Perl, JavaScript. К достоинствам Eclipse можно отнести наличие облачной интерпретации, с помощью которой предоставляется возможность разработки приложений из браузера. На рисунках 7 и 8 представлены логотип и окно Eclipse.



Рисунок 7 - Логотип Eclipse

					ПГУ1.090301.20.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

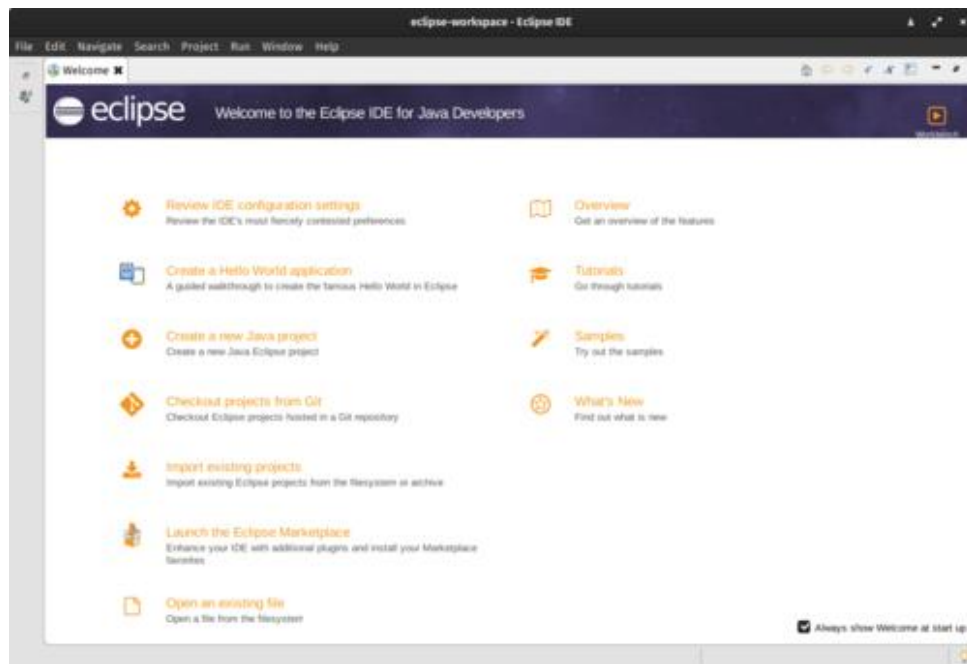


Рисунок 8 - Окно Eclipse

2.2.5 Вывод

Проанализировав возможные варианты среды разработки своего приложения, было принято решение остановить выбор на Android Studio, так как эта программа наиболее доступна и проста в использовании, а также поддерживает все необходимые функции.

3 Архитектура приложения

Архитектура программного обеспечения (ПО) — это фундаментальная структура приложения, определяющая его компоненты, их взаимодействие и основные принципы разработки. Учитывая все особенности программы, необходимо выбрать наиболее подходящий паттерн, чтобы создать качественное приложение. Это и является главной целью Архитектуры ПО, улучшить основные компоненты и характеристики разрабатываемой программы.

В разработке Android-приложений сегодня широко применяется архитектурная модель MVVM. Главное ее свойство – она разделяет проект на три важных части: модель, представление, модель-представление

Модель отвечает за взаимодействие с базой данных и обработку бизнес-логики, обеспечивая работу с данными и выполнение основных функций приложения. Представление занимается отображением пользовательского интерфейса, получая информацию от модели представления и предоставляя ее пользователю в удобном формате. Модель представления выполняет роль связующего звена между моделью и представлением, управляет состоянием интерфейса, обрабатывает пользовательские действия и подготавливает данные для корректного отображения.

В среде разработки Android Studio предусмотрен набор инструментов Android Architecture Components, который значительно облегчает процесс проектирования, тестирования и сопровождения приложений. Именно этот набор библиотек используется в данном проекте, позволяя разработчикам создавать структурированные и надежные Android-приложения. Такой подход способствует повышению производительности, улучшению удобства работы с кодом и обеспечению стабильности приложения в процессе его эксплуатации.

На рисунке 9 представлена схема взаимодействия модулей архитектуры готового приложения.

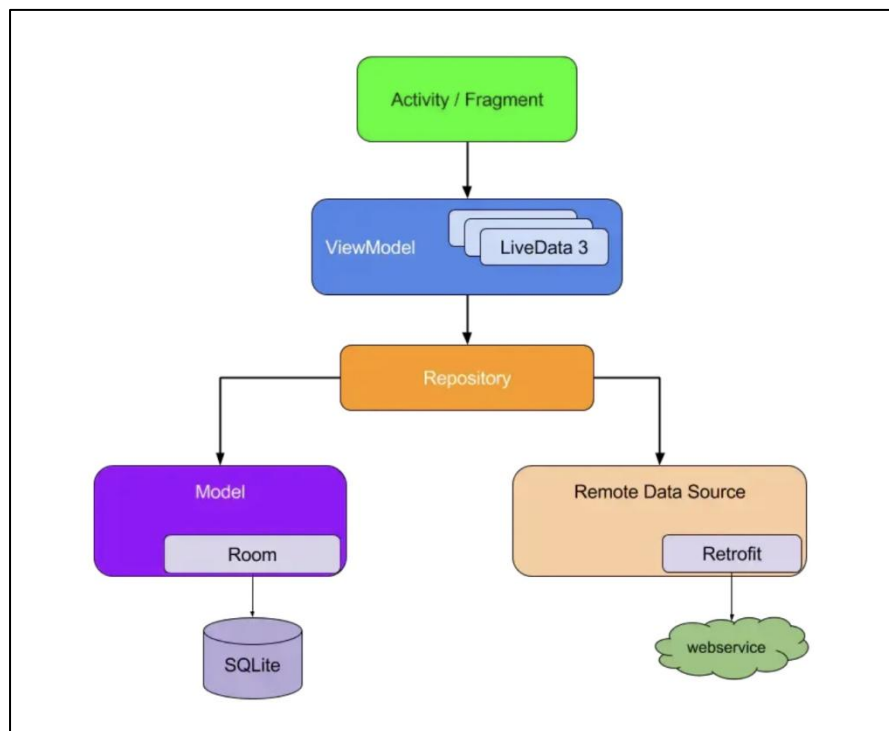


Рисунок 9 - Схема взаимодействия модулей архитектуры готового приложения

На слое View архитектуры MVVM в Android приложениях находятся классы Activity и Fragment. Именно они создают пользовательский интерфейс. Также они играют немаловажную роль в отображении данных. Сами данные поступают в эти классы от классов, находящихся на слое ViewModel. Происходит эта с помощью LiveData.

Жизненный цикл классов, находящихся во View, зависят от ViewModel, так как компоненты именно этого слоя отвечают за управление его состоянием. Данные, хранящиеся в Repository, находятся под контролем LiveData., что так же обеспечивает их целостность и будущее представление.

Приложение включает в себя такие ключевые Activity, как LoginActivity (предназначена для аутентификации пользователя), MainActivity (главная активность, обеспечивающая навигацию между остальными экранами и фрагментами), TripActivity (просмотр погоды) и GalleryActivity (отображение фотографий вещей).

Repository играет роль посредника между ViewModel и базой данных (Room) или (Retrofit). Он выполняет две главные функции:

- определяет, откуда получить данные (из локальной базы или удаленного сервера);
- преобразует данные в удобный формат и передает их ViewModel.

Главное преимущество использования Repository – разделение логики получения данных. Именно благодаря этому ViewModel нет необходимости загружать данные напрямую, так как есть посредник в виде этого класса. Эта структура помогает делать код более гибким и модульным

Слой Model взаимодействует с локальной базой данных и хранение информации. В структуре Android приложений для работы с базами данных нередко выбирается библиотека Room.

Room - «обертка» над SQLite, которая является встроенной базой данных Android (располагается локально на устройстве). Она упрощает работу с БД, что освобождает разработчиков от необходимости писать сложные SQL-запросы вручную. Библиотека Room использует DAO (Data Access Object), что делает обращение к данным еще проще, и автоматически преобразует данные SQL в объекты языка программирования Kotlin.

ViewModel – слой, обеспечивающий обработку данных, поступающих от Repository по запросу от Activity или Fragment. Он так же управляет состоянием UI и реализует взаимодействие между View и Repository. Этот слой не привязан к жизненному циклу View и не удаляется, если сменить конфигурацию (например, повернуть экран телефона). ViewModel содержит LiveData, с помощью которой можно наблюдать за изменением данных. Если из Repository поступает новая информация, LiveData сразу сообщает об этом слой View, после чего происходит обновление UI. Из этого следует, что ViewModel хранит и управляет данными, что позволяет приложению быть более функциональным [5].

На рисунке 10 представлена UML-диаграмма последовательности.

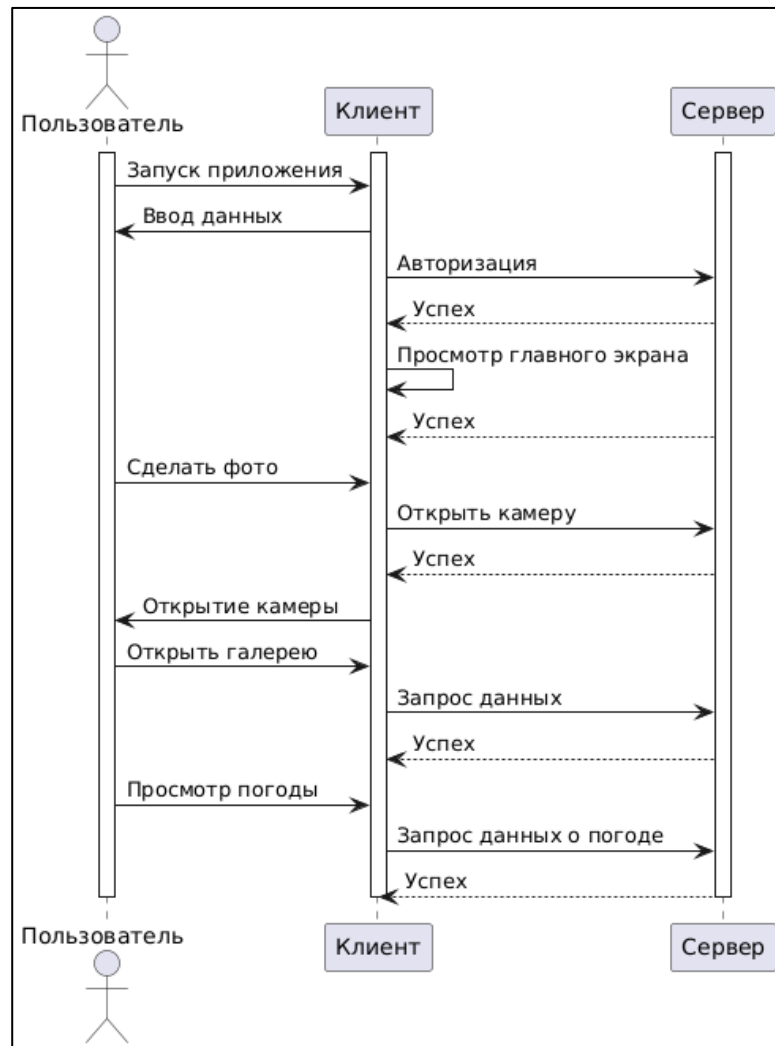


Рисунок 10 - UML-диаграмма последовательности

На рисунке 11 представлена UML-диаграмма классов.

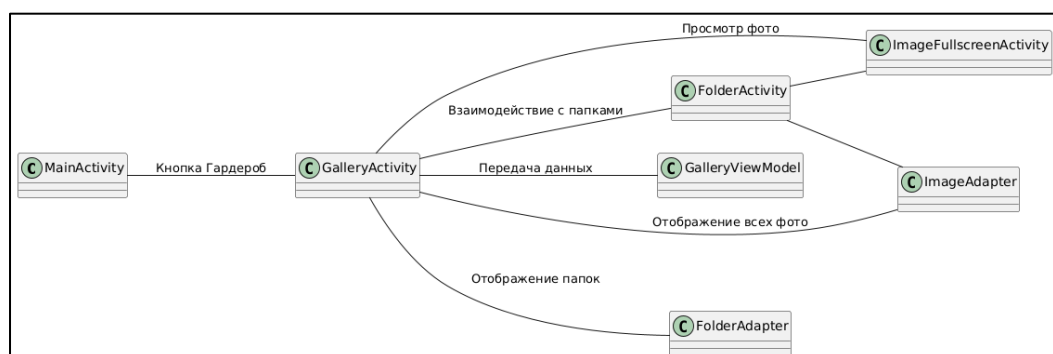


Рисунок 11 – UML-диаграмма компонентов

4 Реализация приложения

4.1 Описание функционала

В данном приложении для пользования полным функционалом клиентам необходимо зарегистрироваться или авторизоваться, если аккаунт уже создан. После выполнения одной из этих процедур станут доступны следующие действия:

- добавление фотографий одежды;
- создание образов;
- возможность редактирования образов;
- удаление фото;
- просмотр погоды.

Приложение предоставляет пользователю возможность воссоздать свой гардероб в одном месте и планировать, что положить в чемодан для предстоящих поездок. Для этого предлагается весь необходимый функционал. Программа позволяет пользователю быстро создать образы для необходимых случаев, а также отсортировать свои вещи по определенным критериям (брюки, рубашки и т.д.). Также, благодаря приложению, отпадает необходимость смотреть в интернете погоду в нужном городе, это можно сделать в самом приложении. Пользователь может спланировать и организовать свой гардероб для поездки без лишних и ненужных действий.

После запуска пользователю предлагается ввести логин и пароль, если он уже авторизован в системе. При успешном входе в учетную запись пользователю открывается главный экран приложения, который содержит все основные функции. В случае, если пользователь еще не авторизован в системе, ему доступна только функция регистрации. Без создания учетной записи все возможности приложения будут недоступны.

На рисунке 12 представлен главный экран приложения.

					<i>ПГУ1.090301.20.001 ПЗ</i>	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

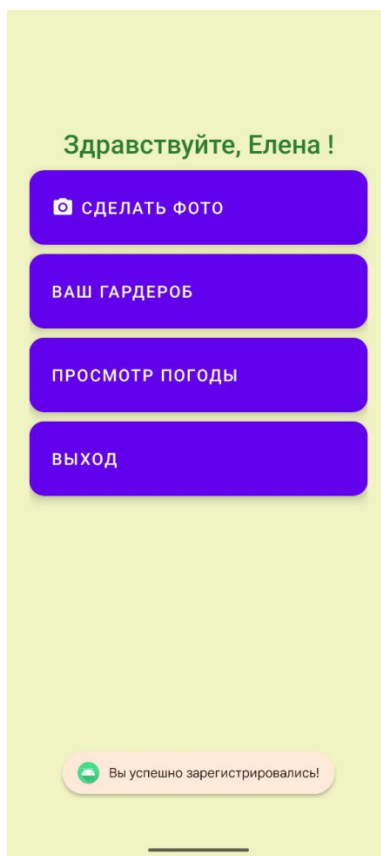


Рисунок 12 – Главный экран приложения

Если пользователь успешно вошел в аккаунт, он может использовать весь функционал приложения: добавить фото одежды, просмотреть все имеющиеся фотографии, добавить нов для образов или сортировки, удалить фото вещи, посмотреть погоду для предстоящей поездки. Все возможности интуитивно понятны.

Если пользователю необходимо сменить учетную запись, он может выйти из текущей по кнопке на главном экране.

4.2 Описание работы приложения

Пользователь будет взаимодействовать исключительно с клиентской частью приложения. Чтобы программное обеспечение работало на телефоне пользователя корректно, необходимо, чтобы на нем стояла минимально требуемая версия Android. Также требуется достаточного количества свободной памяти на устройстве. Если эти условия соблюдены, пользователю

необходимо войти в учетную запись или создать новую, то есть зарегистрироваться. После этого клиент проверит, совпадают ли данные с теми, что хранятся на сервере. Если авторизация будет успешной, то пользователю откроется страница главного экрана приложения.

При нажатии выборе сделать фото, клиент отправляет на сервер запрос с открытием камеры и, если разрешение предоставлено, открывается системная камера телефона.

При открытии «Гардероба» или одного из каталогов, клиент запрашивает у сервера нужные данные в виде фото и выгружает их.

При нажатии на «Посмотреть погоду» клиент отправляет серверу запрос с загрузкой данных о погоде, и с помощью API погоды сервер передает эти данные.

Все варианты взаимодействий пользователя и клиента, клиента и сервера представлены на рисунках 13 и 14.

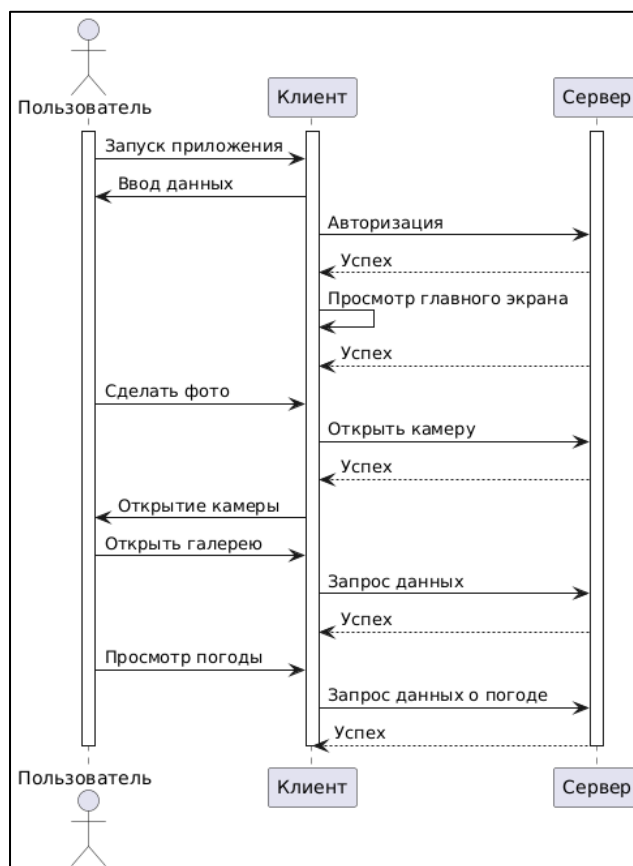


Рисунок 13 – UML-диаграмма последовательности

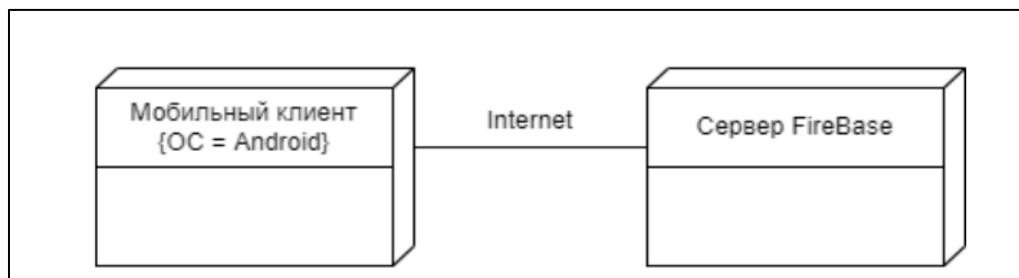


Рисунок 13 – UML-диаграмма развертывания

4.3 Описание компонентов приложения

Основные компоненты приложения: MainActivity, GalleryActivity, TripActivity, CreateFolderActivity, FolderActivity и FolderSelectionActivity.

MainActivity отображает экран, на котором видна вся информация об аккаунте пользователя, а также кнопки, по которым можно перейти на другие компоненты. При нажатии на «Добавить вещь» всплывает окно с запросом на разрешение использования приложением камеры, открывается камера устройства с возможностью делать фото. После того, как фотография сделана, она будет отображаться в GalleryActivity. При нажатии на «Гардероб» будет осуществляться переход на GalleryActivity, а кнопка «Просмотр погоды» перенаправляет на TripActivity.

Компонент GalleryActivity отображает каталоги и фотографии, сделанные пользователем в приложении. При нажатии на фото, оно отображается на весь экран, при нажатии на дополнительную кнопку в правом углу изображения всплывает контекстное меню с дополнительными возможностями: удалить или добавить в существующий каталог. При выборе «Добавить в существующий каталог» открывается новое контекстное меню со списком всех каталогов, и при выборе нужного, происходит переход на нужную FolderActivity. При нажатии на «Создать каталог» осуществляется перенаправление на CreateFolderActivity, нажатие на любой из каталогов реализует переход к FolderActivity.

CreateFolderActivity позволяет создать новый каталог для хранения фотографий. Можно ввести название нового каталога, и при нажатии «Создать каталог» осуществляется переход к FolderActivity.

Компонент FolderActivity отображает все фотографии, скопированные в выбранный каталог.

TripActivity отображает экран с двумя полями, в которых можно ввести даты начала и окончания поездки, а также выпадающий список городов. При нажатии на «Показать погоду» отображается погода в выбранном городе в указанные даты, после чего появляется кнопка «В галерею», которая перенаправляет пользователя в GalleryActivity.

GalleryViewModel осуществляет управление данными пользователей (изображения, каталоги). Это необходимо, чтобы при смене конфигурации все данные остались целы и в прежнем виде. Вдобавок к этому, данный компонент реализует загрузку данных.

На рисунке 15 представлена UML-диаграмма компонентов.

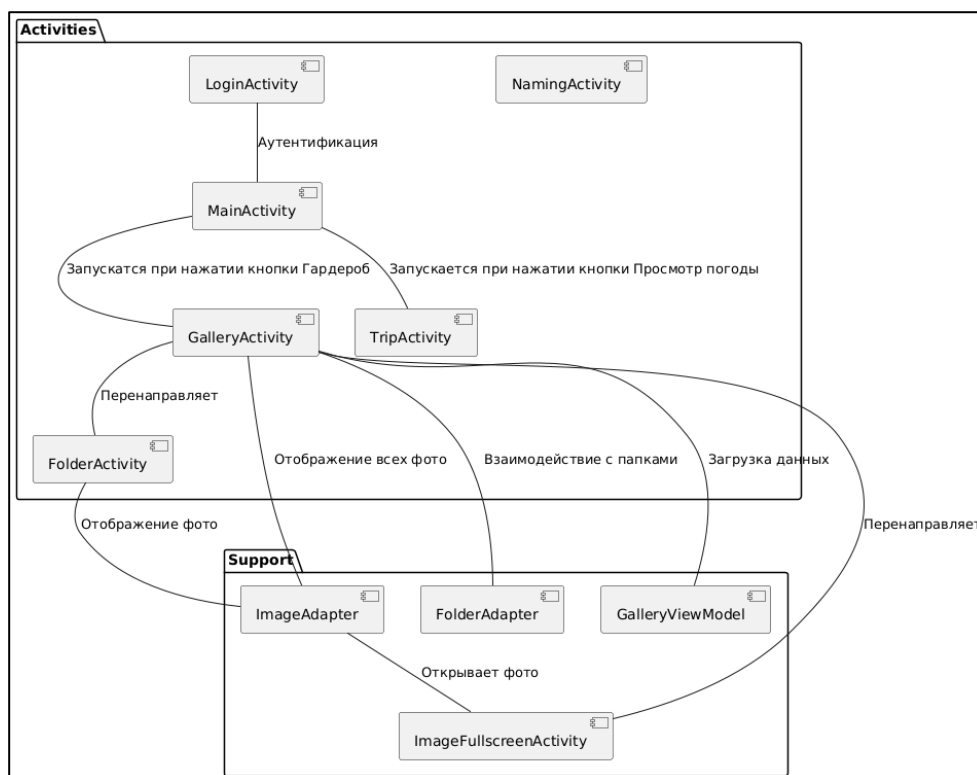


Рисунок 15 – UML-диаграмма компонентов

4.4 Описание интерфейса приложения

После запуска приложения пользователю открывается экран с авторизацией. На нем есть два поля для ввода логина и пароля, кнопки «Войти» и «Зарегистрироваться». Если авторизация прошла без ошибок, ему открывается главный экран приложения, на котором отображаются кнопки с основными вариантами использования: «Сделать фото», «Гардероб», «Просмотр погоды» и «Выход из аккаунта».

На рисунке 16 представлен экран авторизации.

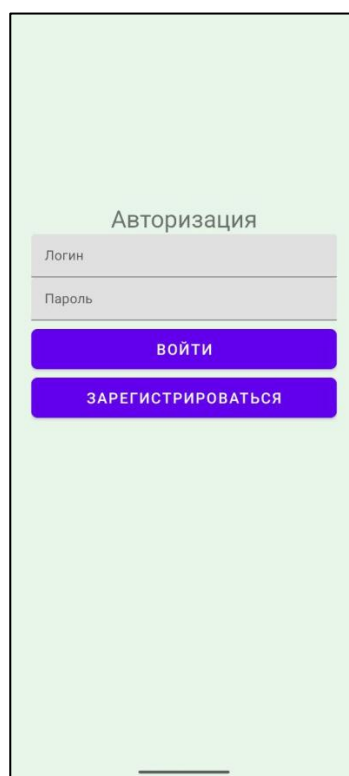


Рисунок 16 – Экран авторизации

При нажатии на кнопку «Гардероб» открывается экран со всеми фото и каталогами с образами, которые сделал пользователь. Кнопка «Просмотр погоды» перенаправляет на экран, в котором можно сделать запрос о погоде. При нажатии на кнопки «Сделать фото» и «Выйти из аккаунта» открывается камера и происходит выход из аккаунта соответственно.

На рисунке 17 представлен главный экран приложения.

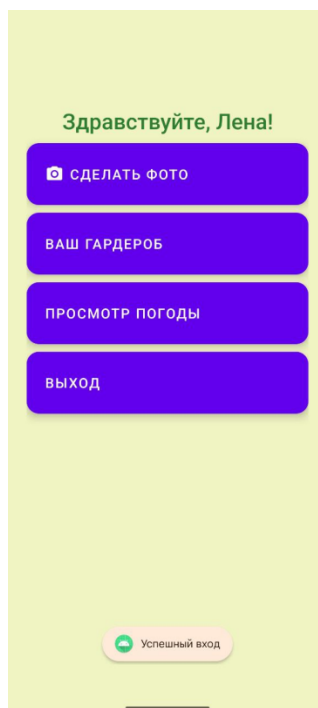


Рисунок 17 – Главный экран

На экране «Гардероб» отображаются все фотографии, сделанные пользователем, а также каталоги с образами, которые он создал. При нажатии на фотографию она открывается в полноформатном режиме. При нажатии на кнопку «Создать каталог» открывается окно, в котором можно ввести название каталога и создать ее, после чего на экране «Гардероба» появляется кнопка со ссылкой на эту каталоги.

На рисунках 18 и 19 представлены экран «Гардероб» и экран одного из каталогов соответственно.

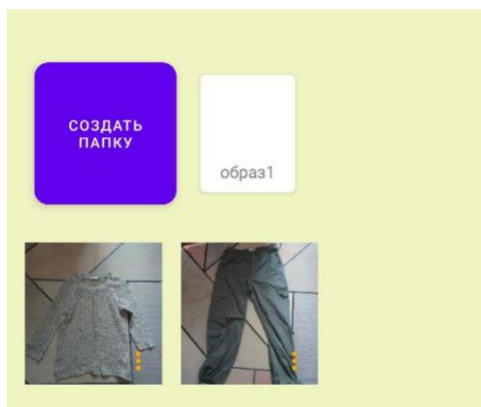


Рисунок 18 – Экран Гардероба

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.20.001 ПЗ

Лист

29

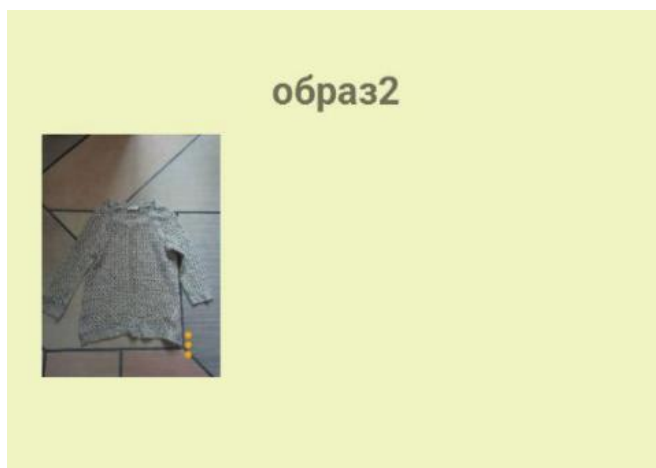


Рисунок 19 – Вид одного из каталогов

При нажатии на «Просмотр погоды» на главном экране пользователя перенаправляет на экран, в котором можно ввести даты начала и окончания поездки, выбрать город и в соответствии с введенными данными посмотреть погоду в выбранном месте.

На рисунке 20 представлен экран «Просмотр погоды»

 A screenshot of a mobile application screen titled 'Узнать погоду' (Know the weather). It features two input fields for dates: 'Начало поездки (ГГГГ-ММ-ДД)' (Start of trip) and 'Конец поездки (ГГГГ-ММ-ДД)' (End of trip). Below these is a dropdown menu currently showing 'Moscow'. At the bottom is a prominent blue button labeled 'показать погоду' (show weather). The background is a solid light yellow color.

Рисунок 20 – Экран просмотра погоды

4.5 Тестирование приложения

Тестирование является важным этапом при разработке приложения. Оно может помочь найти все неточности в работе приложения, ошибки и понять, удобно ли пользоваться приложением, или нужно поменять интерфейс. Все это делается для того, чтобы на выходе был максимально удобный и рабочий продукт.

Чтобы найти все возможные ошибки необходимо протестировать:

- регистрацию;
- авторизацию;
- возможность делать фото;
- добавление собственного каталога.
- просмотр погоды.

4.5.1 Регистрация

При открытии приложения отображается окно авторизации, представлено на рисунке 21. Если пользователь не зарегистрирован, ему предлагается сделать это по кнопке «Зарегистрироваться», после чего откроется новый экран. На нем есть три поля – для ввода логина, пароля и для повторения пароля.

Экран регистрации представлен на рисунке 22.

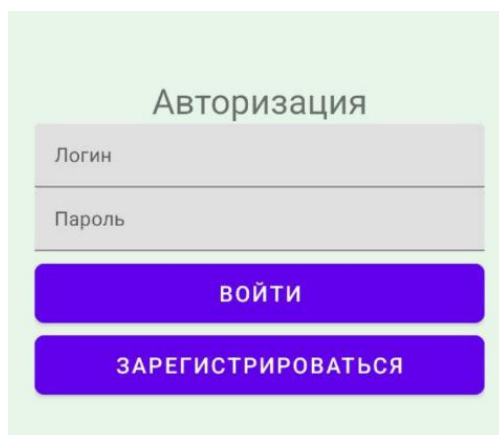


Рисунок 21– Экран авторизации

Регистрация

Имя

Email

Пароль

Подтверждение пароля

ЗАРЕГИСТРИРОВАТЬСЯ

Рисунок 22 – Экран регистрации

При неправильном вводе данных, например логина, или при несовпадении введенных паролей, будет показано сообщение о совершенных ошибках. На рисунках 23 и 24 представлены сообщения с ошибками при вводе данных.

Регистрация

Имя
Елена

Email
lena

Пароль

Подтверждение пароля

ЗАРЕГИСТРИРОВАТЬСЯ

lena

q w e r t y u i o p
a s d f g h j k l
z x c v b n m

?123 Ошибка: Неподходящий формат email →!

Рисунок 23 – Ошибка при вводе логина

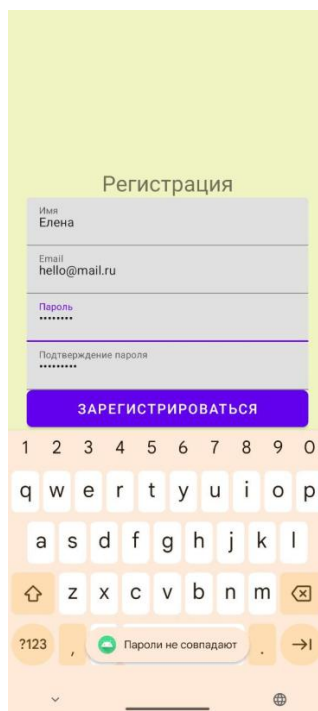


Рисунок 24 – Ошибка при вводе пароля

До подтверждения регистрации в базе данных аккаунтов был только один аккаунт, представлено на рисунке 25.

<input type="text" value="Search by email address, phone number, or user UID"/> Add user ↺ ⋮				
Identifier	Providers	Created ↓	Signed In	User UID
try@mail.ru	✉	Jun 1, 2025	Jun 1, 2025	OnZ6jE5WEBbskZ04mVyHTS...
Rows per page: 50 1 – 1 of 1				

Рисунок 25 – Старая база данных аккаунтов

Но после регистрации база данных обновилась и там стали видны все аккаунты. Обновленная база данных аккаунтов представлена на рисунке 26.

<input type="text" value="Search by email address, phone number, or user UID"/> Add user ↺ ⋮				
Identifier	Providers	Created ↓	Signed In	User UID
hello@mail.ru	✉	Jun 1, 2025	Jun 1, 2025	wSJkoyMF2iFGJJpIMOB5TJ5ih...
try@mail.ru	✉	Jun 1, 2025	Jun 1, 2025	OnZ6jE5WEBbskZ04mVyHTS...
Rows per page: 50 1 – 2 of 2				

Рисунок 26– Обновленная база данных аккаунтов

Если все данные введены верно, то пользователю открывается главный экран.

4.5.2 Авторизация

Если пользователь уже заводил учетную запись в приложении, он может ввести логин и пароль на экране авторизации. При некорректно введенных данных приложение выдаст предупреждение с сообщением об ошибке. На рисунке 27 представлена ошибка при некорректно введенных данных.

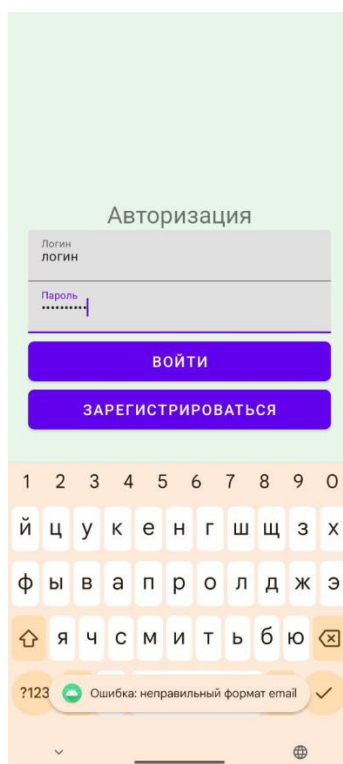


Рисунок 27 – Ошибка при вводе логина при авторизации

После прохождения авторизации пользователя перенаправит на главный экран приложения.

4.5.3 Возможность делать фото

На главном экране по кнопке «Сделать фото» предоставляется возможность добавить фотографию в свой виртуальный «Гардероб». На рисунках 28 и 29 показано, что сделанная фотография сохраняется в «Гардеробе».



Рисунок 28 – Открытие камеры



Рисунок 29 – Вид Гардероба

4.5.4 Создание каталога

При переходе в раздел «Гардероб» отображается экран со всеми фото и каталогами. При нажатии на кнопку «Создать каталог» открывается активность, в которой можно ввести название нового каталога и создать ее по кнопке «Создать».

Если не ввести название каталога в выделенное поле, то выведется сообщение о том, что необходимо его заполнить. На рисунке 30 представлен вывод сообщения о вводе названия каталога.

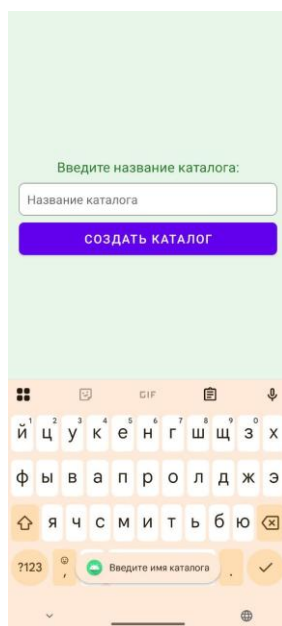


Рисунок 30 – Вывод сообщения о том, что необходимо ввести название каталога

При нажатии на кнопку «Добавить фото» можно добавить фото из «Гардероба» в созданный каталог, тем самым создав новый образ. На рисунках 31 и 32 представлено создание каталога.

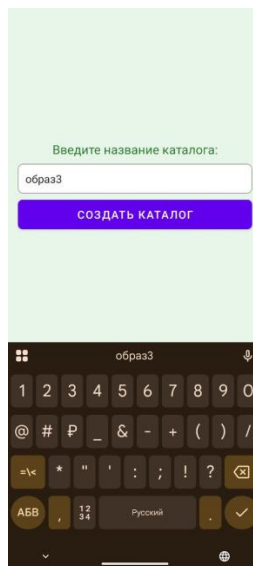


Рисунок 31 – Создание каталога



Рисунок 32 – Вид гардероба с новым каталогом

После создания каталога появляется возможность скопировать в него любое фото. Копирование фото представлено на рисунках 33, 34, 35 и 36.

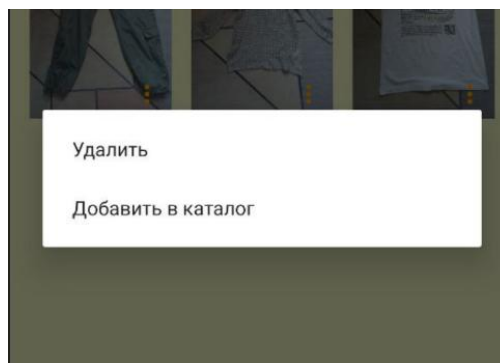


Рисунок 33 – Контекстное меню с выбором действий с изображением

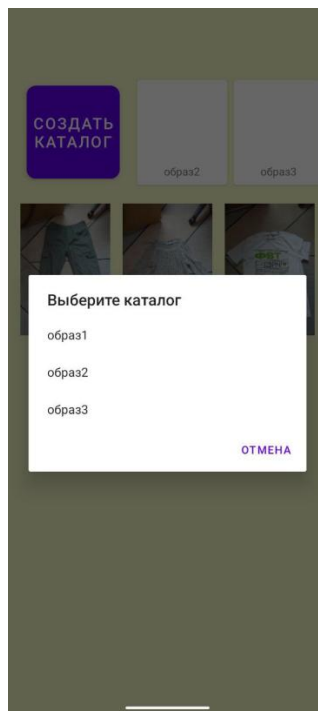


Рисунок 34 – Выбор каталога

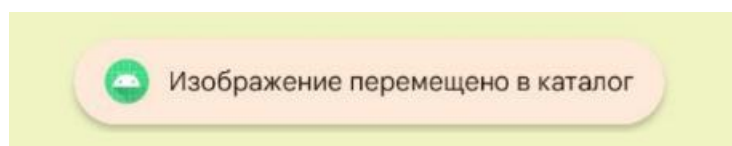


Рисунок 35 – Копирование фото в каталог

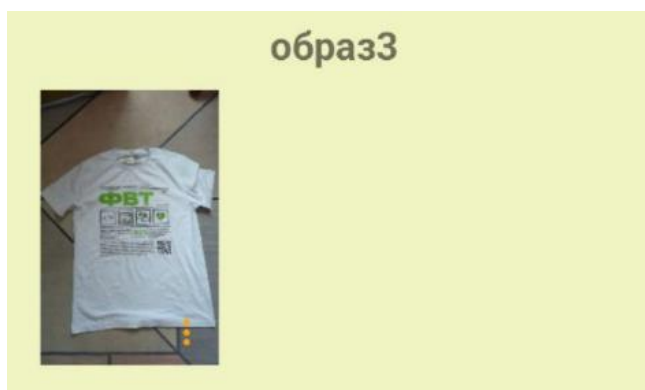


Рисунок 36 – Экран каталога с фото

4.5.5 Просмотр погоды

При нажатии на кнопку «Просмотр погоды», пользователь переходит на активность, в которой есть два поля для ввода дат – начала и окончания

поездки, выпадающий список городов и кнопка «Показать погоду». На рисунке 37 представлен экран просмотра погоды.

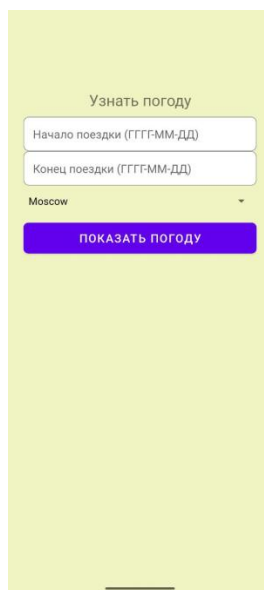


Рисунок 37 – Экран просмотра погоды

Если не заполнить все нужные поля и нажать кнопку «Показать погоду», то будет показано сообщение о необходимости заполнить все поля. На рисунке 32 представлен вывод сообщения с пустыми полями.

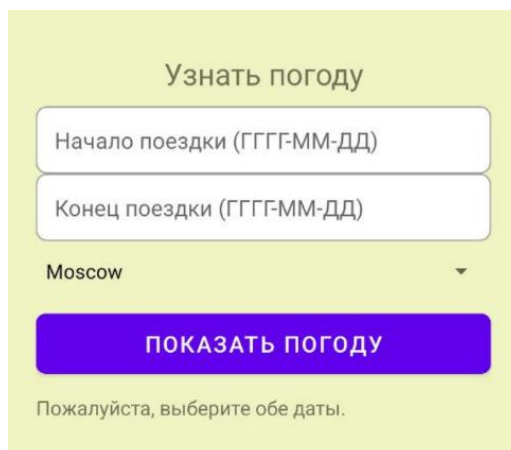


Рисунок 38 – Вывод сообщения о необходимости заполнить все
нужные поля

После ввода всех данных и нажатия на кнопку «Показать погоду», приложение показывает погоду на выбранные даты в нужном городе, а так же

появляется кнопка «В Гардероб», нажав на которую пользователь перейдет в раздел «Гардероб». На рисунке 39 представлен экран выбора даты поездки.

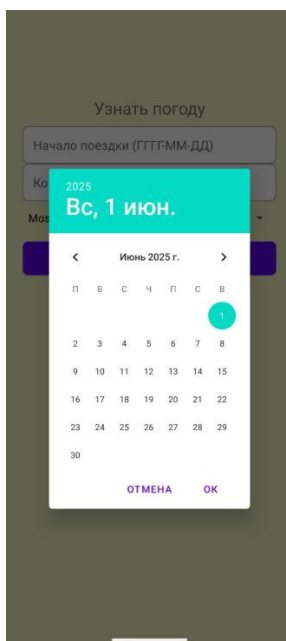


Рисунок 39 – Экран выбора даты

После заполнения всех необходимых данных и нажатия на кнопку «Показать погоду», появляются данные с погодой на выбранные даты и город. Экран с загруженными данными представлен на рисунке 40.

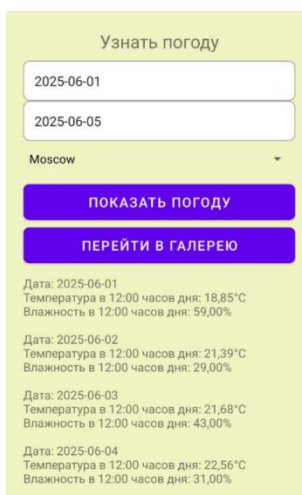


Рисунок 40 – Экран с обновленными данными

4.5.6 Итоги тестирования

Результаты тестирования представлены в таблице 1.

					ПГУ1.090301.20.001 ПЗ	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1 – Результаты тестирования приложение

Суть теста	Тестируемые данные	Ожидаемый результат	Полученный результат
Ввод недопустимого значения в поле логина	lena	Вывод сообщения с ошибкой	Совпадает с ожидаемым
Ввод несовпадающих паролей	11111111 11111112	Вывод сообщения о том, что пароли не совпадают	Совпадает с ожидаемым
Создание нового каталога	Поле с названием каталога не заполнено	Вывод сообщения о том, что необходимо ввести название каталога	Совпадает с ожидаемым
Добавление изображения в каталог	Выбор необходимых полей в контекстных меню	Вывод сообщения о том, что изображение скопировано в каталог	Совпадает с ожидаемым
Просмотр погоды	Оставить поля с датами незаполненными	Вывод сообщения о том, что необходимо заполнить все поля	Совпадает с ожидаемым

В ходе проведения тестирования было выяснено, что ПО работает верно и соответствует всем требованиям поставленного задания.

4.6 Вывод

В результате проектирования и разработки приложения были реализованы поставленные цели и задачи. Был проведен анализ и описание компонентов программы, и схема взаимодействия клиента и сервера.

Проведенное тестирование показало, что в программе учтены типовые ошибки, которые может допустить пользователь, пользуясь данной программой, и описаны обработчики данных ошибок. Благодаря проделанной работе приложение работает корректно.

5 Безопасность жизнедеятельности

Для обеспечения безопасности пользователей и разработчиков, которым в силу своих работ необходимо проводить много времени за компьютером, были описаны правила по нормам безопасности за персональными компьютерами и мобильными устройствами. Если не соблюдать технику безопасности при работе с ПК или мобильным устройством, есть вероятность получения травм.

5.1 Правила и техника безопасности при работе с ПК

При работе за компьютером есть факторы, которые могут негативно влиять на человека, а именно:

- порча осанки из-за неудобной позы;
- инфракрасное и электромагнитное излучения, в некоторых случаях к ним можно добавить рентгеновское;
- повышенный уровень шума, создаваемого рабочей техникой;
- недостаточное освещение рабочего места.

Чтобы минимизировать вероятность сильного воздействия факторов на рабочем месте необходимо соблюдать некоторые правила.

Для комфортной работы и сохранения зрения нужно обеспечить хорошую освещенность рабочего помещения. Важно, чтобы источник света не мешал работе и нахождению в помещении. Необходимо учитывать и естественное, и искусственное освещение и контролировать его яркость.

Так же нужно учитывать само рабочее место.

Необходимо следить за осанкой во избежание проблем со спиной (сколиоза или грыж). Для этого нужно соблюдать некоторые нормы:

- расстояние от лица до экрана ПК должно быть равно вытянутой руке;
- наклон монитора должен составлять около 30 градусов;
- необходимо сидеть на правильном эргономичном стуле, высоту которого можно регулировать.

В целях достижения наивысшей безопасности необходимо следить, чтобы рабочий стол не был заставлен ненужными вещами во избежание поломок. Также необходимо следить за самой техникой, не оставлять ее включенной без присмотра и следить за ее исправным состоянием. Эти меры безопасности применяются чтобы избежать воздействия сильного электромагнитного излучения [6].

Ко всему этому рекомендуется раз в полтора часа делать зарядку для глаз, чтобы снизить нагрузку на них от воздействия техники.

5.2 Правила безопасности при работе со смартфоном

Так как разработанное приложение предназначено для работы на телефонах с ОС Android, необходимо учитывать возможные риски обращения с мобильным устройством.

Не рекомендуется пытаться снять заводские части телефона или каким-либо образом стараться изменить в нем что-то самостоятельно. Не следует ронять устройство или стараться его согнуть. Рекомендуется держать телефон дальше от воды и огня.

При неправильной зарядке смартфона возможен риск пожара, поэтому всегда необходимо следить за оригинальностью зарядного устройства.

Так как мобильное устройство может выделять радиоизлучение, не рекомендуется хранить телефон в сумке или кармане брюк, чтобы излучение не воздействовало на жизненно важные органы. В связи с этим, при телефонном разговоре нужно изменять положение устройства в руке относительно уха и головы. Хотя и на данный момент еще не было произведено подтвержденных исследований на эту тему, во избежание пагубного влияния радиоизлучения рекомендуется следовать данным правилам [7].

Заключение

В ходе выполнения выпускной квалификационной работы разработано мобильное приложение по формированию гардероба путешественника. В процессе реализации были применены современные технологии разработки Android-приложений: среда разработки Android Studio, язык программирования Kotlin, облачная СУБД Firebase.

В ВКР реализованы следующий функционал: регистрация и авторизация пользователей, управление медиафайлами с возможностью сохранения и просмотра созданных изображений, создание каталогов, возможность копирования фото в каталоги, возможность мониторинга погоды по дате и локации и предоставления пользователю права выбора создания варианта гардероба для поездки.

Авторизация пользователей осуществляется при помощи Firebase Authentication. Для мониторинга погоды в приложении применяется привязка к сервису OpenWeatherApi.

В качестве дальнейшего развития приложения рассматривается добавление функции с отслеживанием местоположения и обработки данных погодных условий с помощью нейронных сетей, для автоматического формирования гардероба будущей поездки.

Результаты тестирования показали, что приложение соответствует поставленным задачам и его минимальный функционал поможет в формировании гардероба путешественника.

Список использованных источников

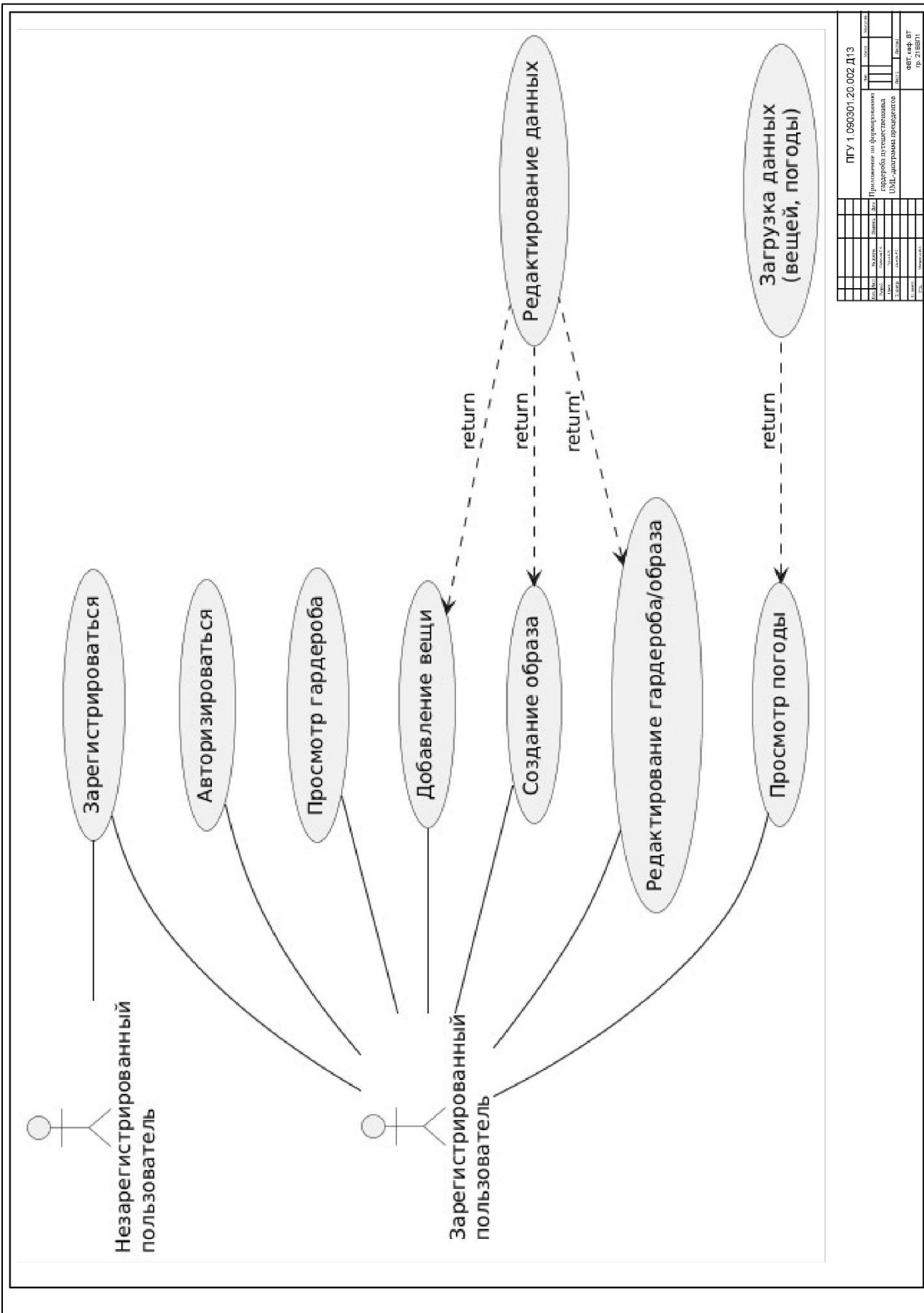
1. Статья Android Usage Statistics 2025: Devices & Market Share. [Электронный ресурс] / Режим доступа: <https://www.demandsage.com/android-statistics/> свободный (дата обращения: 30.05.2025).
2. Документация по Kotlin. [Электронный ресурс] / Режим доступа: <https://kotlinlang.org/docs/getting-started.html> свободный (дата обращения 30.05.2025).
3. Айгнер С., Елизаров Р., Исакова С., Жемеров Д. Kotlin в действии. 2-е изд. – СПб.: Питер, 2025. – 440 с. (дата обращения 30.05.2025).
4. Документация Android Studio. [Электронный ресурс] / Режим доступа: [https:// developer.android.com/studio](https://developer.android.com/studio) свободный (дата обращения 30.05.2025).
5. Статья Guide to app architecture. [Электронный ресурс] / Режим доступа: <https://developer.android.com/topic/architecture> свободный (дата обращения 31.05.2025).
6. Статья Основные правила безопасности при работе с компьютером: [Электронный ресурс] / Режим доступа: https://sysobr.ru/shkoly/shkola12/bezopasnost/Pravila_TV_pri_rabote_na_komputere.pdf свободный (дата обращения: 31.05.2025).
7. Статья Гид по максимально безопасному использованию смартфона для детей (и взрослых): [Электронный ресурс] / Режим доступа: <https://beeline.ru/articles/gid-po-maksimalno-bezopasnomu-ispolzovaniyu-smartfona-dlya-detey-i-vzroslykh/> свободный (дата обращения: 31.05.2025).

Приложение А
(обязательное)

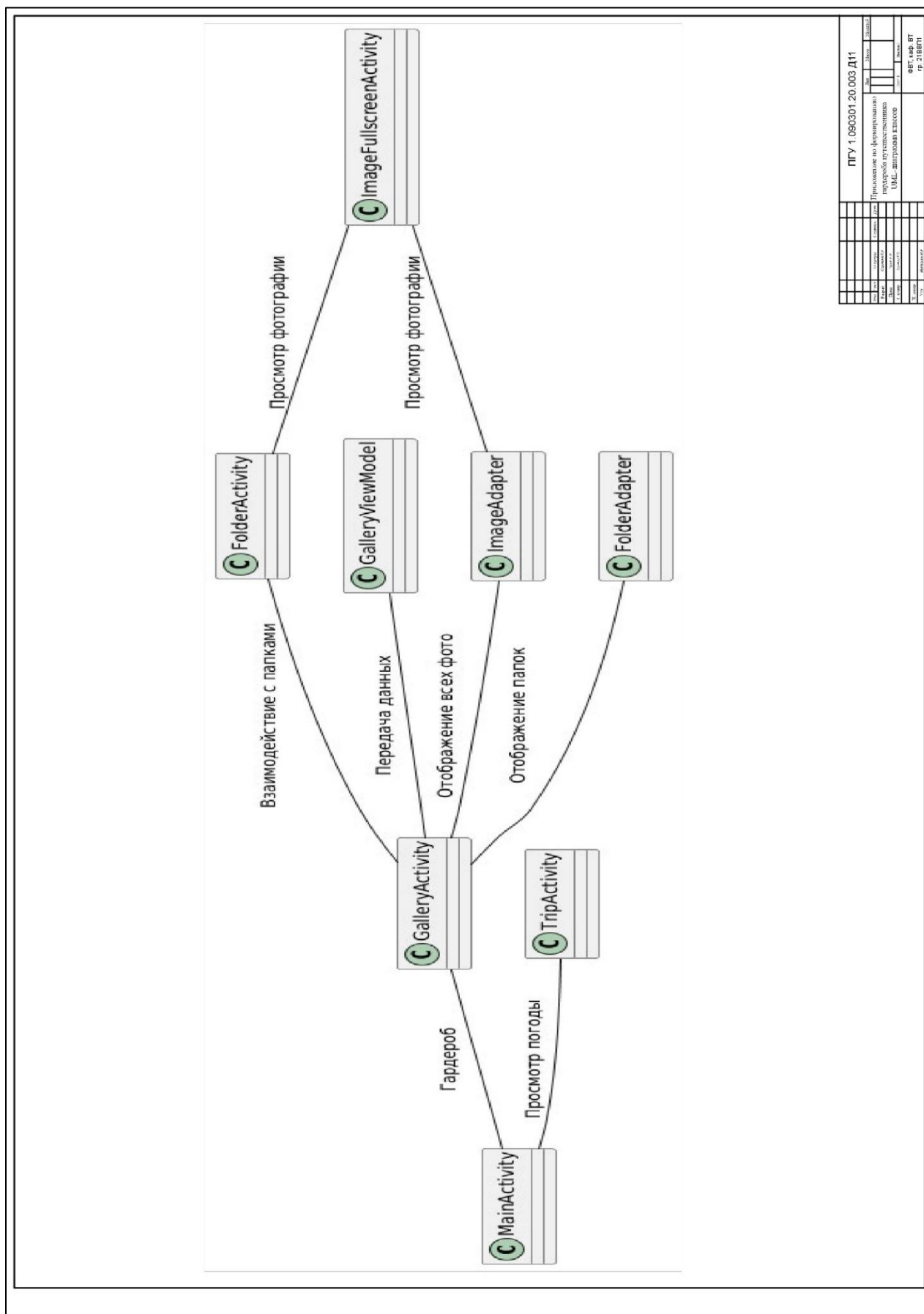
Приложение по формированию гардероба путешественника
UML-диаграммы

					<i>ПГУ1.090301.20.001 ПЗ</i>	Лист
						46
Изм.	Лист	№ докум.	Подпись	Дата		

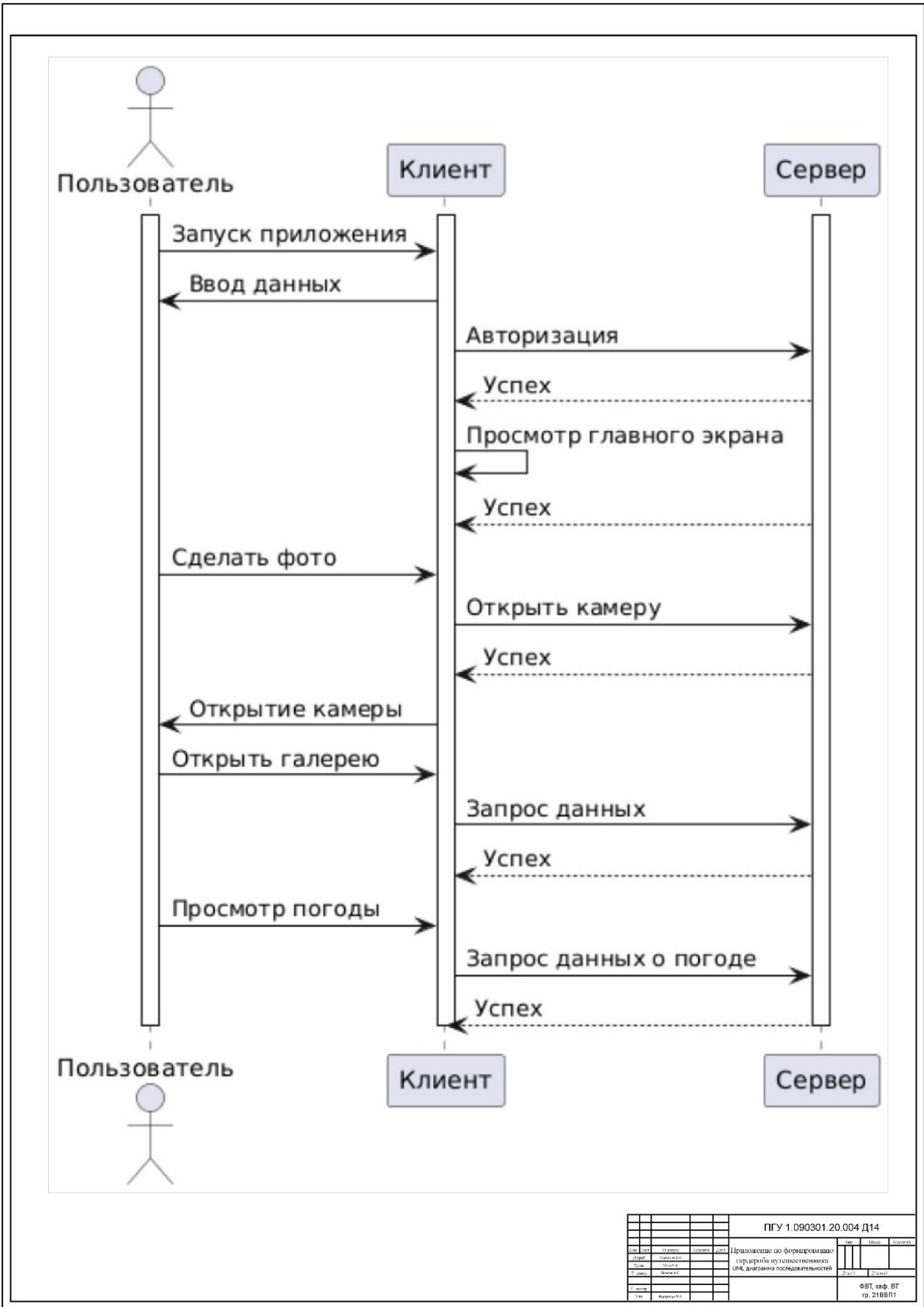
Приложение А.1 – UML-диаграмма прецедентов



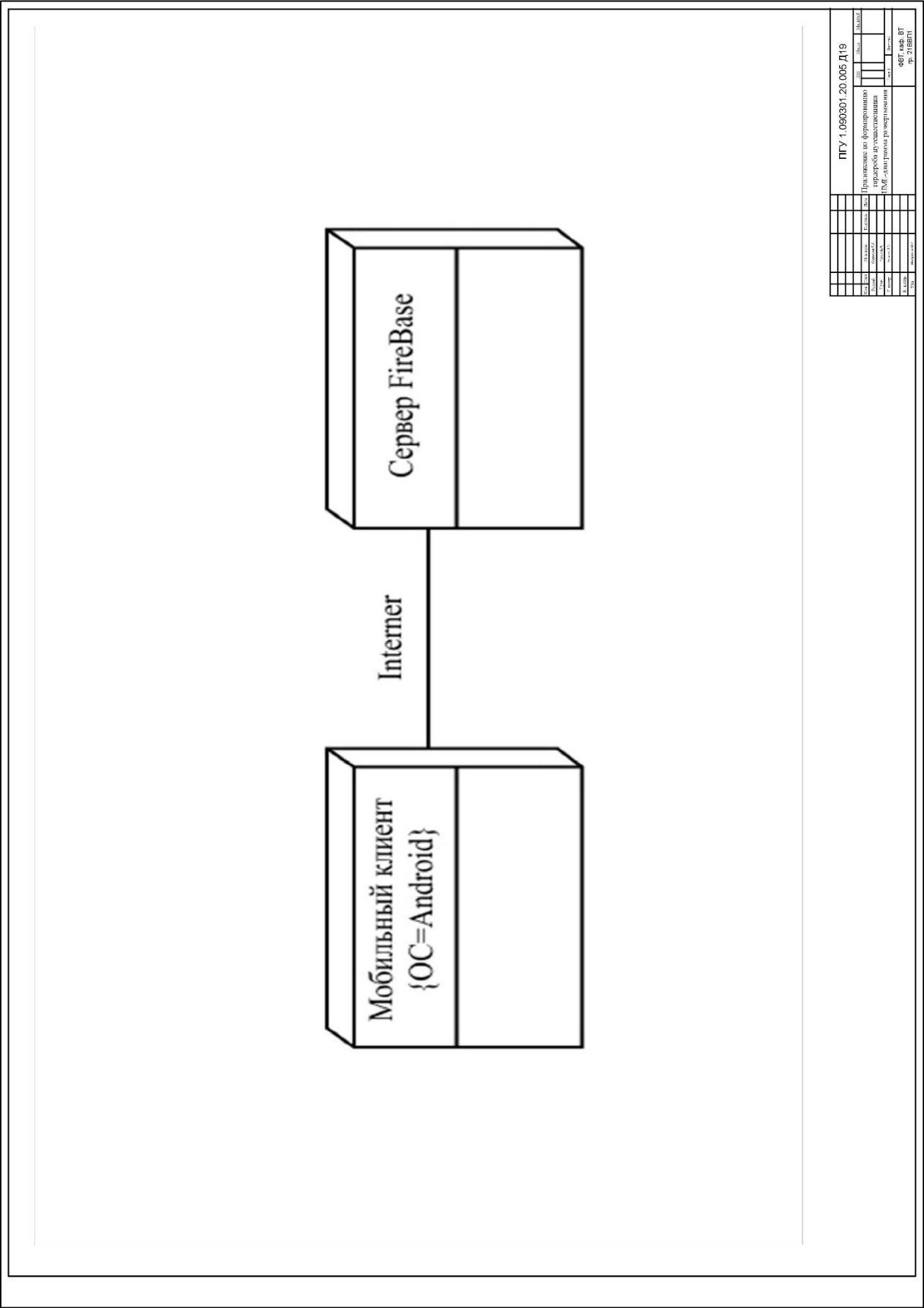
Приложение А.2 – UML-диаграмма классов



Приложение А.3 – UML-диаграмма последовательностей



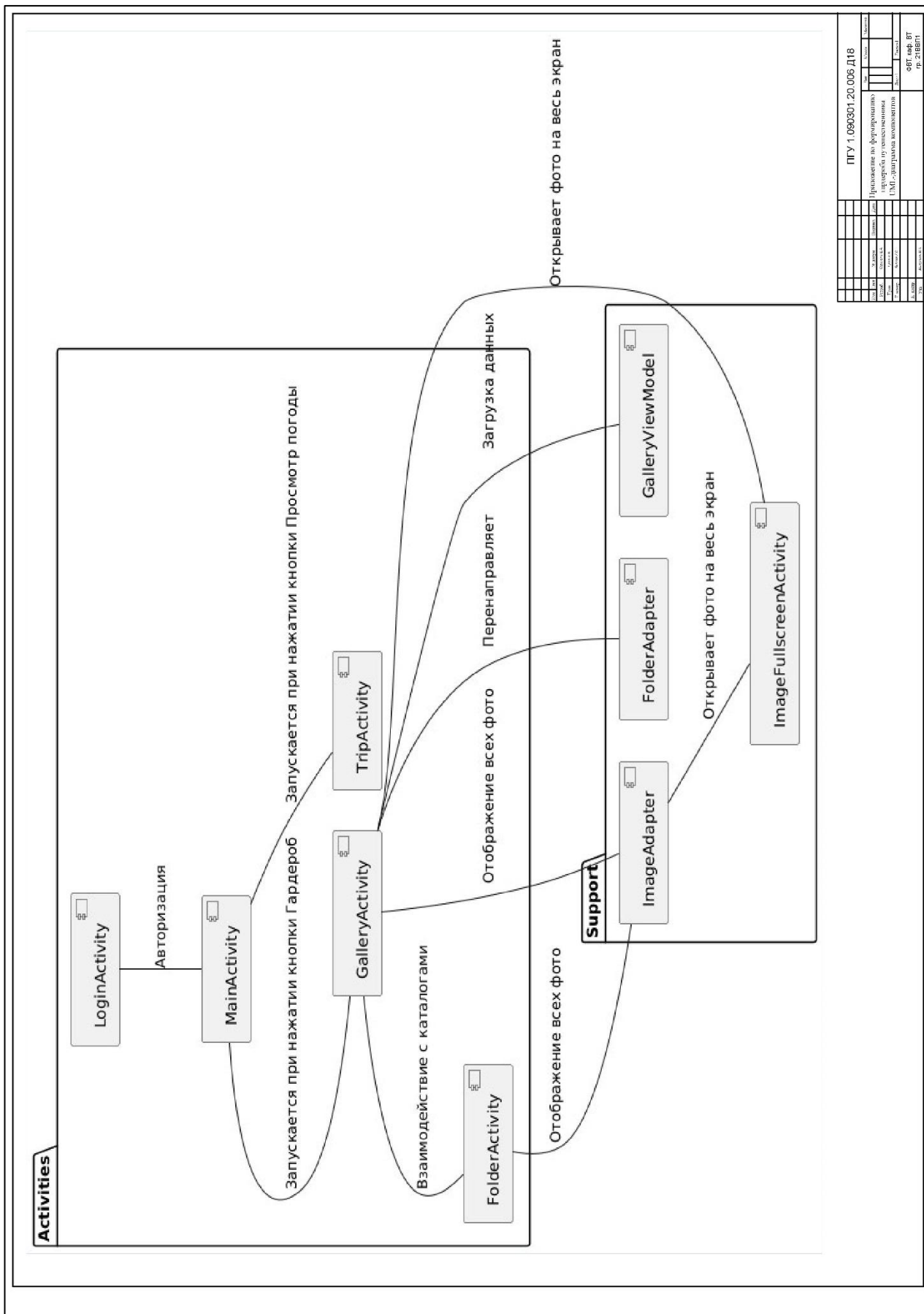
Приложение А.4 – UML-диаграмма развертывания



ПГУ 1.090301.20.005 Д19									
Приложение по формированию									
прибора к устройству									
Публ. диаграмма развертывания									
487, мф. ВГ									
пр. 21.08.21									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.5 – UML-диаграмма компонентов



Приложение Б

(обязательное)

Приложение по формированию гардероба путешественника

Листинг программы

					<i>ПГУ1.090301.20.001 ПЗ</i>	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

Файл MainActivity

```
package com.example.wardrobe.main

import android.Manifest
import android.app.Activity
import android.content.ActivityNotFoundException
import android.content.Context
import android.content.Intent
import android.content.pm.PackageManager
import android.graphics.Bitmap
import android.net.Uri
import android.os.Build
import android.os.Bundle
import android.os.Environment
import android.provider.MediaStore
import android.util.Log
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.example.wardrobe.gallery.GalleryActivity
import com.example.wardrobe.login.LoginActivity
import com.example.wardrobe.trips.TripActivity
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.auth.ktx.auth
import com.google.firebase.ktx.Firebase
import dagger.hilt.android.AndroidEntryPoint
import java.io.IOException
import java.io.File
import java.io.FileOutputStream
import com.example.wardrobe.R

@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth
    private lateinit var greetingTextView: TextView // Add this
    private lateinit var logoutButton: Button
    private lateinit var takePictureButton: Button
    private lateinit var viewGalleryButton: Button
    private lateinit var viewTripButton: Button
    // Remove Firebase dependencies (db, storage)
```

```

private val PERMISSION_REQUEST_CODE = 123
private val REQUEST_IMAGE_CAPTURE = 1
private val TAG = "MainActivity"

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    auth = Firebase.auth // Keep authentication

    // Проверяем, авторизован ли пользователь
    if (auth.currentUser == null) {
        val intent = Intent(this, LoginActivity::class.java)
        startActivity(intent)
        finish()
        return
    }

    setContentView(R.layout.activity_main)

    greetingTextView = findViewById(R.id.greetingTextView)
// Add this
    logoutButton = findViewById(R.id.logoutButton)
    takePictureButton = findViewById(R.id.takePictureButton)
    viewGalleryButton = findViewById(R.id.viewGalleryButton)
    viewTripButton = findViewById(R.id.viewTripButton)

    // Get and display user name
    val user = auth.currentUser
    user?.let {
        val name = it.displayName
        greetingTextView.text = "Здравствуйете, $name!"
    }

    takePictureButton.setOnClickListener {
        if (checkCameraPermission()) {
            dispatchTakePictureIntent()
        } else {
            requestCameraPermission()
        }
    }

    viewGalleryButton.setOnClickListener {
        val intent = Intent(this,
GalleryActivity::class.java)
        startActivity(intent)
    }

```

```

    }

    viewTripButton.setOnClickListener {
        val intent = Intent(this, TripActivity::class.java)
        startActivity(intent)
    }

    logoutButton.setOnClickListener {
        auth.signOut()
        val intent = Intent(this, LoginActivity::class.java)
        startActivity(intent)
        finish()
    }
}

private fun isCameraAvailable(): Boolean {
    val hasCamera =
packageManager.hasSystemFeature(PackageManager.FEATURE_CAMERA_ANY)

    val cameraIntent =
Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    val hasCameraApp =
cameraIntent.resolveActivity(packageManager) != null
    return hasCamera && hasCameraApp
}

private fun checkCameraPermission(): Boolean {
    return ContextCompat.checkSelfPermission(this,
Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED
}

private fun requestCameraPermission() {
    ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.CAMERA), PERMISSION_REQUEST_CODE)
}

override fun onRequestPermissionsResult(requestCode: Int,
permissions: Array<String>, grantResults: IntArray) {
    super.onRequestPermissionsResult(requestCode,
permissions, grantResults)
    if (requestCode == PERMISSION_REQUEST_CODE) {
        if (grantResults.isNotEmpty() && grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            Toast.makeText(this, "Разрешение на камеру
получено", Toast.LENGTH_SHORT).show()
            dispatchTakePictureIntent()
        }
    }
}

```

```

        } else {
            Toast.makeText(this, "Разрешение на камеру
отклонено", Toast.LENGTH_SHORT).show()
        }
    }

    private fun dispatchTakePictureIntent() {
        if (!isCameraAvailable()) {
            Log.e(TAG, "dispatchTakePictureIntent: No camera app
found")
            showNoCameraError()
            return
        }

        val takePictureIntent =
Intent(MediaStore.ACTION_IMAGE_CAPTURE)
        try {
            Log.d(TAG, "dispatchTakePictureIntent: Launching
camera")
            startActivityForResult(takePictureIntent,
REQUEST_IMAGE_CAPTURE)
        } catch (e: ActivityNotFoundException) {
            Log.e(TAG, "dispatchTakePictureIntent: Activity not
found", e)
            showNoCameraError()
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        Log.d(TAG, "onActivityResult: requestCode=$requestCode,
resultCode=$resultCode")
        if (requestCode == REQUEST_IMAGE_CAPTURE) {
            if (resultCode == Activity.RESULT_OK) {
                val imageBitmap = data?.extras?.get("data") as?
Bitmap

                imageBitmap?.let { bitmap ->
                    val userId = auth.currentUser?.uid ?: "" //
User authentication
                    val imageFile = saveImageLocally(this,
bitmap, userId)

                    if (imageFile != null) {

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        Log.d(TAG, "Image saved locally to:
${imageFile.absolutePath}")
        startActivity(Intent(this,
GalleryActivity::class.java))
    } else {
        Toast.makeText(this, "Не удалось
сохранить изображение", Toast.LENGTH_SHORT).show()
    }
    } ?: run {
        Toast.makeText(this, "Не удалось получить
Bitmap", Toast.LENGTH_SHORT).show()
    }
    } else if (resultCode == Activity.RESULT_CANCELED) {
        Log.d(TAG, "onActivityResult: Camera cancelled")
        Toast.makeText(this, "Съемка отменена",
Toast.LENGTH_SHORT).show()
    } else {
        Log.e(TAG, "onActivityResult: Camera failed,
resultCode=$resultCode")
        Toast.makeText(this, "Ошибка при съемке",
Toast.LENGTH_SHORT).show()
    }
    }
}

private fun showNoCameraError() {
    AlertDialog.Builder(this)
        .setTitle("Ошибка камеры")
        .setMessage("На устройстве не найдено приложение
камеры. Установите приложение камеры и попробуйте снова.")
        .setPositiveButton("OK") { dialog, _ ->
dialog.dismiss() }
        .show()
}

private fun saveImageLocally(context: Context, bitmap:
Bitmap, userId: String): File? {
    val filename = "IMG_${System.currentTimeMillis()}.jpg"
    val directory =
File(context.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
, userId) // Create a user-specific directory

    if (!directory.exists()) {
        if (!directory.mkdirs()) {
            Log.e(TAG, "Error: Failed to create directory")
            return null

```

```

        }
    }

    val file = File(directory, filename)

    try {
        FileOutputStream(file).use { out ->
            if (!bitmap.compress(Bitmap.CompressFormat.JPEG,
90, out)) {
                throw IOException("Failed to compress
bitmap")
            }
        }
        return file
    } catch (e: IOException) {
        Log.e(TAG, "Error saving image locally", e)
        return null
    }
}

companion object {
    private const val TAG = "MainActivity"
    private const val REQUEST_IMAGE_CAPTURE = 1
    private const val PERMISSION_REQUEST_CODE = 123
}
}

```

Файл GalleryActivity.kt

```

package com.example.wardrobe.gallery

import android.app.Activity
import android.app.AlertDialog
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.provider.MediaStore
import android.util.Log
import android.view.ContextMenu
import android.view.MenuInflater
import android.view.MenuItem
import android.view.View
import android.widget.Button
import android.widget.EditText

```

					ПГУ1.090301.20.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

```

import android.widget.Toast
import androidx.activity.result.contract.ActivityResultContracts
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.GridLayoutManager
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.wardrobe.R
import dagger.hilt.android.AndroidEntryPoint
import java.io.File

@AndroidEntryPoint
class GalleryActivity : AppCompatActivity() {

    private lateinit var recyclerView: RecyclerView
    private lateinit var imageAdapter: ImageAdapter
    private lateinit var folderRecyclerView: RecyclerView
    private lateinit var folderAdapter: FolderAdapter
    private var selectedImageUri: Uri? = null
    private val viewModel: GalleryViewModel by viewModels()

    private val createFolderLauncher =

registerForActivityResult(ActivityResultContracts.StartActivityF
orResult()) { result ->
    if (result.resultCode == Activity.RESULT_OK) {
        val folderName =
result.data?.getStringExtra("folderName")
        folderName?.let {
            viewModel.createFolder(it) { success ->
                if (success) {
                    viewModel.loadFolders() // Refresh
folders after creation
                    Toast.makeText(this, "Папка
создана", Toast.LENGTH_SHORT).show()
                } else {
                    Toast.makeText(this, "Не удалось
создать папку", Toast.LENGTH_SHORT).show()
                }
            }
        }
    } else {
        Toast.makeText(this, "Создание папки отменено",
Toast.LENGTH_SHORT).show()
    }
}
}

```

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_gallery)

    initView()
    setupAdapters()
    observeViewModel()
    viewModel.loadData()
}

private fun initView() {
    recyclerView = findViewById(R.id.recyclerView)
    recyclerView.layoutManager = GridLayoutManager(this, 3)

    folderRecyclerView =
findViewById(R.id.folderRecyclerView)
    folderRecyclerView.layoutManager = LinearLayoutManager(
        this,
        LinearLayoutManager.HORIZONTAL,
        false
    )
}

private fun setupAdapters() {
    imageAdapter = ImageAdapter(
        emptyList(),
        { imageUri -> openImage(imageUri) },
        { imageUri ->
            selectedImageUri = imageUri
            openContextMenu(recyclerView)
        }
    )
    recyclerView.adapter = imageAdapter

    folderAdapter = FolderAdapter(this, emptyList()) {
folder ->
        openFolder(folder)
    }
    folderRecyclerView.adapter = folderAdapter
    registerForContextMenu(recyclerView)

    val createFolderButton =
findViewById<Button>(R.id.createFolderButton)
    createFolderButton.setOnClickListener {
        //showCreateFolderDialog()
    }
}

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        launchCreateFolderActivity()
    }
}

private fun observeViewModel() {
    viewModel.imageList.observe(this) { imageUri ->
        Log.d("GalleryActivity", "observeViewModel:
imageList changed, updating adapter")
        imageAdapter.imageList = imageUri
        imageAdapter.notifyDataSetChanged()
    }

    viewModel.folderList.observe(this) { folders ->
        folderAdapter.updateData(folders)
        addFolderButtons(folders)
    }

    viewModel.showToast.observe(this) { message ->
        message?.let {
            Toast.makeText(this, it,
Toast.LENGTH_SHORT).show()
            viewModel.clearToastMessage()
        }
    }
}

private fun addFolderButtons(folders: List<File>) {
    val buttonContainer =
findViewById<RecyclerView>(R.id.folderRecyclerView)
    val layoutManager = LinearLayoutManager(this,
LinearLayoutManager.HORIZONTAL, false)

    buttonContainer.layoutManager = layoutManager
    buttonContainer.adapter = FolderAdapter(this, folders) {
folder ->
        openFolder(folder)
    }
}

override fun onCreateContextMenu(
    menu: ContextMenu,
    v: View,
    menuInfo: ContextMenu.ContextMenuInfo?
) {
    super.onCreateContextMenu(menu, v, menuInfo)
    val inflater: MenuInflater = menuInflater

```

```

        inflater.inflate(R.menu.context_menu, menu)
    }

    override fun onContextItemSelected(item: MenuItem): Boolean
    {
        return when (item.itemId) {
            R.id.menu_delete -> {
                selectedImageUri?.let { uri ->
                    viewModel.deleteImage(uri.path ?: return
false)

                    return true
                }
                false
            }
            R.id.menu_add_to_folder -> {
                selectedImageUri?.let { uri ->
                    showFolderSelectionDialog(uri)
                    return true
                }
                false
            }
            else -> super.onContextItemSelected(item)
        }
    }

    private fun openImage(imageUri: Uri) {
        val intent = ImageFullscreenActivity.newIntent(this,
imageUri.toString())
        startActivity(intent)
    }

    private fun openFolder(folder: File) {
        val intent = Intent(this, FolderActivity::class.java)
        intent.putExtra(FolderActivity.EXTRA_FOLDER_PATH,
folder.absolutePath)
        startActivity(intent)
    }

    private fun showFolderSelectionDialog(imageUri: Uri) {
        viewModel.getFolderNames { folders ->
            folders?.let { folderNames ->
                AlertDialog.Builder(this)
                    .setTitle("Выберите папку")
                    .setItems(folderNames.toTypedArray()) { _,
which ->

                                val selectedFolderName =

```

```

folderNames[which]
                                // Find the File object for the selected
folder name
                                val selectedFolder =
viewModel.folderList.value?.find { it.name == selectedFolderName
}
                                if (selectedFolder != null) {
                                    Log.d("GalleryActivity", "Selected
folder: ${selectedFolder.absolutePath}")
                                    Log.d("GalleryActivity", "Selected
image URI: ${imageUri.path}")

viewModel.moveImageToFolder(imageUri.path!!,
selectedFolder.absolutePath)
                                } else {
                                    Toast.makeText(this, "Folder not
found", Toast.LENGTH_SHORT).show()
                                }
                                }
                                .setNegativeButton("Отмена", null)
                                .show()
                                } ?: run {
                                    Toast.makeText(
                                        this,
                                        "Нет доступных папок",
                                        Toast.LENGTH_SHORT
                                    ).show()
                                }
                                }

private fun showCreateFolderDialog() {
    val intent = Intent(this,
CreateFolderActivity::class.java)
    //intent.putExtra("image_path", "no_path_needed")
    createFolderLauncher.launch(intent)

}

private fun launchCreateFolderActivity() {
    val intent = Intent(this,
CreateFolderActivity::class.java)
    createFolderLauncher.launch(intent)
}

override fun onActivityResult(requestCode: Int, resultCode:
Int, data: Intent?) {

```

```

        super.onActivityResult(requestCode, resultCode, data)
        if (requestCode == 1 && resultCode ==
Activity.RESULT_OK) {
            viewModel.loadFolders() // Refresh folders after
creation
            Toast.makeText(this, "Папка создана",
Toast.LENGTH_SHORT).show()
        } else if (requestCode == 1 && resultCode ==
Activity.RESULT_CANCELED) {
            Toast.makeText(this, "Создание папки отменено",
Toast.LENGTH_SHORT).show()
        }
    }
}

```

Файл GalleryViewModel.kt

```

package com.example.wardrobe.gallery

import android.app.Application
import android.content.Context
import android.net.Uri
import android.os.Environment
import android.util.Log
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModelScope
import com.google.common.reflect.TypeToken
import com.google.firebase.auth.FirebaseAuth
import com.google.gson.Gson
//import com.google.gson.reflect.TypeToken
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import java.io.File
import javax.inject.Inject

@HiltViewModel
class GalleryViewModel @Inject constructor(
    private val app: Application,
    private val auth: FirebaseAuth
) : AndroidViewModel(app) {

    private val _imageList = MutableLiveData<List<Uri>>()

```

					ПГУ1.090301.20.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

```

val imageUrl: LiveData<List<Uri>> = _imageUrl

private val _folderList = MutableLiveData<List<File>>()
val folderList: LiveData<List<File>> = _folderList

private val _showToast = MutableLiveData<String?>()
val showToast: LiveData<String?> = _showToast

// HashMap to store image-folder relationships
private val _folderImagesMap =
MutableLiveData<MutableMap<String, MutableList<String>>>()
    val folderImagesMap: LiveData<MutableMap<String,
MutableList<String>>> = _folderImagesMap

private val sharedPreferences =
app.getSharedPreferences("folder_images", Context.MODE_PRIVATE)

init {
    loadFolderImagesMap() // Загружаем
    if (_folderImagesMap.value == null) {
        _folderImagesMap.value = mutableMapOf()
    }
    loadData()
}

fun loadData() {
    loadImages()
    loadFolders()
}

private fun loadImages() {
    viewModelScope.launch {
        try {
            val userId = auth.currentUser?.uid ?:
return@launch
            val imageUris = getAllImagesForUser(userId)
            _imageUrl.postValue(imageUris)
        } catch (e: Exception) {
            Log.e("GalleryViewModel", "Error loading
images", e)
            showToastMessage("Failed to load images:
${e.message}")
        }
    }
}

```

```

fun loadFolders() {
    viewModelScope.launch {
        try {
            val userId = auth.currentUser?.uid ?:
return@launch
            val folders = getAllFoldersForUser(userId)
            _folderList.postValue(folders)
        } catch (e: Exception) {
            Log.e("GalleryViewModel", "Error loading
folders", e)
            showToastMessage("Failed to load folders:
${e.message}")
        }
    }
}

private suspend fun getAllImagesForUser(userId: String):
List<Uri> = withContext(Dispatchers.IO) {
    val imageUris = mutableListOf<Uri>()
    val storageDir =
app.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
    val userDir = File(storageDir, userId)

    if (userDir.exists() && userDir.isDirectory) {
        val allFiles = userDir.listFiles() ?: emptyArray()
        val folderPaths =
_folderImagesMap.value?.values?.flatten() ?: emptyList()

        allFiles.forEach { file ->
            if (file.isFile && file.name.endsWith(".jpg") &&
!folderPaths.contains(file.absolutePath)) {
                imageUris.add(Uri.fromFile(file))
            }
        }
    }
    imageUris
}

private suspend fun getAllFoldersForUser(userId: String):
List<File> = withContext(Dispatchers.IO) {
    val folders = mutableListOf<File>()
    val storageDir =
app.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
    val userDir = File(storageDir, userId)

    if (userDir.exists() && userDir.isDirectory) {

```

```

        userDir.listFiles()?.filter { it.isDirectory }?.let
{ folders.addAll(it) }
    }
    folders
}

fun deleteImage(imagePath: String) {
    viewModelScope.launch {
        try {
            // Remove image from main list
            val file = File(imagePath)
            if (file.exists()) {
                if (file.delete()) {
                    showToastMessage("Изображение удалено")
                } else {
                    showToastMessage("Не удалось удалить
изображение")
                }
            } else {
                showToastMessage("Изображение не найдено")
            }

            // Remove from any folder it might be in
            _folderImagesMap.value?.forEach { (folder,
images) ->
                images.remove(imagePath)
            }

            _folderImagesMap.postValue(_folderImagesMap.value) // Trigger
update

            saveFolderImagesMap()
            loadData() // Refresh data after deletion
        } catch (e: Exception) {
            Log.e("GalleryViewModel", "Error deleting
image", e)
            showToastMessage("Error deleting image:
${e.message}")
        }
    }
}

fun moveImageToFolder(imagePath: String,
destinationFolderName: String) {
    viewModelScope.launch {
        try {
            Log.d("GalleryViewModel", "moveImageToFolder:
imagePath=$imagePath,

```

```

destinationFolderName=$destinationFolderName")
        Log.d("GalleryViewModel", "moveImageToFolder:
_folderImagesMap.value before=${_folderImagesMap.value}")

        val srcFile = File(imagePath)
        val destFile = File(destinationFolderName,
srcFile.name)

        if (srcFile.exists()) {
            srcFile.copyTo(destFile, overwrite = true)
            Log.d("GalleryViewModel", "Файл успешно
скопирован: ${destFile.absolutePath}")

            // Теперь добавляем изображение в список
папки

            val currentMap = _folderImagesMap.value ?:
mutableMapOf()

            val imageList =
currentMap.getOrPut(destinationFolderName) { mutableListOf() }
            imageList.add(destFile.absolutePath)
            _folderImagesMap.postValue(currentMap)

            Log.d("GalleryViewModel",
"moveImageToFolder: _folderImagesMap.value
after=${_folderImagesMap.value}")
            showMessage("Изображение перемещено в
папку")

            saveFolderImagesMap() // Сохраняем
loadData() // reload the data

        } else {
            showMessage("Исходный файл не найден")
        }
    } catch (e: Exception) {
        Log.e("GalleryViewModel", "Ошибка при
перемещении изображения", e)
        showMessage("Ошибка при перемещении:
${e.localizedMessage}")
    }
}

fun getFolderNames(callback: (List<String>?) -> Unit) {
    viewModelScope.launch {

```

```

        val userId = auth.currentUser?.uid ?: return@launch
        val storageDir =
app.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
        val userDir = File(storageDir, userId)

        val folderList = if (userDir.exists() &&
userDir.isDirectory) {
            userDir.listFiles { file -> file.isDirectory
}?.map { it.name }?.toList()
            } else {
                emptyList()
            }

        withContext(Dispatchers.Main) {
            callback(folderList)
        }
    }
}

fun getImagesInFolder(folderPath: String): List<Uri> {
    Log.d("GalleryViewModel", "getImagesInFolder:
folderPath=$folderPath")
    Log.d("GalleryViewModel", "getImagesInFolder:
_folderImagesMap.value=${_folderImagesMap.value}")
    val imagePaths = _folderImagesMap.value?.get(folderPath)
?: emptyList()

    val uris = imagePaths.mapNotNull { imagePath ->
        val file = File(imagePath)
        if (file.exists() && file.isFile) {
            val uri = Uri.fromFile(file)
            Log.d("GalleryViewModel", "getImagesInFolder:
Found URI: $uri for path: $imagePath")
            uri
        } else {
            Log.w("GalleryViewModel", "getImagesInFolder:
File does not exist or is not a file: $imagePath")
            null
        }
    }
    Log.d("GalleryViewModel", "getImagesInFolder: Returning
URIs: $uris")
    return uris
}

fun showToastMessage(message: String) {

```

```

        _showToast.postValue(message)
    }

    fun clearToastMessage() {
        _showToast.postValue(null)
    }

    fun createFolder(folderName: String, callback: (Boolean) ->
Unit) {
        viewModelScope.launch {
            try {
                val userId = auth.currentUser?.uid ?:
return@launch
                val storageDir =
app.getExternalFilesDir(Environment.DIRECTORY_PICTURES)
                val userDir = File(storageDir, userId)

                if (!userDir.exists()) {
                    userDir.mkdirs()
                }

                val newFolder = File(userDir, folderName)
                val success = if (!newFolder.exists()) {
                    newFolder.mkdirs()
                } else {
                    false
                }
                loadData()
                withContext(Dispatchers.Main) {
                    callback(success)
                }
            } catch (e: Exception) {
                Log.e("GalleryViewModel", "Error creating
folder", e)
                withContext(Dispatchers.Main) {
                    callback(false)
                }
            }
        }
    }

    private fun saveFolderImagesMap() {
        val editor = sharedPreferences.edit()
        val gson = Gson()
        val json = gson.toJson(_folderImagesMap.value)
        editor.putString("map", json)
    }

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.20.001 ПЗ

Лист

70

```

        editor.apply()
        Log.d("GalleryViewModel", "Saved folderImagesMap to
SharedPreferences")
    }

    private fun loadFolderImagesMap() {
        val gson = Gson()
        val json = sharedPreferences.getString("map", null)
        val type = object : TypeToken<MutableMap<String,
MutableList<String>>>() {}.type

        val loadedMap: MutableMap<String, MutableList<String>>?
= gson.fromJson(json, type)
        // Check for null and provide a default empty map if
needed
        _folderImagesMap.postValue(loadedMap ?: mutableMapOf())
        Log.d("GalleryViewModel", "Loaded folderImagesMap from
SharedPreferences")
    }
}

```

Приложение В
(Обязательное)

Приложение по формированию гардероба путешественника

Презентация

					<i>ПГУ1.090301.20.001 ПЗ</i>	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

ПРИЛОЖЕНИЕ ПО ФОРМИРОВАНИЮ ГАРДЕРОБА ПУТЕШЕСТВЕННИКА

Автор работы: студентка гр. 21ВВП1 Сорокина Елена
Андреевна

Руководитель: д.т.н., профессор Гурин Евгений Иванович

1

ЦЕЛИ РАБОТЫ

Цель выпускной квалификационной работы – разработка функционального и удобного Android-приложения, которое может помочь путешественникам организовать свой гардероб без необходимости постоянно обращаться к своим вещам.

В приложении должно быть реализовано:

- регистрация и авторизация пользователей;
- возможность делать фотографии в самом приложении;
- просмотр сделанных фото;
- составление образов;
- просмотр погоды на нужные даты и город.

2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.20.001 ПЗ

Лист

73

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И АКТУАЛЬНОСТЬ

- Глобальная цифровизация
- Необходимость использовать телефон во всех повседневных делах
- Удобство
- Весь гардероб собран в одном месте, что помогает и упрощает формирование багажа

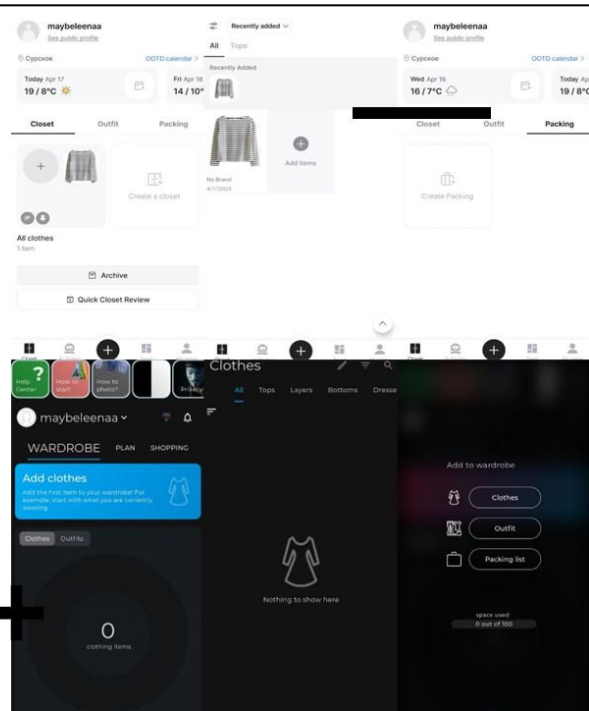
3

ОБЗОР АНАЛОГОВ

Приложение Acloset

Приложение GetWardrobe

4



ОБЗОР АНАЛОГОВ

Приложение PackPoint



5

СТЕК ТЕХНОЛОГИЙ



Firebase



Kotlin



6

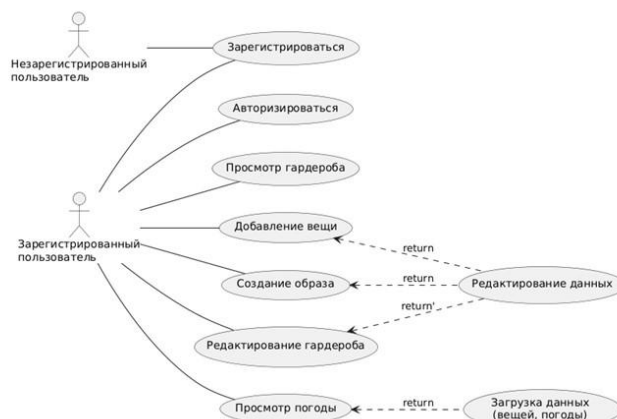
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.20.001 ПЗ

Лист

75

UML- ДИАГРАММА ПРЕЦЕДЕНТОВ



7

ЗАКЛЮЧЕНИЕ И ПЕРСПЕКТИВЫ РАЗВИТИЯ

–Реализация

Приложение со всеми необходимыми функциями и привязкой к API погоды

–Дальнейшее развитие

Обеспечение большей безопасности данных пользователей и расширение функций гардероба

8

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.20.001 ПЗ

Лист

76

СПАСИБО ЗА
ВНИМАНИЕ



9

					ПГУ1.090301.20.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

