

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

РАЗРАБОТКА МОБИЛЬНОГО
ПРИЛОЖЕНИЯ ДЛЯ ЗАПИСИ В
АВТОСЕРВИС

Студент

25.05.26
(подпись, дата)

Самофалова Александра Владимировна
(ФИО полностью)

Руководитель

30.05.26
(подпись, дата)

А.В. Кузнецов
(фамилия, инициалы)

Нормоконтролёр

30.05.26
(подпись, дата)

А.С. Бычков
(фамилия, инициалы)

Работа допущена к защите (протокол заес. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой

(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

19 июня 2026г. № 37)

Секретарь ГЭК

(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 07 » 04 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Самофалова Александра Владимировна гр. 22ВВП2 факультета вычислительной техники

2. Тема работы Разработка мобильного приложения для записи в автосервис

Тема утверждена приказом ПГУ № 347/о _____ от "07" _____ апреля _____ 2026 г.

3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Кузнецов А.В.

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

Разработать мобильное приложение, позволяющее записываться на обслуживание автомобиля онлайн, и его серверную часть. Язык программирования Kotlin, среда разработки клиентской части Android Studio, среда разработки серверной части IntelliJ IDEA. Программа должна обладать интуитивно понятным интерфейсом и содержать следующие технологии:

1. REST API
2. Асинхронное программирование
3. JSON Web Token
4. Material design 3

5. Перечень подлежащих разработке вопросов

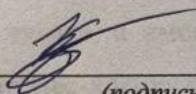
Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки мобильного приложения и серверной части. Выбрать язык программирования, среду разработки и базу данных

6. Календарный график выполнения работы

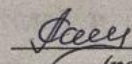
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	21	07.04.26 – 09.04.26	
2	Анализ требований	21	10.04.26 – 12.04.26	
3	Выбор технологий, применяемых при проектировании и разработке	16	13.04.26 – 14.04.26	
4	Разработка серверной части	91	15.04.26 – 30.04.26	
5	Создание графического интерфейса	64	01.05.26 – 07.05.26	
6	Разработка мобильного приложения	90	08.05.26 – 18.05.26	
7	Оформление пояснительной записки	21	19.05.26 – 21.05.26	

Дата выдачи задания "07" апреля 2026г.

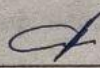
Руководитель бакалаврской работы


(подпись, дата)

Задание к исполнению принял студент


(подпись, дата)

Работу к защите допустить
Декан факультета

 Л. Р. Ресеева
17.06.2022 (подпись, дата)

Реферат

Бакалаврская работа содержит 130 листов, 45 рисунков, 4 таблицы, 12 источников и 3 приложения.

АВТОСЕРВИС, ОНЛАЙН-ЗАПИСЬ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ANDROID, KOTLIN, SPRING BOOT, POSTGRESQL, БАЗА ДАННЫХ, JWT-АУТЕНТИФИКАЦИЯ

Цель работы – разработать клиент-серверную систему, позволяющую автоматизировать и оптимизировать процесс записи клиентов на техническое обслуживание в автосервис.

Объектами разработки являются:

- серверное приложение на базе фреймворка Spring Boot для обработки входящих запросов, обеспечения безопасности, администрирования рабочих графиков мастеров и сохранения информации в базе данных;
- клиентское мобильное приложение под управлением ОС Android, предназначенное для взаимодействия пользователей с системой, оформления онлайн-записи.

В результате проделанной работы разработано приложение, полностью соответствующее поставленной задаче, а именно: разработана серверная часть приложения, которая реализует бизнес-логику системы, обеспечивает безопасную JWT-аутентификацию пользователей и сохраняет данные в реляционной БД PostgreSQL, создана клиентская часть приложения в виде мобильного Android-приложения на языке Kotlin с графическим интерфейсом, позволяющая клиентам осуществлять онлайн-запись на услуги, а мастерам – просматривать записи и настраивать расписание своей работы.

					ПГУ1.090301.45.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Разработка мобильного приложения для записи в автосервис Пояснительная записка	Лит.	Лист	Листов
Разраб.		Самофалова А.В.		25.05.26				
Провер.		Кузнецов А.В.		30.05.26			2	66
Н. Контр		Бычков А.С.		30.05.26		ФВТ, каф. ВТ гр. 228ВП2		
Утверд.		Митрохин М.А.		30.05.26				

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Анализ предметной области	7
1.1 Анализ актуальности темы	7
1.2 Анализ аналогов	8
1.2.1 Мобильное приложение «FIT SERVICE»	8
1.2.2 Мобильное приложение «UREMONT»	10
1.2.3 Мобильное приложение «Wilgood»	12
1.2.4 Вывод	14
1.3 Постановка задачи на разработку	14
2 Выбор инструментов разработки	17
2.1 Выбор языка программирования	17
2.1.1 Язык программирования «Java»	17
2.1.2 Язык программирования «Kotlin»	17
2.1.3 Вывод	18
2.2 Выбор фреймворка для разработки серверной части	18
2.3 Выбор среды разработки	19
2.3.1 Выбор среды разработки для клиентской части	19
2.3.2 Выбор среды разработки для серверной части	21
2.4 Выбор базы данных	22
3 Архитектура приложения	23
3.1 Архитектура клиентской части	23
3.2 Архитектура серверной части	24
4 Реализация приложения	25
4.1 Описание функционала	25
4.2 Описание работы приложения	26
4.3 Реализация клиентской части	30
4.3.1 Presentation layer	31

4.3.2 Domain layer	33
4.3.3 Data layer	34
4.4 Реализация серверной части	35
5 Тестирование приложения	40
5.1 Регистрация	40
5.2 Авторизация	42
5.3 Добавление автомобиля	44
5.4 Поиск и запись к мастеру	48
5.5 Редактирование профиля	51
5.6 Редактирование графика работы мастера	54
5.7 Изменение статуса записи	57
5.8 Добавление оценки и отзыва	60
5.9 Вывод	62
6 Безопасность жизнедеятельности	63
6.1 Правила и техника безопасности при работе с ПК	63
6.2 Правила и техника безопасности при работе со смартфоном	64
Заключение	65
Список использованных источников	66
Приложение А – UML-диаграммы	67
Приложение Б – Листинг программы	73
Приложение В – Презентация	124

Перечень сокращений и обозначений

В данной работе применяют следующие сокращения и обозначения.

ПК – персональный компьютер

ПО – программное обеспечение

СТО – станция техобслуживания

СУБД – система управления базами данных

API – (от англ. Application Programming Interface) – интерфейс программирования приложений)

HTTP – (от англ. Integrated Development Environment) – протокол передачи данных

IDE – (от англ. Integrated Development Environment) – интегрированная среда разработки

JSON – (от англ. JavaScript Object Notation) – текстовый формат обмена данными
JVM – Java Virtual Machine (виртуальная машина Java)

REST – (от англ. Representational State Transfer) – стиль взаимодействия компонентов приложения в сети

SDK – (от англ. Software Development Kit) – набор инструментов, библиотек, документации и других ресурсов

SQL – (от англ. Structured Query Language) – язык структурированных запросов

UML – (от англ. Unified Modeling Language) – унифицированный язык моделирования

Введение

В современном мире многие люди являются владельцами автомобилей. Автомобили требуют периодического обслуживания или могут ломаться. В связи с этим у автовладельцев возникает потребность записи в автосервис и поиска хорошего мастера, что заставляет обзванивать большое количество автосервисов и тратить много времени.

Использование мобильного приложения может стать хорошим решением для повышения удобства записи в автосервис. Такое приложение поможет ускорить и облегчить поиск автосервиса, выбирать мастеров по рейтингу и отзывам других пользователей, хранить информацию об автомобилях владельца и их обслуживании, а также записываться на обслуживание в автосервис, не тратя на это много времени.

Цели выпускной квалификационной работы:

- исследовать тему и существующие решения;
- определить функциональные требования к приложению для записи в автосервис;
- разработать серверную часть приложения;
- реализовать мобильный клиент с удобным и современным интерфейсом;
- провести тестирование разработанного приложения, проверив основные функции;
- оценить практическую применимость и удобство использования;
- предложить рекомендации по дальнейшему развитию и расширению функционала;
- сделать выводы о проделанной работе и предложить идеи дальнейшего развития данного приложения.

1 Анализ предметной области

1.1 Анализ актуальности темы

В настоящее время автомобили являются одним из наиболее распространенных видов транспорта, а средний возраст легковых автомобилей в России составляет 15,5 лет, что является причиной большого спроса на услуги технического обслуживания и ремонта транспортных средств [1].

Согласно исследованиям на долю официальных дилеров приходится 8% от объема рынка по техобслуживанию и ремонту, на независимые СТО 37%, а на частных мастеров 54% [2]. В таких условиях перед клиентами встает вопрос поиска проверенного мастера, а перед несетевым СТО – привлечения клиентов.

Несмотря на то, что наше время происходит цифровизация сфер медицины, доставки, такси, в сфере автосервиса запись часто по-прежнему ведется по телефону. Такой способ записи оказывается неудобным вариантом для клиентов из-за необходимости ожидания ответа, отсутствия информации о доступном времени записи, соответственно этот процесс расходует много времени.

Использование мобильных приложений для онлайн-записи имеет значительные преимущества для автовладельцев, такие как:

- экономия времени. Сравнить несколько сервисов через приложение по цене, выбрать удобное время через календарь оказывается значительно быстрее, чем обзванивать каждый сервис отдельно по телефону;
- возможность записаться на услугу в нерабочее время;
- отслеживание истории обслуживания автомобиля;
- надежность, так как владелец может выбрать понравившийся автосервис по отзывам других клиентов и не переживать о качестве обслуживания.

Для СТО мобильные приложения – это инструмент для привлечения новых клиентов, автоматизации процесса записи.

В связи с этим разработка мобильного приложения для записи в автосервис является актуальной задачей. Такое приложение позволит повысить удобство взаимодействия между клиентами и сервисами.

1.2 Анализ аналогов

На данный момент уже существуют реализации мобильных приложений для записи в автосервис. В качестве аналогов рассмотрены следующие приложения:

- FIT SERVICE;
- UREMONT;
- Wilgood;

Проведен анализ, выявлены сильные и слабые стороны каждого приложения.

1.2.1 Мобильное приложение «FIT SERVICE»

Приложение предназначено для записи в автосервис сети «FIT SERVICE» и поиска ближайших станций [3]. Интерфейс приложения представлен на рисунках 1 и 2.

Достоинства:

- удобная онлайн-запись. Можно выбрать удобное время и дату, а также записаться на определенную услугу;
- возможность описания проблемы в чате с поддержкой, где помогут решить проблему;
- поиск автосервисов. В приложении реализовано отображение автосервисов на карте;
- сервисная книга автомобиля, где отображается история обслуживания;
- возможность отслеживания каждого этапа ремонта.

Недостатки:

- приложение создано для конкретной сети автосервисов;
- коммуникация осуществляется только через операторов поддержки;
- отсутствует возможность выбора конкретного мастера.

					ПГУ1.090301.45.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

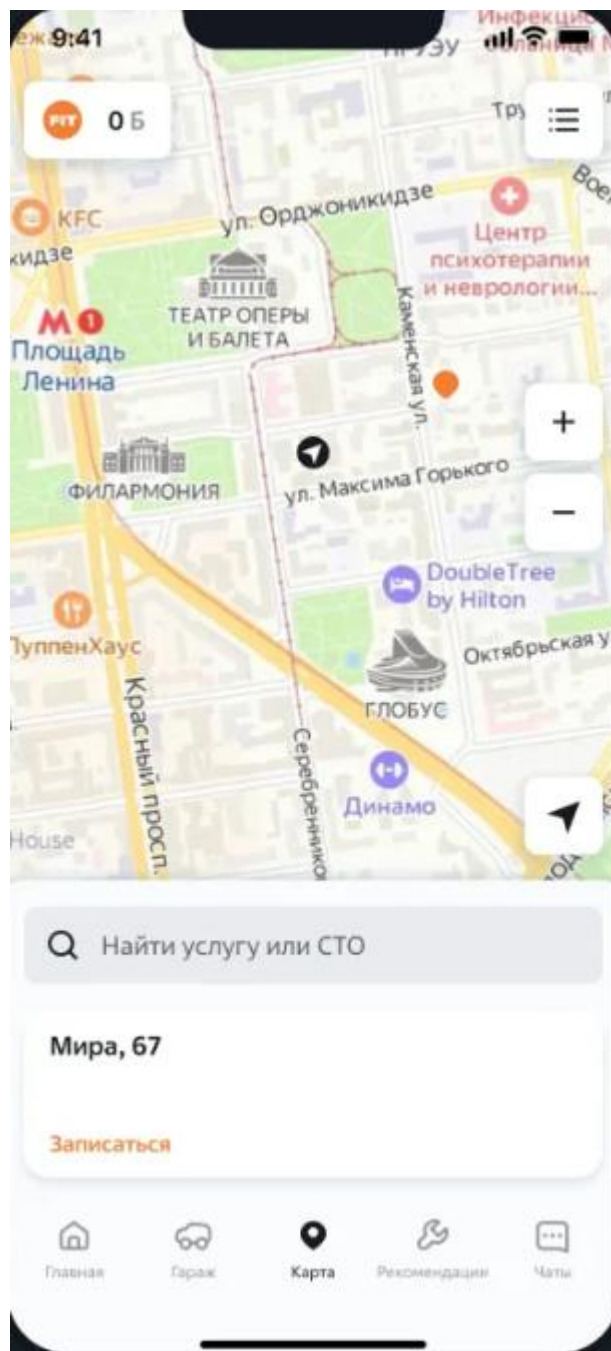


Рисунок 1 – Страница карты с СТО поблизости

Рисунок 2 – Страница записи на СТО

1.2.2 Мобильное приложение «UREMONT»

Мобильное приложение, позволяющее пользователям находить сервисы, сравнивать предложения и записываться на услуги [4]. Интерфейс приложения представлен на рисунке 3.

Достоинства:

- умный подбор СТО с помощью интеллектуального помощника;
- отображение автосервисов и цен на карте;

					ПГУ1.090301.45.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

- возможность отслеживания этапов ремонта;
- чат с СТО;
- история обслуживания.

Недостатки:

- отсутствует возможность выбора конкретного мастера;
- при создании заявки можно выбрать только ориентировочное время записи на услугу;
- перегруженный интерфейс приложения из-за большого количества услуг, таких как оформление страхового полиса и получение предложений от страховых компаний.

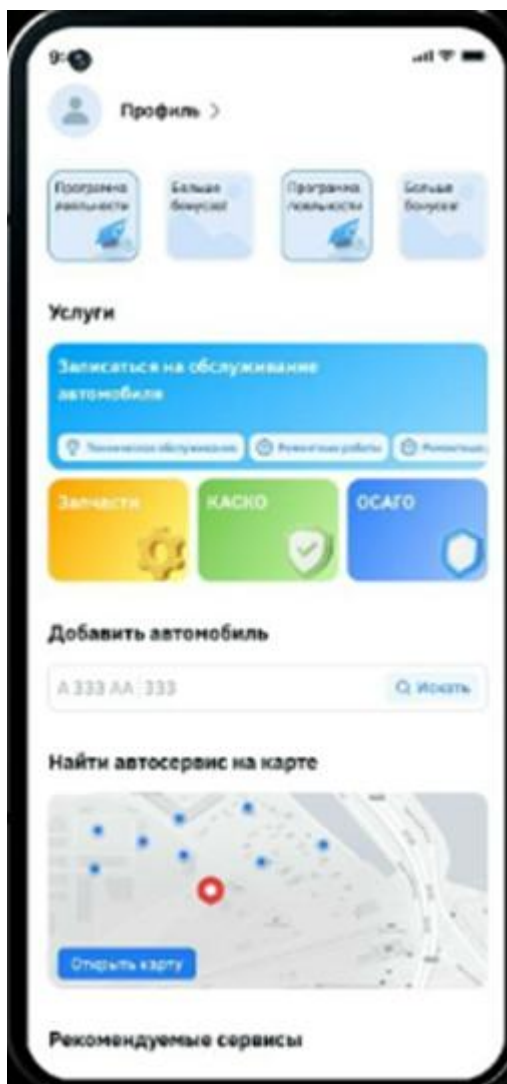


Рисунок 3 – Главная страница с доступными функциями

1.2.3 Мобильное приложение «Wilgood»

Приложение сети автосервисов, позволяющее упростить взаимодействие сервисов с клиентами и записи на обслуживание автомобиля [5]. Интерфейс приложения представлен на рисунках 4 и 5.

Достоинства:

- поиск автосервиса с помощью геолокации;
- удобная онлайн запись в автосервис с выбором даты и времени;
- хранение истории обслуживания автомобиля;
- отображение статуса ремонта.

Недостатки:

- приложение работает только с конкретной сетью автосервисов;
- невозможность выбрать мастера, у которого клиент хотел бы обслуживаться.



Рисунок 4 – Главная страница приложения



Рисунок 5 – Страница истории обслуживания

1.2.4 Вывод

Сравнительный анализ аналогов представлен в таблице 1

Таблица 1 – Достоинства и недостатки приложений для онлайн-записи

	FIT SERVICE	UREMONT	Wilgood
Карта	+	+	+
Отслеживание статуса ремонта	+	+	+
История обслуживания	+	+	+
Выбор мастера	-	-	-
Удобство выбора даты и времени	+	-	+

В результате проведенного анализа существующих аналогов выяснилось, что их функционал почти не меняется. Основное отличие заключается в том, что одни приложения работают для определенной сети автосервисов, а другие позволяют найти и сравнить различные сервисы. Стоит отметить, что основным недостатком всех аналогов является отсутствие возможности выбора мастера, хотя большинство автовладельцев предпочтут получить услугу у проверенного мастера. Поэтому разработка мобильного приложения для записи в автосервис, является актуальной темой.

1.3 Постановка задачи на разработку

Необходимо разработать приложение для телефонов на базе операционной системы Android, позволяющее осуществлять поиск автосервиса и записываться на обслуживание онлайн.

Требования к клиентской части:

– реализация экранов регистрации и авторизации пользователя, валидация данных клиента;

– форма для текстового описания проблемы, календарь для выбора даты и список для выбора доступного времени, экран выбора мастера;

– отображение списка мастеров;

– отображение списка текущих и завершенных услуг;

– экран гаража с информацией об автомобилях пользователя;

– форма для отзыва с выбором оценки по 5-балльной шкале и текстовым полем. Просмотр списка отзывов других пользователей в карточке мастера.

– отображение списка записей, назначенных на текущего мастера;

– управление статусом заказа;

– просмотр деталей заказа;

– управление учетной записью мастера и графиком работы для мастера, редактирование и добавление личной информации для всех пользователей.

Минимальные системные требования для клиента:

– операционная система Android 8.0;

– наличие подключения к интернету.

Требования к серверной части:

– обработка запросов на регистрацию и авторизацию, разграничение прав доступа для ролей «Клиент», «Мастер»;

– реализация создания записи в базе данных с привязкой к идентификатору пользователя и мастера с логикой проверки доступности мастера в выбранное время;

– хранение о мастерах и ее передача;

– хранение истории ремонта в базе данных и обработка запросов на получение истории;

– хранение отзывов, пересчет среднего рейтинга мастера при добавлении новой оценки;

					ПГУ1.090301.45.001 ПЗ	Лист
						15
Изм.	Лист	№ докум.	Подпись	Дата		

– создание, чтение, обновление, удаление информации о пользователях, мастерах, записях и автомобилях.

Минимальные системные требования:

- сервер Linux Ubuntu 24.04;
- наличие стабильного интернет-соединения.

					ПГУ1.090301.45.001 ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

2 Выбор инструментов разработки

Для эффективной разработки качественного продукта необходимо выбрать подходящие инструменты. Это позволит избежать проблем и ускорить процесс разработки.

2.1 Выбор языка программирования

Создание качественного мобильного приложения для онлайн-записи в автосервис во многом зависит от грамотного выбора языка программирования. Данный инструмент должен гарантировать высокую скорость обработки данных, защищенность информации и общую отказоустойчивость системы.

На сегодняшний день в индустрии мобильной разработки ключевыми конкурирующими технологиями являются Java и Kotlin. С целью выявления оптимального инструмента в работе проводится аналитический обзор ключевых характеристик и специфики применения каждого из указанных языков.

2.1.1 Язык программирования «Java»

Java является одним из наиболее популярных языков программирования для разработки на Android [6]. Ключевой особенностью языка является платформенная независимость благодаря использованию виртуальной машины Java (JVM). Код, написанный на Java, компилируется в байт-код, который может запускаться на любой платформе, где установлена JVM.

Java реализует объектно-ориентированный подход, существует много библиотек и фреймворков, что облегчает разработку на этом языке, имеется много информации о языке в открытом доступе. Java может использоваться и для разработки серверного приложения.

2.1.2 Язык программирования «Kotlin»

Kotlin – современный язык со статической типизацией [7]. Kotlin в последнее время считается более приоритетным языком программирования для разработки мобильных приложений. Kotlin работает на платформе JVM и

полностью совместим с Java, что позволяет использовать библиотеки, написанные для Java, имеет поддержку функционального программирования.

Существенным достоинством является наличие в языке Kotlin корутинов, позволяющих выполнять операции в асинхронном режиме. Корутины являются более легковесными, чем потоки, так как требуют меньше оперативной памяти. Они позволяют выполнять сетевые запросы, например, запись на услугу, загрузку данных о мастерах, асинхронно без блокировки интерфейса приложения, что положительно скажется на пользовательском опыте.

Функции, написанные на Kotlin, требуют меньше кода, чем те же, написанные на Java. Он также имеет Null-безопасность – особенность системы типов, которая позволяет избежать ошибки при обращении к пустым ссылкам на этапе компиляции, что повышает надежность работы приложения.

Kotlin может применяться как для разработки клиентской части мобильного приложения, так и для серверной.

2.1.3 Вывод

Проанализировав оба языка, было принято решение выбрать язык программирования Kotlin для разработки приложения для записи в автосервис. Kotlin предлагает более современные средства разработки, что позволит минимизировать затраты времени и разработать безопасное приложение с минимальным количеством кода.

2.2 Выбор фреймворка для разработки серверной части

Для упрощения разработки серверной части будет использоваться фреймворк Spring Boot [8].

Выбор обусловлен следующими особенностями:

- фреймворк позволяет автоматически конфигурировать приложение на основе добавленных зависимостей;

– имеет набор стартеров, удобных дескрипторов, которые объединяют библиотеки, используемые для определенных целей в один пакет, что позволяет подключать целые стеки технологий одной строкой;

имеет встроенный сервер, что позволяет не устанавливать дополнительное серверное программное обеспечение в среде развертывания, и приложение сразу готово к выполнению;

– фреймворк Spring Boot позволяет использовать модули экосистемы Spring, которые представляют собой готовые инструменты для решения различных задач, например, работа с базой данных, обеспечение безопасности приложения.

Использование Spring Boot позволит существенно сократить время разработки серверной части.

2.3 Выбор среды разработки

Выбор среды разработки напрямую влияет на эффективность и скорость разработки, поэтому необходимо выбрать подходящую под задачу среду, которая будет предоставлять необходимый функционал и инструменты разработки, отладки и тестирования.

2.3.1 Выбор среды разработки для клиентской части

В качестве среды разработки была выбрана Android Studio – официальная IDE от Google, созданная на основе IntelliJ IDEA, для создания приложений на базе Android [9].

Она поддерживает официальные языки разработки приложений под Android – Kotlin и Java.

Основные возможности Android Studio:

– официальная поддержка Google, благодаря чему она первой получает поддержку новых библиотек, API и инструментов;

– удобный редактор кода с умным автодополнением и проверкой ошибок,

- визуальный конструктор интерфейсов (рисунок 6), который позволяет, видеть изменения в интерфейсе без запуска на устройстве;
- инструменты для отслеживания производительности, позволяющие просматривать использование процессора, памяти, сети, что помогает оптимизировать приложения;
- различные инструменты для отладки, встроенный эмулятор Android, который позволяет тестировать приложения на различных версиях операционной системы и устройствах с различными размерами экрана.

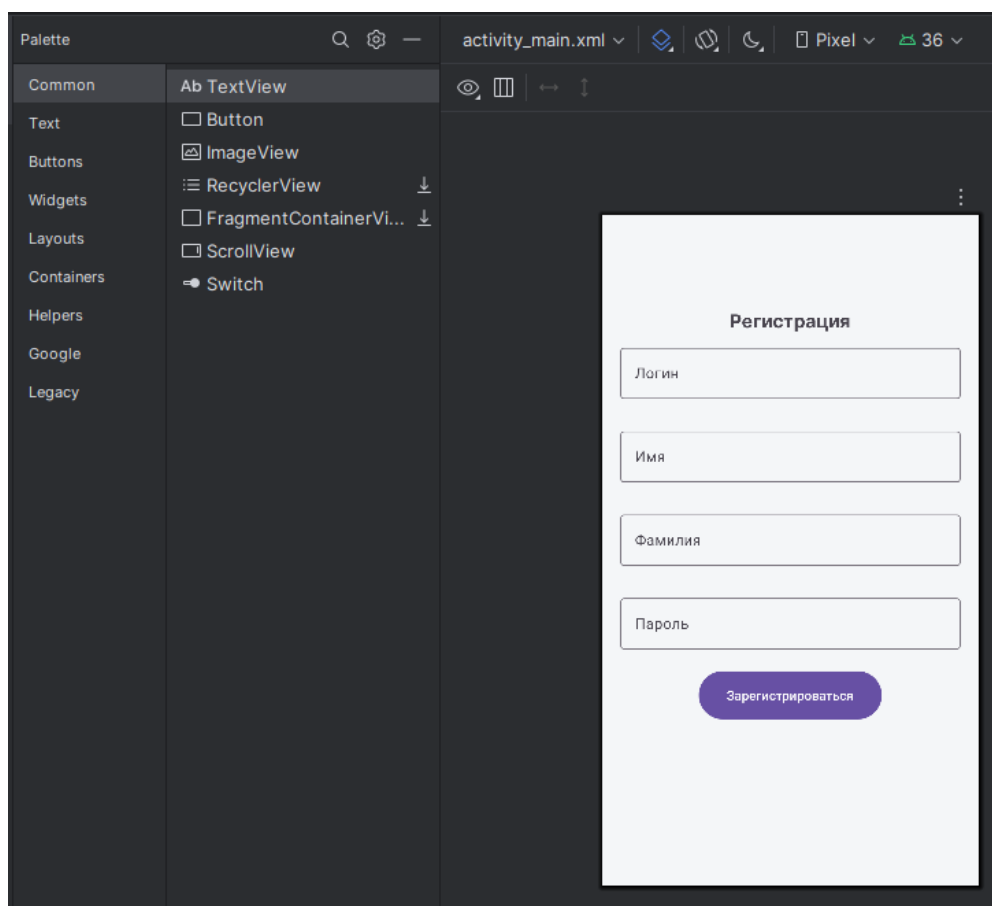


Рисунок 6 – Визуальный конструктор интерфейса

По перечисленным выше причинам Android Studio – хорошее решение для разработки клиентской части Android приложения.

2.3.2 Выбор среды разработки для серверной части

Для разработки серверной части была выбрана IntelliJ IDEA – среда разработки для Java и Kotlin [10].

Основные функциональные возможности IntelliJ IDEA:

- поддержка Kotlin, Java, Spring Gradle, Git, поэтому не требуется дополнительная установка и настройка расширений
- редактор кода с автодополнением. Среда разработки анализирует контекст и предлагает наиболее вероятные варианты автодополнения;
- простой отладчик, позволяющий выявлять ошибки в работе программы, анализировать переменные;
- встроенные инструменты для работы с базами данных. Можно писать SQL-запросы и проверять данные в таблицах в самой среде разработки;
- проверка качества кода с помощью инспекций, при обнаружении ошибок предлагается устранить их с помощью контекстных действий;
- автоматический рефакторинг. Функция позволяет безопасно переименовывать элементы кода, изменять структуру кода.

На рисунке 7 представлены инструменты для работы с базами данных.

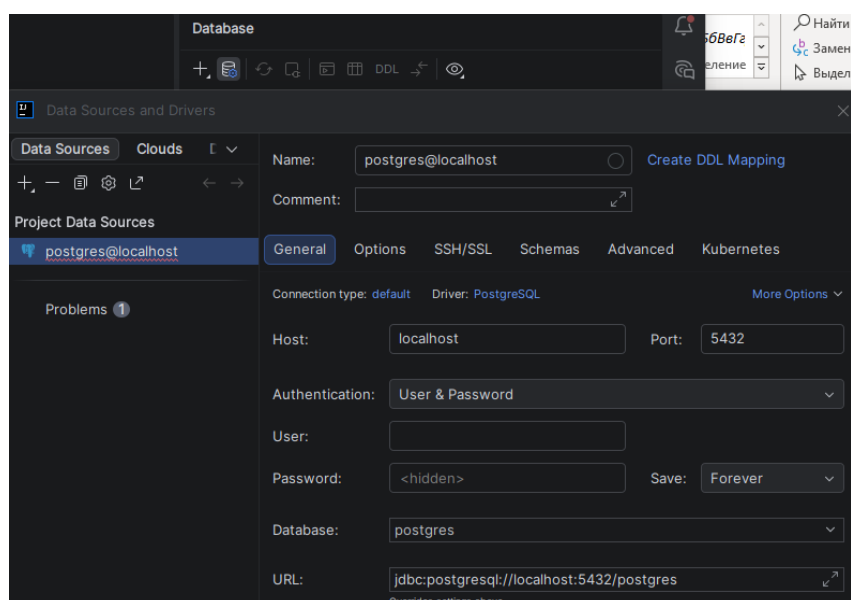


Рисунок 7 – Инструменты для работы с базой данных в IntelliJ IDEA

2.4 Выбор базы данных

В качестве базы данных выбрана PostgreSQL. PostgreSQL является одной из наиболее развитых и надежных реляционных систем управления базами данных.

Одним из ключевых преимуществ является поддержка транзакций и механизмов обеспечения целостности данных. В разрабатываемом приложении база данных будет хранить информацию о пользователях, автомобилях, автосервисах, записях на обслуживание и истории ремонта. Все эти данные связаны между собой, поэтому важно обеспечить корректность операций добавления, изменения и удаления информации. PostgreSQL реализует механизм транзакций, позволяющий гарантировать сохранение целостности данных даже при возникновении ошибок или сбоев.

Важным фактором является и хорошая интеграция PostgreSQL с серверными приложениями, разработанными с использованием фреймворка Spring Boot.

					ПГУ1.090301.45.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

3 Архитектура приложения

Для обеспечения масштабируемости, облегчения тестирования и поддержки приложения необходимо выбрать архитектурный паттерн.

3.1 Архитектура клиентской части

Чистая архитектура – это набор принципов проектирования программного обеспечения, который описан в книге Роберта К. Мартина. Этот подход к проектированию ПО заключается в разделении кода на слои, каждый из которых отвечает за свою задачу. Роберт Мартин выделяет 4 слоя:

- сущности;
- варианты использования;
- адаптеры интерфейсов;
- фреймворки и драйверы.

Сущности являются бизнес-объектами, которые содержат в себе общие правила работы приложения.

Слой вариантов использования содержит конкретные правила, реализует бизнес-логику, управляет потоком данных от сущностей и к сущностям, то есть позволяет сущностям взаимодействовать друг с другом.

Адаптеры интерфейсов служат для преобразования данных из формата удобного для использования сущностями и вариантами использования в формат удобный для внешних слоев, например, базы данных. Аналогично данные преобразуются обратно.

Фреймворки и драйверы являются внешним слоем, содержащим инструменты, такие как база данных, пользовательский интерфейс, веб-фреймворки.

Такая архитектура позволяет отделить бизнес-логику от пользовательского интерфейса, добиться независимости от используемых инструментов и возможности изменить их при необходимости, облегчить тестируемость,

					ПГУ1.090301.45.001 ПЗ	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

поддержку и масштабируемость приложения.

Клиентское приложение состоит из нескольких слоев:

- data layer;
- domain layer;
- presentation layer;

Слой данных (data layer) отвечает за взаимодействие с сервером и доступом к данным, полученным от сервера. Он содержит интерфейсы и классы, которые используются для получения отправки запросов и получения данных от сервера.

Слой домена (domain layer) отвечает за бизнес-логику. На этом уровне содержатся классы, которые отвечают за конкретные функции приложения, интерфейсы для доступа к данным, а также структуры данных, используемые в бизнес-логике.

Слой представлений (presentation layer) отвечает за пользовательский интерфейс. Его задачами являются отображение информации пользователю, получение ввода от пользователя, передача данных в бизнес-логику.

3.2 Архитектура серверной части

Для разработки серверной части была выбрана многослойная архитектура. Приложение делится на три слоя, каждый из которых отвечает за свою задачу:

- контроллер. Контроллер принимает входящие HTTP-запросы, вызывает сервис, отвечающий за необходимую функцию, и отправляет HTTP-ответ клиенту;
- сервис. Сервисы содержат бизнес-логику приложения, управляет работой репозитория;
- репозиторий. Репозитории отвечают за доступ к данным, с помощью них осуществляется работа с базой данных. Многослойная архитектура осуществляет принцип разделения ответственности, позволяет легко тестировать приложение, обеспечивает масштабируемость.

4 Реализация приложения

4.1 Описание функционала

В приложении предусмотрена регистрация и вход для идентификации пользователя, а также ролевая модель, содержащая роли клиента и мастера. В зависимости от роли меняется предоставляемый функционал.

Клиенту после авторизации доступны такие возможности как:

- поиск мастера;
- просмотр карточки мастера;
- просмотр отзывов;
- добавление отзыва после выполненной услуги;
- запись на СТО;
- отмена записи;
- просмотр истории записей;
- добавление информации об автомобилях;
- просмотр и редактирование профиля.

Мастеру после выполненной авторизации предоставляется следующий функционал приложения:

- просмотр записей к себе;
- изменение статуса записи;
- просмотр и редактирование личного профиля и профиля мастера.

При запуске приложения происходит проверка был ли до этого авторизован пользователь. Если авторизация не выполнялась, то пользователю отображается экран авторизации и предлагается войти в приложение или зарегистрировать новый аккаунт. Варианты использования представлены на рисунке 12.

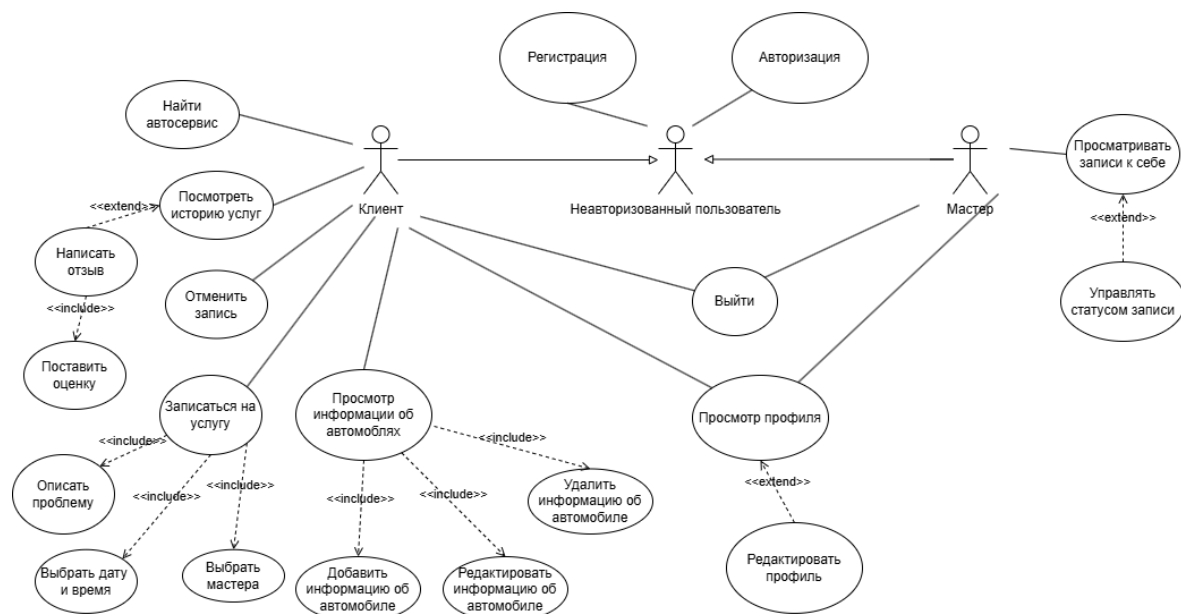


Рисунок 12 – UML диаграмма вариантов использования

4.2 Описание работы приложения

Конечный пользователь выполняет все необходимые операции в визуальном интерфейсе мобильного приложения. Для доступа к полному функционалу приложения необходимо войти в аккаунт или зарегистрировать новый. После ввода пользовательских данных они отправляются на сервер для проверки на существование или записи в базу данных в случае регистрации. В случае успешной авторизации пользователю предоставляется полный функционал приложения в зависимости от его роли. Диаграмма деятельности, отображающая функции регистрации и авторизации, изображена на рисунке 13.

Пользователь использует функции приложения, инициируя запросы к серверу для получения или отправки необходимых данных. Для взаимодействия клиента и сервера используется REST API – архитектурный подход взаимодействия через протокол HTTP. Для отправки данных используются HTTP-запросы GET, POST, PUT, DELETE в формате JSON.

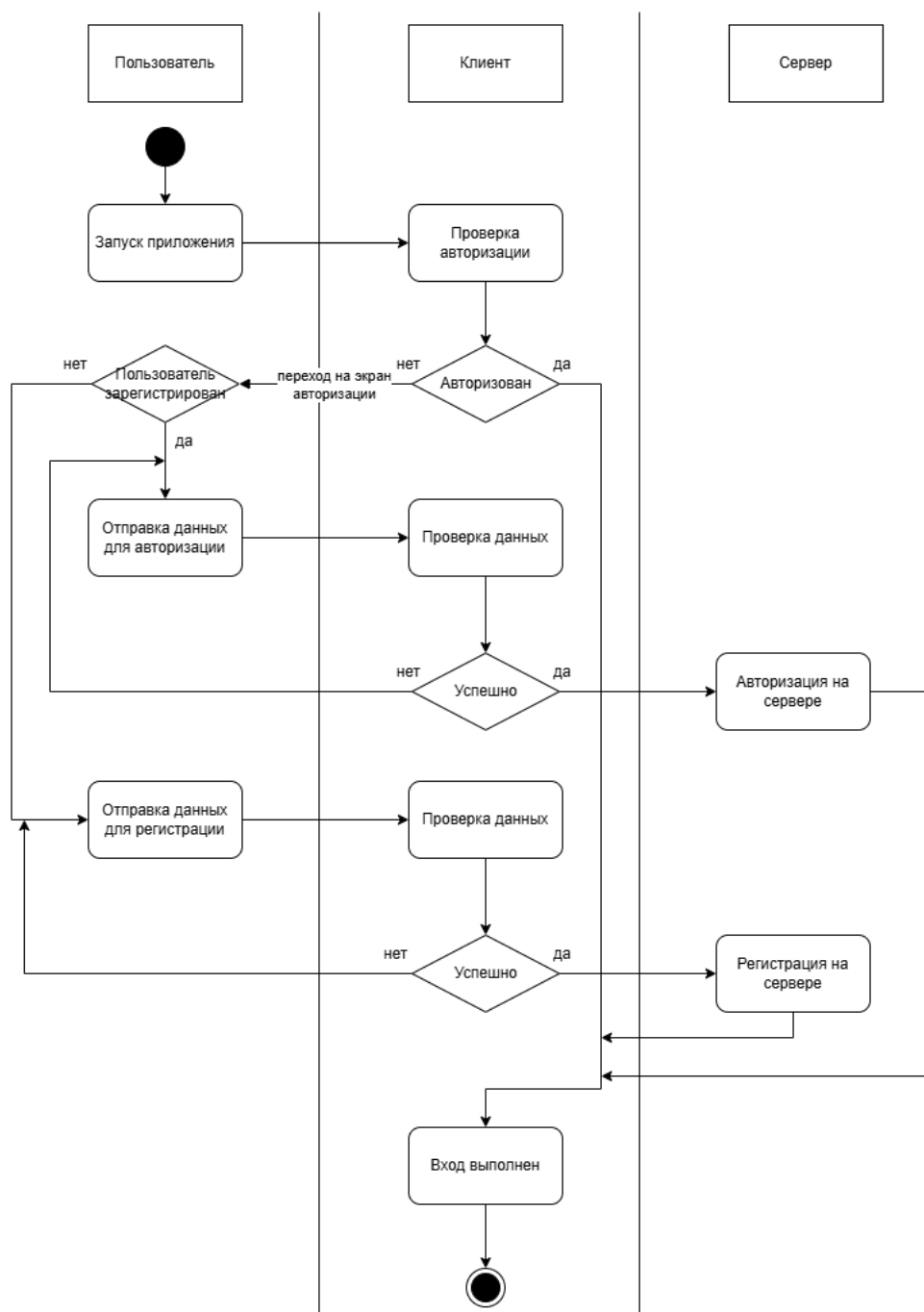


Рисунок 13 – UML диаграмма деятельности для регистрации и авторизации

Для просмотра списка мастеров клиент отправляет GET-запрос на сервер, сервер обрабатывает запрос, извлекает данные из базы данных и отправляет ответ на клиент, клиент в свою очередь обрабатывает ответ и отображает список на экране. Аналогичным образом происходит просмотр списка машин пользователя и истории записей в автосервис и получение данных пользователя в профиле.

Для создания записи в автосервис пользователь вводит данные в форме

Изм.	Лист	№ докум.	Подпись	Дата

необходимые для записи и клиент отправляет POST-запрос на сервер, сервер заносит данные в базу данных и отправляет ответ клиенту. Подобным образом выполняются функции добавления автомобилей в гараж. Взаимодействие пользователей с записями изображено на рисунках 14 и 15.

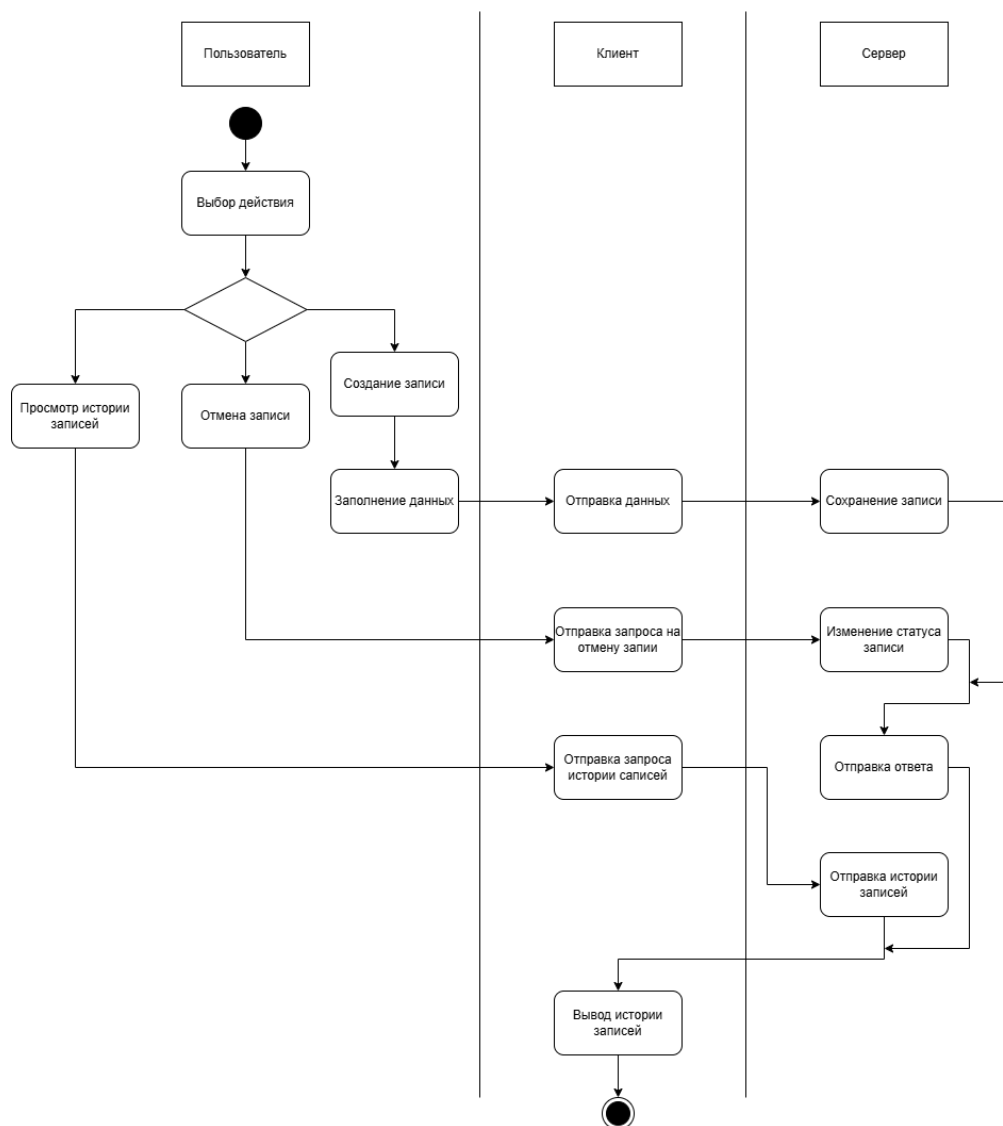


Рисунок 14 – UML диаграмма деятельности для взаимодействия пользователя с ролью CLIENT с записью в автосервис

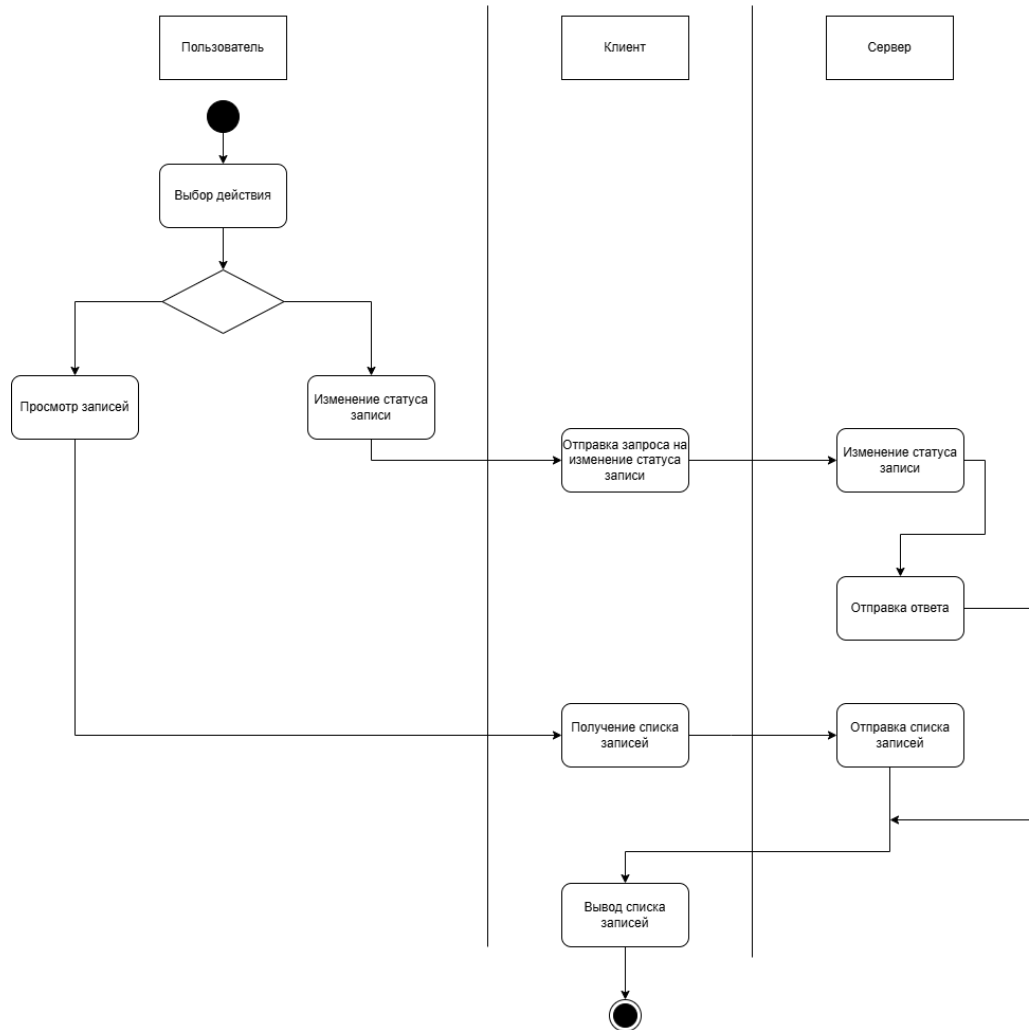


Рисунок 15 – UML диаграмма деятельности для взаимодействия пользователя с ролью MASTER с записью в автосервис

Отмена записи и изменение статуса записи происходит с помощью PUT-запроса на сервер с идентификатором записи и статусом, на который будет изменен текущий статус. Сервер обрабатывает данные, изменяет их и отправляет ответ клиенту. Для изменения данных профиля, изменения мастером своего графика работы пользователь должен на странице профиля ввести новые данные и инициировать запрос на изменение на сервер, который занесет изменения в базу данных. Удаление данных выполняется с помощью DELETE-запроса.

Выполняемые пользователем, клиентом и сервером действия изображены на диаграмме последовательности на рисунке 16.

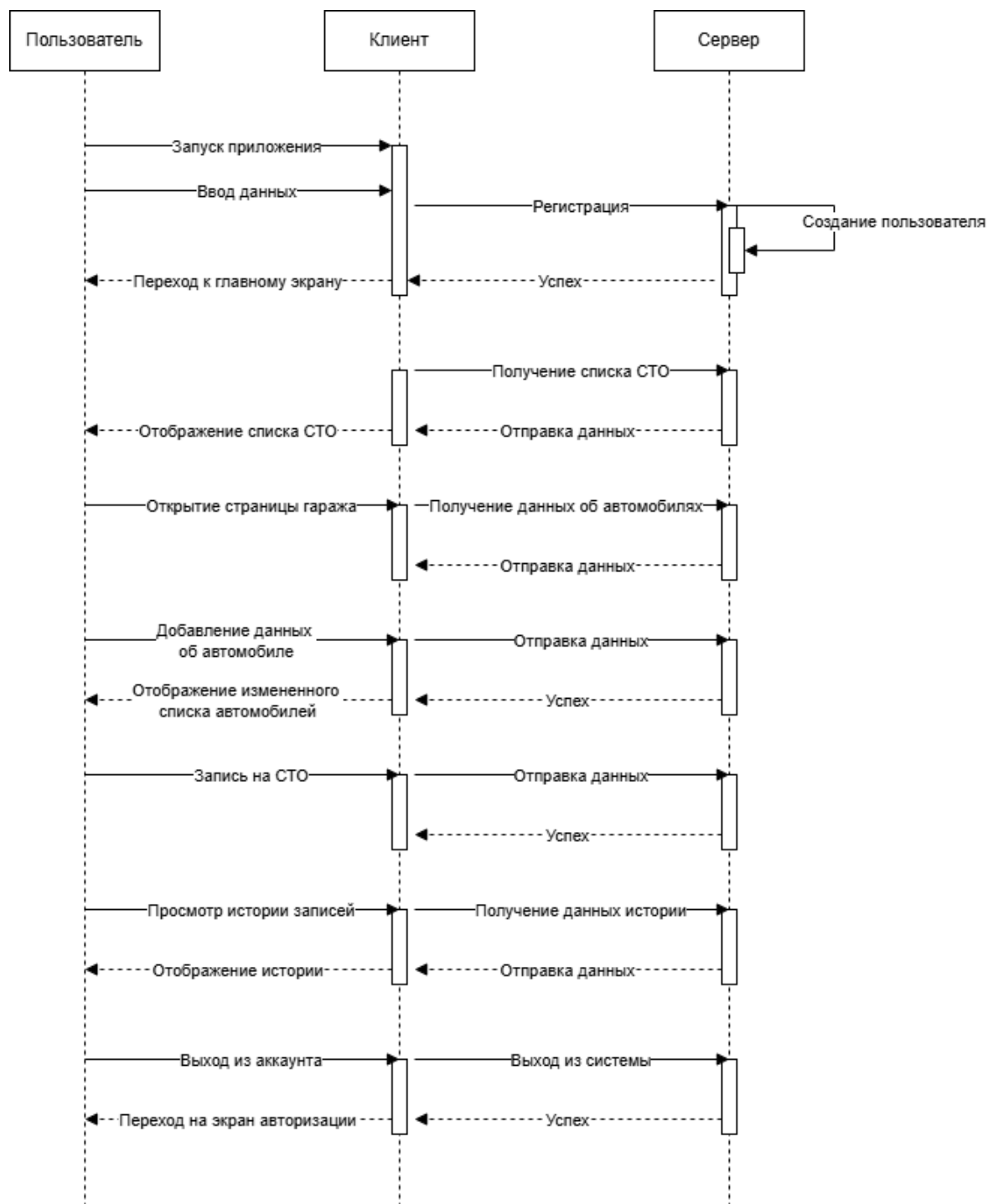


Рисунок 16 – UML диаграмма последовательности

4.3 Реализация клиентской части

Для рассмотрения реализации приложения возьмем в пример функцию регистрации пользователя.

4.3.1 Presentation layer

На уровне представления содержатся классы, отвечающие за пользовательский интерфейс. RegisterActivity отвечает за отображение интерфейса и хранит в себе экземпляр класса SignUpViewModel.

SignUpViewModel хранит состояние интерфейса с помощью класса AuthState, который описывает состояния экрана регистрации: Idle (ожидание), Loading (запрос выполняется), Success (успех) и Error (ошибка с текстом причины).

Листинг 1. Класс AuthState:

```
sealed class AuthState {  
    object Idle: AuthState()  
    object Loading: AuthState()  
    data class Success(val role: String) : AuthState()  
    data class Error(val message: String) : AuthState()  
}
```

Класс SignUpViewModel также выполняет вызов сценария регистрации. Функция register запускает асинхронный корутин. Он переводит экран в состояние Loading, вызывает сценарий SignUpUseCase и с помощью метода fold обрабатывает результат операции. В случае успеха обращаются выполняются сценарии saveTokenUseCase и saveRoleTokenUseCase для сохранения токена и роли.

Листинг 2. Функция register класса SignUpViewModel:

```
fun register(request: SignUpRequest){  
    viewModelScope.launch{  
        _state.value = AuthState.Loading  
  
        signUpUseCase(request).fold(  
            onSuccess = {  
                result ->  
                saveTokenUseCase(result.token)  
                saveRoleUseCase(result.role.name)  
                _state.value=  
AuthState.Success(result.role.name)  
            },  
            onFailure = {  
                exception ->  
                _state.value=  
AuthState.Error(exception.message ?: "Ошибка регистрации")  
            })  
    }  
}
```

Обработка состояния происходит асинхронно в фоновом потоке с использованием корутин и потока состояний StateFlow. RegisterActivity находится в постоянном наблюдении за потоком состояний. Функция renderState отвечает за реактивное изменение интерфейса в зависимости от текущего статуса операции. При Loading блокирует кнопку, предотвращая повторные запросы. При Success сохраняет сессию и осуществляет ветвление навигации: пользователи с ролью MASTER направляются на экран заполнения профиля мастера, а пользователи с ролью CLIENT — на главный экран. При Error выводит сообщение об ошибке на экран. Функция observeViewModel запускает корутин, привязанный к жизненному циклу экрана, и с помощью метода repeatOnLifecycle собирает изменения состояний экрана.

Листинг 2. Класс RegisterActivity:

```
private fun renderState(state: AuthState) {
    when(state) {
        is AuthState.Idle -> {
            binding.regButton.isEnabled = true
        }
        is AuthState.Loading -> {
            binding.regButton.isEnabled = false
        }
        is AuthState.Success -> {
            Toast.makeText(this, "Регистрация успешна!",
                Toast.LENGTH_SHORT).show()
            if (role == UserRole.MASTER) {
                val intent = Intent(this,
                    MasterProfileRegisterActivity::class.java)
                intent.flags =
                    Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                startActivity(intent)
                finish()
            } else {
                val intent = Intent(this,
                    ClientActivity::class.java)
                intent.flags =
                    Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                startActivity(intent)
                finish()
            }
        }
        is AuthState.Error -> {
            binding.regButton.isEnabled = true
        }
    }
}
```

```

        Toast.makeText(this, state.message,
Toast.LENGTH_LONG).show()
    }
}
}
private fun observeViewModel() {
    lifecycleScope.launch {
        repeatOnLifecycle(Lifecycle.State.STARTED) {
            viewModel.state.collect {
                state-> renderState(state)
            }
        }
    }
}
}

```

Как только пользовательские данные обработаются с успехом произойдет переход на главный экран приложения.

4.3.2 Domain layer

Уровень домена выступает связующим звеном между слоем представления и слоем данных. Для этого на данном уровне содержится интерфейс репозитория `AuthRepository`, он определяет набор функций для авторизации и регистрации пользователя.

Класс `SignUpUseCase` реализует функцию регистрации, которую выполняет приложение на уровне представления. Она принимает в качестве параметра объект `SignUpRequest`, содержащий данные пользователя для регистрации, проверяет корректность введенных данных, проверяет поля на заполненность с помощью условий.

Листинг 3. Класс `SignUpUseCase`:

```

class SignUpUseCase @Inject constructor(private val repository:
AuthRepository) {
    suspend operator fun invoke(request: SignUpRequest):
Result<AuthResult>{
        if(request.password.isBlank() ||
request.email.isBlank() || request.surname.isBlank() ||
request.name.isBlank()){
            return
Result.failure(IllegalArgumentException("Поля не должны быть
пусты"))
        }else
        if(!Patterns.EMAIL_ADDRESS.matcher(request.email).matches()){
            return
Result.failure(IllegalArgumentException("Неверный формат почты"))
        }
    }
}

```

```

        } else if(request.password.length<8){
            return
Result.failure(IllegalArgumentException("Пароль должен быть не менее
8 символов"))
        }
        return repository.register(request)
    }
}

```

После проверки SignUpUseCase вызывает функцию register() интерфейса AuthRepository и передает данные пользователя на уровень данных.

Листинг 4. Интерфейс AuthRepository:

```

interface AuthRepository {
    suspend fun register(request: SignUpRequest):
Result<AuthResult>
    suspend fun signIn(request: SignInRequest):
Result<AuthResult>
}

```

4.3.3 Data layer

В этом слое происходит взаимодействие с сервером. Интерфейс AuthApi содержит функции для отправки HTTP-запросов регистрации и входа в приложение на сервер.

Интерфейс AuthApi содержит сетевые запросы, которые используются для авторизации и регистрации пользователя. Для выполнения сетевых запросов используется библиотека Retrofit. Для регистрации используется метод register(), который принимает в качестве параметра объект SignUpRequest.

Листинг 5. Интерфейс AuthApi:

```

interface AuthApi {
    @POST("/api/auth/register")
    suspend fun register(@Body request: SignUpRequest):
Response<AuthResult>

    @POST("/api/auth/login")
    suspend fun signIn(@Body request: SignInRequest):
Response<AuthResult>
}

```

Класс AuthRepositoryImpl содержит реализацию определенного на доменном уровне интерфейса AuthRepository, в котором представлены функции регистрации и входа. AuthRepositoryImpl использует функции интерфейса

AuthApi для отправки запросов. Для выполнения регистрации используется метод register(), в котором выполняется сетевой запрос. В случае успешного выполнения запроса будут получены данные авторизации пользователя, включающие в себя токен и роль пользователя, а в случае ошибки будет перехвачено исключение.

Листинг 6. Функция register класса AuthRepositoryImpl:

```
override suspend fun register(request: SignUpRequest):
Result<AuthResult>{
    return try {
        val response = api.register(request)
        if (response.isSuccessful && response.body() !=
null) {
            Result.success(response.body()!!)
        }else Result.failure(Exception("Пользователь с
таким email уже существует"))
        }catch(e: Exception){
            Result.failure(Exception("Нет сети"))
        }
    }
}
```

На рисунке 17 представлена диаграмма классов, которые отвечают за регистрацию пользователя.

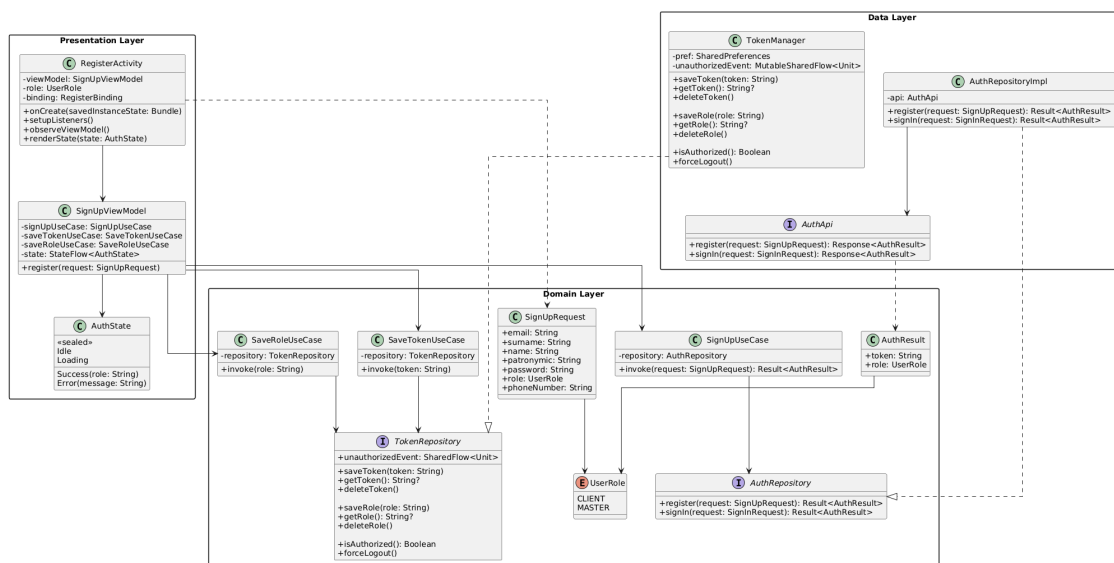


Рисунок 17 – UML диаграмма классов клиента для регистрации

4.4 Реализация серверной части

Рассмотрим реализацию регистрации на сервере.

Безопасность системы базируется на концепции stateless (архитектура без сохранения состояния сессии на сервере), где подлинность каждого запроса валидируется с помощью криптографических токенов.

JwtService отвечает за криптографические операции над строками JWT. Функция generateToken формирует полезную нагрузку токена, куда уникальный идентификатор и роль пользователя. Это позволяет мобильному клиенту и серверу считывать роль пользователя без повторных запросов к базе данных.

Функция buildToken осуществляет сборку итоговой строки: подставляет субъект (email), время создания, время деградации (expiration) и подписывает токен криптографическим секретным ключом. Функции извлечения данных extractEmail, extractRole, extractId выполняют обратный разбор криптографической строки, верифицируя подпись сервера.

Слой Controller осуществляет получение HTTP-запросов от клиентской части приложения и отправку ответа. За получение запроса на авторизацию и отправку ответа отвечает класс AuthController, который содержит функцию signUp, она получает запрос на регистрацию, в теле которого передается информация пользователя, и вызывает функцию signUp из класса AuthService, которая содержит бизнес-логику и обрабатывает данные из запроса. После получения результата обработки клиенту возвращается ответ в виде объекта AuthResponse.

Листинг 6. Класс AuthController:

```
@RestController
@RequestMapping("/api/auth")
class AuthController(private val authService: AuthService) {
    @PostMapping("/register")
    fun signUp(@RequestBody request: SignUpRequest):
    ResponseEntity<AuthResponse> {
        val response = authService.signUp(request)
        return ResponseEntity.ok(response)
    }
    @PostMapping("/login")
    fun signIn(@RequestBody request: SignInRequest):
    ResponseEntity<AuthResponse>{
        val response = authService.signIn(request)
        return ResponseEntity.ok(response)
    }
}
```

```

    }
}

```

Слой Service содержит бизнес-логику серверной части приложения, он отвечает за обработку данных, взаимодействие с репозиториями.

Функции регистрации и авторизации содержится в классе AuthService. Функция signUp класса AuthService реализует функцию регистрации пользователя.

Листинг 7. Функция signUp класса AuthService:

```

fun signUp(request: SignUpRequest): AuthResponse{
    if(userRepository.existsByEmail(request.email)){
        throw IllegalArgumentException("Пользователь с
таким логином уже существует")
    }
    Val hashedPassword=
passwordEncoder.encode(request.password)
    val newUser = User(
        email = request.email,
        surname = request.surname,
        name = request.name,
        patronymic = request.patronymic,
        passwordHash = hashedPassword,
        role = request.role,
        phoneNumber = request.phoneNumber
    )
    userRepository.save(newUser)
    val token = jwtService.generateToken(newUser.id!!,
newUser.email, newUser.role.name)
    return AuthResponse(token, newUser.role.name)
}

```

Реализованная функция осуществляет поиск в БД на предмет наличия пользователя со входным логином. При отсутствии совпадений выполняется безопасное преобразование пароля через PasswordEncoder и создание нового объекта типа User. После записи сущности в хранилище данных генерируется веб-токен (JWT), включающий в себя ID и роль созданного профиля. Завершая свою работу, функция отправляет ответ в виде структуры AuthResponse, куда упаковываются сформированный ключ доступа и информация о роли.

Слой repository осуществляет работу с базой данных. Структура базы данных, с которой осуществляется работа представлена на рисунке 18.

Для сохранения пользователя в базу данных используется интерфейс

					ПГУ1.090301.45.001 ПЗ	Лист
						37
Изм.	Лист	№ докум.	Подпись	Дата		

UserRepository, который наследуется от JpaRepository<User, Long>, поэтому методы сохранения, обновления, удаления и добавления пользователей реализуются автоматически. В интерфейсе UserRepository также определены методы existsByEmail и findByEmail, которые отвечают за проверку на существование и поиск по электронной почте соответственно.

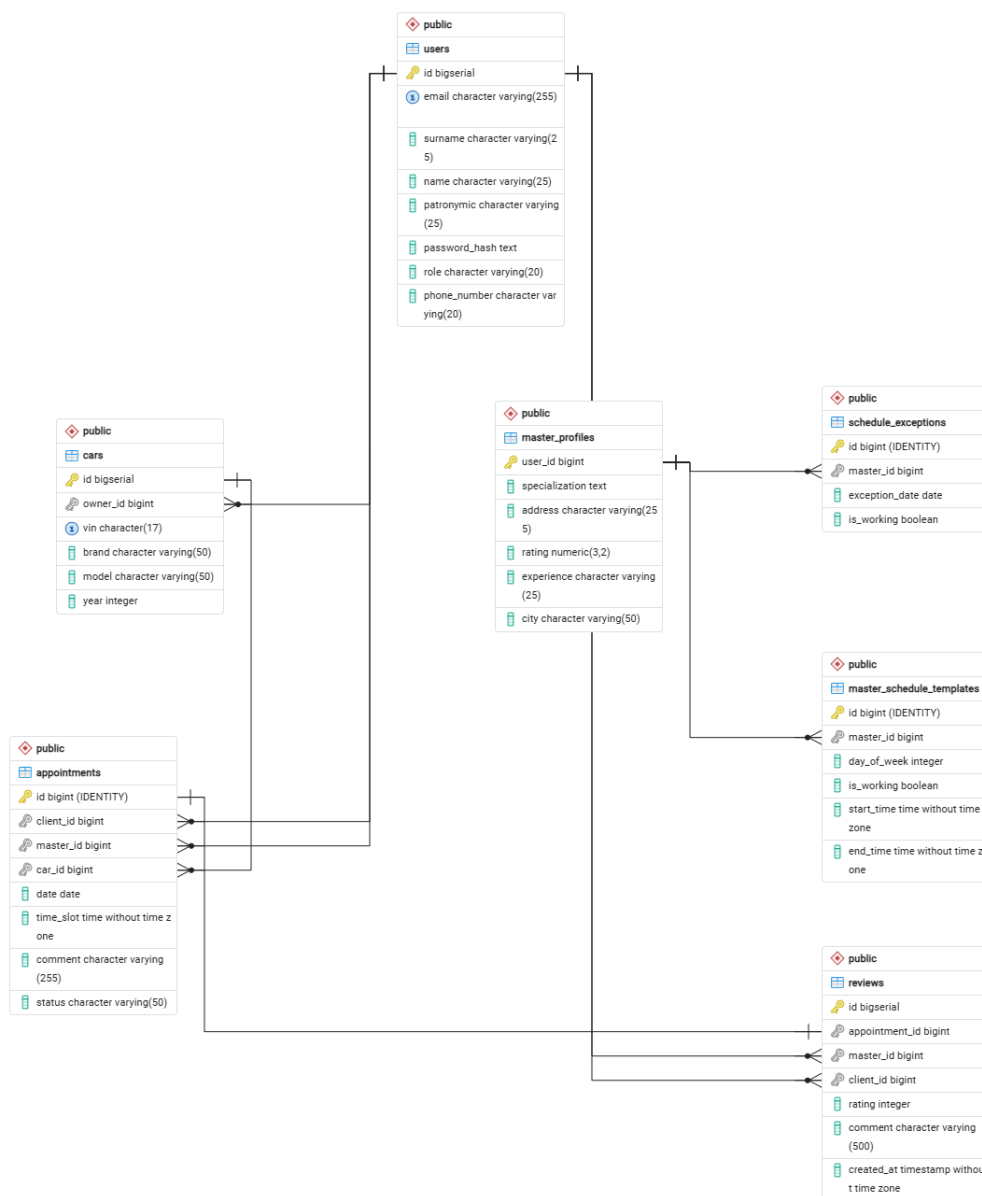


Рисунок 18 – ER-диаграмма базы данных

Листинг 8. Функция signUp класса AuthService:

```
@Repository
interface UserRepository: JpaRepository<User, Long> {
```

```

    fun existsByEmail(email: String): Boolean
    fun findByEmail(email: String): User?
}

```

User класс-сущность, отображающий структуру таблицы users базы данных. С помощью аннотаций @Id, @Column, @Enumerated задаются правила валидации на уровне СУБД: уникальность email-адреса, автоматическая генерация идентификаторов и сохранение перечислений ролей (UserRole) в виде строковых значений.

На рисунке 19 представлена диаграмма классов на стороне сервера, которые отвечают за регистрацию пользователя.

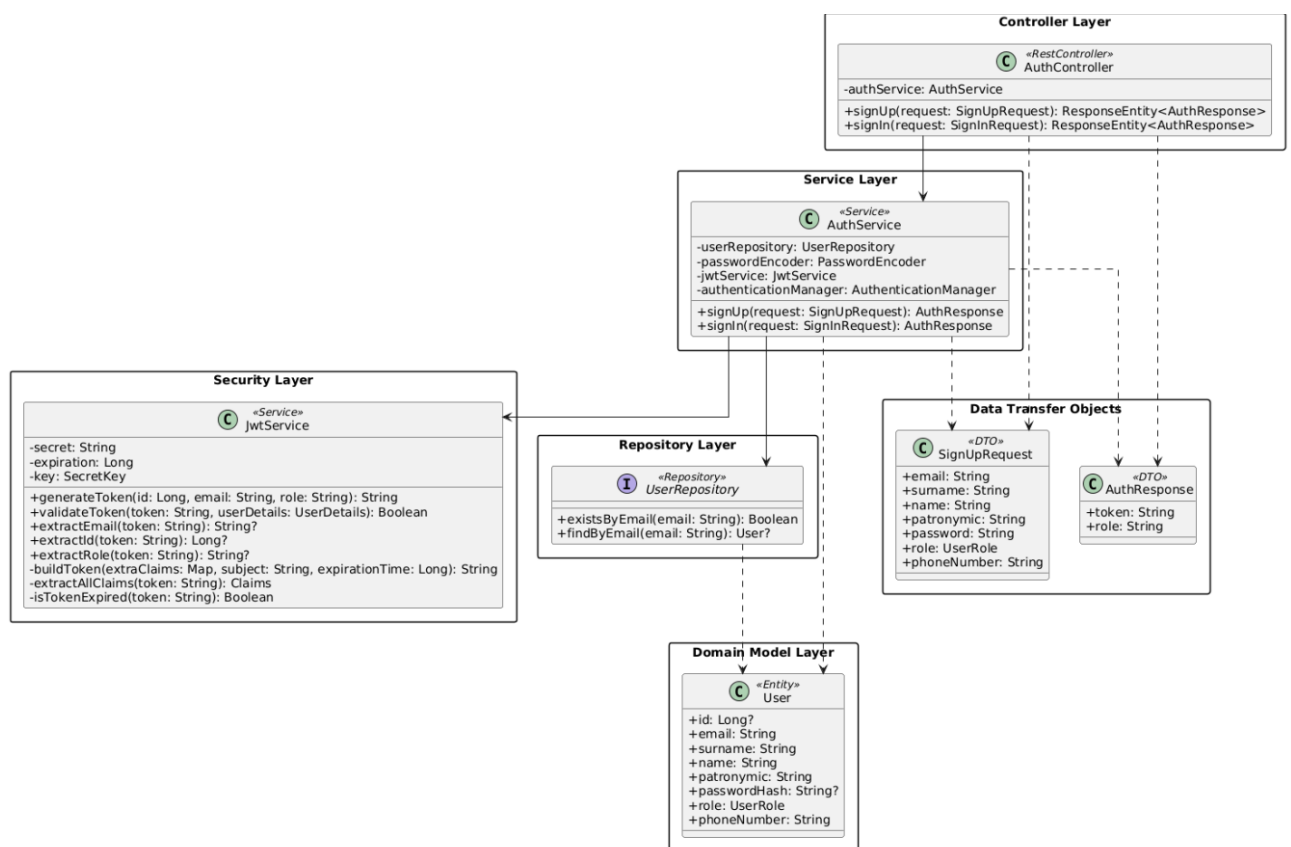


Рисунок 19 – UML диаграмма классов сервера для регистрации

5 Тестирование приложения

Тестирование приложения является важным этапом разработки ПО, позволяет проверить правильность, надежность работы приложения, выявить ошибки, возникающие в процессе его использования, а также оценить удобство продукта.

Протестируем несколько функций:

- регистрация;
- авторизация;
- добавление и удаление автомобиля;
- запись к мастеру;
- редактирование профиля;
- редактирование графика работы мастера;
- изменение статуса записи
- добавление оценки и отзыва.

5.1 Регистрация

При первом запуске приложения пользователю необходимо зарегистрировать аккаунт. На странице авторизации, показанной на рисунке 20, если у пользователя нет аккаунта, он может выбрать зарегистрироваться в качестве клиента или в качестве мастера, в зависимости от выбора будет присвоена роль.

					ПГУ1.090301.45.001 ПЗ	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

Рисунок 20 – Экран авторизации

После выбора отрывается страница регистрации, где пользователю необходимо ввести уникальный email, фамилию, имя, отчество, пароль и номер телефона. На рисунке 21 представлен вывод ошибки на экране регистрации при слишком коротком пароле.

Рисунок 21 – Ошибка регистрации

При успешной регистрации пользователь с ролью клиента перейдет на главный экран приложения, а пользователю с ролью мастера будет необходимо заполнить профиль мастера: имя или название СТО, опыт работы, специализация, город и адрес. Экран регистрации профиля мастера показан на рисунке 22

Рисунок 22 – Регистрация профиль мастера

5.2 Авторизация

Для того чтобы войти в аккаунт пользователю необходимо ввести свои данные и нажать на кнопку «Войти». В случае некорректных данных на экране появляется сообщение об этом, пример такого сообщения показан на рисунке 23.

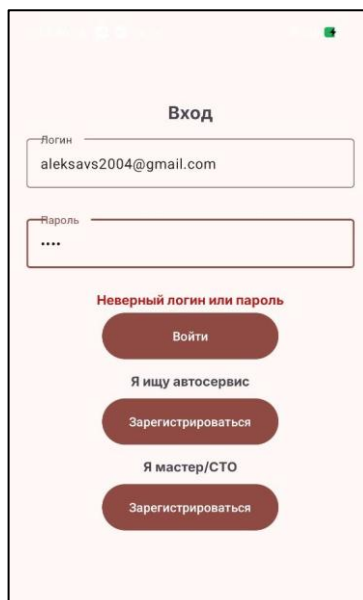


Рисунок 23 – Ошибка входа

После успешной авторизации пользователь попадает на главный экран приложения, который иллюстрирует рисунок 24.

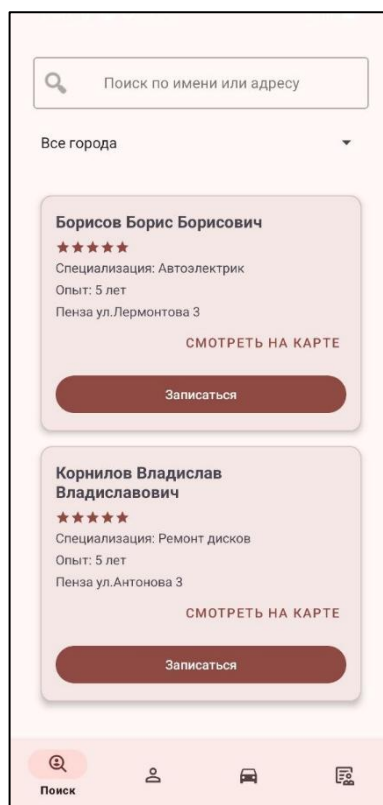


Рисунок 24 – Главный экран приложения

Процесс тестирования авторизации представлен в таблице 2.

Таблица 2 – Тестирование авторизации

№	Название	Шаги тестирования	Ожидаемый результат	Фактический результат	Статус прохождения
1	Ошибка: пустые поля	1. Оставить поля пустыми 2. Нажать кнопку «Войти»	Вывод ошибки «Поля не могут быть пустыми»	Ошибка выведена, вход не выполнен	+
2	Ошибка: неверный логин и пароль	1. Ввести некорректные данные 2. Нажать кнопку «Войти»	Вывод ошибки «Неверный логин или пароль»	Ошибка выведена, вход не выполнен	+
3	Успешная авторизация	1. Ввести корректные данные 2. Нажать кнопку «Войти»	Вывод сообщения об успехе и переход на главный экран приложения	Вывод сообщения «Вход успешен», вход выполнен	+

5.3 Добавление автомобиля

Для добавления автомобиля на странице гаража пользователю необходимо нажать на кнопку добавления в углу экрана и снизу появится форма для заполнения данных. В случае если пользователь ввел некорректные данные или

оставил поля пустыми, запрос на добавления машины не отправится на сервер, а на экран выведется ошибка (рисунок 25).

Мой гараж

Kia Rio

Год: 2020 г.в.

VIN: WVWZZZ3CZPE000000

Добавить автомобиль

VIN

RARAFSDTDTDRA1213

Марка (например, Kia)

LADA

Модель (например, Rio)

GRANTA

Год выпуска (например, 2020)

2015

Сохранить

Рисунок 25 – Добавление автомобиля

В случае введения корректных данных, и успешного добавления автомобиля список автомобилей на странице гаража обновится, и появится карточка добавленного авто, как показано на рисунке 26.

Для удаления автомобиля пользователю необходимо нажать на иконку автомобиля, после чего появится диалоговое окно с подтверждением удаления, что показано на рисунке 27.

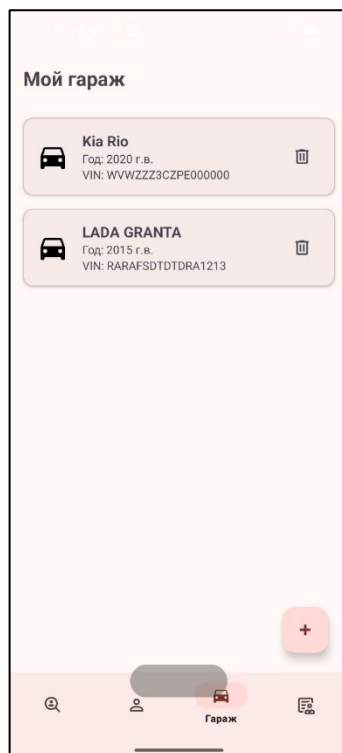


Рисунок 26 – Добавленный автомобиль

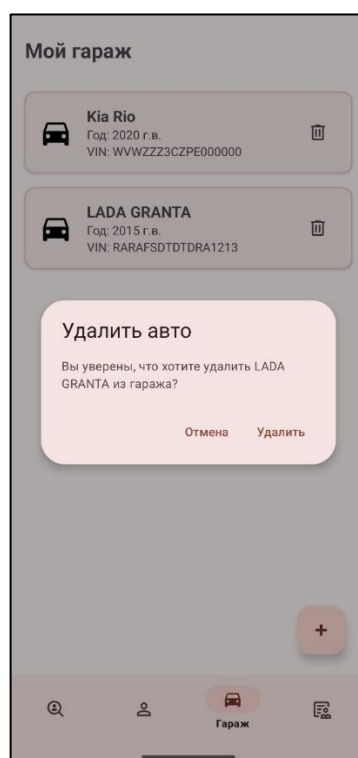


Рисунок 27 – Удаление автомобиля

Процесс тестирования добавления автомобиля представлен в таблице 3.

					ПГУ1.090301.45.001 ПЗ	Лист
						46
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 3 – Тестирование добавления автомобиля

№	Название	Шаги тестирования	Ожидаемый результат	Фактический результат	Статус прохождения
1	Ошибка: пустые поля	1. Оставить поля пустыми 2. Нажать кнопку «Сохранить»	Вывод ошибки «Заполните поля»	Ошибка выведена, данные не добавлены	+
2	Ошибка: некорректные данные	1. Ввести некорректные данные 2. Нажать кнопку «Сохранить»	Вывод ошибки «VIN номер должен содержать ровно 17 символов»	Ошибка выведена, данные не добавлены	+
3	Успешное добавление автомобиля	1. Ввести корректные данные 2. Нажать кнопку «Сохранить»	Вывод сообщения об успехе, обновление списка автомобиле й	Вывод сообщения «Машина добавлена», данные добавлены	+

5.4 Поиск и запись к мастеру

Для записи к мастеру пользователь должен на представленной на рисунке 28 странице поиска мастеров выбрать подходящего мастера. Кликнув на кнопку «Смотреть на карту», пользователь может посмотреть адрес мастера в приложении карт, установленном на телефоне.

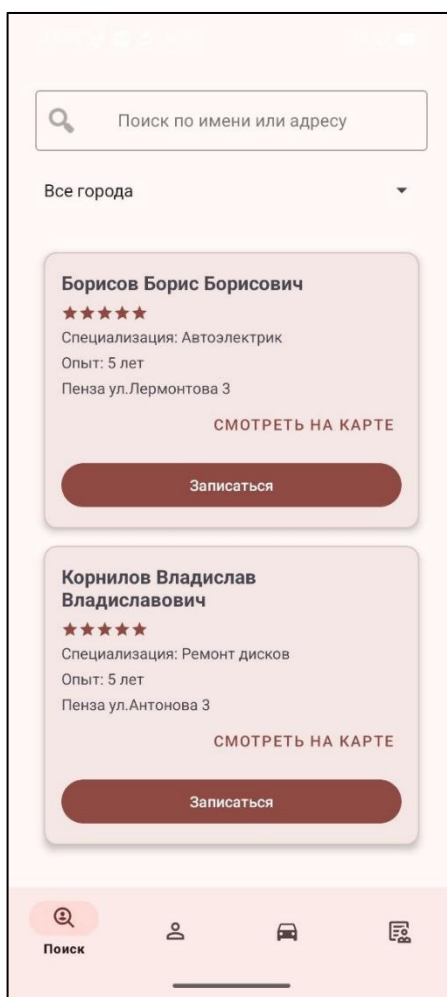


Рисунок 28 – Страница поиска мастеров

Осуществить поиск можно с помощью фильтрации по городу и поисковой строки, где пользователь вводит необходимую информацию. Поиск мастеров по фильтру и запросу в поисковой строке показан на рисунках 29 и 30.



Рисунок 29 – Поиск мастеров по фильтру

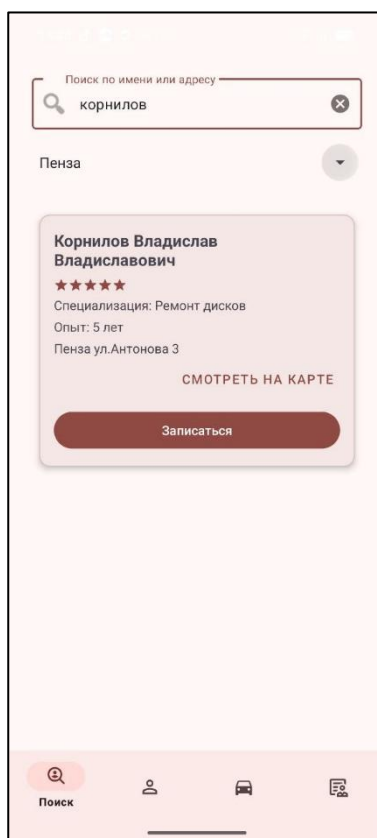


Рисунок 30 – Поиск мастеров с помощью поисковой строки

Чтобы перейти на страницу записи необходимо нажать на кнопку записаться на карточке мастера. После нажатия на кнопку пользователь переходит на страницу записи, которая показана на рисунке 31.

Рисунок 31 – Страница записи к мастеру

В случае введения корректных данных запись успешно выполнится (рисунок 32). В случае если поля пусты, кнопка записи не будет доступна.

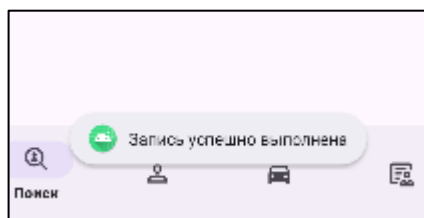


Рисунок 32 – Успешная запись к мастеру

Процесс тестирования представлен в таблице 4

Таблица 4 – Тестирование записи к мастеру

№	Название	Шаги тестирования	Ожидаемый результат	Фактический результат	Статус прохождения
1	Успешная запись	1. Нажать на кнопку записи 2. Ввести корректные данные 3. Нажать «Записаться»	Вывод сообщения об успешности записи	Выведено сообщение «Запись успешно выполнена»	+
2	Пустые поля	1. Нажать на кнопку записи 2. Оставить поля пустыми	Недоступность кнопки записи	Кнопка для выполнения записи недоступна	+

5.5 Редактирование профиля

В нижнем меню навигации пользователь с любой может открыть личный профиль, в котором отображается информация о пользователе: фамилия, имя, отчество, email и номер телефона, что показано на рисунке 33.

Мой профиль

Логин
vlad@ya.ru

Фамилия
Корнилов

Имя
Владислав

Отчество
Михайлович

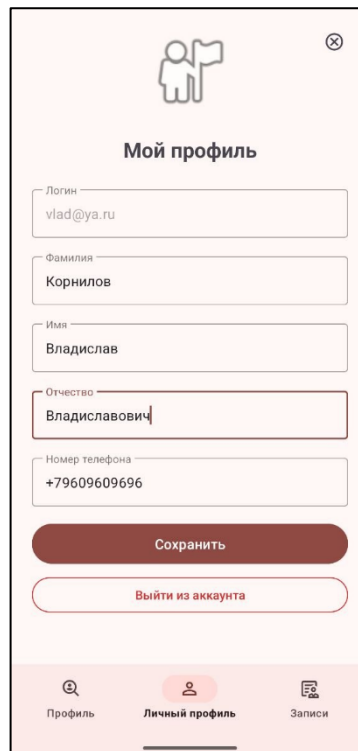
Номер телефона
+79609609696

Выйти из аккаунта

Профиль **Личный профиль** Записи

Рисунок 33 – Страница записи к мастеру

Для редактирования информации пользователю необходимо нажать на кнопку редактирования, ввести новые данные и нажать на кнопку сохранения, как показано на рисунке 34. При успешном сохранении данные на странице обновятся. Рисунок 35 иллюстрирует успешное обновление данных



Мой профиль

Логин
vlad@ya.ru

Фамилия
Корнилов

Имя
Владислав

Отчество
Владиславович

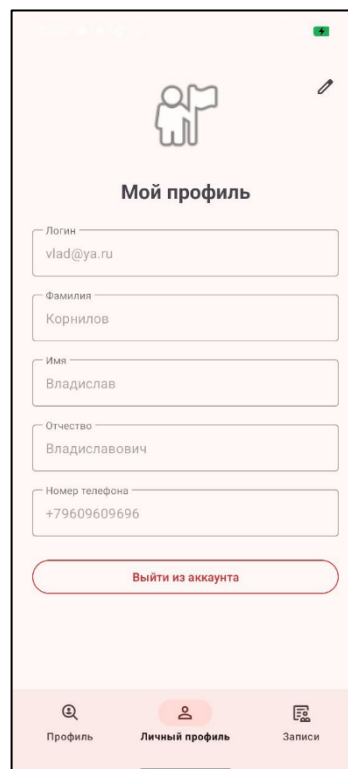
Номер телефона
+79609609696

Сохранить

Выйти из аккаунта

Профиль Личный профиль Записи

Рисунок 34 – Редактирование личного профиля



Мой профиль

Логин
vlad@ya.ru

Фамилия
Корнилов

Имя
Владислав

Отчество
Владиславович

Номер телефона
+79609609696

Выйти из аккаунта

Профиль Личный профиль Записи

Рисунок 35 – Обновленный личный профиль

Пользователь с ролью «MASTER» может открыть страницу профиля мастера, нажав на иконку в нижнем меню (рисунок 36). Так же, как и личный профиль может редактироваться профиль мастера.

Рисунок 36 – Страница профиля мастера

5.6 Редактирование графика работы мастера

На странице профиля мастера мастер может нажать на кнопку редактирования графика работы, после чего откроется страница редактирования графика со списком дней недели (рисунок 37).

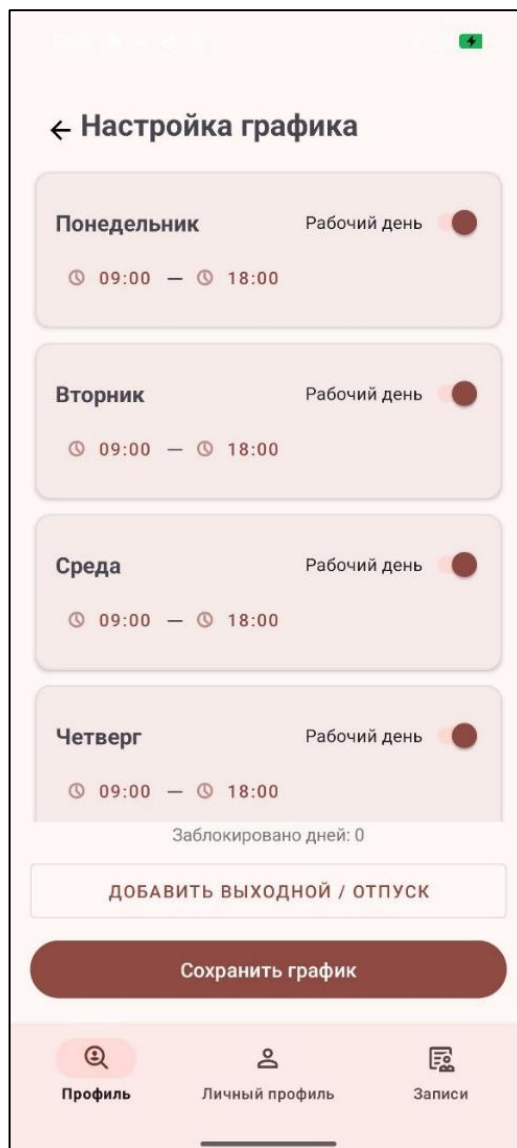


Рисунок 37 – Страница редактирования графика работы

На карточке дня недели есть переключатель, который делает день в расписании мастера рабочим или нерабочим в зависимости от изначального состояния. Также на карточке есть отображение времени работы, для редактирования которого необходимо нажать на время, после чего откроется окно выбора времени (рисунок 38).

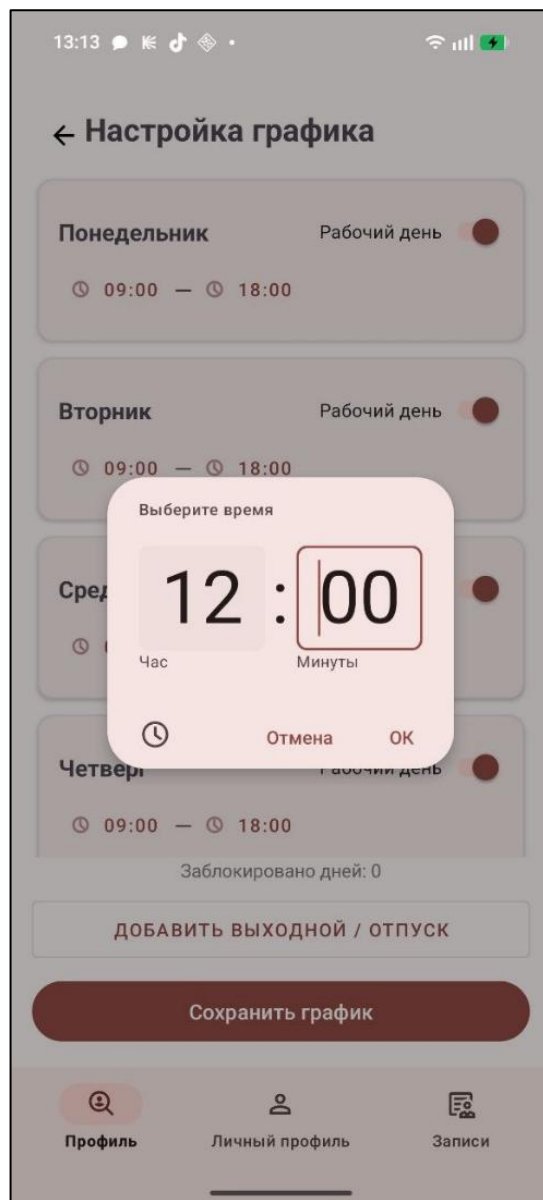


Рисунок 38 – Окно выбора времени начала работы

Пользователь может добавить один разовый нерабочий день в свое расписание, нажав на кнопку под списком. После нажатия на кнопку откроется календарь для выбора дня (рисунок 39).



Рисунок 39 – Окно выбора выходного дня

5.7 Изменение статуса записи

На странице истории заказов в списке текущих записей клиент может нажать на кнопку отмены, после чего откроется диалоговое окно подтверждения (рисунок 40). При подтверждении действия запись перемещается в список завершенных записей (рисунок 41).

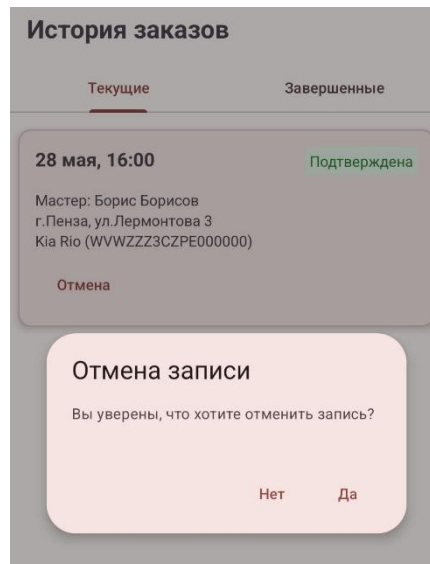


Рисунок 40 – Диалоговое окно подтверждения отмены записи

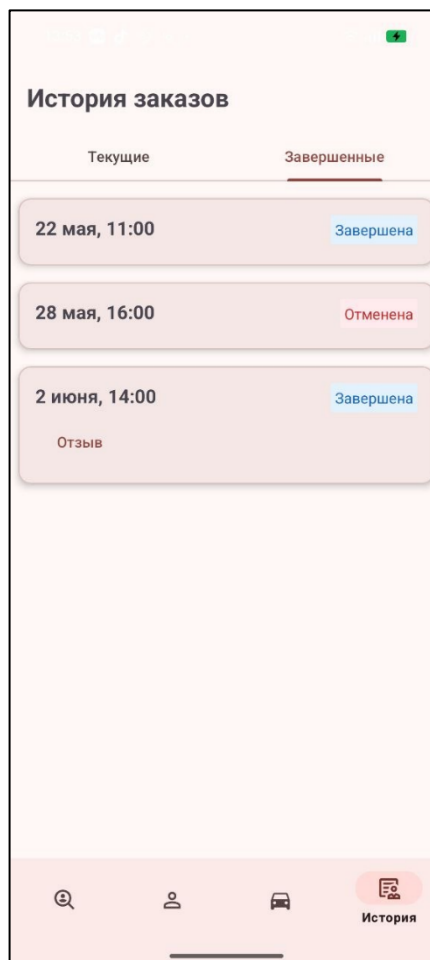


Рисунок 41 – Список завершенных заказов

На странице записей мастер может нажать на кнопку «Статус» конкретной активной записи, после чего откроется диалоговое окно со списком доступных статусов («В работе», «Завершить») (рисунок 42).

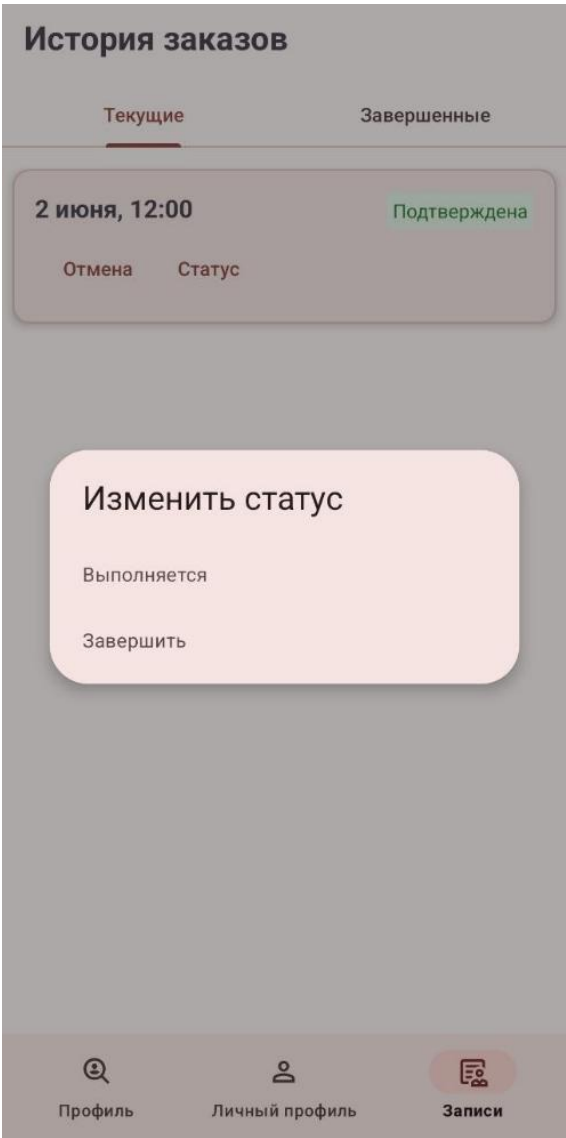


Рисунок 42 – Диалоговое окно изменения статуса записи

При выборе мастером соответствующего этапа и нажатии кнопки сохранения изменений система фиксирует новый статус в базе данных и отображает его на карточке, что иллюстрирует рисунок 43.

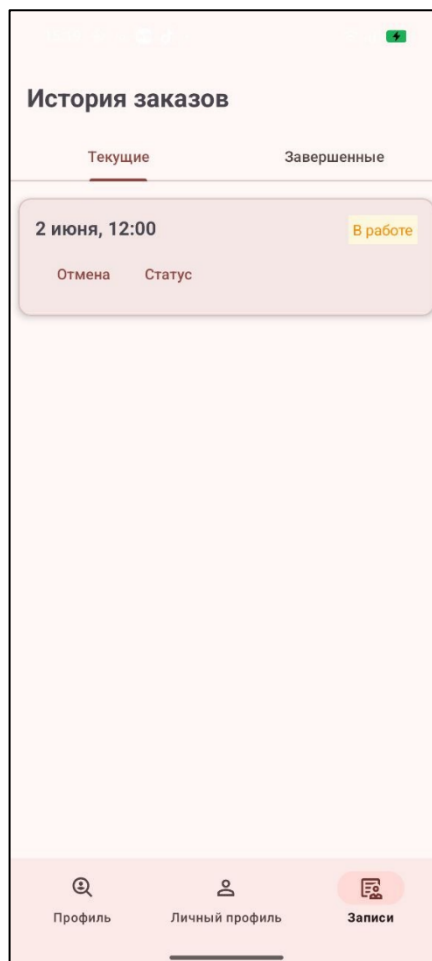


Рисунок 43 – Изменение статуса записи

5.8 Добавление оценки и отзыва

На странице истории заказов в списке завершенных записей клиент может оставить отзыв на запись со статусом «Выполнена», нажав на кнопку «Отзыв». После нажатия на кнопку открывается диалоговое окно с текстовым полем и шкалой оценивания. Пользователь должен поставить оценку от 1 до 5 и по желанию написать текст отзыва и отправить отзыв, нажав на кнопку отправки. В случае попытки отправки полностью пустого отзыва или отзыва без оценки выведется сообщение о необходимости поставить оценку (рисунок 44).

					ПГУ1.090301.45.001 ПЗ	Лист
						60
Изм.	Лист	№ докум.	Подпись	Дата		

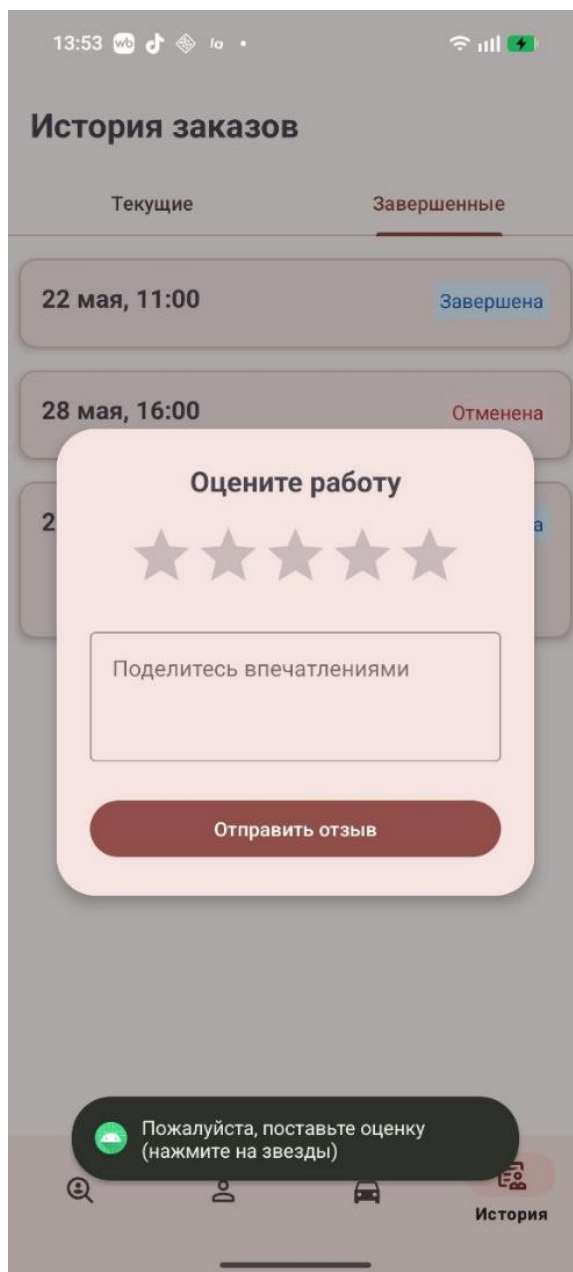


Рисунок 44 – Окно отзыва

Просмотр отзывов на услуги мастера осуществляется по клику на карточку мастера на странице поиска автосервисов, после которого открывается страница с информацией о мастере и списком отзывов (рисунок 45).

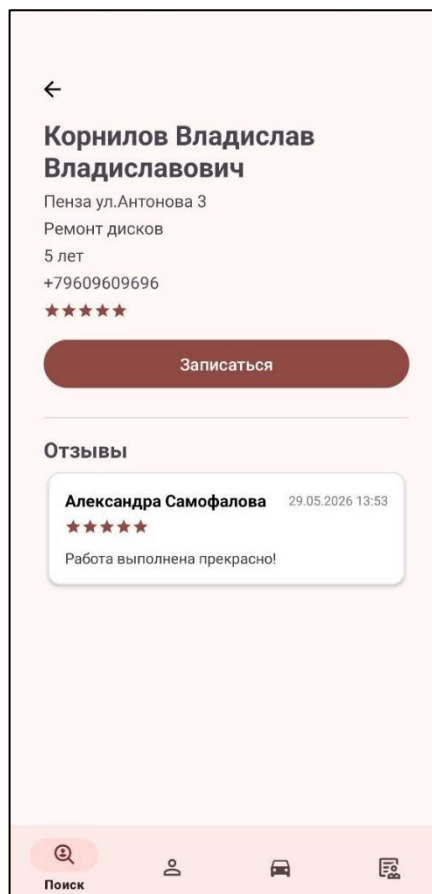


Рисунок 45 – Страница информации о мастере с отзывом

5.9 Вывод

В результате проведенного тестирования получилось выявить и исправить выявленные ошибки. Это позволило сделать более надежные программные модули серверной и клиентской частей, которые функционируют стабильно и полностью соответствуют предъявленным требованиям.

					ПГУ1.090301.45.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		62

6 Безопасность жизнедеятельности

В процессе проектирования программного обеспечения и последующего его использования человек проводит большое количество времени за компьютером. Длительное нахождение в статичной позе перед экраном ПК и непрерывное использование мобильных гаджетов оказывают негативное давление на организм человека. Для снижения негативного влияния необходимо соблюдать определенные правила при работе с ПК и мобильными устройствами.

В данном разделе будут рассмотрены правила безопасности при работе с ПК, мобильными устройствами и рекомендации по организации рабочего места.

6.1 Правила и техника безопасности при работе с ПК

В процессе работы с персональным компьютером человек сталкивается с рядом скрытых угроз:

- риском электротравм при контакте с поврежденными кабелями или корпусами;
- постоянным воздействием электромагнитного излучения;
- перенапряжением органов зрения из-за некорректного контраста и бликов;
- статической нагрузкой на позвоночник и суставы кистей рук.

Для снижения вредного воздействия ПК рабочая зона организуется строго в соответствии с актуальными гигиеническими нормативами [11]. Искусственное освещение стола должно быть мягким и рассеянным (в пределах 300–500 лк), предотвращающим появление направленных отражений на мониторе.

Особое внимание уделяется геометрии посадки. Оптимальное расстояние от глаз до дисплея составляет около 60–70 см. Конструкция рабочего кресла должна поддерживать естественные изгибы позвоночника, а высота сиденья – обеспечивать упор стоп в пол под прямым углом в коленях. Чтобы минимизировать воздействие электромагнитного фона, задние панели системных блоков и мониторов (где излучение наиболее интенсивно) рекомендуется

ориентировать в сторону стен, а не в сторону соседних рабочих мест. Кроме того, при непрерывной работе за компьютером обязательны перерывы по 5–10 минут через каждый час для разминки глаз и смены положения тела [11].

6.2 Правила и техника безопасности при работе со смартфоном

Поскольку создаваемое приложение ориентировано на смартфоны, необходимо зафиксировать правила безопасного взаимодействия с мобильными интерфейсами. Риски здесь делятся на технические и физиологические.

С точки зрения технической безопасности, ядром смартфона является литий-ионный аккумулятор, чувствительный к внешним воздействиям. Категорически запрещаются удары по корпусу, его разгерметизация, использование несертифицированных зарядных устройств и эксплуатация аппарата в условиях экстремального нагрева или высокой влажности – это может запустить необратимый процесс теплового разгона батареи и привести к взрыву [12].

С физиологической стороны, портативный формат устройств провоцирует появление специфических патологий. К ним относятся сильное мышечное перенапряжение из-за постоянного наклона головы к экрану и зрительная усталость. Свечение современных дисплеев в синем спектре в вечернее время подавляет выработку гормона сна, что нарушает биоритмы пользователя. Для защиты здоровья рекомендуется активировать в операционной системе смартфона автоматический режим защиты глаз, делать регулярные паузы и избегать длительного чтения с экрана в полной темноте.

Заключение

Основным итогом выполнения выпускной квалификационной работы является проектирование и реализация мобильного приложения для онлайн-записи на техническое обслуживание автомобилей. Разработанный программный продукт в полной мере отвечает требованиям исходного технического задания. В процессе достижения цели исследования были успешно решены следующие задачи:

- анализ темы и сравнительный анализ существующих аналогов;
- выбор языка программирования, сред разработки, базы данных и фреймворка;
- анализ архитектуры разрабатываемого приложения;
- реализована серверная бизнес-логика (бэкенд) и клиентская мобильная часть приложения;
- тестирование программного обеспечения.

В результате был создан мобильный клиент, написанный на языке Kotlin согласно принципам чистой архитектуры. Также была разработана серверная часть приложения, написанная на языке Kotlin с использованием многослойной архитектуры и фреймворка Spring Boot. Были получены навыки асинхронного программирования, организовано отказоустойчивое сетевое взаимодействие на основе архитектурного стиля REST API, развернута реляционная система управления базами данных PostgreSQL, а также реализовано разграничение доступа и обеспечена безопасность авторизации и сессий пользователей с использованием JWT-токенов.

В качестве дальнейшего развития проекта может быть выполнено масштабирование системы за счет внедрения спутниковых картографических сервисов и системных служб геолокации Android для автоматического поиска ближайших мастерских, push-уведомлений для оповещения об изменении статуса записей, а также чатов для связи автовладельцев с мастерами.

					ПГУ1.090301.45.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. autostat.ru: [Электронный ресурс]. URL: <https://www.autostat.ru/infographics/59994/> (дата обращения: 08.04.2026)
2. autostat.ru: [Электронный ресурс]. URL: <https://www.autostat.ru/infographics/61565/> (дата обращения: 08.04.2026)
3. fitauto.ru: [Электронный ресурс]. URL: <https://fitauto.ru/> (дата обращения: 09.04.2026)
4. uremont.com: [Электронный ресурс]. URL: <https://uremont.com/> (дата обращения: 09.04.2026)
5. wilgood.ru: [Электронный ресурс]. URL: <https://wilgood.ru/> (дата обращения: 09.04.2026)
6. metanit.com: [Электронный ресурс]. URL: <https://metanit.com/java/tutorial/1.1.php> (дата обращения: 15.04.2026)
7. kotlinlang.com: [Электронный ресурс]. URL: <https://kotlinlang.org/> (дата обращения: 15.04.2026)
8. spring.io: [Электронный ресурс]. URL: <https://spring.io/projects/spring-boot> (дата обращения: 16.04.2026)
9. developer.android.com: [Электронный ресурс]. URL: <https://developer.android.com/studio> (дата обращения: 16.04.2026)
10. www.jetbrains.com: [Электронный ресурс]. URL: <https://www.jetbrains.com/ru-ru/idea/features/> (дата обращения: 17.04.2026)
11. СанПиН 1.2.3685-21. Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания // Постановление Главного государственного санитарного врача РФ от 28 января 2021 г. – 2021. № 2.
12. ГОСТ Р МЭК 62133-2-2019. Аккумуляторы литий-ионные для портативных применений. Требования безопасности. – Москва : Стандартинформ, 2020. – 28 с.

Приложение А

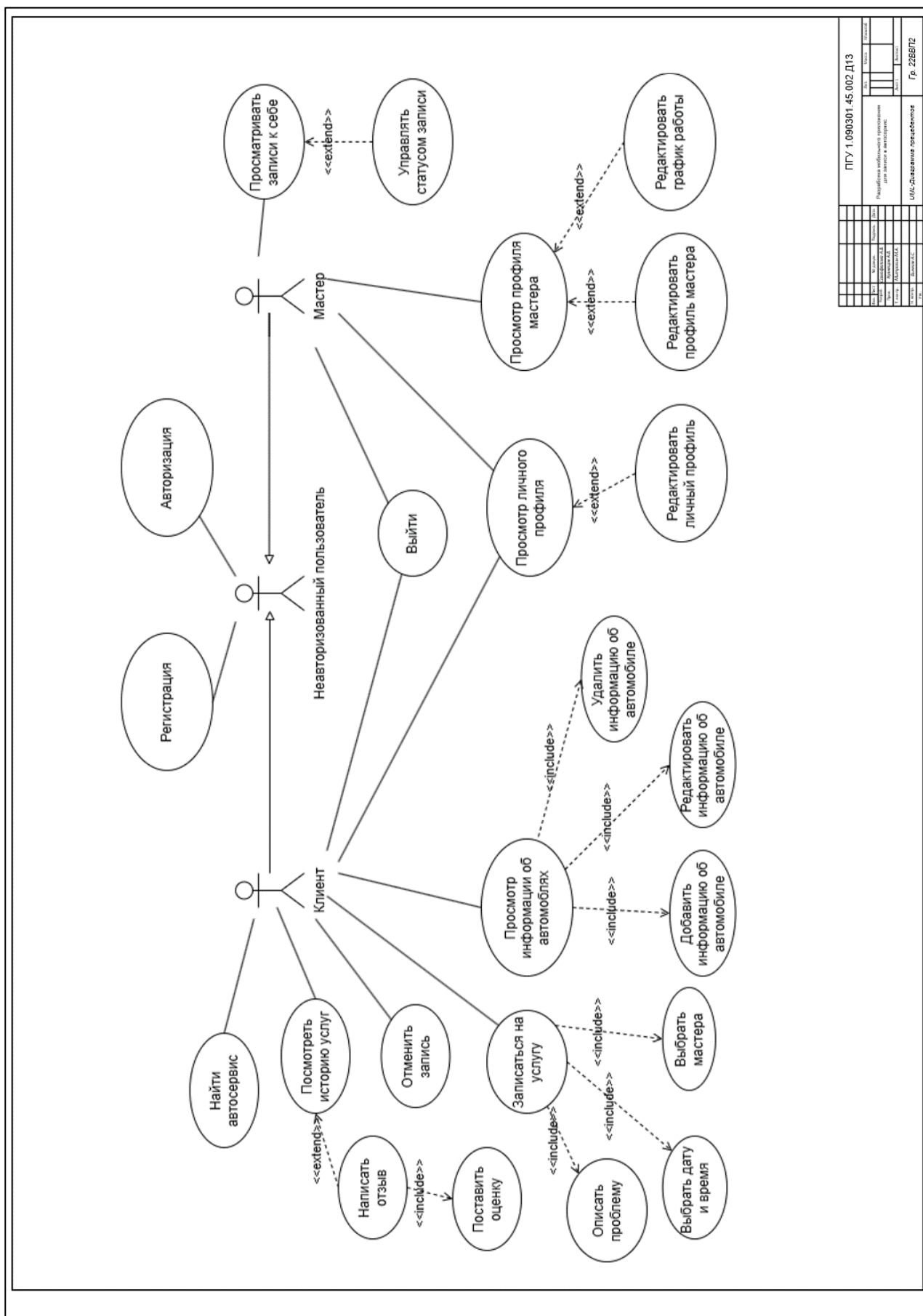
(обязательное)

Разработка мобильного приложения для записи в автосервис

UML-диаграммы

					ПГУ1.090301.45.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма прецедентов



ПГУ 1.090301.45.002 Д13		Имя	Фамилия	Отчество	Дата
Разработка системы управления автосервисом для записи и отмены записей		Имя	Фамилия	Отчество	Дата
ИИС: Управление автосервисом		Имя	Фамилия	Отчество	Дата
Гр. 228672		Имя	Фамилия	Отчество	Дата

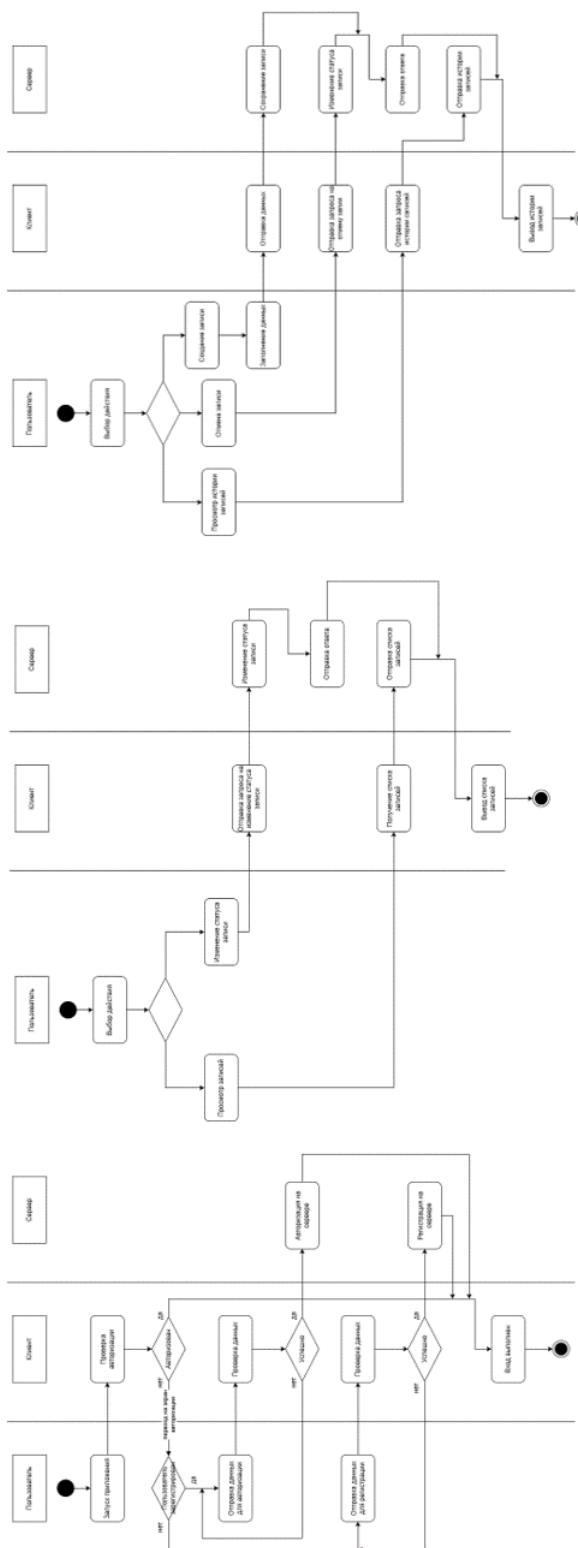
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.45.001 ПЗ

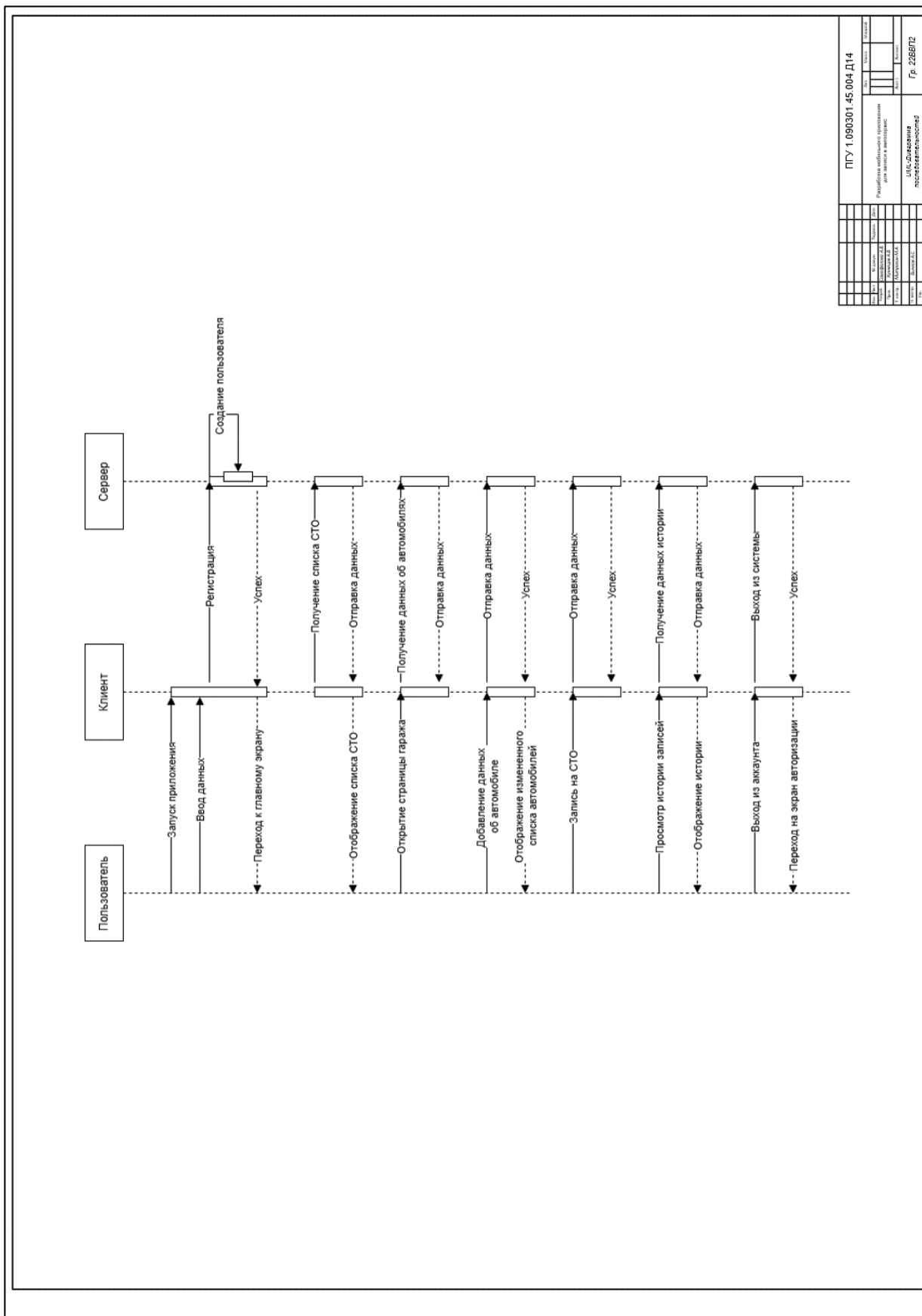
Лист

68

Приложение А.2 – Диаграмма деятельности

[illegible]

Приложение А.3 – Диаграмма последовательностей



ПУ 1.090301.45.004 Д14									
Разработка мобильного приложения для работы в интернете									
№	1	2	3	4	5	6	7	8	9
Имя	Иванов	Петров	Сидоров	Кузнецов	Левченко	Васильев	Михайлов	Попов	Смирнов
Фамилия	Иванов	Петров	Сидоров	Кузнецов	Левченко	Васильев	Михайлов	Попов	Смирнов
Имя-Отчество	Иванов	Петров	Сидоров	Кузнецов	Левченко	Васильев	Михайлов	Попов	Смирнов
Пол	М	М	М	М	М	М	М	М	М
Дата	01.01.2020	02.02.2020	03.03.2020	04.04.2020	05.05.2020	06.06.2020	07.07.2020	08.08.2020	09.09.2020
Время	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00
Гр	2266172								

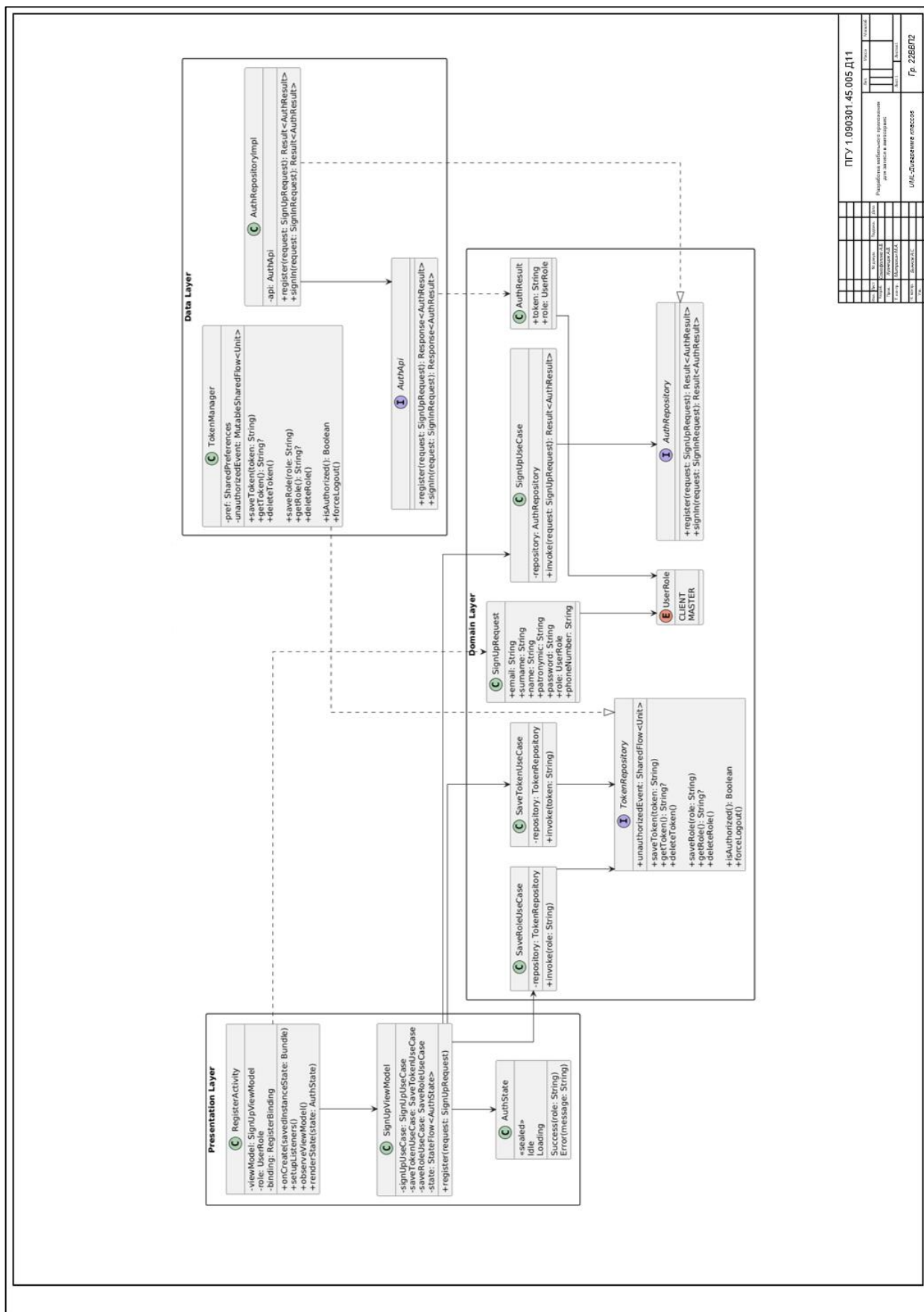
Изм.	Лист	№ докум.	Подпись	Дата

ПУ1.090301.45.001 ПЗ

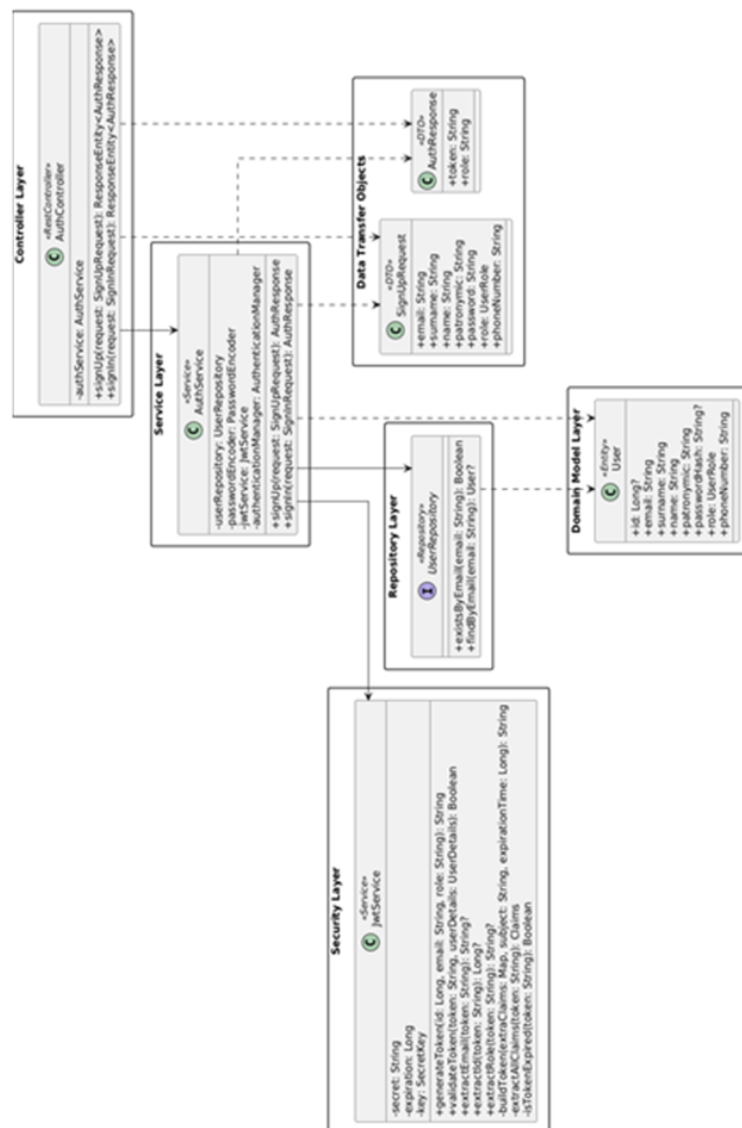
Лист

70

Приложение А.4 – Диаграмма классов клиентской части



Приложение А.5 – Диаграмма классов серверной части

[illegible]

Приложение Б

(обязательное)

Разработка мобильного приложения для записи в автосервис

Листинг программы

					ПГУ1.090301.45.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «WelcomeScreen.kt»

```
package com.example.automaster.presentation.view
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.example.automaster.domain.usecase.CheckAuthUseCase
import com.example.automaster.domain.usecase.DeleteTokenUseCase
import com.example.automaster.domain.usecase.GetRoleUseCase
import dagger.hilt.android.AndroidEntryPoint
import javax.inject.Inject

@AndroidEntryPoint
class WelcomeScreen : AppCompatActivity() {
    @Inject
    lateinit var checkAuthUseCase: CheckAuthUseCase
    @Inject
    lateinit var getRoleUseCase: GetRoleUseCase
    @Inject
    lateinit var deleteTokenUseCase: DeleteTokenUseCase
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        checkAuth()
    }

    private fun checkAuth(){
        if(checkAuthUseCase()){
            startMainActivity()
            return
        }
        startSignInActivity()
    }

    private fun startMainActivity(){
        val role = getRoleUseCase()

        val intent = when (role) {
            "MASTER" -> Intent(this, MasterActivity::class.java)
            "CLIENT" -> Intent(this, ClientActivity::class.java)
            else -> {
                deleteTokenUseCase()
                Intent(this, SignInActivity::class.java)
            }
        }
        intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
        startActivity(intent)
        finish()
    }

    private fun startSignInActivity(){
        val intent = Intent(this, SignInActivity::class.java)
```

					ПГУ1.090301.45.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		74

```

        intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
        startActivity(intent)
        finish()
    }
}

```

Файл «RegisterActivity.kt»

```

package com.example.automaster.presentation.view
import android.content.Intent
import android.os.Build
import android.os.Bundle
import android.view.View
import android.widget.Toast
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.Lifecycle
import com.example.automaster.databinding.RegisterBinding
import
com.example.automaster.presentation.viewmodel.SignUpViewModel
import kotlin.getValue
import androidx.lifecycle.LifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import com.example.automaster.domain.model.SignUpRequest
import com.example.automaster.domain.model.UserRole
import com.example.automaster.presentation.state.AuthState
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
@AndroidEntryPoint
class RegisterActivity: AppCompatActivity() {
    private val viewModel: SignUpViewModel by viewModels()
    private lateinit var role: UserRole
    private lateinit var binding: RegisterBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = RegisterBinding.inflate(layoutInflater)
        setContentView(binding.root)
        role = if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
            intent.getSerializableExtra("USER_ROLE",
UserRole::class.java)
        } else {
            @Suppress("DEPRECATION")
            intent.getSerializableExtra("USER_ROLE") as?
UserRole
        } ?: UserRole.CLIENT
        setupListeners()
        observeViewModel()
    }

    private fun setupListeners() {
        binding.regButton.setOnClickListener {

```

```

        val name = binding.textName.text.toString()
        val surname = binding.textSurname.text.toString()
        val patronymic =
binding.textPatronymic.text.toString()
        val login = binding.textLogin.text.toString()
        val password = binding.textPassword.text.toString()
        val phone = binding.textNumber.text.toString()
        val request = SignUpRequest(login, surname, name,
patronymic, password, role, phone)
        viewModel.register(request)
    }
}

private fun renderState(state: AuthState) {
    when(state) {
        is AuthState.Idle -> {
            binding.regButton.isEnabled = true
        }
        is AuthState.Loading -> {
            binding.regButton.isEnabled = false
        }
        is AuthState.Success -> {
            binding.tvError.visibility = View.GONE
            Toast.makeText(this, "Регистрация успешна!",
Toast.LENGTH_SHORT).show()
            if (role == UserRole.MASTER) {
                val intent = Intent(this,
MasterProfileRegisterActivity::class.java)
                intent.flags =
Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                startActivity(intent)
                finish()
            } else {
                val intent = Intent(this,
ClientActivity::class.java)
                intent.flags =
Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
                startActivity(intent)
                finish()
            }
        }
        is AuthState.Error -> {
            binding.regButton.isEnabled = true
            if (state.message != "Нет сети") {
                binding.tvError.text = state.message
                binding.tvError.visibility = View.VISIBLE
            } else Toast.makeText(this, state.message,
Toast.LENGTH_LONG).show()
        }
    }
}

```

					ПГУ1.090301.45.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		76

```

        private fun observeViewModel() {
            lifecycleScope.launch {
                repeatOnLifecycle(Lifecycle.State.STARTED) {
                    viewModel.state.collect {
                        state-> renderState(state)
                    }
                }
            }
        }
    }
}

```

Файл «SignInActivity.kt»

```

package com.example.automaster.presentation.view

import android.content.Intent
import android.os.Bundle
import android.view.View
import android.widget.Toast
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.lifecycleScope
import androidx.lifecycle.repeatOnLifecycle
import com.example.automaster.databinding.SignInBinding
import com.example.automaster.domain.model.SignInRequest
import com.example.automaster.domain.model.UserRole
import com.example.automaster.presentation.state.AuthState
import
com.example.automaster.presentation.viewmodel.SignInViewModel
import com.example.automaster.data.local.TokenManager
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
import javax.inject.Inject
import kotlin.getValue

@AndroidEntryPoint
class SignInActivity: AppCompatActivity() {
    private val viewModel: SignInViewModel by viewModels()
    private lateinit var binding: SignInBinding
    @Inject
    lateinit var tokenManager: TokenManager
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = SignInBinding.inflate(layoutInflater)
        setContentView(binding.root)
        setListeners()
        observeViewModel()
    }
    private fun setListeners() {
        binding.signInButton.setOnClickListener {
            val login = binding.textLogin.text.toString()
            val password = binding.textPassword.text.toString()

```

```

        val request = SignInRequest(login, password)
        viewModel.signIn(request)
    }
    binding.regCButton.setOnClickListener {
        val intent = Intent(this,
RegisterActivity::class.java)
        intent.putExtra("USER_ROLE", UserRole.CLIENT)
        startActivity(intent)
        finish()
    }
    binding.regMButton.setOnClickListener {
        val intent = Intent(this,
RegisterActivity::class.java)
        intent.putExtra("USER_ROLE", UserRole.MASTER)
        startActivity(intent)
        finish()
    }
}
private fun goToMain() {
    val role = tokenManager.getRole()

    if (role == "MASTER") {
        val intent = Intent(this,
MasterActivity::class.java)
        intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
        startActivity(intent)
        finish()
    } else {
        val intent = Intent(this,
ClientActivity::class.java)
        intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
Intent.FLAG_ACTIVITY_CLEAR_TASK
        startActivity(intent)
        finish()
    }
}
private fun renderState(state: AuthState) {
    when (state) {
        is AuthState.Idle -> {
            binding.signInButton.isEnabled = true
            binding.regCButton.isEnabled = true
            binding.regMButton.isEnabled = true
        }
        is AuthState.Loading -> {
            binding.signInButton.isEnabled = false
            binding.regCButton.isEnabled = false
            binding.regMButton.isEnabled = false
        }
        is AuthState.Success -> {
            Toast.makeText(this, "Вход успешен!",
Toast.LENGTH_SHORT).show()

```

```

        goToMain()
    }
    is AuthState.Error -> {
        binding.signInButton.isEnabled = true
        binding.regCButton.isEnabled = true
        binding.regMButton.isEnabled = true
        if(state.message != "Нет сети") {
            binding.tvError.text = state.message
            binding.tvError.visibility = View.VISIBLE
        }else Toast.makeText(this, state.message,
Toast.LENGTH_LONG).show()
        }
    }
}

private fun observeViewModel(){
    lifecycleScope.launch{
        repeatOnLifecycle(Lifecycle.State.STARTED){
            viewModel.state.collect{
                state-> renderState(state)
            }
        }
    }
}
}

```

Файл «MasterListFragment.kt»

```

package com.example.automaster.presentation.view

import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.AdapterView
import android.widget.AdapterView.AdapterView
import android.widget.ArrayAdapter
import android.widget.Toast
import androidx.core.widget.doOnTextChanged

import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.automaster.R
import com.example.automaster.databinding.MasterListBinding
import
com.example.automaster.presentation.state.MasterListState
import
com.example.automaster.presentation.viewmodel.MasterListViewModel
import dagger.hilt.android.AndroidEntryPoint

```

```

import kotlinx.coroutines.launch
import kotlin.getValue
import androidx.core.net.toUri

@AndroidEntryPoint
class MasterListFragment: Fragment() {
    private lateinit var adapter: MasterListAdapter
    private var _binding: MasterListBinding? = null
    private val _binding get() = _binding!!
    private val viewModel: MasterListViewModel by viewModels()
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = MasterListBinding.inflate(inflater,
container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        setupRecyclerView()
        setupCitySpinner()
        observeViewModel()
        setupSearch()
    }

    private fun setupSearch() {
        binding.etSearch.doOnTextChanged { text, _, _, _ ->
            viewModel.onSearchQueryChanged(text?.toString() ?:
"" )
        }
    }

    private fun setupCitySpinner() {
        val cities = listOf("Все города", "Москва", "Санкт-
Петербург", "Казань", "Новосибирск", "Пенза", "Ульяновск")

        val spinnerAdapter = ArrayAdapter(
            requireContext(),
            android.R.layout.simple_spinner_dropdown_item,
            cities
        )
        binding.citySpinner.adapter = spinnerAdapter

        binding.citySpinner.onItemSelectedListener = object :
AdapterView.OnItemSelectedListener {
            override fun onItemSelected(parent:
AdapterView<*>?, view: View?, position: Int, id: Long) {

```

					ПГУ1.090301.45.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

```

        val selectedCity = cities[position]
        viewModel.onCityFilterChanged(selectedCity)
    }

    override fun onNothingSelected(parent:
AdapterView<*>?) {}
    }

    private fun setupRecyclerView() {
        adapter = MasterListAdapter(
            onMapClick = { master ->
                openExternalMap(master.city + " " +
master.address)
            },
            onBookClick = { master ->
                val bundle = Bundle().apply {
                    putLong("MASTER_ID", master.id)
                    putString("MASTER_NAME", master.name)
                    putString("MASTER_ADDRESS", master.city + "
" + master.address)
                }

                findNavController().navigate(R.id.action_masterListFragment_to_appoi
ntmentFragment, bundle)
                Toast.makeText(requireContext(), "Переходим к
записи к: ${master.name}", Toast.LENGTH_SHORT).show()
            },
            onCardClick = { master ->
                val bundle = Bundle().apply {
                    putLong("MASTER_ID", master.id)
                    putString("MASTER_NAME", master.name)
                    putString("MASTER_ADDRESS", master.city + "
" + master.address)
                    putDouble("MASTER_RATING", master.rating)
                    putString("MASTER_SPECIALIZATION",
master.specialization)
                    putString("MASTER_EXPERIENCE",
master.experience)
                }

                findNavController().navigate(R.id.action_masterListFragment_to_maste
rInfoFragment, bundle)
            }
        )

        binding.mastersList.layoutManager =
LinearLayoutManager(requireContext())
        binding.mastersList.adapter = adapter
    }

    private fun observeViewModel() {

```

					ПГУ1.090301.45.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

```

        viewLifecycleOwner.lifecycleScope.launch {
            viewModel.state.collect { state ->
                when (state) {
                    is MasterListState.Loading -> {
                        binding.progressBar.visibility =
View.VISIBLE
                        binding.mastersList.visibility =
View.GONE
                    }
                    is MasterListState.Success -> {
                        binding.progressBar.visibility =
View.GONE
                        binding.mastersList.visibility =
View.VISIBLE
                        adapter.submitList(state.masters)
                    }
                    is MasterListState.Error -> {
                        binding.progressBar.visibility =
View.GONE
                        Toast.makeText(requireContext(),
state.message, Toast.LENGTH_LONG).show()
                    }
                }
            }
        }
    }
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    private fun openExternalMap(address: String) {
        val encodedQuery = Uri.encode(address)
        val uri = "geo:0,0?q=$encodedQuery"
        val intent = Intent(Intent.ACTION_VIEW, uri.toUri())

        try {
            startActivity(intent)
        } catch (e: Exception) {
            Toast.makeText(requireContext(), "Не найдено
приложение карт", Toast.LENGTH_SHORT).show()
        }
    }
}

```

Файл «MasterListAdapter.kt»

```

package com.example.automaster.presentation.view

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter

```

					ПГУ1.090301.45.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		82

```

import androidx.recyclerview.widget.RecyclerView
import com.example.automaster.databinding.ItemMasterBinding
import com.example.automaster.domain.model.MasterProfile

class MasterListAdapter(
    private val onMapClick: (MasterProfile) -> Unit,
    private val onBookClick: (MasterProfile) -> Unit,
    private val onCardClick: (MasterProfile) -> Unit
) : ListAdapter<MasterProfile,
MasterListAdapter.MasterViewHolder>(MasterDiffCallback()) {

    override fun onCreateViewHolder(parent: ViewGroup,
viewType: Int): MasterViewHolder {
        val binding = ItemMasterBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        )
        return MasterViewHolder(binding)
    }

    override fun onBindViewHolder(holder: MasterViewHolder,
position: Int) {
        val master = getItem(position)
        holder.bind(master, onMapClick, onBookClick,
onCardClick)
    }

    class MasterViewHolder(private val binding:
ItemMasterBinding) : RecyclerView.ViewHolder(binding.root) {
        fun bind(
            master: MasterProfile,
            onMapClick: (MasterProfile) -> Unit,
            onBookClick: (MasterProfile) -> Unit,
            onCardClick: (MasterProfile) -> Unit
        ) {
            binding.tvMasterName.text = master.name
            binding.tvAddress.text = buildString {
                append(master.city)
                append(" ")
                append(master.address)
            }
            binding.tvSpecialization.text =
master.specialization
            binding.tvExperience.text = master.experience
            binding.rbMasterAverage.rating =
master.rating.toFloat()

            binding.btnOpenMap.setOnClickListener {
onMapClick(master) }
            binding.btnBook.setOnClickListener {
onBookClick(master) }
        }
    }
}

```

```

        binding.tvMasterCard.setOnClickListener {
onCardClick(master) }
    }
}

class MasterDiffCallback :
DiffUtil.ItemCallback<MasterProfile>() {
    override fun areItemsTheSame(oldItem: MasterProfile,
newItem: MasterProfile): Boolean {
        return oldItem.id == newItem.id
    }
    override fun areContentsTheSame(oldItem: MasterProfile,
newItem: MasterProfile): Boolean {
        return oldItem == newItem
    }
}
}
}

```

Файл «AppointmentFragment.kt»

```

package com.example.automaster.presentation.view

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.AdapterView
import android.widget.Toast
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.LifecycleScope
import androidx.navigation.fragment.findNavController
import com.example.automaster.databinding.AppointmentBinding
import com.example.automaster.domain.model.BookingRequest
import com.example.automaster.domain.model.Car
import
com.example.automaster.presentation.state.AppointmentState
import
com.example.automaster.presentation.viewmodel.AppointmentViewModel
import
com.google.android.material.datepicker.CalendarConstraints
import
com.google.android.material.datepicker.DateValidatorPointForward
import
com.google.android.material.datepicker.MaterialDatePicker
import
com.google.android.material.dialog.MaterialAlertDialogBuilder
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.Date
import java.util.Locale
import kotlin.getValue

```

```

import android.os.Parcel
import android.os.Parcelable
import java.util.Calendar
import java.util.TimeZone

@AndroidEntryPoint
class AppointmentFragment: Fragment() {
    private var _binding: AppointmentBinding? = null
    private val binding get() = _binding!!
    private val viewModel: AppointmentViewModel by viewModels()

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

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val masterId = arguments?.getLong("MASTER_ID") ?:
return
        val masterName = arguments?.getString("MASTER_NAME") ?:
""
        val masterAddress =
arguments?.getString("MASTER_ADDRESS") ?: ""
        val masterInfo = "$masterName ($masterAddress)"
        viewModel.selectedMasterId = masterId
        binding.sto.text = masterInfo
        setupListeners()
        observeViewModel()
    }

    private fun setupListeners(){
        binding.dateTime.setOnClickListener {
            showDatePicker()
        }
        binding.bookBtn.setOnClickListener {
            val request = BookingRequest(
                masterId = viewModel.selectedMasterId,
                carId = viewModel.selectedCarId,
                date = viewModel.selectedDate,
                timeSlot = viewModel.selectedTime,
                comment = binding.commentText.text.toString()
            )

            viewModel.createAppointment(request)
        }
    }
}

```

```

        binding.carDropdown.setOnItemClickListener{
            parent, _, position, _ ->
                val selectedCar =
parent.getItemAtPosition(position) as Car
                viewModel.selectedCarId = selectedCar.id
        }
        binding.btnBack.setOnClickListener{
            findNavController().popBackStack()
        }
    }

    private fun showDatePicker() {
        val blockedDatesSet = viewModel.blockedDates

        val constraintsBuilder = CalendarConstraints.Builder()

        .setValidator(BlockedDatesValidator(blockedDatesSet))

        val datePicker =
MaterialDatePicker.Builder.datePicker()
            .setTitleText("Выберите дату визита")
            .setCalendarConstraints(constraintsBuilder.build())
            .build()

        datePicker.addOnPositiveButtonClickListener { selection
->
            val sdf = SimpleDateFormat("yyyy-MM-dd",
Locale.US).apply {
                timeZone = TimeZone.getTimeZone("UTC")
            }
            val dateStr = sdf.format(Date(selection))

            viewModel.selectedDate = dateStr
            viewModel.getTimeSlot(viewModel.selectedMasterId,
dateStr)
        }

        datePicker.show(parentFragmentManager, "DATE_PICKER")
    }

    private fun showTimeSlotsDialog(slots: List<String>) {
        val arraySlots = slots.toTypedArray()

        MaterialAlertDialogBuilder(requireContext())
            .setTitle("Выберите время")
            .setItems(arraySlots) { _, which ->
                val selectedTime = arraySlots[which]
                viewModel.selectedTime = selectedTime
            }

        binding.dateTime.setText("${viewModel.selectedDate} в
${selectedTime}")
    }

```

					ПГУ1.090301.45.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

        }
        .show()
    }

    private fun setCarsDropdown(cars: List<Car>){
        if(cars.isEmpty()){
            binding.carDropdown.setText("У вас пока нет
автомобилей", false)
            binding.carDropdown.isEnabled = false
            return
        }
        val adapter = ArrayAdapter(
            requireContext(),
            android.R.layout.simple_dropdown_item_1line,
            cars
        )
        binding.carDropdown.setAdapter(adapter)
        binding.carDropdown.isEnabled = true
    }
    private fun observeViewModel() {
        viewLifecycleOwner.lifecycleScope.launch {
            viewModel.state.collect { state ->
                when (state) {
                    is AppointmentState.Idle -> {
                        binding.progressBar.visibility =
View.GONE
                    }
                    is AppointmentState.Loading -> {
                        binding.progressBar.visibility =
View.VISIBLE
                        binding.bookBtn.isEnabled = false
                    }
                    is AppointmentState.Success -> {
                        binding.progressBar.visibility =
View.GONE
                        binding.bookBtn.isEnabled = true
                        Toast.makeText(requireContext(),
state.message, Toast.LENGTH_LONG).show()
                        findNavController().popBackStack()
                    }
                    is AppointmentState.Error -> {
                        binding.progressBar.visibility =
View.GONE
                        binding.bookBtn.isEnabled = true
                        Toast.makeText(requireContext(),
state.message, Toast.LENGTH_LONG).show()
                    }
                    is AppointmentState.TimeSlotLoaded -> {
                        binding.progressBar.visibility =
View.GONE
                        binding.bookBtn.isEnabled = true
                        if(state.slots.isEmpty()){

```

```

Toast.makeText(requireContext(),
"Нет свободного времени на эту дату", Toast.LENGTH_LONG).show()
    }else{
        showTimeSlotsDialog(state.slots)
    }
}
is AppointmentState.CarsLoaded -> {
    binding.progressBar.visibility =
View.GONE
        setCarsDropdown(state.cars)
    }
}
}
}
}

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

}

class BlockedDatesValidator(private val blockedDates:
Set<String>) : CalendarConstraints.DateValidator {

    private val formatter = SimpleDateFormat("yyyy-MM-dd",
Locale.US).apply {
        timeZone = TimeZone.getTimeZone("UTC")
    }

    override fun isValid(dateLong: Long): Boolean {
        val calendar =
Calendar.getInstance(TimeZone.getTimeZone("UTC"))
        calendar.set(Calendar.HOUR_OF_DAY, 0)
        calendar.set(Calendar.MINUTE, 0)
        calendar.set(Calendar.SECOND, 0)
        calendar.set(Calendar.MILLISECOND, 0)
        if (dateLong < calendar.timeInMillis) return false

        val dateString = formatter.format(Date(dateLong))
        return !blockedDates.contains(dateString)
    }

    override fun writeToParcel(dest: Parcel, flags: Int) {
        dest.writeStringList(blockedDates.toList())
    }

    override fun describeContents(): Int = 0
    companion object CREATOR :
Parcelable.Creator<BlockedDatesValidator> {
        override fun createFromParcel(parcel: Parcel):
BlockedDatesValidator {

```

```

        val list = parcel.createStringArrayList() ?:
emptyList<String>()
        return BlockedDatesValidator(list.toSet())
    }
    override fun newArray(size: Int):
Array<BlockedDatesValidator?> = arrayOfNulls(size)
    }
}

```

Файл «SignUpViewModel.kt»

```

package com.example.automaster.presentation.viewmodel

import androidx.lifecycle.ViewModel
import com.example.automaster.domain.model.SignUpRequest
import com.example.automaster.domain.usecase.SignUpUseCase
import com.example.automaster.presentation.state.AuthState
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import androidx.lifecycle.viewModelScope
import com.example.automaster.data.local.TokenManager
import com.example.automaster.domain.usecase.SaveRoleUseCase
import com.example.automaster.domain.usecase.SaveTokenUseCase
import kotlinx.coroutines.launch

@HiltViewModel
class SignUpViewModel @Inject constructor(
    private val signUpUseCase: SignUpUseCase,
    private val saveTokenUseCase: SaveTokenUseCase,
    private val saveRoleUseCase: SaveRoleUseCase
): ViewModel(){
    private val _state =
MutableStateFlow<AuthState>(AuthState.Idle)
    val state: StateFlow<AuthState> = _state.asStateFlow()

    fun register(request: SignUpRequest){
        viewModelScope.launch{
            _state.value = AuthState.Loading

            signUpUseCase(request).fold(
                onSuccess = {
                    result ->
                    saveTokenUseCase(result.token)
                    saveRoleUseCase(result.role.name)
                    _state.value =
AuthState.Success(result.role.name)
                },
                onFailure = {
                    exception ->

```

```

        _state.value =
        AuthState.Error(exception.message ?:"Ошибка регистрации")
    }
    )
}
}
}

```

Файл «SignInViewModel.kt»

```

package com.example.automaster.presentation.viewmodel

import androidx.lifecycle.ViewModel
import com.example.automaster.presentation.state.AuthState
import dagger.hilt.android.lifecycle.HiltViewModel
import javax.inject.Inject
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import androidx.lifecycle.viewModelScope
import com.example.automaster.data.local.TokenManager
import com.example.automaster.domain.model.SignInRequest
import com.example.automaster.domain.usecase.SaveRoleUseCase
import com.example.automaster.domain.usecase.SaveTokenUseCase
import com.example.automaster.domain.usecase.SignInUseCase
import kotlinx.coroutines.launch

@HiltViewModel
class SignInViewModel @Inject constructor(
    private val signInUseCase: SignInUseCase,
    private val saveTokenUseCase: SaveTokenUseCase,
    private val saveRoleUseCase: SaveRoleUseCase
): ViewModel(){
    private val _state =
MutableStateFlow<AuthState>(AuthState.Idle)
    val state: StateFlow<AuthState> = _state.asStateFlow()

    fun signIn(request: SignInRequest){
        viewModelScope.launch{
            _state.value = AuthState.Loading

            signInUseCase(request).fold(
                onSuccess = {
                    result ->
                    saveTokenUseCase(result.token)
                    saveRoleUseCase(result.role.name)
                    _state.value =
AuthState.Success(result.role.name)
                },
                onFailure = {
                    exception ->

```

```

        _state.value =
AuthState.Error(exception.message ?:"Ошибка регистрации")
    }
    )
}
}
}

```

Файл «AuthState.kt»

```

package com.example.automaster.presentation.state

sealed class AuthState {
    object Idle: AuthState()
    object Loading: AuthState()
    data class Success(val role: String) : AuthState()
    data class Error(val message: String) : AuthState()
}

```

Файл «SignUpUseCase.kt»

```

package com.example.automaster.domain.usecase

import android.util.Patterns
import com.example.automaster.domain.model.AuthResult
import com.example.automaster.domain.model.SignUpRequest
import com.example.automaster.domain.repository.AuthRepository
import javax.inject.Inject

class SignUpUseCase @Inject constructor(private val repository:
AuthRepository) {
    suspend operator fun invoke(request: SignUpRequest):
Result<AuthResult>{
        if(request.password.isBlank() ||
            request.email.isBlank() ||
            request.surname.isBlank() ||
            request.name.isBlank() ||
            request.phoneNumber.isBlank()){
            return Result.failure(IllegalArgumentException("Поля
не должны быть пусты"))
        }else
        if(!Patterns.EMAIL_ADDRESS.matcher(request.email).matches()){
            return
Result.failure(IllegalArgumentException("Неверный формат
почты"))
        } else if(request.password.length<8){
            return
Result.failure(IllegalArgumentException("Пароль должен быть не
менее 8 символов"))
        }else
        if(!Patterns.PHONE.matcher(request.phoneNumber).matches()){
            return
Result.failure(IllegalArgumentException("Неверный формат номера
телефона"))
        }
    }
}

```

```

    }
    return repository.register(request)
}
}

```

Файл «SignInUseCase.kt»

```

package com.example.automaster.domain.usecase

import android.util.Patterns
import com.example.automaster.domain.model.AuthResult
import com.example.automaster.domain.model.SignInRequest
import com.example.automaster.domain.repository.AuthRepository
import javax.inject.Inject

class SignInUseCase @Inject constructor(private val repository:
AuthRepository) {
    suspend operator fun invoke(request: SignInRequest):
Result<AuthResult>{
        if(request.password.isBlank() ||
request.email.isBlank()){
            return Result.failure(IllegalArgumentException("Поля
не должны быть пусты"))
        }else
        if(!Patterns.EMAIL_ADDRESS.matcher(request.email).matches()){
            return
Result.failure(IllegalArgumentException("Неверный формат
почты"))
        }
        return repository.signIn(request)
    }
}

```

Файл «AuthRepository.kt»

```

package com.example.automaster.domain.repository

import com.example.automaster.domain.model.AuthResult
import com.example.automaster.domain.model.SignInRequest
import com.example.automaster.domain.model.SignUpRequest

interface AuthRepository {
    suspend fun register(request: SignUpRequest):
Result<AuthResult>
    suspend fun signIn(request: SignInRequest):
Result<AuthResult>
}

```

Файл «SignUpRequest.kt»

```

data class SignUpRequest(
    val email: String,
    val surname: String,
    val name: String,
    val patronymic: String,

```

					ПГУ1.090301.45.001 ПЗ	Лист
						92
Изм.	Лист	№ докум.	Подпись	Дата		

```

        val password: String,
        val role: UserRole,
        val phoneNumber: String
    )

```

Файл «SignInRequest.kt»

```

data class SignInRequest(
    val email: String,
    val password: String
)

```

Файл «AuthResult.kt»

```

data class AuthResult(
    val token: String,
    val role: UserRole
)

```

Файл «AuthRepositoryImpl.kt»

```

class AuthRepositoryImpl @Inject constructor(
    private val api: AuthApi
): AuthRepository{
    override suspend fun register(request: SignUpRequest):
    Result<AuthResult>{
        return try {
            val response = api.register(request)
            if (response.isSuccessful && response.body() !=
null) {
                Result.success(response.body()!!)
            }else Result.failure(Exception("Пользователь с таким
email уже существует"))
        }catch(e: Exception){
            Result.failure(Exception("Нет сети"))
        }
    }

    override suspend fun signIn(request: SignInRequest):
    Result<AuthResult>{
        return try{
            val response = api.signIn(request)
            if (response.isSuccessful && response.body() !=
null) {
                Result.success(response.body()!!)
            }else Result.failure(Exception("Неверный логин или
пароль"))
        } catch(e: Exception){
            Result.failure(Exception("Нет сети"))
        }
    }
}

```

Файл «AuthApi.kt»

```

package com.example.automaster.data.api

```

```

import com.example.automaster.domain.model.AuthResult
import com.example.automaster.domain.model.SignInRequest
import com.example.automaster.domain.model.SignUpRequest
import retrofit2.Response
import retrofit2.http.POST
import retrofit2.http.Body

interface AuthApi {
    @POST("/api/auth/register")
    suspend fun register(@Body request: SignUpRequest):
    Response<AuthResult>

    @POST("/api/auth/login")
    suspend fun signIn(@Body request: SignInRequest):
    Response<AuthResult>
}

```

Файл «AppointmentViewModel.kt»

```

package com.example.automaster.presentation.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.automaster.domain.model.BookingRequest
import com.example.automaster.domain.model.Car
import
com.example.automaster.domain.usecase.BookAppointmentUseCase
import
com.example.automaster.domain.usecase.GetBlockedDatesUseCase
import com.example.automaster.domain.usecase.GetCarsUseCase
import com.example.automaster.domain.usecase.GetTimeSlotUseCase
import
com.example.automaster.presentation.state.AppointmentState
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import java.time.LocalDate
import javax.inject.Inject

@HiltViewModel
class AppointmentViewModel @Inject constructor(
    private val bookAppointmentUseCase: BookAppointmentUseCase,
    private val getTimeSlotUseCase: GetTimeSlotUseCase,
    private val getCarsUseCase: GetCarsUseCase,
    private val getBlockedDatesUseCase: GetBlockedDatesUseCase
): ViewModel() {
    private val _state =
MutableStateFlow<AppointmentState>(AppointmentState.Idle)
    val state: StateFlow<AppointmentState> =
_state.asStateFlow()
}

```

					ПГУ1.090301.45.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		94

```

var selectedMasterId: Long = 0
    set(value) {
        field = value
        if (value != 0L) loadBlockedDates(value)
    }

var selectedDate: String = ""
var selectedTime: String = ""
var selectedCarId: Long = 0

var blockedDates: Set<String> = emptySet()
    private set

private var cachedCars: List<Car> = emptyList()

init {
    loadCars()
}

private fun loadBlockedDates(masterId: Long) {
    viewModelScope.launch {
        getBlockedDatesUseCase(masterId).fold(
            onSuccess = { dates ->
                blockedDates = dates.toSet()
                _state.value = AppointmentState.Idle
            },
            onFailure = { error ->
                _state.value =
AppointmentState.Error(error.message ?: "Ошибка загрузки
календаря")
            }
        )
    }
}

fun getTimeSlot(masterId: Long, date: String) {
    viewModelScope.launch {
        _state.value = AppointmentState.Loading
        getTimeSlotUseCase(masterId, date).fold(
            onSuccess = { slots ->
                _state.value =
AppointmentState.TimeSlotLoaded(slots)
            },
            onFailure = { error ->
                _state.value =
AppointmentState.Error(error.message ?: "Ошибка")
            }
        )
    }
}

```

```

        fun createAppointment(request: BookingRequest) {
            viewModelScope.launch {
                _state.value = AppointmentState.Loading
                bookAppointmentUseCase(request).fold(
                    onSuccess = {
                        _state.value =
AppointmentState.Success("Запись успешно выполнена")
                    },
                    onFailure = { error ->
                        _state.value =
AppointmentState.Error(error.message ?: "Ошибка")
                    }
                )
            }
        }

        fun loadCars() {
            viewModelScope.launch {
                _state.value = AppointmentState.Loading
                getCarsUseCase().fold(
                    onSuccess = { cars ->
                        cachedCars = cars
                        _state.value =
AppointmentState.CarsLoaded(cars)
                    },
                    onFailure = { error ->
                        _state.value =
AppointmentState.Error(error.message ?: "Ошибка! Не удалось
загрузить автомобили")
                    }
                )
            }
        }
    }
}

```

Файл «AppointmentState.kt»

```

package com.example.automaster.presentation.state

import com.example.automaster.domain.model.Car

sealed class AppointmentState {
    object Idle: AppointmentState()
    object Loading: AppointmentState()
    data class Success(val message: String) : AppointmentState()
    data class Error(val message: String) : AppointmentState()
    data class TimeSlotLoaded (val slots: List<String>) :
AppointmentState()
    data class CarsLoaded(val cars: List<Car>) :
AppointmentState()
}

```

Файл «BookingApi.kt»

```
package com.example.automaster.data.api

import com.example.automaster.domain.model.BookingRequest
import retrofit2.Response
import retrofit2.http.Body
import retrofit2.http.GET
import retrofit2.http.POST
import retrofit2.http.Query
import java.time.LocalDate

interface BookingApi {
    @POST("/api/appointments/appointment")
    suspend fun createAppointment(@Body request: BookingRequest)

    @GET("/api/appointments/appointment/timeslots")
    suspend fun getTimeSlot(
        @Query("masterId") masterId: Long,
        @Query("date") date: String): Response<List<String>>

    @GET("/api/appointments/appointment/blocked-dates")
    suspend fun getBlockedDates(
        @Query("masterId") masterId: Long
    ): Response<List<String>>
}
```

Файл «BookingRepository.kt»

```
package com.example.automaster.domain.repository
import com.example.automaster.domain.model.BookingRequest
import java.time.LocalDate

interface BookingRepository {
    suspend fun bookAppointment(request: BookingRequest):
    Result<Unit>
    suspend fun getTimeSlot(masterId: Long, date:
    String):Result<List<String>>
    suspend fun getBlockedDates(masterId: Long):
    Result<List<String>>
}
```

Файл «BookingRepositoryImpl.kt»

```
package com.example.automaster.data.repository

import com.example.automaster.data.api.BookingApi
import com.example.automaster.domain.model.BookingRequest
import
com.example.automaster.domain.repository.BookingRepository
import java.time.LocalDate
import javax.inject.Inject

class BookingRepositoryImpl @Inject constructor(
```

					ПГУ1.090301.45.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		97

```

        private val api: BookingApi
    ): BookingRepository{
        override suspend fun bookAppointment(request:
BookingRequest): Result<Unit>{
            return try{
                api.createAppointment(request)
                Result.success(Unit)
            } catch(e: Exception){
                Result.failure(e)
            }
        }

        override suspend fun getTimeSlot(masterId: Long, date:
String): Result<List<String>> {
            return try {
                val response = api.getTimeSlot(masterId, date)
                if (response.isSuccessful && response.body() !=
null) {
                    Result.success(response.body()!!)
                }else Result.failure(Exception("Не удалось получить
список доступного времени"))
            }catch(e: Exception){
                Result.failure(e)
            }
        }

        override suspend fun getBlockedDates(masterId: Long):
Result<List<String>> {
            return try {
                val response = api.getBlockedDates(masterId)
                if (response.isSuccessful && response.body() !=
null) {
                    Result.success(response.body()!!)
                } else {
                    Result.failure(Exception("Не удалось загрузить
недоступные даты"))
                }
            } catch (e: Exception) {
                Result.failure(Exception("Ошибка подключения к
сети"))
            }
        }
    }
}

```

Файл «BookAppointmentUseCase.kt»

```

package com.example.automaster.domain.usecase

import com.example.automaster.domain.model.BookingRequest
import
com.example.automaster.domain.repository.BookingRepository
import java.time.LocalDate
import javax.inject.Inject

```

```

class BookAppointmentUseCase @Inject constructor(private val
repository: BookingRepository) {
    suspend operator fun invoke(request: BookingRequest):
Result<Unit>{
        return repository.bookAppointment(request)
    }
}

class GetTimeSlotUseCase @Inject constructor(private val
repository: BookingRepository) {
    suspend operator fun invoke(masterId: Long, date: String):
Result<List<String>>{
        return repository.getTimeSlot(masterId, date)
    }
}

class GetBlockedDatesUseCase @Inject constructor(
    private val repository: BookingRepository
) {
    suspend operator fun invoke(masterId: Long):
Result<List<String>> {
        if (masterId <= 0) {
            return
Result.failure(IllegalArgumentException("Неверный ID мастера"))
        }
        return repository.getBlockedDates(masterId)
    }
}

```

Файл «register.xml»

```

<?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"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".presentation.view.WelcomeScreen">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/inter_bold"
        android:layout_gravity="center"
        android:layout_marginTop="50dp"
        android:textSize="20sp"
        android:text="@string/registration"/>

```

```

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilLogin"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:inputType="textEmailAddress"
        android:hint="@string/login" />
</com.google.android.material.textfield.TextInputLayout>
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilSurname"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textSurname"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="@string/surname" />
</com.google.android.material.textfield.TextInputLayout>
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilName"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="@string/name" />
</com.google.android.material.textfield.TextInputLayout>
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilPatronymic"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textPatronymic"
        android:layout_width="match_parent"

```

```

        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="@string/patronymic" />
</com.google.android.material.textfield.TextInputLayout>
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilPassword"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="@string/password"
        android:inputType="textPassword"/>
</com.google.android.material.textfield.TextInputLayout>
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilNumber"
    android:layout_gravity="center"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/textNumber"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:inputType="phone"
        android:hint="@string/tel_number" />
</com.google.android.material.textfield.TextInputLayout>
<TextView
    android:id="@+id/tvError"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:fontFamily="@font/inter_bold"
    android:layout_gravity="center"
    android:layout_marginTop="5dp"
    android:textSize="15sp"
    android:textColor="@color/md_theme_error"
    android:visibility="gone"
    android:text="Ошибка"/>
<com.google.android.material.button.MaterialButton
    android:id="@+id/regButton"
    style="@style/Widget.Material3.Button"
    android:layout_width="200dp"
    android:layout_height="60dp"
    android:layout_gravity="center"
    android:text="@string/reg_button">

```

```
</com.google.android.material.button.MaterialButton>
```

```
</LinearLayout>
```

Файл «appointment.xml»

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:id="@+id/main"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:padding="16dp"
tools:context=".presentation.view.WelcomeScreen">
<ImageButton
    android:id="@+id/btnBack"

style="@style/Widget.MaterialComponents.Button.OutlinedButton"
    android:layout_width="48dp"
    android:layout_height="48dp"
    android:layout_marginTop="16dp"
    android:layout_marginEnd="16dp"
    android:src="@drawable/outline_arrow_back_24"
    android:contentDescription="@string/back"

android:background="?attr/selectableItemBackgroundBorderless"
    android:elevation="2dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="0dp"
    app:layout_constraintBottom_toTopOf="@id/bookBtn"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:fontFamily="@font/inter_bold"
            android:layout_gravity="center"
            android:layout_marginTop="30dp"
            android:textSize="20sp"
            android:text="Запись"/>

        <TextView
```

					ПГУ1.090301.45.001 ПЗ	Лист
						102
Изм.	Лист	№ докум.	Подпись	Дата		

```

        android:id="@+id/sto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/inter_bold"
        android:layout_gravity="center"
        android:layout_marginTop="30dp"
        android:textSize="20sp"
        android:text="Мастер"/>

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/car_layout"

    style="@style/Widget.Material3.TextInputLayout.FilledBox.Exposed
    DropdownMenu"

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="10dp"
        android:hint="Выберите автомобиль">

        <AutoCompleteTextView
            android:id="@+id/car_dropdown"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="none" />

    </com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/comment_layout"

    style="@style/Widget.Material3.TextInputLayout.OutlinedBox"
        android:layout_gravity="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/comment_text"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Что случилось?" />

    </com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/date_time_layout"

    style="@style/Widget.Material3.TextInputLayout.OutlinedBox"
        android:layout_gravity="center"

```

```

        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Дата и время визита">

<com.google.android.material.textfield.TextInputEditText
        android:id="@+id/date_time"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:inputType="datetime"
        android:clickable="true"
        android:focusable="false" />

</com.google.android.material.textfield.TextInputLayout>
    </LinearLayout>
</ScrollView>
<com.google.android.material.button.MaterialButton
        android:id="@+id/bookBtn"
        style="@style/Widget.Material3.Button"
        android:layout_width="200dp"
        android:layout_height="60dp"
        android:layout_gravity="center"
        android:text="Записаться"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent">

</com.google.android.material.button.MaterialButton>
<ProgressBar
        android:id="@+id/progressBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:visibility="gone"
        tools:visibility="visible"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Файл «GarageFragment.kt»

```

package com.example.automaster.presentation.view

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Toast
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.LifecycleScope

```

```

import androidx.recyclerview.widget.LinearLayoutManager
import com.example.automaster.databinding.GarageBinding
import com.example.automaster.databinding.SheetAddCarBinding
import com.example.automaster.presentation.state.AddCarState
import com.example.automaster.presentation.state.GarageState
import
com.example.automaster.presentation.viewmodel.GarageViewModel
import com.google.android.material.bottomsheet.BottomSheetDialog
import
com.google.android.material.dialog.MaterialAlertDialogBuilder
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.launch

@AndroidEntryPoint
class GarageFragment : Fragment() {
    private val viewModel: GarageViewModel by viewModels()
    private lateinit var adapter : GarageAdapter

    private var addCarDialog: BottomSheetDialog? = null

    private var _binding: GarageBinding? = null
    private val binding get() = _binding!!
    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        _binding = GarageBinding.inflate(inflater, container,
false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.fabAddCar.setOnClickListener {
            showAddCarBottomSheet()
        }
        setupRecyclerView()
        observeViewModel()
    }

    private fun setupRecyclerView() {
        adapter = GarageAdapter(
            onDeleteClick = { car ->
                MaterialAlertDialogBuilder(requireContext())
                    .setTitle("Удалить авто")
                    .setMessage("Вы уверены, что хотите удалить
${car.brand} ${car.model} из гаража?")
                    .setPositiveButton("Удалить") { _, _ ->
                        viewModel.deleteCar(car.id)
                    }
            }
        )
    }
}

```

```

        }
        .setNegativeButton("Отмена", null)
        .show()
    }
)
binding.rvCars.layoutManager =
LinearLayoutManager(requireContext())
binding.rvCars.adapter = adapter
}
private fun showAddCarBottomSheet() {
    val dialogBinding =
SheetAddCarBinding.inflate(layoutInflater)
    addCarDialog = BottomSheetDialog(requireContext())
    addCarDialog?.setContentView(dialogBinding.root)
    dialogBinding.btnSaveCar.setOnClickListener {
        val vin = dialogBinding.etVin.text.toString()
        val brand = dialogBinding.etBrand.text.toString()
        val model = dialogBinding.etModel.text.toString()
        val year = dialogBinding.etYear.text.toString()
        viewModel.addCar(vin, brand, model, year)
    }
    addCarDialog?.setOnDismissListener {
viewModel.resetAddState() }
    addCarDialog?.show()
}
private fun observeViewModel() {
    viewLifecycleOwner.lifecycleScope.launch {
        viewModel.listState.collect { state ->
            when (state) {
                is GarageState.Success -> {
                    adapter.submitList(state.cars)
                }
                is GarageState.Error -> {
                    Toast.makeText(requireContext(),
state.message, Toast.LENGTH_SHORT).show()
                }
                is GarageState.Loading -> {
                }
            }
        }
    }

    viewLifecycleOwner.lifecycleScope.launch {
        viewModel.addState.collect { state ->
            when (state) {
                is AddCarState.Success -> {
                    addCarDialog?.dismiss()
                    Toast.makeText(requireContext(),
"Успешно!", Toast.LENGTH_SHORT).show()
                    viewModel.resetAddState()
                }
            }
        }
    }
}

```



```

        binding.tvCarBrandModel.text = "${car.brand}
        ${car.model}"
        binding.tvCarYear.text = "Год: ${car.year} г.в."
        binding.tvVin.text = "VIN: ${car.vin}"
        binding.btnDelete.setOnClickListener {
onDeleteClick(car) }
    }

    }

    class CarDiffCallback : DiffUtil.ItemCallback<Car>() {
        override fun areItemsTheSame(oldItem: Car, newItem: Car)
= oldItem.id == newItem.id
        override fun areContentsTheSame(oldItem: Car, newItem:
Car) = oldItem == newItem
    }
}

```

Файл «GarageViewModel.kt»

```

package com.example.automaster.presentation.viewmodel

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.automaster.domain.usecase.AddCarUseCase
import com.example.automaster.domain.usecase.DeleteCarUseCase
import com.example.automaster.domain.usecase.GetCarsUseCase
import com.example.automaster.presentation.state.AddCarState
import com.example.automaster.presentation.state.GarageState
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class GarageViewModel @Inject constructor(
    private val getCarsUseCase: GetCarsUseCase,
    private val addCarUseCase: AddCarUseCase,
    private val deleteCarUseCase: DeleteCarUseCase
) : ViewModel() {

    private val _listState =
MutableStateFlow<GarageState>(GarageState.Loading)
    val listState = _listState.asStateFlow()

    private val _addState =
MutableStateFlow<AddCarState>(AddCarState.Idle)
    val addState = _addState.asStateFlow()

    init {
        loadCars()
    }
}

```

```

fun loadCars() {
    viewModelScope.launch {
        _listState.value = GarageState.Loading
        getCarsUseCase().fold(
            onSuccess = { _listState.value =
GarageState.Success(it) },
            onFailure = { _listState.value =
GarageState.Error(it.message ?: "Ошибка") }
        )
    }
}

fun addCar(vin: String, brand: String, model: String,
yearString: String) {
    viewModelScope.launch {
        _addState.value = AddCarState.Loading
        val year = yearString.toIntOrNull() ?: 0

        addCarUseCase(vin, brand, model, year).fold(
            onSuccess = {
                _addState.value = AddCarState.Success
                loadCars()
            },
            onFailure = { _addState.value =
AddCarState.Error(it.message ?: "Ошибка") }
        )
    }
}

fun deleteCar(carId: Long) {
    viewModelScope.launch {
        _addState.value = AddCarState.Loading

        deleteCarUseCase(carId).fold(
            onSuccess = {
                _addState.value = AddCarState.Success
                loadCars()
            },
            onFailure = { _addState.value =
AddCarState.Error(it.message ?: "Ошибка") }
        )
    }
}

fun resetAddState() {
    _addState.value = AddCarState.Idle
}

```

Файл «GarageState.kt»

package com.example.automaster.presentation.state

					ПГУ1.090301.45.001 ПЗ	Лист
						109
Изм.	Лист	№ докум.	Подпись	Дата		

```

import com.example.automaster.domain.model.Car

sealed class GarageState {
    object Loading : GarageState()
    data class Success(val cars: List<Car>) : GarageState()
    data class Error(val message: String) : GarageState()
}

Файл «AddCarUseCase.kt»

package com.example.automaster.domain.usecase

import com.example.automaster.domain.repository.CarRepository
import javax.inject.Inject

class AddCarUseCase @Inject constructor(private val repository:
CarRepository) {
    suspend operator fun invoke(vin: String, brand: String,
model: String, year: Int): Result<Unit> {
        val cleanBrand = brand.trim()
        val cleanModel = model.trim()
        val cleanVin = vin.trim().uppercase()
        if (cleanBrand.isBlank() || cleanModel.isBlank() ||
cleanVin.isBlank() || year.toString().isEmpty()) {
            return Result.failure(Exception("Заполните поля"))
        }
        if (cleanVin.length != 17) {
            return Result.failure(Exception("VIN номер должен
содержать ровно 17 символов"))
        }
        val vinRegex = "[A-HJ-NPR-Z0-9]{17}$".toRegex()
        if (!cleanVin.matches(vinRegex)) {
            return Result.failure(Exception("Некорректный формат
VIN (используйте только латинские буквы и цифры, кроме I, O,
Q)"))
        }

        if (year < 1900 || year > 2026) {
            return Result.failure(Exception("Некорректный год
выпуска"))
        }
        return repository.addCar(vin, brand, model, year)
    }
}

class DeleteCarUseCase @Inject constructor(private val
repository: CarRepository) {
    suspend operator fun invoke(carId: Long): Result<Unit>{
        return repository.deleteCar(carId)
    }
}

Файл «CarRepository.kt»

```

```

package com.example.automaster.domain.repository
import com.example.automaster.domain.model.Car
interface CarRepository {
    suspend fun getCars(): Result<List<Car>>
    suspend fun addCar(vin: String, brand: String, model:
String, year: Int): Result<Unit>
    suspend fun deleteCar(carId: Long): Result<Unit>
}

```

Файл «CarRepositoryImpl.kt»

```

package com.example.automaster.data.repository

import com.example.automaster.data.api.CarApi
import com.example.automaster.domain.model.AddCarRequest
import com.example.automaster.domain.model.Car
import com.example.automaster.domain.repository.CarRepository
import javax.inject.Inject

class CarRepositoryImpl @Inject constructor(private val carApi:
CarApi): CarRepository {
    override suspend fun getCars(): Result<List<Car>> {
        return try {
            val response = carApi.getMyCars()
            if (response.isSuccessful)
Result.success(response.body() ?: emptyList())
            else Result.failure(Exception("Ошибка:
${response.code()}"))
        } catch (e: Exception) { Result.failure(Exception("Нет
сети")) }
    }

    override suspend fun addCar(vin: String, brand: String,
model: String, year: Int): Result<Unit> {
        return try {
            val response = carApi.addCar(AddCarRequest(vin,
brand, model, year))
            if (response.isSuccessful) Result.success(Unit)
            else Result.failure(Exception("Не удалось добавить
авто"))
        } catch (e: Exception) { Result.failure(Exception("Нет
сети")) }
    }

    override suspend fun deleteCar(carId: Long): Result<Unit>{
        return try {
            val response = carApi.deleteCar(carId)
            if (response.isSuccessful) Result.success(Unit)
            else Result.failure(Exception("Не удалось удалить
авто"))
        } catch (e: Exception) { Result.failure(Exception("Нет
сети")) }
    }
}

```

```
}
```

Файл «AuthController.kt»

```
package com.example.autoapp.controller

import com.example.autoapp.dto.AuthResponse
import com.example.autoapp.dto.SignUpRequest
import com.example.autoapp.dto.SignInRequest
import com.example.autoapp.service.AuthService
import org.springframework.http.ResponseEntity
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
@RequestMapping("/api/auth")
class AuthController(private val authService: AuthService) {
    @PostMapping("/register")
    fun signUp(@RequestBody request: SignUpRequest):
    ResponseEntity<AuthResponse> {
        val response = authService.signUp(request)
        return ResponseEntity.ok(response)
    }
    @PostMapping("/login")
    fun signIn(@RequestBody request: SignInRequest):
    ResponseEntity<AuthResponse>{
        val response = authService.signIn(request)
        return ResponseEntity.ok(response)
    }
}
```

Файл «UserService.kt»

```
package com.example.autoapp.service

import com.example.autoapp.dto.CarResponse
import com.example.autoapp.dto.ProfileDto
import com.example.autoapp.repository.UserRepository
import jakarta.transaction.Transactional
import org.springframework.data.repository.findByIdOrNull
import org.springframework.stereotype.Service

@Service
class UserService(
    private val userRepository: UserRepository
){
    fun getProfile(userId: Long): ProfileDto {
        val user = userRepository.findByIdOrNull(userId)
        ?: throw IllegalArgumentException("Пользователь с ID
$userId не найден")
        return ProfileDto(
            email = user.email,
```

					ПГУ1.090301.45.001 ПЗ	Лист
						112
Изм.	Лист	№ докум.	Подпись	Дата		

```

        surname = user.surname,
        name = user.name,
        patronymic = user.patronymic,
        number = user.phoneNumber
    )
}

@Transactional
fun updateProfile(profile: ProfileDto): ProfileDto {
    val user = userRepository.findByEmail(profile.email)
        ?: throw IllegalArgumentException("Пользователь с ID
${profile.email} не найден")
    user.surname = profile.surname
    user.name = profile.name
    user.patronymic = profile.patronymic
    user.phoneNumber = profile.number
    val updatedUser = userRepository.save(user)
    return ProfileDto(
        updatedUser.email,
        updatedUser.surname,
        updatedUser.name,
        updatedUser.patronymic,
        updatedUser.phoneNumber
    )
}
}

```

Файл «UserRepository.kt»

```

@Repository
interface UserRepository: JpaRepository<User, Long> {
    fun existsByEmail(email: String): Boolean
    fun findByEmail(email: String): User?
}

```

Файл «User.kt»

```

package com.example.autoapp.domain

import com.example.autoapp.dto.UserRole
import jakarta.persistence.Column
import jakarta.persistence.Entity
import jakarta.persistence.EnumType
import jakarta.persistence.Enumerated
import jakarta.persistence.GeneratedValue
import jakarta.persistence.GenerationType
import jakarta.persistence.Id
import jakarta.persistence.Table
import org.springframework.security.core.userdetails.UserDetails

@Entity
@Table(name = "users")
class User(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)

```

					ПГУ1.090301.45.001 ПЗ	Лист
						113
Изм.	Лист	№ докум.	Подпись	Дата		

```

        var id: Long? = null,
        @Column(unique = true, nullable = false)
        var email: String,
        @Column(nullable = false)
        var surname: String,
        @Column(nullable = false)
        var name: String,
        @Column
        var patronymic: String,
        @Column(nullable = false)
        var passwordHash: String?,
        @Enumerated(EnumType.STRING)
        var role: UserRole,
        @Column(name = "phone_number", nullable = false)
        var phoneNumber: String
    )

```

Файл «AppointmentController.kt»

```

package com.example.autoapp.controller

import com.example.autoapp.config.CustomUserDetails
import com.example.autoapp.dto.AppointmentDto
import com.example.autoapp.dto.AppointmentStatus
import com.example.autoapp.dto.BookingRequest
import com.example.autoapp.service.AppointmentService
import org.springframework.format.annotation.DateTimeFormat
import org.springframework.http.ResponseEntity
import org.springframework.security.core.annotation.AuthenticationPrincipal
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.PathVariable
import org.springframework.web.bind.annotation.PostMapping
import org.springframework.web.bind.annotation.PutMapping
import org.springframework.web.bind.annotation.RequestBody
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RequestParam
import org.springframework.web.bind.annotation.RestController
import java.time.LocalDate

@RestController
@RequestMapping("/api/appointments")
class AppointmentController(private val appointmentService:
AppointmentService) {
    @PostMapping("/appointment")
    fun appointment(@AuthenticationPrincipal userDetails:
CustomUserDetails,
                    @RequestBody request:
BookingRequest): ResponseEntity<Unit> {
        appointmentService.createAppointment(userDetails.id,
request)
        return ResponseEntity.ok().build()
    }
}

```

					ПГУ1.090301.45.001 ПЗ	Лист
						114
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }
    @GetMapping("/appointment/timeslots")
    fun getTimeSlots(
        @RequestParam masterId: Long,
        @RequestParam @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE_TIME) date: LocalDate,
    ): ResponseEntity<List<String>>{
        val slots = appointmentService.getTimeSlots(masterId,
date)
        return ResponseEntity.ok(slots)
    }
    @GetMapping("appointment/blocked-dates")
    fun getBlockedDates(@RequestParam masterId: Long):
ResponseEntity<List<String>> {
        val unavailableDates =
appointmentService.getUnavailableDates(
            masterId = masterId,
            startFrom = LocalDate.now(),
            monthsAhead = 3
        )
        return ResponseEntity.ok(unavailableDates)
    }

    @GetMapping("/client")
    fun getClientAppointments(@AuthenticationPrincipal
userDetails: CustomUserDetails):
ResponseEntity<List<AppointmentDto>> {
        return
ResponseEntity.ok(appointmentService.getClientAppointments(userD
etails.id))
    }

    @GetMapping("/master")
    fun getMasterAppointments(@AuthenticationPrincipal
userDetails: CustomUserDetails):
ResponseEntity<List<AppointmentDto>> {
        return
ResponseEntity.ok(appointmentService.getMasterAppointments(userD
etails.id))
    }

    @PutMapping("/{id}/cancel")
    fun cancelAppointment(@PathVariable id: Long):
ResponseEntity<Void> {
        appointmentService.changeStatus(id,
AppointmentStatus.CANCELLED)
        return ResponseEntity.ok().build()
    }

    @PutMapping("/{id}/status")
    fun updateStatus(

```

					ПГУ1.090301.45.001 ПЗ	Лист
						115
Изм.	Лист	№ докум.	Подпись	Дата		

```

        @PathVariable id: Long,
        @RequestParam status: AppointmentStatus
    ): ResponseEntity<Void> {
        appointmentService.changeStatus(id, status)
        return ResponseEntity.ok().build()
    }
}

```

Файл «AppointmentRepository.kt»

```

package com.example.autoapp.repository

import com.example.autoapp.domain.Appointment
import org.springframework.data.jpa.repository.JpaRepository
import org.springframework.stereotype.Repository
import java.time.LocalDate

@Repository
interface AppointmentRepository: JpaRepository<Appointment,
Long> {
    fun findByMasterIdAndDate(masterId: Long, date: LocalDate):
List<Appointment>
    fun findByClientId(clientId: Long): List<Appointment>

    fun findByMasterId(masterId: Long): List<Appointment>
}

```

Файл «AppointmentService.kt»

```

package com.example.autoapp.service

import com.example.autoapp.domain.Appointment
import com.example.autoapp.dto.AppointmentDto
import com.example.autoapp.repository.ReviewRepository
import com.example.autoapp.dto.AppointmentStatus
import com.example.autoapp.dto.BookingRequest
import com.example.autoapp.dto.BookingResponse
import com.example.autoapp.repository.AppointmentRepository
import com.example.autoapp.repository.CarRepository
import com.example.autoapp.repository.MasterRepository
import
com.example.autoapp.repository.ScheduleExceptionRepository
import com.example.autoapp.repository.ScheduleTemplateRepository
import com.example.autoapp.repository.UserRepository
import jakarta.transaction.Transactional
import org.springframework.data.repository.findByIdOrNull
import org.springframework.stereotype.Service
import java.time.LocalDate
import java.time.LocalDateTime
import java.time.LocalTime
import java.time.format.DateTimeFormatter
import java.util.Locale

```

```

@Service
class AppointmentService(
    private val appointmentRepository: AppointmentRepository,
    private val userRepository: UserRepository,
    private val carRepository: CarRepository,
    private val masterRepository: MasterRepository,
    private val reviewRepository: ReviewRepository,
    private val templateRepository: ScheduleTemplateRepository,
    private val exceptionRepository: ScheduleExceptionRepository
) {
    fun createAppointment(clientId: Long, request:
BookingRequest): BookingResponse {
        val car = carRepository.findById(request.carId)
            .orElseThrow{ RuntimeException("Машина не найдена")
        }

        val localDate = java.time.LocalDate.parse(request.date)
        val localTime =
java.time.LocalTime.parse(request.timeSlot)
        val appointment = Appointment(
            client = userRepository.getReferenceById(clientId),
            master =
masterRepository.getReferenceById(request.masterId),
            car = car,
            date = localDate,
            timeSlot = localTime,
            comment = request.comment,
            status = AppointmentStatus.CONFIRMED
        )
        val saved = appointmentRepository.save(appointment)
        return BookingResponse(saved.id!!, "Вы успешно
записались в автосервис", saved.status)
    }

    fun getTimeSlots(masterId: Long, date: LocalDate):
List<String>{
        val isException =
exceptionRepository.existsByMasterIdAndExceptionDate(masterId,
date)

        if(isException){
            return emptyList()
        }

        val dayOfWeek = date.dayOfWeek.value
        val template =
templateRepository.findByMasterIdAndDayOfWeek(masterId,
dayOfWeek)

        if (template == null || !template.isWorking ||
template.startTime == null || template.endTime == null) {
            return emptyList()
        }

        val allPossibleSlots = mutableListOf<LocalTime>()
        var currentSlot = template.startTime!!
    }

```

```

        while (currentSlot.plusHours(1) <= template.endTime) {
            allPossibleSlots.add(currentSlot)
            currentSlot = currentSlot.plusHours(1)
        }

        val existingAppointments =
appointmentRepository.findByMasterIdAndDate(masterId, date)
        val occupiedTimes = existingAppointments.map {
it.timeSlot }

        val availableSlots = allPossibleSlots.filter { slot ->
!occupiedTimes.contains(slot)
        }

        return availableSlots.map { it.toString() }
    }

    private fun formatDateTime(date: LocalDate, time:
LocalTime): String {
        val formatter = DateTimeFormatter.ofPattern("d MMMM,
HH:mm", Locale("ru"))
        return LocalDateTime.of(date, time).format(formatter)
    }

    fun getUnavailableDates(masterId: Long, startFrom:
LocalDate, monthsAhead: Long): List<String> {
        val unavailableDates = mutableListOf<String>()
        val endDate = startFrom.plusMonths(monthsAhead)

        var checkDate = startFrom
        while (!checkDate.isAfter(endDate)) {
            val availableSlots = getTimeSlots(masterId,
checkDate)

            if (availableSlots.isEmpty()) {
                unavailableDates.add(checkDate.toString())
            }
            checkDate = checkDate.plusDays(1)
        }
        return unavailableDates
    }

    fun getClientAppointments(clientId: Long):
List<AppointmentDto> {
        val appointments =
appointmentRepository.findByClientId(clientId)

        return parseAppointment(appointments)
    }

```

```

        fun getMasterAppointments(masterId: Long):
List<AppointmentDto> {
            val appointments =
appointmentRepository.findByMasterId(masterId)

            return parseAppointment(appointments)
        }

        fun parseAppointment(appointments: List<Appointment>):
List<AppointmentDto> {
            val appointmentsDto = appointments.map { appointment ->
                val hasReview =
reviewRepository.existsByAppointmentId(appointment.id ?: 0L)

                AppointmentDto(
                    id = appointment.id ?: 0L,
                    clientId = appointment.client.id ?: 0L,
                    masterId = appointment.master.id,
                    date = formatDateTime(appointment.date,
appointment.timeSlot),
                    status = appointment.status.name,
                    masterName = appointment.master.name,
                    masterAddress = appointment.master.address,
                    clientName = "${appointment.client.surname}
${appointment.client.name} (${appointment.client.patronymic})",
                    comment = appointment.comment,
                    clientPhone = appointment.client.phoneNumber,

                    carDetails = "${appointment.car.brand}
${appointment.car.model} (${appointment.car.vin})",
                    hasReview = hasReview
                )
            }
            return appointmentsDto
        }
        @Transactional
        fun changeStatus(id: Long, status: AppointmentStatus) {
            val appointment =
appointmentRepository.findByIdOrNull(id)
                ?: throw IllegalArgumentException("Запись с ID $id
не найдена")

            appointment.status = status

            appointmentRepository.save(appointment)
        }
    }

    Файл «Appointment.kt»

package com.example.autoapp.domain

```

```

@Entity
@Table(name = "appointments")
class Appointment (
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "client_id", nullable = false)
    val client: User,
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "master_id", nullable = false)
    val master: MasterProfile,
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "car_id", nullable = false)
    val car: Car,
    val date: LocalDate,
    val timeSlot: LocalTime,
    val comment: String,
    @Enumerated(EnumType.STRING)
    var status: AppointmentStatus
)

```

Файл «AppointmentDto.kt»

```

package com.example.autoapp.dto

data class AppointmentDto(
    val id: Long,
    val masterId: Long,
    val clientId: Long,
    val date: String,
    val status: String,
    val masterName: String,
    val masterAddress: String,
    val clientName: String,
    val comment: String,
    val clientPhone: String,
    val carDetails: String,
    val hasReview: Boolean = false
)

```

Файл «AppointmentStatus.kt»

```

package com.example.autoapp.dto

enum class AppointmentStatus {
    CONFIRMED,
    CANCELLED,
    PENDING,
    COMPLETED
}

```

Файл «CarService.kt»

```

package com.example.autoapp.service

```

					ПГУ1.090301.45.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		120

```

import com.example.autoapp.domain.Car
import com.example.autoapp.domain.User
import com.example.autoapp.dto.AddCarRequest
import com.example.autoapp.dto.CarResponse
import com.example.autoapp.repository.CarRepository
import org.springframework.stereotype.Service

@Service
class CarService(
    private val carRepository: CarRepository
) {
    fun addCar(clientId: Long, request: AddCarRequest):
    CarResponse{
        val newCar = Car(
            ownerId = clientId,
            vin = request.vin,
            brand = request.brand,
            model = request.model,
            year = request.year
        )
        val saved = carRepository.save(newCar)
        return CarResponse(saved.id, saved.vin, saved.brand,
        saved.model, saved.year)
    }

    fun getCars(clientId: Long): List<CarResponse>{
        return carRepository.findAllByOwnerId(clientId).map{
            CarResponse(it.id!!, it.vin, it.brand, it.model,
it.year)
        }
    }

    fun deleteCar(carId: Long){
        carRepository.deleteById(carId)
    }
}

```

Файл «CarController.kt»

```

package com.example.autoapp.controller

@RestController
@RequestMapping("/api/car")
class CarController(private val carService: CarService){
    @PostMapping
    fun addCar(
        @AuthenticationPrincipal userDetails: CustomUserDetails,
        @RequestBody request: AddCarRequest):
    ResponseEntity<CarResponse> {
        return
    ResponseEntity.ok(carService.addCar(userDetails.id, request))
    }
}

```

					ПГУ1.090301.45.001 ПЗ	Лист
						121
Изм.	Лист	№ докум.	Подпись	Дата		

```

    @GetMapping
    fun getCar(@AuthenticationPrincipal userDetails:
CustomUserDetails): ResponseEntity<List<CarResponse>> {
        return
ResponseEntity.ok(carService.getCars(userDetails.id))
    }

    @DeleteMapping
    fun deleteCar(@RequestParam carId: Long):
ResponseEntity<Unit>{
        carService.deleteCar(carId)
        return ResponseEntity.ok().build()
    }
}

```

Файл «JwtService.kt»

```

package com.example.autoapp.config

import com.example.autoapp.domain.User
import io.jsonwebtoken.Claims
import io.jsonwebtoken.Jwts
import io.jsonwebtoken.security.Keys
import org.springframework.beans.factory.annotation.Value
import org.springframework.security.core.userdetails.UserDetails
import org.springframework.stereotype.Service
import java.util.Date

@Service
class JwtService(
    @Value("\${jwt.secret}") private val secret: String,
    @Value("\${jwt.expiration}") private val expiration: Long
) {
    private val key = Keys.hmacShaKeyFor(secret.toByteArray())
    fun generateToken(id: Long, email: String, role: String):
String{
        val extraClaims = mapOf(
            "id" to id,
            "role" to role
        )
        return buildToken(extraClaims, email, expiration)
    }

    private fun buildToken(extraClaims: Map<String, Any>,
subject: String, expirationTime: Long): String{
        return Jwts.builder()
            .claims(extraClaims)
            .subject(subject)
            .issuedAt(Date())
            .expiration(Date(System.currentTimeMillis() +
expirationTime))
            .signWith(key)
    }
}

```

					ПГУ1.090301.45.001 ПЗ	Лист
						122
Изм.	Лист	№ докум.	Подпись	Дата		

```

        .compact()
    }

    fun extractEmail(token: String): String?{
        return extractClaim(token, Claims::getSubject)
    }

    fun extractId(token: String): Long?{
        val claims = extractAllClaims(token)
        return claims["id"]?.toString()?.toLongOrNull()
    }

    fun extractRole(token: String): String?{
        val claims = extractAllClaims(token)
        return claims["role"]?.toString()
    }

    fun validateToken(token: String, userDetails: UserDetails):
Boolean{
        val email = extractEmail(token)
        return (email == userDetails.username) &&
!isTokenExpired(token)
    }

    private fun <T> extractClaim(token: String, claimsResolver:
(Claims) -> T): T {
        val claims = extractAllClaims(token)
        return claimsResolver(claims)
    }

    private fun extractAllClaims(token: String): Claims{
        return Jwts.parser()
            .verifyWith(key)
            .build()
            .parseSignedClaims(token)
            .payload
    }

    private fun isTokenExpired(token: String): Boolean{
        return extractExpiration(token).before(Date())
    }

    private fun extractExpiration(token: String): Date{
        return extractClaim(token, Claims::getExpiration)
    }
}

```

Приложение В

(Обязательно)

Разработка мобильного приложения для записи в автосервис

Презентация

					ПГУ1.090301.45.001 ПЗ	Лист
						124
Изм.	Лист	№ докум.	Подпись	Дата		

Разработка мобильного приложения для записи в автосервис

Автор работы: студент группы 22ВВП2 Самофалова А.В.

Научный руководитель: доцент кафедры ВТ Кузнецов А.В.

Актуальность работы

В настоящее время автомобили являются одним из наиболее распространенных видов транспорта, а средний возраст легковых автомобилей в России составляет 15,5 лет, что является причиной большого спроса на услуги технического обслуживания и ремонта транспортных средств.

Несмотря на то, что наше время происходит цифровизация сфер медицины, доставки, такси, в сфере автосервиса запись часто по-прежнему ведется по телефону. Такой способ записи оказывается неудобным вариантом для клиентов из-за необходимости ожидания ответа, отсутствия информации о доступном времени записи, соответственно этот процесс расходует много времени.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.45.001 ПЗ

Лист

125

Цели работы

Цель работы - разработать мобильное приложение, позволяющее автоматизировать и оптимизировать процесс записи клиентов на техническое обслуживание в автосервис.

В приложении должно быть реализовано:

- регистрация и аутентификация с разграничением ролей «Клиент» и «Мастер»;
- запись на обслуживание и возможность отмены записи;
- управление гаражом пользователя и ведение истории ремонтов;
- отправка отзывов по 5-балльной шкале и пересчет рейтинга мастера на сервере при добавлении новых оценок;
- автоматическая проверка доступности мастера в выбранный промежуток времени на стороне сервера;
- для роли "Мастер" – управление статусами заказов и личным рабочим графиком

Обзор аналогов

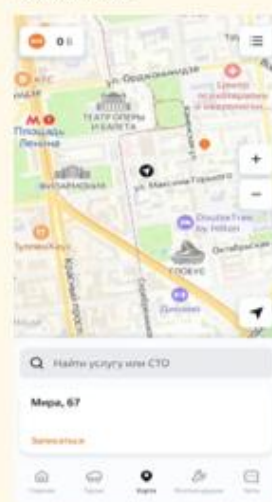
Достоинства:

- удобная онлайн-запись;
- возможность описания проблемы в чате с поддержкой;
- поиск автосервисов на карте;
- история обслуживания;
- возможность отслеживания каждого этапа ремонта.

Недостатки:

- приложение создано для конкретной сети автосервисов;
- коммуникация осуществляется только через операторов поддержки;
- отсутствует возможность выбора конкретного мастера.

FIT SERVICE



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.45.001 ПЗ

Лист

126

Обзор аналогов

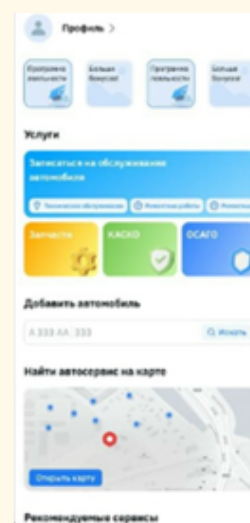
Достоинства:

- умный подбор СТО с помощью интеллектуального помощника;
- отображение автосервисов и цен на карте;
- возможность отслеживания этапов ремонта;
- чат с СТО;
- история обслуживания.

Недостатки:

- отсутствует возможность выбора конкретного мастера;
- при создании заявки можно выбрать только ориентировочное время записи на услугу;
- перегруженный интерфейс приложения из-за большого количества услуг, таких как оформление страхового полиса и получение предложений от страховых компаний.

UREMONT



Обзор аналогов

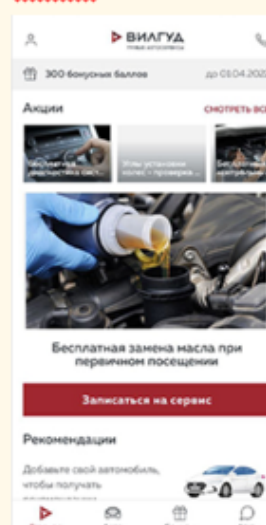
Достоинства:

- поиск автосервиса с помощью карт;
- удобная онлайн запись в автосервис с выбором даты и времени;
- хранение истории обслуживания автомобиля;
- отображение статуса ремонта.

Недостатки:

- приложение работает только с конкретной сетью автосервисов;
- невозможность выбрать мастера, у которого клиент хотел бы обслуживаться.

Wilgood



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.45.001 ПЗ

Лист

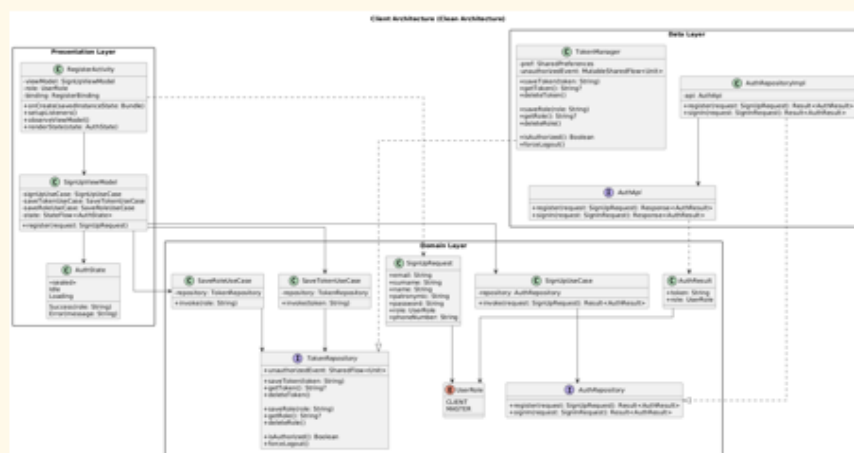
127

Стек технологий

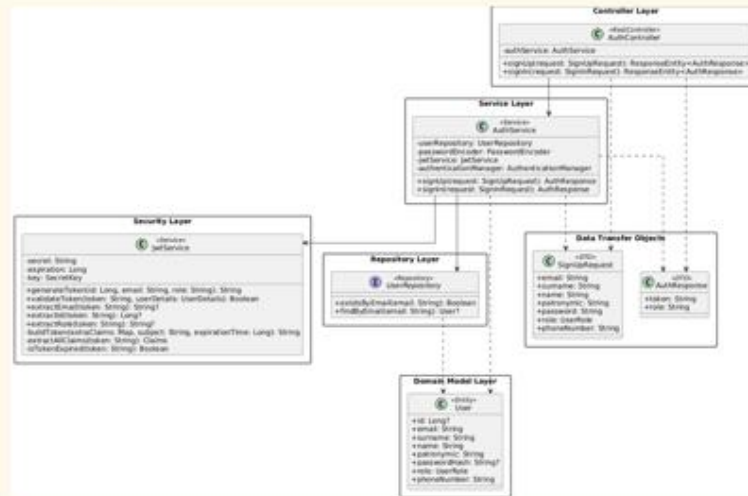


Spring **Boot**

UML-диаграмма классов



UML-диаграмма классов



UML-диаграмма вариантов использования



Приложение для записи в автосервис



Заключение

В результате проделанной работы был успешно разработан и протестировано мобильное приложение, полностью соответствующее заданию.

Разработанное приложение успешно решает поставленную задачу, повышая удобство и взаимодействия между автовладельцами и исполнителями.

В перспективе развития проекта планируется: интеграция интерактивной карты для поиска ближайших мастерских, внедрение push-уведомлений для информирования клиента о стадиях ремонта.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.45.001 ПЗ

Лист

130