МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка Android-приложения «Цифровое спасибо»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-324.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-402
_____ Д.Е. Малков

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-402
Малкову Дмитрию Евгеньевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764/13-12)
Разработка Android-приложения «Цифровое спасибо».
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Material Designs. [Электронный ресурс] URL: https://m3.material.io (дата обращения: 24.01.2024 г.).
3.2. Android Developers. [Электронный ресурс] URL: https://developer.android.com (дата обращения: 17.02.2024 г.).
3.3. Kotlinlang. Официальный сайт языка программирования Kotlin. [Электронный ресурс] URL: https://kotlinlang.org (дата обращения: 24.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Анализ предметной области.
4.2. Проектирование архитектуры мобильного приложения.
4.3. Реализация мобильного приложения.
4.4. Тестирование мобильного приложения.
5. **Дата выдачи задания:** 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП Н.С. Силкина

Задание принял к исполнению Д.Е. Малков

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

На сегодняшний день важность мотивации сотрудников компании сложно переоценить. Компании стремятся к тому, чтобы сотрудникам нравилась их работа в таком случае отдача будет максимальной, а сотрудничество плодотворным.

Финансовые поощрения производят кратковременный эффект. Необходимо иметь такую систему мотивации персонала, которая включает способы нематериальной поддержки, которые апеллируют к высшим потребностям человека таким как признание, саморазвитие, реализация потенциала.

Термин «Цифровое спасибо» описывает практику выражения признания и благодарности сотрудникам с использованием цифровых средств и инструментов. Это включает в себя отправку электронных сообщений, написание благодарственных писем в рабочих чатах, применение специализированных платформ и приложений для выражения признательности.

Эффективность данного метода подтверждается научными статьями и исследованиями такими как статья «The Power of Workplace Recognition» – подтверждает, что регулярное признание и благодарность на рабочем месте улучшают мотивацию, энтузиазм и результативность сотрудников [1].

Статья «The Power of Thanks», опубликованная в престижном журнале Harvard Business Review, тщательно исследует влияние благодарности на улучшение рабочей культуры, стимулирование роста командного взаимодействия и формирование позитивной атмосферы внутри организации [2].

Постановка задачи

Целью выпускной квалификационной работы является разработка Android-приложения «Цифровое спасибо» для улучшения рабочей культуры, основанной на признании и поощрении сотрудников. Мобильное приложение поможет привнести более позитивный и приятный опыт работы

для всех участников и позволяет укрепить взаимоотношения в коллективе. Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области;
2) провести анализ требований;
3) спроектировать архитектуру системы;
4) реализовать и протестировать мобильное приложение.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 50 страниц, объем списка литературы – 15 источников.

В первой главе описывается предметная область разработки мобильных приложений. Также были рассмотрены мобильные приложения, представленные на мировом рынке и выделен необходимый функционал.

Вторая глава посвящена описанию технологий для реализации серверной части приложения. Приведены функции API доступа и основные сущности для взаимодействия пользователя с платформой.

Третья глава посвящена проектированию мобильного приложения. В данной главе проведен анализ технологий для реализации мобильных приложений, приведено описание основных компонентов приложения. Представлена архитектура для построения кодовой базы продукта.

В четвертой главе содержится реализация мобильного приложения с описанием функций и пользовательского интерфейса.

Пятая глава содержит таблицу с результатами тестирования программного продукта.

Приложение А содержит спецификацию вариантов использования.

Приложение Б содержит таблицу сущностей базы данных.

Приложение В содержит диаграмму компонентов UI слоя приложения.

Приложение Г содержит диаграмму деятельности «Отправка благодарности».

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Предметная область проекта

В настоящее время корпоративные приложения стали неотъемлемой частью организационной культуры во многих компаниях. Они широко распространены и предлагают различные функциональные возможности, нацеленные на улучшение взаимодействия и мотивацию сотрудников. Каждое из этих приложений обладает своими особенностями и уникальными характеристиками, которые делают их привлекательными для различных организаций. Однако, несмотря на разнообразие, они все стремятся к общей цели – повышению эффективности и продуктивности работы персонала, созданию благоприятной рабочей атмосферы и укреплению внутреннего сообщества компании.

Результатом выпускной квалификационной работы является Android-приложение «Цифровое спасибо» с функционалом, направленным на построение культуры признания и благодарности в компании.

Анализ аналогичных проектов

В магазине приложений можно выделить несколько аналогов, каждый из которых преследует свою стратегию и подход к достижению этих целей:

1) Nectar – приложение для обмена благодарностями сотрудников [3];

2) Kudos – peer-to-peer платформа для построение рабочей культуры на основе признания [4];

3) Snappy – программа вознаграждения персонала [5].

Рассмотрим общий требуемый функционал такого рода приложений. Главная страница приложения Nectar содержит ленту событий, такие как благодарности сотрудников, победы в вызовах (рисунок 1).

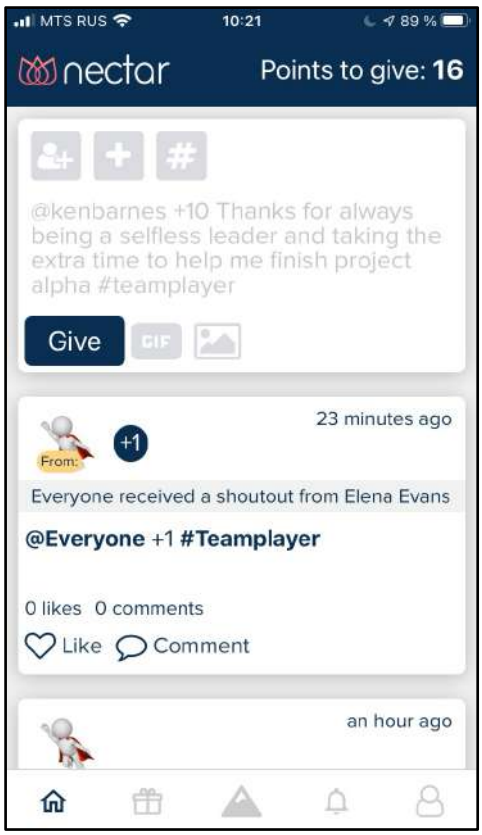

Рисунок 1 – Скриншот главного экрана

Пример экрана отправки благодарности изображен на рисунке 2.

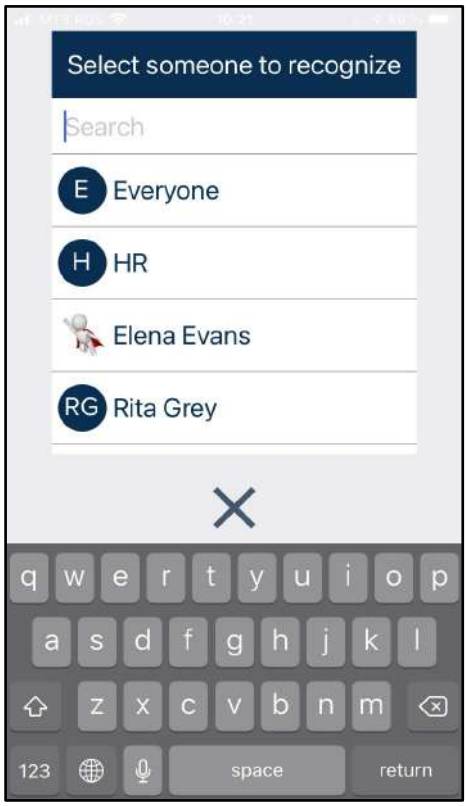

Рисунок 2 – Экран отправки благодарности

В приложение Nectar есть челленджи для сотрудников (рисунок 3), за выполнение которых можно зарабатывать баллы.

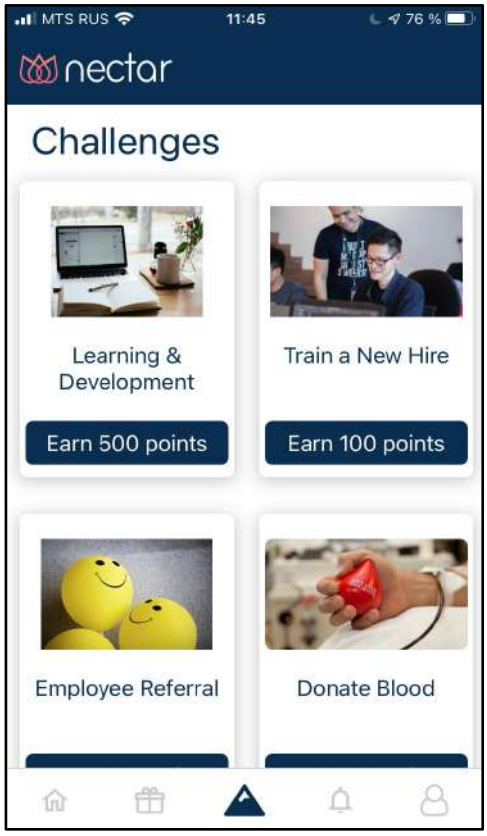

Рисунок 3 – Экран челленджей

В ходе изучения потенциальных приложений-конкурентов, был определен набор критериев для их сравнения, который представлен в таблице 1.

Таблица 1 – Обзор аналогов

Критерии	**Nectar**	**Kudos**	**Snappy**	**Thanks**
Мотивация сотрудников материальными средствами	+	–	+	+
Мотивация сотрудников нематериальными средствами	+	+	+	+
Возможность устанавливать свои виды поощрений от каждой компании	–	+	+	+
Наличие «Вызовов» (Челленджей) для сотрудников	+	–	–	+
Кастомизация продукта под цвета текущей организации	–	–	–	+

Исходя из критериев сравнения, можно отметить необходимый функционал для реализации.

1.2. Обзор технологий для реализации приложения

Наиболее популярными технологиями для реализации мобильных приложений являются View [6] и Jetpack Compose [7] – это две разные технологии для создания пользовательского интерфейса в мобильных приложениях на платформе Android. В зависимости от требований и потребностей проекта, может быть лучше использовать одну из этих технологий. Существуют также кроссплатформенные технологии, позволяющие создавать приложения сразу под несколько платформ, но они имеют свои ограничения в реализации, потому в рамках выпускной квалификационной работы будут рассмотрены исключительно нативные технологии для создания приложений под Android.

View – это основной компонент пользовательского интерфейса в Android, который используется в Android SDK с первой версии платформы. С тех пор появилось множество библиотек, которые помогают решать широкий спектр проблем. View основан на XML разметке и обеспечивает широкий набор базовых возможностей для отображения элементов пользовательского интерфейса таких как текст, кнопки, изображения и так далее. Также при наследовании класса View можно написать свою реализацию любого UI компонента, улучшить существующий или создать новый, что позволит настроить необходимое поведения пользовательского интерфейса, оптимизировать отрисовку компонента, улучшив производительность.

К особенностям Android View относятся:

1) разные языки для описания верстки (XML) и описания бизнес-логики (Kotlin/Java);

2) View хранит состояние самого компонента, такие как, методы и атрибуты;

3) зависимость от платформы;

4) высокая производительность.

Основным недостатком View является то, что некоторые новые возможности трудно, а иногда невозможно внести в старые версии Android из-за сильной связи платформы с базовым классом View.

Jetpack Compose, представленный Google в марте 2019 года, предлагает декларативный подход к разработке пользовательского интерфейса. Разработчики могут создавать пользовательский интерфейс напрямую в коде на Kotlin, что делает процесс разработки более естественным и интуитивным. Нет необходимости переключаться между XML и кодом, что упрощает понимание и поддержку кода. В данной технологии доступны различные готовые компоненты и функции для упрощения процесса разработки. Однако, Jetpack Compose еще относительно новая технология и не обладает такой же стабильностью, как View. Наличие аннотаций, отмечающих код как экспериментальный API, а также несовместимость с некоторыми базовыми библиотеками. Данные факторы могут создать определенные проблемы при использовании Jetpack Compose в качестве единственной технологии для реализации.

Таким образом, выбор остается за View, как более устоявшаяся и надежная технология для реализации.

Вывод по первой главе

В ходе анализа аналогов были выделены ключевые требования к функционалу и выбрана оптимальная технология для разработки итогового приложения. На основе этих данных были сформулированы критерии, которым должно соответствовать окончательное приложение, как указано в таблице 1. Кроме того, особое внимание уделяется дизайну пользовательского интерфейса: он должен быть простым и интуитивно понятным для пользователей.

2. ОПИСАНИЕ API

В разрабатываемом приложении предусмотрена единственная база данных, которая размещена на сервере. В этой базе хранится информация, предоставляемая сервисом.

В реализации серверной части применяется фреймворк Django [8], а в качестве базы данных используется PostgreSQL [9]. Приложение Б содержит таблицу основных сущностей, которые присутствуют в базе данных.

2.1. Функции API

Составление API осуществлялось согласно правилам и рекомендациям REST API, что обеспечивает стандартизацию и удобство использования интерфейса для разработчиков. Такой подход позволяет строго определить цели каждого запроса и обеспечивает соответствие принципам RESTful взаимодействия между клиентом и сервером. Рассмотрим функции API для чтения и записи информации:

1) функции чтения записей из базы данных, GET запросы:

– /events/ – получение списка актуальных событий в текущей организации;

– /user/balance/ – получение баланса пользователя;

– /user/transactions/ – получение списка переводов;

– /user/organizations/ – получение списка организаций, в которых состоит пользователь;

– /challenges/ – получение списка челленджей;

– /user/profile/ – получение данных о себе;

– /employees/ – получение списка сотрудников текущей организации;

– /markets/available/ – получение списка доступных бенефит кафе в текущей организации;

– /markets/{marketplace_id}/cart/ – получение списка товаров в корзине;

– /markets/{marketplace_id}/offers/ – получение списка товаров;

– /markets/{marketplace_id}/orders/ – получение списка истории заказов;

– /get-comments/{objectId} – получение списка комментариев конкретного объекта;

– /get-likes/{objectId} – получение списка лайков у объекта;

2) функции записи информации в базу данных:

– POST /auth/ – авторизация пользователя в системе;

– POST /user/change-organization/ – смена текущей организации;

– POST /cancel-transaction/ – отмена перевода;

– POST /send-coins / – отправка «спасибок» другому пользователю;

– POST /create-comment / – отправка комментария;

– DELETE /delete-comment/ – удаление комментария(только своего);

– PUT /update-profile/ – обновление информации в профиле;

– POST /create-challenge/ – создание челленджа;

– POST /create-challenge-report/ – отправка отчета к челленджу;

– POST /press-like / – отправка реакции;

– POST /market/{marketplace_id}/add-to-cart/ – добавление товара в корзину;

– POST /market/{marketplace_id}/orders/ – оформление заказа для товаров из коризны;

– DELETE /market/{marketplace_id}/cart/{offers_id} – удаление товара из корзины.

Вывод по второй главе

Была описана основная функциональность бекенда, которая будет использоваться при реализации мобильного приложения в рамках выпускной квалификационной работы.

3. ПРОЕКТИРОВАНИЕ

3.1. Анализ требований

Функциональные требования к системе

Основной задачей данной работы является разработка мобильного приложения «Цифровое спасибо» под мобильную платформу Android с функционалом, направленным на взаимодействие коллег между собой. Приложение должно включать в себя экраны со следующим функционалом:

1) главный экран, отображающий ленту произошедших событий;
2) экран события, демонстрирующий детальную информацию;
3) экран баланса, в котором указано количество полученных «спасибок»;
4) экран истории со списком всех благодарностей с участием пользователя;
5) экран участников с полным списком всех участников данной организации и возможностью их поиска;
6) экран профиля участника;
7) экран отправки благодарности, с возможностью выбора кому отправить благодарность;
8) экран своего профиля;
9) экран настройки профиля;
10) экран авторизации;
11) экран смены организации;
12) экран списка челленджей, содержащий список всех челленджей;
13) экран создания челленджа;
14) экран деталей челленджа для просмотра информации о челлендже;
15) экран отправки отчета к челленджу;
16) экран маркета, для просмотра доступных бенефитов;
17) экран деталей бенефита для просмотра подробностей и возможности добавления в корзину;

18) экран корзины с бенефитами для заказа;
19) экран истории заказов;
20) экран настройки приложения.

Нефункциональные требования к системе

Приложение должно быть реализовано для мобильной платформы Android, с поддержкой минимальной версии Android 8. Написание кода должно быть на языке Kotlin. Альтернативой которому служит Java, но является устаревшим решением для написания приложений для Android.

3.2. Описание вариантов использования

На рисунке 4 представлена диаграмма вариантов использования, согласно которой авторизованный пользователь (актер) может выполнять такие действия как:

1) просмотр событий;
2) реагировать (оставлять комментарии и лайки) к всем видам объектов;
3) просмотр баланса;
4) просмотр своей истории событий;
5) благодарить других пользователей;
6) редактировать свой профиль;
7) создавать челленджи;
8) участвовать в челленджах (отправлять отчеты к ним);
9) обменивать внутреннюю валюту на бенефиты своей компании;
10) просмотр товаров в бенефит кафе;
11) просмотр истории заказов;
12) просмотр товаров в корзине.

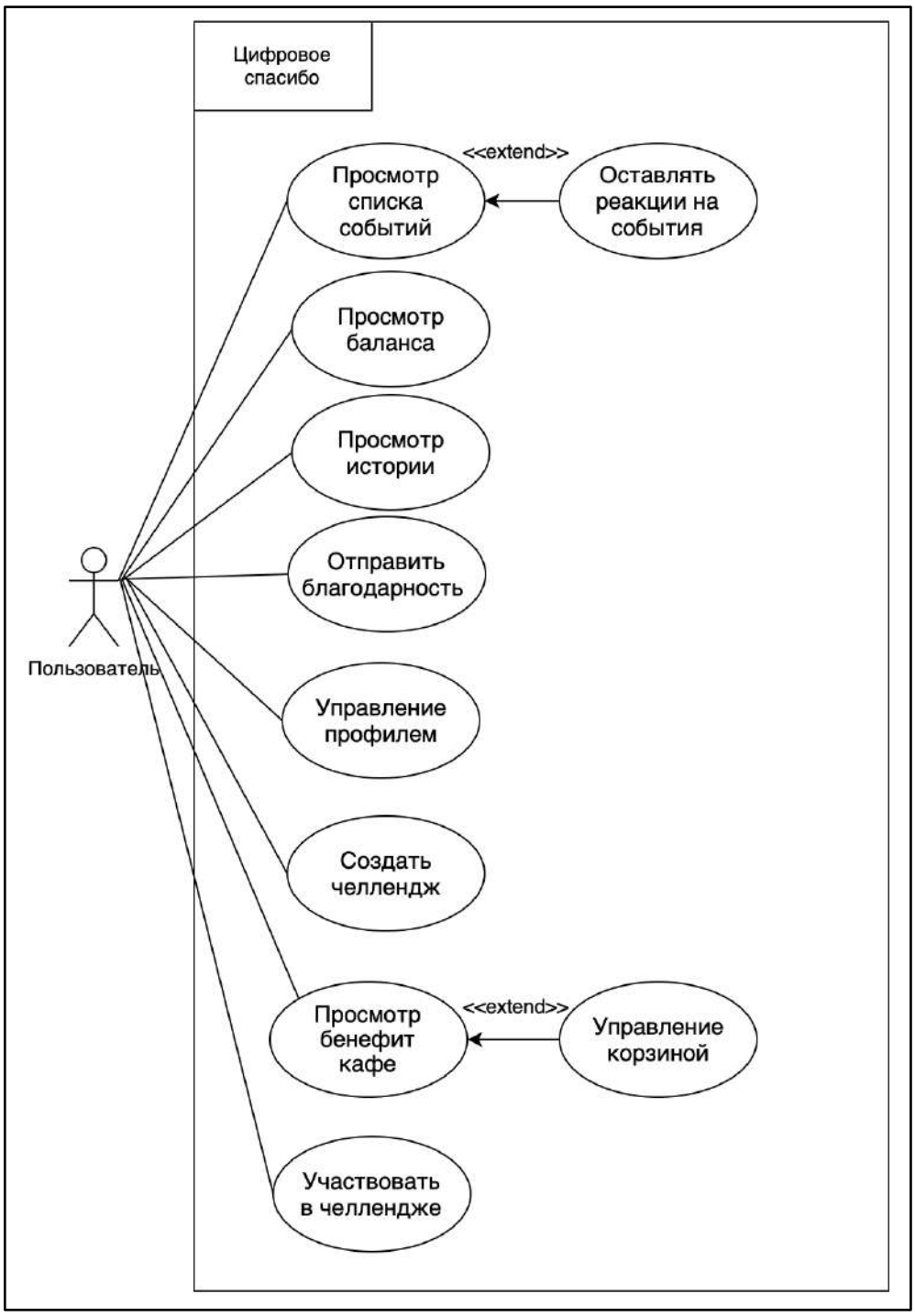

Рисунок 4 – Диаграмма вариантов использования

3.3. Архитектура мобильного приложения

Во время проектирования мобильного приложения были выбраны такие подходы, как Clean Architecture для общей организации кода проекта и MVVM для разделения ответственности в UI слое приложения. Рассмотрим каждый из подходов подробнее.

Clean Architecture подразумевает разделение всего кода приложения на 3 основных слоя (рисунок 5). Слой данных [10], domain слой, UI слой. Такое разделение нужно для снижения связности проекта и следования одному из SOLID [11] принципов, принципу единственной ответственности. Также упростит поддержку и тестирование кода разработчиками.

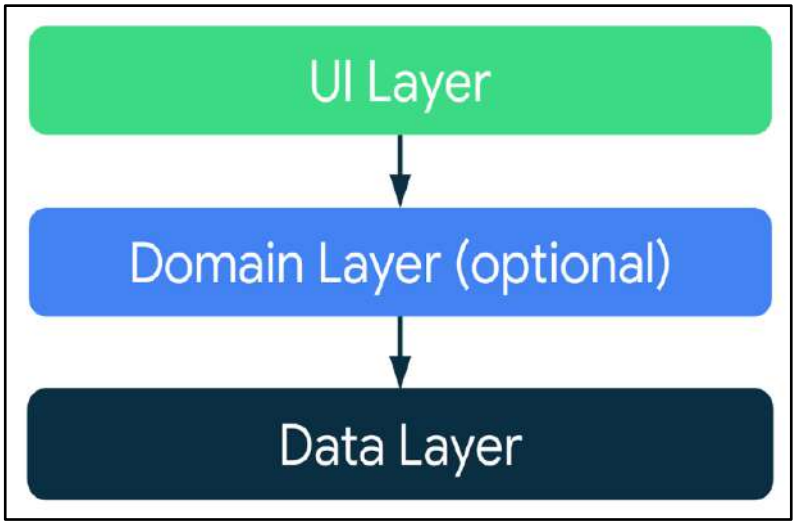

Рисунок 5 – Схема разделения мобильного приложения на логические слои

Data слой – предоставляет необходимые для приложения данные. Реализация слоя данных может включать в себя неограниченное количество источников данных, такие как локальная база данных, удаленная база данных, репозитории для связи источников данных, необходимые сущности. Этот слой является частью слоя фреймворков. Может включать в себя любую сетевую библиотеку, такие как Retrofit [12], Volley [13] или Ktor [14]. Помимо этого, может содержать библиотеку Room [15] для работы с локальной базой данных.

Domain слой является не обязательным слоем в контексте чистой архитектуры, содержит варианты использования и интеракторы для связи слоя данных и слоя презентации. Данный слой может содержать бизнес логику приложения, подготавливать данные для дальнейшего использования их в UI слое.

UI слой – служит только для отображения данных пользователю и обработке событий. Данный слой состоит из State Holder (держатель

состояний) и UI элементов, составляющие экран приложения. Входит в число слоев (рисунок 6), зависимых от фреймворка, потому не должен содержать в себе никакой бизнес логики. ViewModel служит для хранения состояний View элементов и передает данные, ранее полученные из domain слоя на экран. Единственная задача View отобразить данных для пользователя и обработать события, такие как тап по экрану, жесты. Построенное UI слоя должно быть максимально эффективным для достижения лучшего опыта использования.

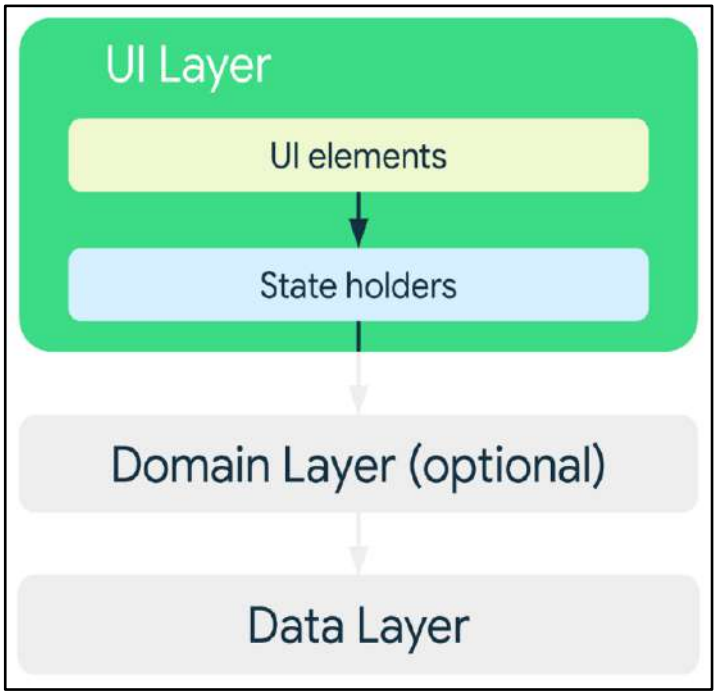

Рисунок 6 – Схема UI слоя мобильного приложения

Таким образом, для организации кодовой базы проекта выбрана Clean Architecture с разделением приложения на слои, каждый из которых имеет свою зону ответственности.

3.4. Компоненты системы

Мобильное приложение построено на основе архитектурного паттерна MVVM, а также по принципу single activity. MVVM (Model-View-ViewModel) – это архитектурный шаблон, используемый в разработке

программного обеспечения. Он разделяет пользовательский интерфейс (View) от бизнес-логики (ViewModel) и данных (Model).

Подход Single Activity подразумевает использование только одной activity на все приложение, где каждый экран и взаимодействие внутри приложения реализуются с помощью fragment. Такой подход обладает рядом преимуществ, например, позволяет упросить работу с жизненным циклом activity, теперь он будет совпадать с жизненным циклом приложения, анимировать переходы между фрагментами значительно проще. Также такой подход позволяет улучшить производительность приложения благодаря тому, что fragment легковеснее чем activity. К тому же благодаря фрагментам, интерфейсы и логика становятся более модульными и легко переиспользуемыми. Рассмотрим подробнее компоненты приложения.

На рисунке 1 приложения В представлена диаграмма основных компонентов UI слоя «Цифровое спасибо».

Приложение имеет следующие модели для представления данных.

1. EventItemModel – модель для хранения и обработки данных о событии, которое произошло.
2. BalanceModel – модель для хранения и обработки данных о балансе.
3. HistoryItemModel – модель для хранения и обработки данных о переводе.
4. ProductModel – модель для хранения и обработки данных о продукте.
5. EmployeeModel – модель для хранения и обработки данных о сотруднике.
6. ChallengeModel – модель для хранения и обработки данных о челлендже.
7. ProfileModel – модель для хранения и обработки данных о профиле.
8. CommentModel – модель для хранения и обработки данных о комментарии.

9. NewThanksModel – модель для хранения и обработки данных о новом переводе.

Также приложение имеет следующие фрагменты для отображения интерфейса.

1. EventsFragment – экран событий приложения, содержит список событий и фильтры для поиска.
2. BalanceFragment – экран баланса в приложении, отображает необходимую информацию о текущем балансе пользователя.
3. HistoryFragment – экран истории в приложении, содержит список переводов.
4. MembersFragment – экран списка сотрудников, содержит список сотрудников с основной информацией о каждом.
5. BenefitCafeFragment – экран бенефит кафе в приложении.
6. ChallengesFragment – экран списка челленджей.
7. ProfileFragment – экран профиля пользователя.
8. EditProfileFragment – экран редактирования профиля пользователя.
9. CommentsFragment – экран с комментариями.
10. NewThanksFragment – экран для отправки благодарности.

Приложение имеет следующие ViewModel, которые являются хранителями состояний экранов. Обеспечивают возможность пересоздания фрагментов с сохранением прогресса пользователя.

1. EventsViewModel – ViewModel отвечает за управление экраном событий в приложении.
2. BalanceViewModel – ViewModel отвечает за управление экраном баланса в приложении.
3. HistoryViewModel – ViewModel отвечает за управление экраном истории в приложении.
4. MembersViewModel – ViewModel отвечает за управление экраном списка сотрудников.

5. BenefitCafeViewModel – ViewModel отвечает за управление экраном бенефит кафе в приложении.

6. ChallengesViewModel – ViewModel отвечает за управление экраном списка челленджей.

7. ProfileViewModel – ViewModel отвечает за управление экраном профиля пользователя.

8. EditProfileViewModel – ViewModel отвечает за управление экраном редактирования профиля пользователя.

9. CommentsViewModel – ViewModel отвечает за управление экраном с комментариями.

10. NewThanksViewModel – ViewModel отвечает за управление экраном отправки благодарности.

Вывод по третьей главе

В данной главе был проведен тщательный анализ требований к мобильному приложению с учетом потребностей пользователей и бизнес-целей. Были исследованы и описаны различные варианты использования приложения, учитывая разнообразные сценарии и функциональные возможности. Кроме того, была представлена архитектура и компоненты системы, которые позволяют реализовать заданные требования и обеспечить эффективное взаимодействие между различными модулями приложения.

4. РЕАЛИЗАЦИЯ

4.1. Реализация отправки благодарности пользователю

Отправка благодарности пользователю – это основной функционал приложения. Для его реализации было написано три фрагмента, которые являются экранами. Один компонент ViewModel для хранения состояния и отправки запроса в сеть. На рисунке 2 приложения Г представлена диаграмма деятельности, описывающая процесс отправки благодарности другому пользователю.

На первом фрагменте представлено поле ввода для поиска конкретного получателя и список для вывода сотрудников, который отображает имя, аватар и логин пользователя. Скриншот экрана представлен на рисунке 8.

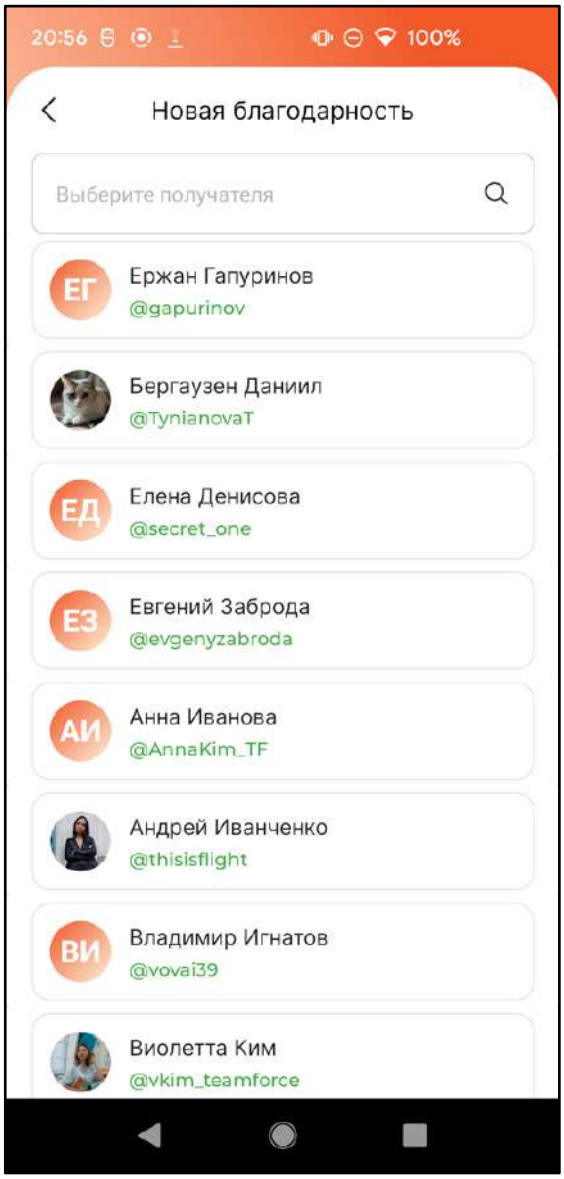

Рисунок 8 – Экран выбора пользователей

При клике на пользователя из списка произойдет переход на следующий экран с настройками для перевода. На экране настроек перевода указан текущий баланс, поле для ввода описания к переводу, теги к переводу, поле для ввода количества «спасибок», кнопка добавления фото и стикеров, а также указание типа перевода (рисунок 9).

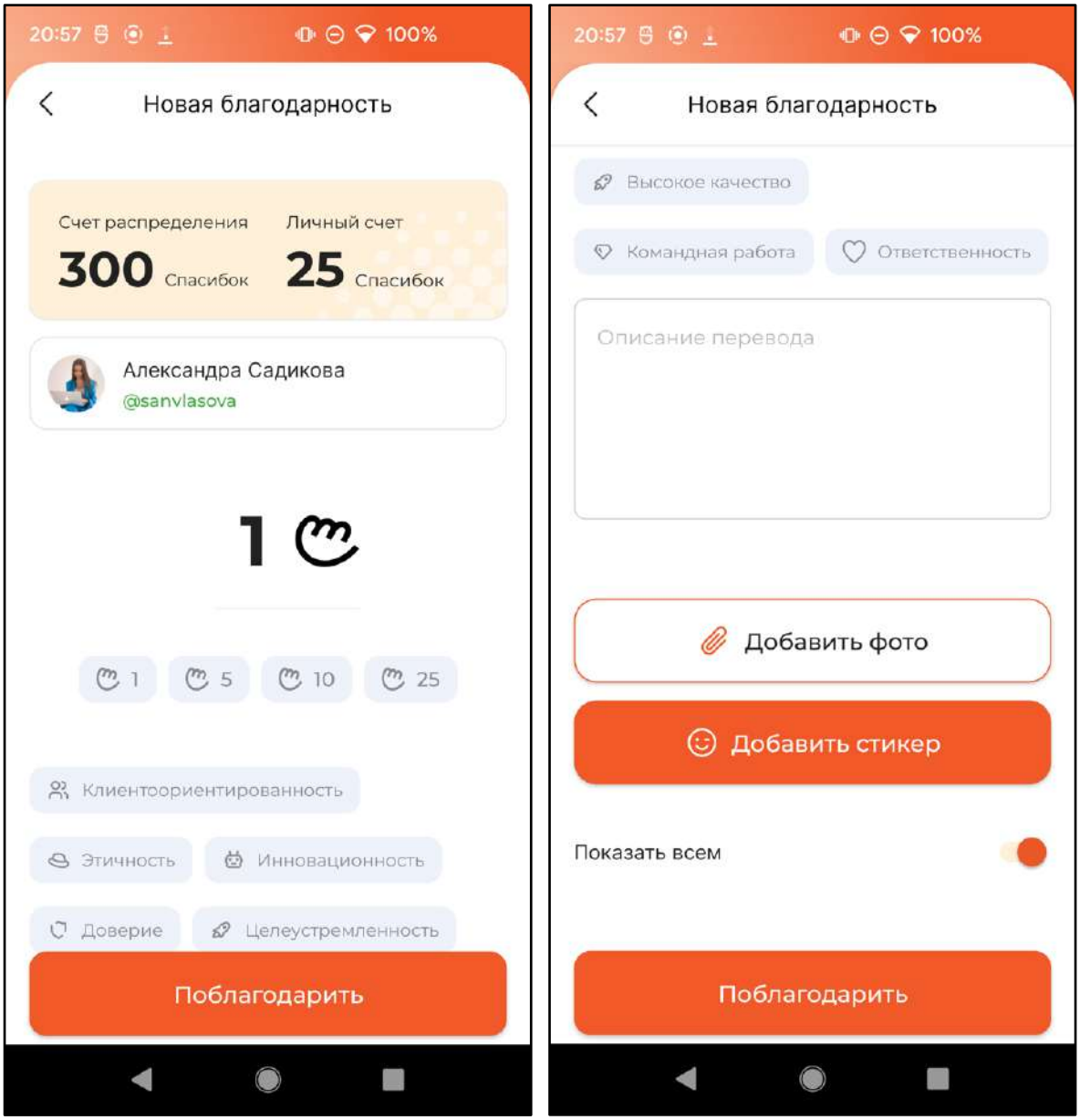

Рисунок 9 – Экран настроек перевода

Для отправки благодарности другому пользователю в фрагменте написана функция `sendCoins`, код представлен в листинге 1.

Листинг 1 – Функция перевода в фрагменте

```
private fun sendCoins () {
        val userId = user?.userId ?: -1
        val countText = binding.countValueEt.text.toString()
        val reason = binding.messageValueEt.text.toString()
        uriToMultipart(paths = listOfImagesUri)
        val isAnon = binding.isAnon.isChecked
        val isPublic = binding.isPublic.isChecked
        tagsToIdTags()
        if (userId != -1 && countText.isNotEmpty() &&
           (reason.isNotEmpty() || listCheckedIdTags.size > 0)
        ) {
            try {
                val count: Int = Integer.valueOf(countText)
                viewModel.sendCoinsWithImage(
                    userId,
                    count,
                    reason,
                    isAnon,
                    isPublic,
                    imagesFilePart,
                    listCheckedIdTags,
                    viewModel.checkedSticker.value?.id
                )
                binding.sendCoinBtnSticky.isClickable = false
                binding.sendCoinBtnSticky.isEnabled = false

            } catch (e: Exception) {
                Toast.makeText(requireContext(), e.message,
Toast.LENGTH_LONG).show()
            }
        } else {
            val snack = Snackbar.make(
                requireView(),
                requireContext().resources. getString(R.string.unsuccess-
fulSendCoins),
                Snackbar.LENGTH_LONG
            )
            snack.setTextMaxLines(3)
                .setTextColor(context?.getColor(R.color.white)!!)
                .setAction(context?.getString(R.string.OK)!!) {
                    snack.dismiss()
                }
            snack.show()
        }
    }
```

Перед отправкой данных на сервер происходит считывание полей ввода, также переключателей. Проверка необходимых данных на наличие, таких как `userId` получателя, описание перевода в поле `reason`, количество «спасибок» в поле `countText`. Поля `userId, countText, reason` являются обязательными. Если одно из обязательных полей будет пустым, а кнопка отправки нажата, то будет выведен диалог с сообщением о необходимости заполнить обязательные поля и перевод не произойдет.

Когда все обязательные поля заполнены, то будет вызван метод viewModel.sendCoinsWithImage, который отправляет запрос на перевод. Код метода представлен в листинге 2.

Листинг 2 – Метод отправки запроса на сервер

```
fun sendCoinsWithImage(
        recipient: Int, amount: Int,
        reason: String,
        isAnon: Boolean,
        isPublic: Boolean,
        imageFilePart: List<MultipartBody.Part?>?,
        listOfTagsCheckedValues: MutableList<Int>?,
        stickerId: Int?,
    ) {
        viewModelScope.launch {
            transactionsRepository.sendCoins(
                recipient,
                amount,
                reason,
                isAnon,
                isPublic,
                imageFilePart,
                listOfTagsCheckedValues,
                stickerId
            ).toResultState(
                onSuccess = {
                    _isSuccessOperation.postValue(true)
                },
                onError = { error, code ->
                    _sendCoinsError.postValue("$code $error")
                },
                onLoading = {
                }
            )
        }
    }
```

В данном методе кроме отправки запроса обрабатывается обратный вызов, считывается код ответа от сервера и сохраняется в Boolean поле true для уведомления фрагмента о том, что запрос выполнен успешно. При ином ответе от сервера, кроме успешного, обрабатывается присылаемый json объект с помощью библиотеки Gson и сохраняется в поле sendCoinsError для дальнейшего считывания его из фрагмента и вывода информации об ошибке пользователю.

Считывание полей из viewModel происходит согласно паттерну Observable. Изменения отслеживаются через поля viewModel и при их изменении происходит обновление UI на экране. Данный подход является

реактивным и позволяет динамически реагировать и показывать необходимый UI для пользователя в каждый момент времени. Код для считывания данных об ошибке переводе представлен в листинге 3.

Листинг 3 – Слушатель ошибок перевода

```
viewModel.sendCoinsError.observe(viewLifecycleOwner) {
            if (it.isNotEmpty()) {
                clearViewState()
                showSnackBar(it)
            }
        }
```

При получении ошибки, очищаются поля ввода, выводится текст ошибки в SnackBar. Кроме ошибки считываются изменения в поле `isSuccessOperation` для отслеживания успешности запроса. Код слушателя представлен в листинге 4.

Листинг 4 – Слушатель поля успешности запроса

```
viewModel.isSuccessOperation. observe(viewLifecycleOwner) {
            if (it) {
                showResultTransaction(amountThanks, user)
                clearCheckedItem()
                viewModel.setSuccessOperationFalse()
            }
        }
```

При успешном переводе слушатель получает `true`. Тогда происходит очистка поля фрагмента. Выводится сообщение об успешности операции и вызывается метод `showResultTransaction`, код которого представлен в листинге 5.

Листинг 5 – Метод перехода на экран результата перевода

```
private fun showResultTransaction(
        amountThanks: Int, user: UserBean?
    ) {
        clearViewState()
        val bundle = Bundle()
        bundle.putInt(
TransactionResultDialog.AMOUNT_TRANSFER,
amountThanks)
        bundle.putParcelable(
TransactionResultDialog.USER_DATA, user)

        findNavController().navigateSafely(
            R.id.action_global_transactionResultDialog,
            bundle,
            OptionsTransaction().optionForResultTransaction
        )
    }
```

Метод `showResultTransaction` принимает аргументы для дальнейшей их передачи в следующий фрагмент для вывода экрана результата перевода (рисунок 10). Для передачи данных из фрагментов используется класс `Bundle` из Android SDK. Это механизм для передачи данных между компонентами приложения такими как активности, фрагменты, сервисы. Он позволяет упаковывать и хранить различные типы данных, такие как примитивные типы данных, строки, массивы, списки, так и `parcelable` объекты и другие сериализуемые типы объектов в формате ключ-значение, где ключом выступает строка. Также вызывается метод `clearViewState` для сброса состояния экрана к изначальному, чтобы при следующем переводе, все поля были пустыми.

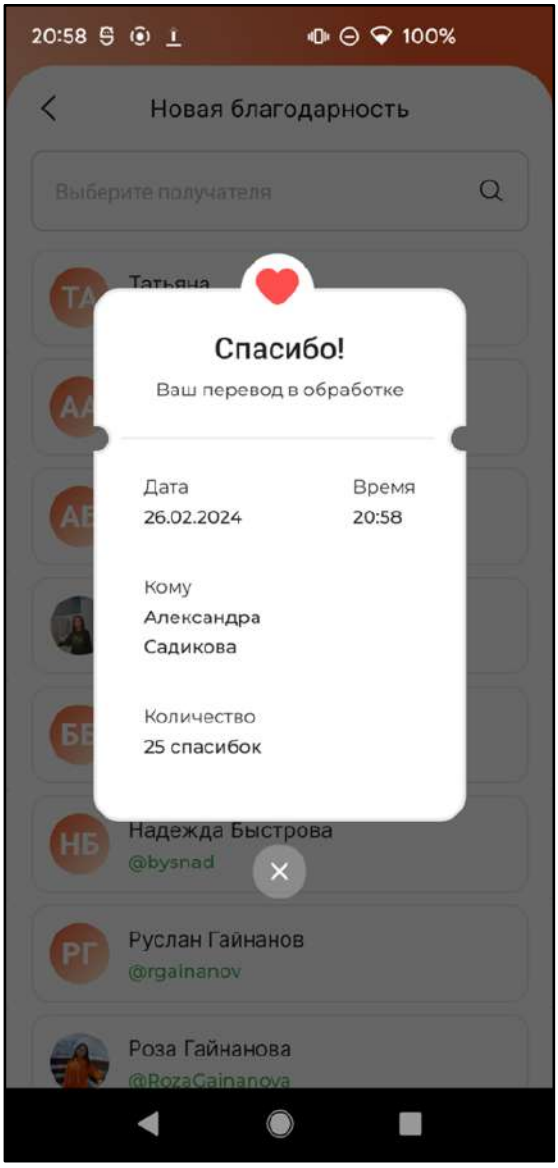

Рисунок 10 – Экран результата перевода

4.2. Реализация редактирования профиля

Для визуализации логики взаимодействия компонентов в системе в течение определенного времени при редактировании профиля, реализована диаграмма последовательности, описывающая процесс редактирования профиля. Диаграмма представлена на рисунке 11.

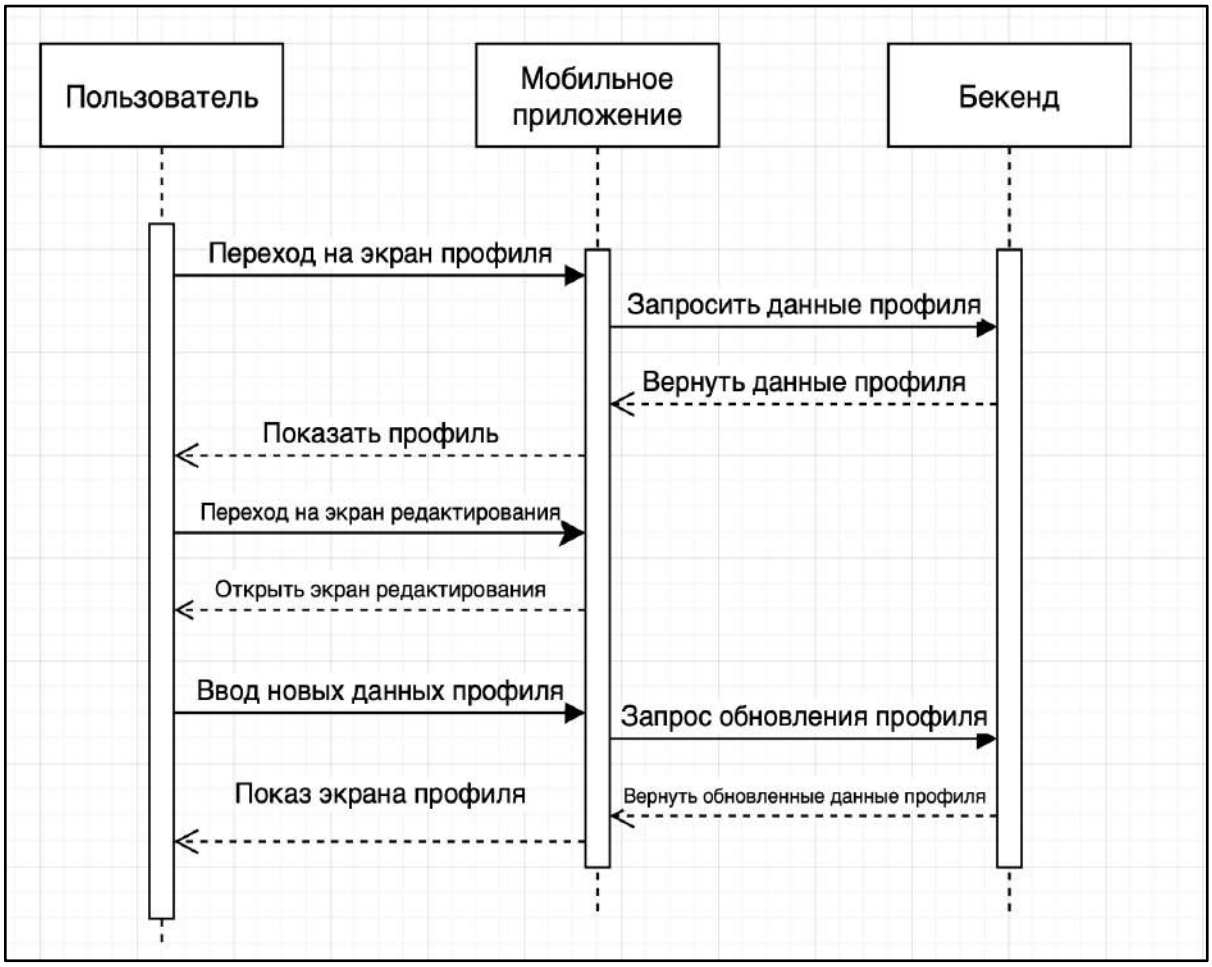

Рисунок 11 – Редактирование профиля

Для редактирования профиля, нужно попасть на целевой экран для изменения данных своего профиля. Попасть на него можно только через экран профиля, представленный на рисунке 12. Экран профиля показывает текущий статус пользователя, аватар, местоположение (если дано разрешение на считывание геолокации пользователя) также остальную информацию такую как контактные данные, ФИО, должность и подразделение сотрудника.

Рисунок 12 – Экран профиля

По клику на иконку редактирования происходит переход на экран редактирования профиля (рисунок 13). На нем пользователь имеет возможность редактировать личную информацию о себе такую как ФИО, пол, контактные данные. Поле даты рождения имеет дополнительную возможность для скрытия года, в таком случае другие пользователи при просмотре профиля увидят только день и месяц. Есть возможность привязать VK ID к своему аккаунту для быстрой авторизации в организациях. Такие данные как

должность и подразделение сотрудника, редактируется администратором при необходимости.

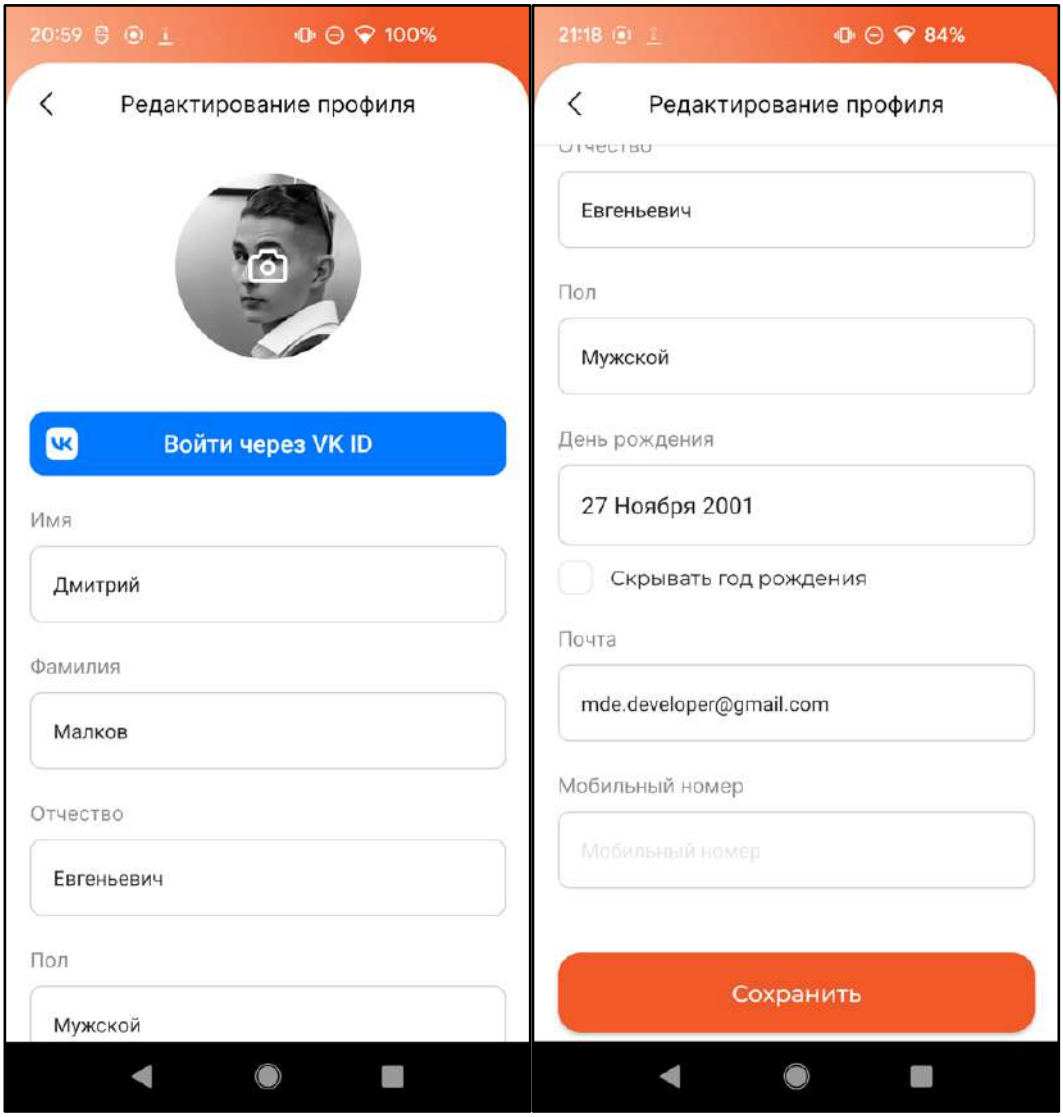

Рисунок 13 – Экран редактирования профиля

На экране редактирования профиля сразу вводятся уже заполненные поля. При переходе на фрагмент редактирования получаем модель профиля (листинг 6) через Bundle. Модель профиля представляет информацию о пользователе, для дальнейшего вывода в приложении.

Листинг 6 – Модель профиля

```
@Parcelize
data class ProfileModel(
    val id: String,
    val contacts: List<ContactModel>,
    val organization: String?,
    val department: String?,
    val tgId: String?,
```

```
    val tgName: String?,
    val photo: String?,
    val hiredAt: String?,
    val birthday: String?,
    val surname: String?,
    val firstname: String?,
    val middleName: String?,
    val nickname: String?,
    val jobTitle: String?,
    val status: String?,
    val gender: String?,
    val showYearOfBirth: Boolean = false,
    ): Parcelable
```

При клике на кнопку «Сохранить» происходит считывание полей ввода и отправка запроса на обновление профиля с отправкой всех параметров метода (листинг 7).

Листинг 7 – Отправка запроса на обновление информации профиля

```
viewModel.loadUpdateProfile(
                    firstName = firstName,
                    surname = surname,
                    middleName = middleName,
                    birthDay = viewModel.birthday.value,
                    showYearOfBirth = showYear,
                    gender = viewModel.gender.value
                )
```

Код функции для обновления профиля пользователя приведен в листинге 8.

Листинг 8 – Функция отправки запроса и обработки ответа сервера

```
fun loadUpdateProfile(
        userId: String,
        tgName: String?, surname: String?,
        firstName: String?, middleName: String?, nickname: String?, status:
String?,
        birthDay: String?, showYearOfBirth: Boolean, gender: String?
    ) {
        _isLoading.value = true
        viewModelScope.launch {
            withContext(Dispatchers.IO) {
                _isLoading.postValue(true)
                when (val result = profileRepository.loadUpdateProfile(
                    userId = userId, tgName, surname,
                    firstName, middleName, nickname, status, birthDay,
showYearOfBirth, gender
                )) {
                    is ResultWrapper.Success -> {
                        _updateProfile.postValue(result.value)
                    }
                    else -> {
                        if (result is ResultWrapper.GenericError) {
                            _updateProfileError.postValue("${result.code}
${result.error}")
```

```
                        }
                    }
                }
                _isLoading.postValue(false)
            }
        }
    }
```

При успешном обновлении профиля происходит возврат на экран профиля. При получении ошибки пользователю выводится сообщение, в таком случае возврата на предыдущий экран не происходит.

4.3. Реализация брендирования

Возможность брендирования представляет собой динамическую смену цветов у view во всем приложении. Получение цветовой схемы происходит с сервера. Запрос к базе представлен в листинге 9.

Листинг 9 – Запрос цветовой схемы из базы данных

```
  @GET("/organizations/{id}/brand/")
    suspend fun getOrganizationBranding(
        @Path("id") organizationId: Int,
    ): ColorsJson
```

Пример сущности `ColorsJson`, которая представлена в виде `data class` (листинг 10) служит для хранение набора цветов, используемых для окрашивания пользовательского интерфейса.

Листинг 10 – Сущность данных цветов

```
data class ColorsJson(
    @SerializedName("general-brand")
    val generalBrand: String,
    @SerializedName("general-brand-secondary")
    val generalBrandSecondary: String,
    @SerializedName("minor-info")
    val minorInfo: String,
    @SerializedName("minor-error")
    val minorError: String,
    @SerializedName("minor-success")
    val minorSuccess: String,
    @SerializedName("minor-warning")
    val minorWarning: String,
    @SerializedName("general-contrast")
    val generalContrast: String,
    @SerializedName("general-midpoint")
    val generalMidpoint: String,
    @SerializedName("general-negative")
    val generalNegative: String,
    @SerializedName("minor-info-secondary")
```

```
    val minorInfoSecondary: String,
    @SerializedName("minor-error-secondary")
    val minorErrorSecondary: String,
    @SerializedName("minor-success-secondary")
    val minorSuccessSecondary: String,
    @SerializedName("minor-warning-secondary")
    val minorWarningSecondary: String,
    @SerializedName("minor-negative-secondary")
    val minorNegativeSecondary: String,
    @SerializedName("general-contrast-secondary")
    val generalContrastSecondary: String,
    val extra1: String,
    val extra2: String,
    )
```

Для реализации установки цветовой схемы был описан интерфейс с целевым методом установки цвета. Исходный код интерфейса приведен в листинге 11.

Листинг 11 – Интерфейс для описания метода смены цвета

```
interface Themable {
    fun setThemeColor(theme: ColorsModel)
}
```

Каждый View элемент в приложении должен реализовать данный интерфейс для динамической смены цвета. Пример View элемента, реализующего интерфейс `Themable` в листинге 12.

Листинг 12 – Класс, реализующий интерфейс Themable

```
class ThemableCheckBox @JvmOverloads constructor(
    context: Context,
    attrs: AttributeSet? = null,
    defStyleAttr: Int = R.attr.checkboxStyle
): androidx.appcompat.widget.AppCompatCheckBox(context, attrs, def-
StyleAttr), Themable {
    private val btnColorTint = ColorStateList( arrayOf(
            intArrayOf(R.attr.state_checked),
            intArrayOf(-R.attr.state_checked),
        ), intArrayOf(
            Color.parseColor(Branding.appTheme.mainBrandColor),
            Color.parseColor(Branding.appTheme.generalContrastColor)
        )
    )
    override fun setThemeColor(theme: ColorsModel) {
        with(this) { buttonTintList = btnColorTint }
    }
}
```

Данный класс представляет собой `CheckBox` с возможностью динамической смены цвета. В функции `setThemeColor` происходит присваивание

нужного цвета для View. Цвета передаются как параметр функции `theme` с типом `ColorsModel`, данный `data class` представлен в листинге 13.

Листинг 13 – Класс данных `ColorsModel`

```
data class ColorsModel(
    // General Color
    val mainBrandColor: String = ConstBranding.GENERAL_BRAND,
    val secondaryBrandColor: String = ConstBranding.GENERAL_BRAND_SECOND-
ARY,
    val generalBackgroundColor: String = ConstBranding.GENERAL_BACKGROUND,
    val generalContrastColor: String = ConstBranding.GENERAL_CONTRAST,
    val generalContrastSecondaryColor: String = ConstBranding.GENERAL_CON-
TRAST_SECONDARY,
    val midpoint: String = ConstBranding.MIDPOINT,
    val generalNegativeColor: String = ConstBranding.GENERAL_NEGATIVE,

    // Minor Color
    val minorSuccessColor: String = ConstBranding.MINOR_SUCCESS,
    val minorSuccessSecondaryColor: String = ConstBranding.MINOR_SUC-
CESS_SECONDARY,
    val minorErrorColor: String = ConstBranding.MINOR_ERROR,
    val minorErrorSecondaryColor: String = ConstBranding.MINOR_ERROR_SEC-
ONDARY,
    val minorInfoColor: String = ConstBranding.MINOR_INFO,
    val minorInfoSecondaryColor: String = ConstBranding.MINOR_INFO_SECOND-
ARY,
    val minorWarningColor: String = ConstBranding.MINOR_WARNING,
    val minorWarningSecondaryColor: String = ConstBranding.MINOR_WARN-
ING_SECONDARY,
    val minorNegativeSecondaryColor: String = ConstBranding.MINOR_NEGA-
TIVE_SECONDARY,

    // Extra Color
    val extra1: String = ConstBranding.EXTRA_1,
    val extra2: String = ConstBranding.EXTRA_2,
)
```

Модель `ColorModel` является внутренним классом данных для хранения набора цветов и отличается от сущности тем, что имеет предопределенные стандартные значения, для избежания получения ошибки `Null pointer exception` при обращении к полю.

Вызов View метода `setThemeColor` будет происходить в Fragment. Fragment представляет собой полноценный экран приложения и содержит в себе множество View. Для реализации вызова функции брендирования у View была написана обертка над Fragment – BaseFragment, который служит для инкапсуляции некоторой повторяющейся логики, исключая большое количество шаблонного кода . Исходный код Base Fragment приведен в листинге 14.

Листинг 14 – Абстрактный класс BaseFragment

```
typealias Inflate<T> = (LayoutInflater, ViewGroup?, Boolean) -> T

abstract class BaseFragment<VB: ViewBinding>(
    private val inflate: Inflate<VB>
) : Fragment() {

    private var _binding: VB? = null
    val binding
        get() = _binding!!

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

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

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        binding.root.apply {
            traverseViews(this)
        }
        applyTheme()
    }

    private fun traverseViews(rootView: View) {
        val viewsStack = Stack<View>()
        viewsStack.push(rootView)

        while (!viewsStack.isEmpty()) {
            val view = viewsStack.pop()
            if (view is Themable) view.setThemeColor(appTheme)

            if (view is ViewGroup) {
                val childCount = view.childCount
                for (i in childCount - 1 downTo 0) {
                    val childView = view.getChildAt(i)
                    viewsStack.push(childView)
                }
            }
        }
    }

    abstract fun applyTheme()
}
```

Данный абстрактный класс является ключевым элементом приложения, так как он обеспечивает логику инициализации `view binding`, что позволяет удобно обращаться к элементам интерфейса по их идентификаторам. Это значительно упрощает процесс разработки, поскольку исключает

необходимость повторного написания кода для доступа к View на каждом экране фрагмента.

Важной особенностью класса является функция `traverseViews`, которая обеспечивает проход по всем вложенным View внутри фрагмента и вызывает функцию `setThemeColor` для элементов, реализующих интерфейс `Themable`. Это позволяет легко менять цветовую тему для элементов интерфейса.

Необходимо отметить, что класс является абстрактным и содержит абстрактный метод `applyTheme`, который предоставляет дополнительную возможность для настройки внешнего вида UI компонентов. Если View не реализует интерфейс `Themable`, разработчику предоставляется возможность задать атрибуты визуального оформления вручную для view в рамках фрагмента.

Вывод по четвертой главе

В данном разделе была описана реализация мобильного приложения. В результате был разработан программный продукт для платформы Android, в соответствии сформированным требованиям для реализации и выбранным инструментам разработки.

5. ТЕСТИРОВАНИЕ

В данном разделе показаны результаты функционального тестирования мобильного приложения (таблица 3). Тестирование включает в себя ряд вероятных сценариев использования.

Таблица 2 – Результаты тестирования

№	Описание	Шаги	Ожидаемый результат	Результат проверки
1	Запуск приложения	Запустить приложение	Приложение запущено	Пройден
2	Авторизация	Авторизоваться	Пользователь авторизован	Пройден
3	Корректное отображение экранов	1. Авторизоваться 2. Перейти на разные экраны приложения	Все экраны приложения отображаются корректно	Пройден
4	Отправка благодарности	1. Перейти на экран отправки благодарности 2. Выбрать пользователя 3. Поблагодарить его 4. Проверить историю	Благодарность отправлена успешно	Пройден
5	Изменение информации в профиле	1. Перейти на экран профиля 2. Перейти на экран редактирования профиля 3. Изменить данные профиля 4. Вернуться на экран профиля	Данные профиля успешно изменены.	Пройден
6	Отправка комментария к событию	1. Перейти на экран списка событий 2. Перейти на экран комментариев к событию 3. Оставить комментарий к событию	Комментарий к событию успешно отправлен.	Пройден
7	Отправка лайка к событию	1. Перейти на экран списка событий 2. Лайкнуть событие 3. Проверить количество лайков	Обновление количества лайков у события	Пройден
8	Создание челленджа	1. Перейти на экран создания челленджей 2. Создать челлендж 3. Переход на список челленджей	Челлендж создан и появился в списке	Пройден
9	Отправка отчета к челленджу	1. Перейти на экран списка челленджей 2. Выбрать челлендж 3. Выполнить задание из челленджа 4. Отправить отчет о выполнении задания	Успешная отправка отчета к челленджу	Пройден

Окончание таблицы 2

№	Описание	Шаги	Ожидаемый результат	Результат проверки
10	Удаление челленджа	1. Перейти на экран списка челленджей 2. Создать тестовый челлендж 3. Перейти на экран деталей тестового челленджа 4. Удалить челлендж	Переход на экран списка челленджей и вывод сообщение об успешности операции	Пройден
11	Проверка совместимости на различных устройствах	1. Запустить приложение на устройствах под управлением разных оболочек, основанных на операционной системе Android 2. Проверить корректность работы UI элементов на всех экранах	Работа UI компонентов на всех экранах одинаково корректна на каждом устройстве	Пройден
12	Проверка работы брендирования	1. Авторизоваться в организации с нестандартным набором цветов 2. Проверить корректность брендирования UI элементов на всех экранах приложения 3. Выйти из организации 4. Авторизоваться в организации со стандартным набором цветов 5. Проверить корректность брендирования UI элементов	Брендирование UI компонентов на всех экранах корректна	Пройден

Вывод по пятой главе

В результате проведенного тестирования приложения было установлено, что все функции работают корректно, а также не было выявлено никаких ошибок. Тестирование было проведено на устройствах разных производителей, что позволило проверить работоспособность приложения на различных реализациях оболочек операционной системы.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано мобильное приложение «Цифровое спасибо» с целью улучшения рабочей культуры и создания основы для признания достижений сотрудников. Для достижения этой цели были выполнены следующие задачи:

1) проведен анализ существующих решений в предметной области проекта;

2) определены требования к системе и выполнено ее проектирование с учетом функциональности, удобства использования и безопасности данных;

3) разработана архитектура мобильного приложения, обеспечивающая расширяемость и тестируемость программного продукта;

4) реализовано приложение для платформы Android, включающее функции признания сотрудников, возможность создания цифровых благодарностей, участие в челленджах;

5) проведено тестирование приложения для обеспечения его стабильной работы и проверки соответствия заявленным требованиям.

В ходе работы были освоены нативные технологии и инструменты для разработки мобильных приложений для операционной системы Android, с применением языка программирования Kotlin.

Планируется дальнейшее развитие проекта, включающее добавление новых функций таких как возможность проведения опросов среди сотрудников, расширение интерфейса для коммуникации и обратной связи. Добавление UI компонентных и e2e тестов для поддержания качества при нарастающей кодовой базе продукта.

ЛИТЕРАТУРА

1. Исследование The Power of Workplace Recognition. [Электронный ресурс] URL: https://www.gallup.com/analytics/472658/workplace-recognition-research.aspx (дата обращения: 25.02.2024 г.).

2. Статья The power of thanks. [Электронный ресурс] URL: https://news.harvard.edu/gazette/story/2013/03/the-power-of-thanks/ (дата обращения: 22.02.2024 г.).

3. Nectar. [Электронный ресурс] URL: https://nectarhr.com (дата обращения: 04.02.2024 г.).

4. Kudos. [Электронный ресурс] URL: https://www.kudos.com/platform/peer-recognition-software (дата обращения: 04.02.2024 г.).

5. Snappy. [Электронный ресурс] URL: https://www.snappy.com (дата обращения: 04.02.2024 г.).

6. View. [Электронный ресурс] URL: https://developer.android.com/studio/write/layout-editor (дата обращения: 25.02.2024 г.).

7. Jetpack Compose. [Электронный ресурс] URL: https://developer.android.com/jetpack/compose (дата обращения: 25.02.2024 г.).

8. Django. [Электронный ресурс] URL: https://www.djangoproject.com (дата обращения: 25.02.2024 г.).

9. PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org (дата обращения: 28.02.2024 г.).

10. Мартин Р. Чистая архитектура – СПб: Питер, 2019. – 432 с.

11. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПБ: Питер, 2002. – 366 с.

12. Retrofit. [Электронный ресурс] URL: https://square.github.io/retrofit/ (дата обращения: 25.02.2024 г.).

13. Volley. [Электронный ресурс] URL: https://github.com/google/volley (дата обращения: 25.02.2024 г.).

14. Ktor. [Электронный ресурс] URL: https://ktor.io (дата обращения: 25.02.2024 г.).

15. Room. [Электронный ресурс] URL: https://developer.android.com/training/data-storage/room (дата обращения: 25.02.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования для мобильного приложения, приведена в таблицах 1–9.

Таблица 1 – Спецификация прецедента «Просмотр списка событий»

Прецедент: Просмотр списка событий
ID: 1
Краткое описание: Возможность просмотра списка событий
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: Пользователь переходит на экран списка событий
Постусловия: Список событий выведен
Альтернативные потоки: Отсутствуют

Таблица 2 – Спецификация прецедента «Оставить лайк на событие»

Прецедент: оставить лайк на событие
ID: 2
Краткое описание: Возможность лайкнуть событие
Главные актеры: Пользователь
Предусловия: Пользователь находится на экране списка событий
Основной поток: Пользователь кликает на кнопку лайка у события
Постусловия: Количество лайков у события увеличивается на единицу
Альтернативные потоки: Отсутствуют

Таблица 3 – Спецификация прецедента «Просмотр баланса»

Прецедент: просмотр баланса
ID: 3
Краткое описание: Возможность просмотра баланса
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: Пользователь переходит на экран баланса
Постусловия: На экран выведен баланс пользователя
Альтернативные потоки: Отсутствуют

Таблица 4 – Спецификация прецедента «Просмотр истории»

Прецедент: просмотр истории
ID: 4
Краткое описание: Возможность просмотра истории
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: Пользователь переходит на экран истории
Постусловия: На экран выведен история переводов пользователя
Альтернативные потоки: Отсутствуют

Таблица 5 – Спецификация прецедента «Отправка благодарности»

Прецедент: отправка благодарности
ID: 5
Краткое описание: Возможность отправки благодарности
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: Пользователь переходит на экран отправки благодарности
Постусловия: Переход на экран с информацией об успешности перевода
Альтернативные потоки: Отсутствуют

Таблица 6 – Спецификация прецедента «Просмотр профиля»

Прецедент: просмотр профиля
ID: 6
Краткое описание: Возможность просмотра профиля
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: Пользователь переходит на экран профиля
Постусловия: На экран выведен профиль пользователя
Альтернативные потоки: Отсутствуют

Таблица 7 – Спецификация прецедента «Редактирование профиля»

Прецедент: редактирование профиля
ID: 7
Краткое описание: Возможность редактирования профиля
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: 1. Пользователь переходит на экран редактирования профиля 2. Изменяет данные профиля 3. Сохраняет изменения
Постусловия: 1. Вывод сообщения об успешном сохранении изменений 2. Переход на экран профиля с новыми данными
Альтернативные потоки: Отсутствуют

Таблица 8 – Спецификация прецедента «Создать челлендж»

Прецедент: создать челлендж
ID: 8
Краткое описание: Возможность создать челлендж
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: 1. Пользователь переходит на экран создания челленджа 2. Вводит необходимые данные о челлендже 3. Кликает на кнопку создать
Постусловия: 1. Вывод сообщения об успешном создании челленджа 2. Переход на экран списка челленджей
Альтернативные потоки: Отсутствуют

Таблица 9 – Спецификация прецедента «Участвовать в челлендже»

Прецедент: участвовать в челлендже
ID: 9
Краткое описание: Возможность участвовать в челлендже
Главные актеры: Пользователь
Предусловия: Пользователь авторизован
Основной поток: 1. Пользователь переходит на экран списка челленджей 2. Кликает на челлендж в котором хочет поучаствовать 3. Переходит на экран деталей челленджа 4. Кликает на кнопку отправки отчета к челленджу 5. Заполняет данные отчета 6. Отправляет отчет
Постусловия: 1. Вывод сообщения об успешной отправки отчета 2. Возврат на экран деталей челленджа
Альтернативные потоки: Отсутствуют

Приложение Б. Таблица сущностей базы данных

Таблица 10 – Сущности базы данных

№	Сущность	Поля
1	Event	id – идентификатор события event_object_id – идентификатор объекта события time – дата события scope_id – идентификатор области видимости(организации) user_id – идентификатор пользователя likes_amount – количество лайков comments_amount – количество комментариев user_liked – поставил ли лайк текущий пользователь
2	Balance	id – идентификатор баланса owner_id – идентификатор владельца income – количество полученных distr – количество доступных для раздачи expire_date – дата сгорания баланса
3	Organization	id – идентификатор name – название photo – фото head_of_department_id – идентификатор главного отдела top_id – идентификатор главы организации
4	Challenge	id – идентификатор челленджа created_at – дата создания updated_at – дата обновления name – название description – описание photo – фото scope_id – идентификатор видимости challenge_mode – настройка челленджа creator_id – идентификатор создателя creator_name– имя создателя creator_photo– аватар создателя start_balance – баланс на старте participants_count – количество участников winners_count – количество победителей algorithm_type – тип алгоритма подсчета победителя
5	Profile	id – идентификатор пользователя tg_name – ник в телеграмме photo – фото пользователя hired_at – дата найма surname – фамилия пользователя first_name – имя пользователя middle_name – отчество пользователя nickname – никнейм пользователя department_id – идентификатор отдела сотрудника organization_id – идентификатор организации сотрудника date_of_birth – дата рождения status – рабочий статус phone – номер телефона email – рабочая почта

Продолжение таблицы 10 приложения Б

№	Сущность	Поля
6	Market	id – идентификатор маркета name – название description – описание logo – логотип created_by_id – идентификатор создателя organization_owner_id – идентификатор владеющей организации thanks_period_interval – интервал периода benefits – список с товарами
7	Cart	id – идентификатор заказа benefits – список товаров в корзине price – общая стоимость created_at – дата создания updated_at – дата обновления marketplace_id – идентификатор родительского маркета categories – список категорий заказа amount – количество позиций в заказе
8	Offer	id – идентификатор предложения created_at – дата создания updated_at – дата обновления name – название description – описание photo – фото price – цена marketplace_id – идентификатор родительского маркета sold_amount – количество проданных rest_amount – осталось offer_type – тип товара selected – количество лежащих в корзине в данный момент categories – список категорий товара
9	Order	id – идентификатор заказа benefits – список товаров в заказе price – общая стоимость order_status – статус заказа transaction_id – идентификатор оплаты created_at – дата создания updated_at – дата обновления marketplace_id – идентификатор родительского маркета categories – список категорий заказа amount – количество позиций в заказе
10	Comment	id – идентификатор text – текст комментария photo – фото sticker – стикер комментария likes_amount – количество лайков у комментария object_id – идентификатор родительского объекта gif – гиф ссылка created_at – дата создания edited_at – дата обновления user_liked – поставил ли текущий пользователь лайк

Окончание таблицы 10 приложения Б

11	Like	id – идентификатор object_id – идентификатор родительского объекта user_id – идентификатор автора username – имя автора user_photo – аватар автора counter – количество лайков date_last_modified – дата последнего редактирования
12	Transaction	id – идентификатор перевода amount – размер перевода recipient_id – идентификатор получателя recipient_name – имя получателя recipient_photo – аватар получателя sender_id – идентификатор отправителя sender_name – имя отправителя sender_photo – аватар отправителя scope_id – идентификатор видимости photos – прикрепленные фотографии reason – причина перевода is_commentable – доступ комментирования tags – теги с благодарностями stickers – прикрепленные стикеры created_at – дата создания updated_at – дата обновления

Приложение В. Диаграмма компонентов UI слоя

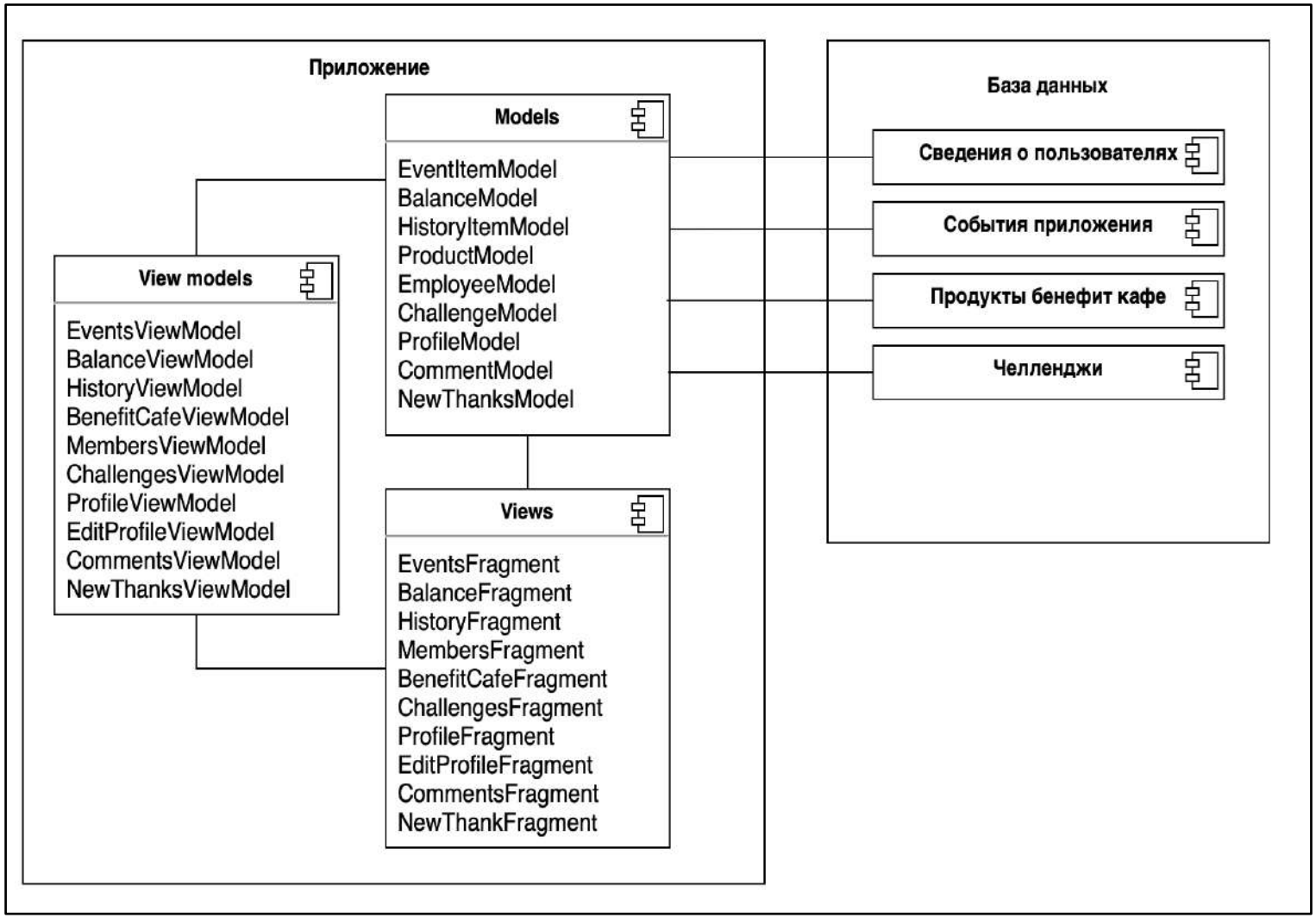

Рисунок 1 – Диаграмма компонентов UI слоя

Приложение Г. Диаграмма деятельности «Отправка благодарности»

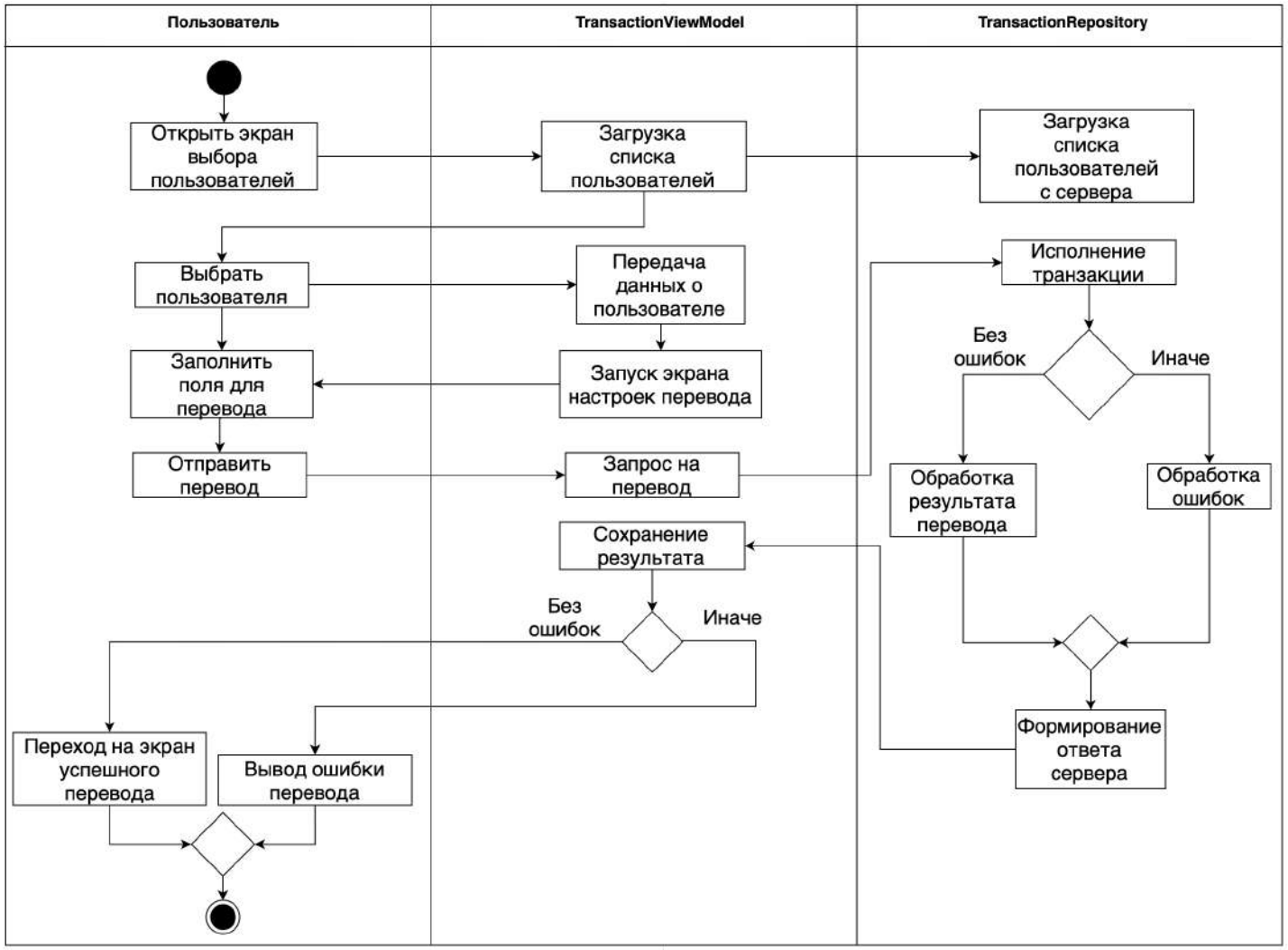

Рисунок 2 – Диаграмма деятельности «Отправка благодарности»