

Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК

Руководитель ОПОП

д-р. техн. наук, профессор

С.П. Сущенко С.П. Сущенко

« 03 » июня 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ УЧЕТА КЛИЕНТОВ

по направлению подготовки 09.03.03 Прикладная информатика
направленность (профиль) «Прикладная информатика»

Андреева Анна Александровна

Руководитель ВКР

канд. техн. наук, доцент

А.С. Шкуркин А.С. Шкуркин

« 28 » мая 2023 г.

Автор работы

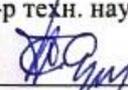
студент группы № 93/902

А.А. Андреева А.А. Андреева

« 28 » мая 2023 г.

Министерство науки и высшего образования Российской Федерации.
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)
Институт прикладной математики и компьютерных наук

УТВЕРЖДАЮ
Руководитель ОПОП
д-р техн. наук, профессор

 С.П. Сущенко
подпись

« 14 » ноября 2022 г.

ЗАДАНИЕ

по выполнению выпускной квалификационной работы бакалавра обучающегося
Андреевой Анне Александровне

Фамилия Имя Отчество обучающегося

по направлению подготовки 09.03.03 Прикладная информатика, направленность
(профиль) «Прикладная информатика»

1 Тема выпускной квалификационной работы
Мобильное приложение для учета клиентов

2 Срок сдачи обучающимся выполненной выпускной квалификационной работы:

а) в учебный офис / деканат – 28.05.2023 б) в ГЭК – 07.06.2023

3 Исходные данные к работе:

Объект исследования – процесс разработки мобильного приложения для ОС Android

Предмет исследования – мобильное приложение для учета клиентов

Цель исследования – спроектировать и разработать мобильное приложение для учета клиентов

Задачи:

Проанализировать предметную область, провести обзор существующих аналогов, сформировать техническое задание, выбрать инструменты разработки, спроектировать архитектуру мобильного приложения, реализовать приложение

Методы исследования:

Теоретическое исследование и практическая реализация

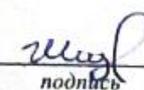
Организация или отрасль, по тематике которой выполняется работа, –

Национальный исследовательский Томский государственный университет

4 Краткое содержание работы

Проанализированы технологии для написания мобильного приложения, ссылаясь на техническое задание, выбрана платформа для работы приложения и среда разработки, разработано мобильное приложение для учета клиентов

Руководитель выпускной
квалификационной работы
канд. техн. наук, доцент кафедры ПИ
должность, место работы

 / А.С. Шкуркин
подпись / И.О. Фамилия

Задание принял к исполнению
студент группы 931902
должность, место работы

 / А.А. Андреева
подпись / И.О. Фамилия

АННОТАЦИЯ

Выпускная квалификационная работа бакалавра на тему «Мобильное приложение для учета клиентов» содержит 58 страниц пояснительной записки, 40 рисунков, 13 источников и 3 таблицы.

Ключевые слова: ПРОЕКТИРОВАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ, РАЗРАБОТКА МОБИЛЬНОГО ПРИЛОЖЕНИЯ, БАЗА ДАННЫХ, ANDROID.

Объект исследования: процесс разработки мобильного приложения для ОС Android

Цель работы: спроектировать и разработать мобильное приложение для учета клиентов.

Методы исследования: теоретическое исследование проводилось методом анализа литературы и нормативных справочников, функциональных обязанностей. Практическое обследование объекта проводилось на основе системного анализа.

Результат работы: разработан общий проект информационной системы объекта, обеспечивающий авторизованный вход для администраторов организации; вывод всех заявок на услуги. Система позволяет осуществлять редактирование добавление новых специалистов студии, а также управлением заявками на услуги (отклонить / принять), включая также возможность редактирования информации о заявке на услуги.

Приложение можно использовать и для других организаций специализируемых на: красота, спорт, медицина, образование, досуг и отдых, авто, розничные точки сбыта и прочий бизнес.

ОГЛАВЛЕНИЕ

Введение	3
1 Основы разработки мобильного приложения.....	4
1.1 Компоновка мобильного приложения	4
1.2 Типы мобильных приложений	4
1.3 Виды мобильных приложений	7
1.4 Обзор мобильных операционных систем	8
1.5 Основные этапы разработки мобильного приложения	9
1.6 Анализ аналогичных приложений и источников	10
1.6.1 Обзор литературы	10
1.6.2 Анализ аналогичных приложений.....	11
1.7 Вывод по первой главе	14
2 Проектирование мобильного приложения.....	15
2.1 Описание технического задания	15
2.2 Среда верстки мобильного приложения	16
2.3 Схемы расположения элементов мобильного приложения	17
2.4 Разработка Фреймворка для работы с базой данных.....	23
2.5 Вывод по второй главе.....	29
3 Описание разрабатываемого приложения	30
3.1 Этапы разработки пользовательского интерфейса.....	30
3.1.1 Описание навигационных пунктов.....	30
3.2 Вывод по третьей главе	39
Заключение.....	40
Список использованных источников	41
Приложение А.....	42
Приложение Б.....	47

ВВЕДЕНИЕ

Мобильные приложения являются неотъемлемой частью современной жизни и бизнеса. Они позволяют более эффективно управлять бизнесом, повышать удобство использования услуг для клиентов и улучшать качество обслуживания.

Приложения для учета клиентов востребованы во многих отраслях, таких как торговля, ресторанный бизнес, здравоохранение и многие другие. Ежедневно многие компании сталкиваются с проблемой учета клиентов, и мобильные приложения могут стать отличным решением данной проблемы.

Также стоит отметить, что рынок мобильных приложений постоянно растет, и разработчики приложений должны уметь создавать качественные продукты, учитывая потребности пользователей и требования рынка.

В связи с этим, тема "Мобильное приложение для учета клиентов" актуальна и имеет большой потенциал для развития и применения в бизнесе.

Как показано на рисунке 1.1, большинство людей используют мобильные устройства на базе OS Android. Поэтому для разработки мобильного приложения была выбрана платформа OS Android. Эта операционная система также является одной из наиболее простых и одновременно комплексных платформ.



Рисунок 1.1 – Доля OS Android по сравнению с iOS

Цель выпускной квалификационной работы: спроектировать и разработать мобильное приложение для учета клиентов.

Для реализации поставленной цели были поставлены следующие задачи:

1. Проанализировать предметную область;
2. Провести обзор существующих аналогов;
3. Сформировать техническое задание;
4. Выбрать инструменты разработки;
5. Спроектировать архитектуру мобильного приложения;
6. Реализовать приложение.

1 Основы разработки мобильного приложения

1.1 Компоновка мобильного приложения

Мобильное приложение это программное обеспечение, созданное специально для определенной мобильной платформы (например, iOS, Android и т.д.), которое можно использовать на смартфонах, планшетах, умных часах и других мобильных гаджетах.

Разработка таких приложений требует использования языков программирования высокого уровня и технологий компиляции в машинный код, который обеспечивает максимальную производительность на мобильных устройствах. Но создание мобильных приложений имеет свои особенности:

- мобильные устройства работают от аккумулятора;
- комплектуются менее производительными процессорами, чем настольные ПК.

Кроме того, современные мобильные устройства имеют множество дополнительных модулей, таких как гироскопы, акселерометры, камеры, датчики приближения и освещения, которые могут быть использованы для расширения функционала приложения.

У каждой организации всегда есть выбор и очевидно, что ПК-версия сайта — это первое, что пытается разработать организация. Кроме того, разработав мобильное приложение, которое содержит все содержимое версии для ПК, можно получить множество преимуществ. Одним из них является то, что приложение можно установить на мобильное устройство, и вы можете использовать его в любое время и в офлайн режиме [1].

Разработчики могут разрабатывать мобильные приложения, которые доступны для каждой ОС телефона или планшета в отдельности. Мобильные приложения сегодня имеют особое значение для организаций и компаний – они в свою очередь уделяют большое внимание использованию Интернета. Совершенствование приложения для укрепления авторитета и репутации компании взаимосвязаны.

1.2 Типы мобильных приложений

Приложения для смартфонов стали частью жизни современных людей. Однако, нет гарантии, что каждое приложение будет успешным. Необходима хорошо продуманная концепция, анализ рынка и запросов пользователей, а также высокое качество продукта. Кроме того, важна поддержка и обновление приложения после его выпуска [2]. Для компаний создание приложений позволяет расширить свой бизнес и увеличить количество потенциальных клиентов. Они получают новые каналы коммуникации с аудиторией и могут использовать приложение в качестве инструмента

маркетинга и продвижения своих товаров или услуг.

В целом, мобильные приложения играют важную роль в современном мире и будут продолжать развиваться и улучшаться. Создание качественных и удобных приложений является выгодным для всех участников рынка и может стать ключевым фактором успеха любой компании. Разработчики приложения получают прибыль за счет:

- продажи рекламных блоков;
- платных дополнительных функций;
- платного распространения приложения;
- создание контента на заказ.

Также необходимо обеспечить правильную маркетинговую стратегию и продвижение приложения в социальных сетях и других каналах. Кроме того, важен постоянный анализ и улучшение качества приложения в соответствии с потребностями пользователей. Все эти факторы влияют на доходы разработчика и успех приложения на рынке.

Все типы приложений можно разделить на 4 категории:

1. мобильные игры;
2. промо приложения;
3. контентные-сервисы;
4. социальные сети.

Существуют разнообразные программы, включая сервисы для управления компаниями и другие типы. Однако они менее популярны, чем уже упомянутые. Статистика показывает, что у 70% пользователей смартфонов есть хотя бы одна программа из перечисленных категорий.

Игровые – мобильные игры разнообразны и включают в себя различные жанры: стрелялки, гонки, аркады, квесты, задачи на логику и многое другое. В эту категорию включены как детские, так и взрослые игры. Основная аудитория мобильных игр - люди моложе 27 лет, но с каждым годом старшее поколение все больше интересуется играми. Основной способ заработка на играх - продажа дополнительного контента. На рисунке 1.2 показана мобильная игра



Рисунок 1.2 – Мобильная игра

Под заказ разрабатываются **промо-приложения**, которые помогают бизнесу продвигать свой бренд. Основная цель разработчиков заключается в том, чтобы охватить наибольшее количество клиентов, которые пользуются различными моделями смартфонов. Пользователи могут воспользоваться сервисами для заказа товаров и услуг, выражения своих отзывов, получения информации о скидках и акциях, а также получения бонусов. Наиболее популярными сервисами являются те, которые позволяют заказывать еду на доставку, вызывать такси и покупать билеты на киносеансы. На рисунке 1.3 показано промо-приложение от компании Coca-Cola.



Рисунок 1.3 – Промо-приложение от Coca-Cola

Контентные программы обеспечивают быстрый доступ к специфическому контенту и охватывают разнообразное количество приложений для ОС Android и iOS. Некоторые из примеров контента, которые могут быть включены:

- публикация новостей;
- вдохновляющие цитаты;
- системы похудения и тренировок;
- текущие курсы валют;
- образовательные языковые курсы.

Для генерации дохода на контентных сервисах используются рекламные блоки, и пользователи могут отключать рекламу за дополнительную плату, что увеличивает бюджет проекта. Пример новостного приложения показан на рисунке 1.4.



Рисунок 1.4 – Новостное приложение от Google

Социальные сети – используются для общения и связи с другими людьми. В эту группу входят мессенджеры, социальные сети и программы для видео - и аудиозвонков. Они позволяют быть на связи с друзьями и коллегами, обмениваться информацией и фотографиями, создавать группы для общения и организации совместных мероприятий. Некоторые из них, например, WhatsApp и Viber, также позволяют совершать бесплатные звонки через интернет. У каждой популярной соц. сети есть мобильное приложение. Некоторые из них могут быть установлены еще до покупки смартфона. На рисунке 1.5 показан скриншот приложение социальной сети «ВКонтакте».

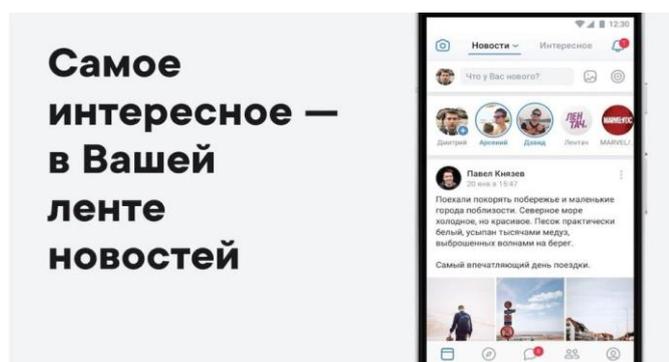


Рисунок 1.5 – Социальная сеть VK

1.3 Виды мобильных приложений

Кроме сортировки по назначению, существует другой тип классификации мобильных приложений, который разделяет программы на три группы в зависимости от их особенностей: гибридные, нативные и приложения для мобильных сайтов. Давайте более детально рассмотрим эти типы мобильных приложений.

Гибридные приложения находятся между нативными и веб-приложениями и имеют ограниченный доступ к аппаратной части устройства, такой как камера, микрофон, геолокация, адресная книга. Они зависят от интернет-соединения, так как загружают контент с внешнего источника, который размещен на сервере. Большинство промо-сервисов принадлежит к этой категории.

Нативные приложения разрабатываются для конкретной операционной системы, такой как iOS, Android и другие. Чтобы охватить более широкую аудиторию, необходимо разработать несколько отдельных приложений для разных операционных систем. Могут иметь одни и те же функции и одинаковый дизайн, но будут различаться программами. Это увеличивает срок работы над проектом и бюджет на разработку.

Нативные сервисы могут работать вне зависимости от подключения к сети интернет, но некоторые из них требуют наличия подключения. Они работают быстро, требуют меньше памяти, тратят небольшое количество заряда батареи.

Веб-приложения представляют собой адаптацию веб-сайтов для пользователей компьютеров и смартфонов. Их создание позволяет посетителям получать доступ к сайту в любое время суток, независимо от наличия персонального компьютера или ноутбука. В отличие от некоторых веб-сервисов, которые нужно загружать и устанавливать, другие веб-приложения автоматически запускаются при посещении сайта через мобильный браузер.

1.4 Обзор мобильных операционных систем

В мире имеется большой выбор языков программирования для того, чтобы разработать мобильные приложения т.к. для разных мобильных устройств нужно применять разные языки программирования. Это зависит от того, что у мобильных устройств разные операционные системы. Я ознакомился со следующими SDK:

1. xCode (iPhone SDK);
2. Android Studio (Android SDK).

Платформа iOS (iPhone SDK)

Под iPhone (а также iPad) разрабатывать рекомендуется на языке Swift (также SwiftUI) или ObjectiveC (при использовании Storyboard). Так же написание приложения может производиться на таких языках как: C и C++. Отладка происходит при помощи среды xCode и симулятора, либо при наличии живого устройства подключенного по USB, либо по воздуху (Wi-Fi).

Платформа OS Android (Android SDK)

Под OS Android вы можете использовать среду Android Studio, которая является более новой средой для разработки и также быстро завоевывающая популярность как

интеллектуальная и удобная. Android Studio активно развивается и поддерживается компанией Google в качестве официальной среды разработки приложений под OS Android.

Под Android рекомендуется разрабатывать приложения на языке Kotlin, а также при необходимости вы можете комбинировать Kotlin с языком Java.

Kotlin [3] – статический типизированный, объектно-ориентированный язык программирования, работающий поверх JVM (Java Virtual Machine) и разрабатываемый компанией JetBrains. Также компилируется в JavaScript и в исполняемый код ряда платформ через инфраструктуру LLVM. Язык назван в честь острова Котлин в Финском заливе, на котором расположен город Кронштадт.

Android-аудитория не только широкая, но и очень разнообразная. Установить приложения на Android могут люди разного достатка и возраста.

Компании - разработчики устройств Android делают свои устройства не только в сегменте премиум устройств, но и предлагают свои бюджетные варианты.

Мною была выбрана среда разработки «Android Studio» т.к. она является официальной средой разработки от компании Google. А также языком программирования в этой среде был выбран – Kotlin.

1.5 Основные этапы разработки мобильного приложения

Создание мобильного приложения [4] – это комплексный, скрупулезный процесс, который предполагает четкое следование плану, соблюдение сроков и достижение взаимопонимания между заказчиком и разработчиком. Весь процесс от идеи до реализации можно разделить на несколько ключевых этапов.

Этапы разработки мобильного приложения:

- 1. Исследование идеи.** На данном этапе нужно понять и определить идею приложения, какие функции оно должно содержать, какие задачи выполнять и для кого;
- 2. Проектирование.** Создание карты, демонстрирующей функционал будущего приложения. Как правило, карта содержит схему экранов и переходов между ними;
- 3. Создание дизайна.** Разработка графических элементов: экранов, фонов, иконок, кнопок и др. Данный этап предполагает проверку на юзабилити – нужно определить, насколько удобно будет пользоваться элементами интерфейса приложения;
- 4. Составление подробного технического задания (ТЗ).** Техническое задание содержит подробное описание функционала мобильного приложения, бизнес-процессы и основные сценарии, которые должны быть в нем реализованы. Техническое задание (ТЗ) составляет заказчик, либо компания-разработчик по требованиям заказчика. После

подготовки технического задания (ТЗ) можно точно оценить временные и денежные затраты на разработку приложения;

5. **Создание прототипов.** Прототип нужен, чтобы показать, как будет работать приложение, он может быть интерактивным и статичным;

6. **Разработка.** После согласования технического задания (ТЗ) и прототипов с заказчиком наступает этап активной разработки. В комплексных и сложных проектах рекомендуется применять MVP (minimum viable product) – минимальный реализованный функционал продукта, который помогает в оценке будущего приложения со стороны заказчика и в дальнейшем планировании разработки;

7. **Тестирование.** После того, как разработка завершена, нужно протестировать финальную версию на мобильных устройствах. На данном этапе выявляют и вычищают все шероховатости и недоработки приложения, готовят его к полноценному релизу;

8. **Передача клиенту** или публикация в магазин приложений (Google Play, AppStore) – это финальный этап разработки, после которого приложение переходит в стадию поддержки. После выявления и исправления всех ошибок и финального согласования с заказчиком, приложение публикуется в Google Play и AppStore. Стоит обратить внимание на то, что после публикации в данных сервисах, можно вносить изменения в приложение без прохождения повторной модерации, что очень удобно для заказчиков;

9. **Техническая поддержка и мониторинг.** За все время пользования вашим приложением будут возникать те или иные технические сложности, ошибки и нештатные ситуации, с которыми будут сталкиваться пользователи. На данном этапе происходит оперативная поддержка и (если потребуется) доработка приложения.

1.6 Анализ аналогичных приложений и источников

1.6.1 Обзор литературы

Рето Майер: «Android 4. Программирование приложений для планшетных компьютеров и смартфонов» [5]. Эта книга - идеальное руководство для тех программистов, кто хочет научиться разрабатывать приложения для мобильного Android. Она представляет собой практический курс по созданию программного обеспечения на базе Android SDK, который обеспечивает абсолютную близость к реальным задачам при помощи примеров. Книга будет полезной для опытных разработчиков-специалистов, которые могут использовать ее как справочник, а также для начинающих в сфере разработки мобильных приложений для Android.

Сильвен Ретабоуил: «Android NDK. Руководство для начинающих» [6]. Откройте

доступ к внутренней природе Android и добавьте мощь C/C++ в свои приложения.

Книга покажет, как создавать мобильные приложения для платформы Android, используя язык C / C ++ и пакет библиотек NDK, и как объединять их программный код на языке Java. В ней вы узнаете, как создавать первые приложения низкого уровня для Android, как взаимодействовать с программным кодом на Java с помощью механизма Java Native Interfaces, как соединить вывод графики и звука, обрабатывать устройства ввода и датчики, отображать графику с помощью библиотеки OpenGL ES и многого другого.

Это издание будет полезно любому разработчику мобильных приложений, как начинающему, так и опытному, уже знакомому с программированием под Android с использованием Android SDK.

Ян Клифтон: «Проектирование пользовательского интерфейса Android» [7].
Создавайте приложения для Android с использованием материального дизайна, который предоставляет привлекательный, функциональный и юзер-ориентированный интерфейс:

- знакомство с основными виджетами пользовательского интерфейса Android;
- максимальное использование эскизов и концептуальных прототипов;
- использование скетчи и концептуальные прототипы для создания оригинальных тем и стилей;
- создание анимационных эффектов для зрелищности и интересности;
- приемы использования улучшенных компонентов.
- использование усовершенствованных компонентов и объединение виджетов для создания эффективных элементов пользовательского интерфейса;
- оптимизация параметров загрузки для повышения популярности вашего приложения.

1.6.2 Анализ аналогичных приложений

В «Google Play» имеется много похожих приложений в категории «Здоровье и фитнес», и мы рассмотрим из них несколько самых популярных и загружаемых приложений.

Приложение «Yclients» – Облачная платформа с мощным функционалом. Помогает клиентам записаться в любое время, а бизнесу — автоматизировать процессы работы с аудиторией. Виджет настраивается под любой бизнес, поддерживает запись на несколько услуг и к нескольким сотрудникам за один раз. Предлагает много отраслевых решений. [8] Скриншот интерфейса приложения показан на рисунке 1.6.

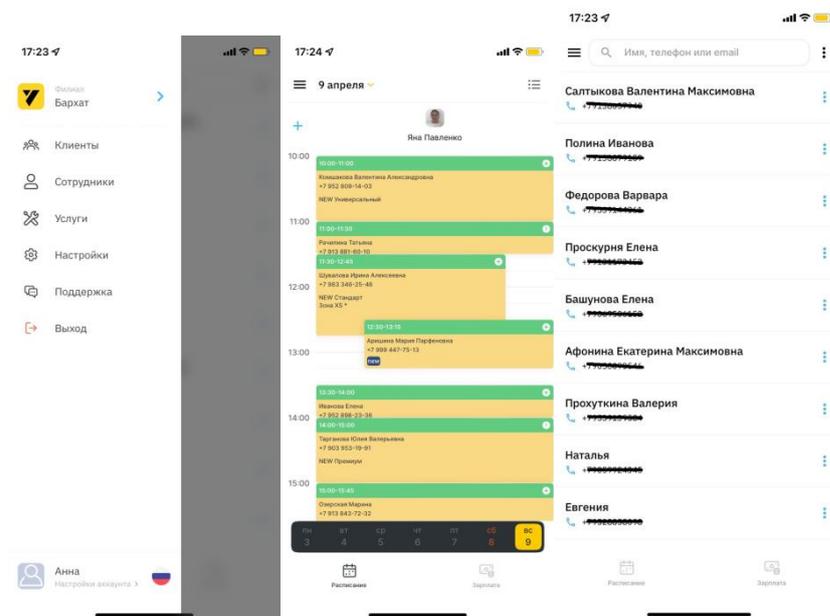


Рисунок 1.6 – Интерфейс приложения «Yclients»

Сама форма регистрации также является быстрым, при регистрации имеется капча – для того, чтобы подтвердить свой номер телефона. Эта же капча работает и для авторизации пользователя в приложении.

Система показывает расписание каждого мастера с данными про клиентов и услугу. Каждый мастер может корректировать график, оставлять комментарии, связываться с клиентами. Похоже на полноценную CRM-систему.

У приложения весьма хорошие отзывы от пользователей, но некоторые из них отмечают сложность интерфейса и необходимость траты времени на понимание работы с приложением. В интернете существуют отдельные видео-уроки объясняющие как пользоваться приложением. В связи с выше сказанным были выделены:

Преимущества:

- богатый функционал приложения;
- хорошая техподдержка;
- много готовых интеграций;
- глубокая статистика и аналитика;
- возможность подключения кассы.

Недостатки:

- сложный интерфейс;
- высокая абонентская плата;
- отсутствует бесплатная версия;
- громоздкая и сложная система.

Приложение «Masters» – это своеобразный клиент для клиентов-организаций в котором происходит регистрация специалиста, работающего на себя, либо студии красоты. Весьма удобный интерфейс и имеется навигационная панель между экранами (фрагментами) приложения. [9] Скриншот интерфейса приложения показан на рисунке 1.7.

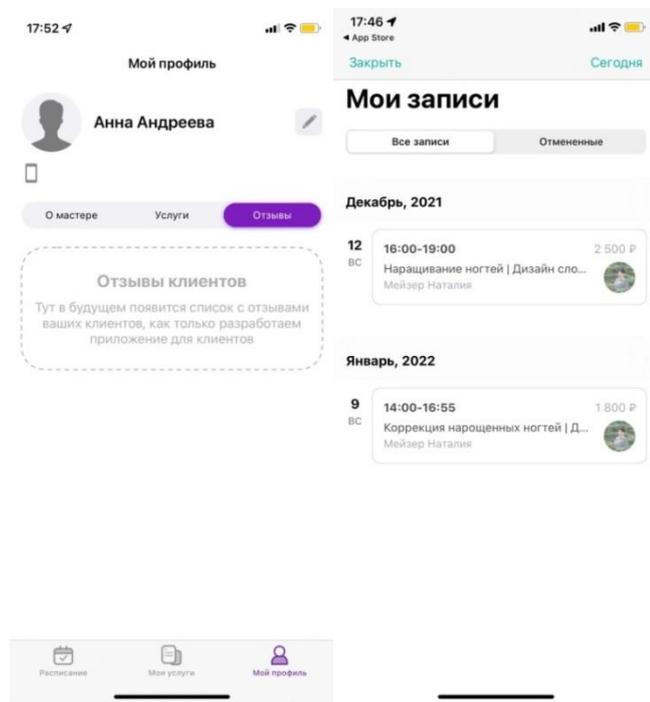


Рисунок 1.7 – Интерфейс приложения «Masters»

Приложение абсолютно бесплатное и имеет ряд задач:

- осуществить on-line запись;
- возможность посмотреть предстоящие визиты (так же имеется возможность отказаться от визита, посмотреть историю посещений и пр.);
- ознакомиться со специалистами, ценами на услуги.

Преимущества:

- простой интерфейс.

Недостатки:

- недоработанное приложение;
- отсутствуют push-уведомления напоминания о записи.

1.7 Вывод по первой главе

После анализа предметной области, были выявлены преимущества мобильного приложения в категории «Здоровье и фитнес». Принято решение после анализа многих похожих приложений (проектов) и популярных сред для разработки – использовать среду программирования «Android Studio» от компании Google, а также язык

программирования «Kotlin».

Проанализировав некоторые источники и аналогичные приложения, был сделан следующий вывод, что эти приложения не имеют понятного интерфейса, обширной информации и отдельной информации о специалистах студии, сами приложения не оптимизированы и не являются быстрыми и также приложения не являются децентрализованными. Тем самым имеют множество серверов, что очень плохо сказывается на производительности самого приложения.

Мобильное приложение улучшает восприятие информации с мобильного устройства, а также заметно сокращает время доступа к информации, имеет удобный интерфейс, раскрывая при этом актуальность создания мобильного приложения.

2 Проектирование мобильного приложения

2.1 Описание технического задания

Мною составлено техническое задание на разработку мобильного приложения для OS Android, состоящее из следующих пунктов:

1. Уровень доступа. Приложение должно поддерживать уровни доступа: Администратор, Специалист и Пользователь. Уровень доступа присваивается при регистрации нового пользователя в базу данных.

– Администратор – Добавляет новых специалистов. Также может редактировать публичную информацию или просматривать все записи к специалистам;

– Специалист – Принимает заявки от клиентов, а также управляет ими (например: отклонить или принять), если специалист не может принять запись от клиента и отклоняет его, то он должен указать причину;

– Пользователь – Может записываться на услуги.

2. База данных. Приложение должно уметь работать с удаленной базой данных SQL, в которой должны быть таблицы: Пользователи, Специалисты (ФИО, биография, отзывы) и таблица с записями на услуги. Описание таблиц:

– Таблица «Пользователи» -- таблица нужна для регистрации / авторизации пользователя в приложении;

– Таблица «Специалисты» -- эта таблица публичная и доступна также и не для авторизованных пользователей, служит для того, чтобы пользователь мог знать информацию о специалистах (ФИО, биография и отзывы);

– Таблица «Записи» -- нужна для учета записей на услуги к специалистам. (Доступна для пользователей с уровнем «Администратор или Специалист»).

3. Регистрация. Приложение должно иметь регистрацию пользователей (для учета их), а также под доступом «Администратор» была возможность регистрировать специалиста.

4. Авторизация. Приложение должно иметь систему авторизации по номеру телефона и паролю пользователя.

5. Навигация. Приложение должно иметь 3 главных экрана: Запись, Специалисты и Профиль.

– На главном экране «Запись» должны присутствовать элементы «Календарь» для выбора даты записи и отображение активной записи на услуги, если ранее была запись

(уровень доступа Пользователь).

– На главном экране «Специалисты» должен присутствовать список всех специалистов и при нажатии на каждый пункт – должен открываться экран с биографией специалиста и отзывами. Этот список публичный и должен отображаться всегда (даже если пользователь не авторизовался в приложении, а также без уровней доступа). При уровне доступа «Администратор», должна иметься кнопка «Добавить специалиста». При нажатии на эту кнопку, должен открыться экран с полями «Имя, Фамилия, Отчество, Номер телефона, Биография и Пароль» и кнопкой «Зарегистрировать специалиста». При регистрации специалиста его уровень должен быть «Специалист».

– На главном экране «Профиль» должны быть пункты «Имя пользователя и его фото» (при нажатии открывается экран с карточкой пользователя), а также пункты «Настройки» и «О нас».

6. **Язык.** Приложение должно поддерживать два языка: Русский и Английский.

7. **Язык программирования.** Приложение должен быть написано на языке – Kotlin.

2.2 Среда верстки мобильного приложения

Для разработки приложения была выбрана среда верстки – Android Studio [10]. Он состоит из следующих компонентов: Android SDK, Android NDK, компоненты графического дизайна, приложение для загрузки компонентов всех версий Android, эмулятор мобильного устройства для запуска приложения, инструменты для тестов и отладки работы приложения.

После того как пользователь запустит среду верстки «Android Studio», то перед ним появится экран со списком активных проектов (если имеются), а также он может создать свой новый проект. И при нажатии на кнопку «создать новый проект», перед ним появится экран с выбором первого макета для проекта. Экран с выбором первого макета показан на рисунке 2.1.

Как только пользователь выберет макет, то перед ним произойдет инициализация среды и подготовки проекта для разработки. После завершения инициализации среды, то пользователь может начинать работу с кодом и в целом с проектом.

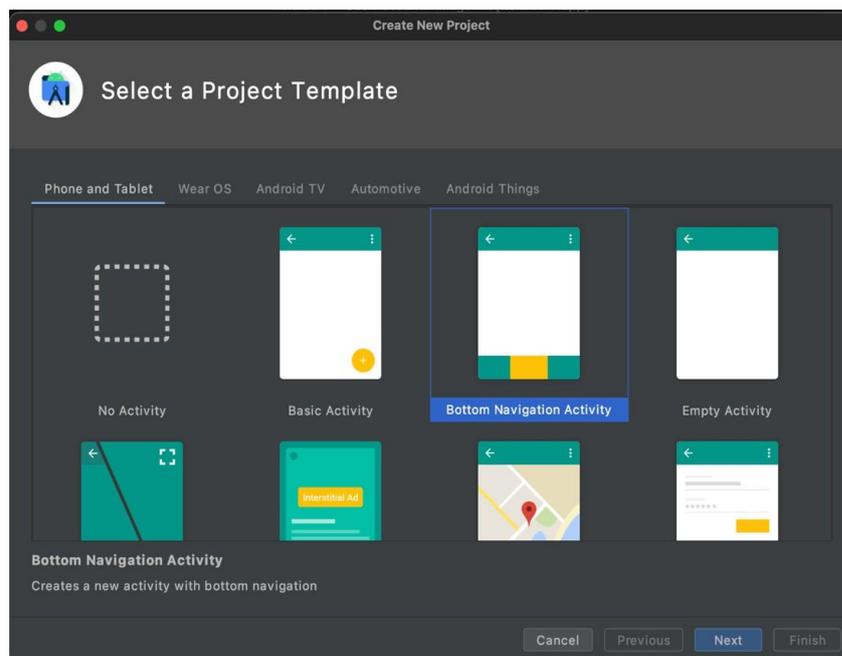


Рисунок 2.1 – Экран для выбора первого макета для создания проекта.

Важным моментом при выборе Android Studio стало то, что она имеет возможность разрабатывать программы практически для всех версий операционной системы Android. Существует инструмент для оценки внешнего вида программы для различных устройств. Многоцветный код упрощает навигацию в больших объемах кода. Программа включает в себя Google Cloud Messaging, целью которой является настройка отправки уведомлений для приложений, при котором задействуются облачные сервисы под управлением операционной системы Android. Таким образом, можно создавать многозадачные программы. Одним из преимуществ данной среды является детальное тестирование приложения.

Данное программное обеспечение призвано предоставить пользователям возможность для коммуникации, просмотра ленты актуальных новостей, а также доступа к Google-картам. Разрабатываемое приложение преимущественно носит справочный характер, в связи с этим доступ к основным функциям приложения возможен без аутентификации пользователей.

2.3 Схемы расположения элементов мобильного приложения

При разработке мобильного приложения продумана общая (без ролей) схема расположения функций и показана на рисунке 2.2. Разрабатывается возможный путь пользователей по экранам приложения. Навигация должна для пользователя быть удобной и понятной.

На рисунках 2.3, 2.4 и 2.5 показаны структурные схемы мобильного приложения для пользователей в ролях «Администратор», «Специалист» и «Пользователь».

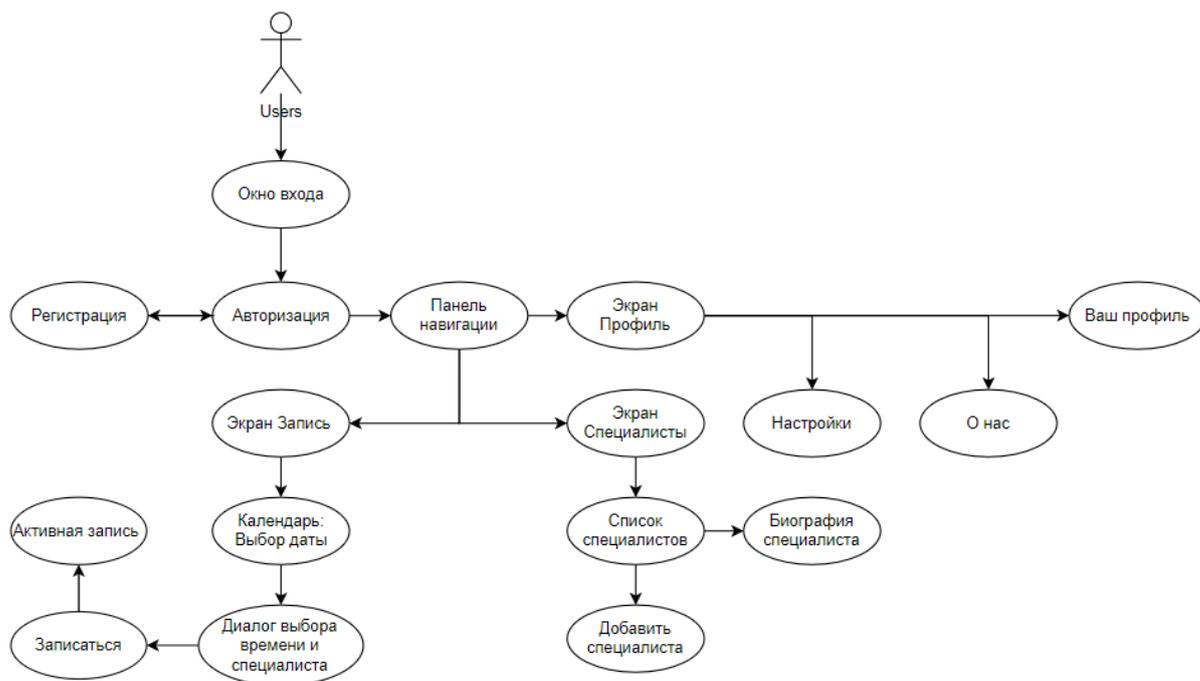


Рисунок 2.2 – Общая структурная схема мобильного приложения.

Приложение имеет панель навигации:

- «Запись» — это экран для записи клиента на услуги, также в этом экране имеется календарь и пункт с активной записью.
- «Специалисты» — это экран, в котором имеется список всех специалистов, при нажатии на нужного специалиста можно прочесть его биографию. Также на этом экране имеется кнопка «Добавить специалиста», она доступна только для Администратора, кнопка выполняет роль регистрации нового специалиста.
- «Профиль» — это экран профиля приложения, в котором имеются пункты: «Ваш профиль», «Настройки» и «О нас». В профиле имеется система проверки авторизации, если пользователь ранее не был авторизован, то экран предложит ему пройти авторизацию, либо регистрацию.

Рассмотрим основной функционал приложения:

1. Календарь – для выбора даты, а также времени, услуги и специалиста;
2. Активная запись на услуги – появляется после записи на процедуру, для уведомления;
3. Список всех специалистов и просмотр его биографии;
4. Добавление нового специалиста (для Администратора);
5. Авторизация пользователя;

6. Регистрация нового пользователя;
7. Профиль пользователя;
8. Настройки приложения;
9. Профиль организации (О нас);
10. Система ролей:

10.1. Пользователь – При регистрации нового клиента ему присваивается эта роль. Клиенту доступна запись на услуги и просмотр биографии специалиста, чтобы клиент мог выбирать специалиста;

10.2. Специалист – При добавлении нового специалиста, ему присваивается эта роль. Специалисту доступно управление записями от клиентов к нему на услуги.

10.3. Администратор – Эта роль организации, которая может добавлять нового специалиста в систему. Полное управление информацией в приложении (редактирование пункта о нас, изменение настроек или управление записями к специалистам).

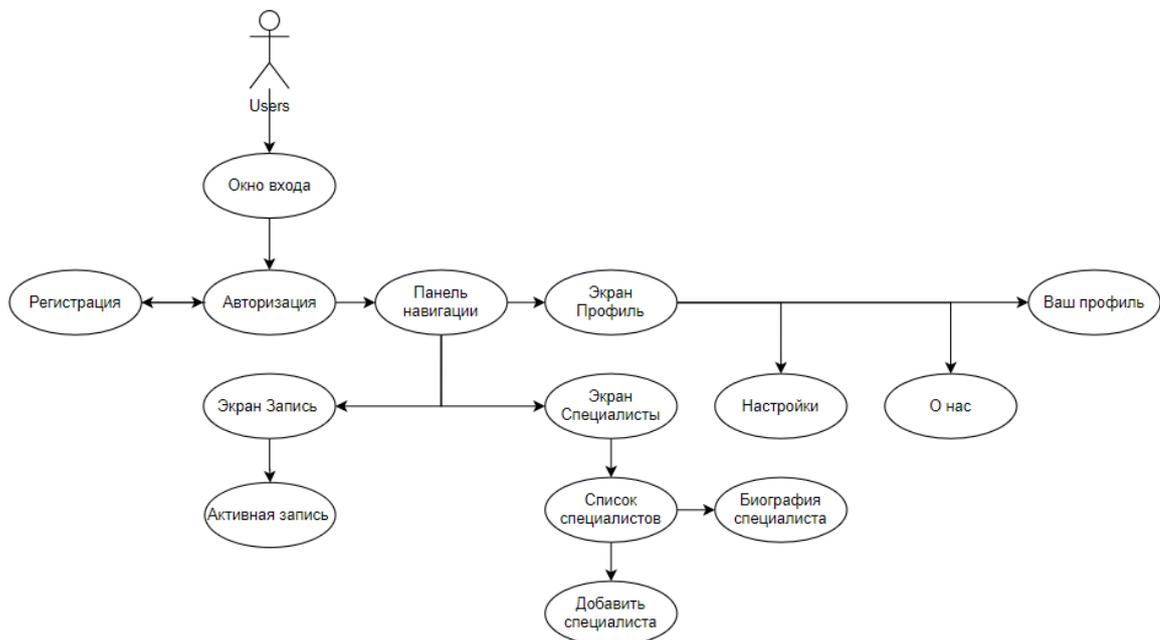


Рисунок 2.3 – Структурная схема мобильного приложения для Администратора

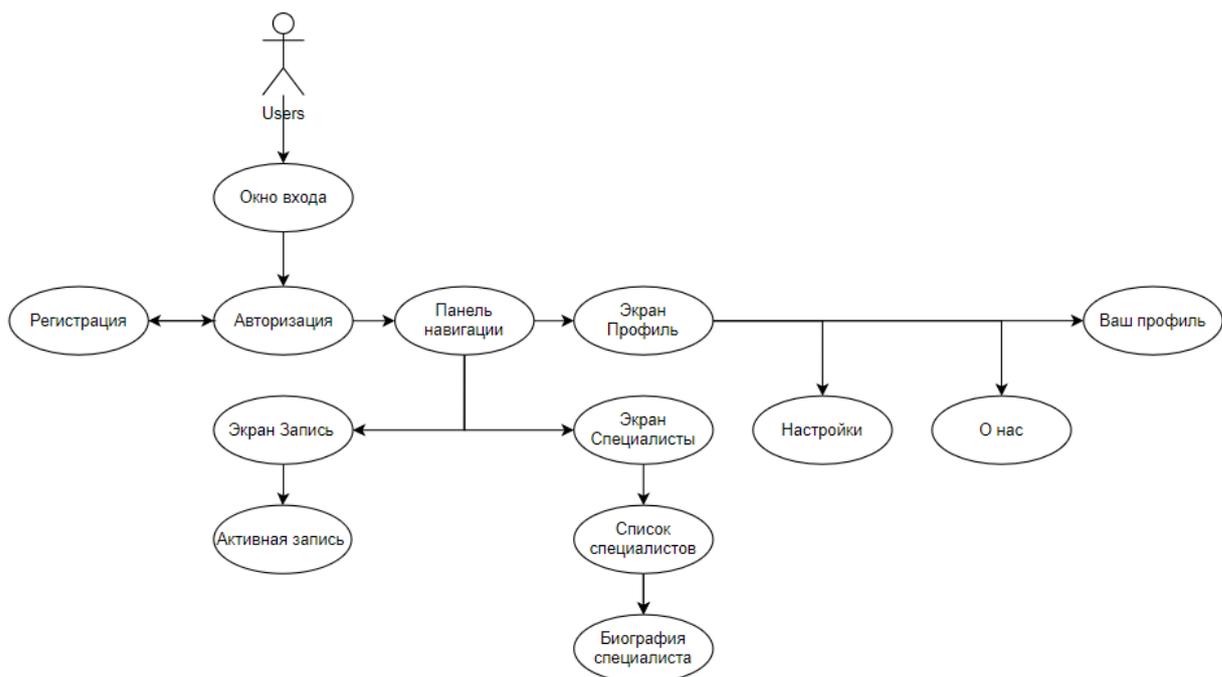


Рисунок 2.4 – Структурная схема мобильного приложения для Специалиста

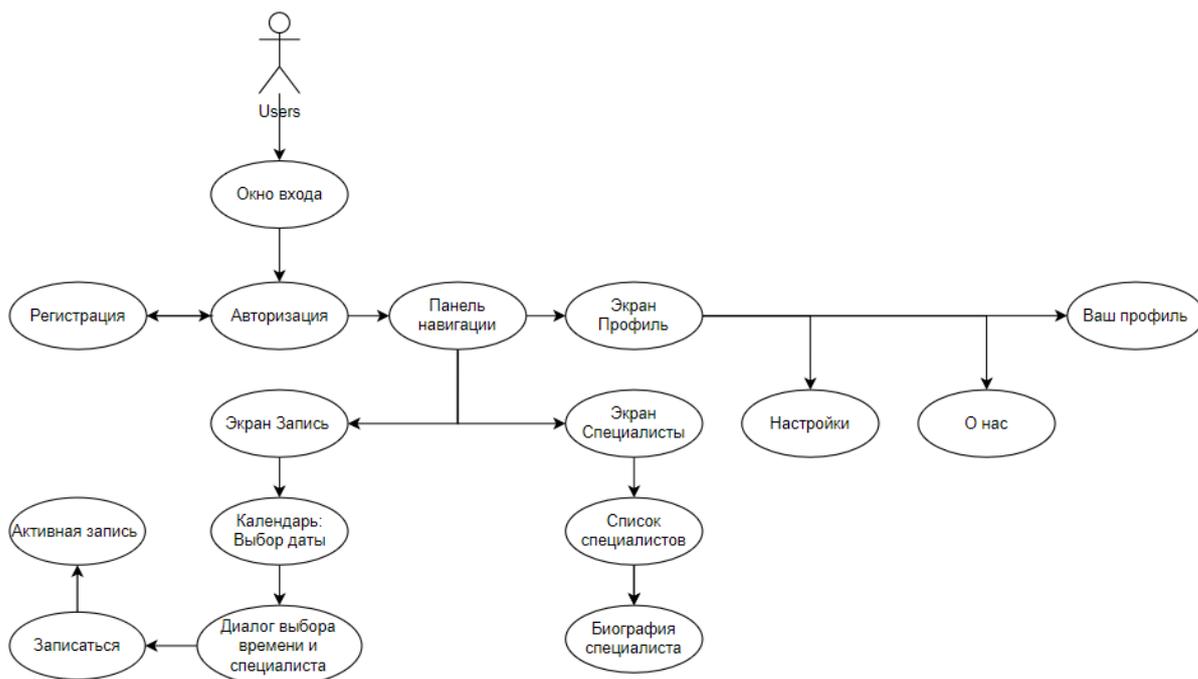


Рисунок 2.5 – Структурная схема мобильного приложения для Пользователя

На рисунке 2.6 спроектирована общая диаграмма вариантов использования на основе сформированных требований к приложению. Для удобства некоторые варианты использования были объединены, в каждой из которых будут раскрыты подробности объединённых вариантов использования.

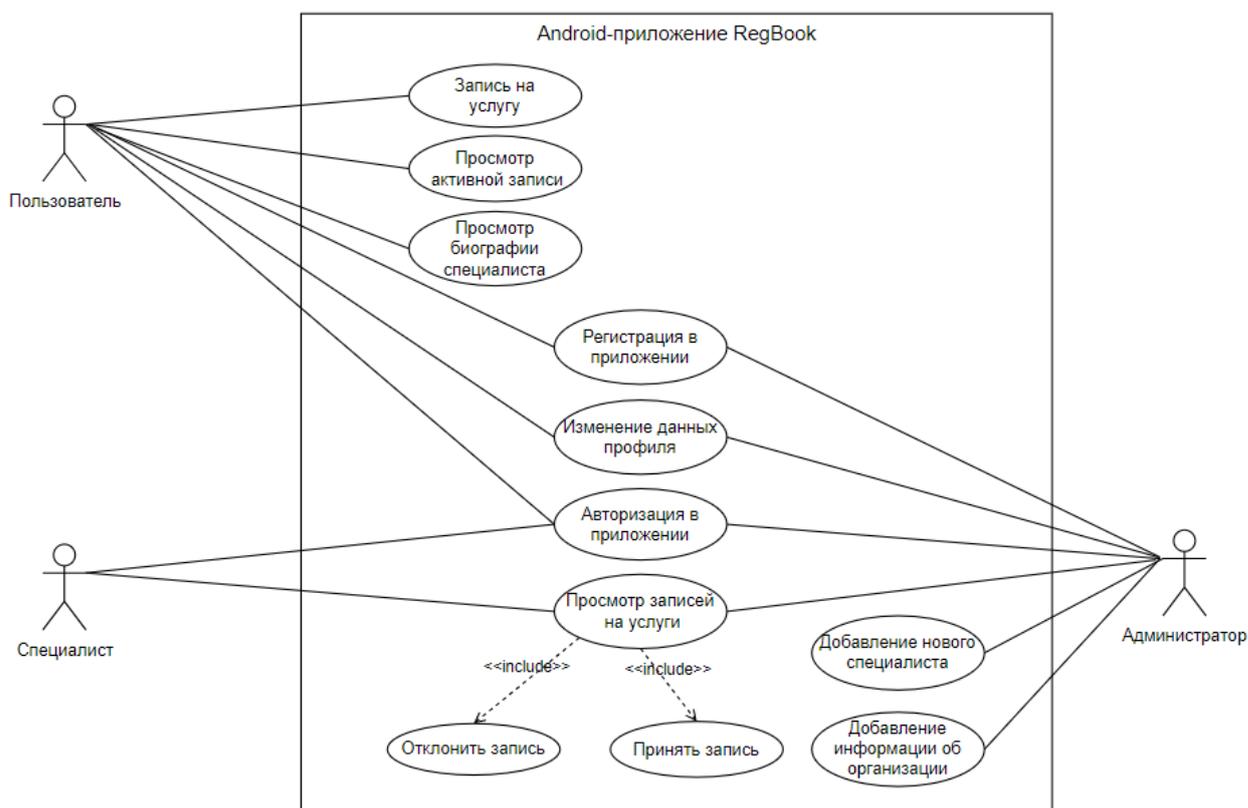


Рисунок 2.6 – Общая диаграмма вариантов использования

Детализация изображенных вариантов использования на диаграмме показана на рисунках 2.7, 2.8, 2.9, 2.10.

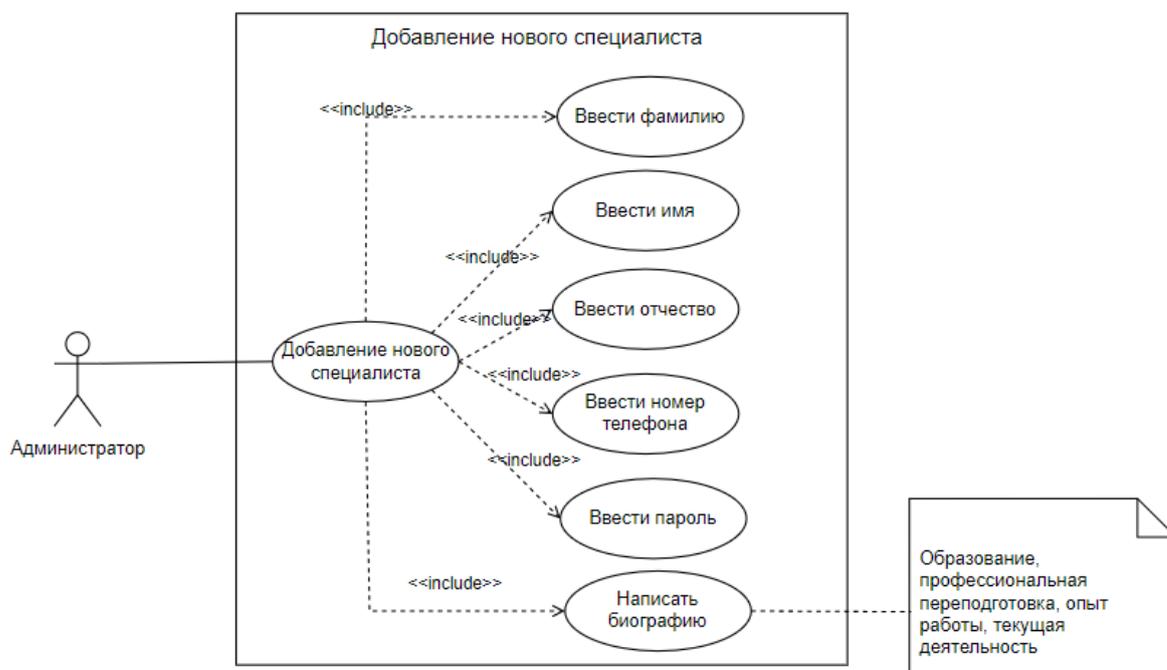


Рисунок 2.7 – Вариант использования «Добавление нового специалиста»

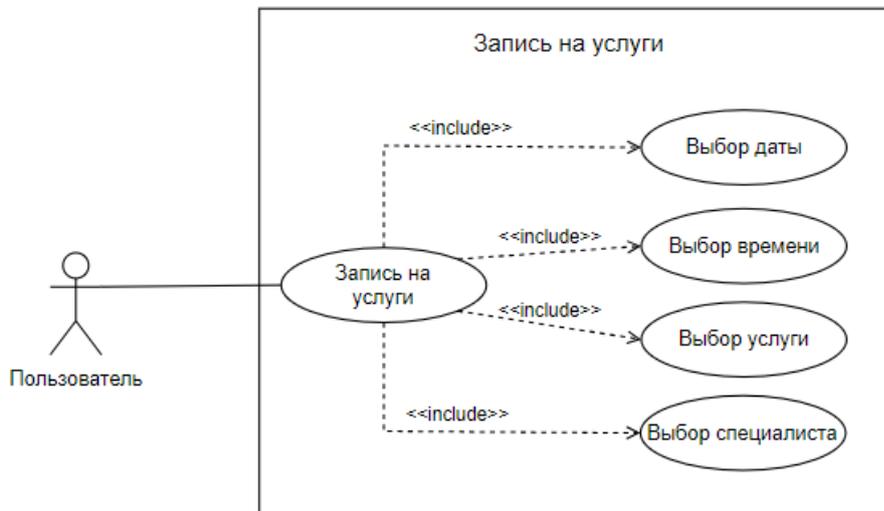


Рисунок 2.8 – Вариант использования «Запись на услуги»

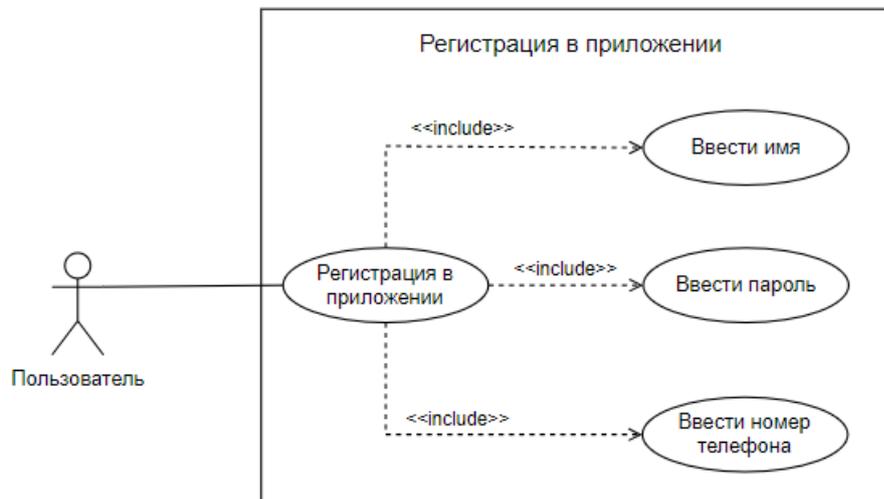


Рисунок 2.9 – Вариант использования «Регистрация в приложении»

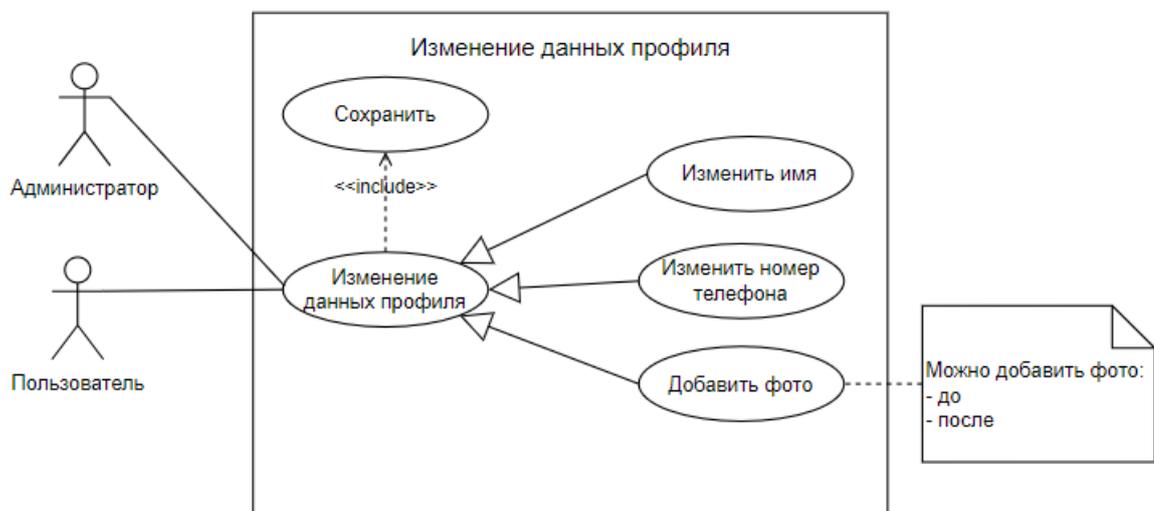


Рисунок 2.10 – Вариант использования «Изменение данных профиля»

2.4 Разработка фреймворка для работы с базой данных

Для разработки фреймворка была выбрана удаленная база данных SQL. Под удаленную базу данных использовался хостинг, созданный через сервис **freesqldatabase.com**. Этот сервис создает базу данных и дает учетные данные (такие как Сервер, Имя пользователя, Порт и Пароль) для администрирования созданной базы данных. Администрирование базы данных проходит через сервис **phpMyAdmin**.

На рисунке 2.11 показана форма для авторизации на сервисе phpMyAdmin. На этой форме требуется указать имя сервера, имя пользователя и пароль.

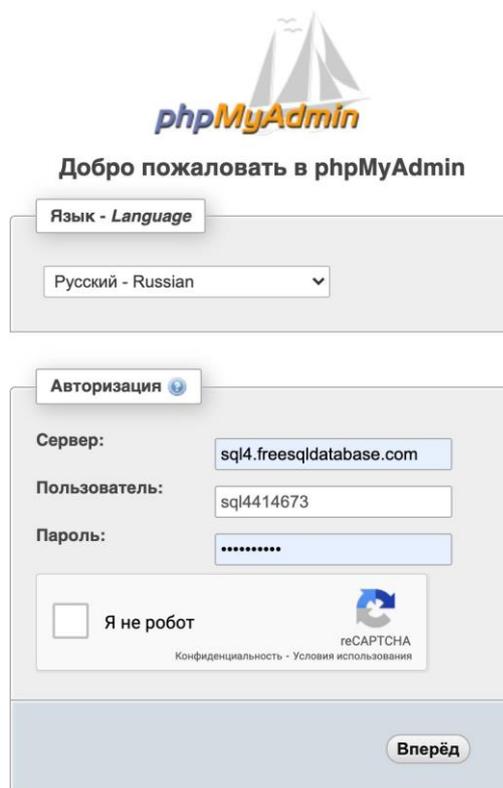


Рисунок 2.11 – Вход phpMyAdmin для администрирования удаленной базы данных

phpMyAdmin [11] – это веб-приложение с открытым кодом, написанное на языке PHP и представляющее собой веб-интерфейс для администрирования СУБД MySQL. PhpMyAdmin позволяет через браузер и не только осуществлять администрирование сервера MySQL, запускать команды SQL и просматривать содержимое таблиц и баз данных. Приложение пользуется большой популярностью у веб-разработчиков, так как позволяет управлять СУБД MySQL без непосредственного ввода SQL команд.

Фреймворк [12] – это программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Текущий фреймворк предназначен для работы сервером SQL, включает следующие методы и функции:

Метод «инициализация (подключение)» к серверу SQL – этот метод предназначен для инициализации самого фреймворка и также подключение к серверу SQL. Инициализировать нужно один раз в активности приложения (MainActivity). На рисунке 2.12 показан код этого метода.

```
try {
    connection = Connection(
        hostname: "sql4.freemsqldatabase.com",
        username: "sql4414673",
        password: "DN3VIXEHWT", port: 3306,
        database: "sql4414673", callback)
} catch (e : Exception) {
    e.printStackTrace()
}
```

Рисунок 2.12 – Метод инициализации (подключения)

Метод «де-инициализация (разрывание)» к серверу SQL – этот метод служит для разрыва соединения к серверу SQL, а также для прекращения работы самого фреймворка. Вызывать рекомендуется этот метод один раз при завершении работы приложения (в MainActivity). На рисунке 2.13 показан код этого метода.

```
// Разрываем подключение к серверу
fun closeConnection()
{
    connection?.close()
    connection = null
}
```

Рисунок 2.13 – Метод де-инициализации

Метод «переподключение» к серверу SQL – этот метод позволяет совершать переподключение к серверу, если произошел разрыв в соединении к серверу. Рекомендуется вызывать метод при работе с базой данных, а также если приложение перешло в спящий режим и при выходе из него следует выполнить переподключение к серверу. На рисунке 2.14 показан код этого метода.

```
// Переподключение к серверу
fun reconnect()
{
    if (connection == null)
        connect( listener: null)
}
```

Рисунок 2.14 – Метод переподключение

Метод «создание таблицы» в базе данных – этот метод позволяет создать новую таблицу в текущей базе данных. Рекомендуется использовать код SQL базы данных. На рисунке 2.15 показан код этого метода.

```

// Можно создать новую таблицу
fun createTable(table: String)
{
    val callback = object : IConnectionInterface
    {...}

    connection?.let { it: Connection
        val statement = it.createStatement()
        statement.execute(table, callback)
    }
}

```

Рисунок 2.15 – Метод создание таблицы

Информацию о пользователях, специалистах и заявок на услуги необходимо где-то хранить. В параграфе 2.4 был выбран способ — с помощью удаленной базы данных, в которой были созданы 3 таблицы с текущим набором команд для базы данных:

1. Таблица «users_table» - таблица (таб. 2.1) учета всех пользователей;
 - CREATE TABLE users_table(Username TEXT,Password CHAR(30),Phone CHAR(30),Role CHAR(20))
2. Таблица «specs_table» - таблица (таб. 2.2) всех специалистов;
 - CREATE TABLE specs_table(LastName TEXT,FirstName TEXT,MiddleName TEXT,Biography TEXT,Phone CHAR(30))
3. Таблица «records_table» - таблица (таб. 2.3) всех заявок к специалистам.
 - CREATE TABLE records_table(RecId TEXT,FromUser CHAR(30),ToUser CHAR(30),TimeStamp TEXT)

Таблица 2.1 – Таблица «Пользователи»

Сущность	Идентификатор таблицы	Атрибут	Тип поля	Кол-во символов
Пользователи	users_table	Username	Text	-
		Password	Char	30
		Phone	Char	30
		Role	Char	20

Таблица 2.2 – Таблица «Специалисты»

Сущность	Идентификатор таблицы	Атрибут	Тип поля	Кол-во символов
Специалисты	specs_table	LastName	Text	-
		FirstName	Text	-
		MiddleName	Text	-
		Biography	Text	-
		Phone	Char	30

Таблица 2.3 – Таблица «Заявки»

Сущность	Идентификатор таблицы	Атрибут	Тип поля	Кол-во символов
Заявки	records_table	RecId	Text	-
		FromUser	Char	30
		ToUser	Char	30
		TimeStamp	Text	-

Метод для создания ячейки нового пользователя в таблице для учета пользователей – этот метод позволяет записывать данные в таблицу «Пользователи». Использовался код для SQL:

```
INSERT INTO users_table (Username, Password, Phone, Role) VALUES ('$userName', '$password', '$phone', '$role')
```

Метод для создания ячейки нового специалиста – этот метод позволяет записывать данные в таблицу «Специалисты». Использовался код для SQL:

```
INSERT INTO specs_table (LastName, FirstName, MiddleName, Biography, Phone) VALUES ('$lastName', '$firstName', '$middleName', '$biography', '$phone')
```

Метод «проверка пользователя» в базе данных – этот метод позволяет проверить пользователя в базе данных по номеру телефона. Использовался код для SQL:

```
SELECT * FROM users_table WHERE Phone='$phone'
```

Метод «проверка пользователя и пароля» в базе данных – этот метод выполняет роль авторизации путем проверки двух значений от двух атрибута «Phone» и «Password». Использовался код для SQL:

```
SELECT * FROM users_table WHERE Phone='$phone', Password='$password'
```

Метод «получение роли» пользователя – этот метод получает роль пользователя, для построения пользовательского интерфейса. Роли: Администратор, Специалист и Пользователь. Использовался код для SQL:

```
SELECT * FROM users_table WHERE Phone='$phone'
```

Метод «получение списка специалистов» из базы данных – этот метод формирует список «List» всех специалистов. Таблица «Специалисты» является публичным и соответственно список является публичным. Использовался код для SQL:

```
SELECT * FROM specs_table
```

Метод «создание заявки» в базе данных – этот метод создает новую ячейку в таблице «Заявки». Использовался код для SQL:

```
INSERT INTO records_table (RecId, FromUser, ToUser, TimeStamp) VALUES ('$id', '$fromUser', '$toUser', '$timestamp')
```

Метод «получение всех заявок» из базы данных – этот метод формирует список всех заявок. В следствии нужен для отображения активных заявок от пользователя. Использовался код для SQL:

```
SELECT * FROM records_table
```

Для управлением фреймворком был разработан класс под названием «UserAccountManager». В этом классе были разработаны методы для регистрации и авторизации пользователя, а также получение и сохранение роли, номера телефона и имя пользователя.

Метод «startRegistration» – в этом методе происходит проверка пользователя в базе данных, если пользователя нет, происходит создание пользовательских данных в базе данных и присваивается роль «Пользователь». Если все прошло успешно, происходит сохранения данных: Номер телефона, пароль, имя пользователя и роль.

На рисунке 2.16 показана блок-схема регистрации пользователя.

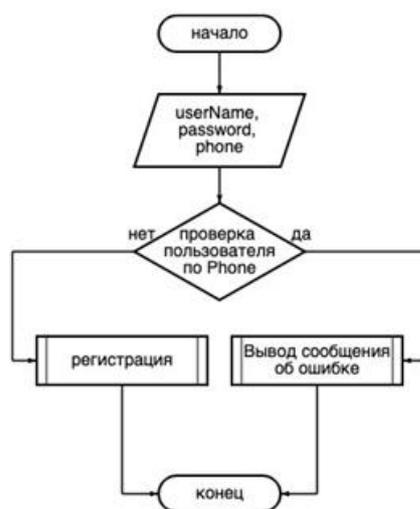


Рисунок 2.16 – Схема регистрации пользователя

Метод «startLogin» – это метод авторизации, вводятся данные: Номер телефона и Пароль, после чего происходит проверка этих данных в базе данных. Если данные не совпадают, но пользователь присутствует, значит ошибка «Неверный пароль». Если данные совпадают, происходит процесс получения роли пользователя и сохранения данных, таких как: Номер телефона, пароль, имя пользователя и роль. На рисунке 2.17 показана блок-схема авторизации пользователя.



Рисунок 2.17 – Схема авторизации пользователя

Метод «startRegistrationStaffer» – этот метод аналогичный методу «startRegistration», регистрирует специалиста в таблице «Пользователи», а также добавляет публичную информацию такую как: ФИО, Номер телефона и Биографию в таблицу «Специалисты». Роль при регистрации: Специалист. Если все прошло успешно, происходит сохранения данных: Номер телефона, пароль, имя пользователя и роль.

На рисунке 2.18 показана блок-схема регистрации специалиста.

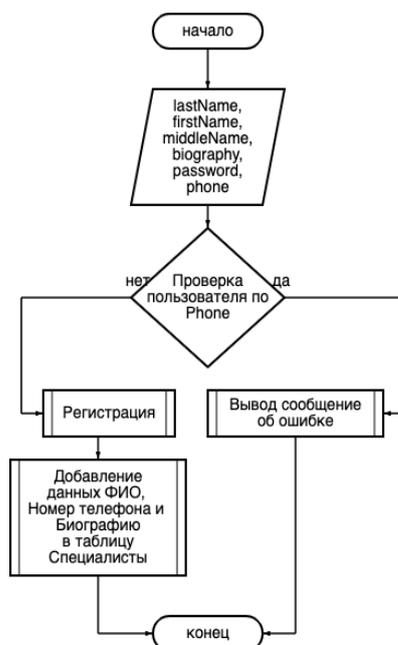


Рисунок 2.18 – Схема регистрации специалиста

Метод «isUserAuth» – этот метод проверяет сохранённые данные пользователя в память устройства (приложения). Если данные имеются, значит пользователь

авторизирован и вызывать метод «autoLogin» не требуется.

Метод «autoLogin» – этот метод автоматически вызывает метод «startLogin» без надобности ввода пароля и номера телефона. Данные берутся из сохранённых данных (памяти устройства).

Метод «getUserPhone» – этот метод получает номер телефона из сохраненных данных (памяти устройства).

Метод «getUserName» – этот метод получает имя пользователя из сохраненных данных (памяти устройства).

Метод «getUserPrivilege» – этот метод получает роль пользователя из сохраненных данных (памяти устройства).

2.5 Вывод по второй главе

В этой главе была проделана работа по разработке общей структуры мобильного приложения, а также был разработан единый фреймворк для работы с базой данных SQL, а также разработан класс для управления фреймворком. В этом классе происходит управление учетной записью пользователя, таких как регистрация, авторизация, добавление нового специалиста и т.д.

Для разработки использовался язык программирования – Kotlin, а сама база данных выбрана в качестве удаленной и для этого был выбран сервис «freesqldatabase.com». А для администрирования базы данных используется сервис «phpMyAdmin».

3 Описание разработанного приложения

3.1 Этапы разработки пользовательского интерфейса

Первоначальным этапом считается исследования целевой аудитории также кейсов продукта. Это дает возможность осознать будущих пользователей приложения, а также сформировать практичный интерфейс, что станет наилучшим для каждого. Мы затрагиваем такие аспекты, как размеры и расположение элементов: button, text, text-style, цвет и стиль оформления.

Вторым этапом идет реализация эскизов – это позволяет нам увидеть интерфейс т.е. его структуру. А «User Flow Diagram» [13] помогает нам сделать иллюстрированную логику проекта.

Третьим этапом мы определяем стиль: от скевоморфизма до минимализма. На этом этапе мы рисуем несколько вариантов стиля и предлагаем клиенту выбор. Большое внимание уделяем современным трендам, масштабированию интерфейса и оцениваем время на разработку дизайна.

Четвертым этапом это демонстрация полной версии дизайна. Сама реализация может быть низкоуровневой, т.к. этот способ показывает саму структуру и описание взаимодействий пользователя с мобильным приложением.

Пятый этап это макет т.е. позволяет продемонстрировать весь дизайн проекта. Контент на этом этапе – статичен и такая форма подачи воспринимается нагляднее предыдущей.

Последним этапом является прототип, который можно проверить и протестировать. Эта детализированная форма финального макета. Полное эмуляция взаимодействий пользователя с интерфейсом. На этом этапе затрачивается большое количество времени.

3.1.1 Описание навигационных пунктов

При первом запуске приложения, пользователь попадает на первый главный экран по умолчанию регистрации или авторизации. Этот экран показан на рисунке 3.1.

При входе пользователь попадает на экран – «Запись». Этот экран показан на рисунке 3.2. На этом экране расположен календарь и список с активными записями (если имеется). Сам контент динамический, то есть зависит от «роли» авторизованного пользователя. Если пользователь авторизуется под учетной записью «Администратор», то на этом экране функция записи на услуги (календарь) не будут видны, останется список со всеми активными записями к специалистам. Если пользователь авторизовался как обычный пользователь, то для него будет доступен календарь для записи на услуги, а

также его список активных записей к специалистам. Если пользователь является специалистом, то для него также как и для администратора, календарь будет скрыт, но список активных заявок будет доступен, т.е. в нем будет отображаться записи от тех пользователей, кто к нему записался на услуги. Код верстки макета XML представлен на рисунке 3.3.

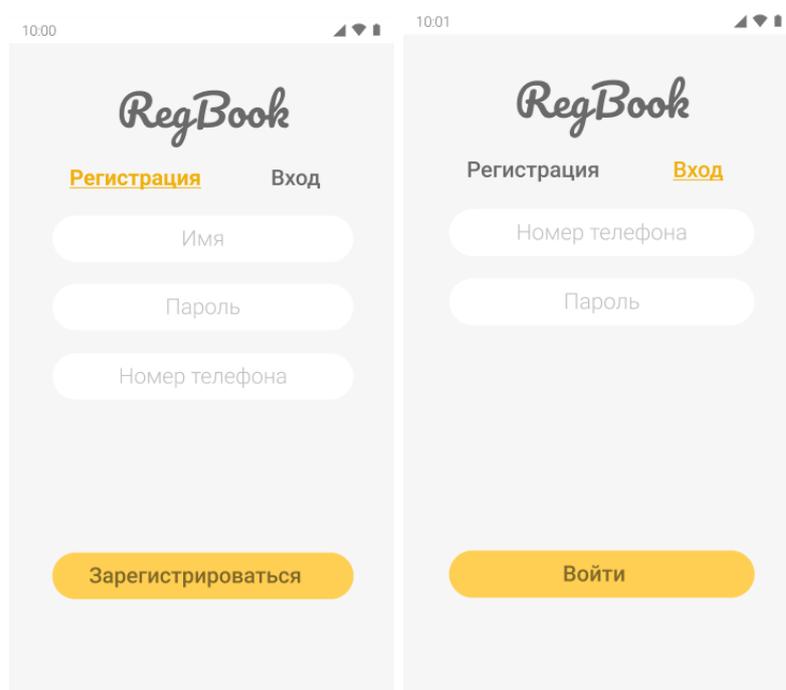


Рисунок 3.1 – Окно регистрации и авторизации



Рисунок 3.2 – Главный экран «Запись»

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <include layout="@layout/toolbar" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <TextView
                android:id="@+id/calendarViewText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="15dp"
                android:layout_marginHorizontal="15dp"
                android:textSize="18sp"
                android:textStyle="bold"
                android:textColor="@?attr/colorTextAndTint"

            <CalendarView
                android:id="@+id/calendarView"
                android:layout_width="match_parent"

```

Рисунок 3.3 – Код макета XML экрана «Запись»

Вторым главным экраном является «Специалисты». Этот экран показан на рисунке 3.4 и является публичным.

Экран отображает список всех специалистов, то есть не авторизованные или авторизованные пользователи могут ознакомиться с биографией специалистов. Также контент здесь является динамическим, то есть если авторизованный пользователь в «Администратор», то на этом экране в нижнем правом углу появится кнопка «Добавить специалиста».

Администратор при нажатии на эту кнопку перейдет на экран регистрации (добавления) нового специалиста. В котором ему необходимо заполнить всю информацию о новом специалисте.

Код верстки макета XML представлен на рисунке 3.5.

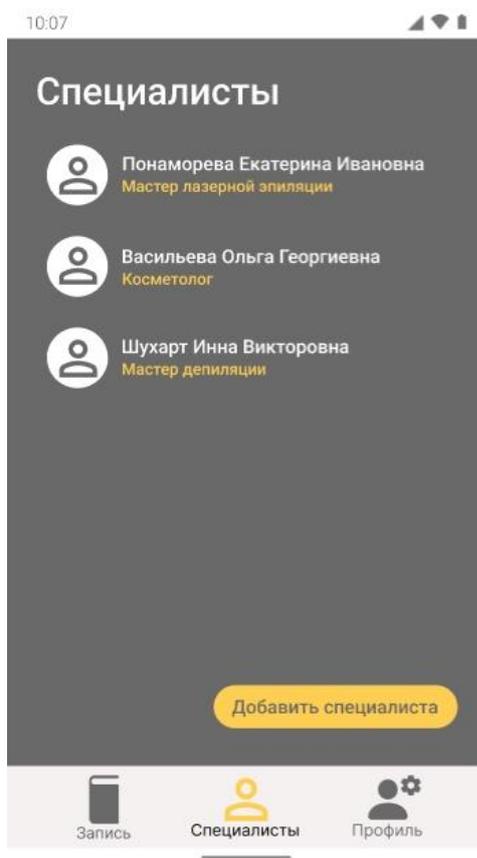


Рисунок 3.4 – Главный экран «Специалисты»

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <include layout="@layout/toolbar" />

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <androidx.recyclerview.widget.RecyclerView
            android:id="@+id/specsRecyclerView"
            android:layout_width="match_parent"
            android:layout_height="match_parent" />

        <TextView
            android:id="@+id/emptyList"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_centerInParent="true"
            android:text="Список пуст!"
            android:visibility="gone" />

        <TextView
            android:id="@+id/addBtn"
            android:background="@drawable/background_button"
            android:layout_width="wrap_content"
            android:layout_height="36dp"
            android:layout_alignParentBottom="true"
    
```

Рисунок 3.5 – Код макета XML экрана «Специалисты»

Третьим главным экраном является «Профиль». Этот экран показан на рисунке 3.6. На этом экране расположены элементы для входа в учетную запись (если не авторизован пользователь), либо переход на экран «Ваш профиль».

А также на этом экране расположены пункты для перехода на экран настроек

приложения и пункт для перехода на экран «О нас».

Код верстки макета XML представлен на рисунке 3.6.

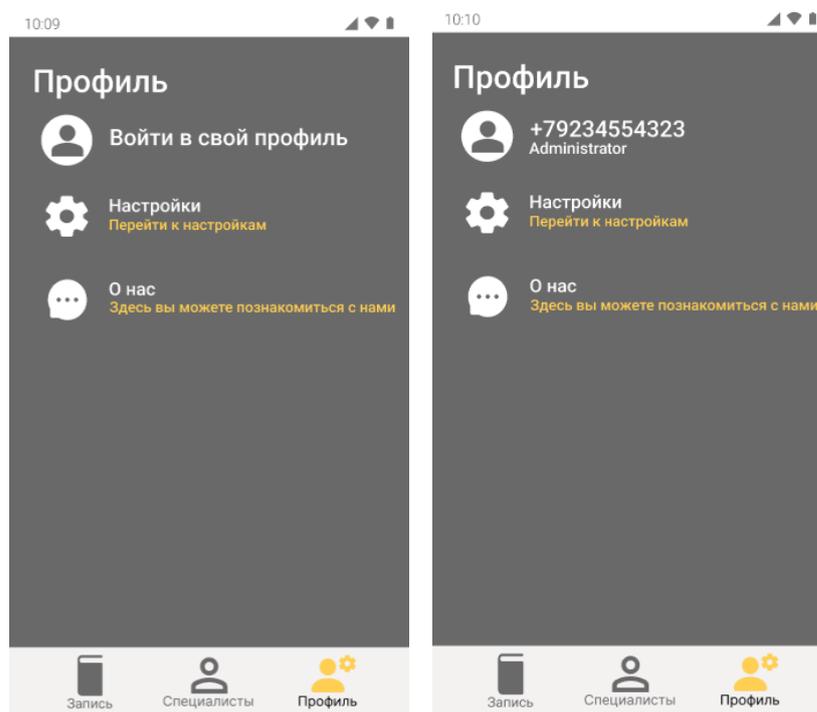


Рисунок 3.6 – Главный экран «Профиль»

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <include layout="@layout/toolbar" />

    <LinearLayout
        android:id="@+id/authContent"
        android:orientation="horizontal"
        android:background="?attr/colorNav"
        android:layout_width="match_parent"
        android:layout_height="70dp"
        android:paddingHorizontal="15dp"
        android:gravity="center_vertical">

        <ImageView
            android:id="@+id/userAvatar"
            android:layout_width="45dp"
            android:layout_height="45dp"
            android:src="@drawable/placeholder"
            app:tint="?attr/colorTextAndTint" />

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <TextView
```

Рисунок 3.7 – Код верстки XML экрана «Профиль»

Когда пользователь авторизован, то на экране «Профиль» появляется его фото, номер телефона и имя. При нажатии на этот пункт, пользователь переходит на экран «Ваш профиль». Этот экран показан на рисунке 3.8.

На этом экране находится информация: Фото пользователя, имя пользователя, номер пользователя и роль пользователя, а также кнопка для выхода из учетной записи.

В будущем на этот экран можно будет добавить различную информацию, например: рейтинг пользователя, просмотр оставленных отзывов.

Код верстки макета XML представлен на рисунке 3.9.

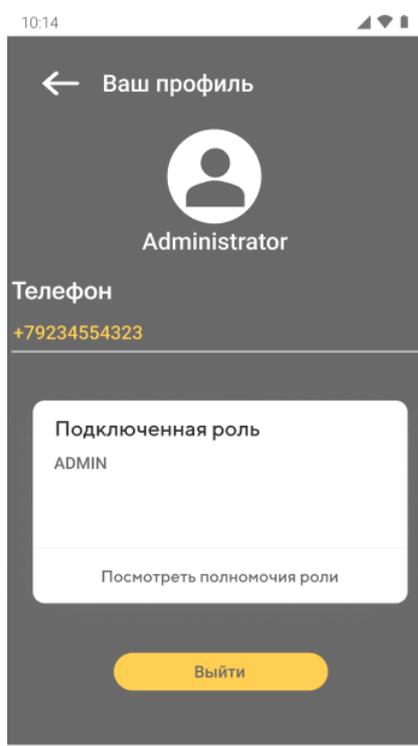


Рисунок 3.8 – Экран «Ваш профиль»

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <include layout="@layout/toolbar" />

    <LinearLayout
        android:orientation="vertical"
        android:background="?attr/colorNav"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingHorizontal="15dp"
        android:gravity="center_horizontal">

        <ImageView
            android:id="@+id/userAvatar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginVertical="15dp"
            android:src="@drawable/placeholder"
            app:tint="?attr/colorTextAndTint" />

        <TextView
            android:id="@+id/userName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginHorizontal="15dp"
            android:textSize="18sp"


```

Рисунок 3.9 – Код верстки XML экрана «Ваш профиль»

Следующий экран – это экран «Биография», этот экран показан на рисунке 3.10. Когда пользователь переходит на главный экран «Специалисты», то ему отображается список всех специалистов.

При нажатии на специалиста из списка, то пользователь попадет на экран с биографией специалиста. Тем самым пользователь может познакомиться со специалистом, его деятельностью и опытом. Код верстки макета XML представлен на рисунке 3.11.



Рисунок 3.10 – Экран «Биография специалиста»

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <include layout="@layout/toolbar" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_margin="15dp">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <TextView
                android:id="@+id/name"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:textColor="?attr/colorTextAndTint"
                android:textSize="18sp"
                android:textStyle="bold" />

            <TextView
                android:id="@+id/biography"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dp"
                android:textColor="?attr/colorTextAndTint"
                />

        </LinearLayout>

    </ScrollView>

</LinearLayout>
```

Рисунок 3.11 – Код верстки XML экрана «Биография специалиста»

Также на главном экране «Специалисты» имеется кнопка «Добавить специалиста», как говорилось ранее, эта кнопка доступна только для администраторов.

Эта кнопка позволяет перейти на экран «Добавить специалиста» и зарегистрировать нового специалиста, заполнив все пункты, такие как: Фамилия, Имя, Отчество, Номер телефона, Пароль и Биография.

Все эти пункты являются обязательными, если пользователь оставить хоть одну строку пустой, то кнопка для регистрации будет недоступна и принудит пользователя заполнить все пункты.

На рисунке 3.12 показан скриншот экрана «Добавить специалиста». Код верстки макета XML представлен на рисунке 3.13.

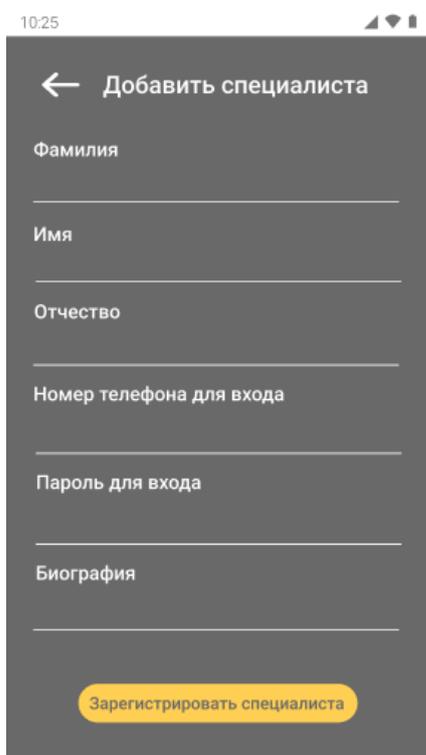


Рисунок 3.12 – Экран «Добавить специалиста»

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@attr/colorPrimary">

    <include layout="@layout/toolbar" />

    <ScrollView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginBottom="15dp">

        <LinearLayout
            android:orientation="vertical"
            android:layout_width="match_parent"
            android:layout_height="wrap_content">

            <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="15dp"
                android:layout_marginHorizontal="15dp"
                android:text="@string/Фамилия"
                android:textSize="14sp"
                android:textStyle="bold"
                android:textColor="@attr/colorTextAndTint" />

            <EditText
                android:id="@+id/lastName"

```

Рисунок 3.13 – Код верстки XML экрана «Добавить специалиста»

Как описывалось ранее на главном экране «Запись» имеется календарь, при нажатии на дату в календаре происходит показ диалогового окна, как показано на рисунке 3.14, в нем пользователь может выбрать время в позиции от 10:00 до 20:00, а также специалиста и услугу, после чего нажать на кнопку «Записаться». Как только пользователь записался на процедуру, то на этом же экране появится активная запись.

Пользователь также может управлять своими активными записями, то есть совершать отмену (удаление) записи.

Код верстки макета XML представлен на рисунке 3.15.

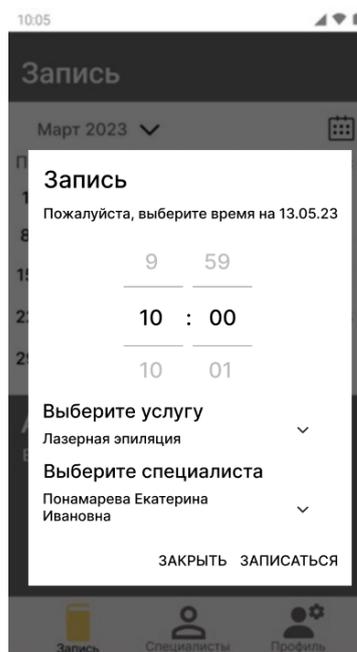


Рисунок 3.14 – Диалоговое окно записи на услуги

Администраторы и специалисты, могут управлять активными записями. Для администратора список с активными записями доступен от всех пользователей ко всем специалистам и при возможности администратор может совершать отмену записи, но при этом он должен указать причину отмены. Клиент об этом должен знать. Также специалист может управлять своими активными записями от клиентов, также может совершать отмену и также должен указывать причину отмены.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal">

    <TimePicker
        android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:timePickerMode="spinner"/>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginHorizontal="25dp"
        android:text="Выберите специалиста"
        android:textSize="16sp"
        android:textColor="?attr/colorTextAndTint" />

    <Spinner
        android:id="@+id/specSpinner"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginHorizontal="9dp" />

</LinearLayout>
```

Рисунок 3.15 – Код верстки XML диалогового окна «Запись на процедуру»

3.2 Вывод по третьей главе

В этой главе был разработан пользовательский интерфейс на основе разработанного фреймворка из проектной модели и выявленных требований к приложению. А также описан каждый экран, что и за что отвечает, спроектированы диаграммы вариантов использования и описана общая структурная схема функционала приложения. Была реализована система записей на проведение услуг, просмотра биографии специалиста и соответственно выбора специалиста и услуги при записи.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы было разработано мобильное приложение для платформы OS Android, позволяющее будущим клиентам (пользователям) записываться на услуги, знакомиться со специалистом узнавая его биографию и заполнять информацией свой профиль.

Были проанализированы технологии для написания мобильного приложения, ссылаясь на техническое задание, выбрана платформа для работы приложения и среда разработки.

В ходе выполнения выпускной квалификационной работы были решены следующие задачи:

1. проанализирована предметная область;
2. проведен обзор существующих аналогов;
3. сформировано техническое задание;
4. выбраны инструменты разработки.
5. спроектирована архитектура мобильного приложения;
6. разработано приложение.

Результатом работы является создание мобильного приложения для учета клиентов.

Проект соответствует всем требованиям технического задания и готов к использованию. Таким образом, задачи выпускной квалификационной работы полностью решены и цель исследования достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хиллегасс А. Objective-C. Программирование для Android [Текст] / Хиллегасс А. – Питер, 2012 г. – 103 с.
2. Статья «Типы мобильных приложений» [Электронный ресурс]. – Режим доступа: <https://punicapp.com/blog/pages/1046/typy-mobilnyh-prilozhenij>. (Дата обращения 02.10.2022)
3. Статья «Kotlin» [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Kotlin> (Дата обращения 01.03.2023)
4. Статья «Этапы разработки мобильного приложения» [Электронный ресурс]. – Режим доступа: <https://wellsoft.pro/blog/etapi> (Дата обращения 03.03.2023)
5. Майер Р. Программирование приложений для планшетных компьютеров и смартфонов [Текст] / Р. Майер – Москва: Эксмо, 2013. – 816 с.
6. Ретабоуил С. Android NDK. Руководство для начинающих [Текст] / Ретабоуил С. – ДМК-Пресс, 2016 г. – 325 с.
7. Клифтон Я. Проектирование пользовательского интерфейса Android [Текст] / Клифтон Я. – ДМК-Пресс, 2017 г. – 202 с.
8. Приложение в магазине «Google Play» – Yclients [Электронный ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=apps.yclients1&hl=ru&gl=US> (Дата обращения 04.04.2023)
9. Приложение в магазине «Google Play» – Masters [Электронный ресурс]. – Режим доступа: <https://play.google.com/store/apps/details?id=ru.dikidi&hl=ru&gl=US> (Дата обращения 04.04.2023)
10. Статья «Android Studio» [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Android_Studio (Дата обращения 15.10.2022)
11. Статья «phpMyAdmin» [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/PhpMyAdmin> (Дата обращения: 05.03.2023)
12. Статья «Фреймворк» [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA> (Дата обращения: 05.04.2023)
13. Статья «User Flow Diagram» [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/496760/> (Дата обращения 10.04.2023)

ПРИЛОЖЕНИЕ А

Исходный код класса UserManager

```
package

com.vg276.beautyapp.server

import android.util.Log
import
com.vg276.beautyapp.utils.SettingsPreferences
import com.vg276.beautyapp.utils.bytesToHex
import
com.vg276.beautyapp.utils.hexToBytes
import java.util.*

/*
 * Пароль сохраняется без шифрования
 * Этот класс создан для управления аккаунтом пользователя
 */

class UserManager(server: SQLServerCore)
{
    private val core = server

    companion object
    {
        // тэги для сохранения
        параметров const val
        PHONE_TAG = "phone"
        const val USER_NAME_TAG =
        "username" const val PASSWORD_TAG =
        "password" const val ROLE_TAG =
        "role"

        // тип ошибок
        enum class UserAccountError
        {
            // Успех
            Success,

            // Произошла ошибка (возможно неверен пароль или номер
            телефона) Failure,

            // Пользователь
            зарегистрирован UserFounded,

            // Ошибка с подключением к серверу или нет подключения к интернету
            ConnectionFailure
        }

        // обратный вызов
        interface UserAccountCallback
        {
            fun onAccount(error: UserAccountError)
        }
    }
}
```

```

// создаем ссылку и используем класс
private var manager : UserAccountManager? = null
fun instance(server: SQLServerCore) : UserAccountManager

{
    if (manager == null)
        manager =
        UserAccountManager(server) return
        manager!!
    }
}

// Процесс регистрации и сохранения параметров для входа
fun startRegistration(userName: String, password: String, phone: String,
    callback: UserAccountCallback?)
{
    core.checkUserForm(phone, object : UserFormListener
    {
        override fun onResult(result: UserFormError, any:
        Any?) { when (result) {
            UserFormError.ConnectFailure -
            >
callback?.onAccount(UserAccountError.ConnectionFailure)
            UserFormError.UserCheckSuccess ->
callback?.onAccount(UserAccountError.UserFounded) UserFormError.UserCheckFailure
            ->
            {
                val u = bytesToHex(userName.toByteArray(Charsets.UTF_8))
                val role =
                UserPrivileges.USER.toString().toLowerCase(Locale.US)
                core.createUser(u, password, phone, role, object :
                UserFormListener {
                    override fun onResult(result: UserFormError, any:
                    Any?) { when (result) {
                        UserFormError.ConnectFailure -
                        >
callback?.onAccount(UserAccountError.ConnectionFailure)
                        UserFormError.FormCreateFailure -
                        >
callback?.onAccount(UserAccountError.Failure)
                        UserFormError.FormCreated ->
                        {

                            phone: String,
                            callback: UserAccountCallback?)
                    {
                        core.checkUserForm(phone, object : UserFormListener
                        {
                            override fun onResult(result: UserFormError, any:
                            Any?) { when (result) {
                                UserFormError.ConnectFailure -
                                >

```



```

    {
        override fun onResult(result: UserFormError, any:
            Any?) { when(result)
            {
                UserFormError.ConnectFailure
                >
callback?.onAccount(UserAccountError.ConnectionFailure)
                UserFormError.AuthFailure ->
callback?.onAccount(UserAccountError.Failure)
                UserFormError.AuthSuccess ->
                {
                    // save
                    if (!isUserAuth())
                    {
                        val u = any as String
                        core.getUserPrivileges(phone, object : UserFormListener
                        {
                            override fun onResult(result: UserFormError, any: Any?)
                            { val privilege = any as UserPrivileges
                                val role = privilege.toString().toLowerCase(Locale.US)
                                SettingsPreferences.instance(core.getContext()).putString(USER_NAME_TAG,
                                    u) SettingsPreferences.instance(core.getContext()).putString(PHONE_TAG,
                                    phone)
password
                                SettingsPreferences.instance(core.getContext()).putString(PASSWORD_TAG,
)
                                SettingsPreferences.instance(core.getContext()).putString(ROLE_TAG, role)
                            }
                        })
                    }
                }
                callback?.onAccount(UserAccountError.Success)
            }
        }
    }
})
}

// Проверка сохраненных параметров для входа
fun isUserAuth() : Boolean
{
    val isPhone = SettingsPreferences.instance(core.getContext()).contains(PHONE_TAG)
    val isPassword =
SettingsPreferences.instance(core.getContext()).contains(PASSWORD_TAG) val isRole =
SettingsPreferences.instance(core.getContext()).contains(ROLE_TAG)
    return isPhone && isPassword && isRole
}

// Автоход в аккаунт пользователя
fun autoLogin(callback: UserAccountCallback?)
{
    val phone = getUserPhone()
    val password = SettingsPreferences.instance(core.getContext()).getString(PASSWORD_TAG,
    "") startLogin(phone, password, callback)
}

fun getUserPhone() : String
{
    return SettingsPreferences.instance(core.getContext()).getString(PHONE_TAG, "")
}

```

```
}  
  
fun getUsername() : String  
  
{  
    val userName = SettingsPreferences.instance(core.getContext()).getString(USER_NAME_TAG,  
        "") return String(hexToBytes(userName), Charsets.UTF_8)  
}  
  
fun getUserPrivilege() : UserPrivileges  
{  
    val role = SettingsPreferences.instance(core.getContext()).getString(ROLE_TAG,  
        "") return UserPrivileges.valueOf(role.toUpperCase(Locale.US))  
}  
  
fun forgot()  
{  
  
}  
}
```

ПРИЛОЖЕНИЕ Б

Исходный код класса SQLServerCore

```
package com.vg276.beautyapp.server

import
android.content.Context
import
android.os.StrictMode
import com.BoardiesITSolutions.AndroidMySQLConnector.*
import
com.BoardiesITSolutions.AndroidMySQLConnector.Exceptions.InvalidSQLPacketException
import com.BoardiesITSolutions.AndroidMySQLConnector.Exceptions.MySQLConnException
import com.BoardiesITSolutions.AndroidMySQLConnector.Exceptions.MySQLException
import
com.vg276.beautyapp.container.RecordContainer
import
com.vg276.beautyapp.container.SpecialistContainer
import com.vg276.beautyapp.utils.hexToBytes
import
com.vg276.beautyapp.utils.logE
import
com.vg276.beautyapp.utils.logI
import java.io.IOException
import java.util.*

// Номер организации также нужен для входа под доступом - Администратор
// Логин - +79234316058
// Пароль - beauty_admin_password_2023

// Таблицы
const val TABLE_USERS = "users_table" // "CREATE TABLE users_table(Username TEXT>Password
CHAR(30),Phone CHAR(30),Role CHAR(20))"
const val TABLE_SPECS = "specs_table" // "CREATE TABLE
specs_table(LastName TEXT,FirstName TEXT,MiddleName TEXT,Biography TEXT,Phone
CHAR(30))"
const val TABLE_RECORDS = "records_table" // "CREATE TABLE records_table(RecId
TEXT,FromUser CHAR(30),ToUser CHAR(30),TimeStamp TEXT)"

// Уровни доступа
enum class
UserPrivileges
{
    // Обычный
    пользователь USER,
    // Сотрудник (специалист)
    STAFFER,
    //
    Администратор
    ADMIN
}

// Статусы
enum class UserFormError
{
```

```

// Статус подключения к серверу
SQL ConnectSuccess,
ConnectFailure,

// Статус
авторизации
AuthFailure,
AuthSuccess,

// Статус проверки пользователя на
сервере UserCheckFailure,
UserCheckSuccess,

// Статус регистрации нового
пользователя FormCreated,
FormCreateFailure,

// Статус для получения
превильегий GetUserPrivileges,

// Статус для получения данных о
специалистах GetAllSpecs,

// Статус для получения данных о всех
записях GetAllRecords
}

// Обратный вызов
interface
UserFormListener
{
    fun onResult(result: UserFormError, any: Any?)
}

class SQLServerCore(context: Context)
{
    private val ctx = context
    private var connection : Connection? = null

    init {
        strictModeInit()
    }

    // Игнорируем проверки
    безопасности private fun
    strictModeInit()
    {
        val policy =
            StrictMode.ThreadPolicy.Builder().permitAll().build()
            StrictMode.setThreadPolicy(policy)
    }

    // Контекст приложения или
    активности fun getContext() : Context
    {
        return ctx
    }
}

```

```

// Подключение к серверу SQL
fun connect(listener: UserFormListener?)
{
    val callback = object : IConnectionInterface
    {
        override fun handleMySQLConnException(p0:
            MySQLConnException?) { logE(p0.toString())
            listener?.onResult(UserFormError.ConnectFailure, null)
        }
        override fun handleException(p0: java.lang.Exception?)
        { logE(p0.toString())
            listener?.onResult(UserFormError.ConnectFailure,
            null)
        }

        override fun handleIOException(p0: IOException?) {
            logE(p0.toString())
            listener?.onResult(UserFormError.ConnectFailure,
            null)
        }

        override fun handleMySQLException(p0:
            MySQLException?) { logE(p0.toString())
            listener?.onResult(UserFormError.ConnectFailure, null)
        }

        override fun
            actionCompleted() {
            logI("SQL server
            connected!")
            listener?.onResult(UserFormError.ConnectSuccess, null)
        }

        override fun handleInvalidSQLPacketException(p0:
            InvalidSQLPacketException?) { logE(p0.toString())
            listener?.onResult(UserFormError.ConnectFailure, null)
        }
    }
    try {
        connection = Connection(
            "sql4.freemsql.com",
            "sql4410388",
            "UHKHvhVwZi", 3306,
            "sql4410388", callback)
    } catch (e :
        Exception) {
        e.printStackTrace()
    }
}

// Переподключение к серверу
fun reconnect()
{
    if (connection ==
        null)
        connect(null)
}

```

```

// Разрываем подключение к серверу
fun closeConnection()
{
    connection?.close()
    connection =
    null
}

// Можно создать новую
таблицу fun createTable(table:
String)
{
    val callback = object : IConnectionInterface
    {
        override fun handleMySQLConnException(p0:
MySQLConnException?) { logE(p0.toString())
        }

        override fun handleException(p0:
java.lang.Exception?) { logE(p0.toString())
        }

        override fun handleIOException(p0:
IOException?) { logE(p0.toString())
        }

        override fun handleMySQLException(p0:
MySQLException?) { logE(p0.toString())
        }

        override fun
        actionCompleted() {
            logI("createTable success!")
        }

        override fun handleInvalidSQLPacketException(p0:
InvalidSQLPacketException?) { logE(p0.toString())
        }
    }

    connection?.let {
        val statement =
        it.createStatement()
        statement.execute(table,
        callback)
    }
}

// Регистрация пользователя (создание новых ячеек в базе данных)
fun createUser(userName: String, password: String, phone: String, role: String,
listener: UserFormListener)
{
    val callback = object : IConnectionInterface
    {
        override fun handleMySQLConnException(p0:
MySQLConnException?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure, null)
        }
    }
}

```

```

override fun handleException(p0: java.lang.Exception?) {
    logE(p0.toString())
    listener.onResult(UserFormError.FormCreateFailure, null)
}

override fun handleIOException(p0: IOException?) {
    logE(p0.toString())
    listener.onResult(UserFormError.FormCreateFailure, null)
}

override fun handleMySQLException(p0:
    MySQLException?) { logE(p0.toString())
    listener.onResult(UserFormError.FormCreateFailure, null)
}

override fun
    actionCompleted() {
        logI("createUser success!")
        listener.onResult(UserFormError.FormCreated, null)
    }

override fun handleInvalidSQLPacketException(p0:
    InvalidSQLPacketException?) { logE(p0.toString())
    listener.onResult(UserFormError.FormCreateFailure, null)
}
}

connection?.let {
    val line = "INSERT INTO $TABLE_USERS (Username, Password, Phone, Role) VALUES
($userName', '$password', '$phone', '$role)"
    val statement =
        it.createStatement()
        statement.execute(line, callback)
    }
}

// Добавляем информацию о специалистах
fun createStaffer(
    lastName: String,
    firstName: String,
    middleName:
String, biography:
String, phone:
String,
    listener: UserFormListener)
{
    val callback = object : IConnectionInterface
    {
        override fun handleMySQLConnException(p0:
            MySQLConnException?) { logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure, null)
        }

        override fun handleException(p0: java.lang.Exception?) {
            logE(p0.toString())
            listener.onResult(UserFormError.FormCreateFailure, null)
        }
    }
}

```

```

override fun handleIOException(p0: IOException?) {
    logE(p0.toString())
    listener.onResult(UserFormError.FormCreateFailure, null)
}

override fun handleMySQLException(p0:
    MySQLException?) { logE(p0.toString())
    listener.onResult(UserFormError.FormCreateFailure, null)
}

override fun
    actionCompleted() {
        logI("createUser success!")
        listener.onResult(UserFormError.FormCreated, null)
    }

override fun handleInvalidSQLException(p0:
    InvalidSQLException?) { logE(p0.toString())
    listener.onResult(UserFormError.FormCreateFailure, null)
}
}

connection?.let {
    val line = "INSERT INTO $TABLE_SPECS (LastName, FirstName, MiddleName, Biography,
Phone) VALUES ('$lastName', '$firstName','$middleName','$biography','$phone')"
    val statement =
        it.createStatement()
        statement.execute(line, callback)
    }
}

// Проверка пользователя в базе данных
fun checkUserForm(phone: String, listener: UserFormListener)
{
    val callback = object : IResultInterface
    {
        override fun handleMySQLConnException(p0:
            MySQLConnException?) { logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure, null)
        }

        override fun handleException(p0:
            java.lang.Exception?) { logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure,
            null)
        }

        override fun handleIOException(p0: IOException?) {
            logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure,
            null)
        }

        override fun executionComplete(p0:
            ResultSet?) { p0?.let {
                var row = it.nextRow
                var result =

```

```

        UserFormError.UserCheckFailure while
        (row != null)
        {
            val p =
            row.getString("Phone") if (p
            == phone)
            {
                result =
                UserFormError.UserCheckSuccess
                break
            }
            row = it.nextRow
        }
        listener.onResult(result, null)
    }
}

override fun handleMySQLException(p0:
MySQLException?) { logE(p0.toString())
listener.onResult(UserFormError.ConnectFailure, null)
}

override fun handleInvalidSQLPacketException(p0:
InvalidSQLPacketException?) { logE(p0.toString())
listener.onResult(UserFormError.ConnectFailure, null)
}

}

connection?.let {
    val line = "SELECT * FROM $TABLE_USERS WHERE
    Phone='$phone'" val statement = it.createStatement()
    statement.executeQuery(line, callback)
}
}

// Авторизация пользователя
fun doLogin(phone: String, password: String, listener: UserFormListener)
{
    val callback = object : IResultInterface
    {
        override fun handleMySQLConnException(p0:
        MySQLConnException?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure, null)
        }

        override fun handleException(p0:
        java.lang.Exception?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure,
        null)
        }

        override fun handleIOException(p0: IOException?) {
        logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure,
        null)
        }
    }
}

```

```

override fun executionComplete(p0:
    ResultSet?) { p0?.let {
        var row = it.nextRow
        var result =
            UserFormError.AuthFailure var u =
            ""
        while (row != null)
        {
            val p =
                row.getString("Password") if (p
                    == password)
            {
                logI("Auth success!")
                u = row.getString("Username")
                result =
                    UserFormError.AuthSuccess
                break
            }
            row = it.nextRow
        }
        listener.onResult(result, u)
    }
}

override fun handleMySQLException(p0:
    MySQLException?) { logE(p0.toString())
    listener.onResult(UserFormError.ConnectFailure, null)
}

override fun handleInvalidSQLPacketException(p0:
    InvalidSQLPacketException?) { logE(p0.toString())
    listener.onResult(UserFormError.ConnectFailure, null)
}

}

connection?.let {
    val line = "SELECT * FROM $TABLE_USERS WHERE
        Phone='$phone'" val statement = it.createStatement()
    statement.executeQuery(line, callback)
}
}

// Получение уровня доступа пользователя
fun getUserPrivileges(phone: String, listener: UserFormListener)
{
    val callback = object : IResultInterface
    {
        override fun handleMySQLConnException(p0:
            MySQLConnException?) { logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure, null)
        }

        override fun handleException(p0:
            java.lang.Exception?) { logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure,
                null)
        }
    }
}

```

```

override fun handleIOException(p0: IOException?) {
    logE(p0.toString())
    listener.onResult(UserFormError.ConnectFailure,
        null)
}
override fun executionComplete(p0:
    ResultSet?) { p0?.let {
        var row =
            it.nextRow while
            (row != null)
            {
                val login =
                    row.getString("Phone") if (login
                        == phone)
                {
                    val role = row.getString("Role")
                    val privilege =
                        UserPrivileges.valueOf(role.toUpperCase(Locale.US))
                    listener.onResult(UserFormError.GetUserPrivileges, privilege)
                    break
                }
                row = it.nextRow
            }
        }
    }
}
override fun handleMySQLException(p0:
    MySQLException?) { logE(p0.toString())
    listener.onResult(UserFormError.ConnectFailure, null)
}

override fun handleInvalidSQLPacketException(p0:
    InvalidSQLPacketException?) { logE(p0.toString())
    listener.onResult(UserFormError.ConnectFailure, null)
}

}

connection?.let {
    val line = "SELECT * FROM $TABLE_USERS WHERE
        Phone='$phone'" val statement = it.createStatement()
    statement.executeQuery(line, callback)
}
}
// Получение данных из таблицы
specs_table fun getListSpecs(listener:
    UserFormListener)
{
    val callback = object : IResultInterface
    {
        override fun handleMySQLConnException(p0:
            MySQLConnException?) { logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure, null)
        }
    }

    override fun handleException(p0:
        java.lang.Exception?) { logE(p0.toString())
}

```

```

        listener.onResult(UserFormError.ConnectFailure,
            null)
    }

    override fun handleIOException(p0: IOException?) {
        logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure,
            null)
    }

    override fun executionComplete(p0:
        ResultSet?) { p0?.let {
        var row = it.nextRow
        val list: ArrayList<SpecialistContainer> =
            ArrayList() while (row != null)
        {
            val spec = SpecialistContainer()
            spec.lastName = String(hexToBytes(row.getString("LastName")), Charsets.UTF_8)
            spec.firstName = String(hexToBytes(row.getString("FirstName")), Charsets.UTF_8)
            spec.middleName = String(hexToBytes(row.getString("MiddleName")),
                Charsets.UTF_8) spec.biography = String(hexToBytes(row.getString("Biography")),
                Charsets.UTF_8) spec.phone = row.getString("Phone")
            list.add(spec)
            row =
                it.nextRow
        }
        listener.onResult(UserFormError.GetAllSpecs, list)

    }
}

    override fun handleMySQLException(p0:
        MySQLException?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure, null)
    }

    override fun handleInvalidSQLPacketException(p0:
        InvalidSQLPacketException?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure, null)
    }

}

connection?.let {
    val line = "SELECT * FROM
    $TABLE_SPECS" val statement =
        it.createStatement()
        statement.executeQuery(line, callback)
    }
}

// Запись на услуги
fun recordUser(
    fromUser:
    String, toUser:
    String,
    timestamp:

```

```

    String,
    listener: UserFormListener?)
{
    val callback = object : IConnectionInterface
    {
        override fun handleMySQLConnException(p0:
            MySQLConnException?) { logE(p0.toString())
            listener?.onResult(UserFormError.ConnectFailure, null)
        }

        override fun handleException(p0: java.lang.Exception?) {
            logE(p0.toString())
            listener?.onResult(UserFormError.FormCreateFailure, null)
        }

        override fun handleIOException(p0: IOException?) {
            logE(p0.toString())
            listener?.onResult(UserFormError.FormCreateFailure, null)
        }

        override fun handleMySQLException(p0:
            MySQLException?) { logE(p0.toString())
            listener?.onResult(UserFormError.FormCreateFailure,
            null)
        }

        override fun
            actionCompleted() {
            logI("recordUser success!")
            listener?.onResult(UserFormError.FormCreated, null)
        }

        override fun handleInvalidSQLPacketException(p0:
            InvalidSQLPacketException?) { logE(p0.toString())
            listener?.onResult(UserFormError.FormCreateFailure, null)
        }
    }
}

connection?.let {
    val id = UUID.randomUUID().toString()
    val line = "INSERT INTO $TABLE_RECORDS (RecId, FromUser, ToUser, TimeStamp)
VALUES ('$id', '$fromUser', '$toUser', '$timestamp')"
    val statement =
        it.createStatement()
    statement.execute(line, callback)
}
}

fun getAllRecords(listener: UserFormListener)
{
    val callback = object : IResultInterface
    {
        override fun handleMySQLConnException(p0:
            MySQLConnException?) { logE(p0.toString())
            listener.onResult(UserFormError.ConnectFailure, null)
        }
        }
    override fun handleException(p0:
        java.lang.Exception?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure,

```

```

        null)
    }
    override fun handleIOException(p0: IOException?) {
        logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure,
            null)
    }

    override fun executionComplete(p0:
        ResultSet?) { p0?.let {
            var row = it.nextRow
            val list: ArrayList<RecordContainer> =
                ArrayList() while (row != null)
            {
                val record = RecordContainer()
                record.recId =
                    row.getString("RecId")
                record.fromUser =
                    row.getString("FromUser") record.toUser =
                    row.getString("ToUser") record.timestamp =
                    row.getString("TimeStamp") list.add(record)
                row = it.nextRow
            }
            listener.onResult(UserFormError.GetAllRecords, list)
        }
    }

    override fun handleMySQLException(p0:
        MySQLException?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure, null)
    }

    override fun handleInvalidSQLPacketException(p0:
        InvalidSQLPacketException?) { logE(p0.toString())
        listener.onResult(UserFormError.ConnectFailure, null)
    }

}
connection?.let {
    val line = "SELECT * FROM $TABLE_RECORDS"
    val statement = it.createStatement()
    statement.executeQuery(line,
        callback)
}
}
}

```

Отчет о проверке на займствования №1



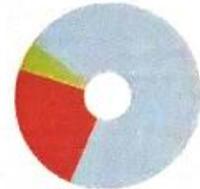
Автор: Андреева Анна Александровна
Проверяющий: Морозова Анна Сергеевна
Организация: Томский Государственный Университет
Отчет предоставлен сервисом «Антиплагиат» - <http://tsu.antiplagiat.ru>

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 39
Начало загрузки: 04.06.2023 05:09:37
Длительность загрузки: 00:00:15
Имя исходного файла: ВКР.docx
Название документа: ВКР
Размер текста: 73 кБ
Символов в тексте: 75013
Слов в тексте: 8559
Число предложений: 823

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Начало проверки: 04.06.2023 05:09:52
Длительность проверки: 00:00:24
Комментарии: не указано
Поиск с учетом редактирования: да
Проверенные разделы: основная часть с. 2-3,5-42, библиография с. 43
Модули поиска: ИПС Адилет, Библиография, Сводная коллекция ЭБС, Интернет Плюс*, Сводная коллекция РГБ, Цитирование, Переводные заимствования (RuEn), Переводные заимствования по eLIBRARY.RU (EnRu), Переводные заимствования по коллекции Гарант: аналитика, Переводные заимствования по коллекции Интернет в английском сегменте, Переводные заимствования по Интернету (EnRu), Переводные заимствования по коллекции Интернет в русском сегменте, Переводные заимствования издательства Wiley, eLIBRARY.RU, СПС ГАРАНТ: аналитика, СПС ГАРАНТ: нормативно-правовая документация, Медицина, Диссертации НББ, Коллекция НБУ, Перефразирования по eLIBRARY.RU, Перефразирования по СПС ГАРАНТ: аналитика, Перефразирования по Интернету, Перефразирования по Интернету (EN), Перефразированные заимствования по коллекции Интернет в английском сегменте, Перефразированные заимствования по коллекции Интернет в русском сегменте, Перефразирования по коллекции издательства Wiley, Патенты СССР, РФ, СНГ, СМИ Россия и СНГ, Шаблоны фразы, Модуль поиска "tsu", Коллцо вузов, Издательство Wiley, Переводные заимствования



СОВПАДЕНИЯ
23,61%

САМОЦИТИРОВАНИЯ
9%

ЦИТИРОВАНИЯ
4,84%

ОРИГИНАЛЬНОСТЬ
71,55%

С результатом ознакомлен весь коллектив А.С.