МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки)

Разработка программного обеспечения
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка мобильного приложения для ветеринарной клиники

Обучающийся К.П. Авешникова
(Инициалы Фамилия) (личная подпись)

Научный руководитель канд. техн. наук, доцент, О.В. Аникина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Название темы бакалаврской работы: «Разработка мобильного приложения для ветеринарной клиники».

Цель данной выпускной квалификационной работы заключается в разработке мобильного приложения, которое обеспечит пользователям легкий доступ к услугам ветеринарной клиники для записи на прием.

Актуальность данного исследования определяется высоким интересом к ветеринарным услугам и необходимостью предоставить владельцам домашних животных удобные и эффективные инструменты. Структура работы включает введение, три главы, заключение и приложения.

Выпускная квалификационная работа состоит из нескольких частей, каждая из которых посвящена разным аспектам разработки мобильного приложения для ветеринарной клиники.

В первой главе анализируется необходимость создания подобного приложения, включая проблемы доступа владельцев животных к необходимой информации и услугам, а также повышение качества обслуживания клиентов.

Вторая глава фокусируется на проектировании мобильного приложения для ветеринарной клиники. Разработка для Android осуществляется с использованием языков Java и Kotlin [1], которые наилучшим образом подходят для создания мобильных решений. В этой части работы определяются ключевые требования и разрабатываются модели данных. Также рассматривается архитектура программного продукта и формируется структура создаваемого мобильного приложения.

Третья глава охватывает процесс разработки приложения, включая проектирование пользовательского интерфейса, создание отдельных модулей для записи на прием и интеграцию интерактивных элементов. В ней также представляется методология реализации проекта.

Ключевые слова: разработка мобильного приложения, база данных СУБД SQLite, kotlin, java, ветеринарная клиника, питомцы.

Оглавление

Введение

В современном мире забота о домашних животных занимает все более важное место в жизни людей, что, в свою очередь, предъявляет повышенные требования к качеству и доступности ветеринарных услуг.

Быстрая запись на прием, удобный график и оперативная связь с клиникой становятся ключевыми факторами при выборе ветеринарного учреждения.

Целью выпускной квалификационной работы является создание мобильного приложения для ветеринарной клиники, которое обеспечит удобный доступ к записи на прием.

Для достижения этой цели необходимо выполнить ряд задач:

- изучить предметную область и определить требования к функционалу и пользовательскому интерфейсу приложения;
- спроектировать архитектуру приложения, включая его структуру, компоненты и принципы взаимодействия между ними;
- реализовать ключевые функциональные модули приложения;
- разработать удобный и интуитивно понятный пользовательский интерфейс, ориентированный на различные категории пользователей;
- провести тестирование приложения и оценить его производительность, надежность и соответствие требованиям.

Объектом исследования является процесс организации записи на прием в ветеринарной клинике.

Предметом исследования является мобильное приложение для операционной системы Android, автоматизирующее процесс записи на прием в ветеринарной клинике.

Структура работы состоит из трех глав, в которых рассматриваются потребности пользователей, а также формулируются функциональные и нефункциональные требования к создаваемому приложению и проектируется его архитектура.

В первой главе обсуждается важность темы, представляются результаты сравнительного анализа текущих рыночных систем, а также описываются ключевые особенности и характеристики разработки.

Во второй главе изложено логическое моделирование и проектирование системы, установлены требования к создаваемому продукту, а также описана архитектура программного обеспечения.

В третьей главе приводится обзор технологий, выбранных для реализации, описывается процесс разработки ключевых функциональных модулей приложения и формирования пользовательского интерфейса. Также осуществляется тестирование приложения и анализируется его соответствие заданным требованиям.

Выпускная квалификационная работа содержит 66 страниц, 30 рисунков, 10 таблиц, а также 29 источников.

Глава 1 Значимость разработки мобильного приложения для ветеринарной клиники

1.1 Проблема доступности и персонализации услуг для владельцев животных

Актуальность создания мобильного приложения для ветеринарной клиники обоснована растущими потребностями владельцев домашних животных в удобном и быстром доступе к ветеринарным услугам и информации. В современных условиях владельцы животных сталкиваются с трудностями в поиске нужной информации о здоровье и лечении своих питомцев, а также с необходимостью записи на прием к ветеринару.

Ситуация усугубляется тем, что многие клиники имеют ограниченное присутствие в интернете и не предлагают удобные инструменты для общения с клиентами. Это приводит к недостаточной информированности владельцев о возможностях, которые предоставляет клиника, а также к задержкам в получении необходимой помощи.

В ответ на актуальные потребности владельцев домашних животных возникает необходимость создания мобильного приложения для ветеринарной клиники. Оно нацелено на предоставление гибкого, доступного и персонализированного подхода к получению ветеринарных услуг и информации. Это приложение будет включать современные технологии, что позволит адаптировать контент и способы взаимодействия в соответствии с уникальными потребностями и предпочтениями пользователей.

Разрабатываемое приложение для записи на прием в ветеринарную клинику будет использовать алгоритмы машинного обучения для анализа пользовательского поведения. Приложение будет включать функционал для удобной записи на прием, а также выбор врачей и просмотр данных.

Создание такого приложения способствует повышению качества ветеринарных услуг, делая их более доступными и удобными для владельцев домашних животных. Это направление в ветеринарной медицине открывает новые горизонты для эффективного общения с клиентами, обеспечивая индивидуализированный подход к каждому обращению.

1.2 Сравнительный обзор платформ для создания мобильных приложений

Современный рынок мобильных технологий предоставляет множество операционных систем, каждая из которых обладает своими уникальными преимуществами и недостатками. Сосредоточение разработки приложений только на одной платформе может существенно ограничить круг потенциальных пользователей, что, в свою очередь, негативно скажется на успешности проекта. Чем большее количество пользователей протестирует приложение, тем выше шансы на его популярность на рынке.

Поэтому выбор операционной системы для разработки является ключевым этапом, который определяет охват аудитории и потенциальные перспективы приложения. Важно учитывать не только текущую популярность системы, но и целевую аудиторию, особенности рынка и предпочтения пользователей.

На рисунке 1 представлены данные о доле пользователей трех самых популярных мобильных операционных систем за последние годы. Это позволяет анализировать существующие тенденции и делать обоснованный выбор платформы для разработки приложений. Анализ подобных статистических данных позволит понять возможности и будущее каждой из систем, что в итоге отразится на успешности проекта в условиях жесткой конкуренции на рынке мобильных технологий.

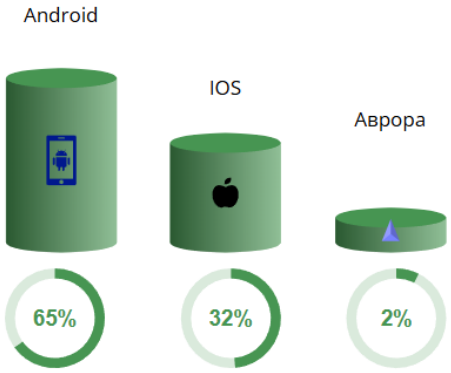

Рисунок 1 – Распределение рыночной доли мобильных операционных систем на рынке смартфонов

На диаграмме выделены три ключевых платформы, которые стоит рассмотреть при создании мобильных приложений:

– Android;

– iOS;

– Аврора.

«Android — это популярная мобильная операционная система, разработанная компанией Google» [2]. Она используется на множестве устройств, включая смартфоны, планшеты, смарт-часа и другие устройства. Основные характеристики и особенности Android перечислены ниже.

Открытость — Android является открытой платформой, что позволяет производителям адаптировать систему под свои устройства и разработчикам создавать собственные приложения.

Магазин приложений Google Play представляет собой обширную библиотеку, содержащую миллионы приложений для скачивания. Это значительно упрощает выполнение различных задач и делает их более доступными для пользователей. Кроме того, Android доступен на широком спектре устройств от различных производителей, что предоставляет пользователям уникальную возможность выбрать то устройство, которое наилучшим образом соответствует их потребностям. Благодаря такой

разнообразии, каждый может найти идеальное решение для себя, будь то смартфон, планшет или другой гаджет.

Настраиваемость — Пользователи могут изменять интерфейс, устанавливать собственные лаунчеры и вносить изменения в систему, чтобы она соответствовала их личным предпочтениям.

Регулярные обновления — Google регулярно выпускает обновления, которые не только добавляют новые функции, но и исправляют уязвимости безопасности.

Связь с сервисами Google — Android обеспечивает глубокую интеграцию с множеством приложений Google, включая Google Drive, Google Maps и другие. Это значительно упрощает взаимодействие пользователей с устройством [6].

Android продолжает развиваться, и его популярность, благодаря своему открытости и разнообразию устройств, только растет. Эта система является отличным выбором для пользователей, ищущих функциональность и широту возможностей в мобильных приложениях [18].

iOS представляет собой мобильную операционную систему, созданную компанией Apple специально для своих устройств, таких как iPhone, iPad и iPod Touch. Ниже приведены ключевые характеристики и отличительные черты iOS, которые делают ее уникальной:

– замкнутая экосистема. iOS создается как закрытая платформа, что обеспечивает высокий уровень безопасности и стабильности. Такой подход позволяет Apple эффективно управлять качеством приложений, доступных в App Store, обеспечивая пользователей надежными и проверенными продуктами. Кроме того, эта структура способствует унификации пользовательского опыта и гармоничному взаимодействию устройств между собой, что делает их более удобными и функциональными в повседневной жизни пользователя;

– App Store. В официальном магазине приложений доступно большое количество приложений и игр, проверенных на соответствие стандартам Apple;
– интерфейс пользователя. iOS предлагает интуитивно понятный и простоватый интерфейс, что делает его доступным для пользователей любого возраста и уровня подготовки;
– регулярные обновления. Apple регулярно выпускает обновления для iOS, которые обеспечивают новые функции, улучшения производительности и исправления безопасности;
– интеграция с экосистемой Apple. iOS хорошо интегрируется с другими продуктами Apple (Mac, Apple Watch, Apple TV), что позволяет пользователям бесшовно взаимодействовать между устройствами;
– высокая производительность. Приложения на iOS обычно работают быстро и отзывчиво благодаря оптимизации системы под аппаратное обеспечение.

iOS является привлекательным выбором для пользователей, которые ценят безопасность, качественные приложения и гармоничное взаимодействие с устройствами Apple. С каждым обновлением система продолжает внедрять инновации и улучшения, что делает её одной из лидирующих мобильных платформ.

Аврора — это относительно, новая операционная система, разработанная на базе Linux, представляет собой интересный вариант для создания мобильных приложений. Основные её характеристики и особенности включают:

– поддержка открытых стандартов, что позволяет разработчикам свободно использовать и адаптировать платформу;
– адаптация к различным устройствам, включая смартфоны, планшеты и IoT-устройства;

– возможность гибкой настройки интерфейса и функционала под потребности пользователей.

Аврора нацелена на создание безопасной и приватной среды для пользователей, что становится важным аспектом в условиях растущих угроз кибер-безопасности. Благодаря своей архитектуре, операционная система предлагает высокий уровень безопасности и защиты данных.

Сравнивая с другими платформами, такими как Android и iOS, Аврора может быть менее популярной, но предоставляет уникальные возможности для нишевых приложений и решений, особенно в странах, где требуется высокая степень конфиденциальности и контроля над данными [19]. Выбор ОС Аврора для разработки мобильного приложения может быть оправдан, если ваша аудитория заинтересована в безопасных и приватных решениях, а также если планируется интеграция с устройствами, которые управляются системой. Для облегчения процесса выбора операционной системы, была создана таблица 1, в которой представлены ключевые преимущества и недостатки ранее рассмотренных операционных систем. Эта таблица поможет быстро сравнить функции и характеристики, а также оценить, какая из систем лучше всего соответствует потребностям. Включение этих данных позволит более информировано подойти к выбору и учесть все аспекты, которые могут повлиять на пользовательский опыт.

Таблица 1 – Анализ и сравнение мобильных операционных систем

Операционная система	Преимущества	Недостатки
Android	Широкий выбор устройств различных ценовых категорий. Возможность настройки и кастомизации интерфейса	Более высокий риск вирусов и вредоносного ПО Разные производители могут предлагать устаревшие версии с задержками обновлений
iOS	Высокий уровень безопасности Удобный и интуитивно понятный интерфейс	Закрытая экосистема с ограниченными возможностями настройки Более высокая стоимость устройств по сравнению с конкурентами

Продолжение таблицы 1

Аврора ОС	Высокий уровень безопасности и защиты данных Поддержка российских разработчиков и программ	Ограниченное количество программ и приложений по сравнению с более популярными ОС Меньшая распространенность может привести к ухудшению поддержки и документации Некоторые пользователи могут столкнуться с неудобством в использовании из-за особенностей интерфейса

Исходя из сравнительного анализа iOS, Аврора ОС и Android, было решено сосредоточиться на создании приложения под Android.

Такое решение обусловлено рядом ключевых преимуществ платформы. Широкий охват – Android лидирует по рыночной доле, что гарантирует максимальную аудиторию для будущего приложения.

Богатый App Store – Google Play Store предлагает обширный каталог приложений, готовых интегрироваться или служить источником идей для разработки. Универсальность – Android поддерживается широким спектром устройств от различных производителей, что обеспечивает доступность приложения для большей части пользователей.

Разработка – Open-source природа Android упрощает разработку и адаптацию под специфические потребности проекта.

Комплекс этих факторов делает Android отличной платформой для воплощения амбициозного проекта с разнообразным функционалом и максимальным привлечением аудитории [20].

1.3 Определение среды разработки

Наиболее популярные платформы для создания мобильных приложений включают:

- android Studio(Kotlin/Java);

- react Native (JavaScript);
- flutter (Dart);
- xamarin (C#);
- ionic(TypeScript/HTML/CSS/JavaScript).

Предлагаю внимательно изучить каждую из платформ отдельно, чтобы лучше понять их уникальные особенности и преимущества. Такой подход позволяет глубже анализировать функции, возможности и ограничения каждой системы, что поможет сделать более обоснованный выбор в зависимости от конкретных потребностей пользователей или проектов.

«Android Studio — официальная интегрированная среда разработки(IDE) для разработки приложений Android [3]. Она подходит для взаимодействия на языках Java и Kotlin [27]. С её помощью разработчики создают приложения для мобильных, планшетов, телевизоров, часов и других устройств. IDE содержит инструменты для разработки, отладки, тестирования и отслеживания производительности приложений. Android Studio — бесплатная, работает на Windows, Mac и Linux. Приложения можно сразу публиковать в магазине Google Play» [9].

«React Native — это популярная платформа мобильных приложений на основе JavaScript, которая позволяет создавать мобильные приложения с собственным интерфейсом для iOS и Android. Фреймворк позволяет создавать приложения для различных платформ, используя одну и ту же кодовую базу.

React Native был впервые выпущен Facebook в качестве проекта с открытым исходным кодом в 2015 году. Всего за пару лет он стал одним из лучших решений, используемых для мобильной разработки. Разработка React Native используется для поддержки некоторых ведущих мобильных приложений в мире, включая Instagram, Facebook и Skype» [10].

«Flutter — комплект средств разработки и фреймворк с открытым исходным кодом для создания мобильных приложений под Android и iOS, веб – приложений, а также настольных приложений под Windows, macOS и Linux

с использованием языка программирования Dart, разработанный и развиваемый корпорацией Google» [11].

Flutter применяет язык Dart, который был создан компанией Google. Он отличается от многих других кроссплатформенных фреймворков тем, что компилирует код Dart в нативный машинный код ARM или Intel, а не использует WebView или интерпретацию через JavaScript. Это обеспечивает более высокую производительность и плавность анимаций. Flutter строится на концепции виджетов, которые служат основой для создания пользовательского интерфейса. Каждый элемент, начиная с простых кнопок и заканчивая сложными макетами, является виджетом, который можно настраивать и сочетать. Это дает возможность формировать уникальный стиль приложений, адаптируя их под конкретные требования пользователей и создавая нестандартные решения для интерфейса. Таким образом, Flutter предлагает гибкость и творческую свободу в проектировании приложений.

«Xamarin — это фреймворк для кроссплатформенной разработки мобильных приложений (iOS, Android, Windows Phone) с использованием языка C#. Идея очень простая. Вы пишете код на своем любимом языке, с применением всех привычных для вас языковых фич типо LINQ, лямбда-выражений, Generic`ов и async`ов. При этом вы имеете полный доступ ко всем возможностям SDK платформы и родному механизму создания UI, получая на выходе приложение, которое, строго говоря, ничем не отличается от нативных и (по крайней мере по заверениям) не уступает им в производительности» [5].

Xamarin (на C#) представляет собой эффективную и доступную платформу для создания кроссплатформенных мобильных приложений, позволяющую использовать код на C# многократно для iOS, Android и Windows. Однако, перед выбором Xamarin необходимо учитывать его недостатки, такие как больший размер приложений и необходимость знания C# и .NET. В целом, Xamarin является отличным выбором для разработчиков, знакомых с C# и .NET, которые хотят создавать высокопроизводительные и нативные кроссплатформенные приложения.

При использовании Xamarin необходимо тщательно выбирать между Xamarin.Forms и Xamarin.Native, учитывая требования к производительности и кастомизации UI.

«Ionic — это набор инструментов с открытым исходным кодом для создания кроссплатформенных мобильных, веб- и настольных приложений с использованием веб-технологий, таких как HTML, CSS и JavaScript/TypeScript. Он предоставляет набор готовых компонентов и инструментов для создания высококачественных интерактивных приложений» [13].

В ходе разработки была создана таблица 2, в которой систематизированы основные достоинства и недостатки. В ней собраны и упорядочены ключевые преимущества и негативные аспекты каждой из рассмотренных платформ. Такой детализированный анализ предоставляет возможность сделать осознанный выбор, принимая во внимание уникальные характеристики проекта и определяя наиболее подходящий инструмент для его реализации.

Таблица 2 – Анализ сравнения сред разработки для ОС Android

Критерии оценивания	Android Studio	React Native	Flutter	Xamarin	Ionic
Официальная поддержка	+	+	+	+	+
UI Designer	+	-	-	-	-
Эмулятор	+	+	-	-	-
Интеграция с Gradle	+	+	+	-	-
Отладка	+	+	+	+	+
Бесплатность	+	+	+	+	+
Профилирование	+	+	+	+	-

Результаты проведенного анализа показывают, что Android Studio выделяется как наиболее подходящая среда для создания приложений для операционной системы Android.

«Это утверждение основано на наличии специализированных инструментов, высокой эффективности работы и интуитивно понятном интерфейсе, а также поддержке популярных языков программирования, таких как Kotlin и Java, а также развитой экосистемой и активным сообществом» [15]. Указанные достоинства способствуют повышению эффективности и комфорта при создании качественных и многофункциональных приложений для платформы Android.

1.4 Характерные черты и особенности разработки

«Android – это мобильная операционная система, разработанная компанией Google и основанная на ядре Linux [28]. Android является самой популярной мобильной ОС в мире, доминируя на рынке смартфонов и планшетов. Кроме того, Android для мобильной разработки – это зрелая, мощная и гибкая платформа, требующая от разработчиков знаний, опыта и постоянного обучения. Однако, огромная аудитория, разнообразие устройств и постоянное развитие делают Android привлекательной и перспективной платформой для создания инновационных и успешных мобильных приложений» [16].

Создание Android-приложений требует от разработчиков не только теоретических знаний, но и практических навыков, позволяющих решать сложные технические задачи. Необходимо учитывать особенности архитектуры Android, обеспечивать совместимость с широким спектром устройств и версий ОС, оптимизировать производительность и эффективно управлять ресурсами. Для создания качественного и надежного приложения важно понимать следующие особенности Android.

Увеличение размера приложения после установки - размер установленного приложения может быть в несколько раз больше, чем исходный APK-файл, из-за процессов распаковки и оптимизации. Необходимо учитывать это при планировании объема хранилища. Кроме того, при

недостатке свободного места на устройстве скорость доступа к файлам может снижаться.

Ограничение памяти для приложений - Android ограничивает объем оперативной памяти, доступной для каждого приложения, что требует оптимизации использования памяти и ресурсов.

Управление многозадачностью - Android поддерживает одновременную работу нескольких приложений, но только одно из них активно отображается на экране. При разработке необходимо учитывать жизненный цикл приложения и его поведение в фоновом режиме. Внутренняя структура операционной системы Android представлена на рисунке 2.

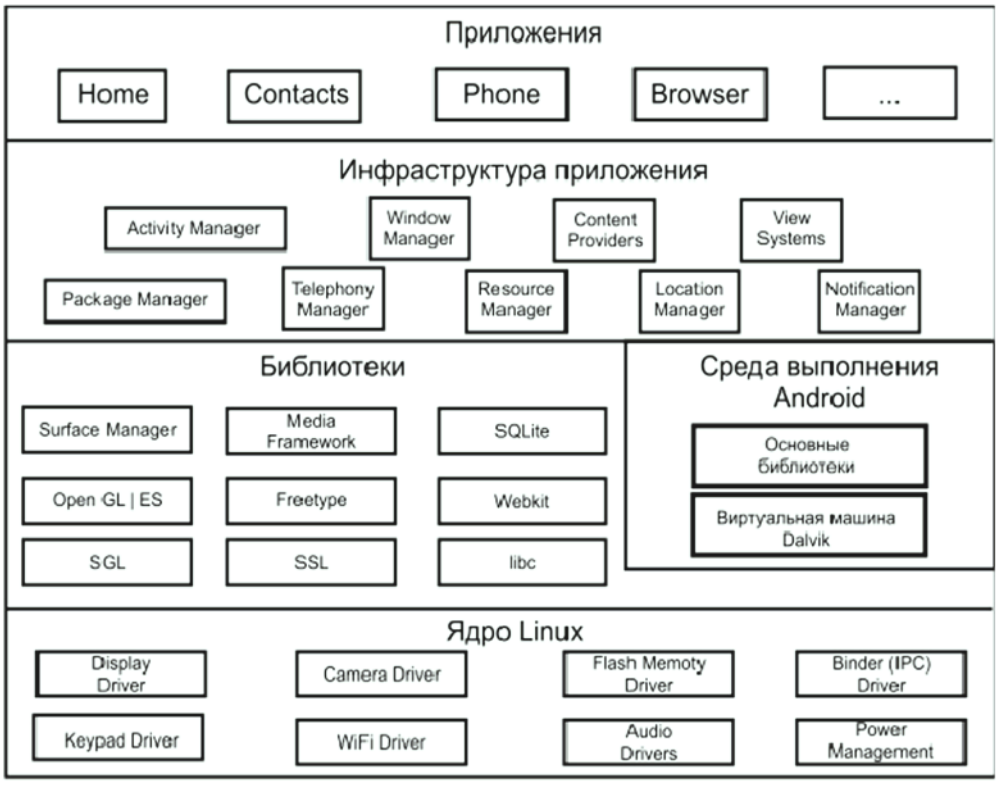

Рисунок 2 – Схематическое представление структуры ОС Android

Архитектура Android представлена многослойной структурой, состоящей из различных компонентов:

- ядро Linux служит основой системы, предоставляя ключевые сервисы, такие как управление памятью, обработка процессов и взаимодействие с драйверами устройств. Это обеспечивает как стабильность работы, так и высокий уровень безопасности системы;
- библиотеки и Android Runtime (ART) представляют собой следующие уровни, содержащие необходимые библиотеки для работы с мультимедиа. ART выполняет компиляцию Ahead-of-Time (AOT), преобразуя код в машинный язык на этапе установки. Это решение значительно повышает производительность приложений и снижает расход энергии, что особенно важно для мобильных устройств, оптимизируя их работу;
- «фреймворк (с английского framework — «каркас, структура») — это набор инструментов, ускоряющих разработку приложений» [8].

Программное обеспечение верхнего уровня включает в себя как предустановленные, так и пользовательские приложения, которые способны взаимодействовать с системными компонентами и друг с другом через API.

Android предоставляет разработчикам широкие возможности для создания разнообразных приложений. Управление этими приложениями осуществляется с помощью специальных компонентов, именуемых "Activities", которые выполняют функции, аналогичные окнам в таких операционных системах, как Windows. Activities играют ключевую роль в том, как пользователи взаимодействуют с приложениями. Каждая Activity проходит через уникальный жизненный цикл, который охватывает стадии от запуска до завершения. Основные этапы жизненного цикла включают инициализацию, паузу, возобновление и завершение. Визуальная схема, представленная на рисунке 3, демонстрирует этот процесс.

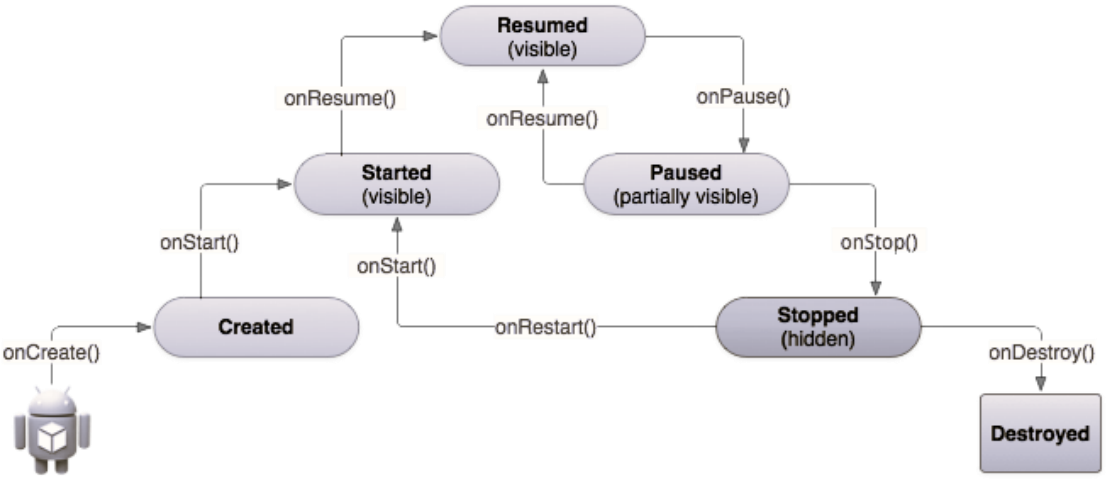

Рисунок 3 – Жизненный цикл приложения в операционной системе Android

Когда приложение запускает Activity, на первом этапе система инициирует метод onCreate(). В ходе этого процесса происходит создание и начальная настройка Activity: разрабатывается интерфейс, инициализируются переменные и подготавливаются необходимые данные.

По завершении начальной настройки активируется метод onStart(), который делает Activity доступным для пользователей. Если данная Activity ранее запускалась и была остановлена, вместо метода onStart() будет вызван метод onRestart(). Такой подход обеспечивает гибкое управление состояниями приложения и оптимизацию пользовательского опыта.

Следующим этапом жизненного цикла Activity является метод onResume(), который активируется после onStart() или когда Activity снова становится видимым после временной остановки, например, при переключении между окнами или диалогами. В onResume() происходит полное взаимодействие Activity с пользователем; именно здесь обычно запускаются анимации, аудиотреки и видеообъекты для создания динамичного пользовательского опыта.

Когда открывается новая Activity или приложение переходит в фоновый режим, вызывается метод onPause(). В этот момент критически важно сохранить все важные данные и приостановить процессы, которые не требуют

выполнения, когда приложение находится в паузе. Если Activity становится полностью невидимым, например, при закрытии пользователем или переключении на другое активное окно, активируется метод onStop().

Заключительный этап жизненного цикла – это вызов метода onDestroy(), который происходит, когда система начинает освобождать ресурсы и завершает работу Activity, или же когда пользователь самостоятельно инициирует закрытие приложения. В данном методе осуществляется финальная уборка: освобождаются ресурсы и закрываются все активные соединения с базами данных, что гарантирует правильное завершение работы приложения.

Вывод по первой главе

Необходимость создания мобильного приложения для ветеринарной клиники обусловлена растущим спросом на удобный онлайн-доступ к услугам и информации со стороны владельцев животных.

Существующие инструменты не удовлетворяют этим потребностям, поэтому разработка приложения, включающего функционал записи на приём, управление данными о пациентах и персонализированный подход, повысит качество и доступность ветеринарных услуг.

Выбор платформы для разработки будет учитывать рыночную долю и целевую аудиторию, с приоритетом на платформы с максимальным охватом Android. Поэтому было принято решение о разработке такого мобильного приложения для ветеринарной клиники.

Глава 2 Логическое моделирование и проектирование системы

2.1 Определение требований к системе

При разработке мобильного приложения крайне важно не только обеспечить его визуальную привлекательность, но и гарантировать, что оно будет интуитивно понятным и удобным для пользователей. В таблице 3 представлены ключевые требования к новому программному обеспечению, сгруппированные по модели FURPS+.

«FURPS — классификация требований к программным системам. Образована от первых букв слов:

- functionality — Функциональные требования: свойства, возможности, безопасность. Являются основными, по этим требованиям строятся диаграммы вариантов использования (Use case diagram);
- usability — Требования к удобству использования (UX): человеческий фактор, эстетика, последовательность, документация;
- reliability — Требования к надежности: частота возможных сбоев, отказоустойчивость, восстанавливаемость, предсказуемость устойчивости;
- performance — Требования к производительности: время отклика, использование ресурсов, эффективность, мощность, масштабируемость;
- supportability — Требования к поддержке: возможность поддержки, ремонтопригодность, гибкость, модифицируемость, модульность, расширяемость, возможность локализации» [12].

Чаще всего FURPS+ включают в себя такие элементы, как производительность, удобство взаимодействия и безопасность.

«Функциональные требования - описывают конкретные действия или поведение системы. Они отвечают на вопрос «Что система должна делать?» и прямо связаны с функциональностью.

Эти требования обычно определяются через сценарии использования, пользовательские истории или спецификации.

Нефункциональные требования касаются характеристик системы, таких как производительность, безопасность, надежность и удобство использования. Они определяют, как система должна вести себя, и влияют на общее качество пользовательского опыта» [7]. В таблице 3 представлены требования к создаваемому мобильному приложению для ветеринарной клиники.

Таблица 3 – Требования к создаваемому мобильному приложению для ветеринарной клиники

Требование	Описание	Приоритет	Метрика
Functionality — Функциональные требования			
Выбор питомца	Пользователь должен иметь возможность ввести данные своего питомца	Высокий	100% корректный выбор питомца
Выбор ветеринарного врача	Пользователь должен иметь возможность ввести данные врача или выбрать его из списка	Высокий	Полный список доступных специализаций
Выбор даты и времени приема	Пользователь должен иметь возможность выбрать удобную дату и время для приема	Высокий	-
Управление записями	Пользователь должен иметь возможность просматривать список предстоящих и прошедших записей	Высокий	-
Usability — Требования к удобству использования			
Понятный интерфейс записи	Процесс записи должен быть интуитивно понятным и простым для пользователя	Высокий	A/B тестирование интерфейса записи
Удобный выбор даты и времени	Интерфейс выбора даты и времени должен быть удобным	Высокий	

Продолжение таблицы 3

Быстрая загрузка расписания	Расписание должно загружаться быстро и без задержек	Высокий	Время загрузки расписания не более 3 секунды
Подтверждение перед записью	Перед окончательным подтверждением записи, пользователь должен видеть сводную информацию	Высокий	
Reliability — Требования к надежности			
Точность расписания	Расписание должно быть актуальным	Высокий	Автоматическая синхронизация с базой ветеринарной клиники
Performance — Требования к производительности			
Быстрая работа	Процесс записи не должен занимать много времени	Высокий	Время от начала записи до подтверждения не более 1 минуты
Оптимизация запросов к серверу	Приложение должно эффективно использовать ресурсы сервера	Средний	
Supportability — Требования к поддержке			
Централизованное управление	Централизованное управление расписанием	Средний	Интеграция с системой управления ветеринарной клиники
Логирование ошибок	Логирование ошибок при записи на прием для исправления ошибок	Средний	

После определения требований к будущему мобильному приложению для ветеринарной клиники, стоит перейти к проектированию системы, принимая во внимание запросы пользователей, которые были выявлены в процессе анализа требований (например, используя FURPS+). Моделирование позволит визуализировать структуру приложения, взаимодействия между компонентами и пользовательские сценарии, обеспечивая соответствие системы потребностям клиники и ее клиентов.

2.2 Анализ и сравнение популярных мобильных приложений для ветеринарной клиники

Предлагаю углубленный анализ и сравнение мобильных приложений, предназначенных для ветеринарных клиник. Этот обзор охватит различные аспекты, включая функциональность, удобство использования, интеграцию с другими системами, стоимость и отзывы пользователей. Мы рассмотрим приложения, предлагающие различные услуги, такие как запись на приём, управление пациентами, ведение истории болезни, онлайн-консультации и другие. Цель анализа — определить лучшие решения для оптимизации работы ветеринарной клиники и повышения качества обслуживания клиентов. В результате будет составлен рейтинг приложений с учётом их сильных и слабых сторон, что поможет ветеринарным клиникам сделать обоснованный выбор.

Vet Manager Plus — мобильное приложение для ветеринарного персонала, оптимизирующее управление клиникой. Оно предоставляет инструменты для эффективной работы, но фокусируется на внутренних процессах клиники.

Pet In Vet — мобильное приложение для клиентов, интегрирующееся с Vet Manager Plus. Оно требует предварительной настройки и предоставления клиентам QR-кода для установки. Функциональность Pet In Vet ориентирована на взаимодействие с клиентами, обеспечивая удобство записи на прием и, возможно, доступ к некоторым данным о питомцев.

Vetamanager — это современная облачная платформа для управления ветеринарными клиниками. Она предоставляет инструменты для эффективного планирования приёмов и управления расписанием, полного контроля медицинской истории пациентов, управления финансами и интеграции с системами онлайн-платежей, учёта клиентов и товаров (включая складской учёт), а также CRM-функционал для работы с клиентами. Платформа легко интегрируется с лабораториями, онлайн-магазинами и

другими сервисами. В основе Vetmanager лежит архитектура REST API, использующая PHP, JavaScript, bash и СУБД MySQL.

Приложения для записи на прием в ветеринарной клинике должны обладать большой базой данных для хранения личной информации пользователей и бесперебойной работой самого приложения. В таблице 4 приведены остальные сравнительные качества вышеперечисленных мобильных приложений.

Таблица 4 – Сравнение мобильных приложений для ветеринарной клиники

Критерии	Мобильное приложение для ветеринарной клиники		
	Vet Manager Plus	Pet In Vet	Vetamanager
Функциональность для пользователей	-	-	+
Удобство использования	+	-	+
Хранение данных	-	-	+
Безопастность	+	+	+
Интеграция	-	-	+
Стоимость	+	+	-

Проанализировав таблицу, выяснили, что она показывает существенные различия в функциональности и стоимости трёх рассматриваемых мобильных приложений для ветеринарных клиник.

Vetamanager демонстрирует наилучшие показатели по функциональности для пользователей, хранению данных и интеграции с другими системами. Однако, его стоимость, вероятно, выше, чем у конкурентов.

Vet Manager Plus имеет высокие оценки по удобству использования и безопасности, но его функциональность и интеграционные возможности выглядят менее развитыми. Стоимость приложения оценивается как приемлемая.

Pet In Vet получает низкие оценки по удобству использования и функциональности. Тем не менее, обеспечивает приемлемый уровень безопасности. Стоимость, как и у Vet Manager Plus, считается приемлемой.

Исследование существующих мобильных приложений для ветеринарных клиник выяснило, что ни одно из них не соответствует всем нашим критериям. В связи с этим было решено создать собственное мобильное приложение, которое предложит полный комплект необходимых функций: удобный интерфейс для записи пациентов на прием, безопасное хранение информации о пациентах и их владельцах, не высокая стоимость приложения. Кроме того, будет обеспечена надежная защита данных и удобный доступ к информации как для персонала клиники, так и для владельцев животных.

2.3 Выбор технологии для логического моделирования мобильного приложения

Выбор технологии логического моделирования мобильного приложения - важный этап, определяющий эффективность процесса разработки и конечный результат. При выборе стоит учитывать несколько факторов:

Сложность проекта: для простых приложений могут быть достаточны более простые инструменты, в то время как для сложных систем с большим количеством взаимодействий потребуется более мощный и гибкий подход.

Тип приложения: особенности приложения могут влиять на выбор наиболее подходящей технологии.

Опыт команды: выбор технологии, с которой команда имеет опыт, ускорит процесс моделирования и снизит вероятность ошибок.

Поддержка и документация: хорошая документация и активное сообщество облегчают процесс обучения и решения возникающих проблем.

Интеграция с другими инструментами: возможность интеграции с инструментами для разработки, тестирования и управления проектом может

повысить эффективность работы. Некоторые популярные технологии логического моделирования мобильных приложений будут описаны ниже.

UML (Unified Modeling Language) – универсальный язык моделирования, позволяющий описывать различные аспекты системы, включая структуру, поведение и взаимодействие компонентов. Подходит для сложных проектов.

Существуют различные UML-инструменты, как бесплатные (например, UMLet), так и коммерческие (например, Enterprise Architect, Visual Paradigm). Позволяет создавать диаграммы классов, диаграммы последовательностей, диаграммы состояний и другие.

BPMN (Business Process Model and Notation) – ориентирован на моделирование бизнес-процессов, но может быть полезен для описания логики работы приложения и пользовательских сценариев. Инструменты: Camunda, Bizagi. Подходит для приложений, где важна последовательность действий и автоматизация.

Flowcharts (Блок-схемы) – простой и понятный способ визуализации алгоритмов и логики работы приложения. Подходит для описания простых сценариев и логики отдельных функций. Инструменты: Draw.io, Lucidchart.

Wireframing и Prototyping инструменты (Axure RP, Figma, Adobe XD, Sketch) – позволяют создавать интерактивные прототипы приложения, имитирующие пользовательский интерфейс и поведение. Помогают проверить концепцию и собрать обратную связь от пользователей до начала разработки.

Mind Maps (XMind, FreeMind) – полезны для структурирования информации, выявления взаимосвязей и генерации идей. Могут быть использованы на ранних стадиях моделирования.

CASE-средства (Computer-Aided Software Engineering) – комплексные инструменты, поддерживающие весь процесс создания программного обеспечения включает несколько ключевых шагов: моделирование,

проектирование, кодирование и тестирование [14]. Примеры инструментов, которые можно использовать, включают IBM Rational Rose и AllFusion ERwin Data Modeler. Если говорить о мобильном приложении для ветеринарной клиники, акцент следует сделать на функции записи, хорошим вариантом будет сочетание нескольких подходов. UML для моделирования структуры данных (например, классы "Врач", "Питомец", "Прием").

BPMN или Flowcharts для описания процесса записи на прием (выбор питомца, врача, времени, подтверждение).

После выбора оптимальной технологии логического моделирования начинается важный этап создания диаграммы определения взаимосвязей для мобильного приложения ветеринарной клиники. На рисунке 4 представлена диаграмма определения взаимосвязей.

Рисунок 4 – Диаграмма определения взаимосвязей

На рисунке показана диаграмма вариантов использования (Use Case Diagram) для системы записи в ветеринарную клинику.

Клиент может:

- просматривать записи;
- отменять запись;

– записываться на прием;
– управлять своим профилем.

Администратор может:

– назначать процедуры;
– просматривать информацию о пациентах;
– управлять записями пациентов.

Диаграмма отображает основные действия, доступные каждому из пользователей системы.

Разрабатываемое приложение будет содержать данные о расписании приема ветеринаров, данные о владельцах питомцев и данные о питомцах.

2.4 Проектирование модели данных

«Моделирование данных создает графическое представление структуры базы данных, определяя сущности, атрибуты и связи для точного представления реальных сценариев. Модель данных служит основой для физического и логического проектирования базы данных. Обычно процесс включает в себя следующие этапы:

Логическая модель данных — подробная модель, которая дополнительно расширяет концептуальную модель данных, определяя все необходимые сущности, атрибуты, отношения и ограничения в структурированном формате. Эта модель открывает путь к физическому проектированию базы данных.

Реализация физической модели данных. Используя логическую модель данных в качестве руководства, база данных создается и заполняется данными путем определения таблиц, индексов и других объектов базы данных» [5].

В базе данных будет храниться информация о ветеринарах, расписании приема ветеринаров, питомцах и владельцах

Выделим основные сущности предметной области:

– сущность «Ветеринар» будет хранить данные о ветеринарах;

– сущность «Расписание» будет хранить данные о расписании приема ветеринаров;

– сущность «Питомец» будет хранить данные о питомцах;

– сущность «Владелец» будет хранить данные о владельцах питомцев.

Определим связи между сущностями.

Каждый питомец принадлежит одному владельцу. Один владелец может иметь множество питомцев. Связь между сущностями «Владелец» и «Питомец» будет «один ко многим» (1:N).

Расписание связано с ветеринарами, так как каждый ветеринар имеет свое расписание приема.

Также расписание связано с питомцами, поскольку питомцы записываются на прием к ветеринарам в определенное время.

Связь между сущностями «Ветеринар» и «Расписание» будет «один ко многим» (1:N).

Связь между сущностями «Питомец» и «Расписание» будет один ко многим (1:N).

Используя эти данные построим диаграмму связей сущностей. Диаграмма связей сущностей представлена на рисунке 5.

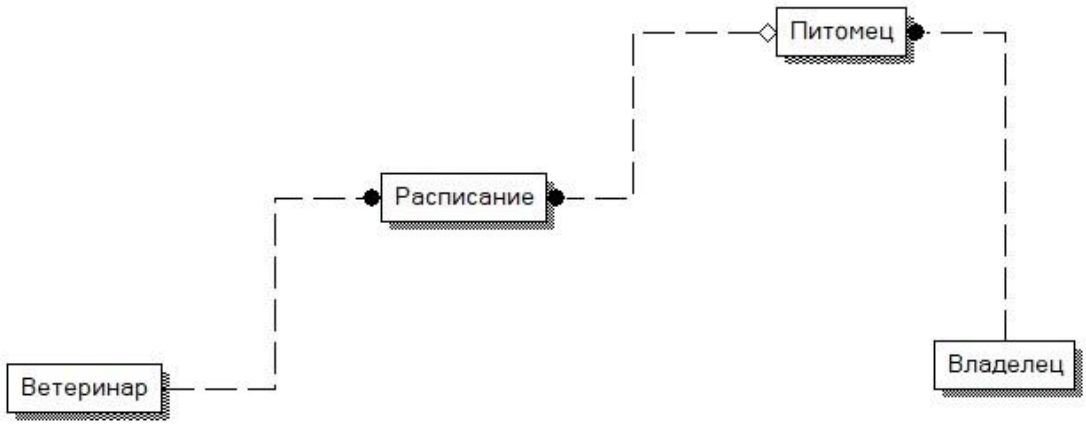

Рисунок 5 – Диаграмма связей

Определим атрибуты сущностей:

– для сущности «Ветеринар» определим следующие атрибуты: код ветеринара, Фамилия, Имя, Отчество, специальность;
– для сущности «Владелец» определим следующие атрибуты: код владельца, Фамилия, Имя, Отчество, дата рождения, телефон;
– для сущности «Питомец» определим следующие атрибуты: код питомца, кличка, дата рождения, тип питомца, код владельца;
– для сущности «Расписание» установим множество атрибутов, таких как код расписания, дата приема, код ветеринара и код питомца.

После того как атрибуты определены, перейдем к созданию логической модели базы данных. На рисунке 6 можно увидеть логическую модель этой базы данных.

Рисунок 6 – Логическая модель базы данных

Так как СУБД SQLite является реляционной СУБД, то она должна соответствовать правилам реляционных отношений. Для этого после создания предварительных отношений и определения атрибутов, необходимо провести нормализацию.

Нормализация – это процесс организации данных в реляционной базе данных с целью минимизации избыточности данных и предотвращения аномалий при обновлении, вставке и удалении данных. Основные нормальные формы включают первую, вторую и третью.

Считается, что таблица находится в 1НФ, если все значения в ее колонках неделимы. Это означает, что каждая ячейка таблицы должна содержать только одно значение, а не множество значений или списки.

В разработанной модели все значения в колонках неделимы, а это означает, что база данных находится в 1НФ.

Считается, что таблица находится во 2НФ, если она уже находится в 1НФ и при этом все ее неключевые атрибуты полностью функционально зависят только от первичного ключа.

Разработанная модель базы данных находится в 1НФ и при этом все ее неключевые атрибуты полностью функционально зависят только от первичного ключа. Поэтому база данных находится в 2НФ.

Считается, что таблица находится в 3НФ, если она уже находится во 2НФ и при этом все ее неключевые атрибуты транзитивно не зависят от первичного ключа. Это подразумевает, что поля, не входящие в ключ, не должны зависеть от других полей, которые также не являются ключевыми.

Таблицы базы данных находятся в 2НФ, но имеют транзитивные зависимости. Поэтому приведем наши базы данных в третью нормальную форму. На следующем рисунке 7 представлена логическая модель базы данных, приведенная к 3НФ.

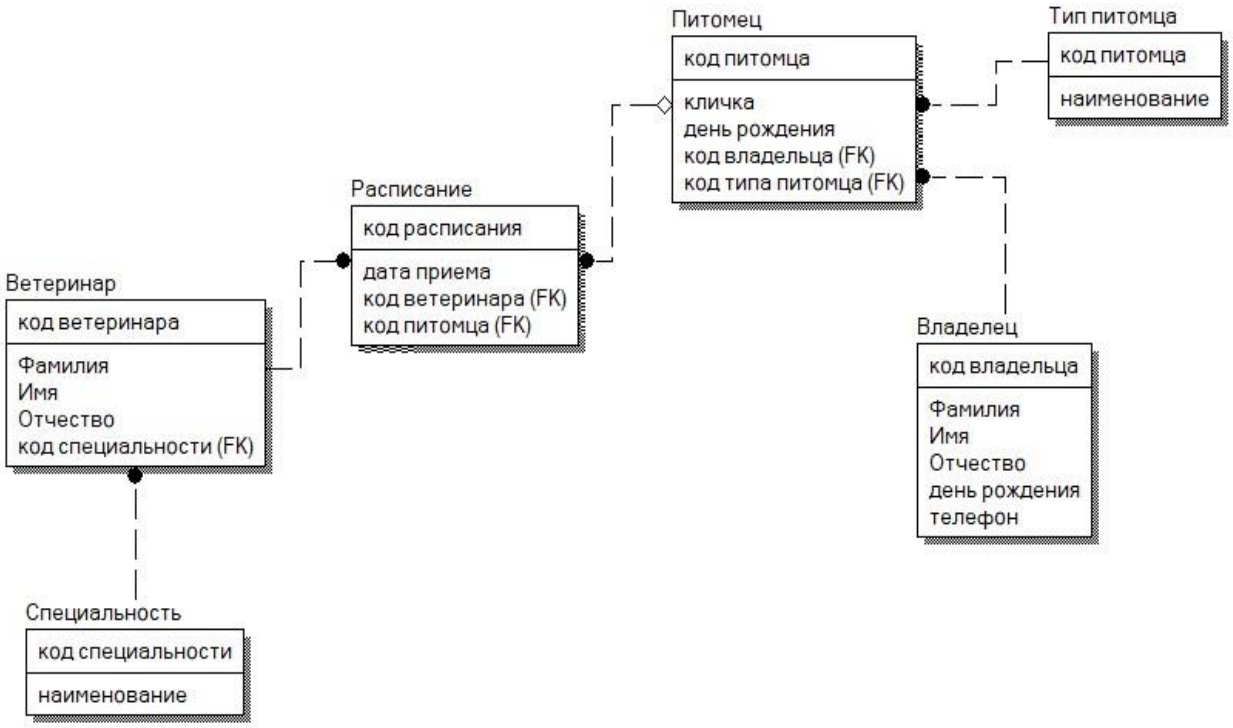

Рисунок 7 – Логическая модель базы данных в 3НФ

2.5 Интеграция физической модели базы данных

После того как будут определены все таблицы базы данных, нам станет необходимо разработать физическую модель этих баз данных в СУБД SQLite. [4].

Поля для таблиц базы данных предоставлены в таблицах 5, 6, 7, 8, 9, 10.

Таблица 5 – Питомцы (Animals)

Атрибут	Поле	Тип	Ключ
Код питомца	ID	INTEGER	PK
Кличка	title	TEXT	
Дата рождения	birthday	TEXT	
Тип животного	typeID	INTEGER	FK
Владелец	ownerID	INTEGER	FK

Таблица 6 - Владельцы (PetsOwner)

Атрибут	Поле	Тип	Ключ
Код владельца	ID	INTEGER	PK
Фамилия	lastNm	TEXT	
Имя	firstNm	TEXT	
Отчество	middleNm	TEXT	
Дата рождения	birthday	TEXT	
Телефон	phone	TEXT	

Таблица 7 - Типы животных (AnimalType)

Атрибут	Поле	Тип	Ключ
Код типа животного	ID	INTEGER	PK
Наименование	title	TEXT	

Таблица 8 - Врачи (Doctor)

Атрибут	Поле	Тип	Ключ
Код врача	ID	INTEGER	PK
Фамилия	lastNm	TEXT	
Имя	firstNm	TEXT	
Отчество	middleNm	TEXT	
специальность	specID	INTEGER	FK

Таблица 9 – Специальности врачей (Specialisation)

Атрибут	Поле	Тип	Ключ
Код специальности	ID	INTEGER	PK
Наименование	title	TEXT	

Таблица 10 – Расписание приема (Schedule)

Атрибут	Поле	Тип	Ключ
Код расписания	ID	INTEGER	PK
Дата приема	DATAPR	TEXT	
Врач	DOCTOR	INTEGER	FK

Зафиксируем информацию из таблиц в физической модели базы данных. На рисунке 7 можно увидеть логическую модель этой базы данных.

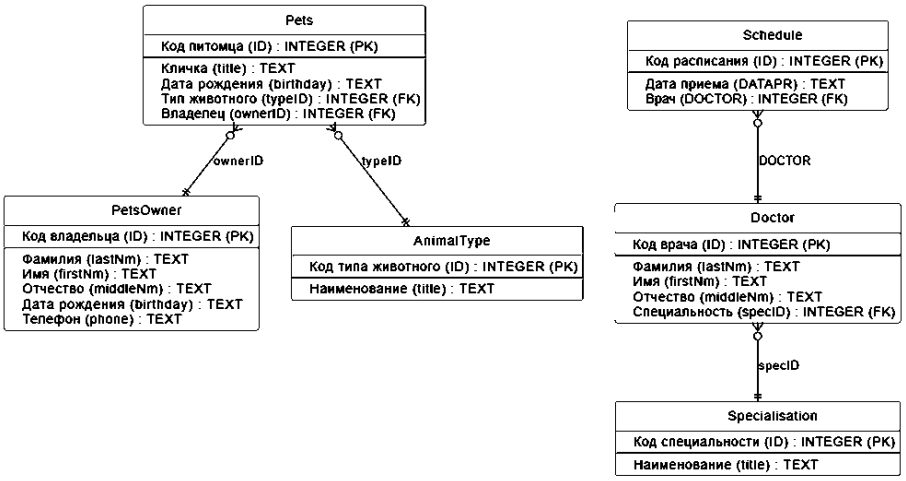

Рисунок 8 – Физическая модель базы данных

На следующем рисунке 9 изображены таблицы, созданные в СУБД SQLite, которые демонстрируют структуру данных и взаимосвязь между различными сущностями в базе данных. Каждая таблица содержит уникальные поля и типы данных, отражающие специфику хранимой информации.

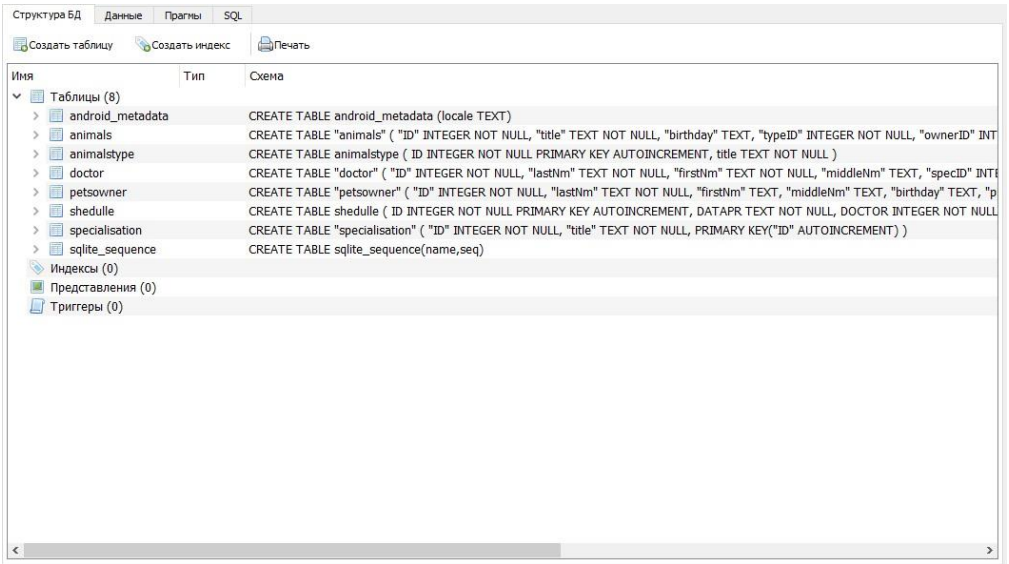

Рисунок 9 – Созданные таблицы базы данных

Правильное создание и структура базы данных существенно влияют на скорость обработки запросов и удобство работы с данными.

2.6 Архитектура программного обеспечения

Архитектура программного продукта определяет структуру, компоненты и взаимодействия мобильного приложения для ветеринарной клиники. Правильный выбор архитектуры обеспечивает масштабируемость, надежность, удобство поддержки и разработки [21].

Мобильное приложение для ветеринарной клиники будет разработано с учетом архитектуры Architecture Components, схема данной архитектуры будет изображена на рисунке 10.

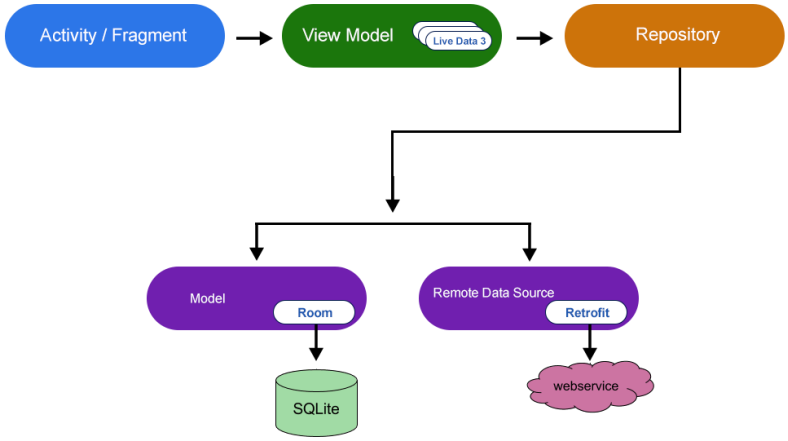

Рисунок 10 – Элементы архитектуры приложения для ветеринарной клиники

Когда архитектурное решение для мобильного приложения будет выбрано, можно перейти к разработке визуализации классов и их взаимодействий через структуру классов приложения. Этот подход способствует лучшему пониманию сложной системы, упрощая анализ её

компонентов и их взаимосвязей. Визуализация также позволяет выявить потенциальные проблемы на ранних этапах разработки и оптимизировать архитектуру для повышения производительности и устойчивости приложения.

На рисунке 11 представлена структура классов мобильного приложения.

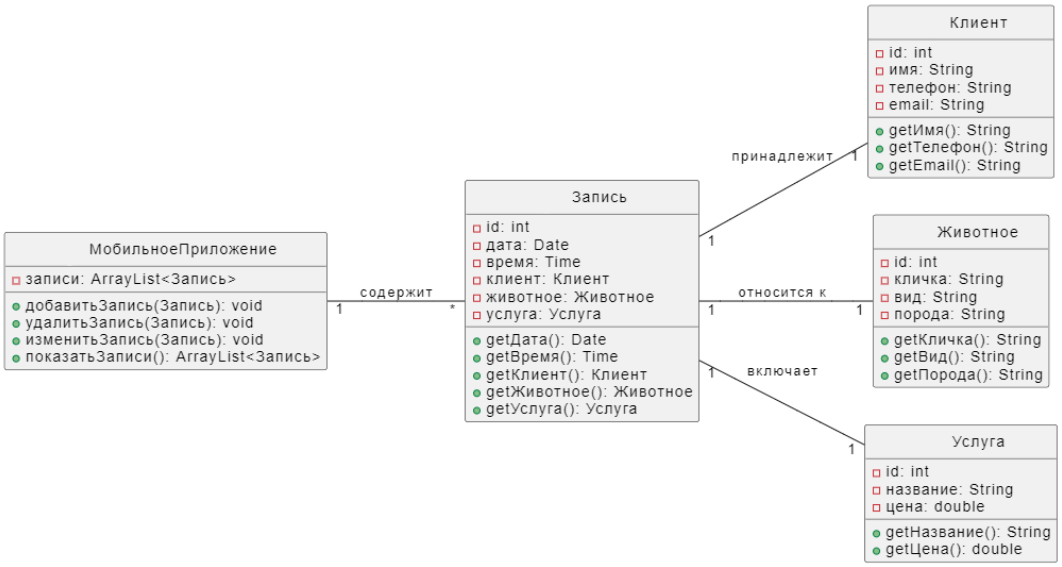

Рисунок 11 – Структура классов мобильного приложения

На рисунке 12 отображена диаграмма развертывания системы. Диаграмма иллюстрирует высокоуровневую системную архитектуру для мобильного приложения записи на прием к врачу.

Справа изображена клиентская часть приложения Mobile Device, в частности, устройство Android в виде файла .apk, в котором есть компоненты пользовательского интерфейса, клиент API (для взаимодействия с серверной частью) и локальное хранилище.

Слева изображен Application Server. Этот сервер обрабатывает логику серверной части приложения. Он включает в себя серверную часть с REST API для мобильной связи, службу аутентификации и компоненты для бизнес-логики.

PostgreSQL хранит данные приложения. База данных содержит такие таблицы данных, как "Пользователи", "Врачи", "Записи на прием" и "Домашние животные".

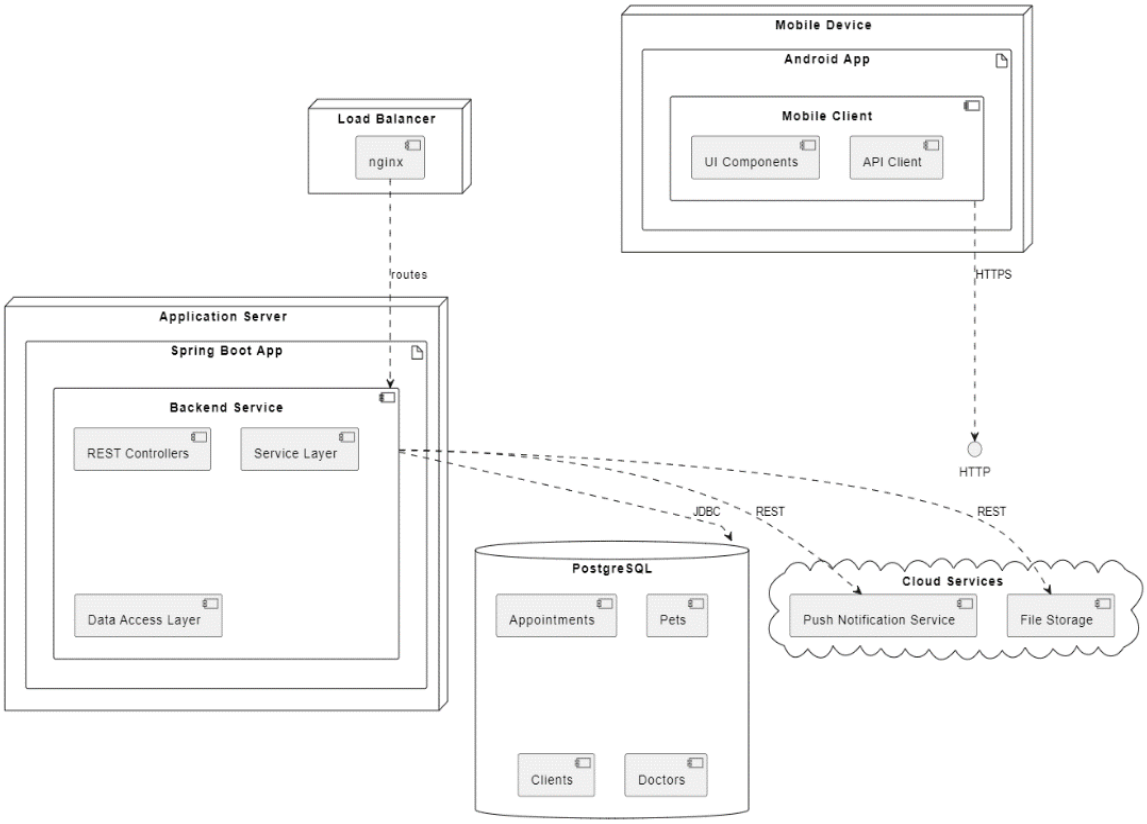

Рисунок 12 – Диаграмма развертывания мобильного приложения и Backend сервиса

Бэкенд-сервис использует многоуровневую архитектуру (контроллеры REST, уровень сервисов, уровень доступа к данным), что способствует разделению задач и удобству обслуживания. Эта схема показывает более сложную архитектуру, чем предыдущая, подробно описывая конкретные технологии и шаблоны проектирования, используемые в бэкэнде.

На рисунках 13 и 14 ниже также предоставлена диаграмма деятельности и диаграмма последовательностей. Диаграммы иллюстрируют процесс записи на прием в приложении для ветеринарной клиники.

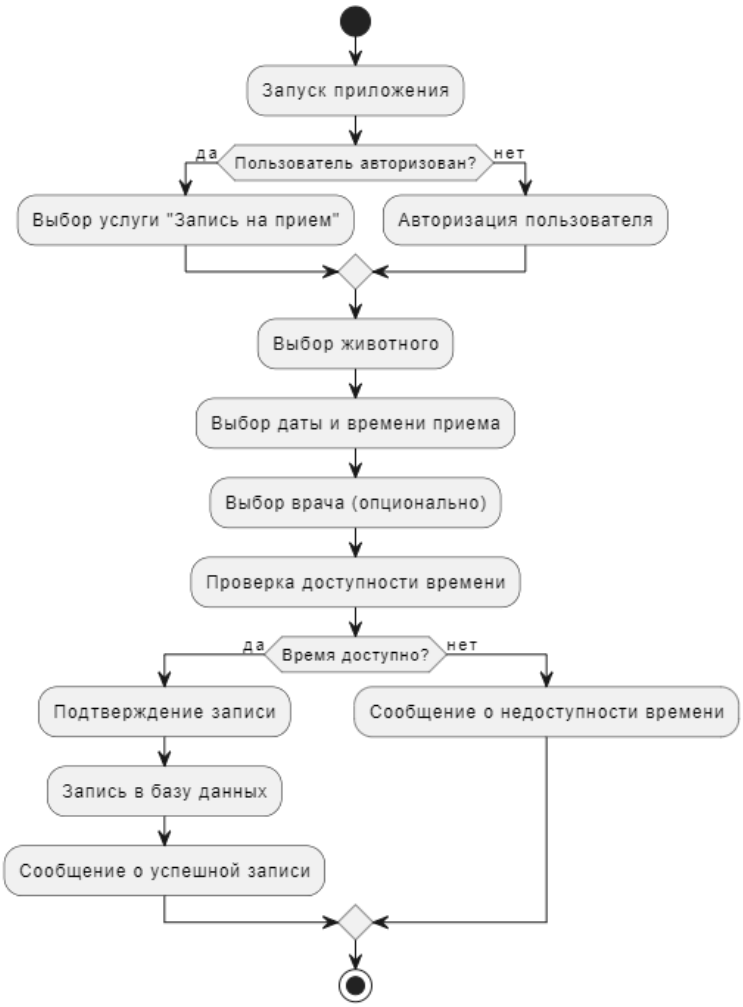

Рисунок 13 – Процедура записи на прием с помощью приложения

На рисунке 13 показана блок-схема процесса записи на прием в мобильном приложении ветеринарной клиники:

- приложение инициализируется и проверяет, выполнена ли авторизация пользователя;
- проверяется, авторизован ли пользователь;
- пользователь переходит к выбору животного, для которого требуется прием;
- следующий шаг – выбор даты и времени визита;
- предусмотрена возможность выбора врача;
- система верифицирует доступность выбранного временного слота.

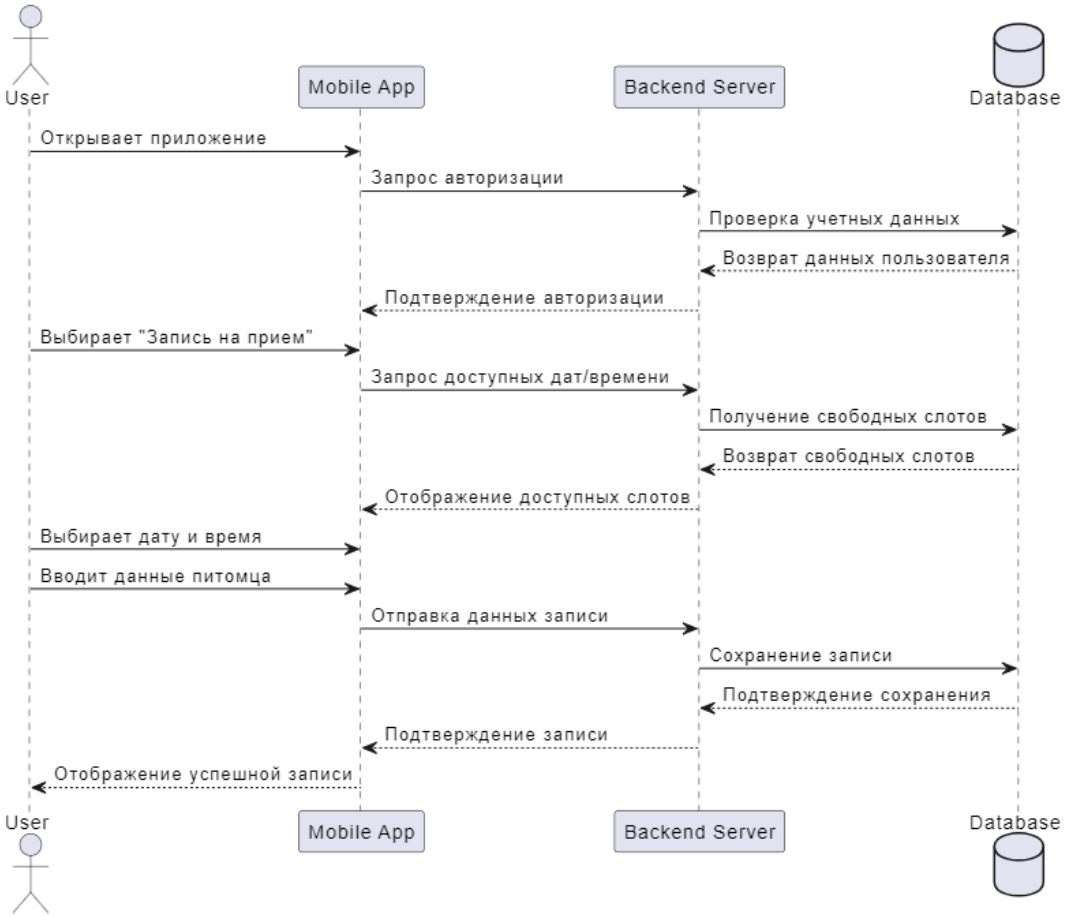

Рисунок 14 – Диаграмма последовательностей

Представленная диаграмма последовательности демонстрирует архитектуру системы записи на прием в ветеринарную клинику, показывая взаимодействие между пользователем (через мобильное приложение), сервером приложений и базой данных. Диаграмма иллюстрирует поток данных и последовательность операций, необходимых для обработки запроса на запись, включая аутентификацию пользователя, валидацию данных, запрос к базе данных для проверки доступности, и обновление базы данных с новой записью.

- пользователь открывает приложение;
- мобильное приложение отправляет запрос на авторизацию серверу;

- сервер проверяет учетные данные в базе данных и возвращает информацию о пользователе;
- мобильное приложение получает подтверждение авторизации;
- пользователь выбирает услугу "Запись на прием";
- мобильное приложение запрашивает у сервера доступные даты и время;
- сервер получает свободные слоты из базы данных и возвращает их приложению;
- приложение отображает доступные слоты пользователю;
- пользователь выбирает дату, время и вводит данные о питомце;
- приложение отправляет данные записи на сервер;
- сервер сохраняет запись в базе данных и подтверждает сохранение;
- приложение отображает пользователю сообщение об успешной записи.

Вывод по второй главе

В данной главе были определены основные требования к будущему мобильному приложению для ветеринарной клиники, опираясь на методологию FURPS+. Особое внимание было уделено не только созданию эстетически привлекательного дизайна, но и обеспечению простоты и понятности для пользователей. Определены ключевые требования в категориях: Functionality, Usability, Reliability, Performance и Supportability, с присвоением приоритетов и метрик для оценки их достижения.

Подчеркнута важность функциональных требований, определяющих действия системы, и нефункциональных требований, влияющих на общее качество пользовательского опыта. Все требования были детально изложены в Таблице 3, включая описание, приоритет и конкретные метрики. Высокий приоритет был присвоен требованиям, связанным с выбором питомца, врача, даты и времени приема, управлением записями, понятным интерфейсом записи, удобным выбором даты и времени, быстрой загрузкой расписания и подтверждением перед записью, а также точностью расписания.

Также в рамках данной главы был выполнен процесс проектирования базы данных для мобильного приложения ветеринарной клиники, начиная с логического моделирования и заканчивая подготовкой к физической реализации. Описан переход от логической модели к физическому проектированию, что является необходимым шагом для создания эффективной и структурированной базы данных.

Определены основные сущности предметной области: "Ветеринар", "Расписание", "Питомец" и "Владелец", а также описаны связи между ними. Установлены связи "один ко многим" (1:N) между "Владельцем" и "Питомцем", "Ветеринаром" и "Расписанием", а также "Питомцем" и "Расписанием".

Определенные сущности и связи позволили построить диаграмму связей сущностей (ER-диаграмму), представленную на рисунке 5, которая наглядно демонстрирует структуру будущей базы данных.

Разработанная модель данных служит прочным фундаментом для последующей физической реализации базы данных, включающей создание таблиц, индексов и других объектов, необходимых для эффективного хранения и управления информацией о ветеринарах, расписании, питомцах и владельцах.

Изложенная информация послужила прочным фундаментом для дальнейшего проектирования системы, ориентированного на удовлетворение потребностей пользователей, выявленных в процессе анализа.

Следующим логичным шагом является переход к этапу проектирования, учитывая обозначенные приоритеты и метрики для эффективного управления рабочим процессом и предоставления качественных услуг.

Глава 3 Практическая реализация мобильного приложения

3.1 Выбор технологий для разработки приложения

При разработке Android-приложения выбор упал на Java в качестве языка программирования, а Android Studio была выбрана как интегрированная среда разработки (IDE). Java была предпочтена вместо других языков, таких как Kotlin, из-за её проверенной временем стабильности и широкой доступности квалифицированных разработчиков, а также из-за зрелости инфраструктуры поддержки Android. Android Studio, как официальная IDE, предоставляет все необходимые инструменты для эффективной разработки, тестирования и отладки приложения. «Android Studio — официальная интегрированная среда разработки (IDE) для разработки приложений Android. Она подходит для взаимодействия на языках Java и Kotlin [26]. С её помощью разработчики создают приложения для мобильных, планшетов, телевизоров, часов и других устройств. IDE содержит инструменты для разработки, отладки, тестирования и отслеживания производительности приложений. Android Studio — бесплатная, работает на Windows, Mac и Linux. Приложения можно сразу публиковать в магазине Google Play» [4].

Рассмотрим ключевые функции Android Studio.

Интеллектуальный редактор кода: предлагает автодополнение кода, рефакторинг и анализ для повышения эффективности и качества кодирования. Гибкая система сборки: работает на базе Gradle, что позволяет настраивать процесс сборки для различных устройств и конфигураций.

Быстрый эмулятор: позволяет тестировать приложение на различных виртуальных устройствах Android без необходимости использования физического устройства [22].

Инструменты отладки: предоставляет мощные функции отладки, включая точки останова, проверку переменных и анализ памяти.

Редактор макетов: визуальный редактор для проектирования пользовательского интерфейса (UI) вашего приложения с использованием перетаскиваемых компонентов [29].

Анализатор APK: позволяет проверять содержимое файла APK (Android Package Kit), который является пакетом, используемым для распространения и установки приложений Android [23].

Профилировщики в реальном времени: отслеживает активность ЦП, памяти и сети в режиме реального времени, помогая оптимизировать производительность вашего приложения.

Поддержка Kotlin и Java: официально поддерживает Java и Kotlin, два популярных языка для разработки Android [17].

Интеграция с другими службами Google: хорошо интегрируется с другими службами Google, такими как Firebase и Google Cloud Platform.

Android Studio — это инструмент для всех, кто хочет разрабатывать приложения Android. Он предоставляет все необходимое в одном месте, делая процесс разработки более эффективным и простым.

Установка минимальной поддерживаемой версии Android в Android Studio подразумевает компромисс. Более высокая минимальная версия упрощает разработку, позволяя легко использовать новые функции Android, не беспокоясь о старых устройствах.

Однако это сокращает потенциальную аудиторию, исключая старые устройства. И наоборот, выбор более низкой минимальной версии увеличивает охват приложения, но вносит проблемы совместимости во время разработки. В конечном счете, выбор минимальной поддерживаемой версии Android обеспечивает баланс между простотой разработки и потенциальным размером аудитории. Опция «Помощь в выборе» отображает список версий Android с соответствующим охватом аудитории, помогая в принятии этого осознанного компромисса.

3.2 Создание приложения для взаимодействия с базой данных

Разработка Android-приложений состоит из нескольких ключевых этапов, включая создание активностей – компонентов, представляющих собой отдельные экраны пользовательского интерфейса. Интерфейс активности создается с использованием XML-файлов, определяющих структуру и стиль визуальных компонентов. Процесс разработки начинается с создания нового проекта в Android Studio. Первый, критически важный шаг - это выбор уникального имени для проекта, которое будет использоваться в качестве идентификатора пакета и во всех файлах проекта. Далее необходимо указать, на какие типы устройств будет нацелено приложение (например, смартфоны, планшеты, Android TV или смарт-часы). Выбор целевой платформы оказывает значительное влияние на дизайн пользовательского интерфейса, доступные API и, следовательно, на общую архитектуру приложения.

На Рисунке 15 показано окно выбора целевой платформы в процессе настройки проекта Android-приложения. Выбор конкретного устройства оказывает существенное влияние на структуру проекта и последующую разработку интерфейса. При разработке приложений для смартфонов и планшетов необходимо учитывать различные размеры экранов, обеспечивая адаптивный дизайн, который автоматически подстраивается под разные разрешения. При разработке для Android TV приоритетом является реализация удобной навигации с помощью пульта дистанционного управления, а также оптимизация интерфейса для большого экрана. Для смарт-часов требуется максимально упрощенный интерфейс, адаптированный для сенсорного управления и небольших экранов. Таким образом, выбор платформы на начальном этапе диктует дальнейшую стратегию разработки, определяя особенности проектирования пользовательского интерфейса и требуемые функциональные возможности.

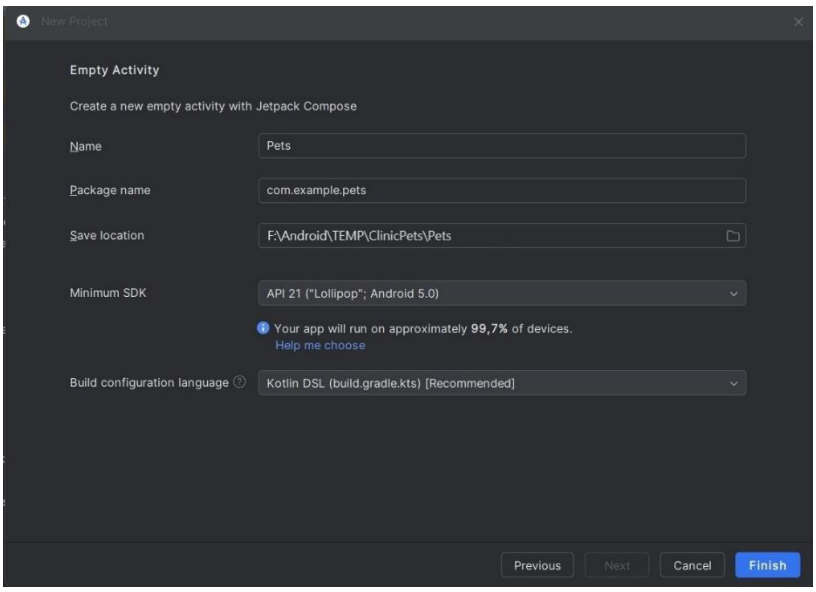

Рисунок 15 – Окно выбора устройства в Android Studio

Начальный выбор целевого устройства имеет критическое значение и закладывает основу для всего дальнейшего процесса разработки Android-приложения. Он определяет не только набор технологий, но и подходы к созданию пользовательского интерфейса и логики приложения. После выбора целевого устройства и создания проекта в Android Studio, разработчики приступают к реализации активностей и разработке пользовательского интерфейса (UI) на основе XML-файлов. Этот этап требует тщательного планирования и тестирования, чтобы обеспечить стабильную работу приложения на выбранной платформе, учитывая особенности различных экранов, способов взаимодействия и аппаратных возможностей устройств. Поскольку разрабатываемое приложение ориентировано на мобильные устройства, правильный выбор при настройке проекта становится особенно важным.

При создании нового проекта в Android Studio, рекомендуется выбрать «Мобильное устройство» для начала разработки, предназначенной для смартфонов или планшетов. После выбора мобильного устройства следующим этапом является выбор шаблона для приложения. Android Studio предоставляет широкий выбор шаблонов, как показано на рисунке 16,

предназначенных для упрощения разработки, предоставляя готовые базовые структуры и компоненты.Эти шаблоны, такие как Empty Activity, Basic Activity и Bottom Navigation Activity, позволяют разработчикам сосредоточиться на уникальных функциях и дизайне, экономя время на первоначальную настройку.

Выбор подходящего шаблона является важным этапом в разработке Android-приложения, поскольку он оказывает непосредственное влияние на исходную архитектуру проекта, а также определяет набор компонентов и библиотек, подключаемых по умолчанию. Правильно выбранный шаблон не только значительно ускоряет процесс разработки, предоставляя готовую структуру и основные элементы, но и задает архитектурные принципы, которые следует учитывать при дальнейшем развитии приложения.

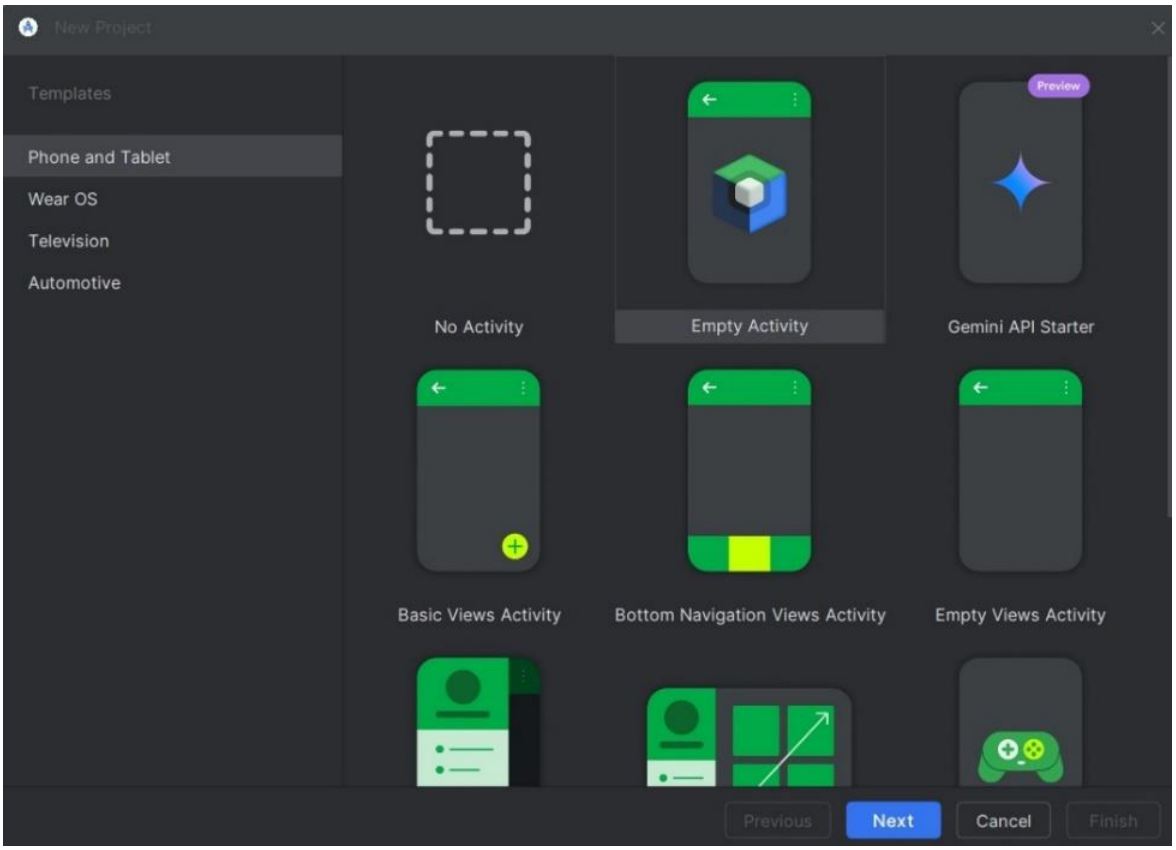

Рисунок 16 – Интерфейс выбора шаблона в Android Studio

После выбора нужных шаблонов запускается окно разработки приложения, рисунок 17 предоставлен ниже.

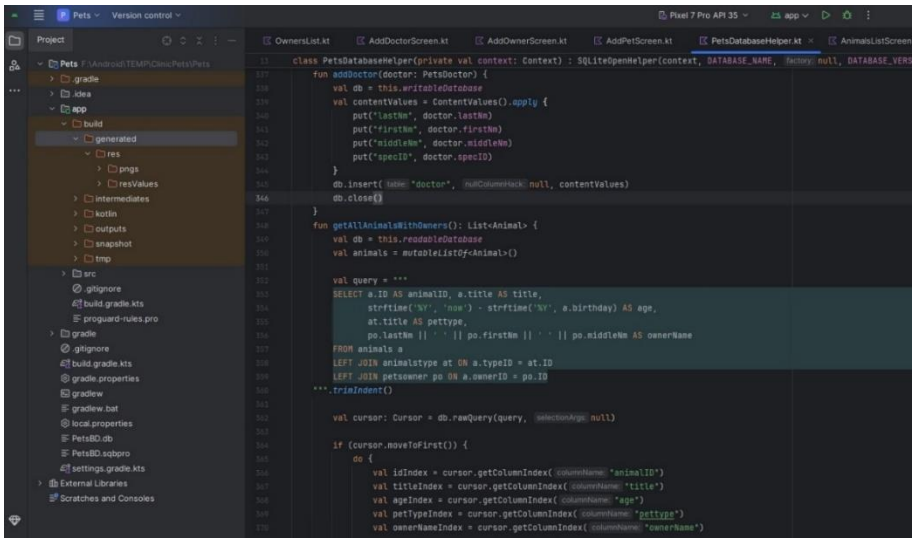

Рисунок 17 – Интерфейс проекта в Android Studio

Слева предоставлена структура проекта в Android Studio, дублирую информацию в рисунке 18.

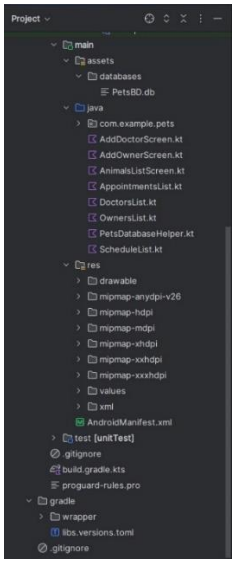

Рисунок 18 – Структура проекта в Android Studio

Структура представляет собой стандартную настройку проекта Android, которая делит проект на исходный код, ресурсы и файлы конфигурации. Соглашение об именовании указывает на приложение, которое, вероятно, связано с управлением домашними животными, встречами и связанными данными. Он также содержит базы данных для хранения этой информации. App – это основной модуль приложения Android. Main – содержит основной исходный код и ресурсы для приложения. Assets – для хранения произвольных файлов, таких как шрифты, текстуры и другие данные. Databases – хранит файл базы данных SQLite с именем PetsBD.db. Приложение использует локальную базу данных для хранения данных о домашних животных.

Java – содержит исходный код Kotlin/Java, Com.example.pets – это имя пакета, которое организует классы [24].

Набор файлов Kotlin (.kt):

- addDoctorScreen.kt;
- addOwnerScreen.kt;
- animalsListScreen.kt;
- appointmentsList.kt;
- doctorsList.kt;
- ownersList.kt;
- petsDatabaseHelper.kt;
- scheduleList.kt.

Эти файлы, представляют различные действия или классы обработки данных, связанные с приложением «Pets». Файлы предполагают такие функции, как добавление врачей, владельцев, список животных, встречи, врачей, владельцев, управление базой данных и планирование.

Res – содержит все ресурсы, такие как макеты, изображения, строки и стили пользовательского интерфейса. Drawable – для хранения графических ресурсов (например, значков, фонов).

Mipmap – для хранения значков запуска приложений с различной плотностью для поддержки различных размеров экрана. mipmap-hdpi, mipmap-

mdpi, mipmap-xhdpi, mipmap-xxhdpi, mipmap-xxxhdpi, mipmap-anydpi-v26 представляют собой версии значков для различных плотностей экрана. Values – содержит XML-файлы, определяющие константы, стили, темы, цвета, строки. Xml – используется для хранения XML-файлов ресурсов. AndroidManifest.xml – важный файл, описывающий важную информацию о вашем приложении для системы Android. Test – (папка с именем test [unitTest]) содержит модульные тесты [25].

Gitignore – еказывает намеренно неотслеживаемые файлы, которые Git должен игнорировать.

Build.gradle.kts (модуль) – содержит конфигурации сборки, специфичные для модуля приложения.

Proguard-rules.pro – определяет правила для ProGuard, инструмента, используемого для сжатия и обфускации.

Скрипты Gradle:

- gradle используется для определения версии gradle по умолчанию для проекта;
- wrapper используется для упаковки gradle;
- libs.versions.toml содержит версии различных библиотек;
- build.gradle.kts (проект) содержит настройки сборки на уровне проекта;
- .kt исходный код Kotlin.

Файл PetsDataBaseHelper.kt отвечает за подключение и работы с базой данных.

В нем находятся методы, с помощью которых осуществляется работа с базой данных.

Рассмотрим некоторые из них. Так как извлечение данных для всех объектов имеет одинаковый алгоритм, то укажем только по одному методу для каждой операции.

На рисунке 19 представлен код метода getAllOwners(), который используется для извлечения необходимых данных.

Рисунок 19 – Метод getAllOwners()

Этот метод предназначен для извлечения полного списка владельцев животных из таблицы petsowner в базе данных.

val db = this.readableDatabase - открываем базу данных в режиме чтения.

val owners = mutableListOf<PetsOwner>() — создаем пустой изменяемый список для хранения объектов PetsOwner.

val cursor: Cursor = db.rawQuery("SELECT * FROM petsowner", null) - выполняем SQL-запрос для получения всех записей из таблицы petsowner. Результат запроса возвращается в виде объекта Cursor.

if (cursor.moveToFirst()) - перемещаем курсор на первую запись. Если данные есть, начинаем их обработку.

Используем метод getColumnIndex, чтобы получить индексы столбцов по их именам. Если столбец не найден, метод возвращает -1.

Для каждого столбца проверяем, что индекс не равен -1. Если индекс корректен, извлекаем данные с помощью методов getInt или getString. Если индекс равен -1, используем значения по умолчанию (0, "", null).

На основе извлеченных данных создаем объект PetsOwner и добавляем его в список owners. While (cursor.moveToNext()) - перемещаем курсор на следующую запись и повторяем процесс, пока не обработаем все записи.

cursor.close() - закрываем курсор, чтобы освободить ресурсы. Db.close() - закрываем соединение с базой данных. Return owners - возвращаем список объектов PetsOwner. На рисунке 20 рассмотрим метод addDoctor(doctor: PetsDoctor) добавления данных доктора.

```
fun addDoctor(doctor: PetsDoctor) {
    val db = this.writableDatabase
    val contentValues = ContentValues().apply
{       put("lastNm", doctor.lastNm)
        put("firstNm", doctor.firstNm)
        put("middleNm", doctor.middleNm)
        put("specID", doctor.specID)
    }
    db.insert("doctor", null, contentValues)
    db.close()
}
```

Рисунок 20 – Метод addDoctor(doctor: PetsDoctor)

Val db = this.writableDatabase - открываем базу данных в режиме записи. Это позволяет выполнять операции вставки, обновления и удаления данных. Val contentValues = ContentValues() - создаем объект ContentValues, который используется для хранения данных в виде пар "ключ-значение". Это необходимо для вставки данных в таблицу. Используем метод put, чтобы добавить данные врача в ContentValues:

- put("lastNm", doctor.lastNm) - добавляем фамилию врача.
- put("firstNm", doctor.firstNm) - добавляем имя врача.
- put("middleNm", doctor.middleNm) - добавляем отчество врача (если есть).
- put("specID", doctor.specID) - добавляем идентификатор специальности врача.

– db.insert("doctor", null, contentValues) - вставляем данные из ContentValues в таблицу doctor.

Первый аргумент "doctor" - имя таблицы. Второй аргумент (null) указывает, что в случае конфликта (например, при дублировании первичного ключа) операция вставки будет отменена. Третий аргумент (contentValues) - данные для вставки.

db.close() - закрываем соединение с базой данных, чтобы освободить ресурсы.

Для работы с каждым объектом имеются свои формы. В работе есть следующие формы:

– addDoctorScreen – форма для добавления данных о докторе;
– addOwnerScreen – форма для добавления данных о владельце;
– addPetsScreen – форма для добавления данных о животных;
– animalsListScreen – форма для просмотра данных о животных;
– appointmentsList – форма для просмотра расписания;
– doctorsList – форма для просмотра данных о докторе;
– ownersList – форма для просмотра данных о владельцах;
– mainActivity – главная форма проекта.

Данные формы снабжены комментариями и находятся в Приложении А.

3.3 Описание функционала приложения

При запуске приложения открывается главный экран с расписанием врачей, расположен на рисунке 21.

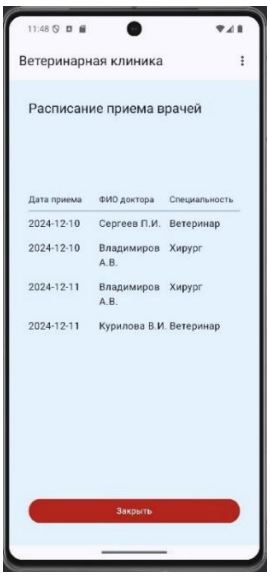

Рисунок 21 – Главный экран расписания приема врачей

Затем, нажав на троеточие в правом верхнем углу можно перейти к функционалу приложения. Далее необходимо внести данные о владельце (Рисунок 22).

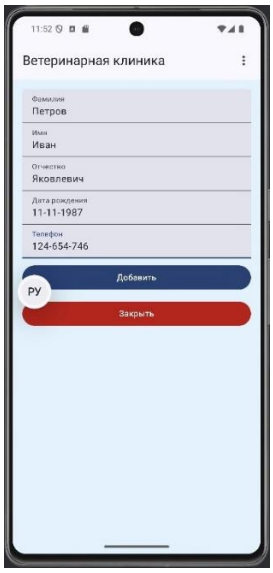

Рисунок 22 – Внесение владельца

Обязательно нужно нажать кнопку «Добавить», чтобы информация о владельце добавилась в базу данных. Об этом поступит уведомление на

экране, что хозяин добавлен и перекинет в окно, где есть вся информация о владельцах (Рисунок 23).

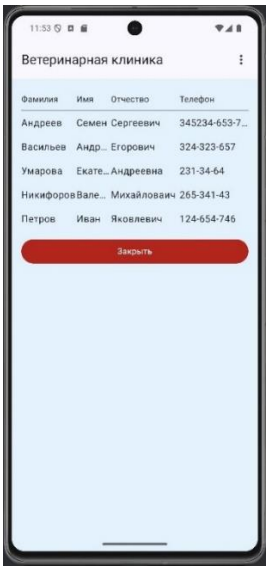

Рисунок 23 – Данные о владельцах

Далее можно перейти в отдел для выбора врача и добавить информацию нем, проиллюстрировано на рисунке 24.

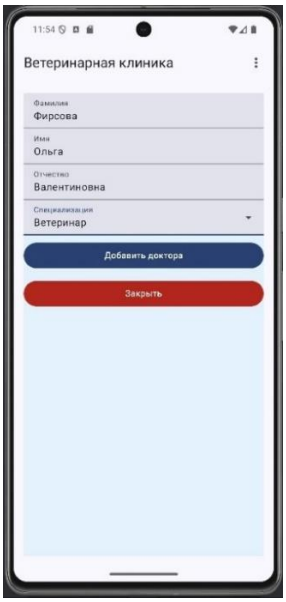

Рисунок 24 – Добавление информации о враче

После ввода всех данных о враче необходимо также нажать на кнопку «Добавить врача», чтобы информация записалась в базу данных. На экране появится уведомление что доктор добавлен.

Следующим шагом будет внесение информации о питомце, выбираем из выпадающего списка ячейку «Питомцы», далее открывается окно, где нужно указать кличку питомца и его вид, изображено на рисунке 25.

Рисунок 25 – Добавление информации о питомце

Обязательным шагом будет выбрать владельца, из тех владельцев, кто у нас был добавлен ранее и выбрать тип животного. Предоставлено на рисунках 26 и 27.

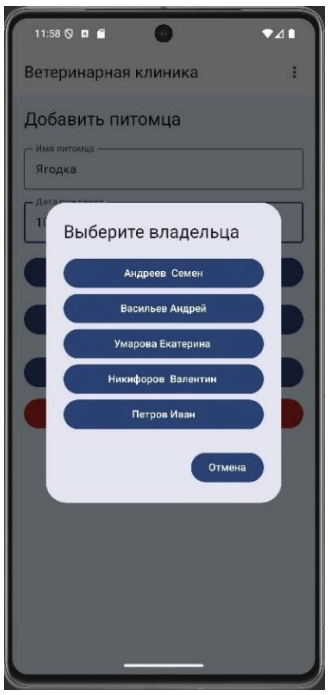

Рисунок 26 – Выбор владельца

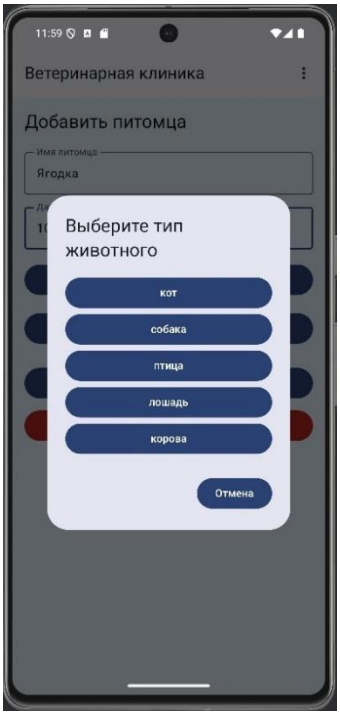

Рисунок 27 – Выбор типа животного

В приложении также присутствует возможность просмотреть всю добавленную информацию о питомцах, необходимо нажать на троеточие и выбрать «Просмотр данных», показано на рисунке 28.

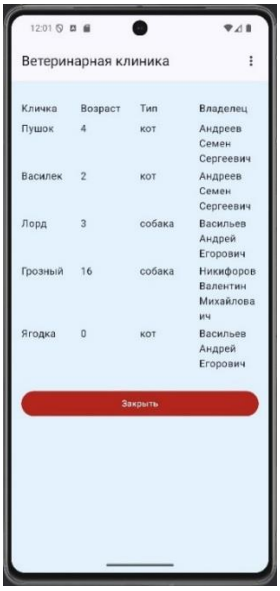

Рисунок 28 – Просмотр данных

В интерфейсе также предусмотрено закрытие приложения соответствующей кнопкой.

Просмотр данных в мобильном приложении включает в себя несколько важный аспектов, которые необходимо учитывать:

Удобный интерфейс — важно, чтобы пользовательский интерфейс был интуитивно понятным и адаптивным. Элементы управления должны быть простыми для навигации и предоставлять удобный доступ к информации.

Формат представления данных — данные могут отображаться в различных форматах: списках, таблицах, карточках. Выбор формата зависит от типа информации и предпочтений пользователей.

Фильтрация и поиск — пользователям должна быть доступна возможность фильтровать и искать данные по различным критериям. Это позволит быстрее находить нужную информацию.

Навигация между данными — пользователи должны иметь возможность легко перемещаться между различными разделами приложения и взаимодействовать с данными, например, переходить к деталям элемента списка.

3.4 Проведение тестирования мобильного приложения

По завершении разработки мобильного приложения наступил важный этап, в рамках которого было решено провести детальное бета-тестирование. Этот процесс направлен на выявление возможных ошибок и оптимизацию общей производительности продукта. Чтобы упростить процесс бета-тестирования, было найдено реальное устройство и пользователь, который имеет свободное время. Это позволило более точно воспроизвести различные размеры экрана и операционную систему. Этот метод также способствовал обнаружению возможных проблем и гарантировал стабильную и высококачественную работу приложений на предусмотренных устройствах.

Ниже предоставлено краткое описание тест-кейсов для бета-тестирования мобильного приложения «Запись на прием к ветеринарному врачу».

№	Тест-кейс	Шаги	Ожидаемый результат	Результат
1	Регистрация нового пользователя	- Открыть приложение. - Выбрать опцию регистрации. - Ввести обязательные данные (имя, email, телефон, пароль). - Нажать «Зарегистрироваться».	- Пользователь успешно зарегистрирован. - Появляется сообщение об успешной регистрации.	Успешно
2	Запись на прием	- Войти в приложение. - Перейти в раздел «Запись на прием». - Выбрать ветеринара и время. - Подтвердить запись.	- Запись успешно создана. - Пользователь получает уведомление о записи.	Успешно
3	Изменение записи на прием	- Войти в приложение. - Перейти в «Мои записи». - Выбрать запись. - Изменить время. - Подтвердить изменения.	- Изменения успешно сохранены. - Появляется подтверждение об изменении записи.	Успешно
4	Удаление записи на прием	- Войти в приложение. - Перейти в «Мои записи». - Выбрать запись для удаления. - Подтвердить удаление.	- Запись успешно удалена. - Пользователь получает уведомление о подтверждении удаления.	Успешно

Рисунок 29 – Тест-кейсы для тестирование мобильного приложения

Для бета-тестирования использовался мобильный мобильный телефон с ОС Android Samsung Galaxy A54 5G с диагональю экрана 6.4 дюйма и разрешением 2340×1080 пикселей в вертикальной ориентации. На рисунке 30 изображен процесс бета-тестирования записи питомца на прием к врачу, продемонстрированный на мобильном телефоне с ОС Android Samsung Galaxy A54.

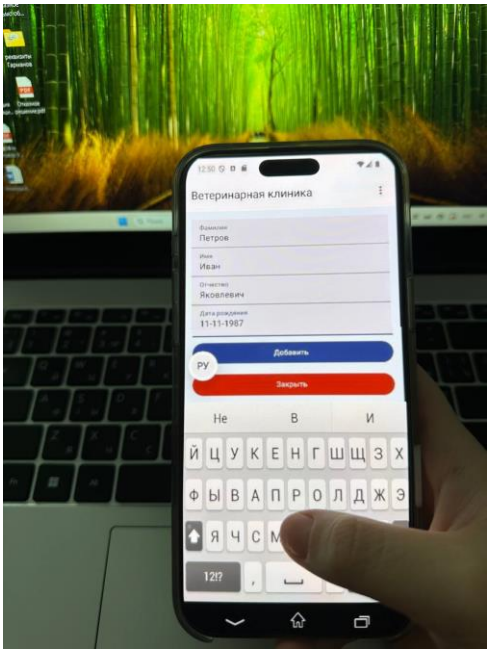

Рисунок 30 – бета тестирование мобильного приложения

После успешного завершения бета-тестирования на реальных устройствах, была получена более точная оценка производительности. Мобильное приложение «Запись на прием к ветеринарному врачу» обладает следующими качествами:

– эффективность, система работает надежно и быстро выполняет задачи;
– гибкость, изменения и обновления могут быть внедрены легко и быстро;

– расширяемость, новые функции и элементы для ОС Android могут быть добавлены без серьезных сбоев.

Вывод по третьей главе

В главе 3 был подробно рассмотрен процесс разработки мобильного приложения для платформы Android, использовав язык программирования Java и интегрированную среду разработки (IDE) Android Studio. Был проведен анализ выбора Java как основного языка программирования, обоснованный широкой поддержкой Android и обширным набором инструментов для разработки, предоставляемых Android Studio. Android Studio, как официальная IDE для Android, которая предлагает удобную среду для создания, отладки, тестирования и сборки приложений, обеспечивая эффективный процесс разработки от начала до конца.

Были описаны ключевые возможности Android Studio, такие как интеллектуальный редактор кода, гибкая система сборки на базе Gradle, быстрый эмулятор, инструменты отладки, визуальный редактор макетов, анализатор APK, профилировщики в реальном времени, поддержка языков Java и Kotlin, а также интеграция с сервисами Google.

Особое внимание уделено выбору минимальной поддерживаемой версии Android, который представляет собой компромисс между использованием новых возможностей платформы и охватом аудитории устройств.

Также раскрыт процесс создания приложения для взаимодействия с базой данных, подчеркнута роль активностей как экранов пользовательского интерфейса, формируемых с помощью XML-файлов. Также рассмотрены начальные шаги разработки проекта в Android Studio, включая выбор имени проекта и целевой платформы, что влияет на настройки и функциональность приложения. Таким образом, глава содержит подробный обзор технических аспектов и инструментов, необходимых для эффективной разработки мобильного приложения под Android.

Заключение

В рамках данной бакалаврской работы было разработано мобильное приложение для записи на прием к ветеринарному врачу. Это приложение отвечает современным требованиям удобства и функциональности, обеспечивая владельцам домашних животных простоту и быстроту в записи на прием, а также возможность получать актуальную информацию о состоянии здоровья своих питомцев.

Мобильное приложение предоставляет владельцам домашних животных простой и интуитивно понятный интерфейс для быстрой записи на прием, выбора специалиста и просмотра доступного расписания, включая историю посещений. Чтобы добиться намеченных целей, были реализованы следующие задачи:

- раскрытие тонкостей предметной области и точное определение функциональных требований и спецификаций интерфейса;
- создание надежной архитектуры приложения, охватывающей структуру, компоненты и протоколы взаимодействия;
- реализованы основные функциональные модули приложения, обеспечивающие запись на прием, просмотр расписания, управление данными о животных и клиентах;
- разработан удобный и интуитивно понятный пользовательский интерфейс, ориентированный на различные категории пользователей;
- проведено тестирование приложения и оценена его производительность, надежность и соответствие требованиям.

Создание данного мобильного приложения является важным достижением в области улучшения доступа к ветеринарным услугам и повышения уровня обслуживания владельцев домашних животных.

Разработанное приложение предоставляет пользователям удобный и функциональный инструмент для управления записями на прием, выбора специалиста, просмотра расписания и получения оперативной информации.

Процесс разработки, основанный на глубоком анализе потребностей пользователей и итеративной разработке с постоянным тестированием, ставил удовлетворение пользовательских требований в приоритет.

В результате проведенного тестирования было подтверждено, что приложение не только стабильно работает и соответствует техническим требованиям, но и полностью отвечает потребностям и ожиданиям пользователей, обеспечивая удобство и эффективность записи на прием и взаимодействия с ветеринарной клиникой. Тестирование подтвердило, что приложение стабильно работает и отвечает потребностям пользователей, и является прочной основой для дальнейшего развития и расширения функциональности в будущем.

Мобильное приложение предоставляет удобный инструмент для управления записями и взаимодействия с ветеринарами, позволяя владельцам животных легко записываться на прием, выбирать врача и время и просматривать историю посещений.

Список используемой литературы и используемых источников

1. Исакова С., Жемеров Д. Kotlin in Action, 2016. – 352 с.
2. История Android: от стартапа до самой популярной мобильной платформы в мире [Электронный ресурс]. URL: https://www.kommersant.ru/doc/6235991?ysclid=m7spf2w4130548357 (дата обращения 03.03.2025).
3. Как пользоваться Android Studio [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/kak-polzovatsya-android-studio/ (дата обращения 12.02.2025).
4. Основы проектирования баз данных [Электронный ресурс]. URL: https://appmaster.io/ru/blog/osnovy-proektirovaniia-baz-dannykh (дата обращения 25.02.2025).
5. Подробно о Xamarin [Электронный ресурс]. URL: https://habr.com/ru/articles/188130/ (дата обращения 12.02.2025).
6. Рекомендации по дизайну материалов Google [Электронный ресурс]. URL: https://material.io/design (дата обращения 13.02.2025).
7. Чем отличаются функциональные и нефункциональные требования [Электронный ресурс]. URL https://dzen.ru/a/ZXwTmuUQoQmOACih (дата обращения 24.02.2025).
8. Что такое фреймворки и какие они бывают [Электронный ресурс]. URL: https://ru.hexlet.io/blog/posts/chto-takoe framework?ysclid=m7ufm8p2v01825095/ (дата обращения 04.03.2025).
9. Что такое Android Studio [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/kak-polzovatsya-android-studio/#chto-takoe (дата обращения 21.02.2025).
10. Что такое React Native [Электронный ресурс]. URL: https://habr.com/ru/articles/596183/ (дата обращения 12.02.2025).
11. Flutter [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Flutter (дата обращения 12.02.2025).

12. FURPS [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/FURPS/ (дата обращения 20.02.2025).

13. Iconic (mobile app framework) [Электронный ресурс]. URL: https://translated.turbopages.org/proxy_u/en-ru.ru.bb80f71e-67c6e7f4-8ae4aca674722d776562/https/en.wikipedia.org/wiki/Ionic_(mobile_app_framework) (дата обращения 04.03.2025).

14. Andrew Hunt, David Thomas, The Pragmatic Programmer : Abstract, 2019. 352 с.

15. Antonio Leiva, Kotlin for Android Developers : Abstract, 2018. 370 с.

16. Bill Phillips, Chris Stewart, Kristin Marsicano, Android Programming: The Big Nerd Ranch Guide : Abstract, 2017. 450 с.

17. Daniel A. McCulloch, Mobile App Development with Kotlin : Abstract, 2019. 300 с.

18. Dawn Griffiths, David Griffiths Head First Android Development : Abstract, 2015. 720 с.

19. Dinesh J. M., Swift for Android : Abstract, 2019. 260 с.

20. Google, Udacity: Developing Android Apps with Kotlin : Abstract, 2020. 150 с.

21. Greg Nudelman, Your First Mobile App : Abstract, 2020. 480 с.

22. Jason Fox, Pro Android Games : Abstract, 2017. 540 с.

23. Jonathan Levin, Android Internals: A Confectioner's Cookbook : Abstract, 2016. 600 с.

24. John Horton, Learning Java by Building Android Games : Abstract, 2015. 350 с.

25. John Horton, Android Development for Beginners : Abstract, 2018. 190 с.

26. Joshua Bloch, Effective Java : Abstract, 2018. 416 с.

27. Josh Skeen, David Greenhalgh, Kotlin Programming: The Big Nerd Ranch Guide : Abstract, 2018. 400 с.

28. Mark Murphy, The Busy Coder's Guide to Android : Abstract, 2021. 672 c.

29. Smyth, Android Studio 4.0 Development Essentials : Abstract, 2020. 320 c.

Приложение А

Исходный код программы

Исходный код программы расположен https://gitlab.com/pets14010/Pets