Тема “**Разработка приложения для владельцев домашних животных”**

Описание:

Дипломная работа посвящена разработке мобильного приложения для Android, которое помогает владельцам домашних животных находить подходящих партнёров для их питомцев. Цель проекта – повысить эффективность поиска партнёра для разведения, предоставив удобный функционал, включая авторизацию, регистрацию с подтверждением почты, просмотр и фильтрацию профилей питомцев, а также создание, редактирование и удаление этих профилей. В приложении также реализован чат для общения между владельцами и push-уведомления о новых сообщениях. Программное обеспечение должно работать на Android 5.0 и выше с поддержкой Google Services.

Критерии оценки:

1. **Устойчивость проекта** – оценка способности приложения работать в долгосрочной перспективе, включая поддержку и обновления.

2. **Коэффициент готовности** – степень завершенности проекта, включая функциональность, тестирование и аналитические инструменты.

3. **Ресурсоёмкость** – использование вычислительных ресурсов приложения (например, памяти, процессора) на устройствах Android.

4. **Коэффициент запаса** – насколько приложение способно справляться с увеличением нагрузки, например, ростом числа пользователей.

5. **Юзабилити** (удобство использования) – легкость использования приложения, включая навигацию, интуитивный интерфейс, доступность функций.

6. **Надежность** – стабильность работы приложения при различных условиях использования.

7. **Эффективность системы аналитики** – качество и полнота аналитических данных для отслеживания пользовательского поведения.

8. **Обеспечение жизненного цикла** (ЖЦ) – наличие документации и инструкций для всех этапов жизненного цикла проекта, включая поддержку и обновления.

9. **Проектные риски** – выявление и минимизация потенциальных рисков, таких как отказ серверной части или ошибки на клиентской стороне.

10. **Тестируемость** – возможность проведения автоматизированного и ручного тестирования приложения, включая покрытие тестами всех критических функций.

11. **Качество каталогизации данных** – структурированность и доступность информации, представленной в приложении (например, профили животных, фильтры поиска партнера).

12. **Доступность интерфейса –** оценка, насколько приложение адаптировано для людей с ограниченными возможностями (например, поддержка экранных читалок, удобные размеры элементов интерфейса, контрастность цветов и т.д.).

13. **Технико-эстетические параметры** – визуальная привлекательность и техническая корректность реализации (например, соответствие стандартам Android-дизайна).

14. **Реализация функциональных требований** – соответствие заявленным требованиям проекта (поиск партнера для питомца, фильтрация и сортировка).

15. **Аварийно-восстановительные меры** – наличие механизмов для восстановления работы приложения после сбоев или потерь данных.

**Практическая работа 2**

1. **Устойчивость проекта**

Для оценки устойчивости проекта рассмотрим ключевые архитектурные решения:

1. Использование Clean Architecture и паттерна MVI (Model-View-Intent) обеспечивает хорошую модульность и низкую связанность компонентов. Это важно для долгосрочной поддержки, так как позволяет вносить изменения в отдельные части системы без риска затронуть остальные компоненты. Clean Architecture разграничивает слои системы (Data, Presentation, View), что упрощает обновления и добавление нового функционала.

2. Паттерн MVI с реализацией через TEA (The Elm Architecture), обеспечивает строгий однонаправленный поток данных, что упрощает тестирование и исключает возможность зацикливания событий. Это повышает стабильность работы приложения и облегчает его поддержку в будущем.

3. Data-слой с унификацией работы с данными через базовые репозитории (SourceRepository и PagingSourceRepository) позволяет снизить вероятность ошибок и ускорить разработку новых функциональностей, что является важным аспектом для расширяемости и поддержки проекта в долгосрочной перспективе.

Таким образом, проект обладает хорошей устойчивостью благодаря использованию современных паттернов и подходов, которые способствуют лёгкости обновления, поддержки и расширяемости.

5/5

**2. Коэффициент готовности**

Для оценки "Коэффициента готовности" на основе предоставленных данных можно сделать следующие выводы:

1. Функциональные модули: Основные функции приложения реализованы, включая авторизацию, регистрацию, просмотр анкет питомцев, фильтрацию, работу с чатом и профилем пользователя. Все эти модули покрывают основной функционал приложения, что указывает на высокий уровень завершённости разработки.

2. Навигация и интерфейс: Основное меню приложения и его взаимодействие через вкладки с функциональными разделами (главная страница, чаты, профиль) также полностью реализованы и описаны.

3. Тестирование и автоматизация: Приложение активно тестируется с использованием unit- и интеграционных тестов. Наличие настроенной системы Continuous Integration на базе Github Actions, которая автоматически запускает тесты и сборку, свидетельствует о высоком уровне готовности проекта и его подготовки к развертыванию. Результаты тестирования в работе приведены не были, хотя и были описаны.

На основе этих данных можно сказать, что проект находится на высоком уровне готовности, около 90-100%, и большая часть функционала уже завершена и протестирована.

5/5

**3. Ресурсоёмкость**

Для оценки "Ресурсоёмкости" на основе предоставленных данных:

1. Требования к техническому обеспечению минимальны — приложение должно работать на смартфоне с Android и требовать подключения к Интернету. Однако нет конкретных требований по ресурсам, таким как объем оперативной памяти или процессор.

2. Тестирование и автоматизация: Систематическое тестирование и использование Continuous Integration на базе Github Actions обеспечивают проверку корректной работы приложения, но нет явного упоминания тестов, направленных на измерение производительности (например, нагрузки на процессор или потребление памяти).

На данный момент приложение кажется ресурсоэффективным, поскольку не требует специальных технических условий. Однако для более точной оценки ресурсоёмкости необходимо провести целевое тестирование производительности на разных устройствах с различными характеристиками, которое не было проведено в данной работе.

4/5

**4. Коэфициент запаса**

На основе имеющейся информации можно дать предварительную оценку "Коэффициента запаса":

1. Архитектура приложения: Использование Clean Architecture и MVI указывает на хорошую модульность и низкую связанность компонентов, что способствует расширяемости приложения. Однако информация о серверной части, масштабируемости или механизмах обработки большого числа запросов не предоставлена. Это важный аспект, так как при увеличении числа пользователей нагрузка на серверы и базу данных может резко возрасти.

2. Data-слой: Репозитории (SourceRepository и PagingSourceRepository) оптимизируют работу с данными и пагинацией, что является плюсом для масштабируемости. Тем не менее, отсутствует информация о механизмах кэширования, распределении нагрузки на серверы или архитектуре базы данных.

3. Отсутствие нагрузочного тестирования: В разделе о тестировании не указано о проведении нагрузочных тестов, которые могли бы показать, как приложение справляется с увеличением числа пользователей.

Коэффициент запаса (или коэффициент устойчивости) можно рассчитать с учетом различных факторов: производительности системы, возможности её масштабирования, текущей нагрузки и потенциальных изменений в будущем. В общем случае формула коэффициента запаса может выглядеть так:

Где К – коэфициент запаса, Рмакс – максимально выдерживаемая нагрузка, Ртек – текущая нагрузка.

На момент написания работы, опираясь на аналитику, было выявлено, что посещаемость приложения составляет 5 человек. При максимальной нагрузке больше 1000 человек. Что говорит о том, что коэфициент запаса приложения на момент написания довольно большой. Но при этом в работе информация о том, как приложение будет справляться с увеличением нагрузки, минимальна. Нет данных о серверной части и механизмах кэширования, балансировки нагрузки или оптимизации под высокие нагрузки, что оставляет вопросы о долгосрочной устойчивости под увеличивающуюся нагрузку.

3/5

**5. Юзабилити**

1. Основное меню: Приложение имеет удобное нижнее навигационное меню с тремя вкладками — Главная, Чаты и Профиль. Вкладки загружаются по мере необходимости (ленивая загрузка), что улучшает производительность и уменьшает потребление ресурсов. Навигация между экранами оптимизирована за счёт переиспользования открытых вкладок, что делает использование приложения более плавным и интуитивным.

2. Домашняя страница: Пользователю предоставлен удобный способ взаимодействия с анкетами питомцев: просмотр карточек, сохранение в избранное, фильтрация и обновление списка. Разнообразие типов полей ввода (простое поле, селектор, диапазон) упрощает взаимодействие с приложением, делая его более доступным и удобным.

3. Анкета питомца: Приложение предоставляет возможность легко взаимодействовать с анкетами питомцев — просматривать данные, добавлять в избранное, начинать диалог с владельцем. В режиме редактирования пользователь может гибко управлять информацией о своём питомце (добавлять, изменять, удалять поля), что делает интерфейс более настраиваемым под индивидуальные потребности.

4. Профиль: Профиль пользователя предоставляет доступ ко всем созданным анкетам питомцев, избранным питомцам и личной информации. Логика отображения информации заимствована из главной вкладки, что обеспечивает единообразие интерфейса и уменьшает необходимость изучать новые элементы управления.

|  |  |
| --- | --- |
|  |  |
| Рис. 12. Splash Screen | Рис. 13. Форма авторизации |

|  |
| --- |
|  |
| Рис. 22. Домашняя страница |

Приложение демонстрирует высокий уровень юзабилити за счёт интуитивного интерфейса и оптимизированной навигации.

5/5

**6. Надежность**

1. Архитектура: Использование Clean Architecture и MVI разделяет логику и интерфейс, предотвращая ошибки и зацикливание событий, что способствует стабильности работы приложения.

2. Авторизация и обработка ошибок: Компоненты для управления авторизацией и обработки ошибок (AuthInterceptor, ErrorHandlingCallAdapter) предотвращают сбои при отказах в доступе и обновлении токенов, что повышает устойчивость системы.

3. Тестирование: Unit- и интеграционные тесты с автоматической сборкой через CI выявляют ошибки на ранних стадиях, обеспечивая надежность приложения.

Приложение разработано с акцентом на стабильность и тестируемость, что способствует его надежной работе в разных условиях.

5/5

**7.** **Эффективность системы аналитики**

Аналитика приложения базируется на подходе Model-View-Intent (MVI), который обеспечивает гибкость и независимость аналитического модуля от бизнес-логики. Это позволяет легко внедрять аналитику без изменения функциональных частей системы, что является плюсом для поддерживаемости приложения.

Применяются такие сервисы, как Google Analytics для отслеживания пользовательской активности, и Firebase Crashlytics для мониторинга критических ошибок и крашей. Охвачены ключевые события, например, открытие экранов и отправка сообщений в чатах, что позволяет анализировать поведение пользователей и оперативно реагировать на проблемы.

Система аналитики разработана качественно, она охватывает важные аспекты пользовательского опыта и технические проблемы. Использование проверенных решений (Google Analytics и Firebase Crashlytics) гарантирует полноту данных для принятия решений. Однако в текущем описании нет информации о деталях пользовательской активности, таких как действия внутри экранов или поведение при взаимодействии с различными элементами приложения.

|  |
| --- |
|  |
| Рис. 40. Отчет Firebase Crashlytics |

4/5

**8. Обеспечение жизненного цикла** (ЖЦ)

Документация и инструкции для жизненного цикла проекта представлены частично. Упомянуты тестирование и автоматизация, что обеспечивает поддержку и обновления. Однако не хватает подробной информации о полном объеме документации, включая техническую документацию, руководства по эксплуатации и обновлениям. В целом, проект охватывает основные этапы жизненного цикла, но требует более полной документации для лучшего обеспечения долгосрочной поддержки.

3/5

**9. Проектные риски**

В работе не представлен специализированный раздел, посвящённый управлению проектными рисками. Тем не менее, общая архитектура приложения, основанная на принципах Clean Architecture и паттерне MVI, способствует снижению потенциальных рисков. Чистая архитектура уменьшает связность компонентов, а использование MVI позволяет эффективно отслеживать и управлять потоками данных, что повышает тестируемость и расширяемость системы.

Отсутствие явного раздела по управлению рисками в документе не снижает ценность выполнения проекта, так как грамотное проектирование архитектуры и тестирование снижают вероятность возникновения ошибок и сбоев. Однако было бы полезно увидеть детальный анализ потенциальных рисков и стратегий их управления для более полного представления о стабильности и надёжности проекта в будущем.

3/5

**10. Тестируемость**

Проект демонстрирует высокий уровень тестируемости благодаря использованию unit- и интеграционных тестов с фреймворками KoTest и Turbine. Эти инструменты позволяют писать понятные и эффективные тесты, что способствует обнаружению и исправлению ошибок на ранних этапах разработки.

Также настроена система Continuous Integration с помощью GitHub Actions, которая автоматически запускает тесты при каждом изменении кода. Это обеспечивает постоянную проверку работоспособности приложения и помогает поддерживать его качество на высоком уровне.

В общем, проект хорошо охватывает аспекты тестируемости, позволяя эффективно тестировать бизнес-логику и асинхронные процессы. Дополнительной информации о тестировании пользовательского интерфейса или специфических случаев использования было бы полезно, но в целом подход к тестированию оценивается положительно.

|  |
| --- |
|  |
| Рис. 36. Иерархия тестовой директории |

|  |
| --- |
|  |
| Рис. 37. Тест |

4/5

**11. Качество каталогизации данных**

Раздел "Data-слой" предоставляет важные детали о том, как в проекте организована работа с данными. В частности, описаны два базовых репозитория — `SourceRepository` и `PagingSourceRepository`, которые структурируют и управляют состоянием данных и страниц.

Положительные аспекты:

1. Структурированное управление состояниями данных: Использование различных состояний (Empty, Loading, Content, Fail) позволяет точно отслеживать и управлять процессами загрузки данных, что способствует качественной обработке ошибок и улучшению взаимодействия с пользователем.

2. Поддержка пагинации и отступов: `PagingSourceRepository` управляет состоянием страницы и поддерживает как пагинацию, так и загрузку данных по отступам, что упрощает работу с большими списками и улучшает пользовательский опыт.

Недостатки:

- Отсутствие информации о метаданных и индексировании: В представленном тексте не упоминается, как каталогизируются метаданные данных или обеспечивается их индексирование. Это важно для понимания полноты и эффективности системы каталогизации данных.

- Нет информации о масштабируемости: Не упомянуто, как структура данных будет масштабироваться при увеличении объема данных или количества пользователей.

Вывод: В целом, архитектура каталогизация данных представляется продуманной и хорошо организованной, с ясным подходом к управлению состояниями данных и пагинацией. Однако для более полной оценки качества каталогизации данных потребуются дополнительные сведения о метаданных, индексировании и масштабируемости системы.

4/5

**12. Доступность интерфейса**

В дипломной работе отсутствует информация о том, как приложение адаптировано для людей с ограниченными возможностями. Не указаны детали о поддержке экранных читалок, доступности элементов интерфейса, их размерах, контрастности цветов или других аспектах доступности.

Отсутствие упоминаний о доступности интерфейса в работе является значительным недостатком. Важно, чтобы приложение учитывало потребности пользователей с ограниченными возможностями, что включает поддержку экранных читалок, обеспечение удобных размеров элементов интерфейса и подходящую контрастность цветов. Без этой информации нельзя оценить, насколько приложение доступно для всех категорий пользователей, что критично для создания инклюзивного и удобного продукта.

1/5

**13. Технико-эстетические параметры**

В работе был выбран фреймворк Jetpack Compose для отрисовки экранов, что свидетельствует о стремлении к высокой визуальной привлекательности и технической корректности реализации. Использование Jetpack Compose обеспечивает следующие преимущества:

1. Разделение состояния и логики отрисовки: Это улучшает производительность приложения и упрощает поддержку, поскольку состояние экрана контролируется отдельно от логики отрисовки.

2. Оптимизация перерисовки: Jetpack Compose оптимизирует перерисовку экрана, что повышает производительность, так как перерисовываются только те участки, состояние которых изменилось.

3. Стандартизация работы с экранами: Разделение каждого экрана на функции отрисовки и открытия улучшает тестируемость и упрощает процесс разработки. Это позволяет тестировать отображение экрана независимо от бизнес-логики и использовать превью для проверки верстки.

Проект демонстрирует высокий уровень технико-эстетических параметров благодаря использованию Jetpack Compose, что обеспечивает как визуальную привлекательность, так и техническую корректность. Стандартизация и оптимизация работы с экранами способствуют улучшению производительности и тестируемости, что полностью удовлетворяет критерию.

5/5

**14. Реализация функциональных требований**

В работе были приведены следующие функциональные требования к системе:

* Ручная (по логину и паролю) авторизация;
* Автоматическая авторизация;
* Регистрация с подтверждением почты по коду;
* Список питомцев с возможностью фильтрации;
* Просмотр всей предоставленной хозяином информации о питомце;
* Создание, удаление и редактирование профиля питомца;
* Добавление питомца в избранное;
* Просмотр профиля пользователя;
* Чат с другим пользователем посредством личных сообщений;
* Push-уведомления о поступивших сообщениях, если пользователь не находится в чате.

Эти функции покрывают основные потребности целевой аудитории, направленные на поиск партнёров для питомцев и взаимодействие между владельцами.

Функциональные требования, изложенные в техническом задании, были полностью выполнены. Это свидетельствует о том, что проект соответствует поставленным целям, обеспечивая необходимую функциональность для пользователей приложения.

5/5

**15. Аварийно-восстановительные меры**

Хотя система описана как надежная и расширяемая, важный аспект аварийно-восстановительных мер не был реализован. Отсутствие механизмов резервного копирования, бекапов данных и восстановления после сбоев представляет значительный риск для стабильной работы приложения, особенно в случае сбоев на серверной стороне или потери данных.

Отсутствие аварийно-восстановительных мер, таких как бекапы и системы восстановления, является серьезным недостатком проекта. Это снижает уровень надежности приложения и увеличивает риск потери данных пользователей, что требует доработки для обеспечения устойчивости системы.

2/5

**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет информатики, математики и компьютерных наук НИУ ВШЭ-Нижний Новгород

Образовательная программа «Программная инженерия»

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Руководитель,  доцент Базовой кафедры группы компаний MERA | УТВЕРЖДАЮ  Академический руководитель образовательной программы «Программная инженерия» |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.В. Лейкин  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.В. Асеева  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |

**Выпускная квалификационная работа**

на тему: **Разработка приложения для владельцев домашних животных**

по направлению подготовки 09.03.04 «Программная инженерия»

Исполнитель: студент группы 19ПИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / М.В. Бредихин /

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.

Нижний Новгород 2023

**Содержание**

[**1.**  **Введение** 2](#_Toc134985977)

[**2.** **Обзор предметной области** 4](#_Toc134985978)

[**2.1.** **Анализ целевой аудитории** 4](#_Toc134985979)

[**2.2.** **Анализ существующих решений** 5](#_Toc134985980)

[**2.2.1.** **Доски объявлений** 5](#_Toc134985981)

[**2.2.2.** **Специализированные сайты** 6](#_Toc134985982)

[**2.2.3.** **Группы в социальных сетях** 6](#_Toc134985983)

[**3.** **Разработка технического задания** 8](#_Toc134985984)

[**3.1.** **Назначение системы** 8](#_Toc134985985)

[**3.2.** **Цели создания системы** 8](#_Toc134985986)

[**3.3.** **Требования к структуре и функционированию системы** 8](#_Toc134985987)

[**3.4.** **Требования к программному обеспечению** 8](#_Toc134985988)

[**3.5.** **Требования к техническому обеспечению** 9](#_Toc134985989)

[**4.** **Проектирование мобильного приложения** 10](#_Toc134985990)

[**4.1.** **Архитектура** 10](#_Toc134985991)

[**4.1.1.** **Model-View-Intent** 10](#_Toc134985992)

[**4.1.2.** **Clean Architecture** 12](#_Toc134985993)

[**4.1.3.** **Data-слой** 12](#_Toc134985994)

[**4.2.** **Архитектура экранов** 17](#_Toc134985995)

[**4.3.** **Диаграмма состояний** 18](#_Toc134985996)

[**5.** **Разработка мобильного приложения** 20](#_Toc134985997)

[**5.1.** **Авторизация** 20](#_Toc134985998)

[**5.2.** **Регистрация** 21](#_Toc134985999)

[**5.2.1.** **Форма регистрации** 22](#_Toc134986000)

[**5.2.2.** **Подтверждение почты** 23](#_Toc134986001)

[**5.2.3.** **Заполнение профиля** 24](#_Toc134986002)

[**5.3.** **Основное меню** 26](#_Toc134986003)

[**5.4.** **Домашняя страница** 27](#_Toc134986004)

[**5.5.** **Анкета питомца** 30](#_Toc134986005)

[**5.6.** **Список чатов** 32](#_Toc134986006)

[**5.7.** **Чат** 33](#_Toc134986007)

[**5.8.** **Профиль** 36](#_Toc134986008)

[**6.** **Тестирование и автоматизация** 37](#_Toc134986009)

[**7.** **Аналитика** 39](#_Toc134986010)

[**8.** **Заключение** 41](#_Toc134986011)

[**9.** **Источники** 42](#_Toc134986012)

# **1. Введение**

В современном мире присутствует повышенный интерес к домашним животным, люди выбирают себе питомца и заботятся о нем, как о полноценном члене своей семьи. Новички и даже опытные заводчики рано или поздно сталкиваются с проблемами, вызванными недостаточным количеством знаний в вопросах ухода за домашними животными. Особенно это касается вопросов поиска идеального партнера для своего любимца, в ходе которого хозяин сталкивается с такими трудностями, как ограниченные контакты, недостаток информации о питомцах поблизости, а также неопытность в данном вопросе. У заводчиков возникает потребность в помощнике, который сможет облегчить процесс поиска, а также ответит на все возникающие у искателя вопросы. Стремительное развитие мобильных и web-технологий позволяет нам предоставить людям такого помощника в виде мобильного приложения.

Данная дипломная работа направлена на создание удобного и надежного клиент-серверного приложения для системы Android, которое позволит пользователям быстро и эффективно находить идеальных партнеров для своих домашних животных. Оно предоставит возможность создавать профили питомцев, фильтровать их по интересующим заводчика параметрам, просматривать профили других пользователей и обмениваться сообщениями через встроенный в приложение чат.

Разработка такого приложения является актуальной задачей в настоящее время, и данная работа будет освещать все этапы разработки от проектирования системы до ее реализации, опишет подходы к тестированию и аналитике, что критически важно для создания стабильного и удобного приложения, а также раскроет перспективы его дальнейшего развития.

Разрабатываемая система состоит из серверного приложения и клиента для системы Android. Работа выполнялась в рамках командного проекта, в которой задачи и ответственность распределялись следующим образом:

* Макаров В. Д. отвечал за разработку серверного приложения;
* Я, Бредихин М. В., отвечал за создание клиента для системы Android.

Данная работа подробно описывает разработку клиентской части программной системы.

Для создания готового к релизу MVP мобильного клиента были поставлены следующие задачи:

* Изучить предметную сферу;
* Провести опрос потенциальных клиентов, целевой аудитории;
* Проанализировать существующие решения;
* Описать функциональность, проработать техническое задание;
* Спроектировать интерфейс и создать макеты экранов в Figma;
* Зафиксировать контракты клиент-серверного взаимодействия;
* Спроектировать архитектуру приложения;
* Создать приложение для системы Android.

# **2. Обзор предметной области**

Предметной областью данной работы является поиск партнеров для случки питомцев, что часто является затруднительной задачей, ввиду необходимости учитывать огромное количество параметров, таких как порода, возраст, здоровье, достижения и многие другие. Во времена огромного количества неструктурированной информации крайне сложно найти то, что сможет полностью удовлетворить требования искателя. Существуют различные ресурсы и приложения, которые помогают владельцам животных в поиске партнера для случки. Однако, многие из них имеют ограничения по породам животных, не предоставляют достаточно информации о животных или не обеспечивают удобный обмен сообщениями между пользователями.

## **2.1. Анализ целевой аудитории**

Перед началом работы мы выделили три основные категории пользователей, потребности которых мы могли бы удовлетворить с помощью нашего приложения. Следующим этапом было проведение проблемных интервью с несколькими участниками из каждой категории. В ходе анализа ответов наших респондентов были сделаны выводы об основных болях наших потенциальных клиентов.

В качестве целевой аудитории мы определили следующие категории пользователей:

* Семьи, владеющие одним или более питомцами более двух лет;
* Люди, приютившие своего первого питомца не ранее года назад;
* Заводчики, для которых питомец является основным или дополнительным средством заработка.

Из каждой категории мы выбрали 10 респондентов и провели с ними проблемные интервью, в ходе которых выявили потребность каждой из них в нашем продукте:

* 55 процентов семей из первой категории задумывались о случке с точки зрения заработка;
* 80 процентов людей с первым питомцем понимают необходимость поиска партнера, но не рассматривают это в качестве заработка;
* 70 процентов заводчиков размещают свои объявления только на Avito, так как там больше всего клиентов.

Также были выявлены основные проблемы, с которыми сталкивается наша целевая аудитория:

* 50 процентов семей с опытом владения домашними питомцами отказались от заработка на случке из-за сложности поиска клиентов;
* 75 процентов опытных семей отметили слишком большое количество объявлений, при скудных возможностях фильтрации;
* 60 процентов хозяев-новичков обеспокоены недостатком информации о потенциальных партнерах в объявлениях, от чего боятся навредить своему питомцу;
* 40 процентов заводчиков считают, что смесь рекомендаций из разных категорий объявлений на досках объявлений приводит к снижению спроса в наименее популярных категориях (к которым они отнесли поиск партнера для питомца).

По данным сервиса Яндекс Wordstat более 56 процентов поисковых запросов, включающих в себя вязку питомцев, являются вопросами о проведении данного процесса. Более того, в среднем более 43 процентов поисковых запросов о конкретном виде питомцев являются вопросами об аспектах ухода за ними, что говорит о необходимости дополнительного информирования пользователей.

Однако, реализовать данную функциональность было решено за рамками MVP, что в свою очередь ставит перед нами задачу учесть дальнейшее расширение системы в данном направлении при проектировании.

## **2.2. Анализ существующих решений**

В ходе анализа рынка были найдены несколько представителей прямых и косвенных конкурентов для нашего продукта, которые мы объединили в 3 категории:

* Доски объявлений (Avito, Юла, EDC.SALE);
* Специализированные сайты (ru-pets, p2pets);
* Группы в социальных сетях.

### **2.2.1. Доски объявлений**

Доски являются самыми популярными площадками для размещения объявлений, за счет их универсальности и большой аудитории. С точки зрения поиска партнера для питомца, данные сервисы предоставляют весь необходимый функционал для размещения и поиска партнера, такой как описание, структурирование информации, фотографии и фильтры, однако имеют ряд ограничений из-за своей универсальности.

Плюсы:

* Большая аудитория;
* Возможность продвигать свое объявление;
* Лимит доверия благодаря популярности;
* Система отзывов и рейтинга.

Минусы:

* Ограниченная структуризация информации. За счет универсальности, часть важной информации идет в текстовое описание;
* Ограниченные фильтры за счет универсальности;
* Смешивание разных категорий объявлений в одной ленте, смещая внимание пользователей на другие сферы;
* Невозможность более глубокого развития в конкретной специализации.

### **2.2.2. Специализированные сайты**

Специализированные сайты – менее популярный способ поиска партнера для своего питомца, хотя они и могут предоставить более обширный функционал в своей сфере.

Плюсы:

* Предоставление дополнительных услуг: консультации, сопутствующие товары;
* Структуризация информации.

Минусы:

* В большинстве случаев устаревший дизайн;
* Низкая аудитория;
* Отсутствие на сайте возможности связаться с автором объявления (только по телефону или во внешних мессенджерах).

Также в ходе анализа рынка не было найдено ни одного специализированного мобильного приложения, которое прямо или косвенно предоставляло возможность поиска партнера для питомца.

### **2.2.3. Группы в социальных сетях**

В социальных сетях, таких как ВКонтакте или Одноклассники, существует множество сообществ по различным тематикам, в том числе и для поиска партнера для домашних животных. Данный способ поиска является скорее возможностью случайно найти подходящее объявление, чем удобным инструментом для целенаправленного поиска.

Плюсы:

* Большая аудитория социальной сети;
* Возможность задать вопросы;
* Возможность найти единомышленников и обсудить любые проблемы.

Минусы:

* Отсутствие структуризации информации;
* Отсутствие поиска и фильтрации;
* Множество отвлекающих от поиска факторов, таких как публикации других сообществ, рекламные посты или бесконтрольное обсуждение;
* Низкий контроль за мошенниками;
* Возможность повышения негативных настроений и запугивания новичков в ходе обсуждений.

# **3. Разработка технического задания**

В результате опроса целевой аудитории и анализа прямых и косвенных конкурентов была выявлена основная функциональность, которая должна быть реализована в рамках MVP, а также было разработано техническое задание.

## **3.1. Назначение системы**

Мобильное приложение для операционной системы Android разрабатывается для помощи хозяевам домашних животных в поиске партнеров для их питомцев.

## **3.2. Цели создания системы**

Сервис создается с целью повышения эффективности бизнеса по разведению домашних животных, а также для помощи в вопросах поиска партнера для питомца.

## **3.3. Требования к структуре и функционированию системы**

В приложении должны быть реализованы следующие функции:

* Ручная (по логину и паролю) авторизация;
* Автоматическая авторизация;
* Регистрация с подтверждением почты по коду;
* Список питомцев с возможностью фильтрации;
* Просмотр всей предоставленной хозяином информации о питомце;
* Создание, удаление и редактирование профиля питомца;
* Добавление питомца в избранное;
* Просмотр профиля пользователя;
* Чат с другим пользователем посредством личных сообщений;
* Push-уведомления о поступивших сообщениях, если пользователь не находится в чате.

## **3.4. Требования к программному обеспечению**

Программное обеспечение пользователя должно соответствовать следующим требованиям:

* Операционная система Android версии не ниже 5.0;
* Наличие Google Services.

## **3.5. Требования к техническому обеспечению**

Техническое обеспечение пользователя должно удовлетворять следующим условиям:

* Смартфон на базе операционной системы Android;
* Имеется доступ в сеть Интернет.

# **4. Проектирование мобильного приложения**

## **4.1. Архитектура**

В основу архитектуры всего проекта легли принципы Clean Architecture и паттерн MVI (Model-View-Intent), которые позволяют наиболее прозрачно отслеживать и контролировать потоки данных в приложении, повышают тестируемость и расширяемость системы.

### **4.1.1. Model-View-Intent**

Идея паттерна MVI заключается в полном обособлении View от бизнес-логики. Задача View-компонента ограничивается в уведомлении Model-компонента о действиях пользователя посредствам Intent-событий, а также в отрисовке состояния экрана, основанном на изменении Model-компонента. Данное поведение полностью удовлетворяет требованиям UDF (Unidirectional Data Flow) за счет отсутствия каких-либо двусторонних связей между компонентами.

|  |
| --- |
|  |
| Рис. 1. Model-View-Intent |

В качестве конкретной реализации данного паттерна была выбрана TEA (The Elm Architecture), созданная на основе одноименного паттерна проектирования для web-приложений. Данная реализация исключает возможность зацикливания событий, а также выделяет отдельные интерфейсы для взаимодействия с Data-слоями.

Поскольку в системе Android *Intent* является зарезервированной системой сущностью, во всех дальнейших схемах и примерах кода оно будет заменено на *Event.*

На схеме ниже представлен принцип работы и направление потоков данных в реализации TEA.

|  |
| --- |
|  |
| Рис. 2. The Elm Architecture |

При взаимодействии пользователя с экраном в *Reducer* отправляется соответствующий *UiEvent,* на основании которого определяется необходимость изменения состояния (*State*) или/и доступа к данным. В случае доступа или каких-либо операций с данными отправляется определенный объект *Command*, который обрабатывается с помощью одного из *Actor*’ов. В результате в *Reducer* возвращается объект *Event* (внутреннее событие), замыкая цикл. Стоит отметить, что к репозиториям, то есть к Data-слою имеет доступ только *Actor*, что позволяет разграничить бизнес-логику и обработку данных.

Более того, отделение состояния от View-слоя повышает тестируемость кода, поскольку для проверки корректности состояния экрана в ответ на действия пользователя достаточно использовать простой тест, отправляя *UiEvent* и проверяя *State* на соответствие ожидаемому.

### **4.1.2. Clean Architecture**

Общая архитектура проекта соответствует правилам Чистой архитектуры, которые позволяют понизить связность системы путем запрета наличия зависимостей наименее часто изменяемых слоев от наиболее изменяемых. В приложении выделены 3 слоя со следующей зоной ответственности:

* Data – доступ и подготовка данных для бизнес-логики;
* Presentation – контроль состояния и бизнес-логика;
* View (UI) – отображение состояния на экране, передача событий.

Взаимодействие слоев между собой происходит на основе принципов MVI, описанных выше. Отдельного внимания стоит проектирование Data-слоя, который внутри также построен на MVI, где в качестве View выступает сам репозиторий. Сделано это для предоставления Presentation-слою стандартизированного и удобного api для доступа к данным, инкапсулируя источник данных и способы их обработки для минимизации зависимостей и потенциальных изменений при обновлении контрактов или при добавлении дополнительной обработки.

### **4.1.3. Data-слой**

Поскольку проект является клиент-серверным приложением, в задачи Data-слоя входит не только предоставление самих данных, но также информации об их состоянии. Пример такого состояния – ошибка загрузки, о которой приложение должно сообщить пользователю, чтобы он смог предпринять какие-либо меры.

Для унификации работы с загружаемыми данными были созданы два базовых репозитория:

* *SourceRepository* для работы с обычными данными;
* *PagingSourceRepository* для работы с пагинированными списками.

*SourceRepository* предоставляет информацию о шести возможных состояниях данных (рис. 3):

1. *Empty* – состояние инициализации, до запроса на обновление данных;
2. *Loading* – загрузка данных после запроса, но до их получения;
3. *Loading.WithContent* – повторная загрузка данных при условии, что предыдущий запрос был успешен;
4. *Loading.WithError* – повторная загрузка данных при условии, что предыдущий запрос завершился с ошибкой;
5. *Content* – успешное завершение запроса, данные готовы к работе;
6. *Fail* – не удалось выполнить запрос.

|  |
| --- |
|  |
| Рис. 3. DataState |

Интерфейс взаимодействия *SourceRepository* содержит два метода (рис. 4):

* *invalidate()* – обновление данных;
* *get()* – получение потока данных.

|  |
| --- |
|  |
| Рис. 4. SourceRepository |

Схема работы *SourceRepository* представлена на рисунке ниже.

|  |
| --- |
|  |
| Рис. 5. Схема SourceRepository |

*PagingSourceRepository* предоставляет информацию о семи возможных состояниях данных:

1. *Initial.Empty* – начальное состояние, до запроса на обновление данных;
2. *Initial.Loading* – загрузка первой страницы данных;
3. *Initial.Fail* – не удалось выполнить запрос на первую страницу данных;
4. *Paged.Refresh* – загрузка первой страницы данных при условии, что ранее уже была успешно загружена хотя бы одна страница;
5. *Paged.Loading* – загрузка следующей страницы данных;
6. *Paged.Content* – страница успешно загружена;
7. *Paged.Fail* – не удалось выполнить запрос на следующую страницу.

|  |
| --- |
|  |
| Рис. 6. PagingDataState |

Интерфейс взаимодействия *PagingSourceRepository* содержит три метода (рис. 7):

* *invalidate()* – полное обновление данных;
* *loadPage()* – загрузка следующей страницы;
* *get()* – получение потока данных.

|  |
| --- |
|  |
| Рис. 7. PagingSourceRepository |

Схема работы *PagingSourceRepository* представлена на рисунке ниже.

|  |
| --- |
|  |
| Рис. 8. Схема PagingSourceRepository |

Помимо состояния самих данных *PagingSourceRepository* также контролирует состояние страницы (*PageState*), которое содержит в себе номер следующей страницы и ее размер, что позволяет более низким слоям не заботиться о том, какую страницу необходимо загрузить.

В случае успешного выполнения запроса, *Reducer* увеличивает номер страницы на единицу и объединяет полученную страницу с ранее загруженными данными.

В случае ошибки при выполнении запроса состояние страницы остается неизменным, то есть следующий вызов *loadPage()* приведет к загрузке той же страницы.

Также поддерживается загрузка страницы по отступу, а не номеру страницы. В качестве отступа может использоваться размер уже загруженного списка. Для имплементации данной логики были использованы отдельные реализации *PageState* и *Reducer*.

Два вышеописанных базовых репозитория позволяют повысить скорость разработки, снизить вероятность ошибок и повысить тестируемость за счет переиспользования большого количества логики обработки данных по всему приложению.

## **4.2. Архитектура экранов**

В качестве фреймворка для отрисовки экранов был выбран Jetpack Compose, поскольку он позволяет полностью вынести состояние экрана за пределы логики отрисовки, делегируя его контроль другим сущностям приложения. Более того, данный фреймворк оптимизирован под отслеживание изменения состояния. Данная оптимизация позволяет не перерисовывать те участки экрана, состояние которых не изменилось. что значительно повышает производительность приложения.

Для ускорения разработки и повышения тестируемости кода было решено также стандартизировать работу с экранами. Каждый экран приложения был разделен на 2 функции:

1. Функция отрисовки экрана, принимающая состояние и функции для каждого способа взаимодействия пользователя;
2. Функция открытия экрана, содержащая в себе параметры и инициализирующая соответствующий экрану *Store* (MVI).

|  |
| --- |
|  |
| Рис. 9. Архитектура экрана |

Данный подход имеет ряд преимуществ:

* Все возможные взаимодействия пользователя с экраном находятся в одном месте;
* Функцию отрисовки экрана можно тестировать отдельно от бизнес-логики, изменяя состояние и отслеживая корректность отрисовки;
* Для функции отрисовки можно использовать превью, чтобы проверить корректность верстки в различных состояниях без установки на устройство.

## **4.3. Диаграмма состояний**

Диаграмма состояний отображает переходы системы между различными состояниями и используется для моделирования и проработки жизненного цикла системы. На рисунке ниже представлена диаграмма состояний приложения до момента, как пользователь получает доступ к контенту.

|  |
| --- |
|  |
| Рис. 10. Диаграмма состояний до доступа к контенту |

На диаграмме отражена возможность автоматической авторизации в случае, если пользователь уже был ранее авторизован. Если же пользователь авторизован не был, или авторизация завершилась неуспешно по причине устаревшего токена, пользователь попадает на экран авторизации, где может ввести свои данные или зарегистрироваться. В ходе регистрации пользователь должен заполнить все поля с уникальными почтой и никнеймом, после чего он должен подтвердить почту с помощью отправленного на нее кода. Перед предоставлением доступа к контенту приложение в любом случае должно проверить, заполнен ли профиль пользователя и только после его заполнения, доступ к контенту приложения будет открыт.

# **5. Разработка мобильного приложения**

## **5.1. Авторизация**

Поскольку приложение поддерживает только авторизованный режим, было необходимо продумать систему авторизации клиент-серверных запросов. Было принято решение использовать 3 контролирующих авторизацию компонента:

* *AuthInterceptor* – перехватывает запросы на сервер и подставляет токен авторизации в заголовок, также отвечает за обновление токена при получении его с сервера;
* *ErrorHandlingCallAdapter* – конвертирует ошибки сервера в известные приложению *Exception*, при получении ошибки *Access Denied* разлогинивает пользователя;
* *AuthController* – обновляет токены в хранилищах, инициализирует пользователя в аналитическом сервисе, контролирует запрос разлогина и сохраняет в кэш данные текущего пользователя.

Взаимодействуя данные компоненты агрегируют в себя всю логику авторизации клиент-серверного взаимодействия. Принцип работы компонентов представлен на схеме ниже.

|  |
| --- |
|  |
| Рис. 11. Схема компонентов авторизации |

Представленное решение позволяет инкапсулировать и централизовать всю логику авторизации в отдельных от основной бизнес-логики модулях, благодаря чему уменьшается связность компонентов и вероятность ошибок, а также облегчается дальнейшая поддержка данного слоя приложения.

С точки зрения пользователя авторизация представлена двумя состояниями:

* Splash Screen, во время которого происходит попытка автоматической авторизации (рис. 12);
* Форма для ручной авторизации (рис. 13).

|  |  |
| --- | --- |
|  |  |
| Рис. 12. Splash Screen | Рис. 13. Форма авторизации |

## **5.2. Регистрация**

Создание нового аккаунта реализовано с помощью флоу из трех экранов:

* Форма регистрации (рис. 14);
* Подтверждение почты (рис. 17);
* Заполнение данных о пользователе (рис. 18).

Каждый последующий экран процесса создания нового пользователя недоступен без успешного прохождения предыдущего.

### **5.2.1. Форма регистрации**

Первый экран во флоу, форма регистрации, представляет собой 4 поля ввода для указания основной информации, необходимой для создания нового пользователя: никнейм, электронная почта и пароль с подтверждением.

|  |
| --- |
|  |
| Рис. 14. Форма регистрации |

Перед отправкой форма валидируется компонентом, спроектированным с помощью паттерна Компоновщик, который возвращает список непрошедших проверку полей. Принцип работы системы валидации представлен на рисунке ниже.

|  |
| --- |
|  |
| Рис. 15. Схема валидации |

Были выделены следующие критерии для успешного прохождения валидации:

* Email соответствует регулярному выражению на рис. 16;
* Никнейм пользователя не пуст и не состоит из пробелов;
* Пароли совпадают, не пусты и не состоят из пробелов.

|  |
| --- |
|  |
| Рис. 16. Регулярное выражение валидации Email |

### **5.2.2. Подтверждение почты**

Следующий этап в регистрации – это подтверждение почты. На данном экране пользователь должен ввести код, пришедший ему на указанную на предыдущем шаге почту. Для напоминания о том, какая именно была использована почта, на экране она также представлена в маскированном виде. Дополнительно пользователь имеет возможность переслать код в случае непредвиденных ошибок.

|  |
| --- |
|  |
| Рис. 17. Подтверждение почты |

Перед отправкой поле также валидируется, приложение не позволит отправить код, состоящий не из цифр. В случае невалидности кода, либо при неудачном подтверждении почты со стороны сервера поле выделяется красным.

### **5.2.3. Заполнение профиля**

Заполнение профиля является заключительным шагом его создания. Экран имеет одно обязательное поле – *Имя*, остальные поля обязательны, если указано одно и более поле после него. К примеру, если указана *Улица*, то поля *Страна* и *Город* становятся обязательными, а поле *Дом* все еще остается опциональным.

|  |
| --- |
|  |
| Рис. 18. Заполнение профиля |

Также присутствует возможность загрузить аватар профиля из памяти устройства. Если аватар уже выбран, при клике на него будет предложено обновить или удалить его.

|  |
| --- |
|  |
| Рис. 19. Клик по аватару |

Метод изменения профиля на сервере находится в авторизованной зоне, поэтому после успешного подтверждения почты происходит автоматическая авторизация по введенным ранее данным.

Экран заполнения профиля также показывается после авторизации в случае, если у текущего пользователя не заполнено единственное обязательное поле – *Имя*.

По клику на *Создать* происходит валидация полей по описанным выше правилам, после чего начинается параллельная отправка аватара и заполненных полей на сервер. После успешного заполнения профиля пользователь получает доступ к основному функционалу приложения.

Этот же экран используется и для изменения данных о пользователе. В данном случае в поля изначально подставляются данные, пришедшие с сервера.

## **5.3. Основное меню**

Основная навигация в приложении представлена в виде нижнего навигационного меню, состоящего из трех вкладок: *Главная*, *Чаты* и *Профиль*.

|  |
| --- |
|  |
| Рис. 20. Основное меню |

Для оптимизации ресурсов используется ленивая загрузка вкладок, то есть если пользователь не перешел в какой-либо пункт меню, он не будет загружен.

Сохранение в backstack переходов между вкладками осуществляется по принципу переиспользования уже открытых вкладок, схема которого представлена на рис. 21. На представленной схеме начальное состояние – последовательно открытые вкладки *Главная*, *Чаты* и *Профиль.* Конечное состояние – переход на вкладку *Чатов,* в результате которого соответствующий экран перемещается на верх стека.

|  |
| --- |
|  |
| Рис. 21. Принцип работы основного меню |

## **5.4. Домашняя страница**

Первым экраном, на который попадает пользователь после успешной авторизации, является список созданных другими пользователями анкет питомцев. Каждый элемент списка представляет собой карточку с краткой информацией о потенциальном партнере.

У пользователя есть следующие варианты взаимодействия со списком:

* Просмотр всей информации о питомце по клику на его карточку;
* Сохранение анкеты в избранное по клику на сердце;
* Фильтрация списка по нажатию на иконку фильтрации;
* Обновление списка с помощью Pull-to-Refresh.

|  |
| --- |
|  |
| Рис. 22. Домашняя страница |

Список является пагинированным с размером страницы в 30 элементов. Запрос на загрузку следующей страницы происходит по достижению пользователем 60-го элемента до конца списка для создания ощущения бесшовности списка.

Пагинация реализована с помощью описанного ранее *PagingSourceRepository* с подгрузкой элементов по номеру и размеру страницы.

Для указания фильтров был создан экран, основанный на Model Driven UI подходе. Данный подход был выбран для возможности быстрого добавления или удаления существующих фильтров.

Были выделены три типа полей с общим интерфейсом, из которых в дальнейшем строится пользовательский интерфейс:

* Простое поле ввода;
* Селектор с ограниченным числом вариантов;
* Поля ввода числового диапазона.

|  |
| --- |
|  |
| Рис. 23. Фильтры |

После указания всех фильтров и их типов, они конвертируются в список элементов, каждый из которых отрисовывается по соответствующему ему паттерну.

Данное решение позволило значительно облегчить добавление новых фильтров, именно этот функционал однозначно будет расширяться в будущем.

|  |
| --- |
|  |
| Рис. 24. FiltersState |

После сохранения фильтров, они конвертируются в *PetListKeyModel,* которая передается на сервер для фильтрации списка питомцев.

|  |
| --- |
|  |
| Рис. 25. Маппер |

## **5.5. Анкета питомца**

По клику на питомца в списке пользователь попадает на экран, где представлена вся информация о потенциальном партнере, предоставленная его хозяином. С этого экрана можно добавить питомца в избранное и начать диалог с его владельцем.

|  |
| --- |
|  |
| Рис. 26. Анкета питомца |

По аналогии с фильтрами, данные для этого экрана изначально приходят в виде цельной модели, которая для дальнейшей отрисовки и обработки разбивается на список из полей пяти типов:

1. *SimplePetField* – простое текстовое поле;
2. *EnumPetField* – селектор с ограниченным числом значений;
3. *AchievementPetField* – поле формата ключ-значение для достижений;
4. *ListPetField* – список текстовых полей;
5. *DatePetField* – поле с датой.

Если поле в модели от сервера имеет значение *null*, данное поле в список не добавляется.

Аналогично фильтрам, поля отрисовываются по соответствующим им паттернам.

Помимо просмотра данных о питомце, пользователь также может создать или изменить анкету своего питомца. Для данной задачи была использована та же реализация presentation-слоя, был заменен только UI, что в свою очередь демонстрирует гибкость MVI-подхода.

В режиме редактирования пользователь имеет возможность изменить значение любого поля, добавить и удалить одно или несколько необязательных полей, а также удалить или скрыть анкету из общего доступа.

|  |  |
| --- | --- |
|  |  |
| Рис. 27. Изменение анкеты | Рис. 28. Добавление полей |

|  |
| --- |
|  |
| Рис. 29. Необязательные поля |

По клику на *Сохранить* список валидируется на предмет наличия пустых или невалидных значений, затем преобразуются в модель для обновления данных на сервере.

## **5.6. Список чатов**

Второй вкладкой в основном меню является список чатов, содержащий в себе все диалоги, в которые пользователь когда-либо заходил, даже если сообщения в данном чате отсутствуют.

|  |
| --- |
|  |
| Рис. 30. Список чатов |

Поскольку после открытия экрана новые сообщения могут обновляться, для поддержки актуальности было решено использовать два источника данных:

* *SourceRepository* со списком чатов с сервера;
* *MessageRoomRepository*, уведомляющий список о последнем полученном сообщении.

При получении нового сообщения оно с помощью *MessageRoomRepository* передается списку чатов для обновления отображаемого сообщения и пересортировки загруженного ранее списка.

Также пользователь имеет возможность обновить чаты самостоятельно, с помощью Pull-to-Refresh.

## **5.7. Чат**

Чат представляет собой приватную комнату для обмена текстовыми сообщениями между двумя пользователями.

|  |
| --- |
|  |
| Рис. 31. Чат |

Для корректной работы диалога были использованы три источника данных:

* Сохраненные на сервере сообщения, полученные с помощью REST API;
* Кэш полученных через *WebSocket* сообщений при открытом чате;
* Кэш отправленных, но еще не доставленных сообщений.

В реальном времени все сообщения, в том числе и отправленные текущим пользователем, приходят на устройство с помощью *WebSocket*. Данный подход позволил точно определить время, когда сообщение было доставлено на сервер, чтобы удалить его из кэша отправленных.

Для отображения сообщений три вышеописанных источника данных объединяются по следующим правилам:

* За основу списка берутся сообщения, полученные с помощью REST API;
* Отправленные и пришедшие из *WebSocket* сообщения объединяются в один список и сортируются по дате отправления;
* Получившийся список ставится перед списком с REST API.

|  |
| --- |
|  |
| Рис. 32. Схема работы Data-слоя чата |

Список сообщений, сохраненных на сервере до открытия диалога, представляет собой пагинированный список, реализованный с помощью описанного выше *PagingSourceRepository.*

Поскольку сообщения приходят с сервера с двух источников данных, подход с загрузкой сообщений по номеру страницы приводил к дублированию, так как между запросами количество сообщений на сервере менялось. Было решено использовать подход с пагинированием по отступу, за который была принята сумма количества пришедших из *WebSocket* и уже полученных через REST API сообщений.

В случае если пользователь не находится в диалоге, он получает уведомление посредством Firebase Cloud Messaging.

Когда приложение открыто, уведомления перехватываются с помощью *FirebaseMessagingService* и фильтруются для исключения показа нотификации в открытом диалоге. Происходит это с помощью идентификатора комнаты, указанном в пришедшем уведомлении, и хранящегося в *MessageRoomRepository* идентификатора текущего открытого чата.

## **5.8. Профиль**

Последней вкладкой основного меню является профиль, в котором пользователь видит все созданные им анкеты питомцев, его избранных питомцев, а также указанную о пользователе информацию.

|  |  |
| --- | --- |
|  |  |
| Рис. 33. Мой профиль | Рис. 34. Избранное |

На данном экране без изменений переиспользуется функционал списка питомцев с главной вкладки приложения. Помимо просмотра информации, пользователь также имеет возможность выйти из аккаунта, либо изменить введенные ранее персональные данные. Для изменения данных о пользователе используется экран создания профиля, описанный ранее.

Визуальное отображение экрана отличается в случае, если открыт профиль не текущего пользователя, чего можно добиться, кликнув на фотографию собеседника в чате. На странице чужого аккаунта недоступны его избранные анкеты, а также нет доступа к изменению данных пользователя.

|  |
| --- |
|  |
| Рис. 35. Чужой профиль |

# **6. Тестирование и автоматизация**

Для тестирования функционала используются unit- и интеграционные тесты, реализованные с помощью фреймворка KoTest.

Поскольку приложение полностью построено на MVI и асинхронной обработке событий с помощью Coroutines Flow, было принято решение использовать фреймворк Turbine. Данный фреймворк предназначен для удобного тестирования потоков данных, что в свою очередь позволило писать максимально читаемые и понятные тесты.

Также было решено отказаться от использования фреймворков, позволяющих мокировать запросы и доступы к ресурсам, поскольку они имеют высокую привязанность к версии языка и часто конфликтуют с другими инструментами. Вместо этого был использован подход с тестовыми реализациями интерфейсов, которые имеют доступ к ресурсам или сети. Данные реализации должны имитировать работу, отдавая готовые данные. Активное использование паттерна Dependency Injection в приложении позволило без трудозатрат реализовать данный подход. На рисунках ниже представлен пример иерархии тестовой директории, а также один из тестов.

|  |
| --- |
|  |
| Рис. 36. Иерархия тестовой директории |

|  |
| --- |
|  |
| Рис. 37. Тест |

Для автоматического тестирования и сборки приложения в репозитории проекта была настроена система Continuous Integration на базе Github Actions. Данная система запускает все тесты в приложении на каждое событие Pull Request или push в main-ветку. В случае успешного прохождения тестирования, запускается сборка приложения, результат которой, готовая к установке debug-сборка, сохраняется в артефакты.

**7. Аналитика**

Одной из важнейших частей MVP приложения является аналитика, поскольку она позволяет оценить и улучшить пользовательский опыт, быстро реагировать и исправлять ошибки, беспокоящие пользователей, а также строить планы дальнейшего развития продукта.

Архитектура MVI позволяет легко встроить в себя аналитический модуль к любой функциональности приложения, не внося никаких изменений в существующую бизнес-логику. Схема внедрения аналитического сервиса в проект представлена на рисунке ниже.

|  |
| --- |
|  |
| Рис. 38. Интеграция аналитики в архитектуру |

Представленный подход к аналитике имеет ряд преимуществ:

* Не вносится никаких изменений в существующую бизнес-логику;
* Аналитика может содержать любую дополнительную логику (замеры, счетчики и так далее);
* Аналитика может основываться на актуальном на момент события состоянии системы;
* Аналитика имеет обозначенный в *AnalyticsController* контракт;
* Аналитика не зависит от используемого сервиса;
* Аналитический сервис может быть изменен без внесения изменения в сбор данных.

На данный момент аналитикой покрыто открытие каждого экрана, а также потенциально проблемные места, такие как отправка сообщений в чатах, для аналитики которых измеряется время между отправкой и получением сообщения из *WebSocket*. Описанный выше подход, совместно с использованием MVI в основе архитектуры приложения позволяет покрыть аналитикой любое событие, возникающее в системе.

На данный момент в качестве аналитического сервиса используется Google Analytics, поскольку данный сервис бесплатно предоставляет всю необходимую о событии и пользователях информацию.

|  |
| --- |
|  |
| Рис. 39. График открытий домашней страницы |

Также к приложению был подключен сервис Firebase Crashlytics, который позволяет отслеживать возникновение критических ошибок и крашей в приложении на устройствах пользователей. Данный инструмент позволит нам отслеживать и быстро реагировать на возникающие у пользователей проблемы.

|  |
| --- |
|  |
| Рис. 40. Отчет Firebase Crashlytics |

# **8. Заключение**

Результатом работы является готовое к релизу MVP Android приложение, предназначенное для удобного поиска партнеров для домашних животных. Приложение полностью удовлетворяет поставленным требованиям. Более того в приложения интегрирована аналитика пользовательского опыта, которая позволит в дальнейшем строить и корректировать планы развития проекта.

В процессе написания данной дипломной работы был произведен анализ предметной области, в том числе интервьюирование потенциальных клиентов на предмет определения актуальности нашего продукта. Было полностью спроектировано и реализовано мобильное приложение с учетом наиболее вероятных направлений дальнейшего развития проекта. Приложение было покрыто тестами, в результате чего были исправлены ошибки в бизнес-логике. Проект выполняет все поставленные перед ним задачи.

В качестве дальнейшего развития проекта планируется:

* Обновление визуального оформления на более современное;
* Увеличенные количества поддерживаемых видов животных;
* Расширение фильтров;
* Развитие функционала чата;
* Интеграция гайдов для начинающих владельцев питомцев.

Ссылка на репозиторий проекта: <https://github.com/VadMack/Petter>

# **9. Источники**

1. Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Дж. Паттерны Объектно-ориентированного проектирования, *Питер*, 2022, 427 с.
2. Мартин, Р. Чистый код, *Питер,* 2022, 463 с.
3. Мартин, Р. Чистая архитектура, *Питер,* 2022, 516 с.
4. Android Documentation [Электронный ресурс]  
   <https://developer.android.com/docs>
5. Appyx Documentation [Электронный ресурс]  
   <https://bumble-tech.github.io/appyx/>
6. Coil Documentation [Электронный ресурс]  
   <https://coil-kt.github.io/coil/>
7. Coroutines Documentation [Электронный ресурс]  
   <https://kotlinlang.org/docs/coroutines-overview.html>
8. Coroutines Flow Documentation [Электронный ресурс]  
   <https://kotlinlang.org/docs/flow.html>
9. Jetpack Compose Documentation [Электронный ресурс]  
   <https://developer.android.com/jetpack/compose>
10. KoTest Documentation [Электронный ресурс]  
    <https://kotest.io/>
11. Kotlinx Serialization Documentation [Электронный ресурс]  
    <https://kotlinlang.org/docs/serialization.html>
12. Retrofit Documentation [Электронный ресурс]  
    <https://square.github.io/retrofit/>
13. Room Database Documentation [Электронный ресурс]  
    <https://developer.android.com/jetpack/androidx/releases/room>
14. Turbine Documentation [Электронный ресурс]  
    <https://github.com/cashapp/turbine>