

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА
на тему
ЦИФРОВАЯ ПЛАТФОРМА АГРЕГАЦИИ
УСЛУГ ДОРОЖНОГО СЕРВИСА И
ТЕХНИЧЕСКОЙ ПОДДЕРЖКИ
АВТОВЛАДЕЛЬЦЕВ

Студент _____ Ахтямов Адель Фаилевич _____
(подпись, дата) (ФИО полностью)

Руководитель _____ Генералова А.А. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В. _____
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г. № 12)

Заведующий кафедрой _____ Козлов А.Ю. _____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А. _____
(подпись) (фамилия, инициалы)

Пенза, 2026

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

Факультет
вычислительной техники

Кафедра
«Математическое обеспечение
и применение ЭВМ»

«Утверждаю»
Заведующий кафедрой
_____ Козлов А.Ю.
« ____ » _____ 20 ____ г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Ахтямов Адель Фаилевич гр. 22ВП2 факультета вычислительной техники

2. Тема работы. Цифровая платформа агрегации услуг дорожного сервиса и технической поддержки автовладельцев.

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025 г.

3. Руководитель бакалаврской работы к.н., доцент кафедры МОиПЭВМ ПГУ Генералова А.А.,

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

Разработать мобильное приложение для операционной системы Android и консольное серверное приложение для операционной системы Windows, представляющие собой цифровую платформу для агрегации услуг дорожного сервиса и технической поддержки автовладельцев. Язык программирования Java, среда разработки Android Studio и IntelliJ IDEA. Программа должна обладать интуитивно понятным интерфейсом и использовать следующие технологии: SharedPreferences, Private Config Files, асинхронное программирование, Material Design 3.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки мобильного приложения. 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	45	06.02.2026 – 27.02.2026	
2	Анализ требований	35	28.02.2026 – 13.03.2026	
3	Выбор технологий, применяемых при проектировании и разработке	55	14.03.2026 – 30.03.2026	
4	Создание графического интерфейса	65	31.03.2026 – 08.04.2026	
5	Разработка мобильного приложения	84	09.04.2026 – 15.05.2026	
6	Оформление пояснительной записки	40	16.05.2026 – 30.05.2026	

Дата выдачи задания "26" декабря 2025 г.

Руководитель бакалаврской работы _____ 26.12.2025 г. Генералова А.А.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025 г. Ахтямов А. Ф.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)

Реферат

Пояснительная записка содержит 120 листов, 30 рисунков, 4 таблицы и 22 использованных источников.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, НАСТОЛЬНОЕ, ПРИЛОЖЕНИЕ, ТЕСТИРОВАНИЕ, SHARED PREFERENCES, PRIVATE CONFIG FILES, ANDROID STUDIO, JAVA, ANDROID

Цель работы – разработать цифровую платформу для агрегации услуг дорожного сервиса и технической поддержки автовладельцев.

Мобильное приложение должно позволять пользователям создавать заявки на эвакуацию или техническую помощь с указанием своей геолокации, просматривать статус выполнения заявки, а исполнителям – принимать заказы и управлять их статусами. Помимо базового функционала создания и обработки заявок, необходимо реализовать систему уведомлений о новых заказах и изменении их статусов в реальном времени.

В конечном итоге было разработано консольное серверное приложение для Windows и мобильное приложение для Android на языке Java, которые удовлетворяют всем поставленным задачам. Была реализована регистрация и сохранение данных пользователей с помощью SharedPreferences и Private Config Files. Само приложение работает на устройствах ОС Android и Windows и разрабатывалось в среде разработки Android Studio и IntelliJ IDEA.

					ПГУ 09.03.04 – 08ВР222.01 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата					
Разраб.		Ахтямов А.Ф.			Цифровая платформа агрегации услуг дорожного сервиса и технической поддержки автовладельцев	Лит.	Лист	Листов	
Провер.		Генералова А.А.					4	120	
Н. Контр		Такташкин Д.В.				гр. 22ВП2			
Утверд.		Козлов А.Ю.							

Содержание

Введение.....	7
1 Приложение «ДорПомощь » для ОС Android	9
1.1 Анализ темы	9
1.2 Анализ аналогов.....	11
1.2.1 Приложение «АвтоСправочник».....	11
1.2.2 Социальная сеть «Drive2».....	13
1.2.3 Сервис «Auto.ru»	15
1.2.4 Обзор сервиса «Evacuator.ru» для вызова эвакуаторов.....	18
1.2.5 Результат сравнения программ-аналогов	20
1.3 Постановка задачи	22
2 Выбор инструментов разработки	24
2.1 Выбор языка программирования.....	24
2.1.1 Язык программирования «Java».....	24
2.1.2 Язык программирования «Kotlin».....	25
2.1.3 Язык программирования «C++».....	25
2.1.4 Обоснование выбора языка программирования	25
2.2 Выбор среды разработки.....	25
2.3 Выбор базы данных	26
3 Архитектура приложения.....	27
3.1 Общие принципы построения системы	27
3.2 Архитектура серверной части.....	28
3.3 Архитектура клиентской части	32
3.4 Протокол взаимодействия.....	36
3.5 База данных.....	37
3.6 Паттерны проектирования	39
3.7 Обеспечение надёжности и безопасности	39
3.8 Итоги архитектурного проектирования.....	40
4 Реализация приложения	41
4.1 Общая организация программного комплекса	41

4.2 Планирование разработки и оценка бюджета	42
4.3 Реализация серверной части	45
4.4 Реализация клиентской части	47
4.5 Тестирование программного комплекса.....	51
4.6 Оценка полученных результатов.....	53
5 Тестирование	54
5.1 Регистрация и авторизация пользователей	54
5.2 Управление автомобилями.....	56
5.3 Ведение журнала технического обслуживания	59
5.4 Отправка команд и сетевое взаимодействие.....	61
5.5 Тестирование офлайн-режима.....	63
5.6 Тестирование работы сервера в многопоточной среде.....	64
5.7 Анализ результатов тестирования.....	65
Заключение	67
Список использованных источников	69
Приложение А (справочное) Глоссарий	71
Приложение Б (обязательное) Листинг программы.....	73

Введение

В современном мире автомобиль давно перестал быть роскошью, став для большинства людей не просто средством передвижения, а необходимым инструментом в работе, путешествиях и повседневной жизни. Однако вместе с удобством эксплуатации приходит и ответственность за техническое состояние транспортного средства, а также необходимость быть готовым к нештатным ситуациям на дороге. Поломка в пути, особенно вдали от крупных городов или в ночное время, способна не только испортить планы, но и создать реальную угрозу безопасности. В такие моменты ключевыми становятся не столько наличие денег на эвакуатор, сколько доступ к правильной информации, понятные инструкции и спокойная оценка ситуации.

Особенность большинства существующих решений заключается в том, что они либо ориентированы исключительно на вызов профессиональной помощи (эвакуаторы, техпомощь), либо представляют собой разрозненные форумы и сообщества автолюбителей, где нужную информацию приходится выискивать часами. При этом в критической ситуации у водителя часто нет времени на долгие поиски: ему нужно быстро понять, можно ли устранить неполадку своими силами, как это правильно сделать и куда обратиться, если самостоятельный ремонт невозможен. Особенно остро эта проблема стоит для владельцев автомобилей, поставляемых на российский рынок с недостаточно проработанной документацией, где штатные руководства по эксплуатации могут быть переведены некачественно, содержать сложные технические формулировки или попросту отсутствовать в удобном для чтения с телефона формате.

Цифровая платформа, предлагаемая в рамках данной выпускной квалификационной работы, призвана решить эту проблему комплексно. В отличие от узкоспециализированных сервисов вызова эвакуаторов или служб технической помощи, наше приложение делает ставку на то, чтобы водитель мог справиться с поломкой сам. Внутри есть понятные инструкции по самым

частым неисправностям, написанные не техническим языком, а простыми словами. Также приложение напоминает о важных сроках: когда менять расходники или продлевать страховку. И еще там есть своя база знаний по конкретным маркам и моделям машин, которую сами пользователи наполняют и пополняют.

Если проблема сложная и решить ее самому не получается, тогда приложение предлагает внешние сервисы. Например, оно показывает ближайшие автосервисы с телефонами и адресами. А в будущем еще можно будет вызвать эвакуатор или мобильного мастера и отслеживать, где он находится в пути.

Отдельная важная штука – это работа с психологией водителя в аварийной ситуации. Как мы знаем из реальной жизни, когда машина ломается в дороге, часто начинается паника или ссоры с пассажирами. Это мешает трезво думать. Поэтому мы рассматриваем идею добавить в приложение офлайн-нейросеть. Она сможет в стрессовый момент спокойно подсказать, что делать, поддержит добрым словом или просто поговорит, чтобы снять напряжение и помочь человеку сосредоточиться.

Само приложение пишется под Android в среде Android Studio на Java. Серверная часть – это отдельное консольное приложение под Windows, она создается в IntelliJ IDEA тоже на Java. Сервер нужен для хранения данных пользователей, синхронизации информации о сервисах и для общения внутри сообщества. Клиент с сервером общаются по сокетам. Но даже без интернета почти всё работает: инструкции, напоминания, база знаний. Это важно, потому что за городом связь часто неустойчивая.

При этом закладывается не только базовый функционал. Архитектура проекта спроектирована так, чтобы в будущем можно было добавлять новые возможности без переписывания всего с нуля. Например, интеграцию с картами, чтобы прокладывать маршрут до ближайшего сервиса. Всё это можно будет добавить позже, потому что основа сделана гибко и аккуратно.

1 Приложение «ДорПомощь » для ОС Android

1.1 Анализ темы

Современный автовладелец сталкивается с целым спектром вызовов, выходящих далеко за рамки собственно вождения. Автомобиль требует регулярного технического обслуживания, своевременной замены расходных материалов, продления страховых полисов и, конечно, готовности к неожиданным поломкам в самых неподходящих местах и время. При этом уровень технической подготовки водителей варьируется крайне широко: от тех, кто способен самостоятельно заменить двигатель, до тех, кто не уверен, как открыть капот или где находится запаска.

Особенно остро проблема нехватки доступной и понятной информации проявляется в ситуациях, когда автомобиль ломается вдали от крупных городов. В таких условиях интернет может отсутствовать или быть крайне нестабильным. Штатные руководства по эксплуатации многих автомобилей, особенно официально поставляемых на российский рынок, зачастую страдают от некачественного перевода, изобилуют сложными техническими терминами и плохо адаптированы для чтения с экрана смартфона [1]. Владельцы автомобилей, ввезённых по параллельному импорту, могут и вовсе не иметь бумажной инструкции на русском языке, сталкиваясь с необходимостью искать и скачивать её в интернете, что в условиях слабого сигнала становится отдельной проблемой.

Существующие решения закрывают лишь отдельные фрагменты этой картины. Сервисы вызова эвакуаторов и технической помощи ориентированы исключительно на подключение профессионалов, хотя многие поломки можно устранить на месте своими силами, имея под рукой правильную инструкцию. Форумы и сообщества автолюбителей содержат огромные массивы полезной информации, но их структура неудобна для оперативного поиска: нужную тему приходится выискивать среди сотен сообщений, а качество советов варьируется от экспертных до откровенно вредных. Приложения—

уведомители о техническом обслуживании обычно привязаны к конкретным сервисным центрам и не дают пользователю самостоятельной информации. Наконец, приложения для чтения документации часто не оптимизированы для работы с техническими руководствами.

Таким образом, существует объективная потребность в едином приложении–компаньоне, которое объединит несколько ключевых функций. Во–первых, база структурированных инструкций для самостоятельного устранения типовых неисправностей. Во–вторых, система напоминаний о критических событиях: замене масла, ремня газораспределительного механизма, тормозных колодок, а также о сроках действия страховки. В–третьих, доступ к технической документации в удобном для телефона формате. В–четвёртых, сообщество владельцев схожих моделей. И только если самостоятельное решение невозможно, приложение должно предложить внешние сервисы: показать ближайшие автосервисы, а в перспективе – дать возможность вызвать эвакуатор или мобильного мастера.

Клиентская часть разрабатывается на Java с использованием Android Studio. Обе части системы используют один язык программирования, что упрощает взаимодействие между ними и позволяет переиспользовать общие модули и структуры данных. Серверная часть реализуется как консольное приложение для Windows в IntelliJ IDEA также на Java, что позволяет организовать надёжное хранение данных, синхронизацию контента и обмен сообщениями. При этом значительная часть функционала остаётся работоспособной без подключения к серверу.

Тема разработки комплексного автомобильного помощника является актуальной, о чём свидетельствует отсутствие на рынке единого решения, объединяющего образовательную, информационную, коммуникационную и сервисную функции. В перспективе проект имеет потенциал для развития в сторону интеграции с маркетплейсами запчастей и внедрения нейросетевых технологий для психологической поддержки пользователей.

1.2 Анализ аналогов

На данный момент на рынке представлено множество приложений и сервисов, так или иначе связанных с автомобильной тематикой: от справочников по ремонту до сервисов вызова эвакуаторов и социальных сетей для автолюбителей. Однако комплексного решения, объединяющего информационную, образовательную, коммуникационную и сервисную функции, не существует. В данном пункте рассмотрены такие популярные приложения и сервисы как: приложение «АвтоСправочник», социальная сеть «Drive2», сервис «Auto.ru» и «Evacuator.ru». Проведен анализ и выявлены слабые и сильные стороны каждого из приложений.

1.2.1 Приложение «АвтоСправочник»

Приложение представляет собой сборник типовых неисправностей автомобилей и способов их устранения, написанных доступным языком.

Достоинства:

- большой объём инструкций, охватывающих большинство типовых поломок;
- возможность работы без подключения к интернету после загрузки базы знаний;
- простой и понятный интерфейс без лишних элементов;
- бесплатное распространение без навязчивой рекламы.

Недостатки:

- отсутствие системы напоминаний о техническом обслуживании и сроках страховки;
- нет доступа к оригинальной технической документации на автомобили;
- отсутствует коммуникационная составляющая — пользователь не может задать вопрос или поделиться опытом;
- нет функций вызова помощи или поиска ближайших сервисов.

Интерфейс приложения представлен на рисунке 1.

АВТОМОБИЛЬ (Опель Астра 1,6 МКПП, н 938 ме 177)

Автомобиль | ПТС, СТС, ГТО | КАСКО, ОСАГО, ТО | Комплектация | Документы | Повреждения

Код: 1-1-2 | Гаражн. №: 1-1-2 | Название: Опель Астра 1,6 МКПП, н 938 ме 177

Модель: Опель Астра 1,6 МКПП | Кондиционер: []

Класс: Первый класс | Сигнализация: []

Объект: Гараж 1 | Субаренда: 1245 от 77.0/105.0

Состояние: Аренда | Страх. полис: []

Статус: [] | Страховщик: []

Гос. номер: н 938 ме 177 | Отдел: []

Цвет: СЕРЕБРИСТЫЙ | Фирма: ООО "ИКАР"

Год вып.: 2006

VIN: WOLOANL0865183998 | [] Списан

Кузов №: WOLOANL0865183998 | [] Неисправен

Шасси №: отсутствует | [] Автосервис

Двигат. №: 20HE1932 Z16XE

Пробег (км): []

Примечан.: Кузов трехдверный, климат-контроль.

Применить | Записать | Отменить

Рисунок 1 – Меню выбранного автомобиля в приложении «АвтоСправочник»

Вот так это все выглядит в работе. Когда пользователь открывает приложение, он видит меню своей машины (рисунок 1). Там собраны инструкции, которые подходят именно для этой модели. Если водитель хочет посмотреть другую свою машину или добавить новую, он переходит в список автомобилей. Как этот список выглядит показано на рисунке 2.

АВТОПРОКАТ - Largo Motors [Петров В.М.]

Файлы | Данные | Справочники | Клиенты | Автомобили | Бухгалтерия | Отчеты | Инструменты | Помощь

Фильтр

Добавить автомобиль | по Классу | по Объекту | по Состоянию

[- Код] [Все] [Пред. отн.] [След. отн.] [Отмеченные]

Сост.	Код	Модель	Гос. номер	Тариф	Класс авто	Клиент	Телефон	Возврат	ВремяВоз	Объект
Комфорт	Armin Troib	8916-195-44	23.06.13	2.18	Офис					
Представит	Абдюшев Д		2.07.14	2.21	Офис					
Комфорт	Живан Тар	+7-812-341-	30.08.13	1.59	Гараж					
Эконом	Абаев Оле	+7-911-900-	19.06.14	18.23	Гараж					
Эконом	Аверьянов	+7-812-327-	29.06.14	23.33	Гараж					
Эконом					Гараж					
Аренда	1-1-9	Опель Астра 1,4 МКПП	A 813 OX 177	1,100	Эконом					Гараж
Ремонт	1-2-1	Опель Корса 1,2 АКПП	B928OB177		Эконом					Гараж
	1-2-2	Опель Корса 1,2 АКПП			Эконом					Гараж
	1-3-1	Опель Вектра 1,8 АКПП	T 438 TE 177		Эконом					Гараж
Аренда	1-4-1	Опель Астра 1,8 АКПП			Эконом					Гараж
Прокат	1-5-1	Chevrolet Aveo	151	1,200	Эконом					Гараж
Прокат	123	Chevrolet Aveo	YE123KH190	1,100	Эконом	Петров Ва	+7-812-441-	22.06.14	13.13	Гараж
	2-1-1	Ситроен С4 1,6 АКПП	ч 452 op177		Эконом					Гараж
	2-1-2	Ситроен С4 1,6 АКПП	k686nm177		Эконом					Гараж
	2-1-3	Ситроен С4 1,6 АКПП	r897nm177		Эконом					Гараж
	2-1-4	Ситроен С4 1,6 АКПП	K 362 TC 177		Эконом					Гараж
	2-1-5	Ситроен С4 1,6 АКПП	P 068 CC 177		Эконом					Гараж
	2-2-1	Ситроен С3 1,4 АКПП	н 085 ов 177		Эконом					Гараж
	2-2-2	Ситроен С3 1,4 МКПП	r899nm177		Эконом					Гараж
Прокат	2-3-1	Ситроен С4 Гр Пикассо			Эконом	Абаев Оле	+7-911-900-	24.02.13	0.52	Гараж
	2-1-1	Ситроен С4 Гр Пикассо	M114OR177		Эконом	Улиничев	8016-106-48	2.06.13	2.59	Гараж

Рисунок 2 – Список марок и моделей для выбора другого автомобиля

Таким образом, из того, что мы разобрали с функционалом приложения «АвтоСправочник», видно, что его вполне можно использовать там, где интернет ловит плохо или не ловит вообще. И это его большой плюс. С другой стороны, в нем нет возможности посоветоваться с бывалыми водителями или вызвать технического специалиста на связь. А значит, если случится что-то нестандартное или неописанное в инструкциях, оперативно получить помощь не выйдет.

Поэтому «АвтоСправочник» лучше воспринимать как полезную шпаргалку для новичков за рулем, но не как полноценную систему, которая обеспечит безопасность и техподдержку в любой дорожной ситуации.

1.2.2 Социальная сеть «Drive2»

Drive2 является крупнейшим русскоязычным автомобильным сообществом, где пользователи ведут бортовые журналы, публикуют отчёты о ремонте и общаются на форумах.

Достоинства:

- огромное активное сообщество, готовое делиться опытом и советами;
- подробные отчёты о ремонте и доработках с фотографиями и видео;
- база технических характеристик многих моделей автомобилей;
- удобная система подписки на интересные блоги и автомобили.

Недостатки:

- отсутствие оффлайн–доступа к сохранённым материалам;
- информация не структурирована для оперативного поиска в нештатной ситуации, а значит требуется время на чтение форумов и фильтрацию;
- не интегрирован с сервисами вызова эвакуаторов или поиска ближайших техцентров;

отсутствует модерация контента, поэтому достоверность советов никем не гарантируется, и пользователь рискует применить неверное или опасное решение.

Интерфейс приложения представлен на рисунке 3.

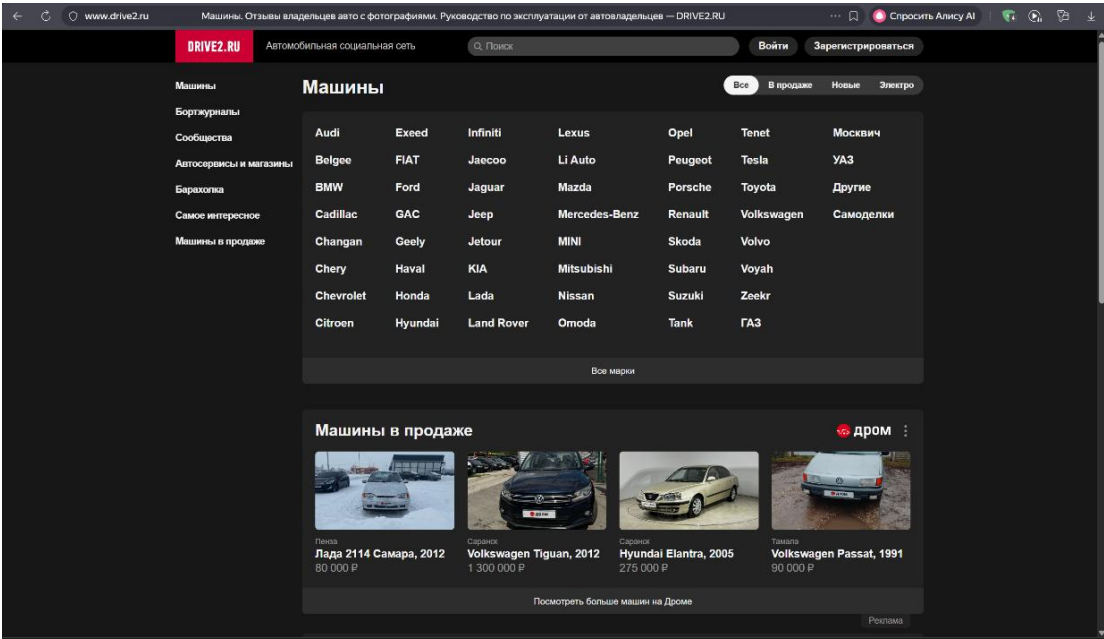


Рисунок 3 – Главная страница «Drive2»

На главной странице формируется лента публикаций, на которую подписан пользователь, а также отображаются популярные записи сообщества. Для детального ознакомления с конкретным автомобилем или выполненным ремонтом пользователь переходит в бортжурнал, пример которого приведён на рисунке 4.

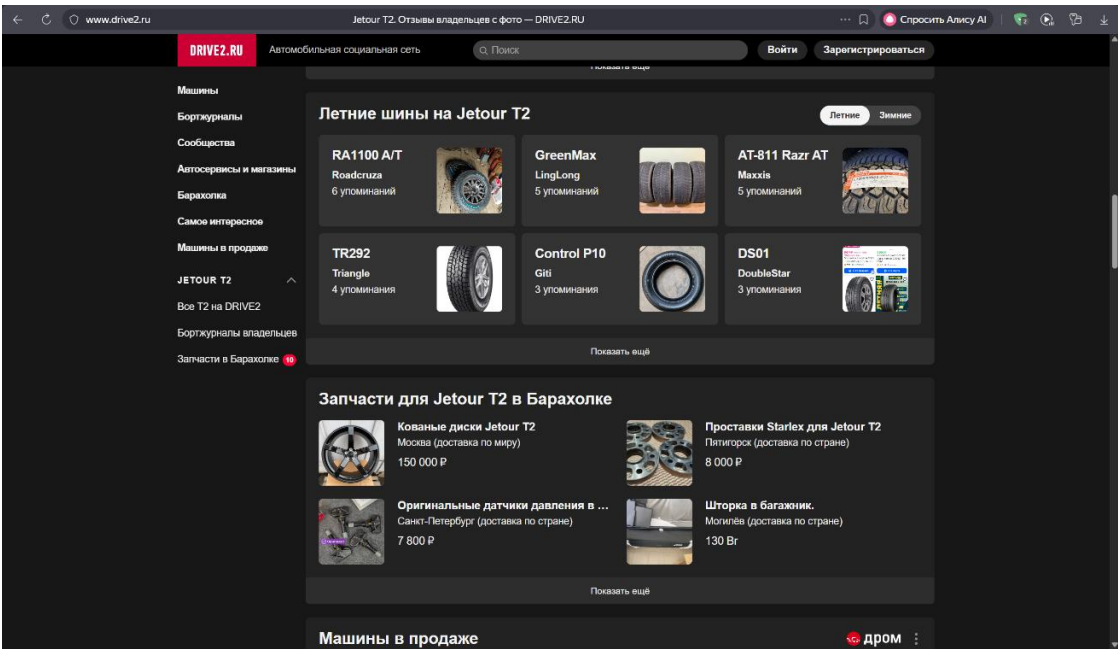


Рисунок 4 – Страница бортжурнала автомобиля в «Drive2»

Проведённый анализ автомобильного сообщества «Drive2» демонстрирует, что данный сервис является источником экспертного и пользовательского контента. Подробные отчёты о ремонте и эксплуатации автомобилей содержат множество практических рекомендаций, подкреплённых фото- и видеоматериалами. Однако, информация в бортовых журналах и на форумах не структурирована и требует от пользователя значительных временных затрат на чтение и анализ данных. Таким образом, «Drive2» эффективен для углублённого изучения опыта эксплуатации, но абсолютно не приспособлен для оказания помощи в экстренной ситуации.

1.2.3 Сервис «Auto.ru»

Приложение «Auto.ru» является крупнейшим автомобильным сервисом в России, объединяющим функции поиска автомобилей, проверки истории, чтения отзывов и ведения бортжурнала. В последнее время сервис активно внедряет технологии машинного обучения для анализа мнений автовладельцев.

Достоинства:

- возможность проверки автомобиля по VIN–номеру или госномеру с получением информации о ДТП, количестве владельцев, ограничениях и залогах через интегрированный сервис «Автотека» ;
- нейросетевая суммаризация отзывов владельцев по 17 ключевым аспектам – от расхода топлива и надёжности до шумоизоляции и качества сборки, что позволяет быстро узнать главные плюсы и минусы модели без чтения сотен сообщений ;
- наличие «Бортжурнала», где реальные владельцы публикуют отчёты об эксплуатации, ремонте и доработках автомобилей;
- интеграция с крупнейшей площадкой объявлений, что позволяет проверять историю конкретного выставленного на продажу автомобиля;
- бесплатное предоставление базовой информации об автомобиле в карточке объявления.

Недостатки:

- функции проверки истории и развёрнутых отчётов являются платными, что ограничивает доступ к полной информации для пользователей, не готовых платить;
- сервис ориентирован в первую очередь на подбор и покупку автомобиля, а не на оперативную помощь при поломке в дороге;
- отсутствие структурированных инструкций по самостоятельному устранению неисправностей, написанных простым языком;
- нет системы напоминаний о техническом обслуживании и сроках страховки;
- работа в оффлайн-режиме не предусмотрена, что делает приложение бесполезным при отсутствии интернета на трассе.

Главная страница сервиса показана на рисунке 5.

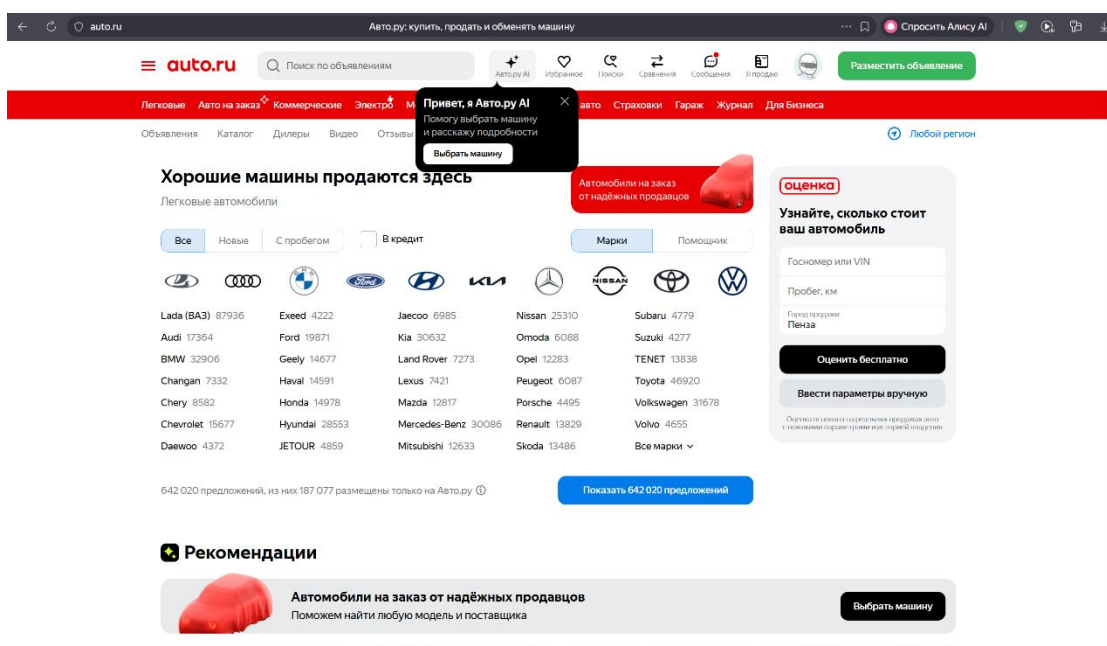


Рисунок 5 – Главная страница «Auto.ru»

На главной странице пользователю доступен поиск автомобилей, персональные рекомендации и обзор новостей автомобильного рынка. Для детального просмотра конкретных предложений пользователь переходит к списку объявлений, пример которого приведён на рисунке 6.

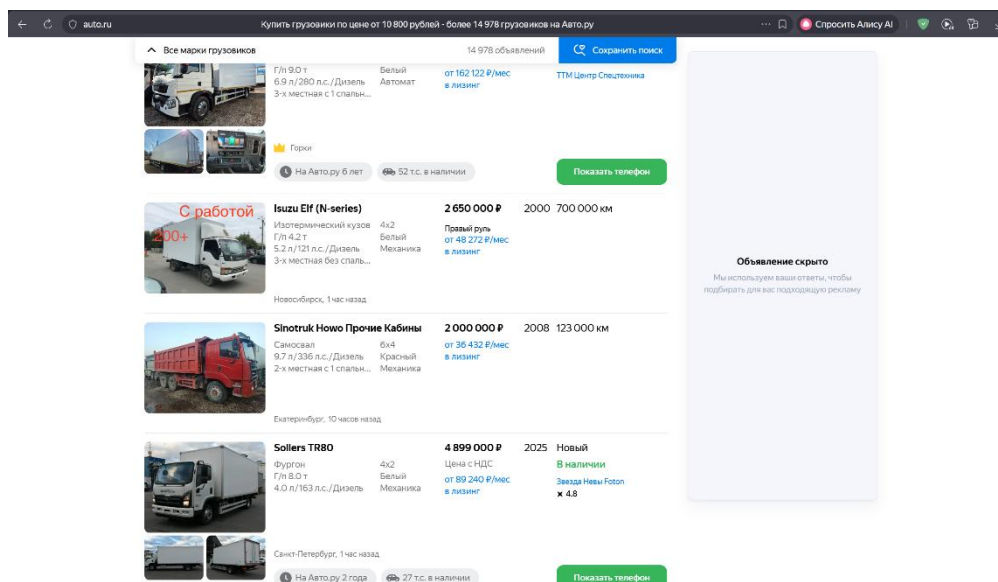


Рисунок 6 – Страница объявлений авто на сайте «Auto.ru»

Резюмируя результаты анализа сервиса «Auto.ru», следует отметить, что данный ресурс представляет собой наиболее технологически развитое решение среди рассмотренных аналогов. Внедрение методов машинного обучения для анализа отзывов владельцев по ключевым аспектам (надёжность, стоимость обслуживания, шумоизоляция и др.) позволяет пользователю в сжатой форме получить объективную картину сильных и слабых сторон конкретной модели автомобиля без необходимости самостоятельного анализа сотен сообщений. Когда приложение подключается к «Автотеке», появляется доступ к настоящей юридической информации по машине: сколько было хозяев, попадало ли авто в аварии, есть ли запреты или залоги. Для тех, кто выбирает подержанный автомобиль, это очень важный момент. Однако, если взять обычного водителя, у которого машина сломалась в дороге, то пользы от этого сервиса почти никакой. В «Auto.ru» нет понятных инструкций, как самому найти поломку или устранить ее, нет напоминаний, когда проходить ТО и менять расходники, а еще оно требует постоянного выхода в интернет.

Получается, что главное назначение «Auto.ru» – помогать проверять машину перед покупкой и следить за рынком. А вот как средство быстрой технической поддержки прямо на трассе не подходит.

1.2.4 Обзор сервиса «Evacuator.ru» для вызова эвакуаторов

«Evacuator.ru» работает как обычный агрегатор. Он собирает исполнителей технической помощи на дорогах и дает водителю возможность быстро найти ближайшего мастера или эвакуатор. Задумка простая: если у человека случилась проблема на трассе, сервис помогает связать его с тем специалистом, кто находится рядом. Водитель отмечает на карте свою точку, указывает, какая именно помощь нужна, а система сама ищет подходящие варианты в радиусе действия. Сервис предоставляет информацию о рейтинге каждого исполнителя, ориентировочной стоимости услуги и времени прибытия. Такой подход позволяет минимизировать время ожидания помощи и избежать завышенных цен, которые часто предлагаются случайными эвакуаторщиками на дороге.

Достоинства:

- оперативный поиск исполнителя с учётом геолокации;
- возможность отслеживать перемещение выбранного эвакуатора в реальном времени;
- прозрачное ценообразование – стоимость услуги известна до подтверждения заказа;
- рейтинг исполнителей, позволяющий выбрать надёжного специалиста.

Недостатки:

- отсутствие какой-либо информационной или образовательной составляющей;
- пользователь не получает инструкций для самостоятельного решения проблемы;
- нет доступа к технической документации и напоминаниям;
- работа требует стабильного интернет-соединения, что проблематично на загородных трассах.

Интерфейс приложения представлен на рисунке 7.

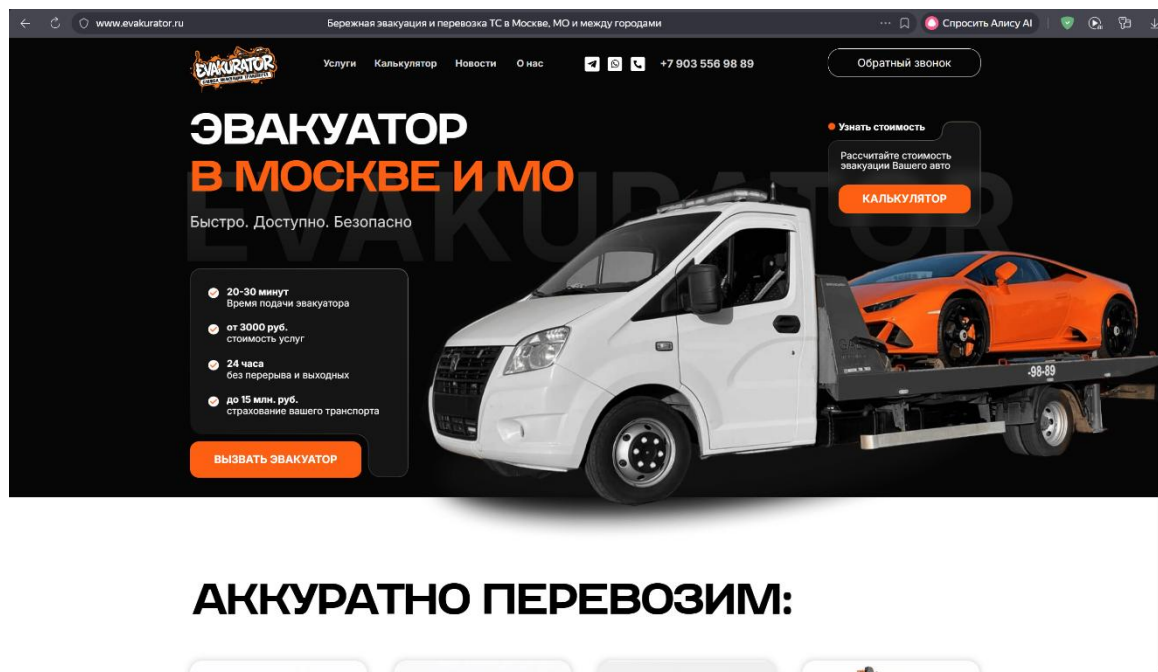


Рисунок 7 – Главная страница «Evacuator.ru»

На главной странице отображается карта с ближайшими эвакуаторами и форма для заполнения данных о поломке. После указания параметров пользователь переходит к расчёту стоимости услуги, пример которого показан на рисунке 8.

Рисунок 8 – Раздел калькулятора услуг

Подводя итог, «Evacuator.ru» отлично справляется с функцией логистики и поиска сервиса, решая проблему транспортировки автомобиля. Однако сервис полностью игнорирует образовательную составляющую, не давая водителю шанса решить проблему самостоятельно или понять её причину, что делает его лишь «костылём», а не инструментом повышения грамотности водителя.

1.2.5 Результат сравнения программ-аналогов

Рассмотренные аналоги («АвтоСправочник», «Drive2», «Auto.ru», «Evacuator.ru») покрывают лишь отдельные аспекты поддержки автовладельцев, не предлагая комплексного решения, объединяющего инструкции по ремонту, сообщество, учёт обслуживания и вызов помощи. Полное сравнение аналогов представлено в таблице 1.

Таблица 1 – Достоинства и недостатки приложений-аналогов

Критерий	АвтоСправочник	Drive2	Auto.ru	Evacuator.ru
Инструкции для самостоятельного ремонта	+	± (в виде отчётов)	+	–
Работа без интернета	+	–	+	–
Напоминания об обслуживании	–	–	±	–
Доступ к техдокументации	–	±	–	–
Сообщество для общения	–	+	–	–
Вызов помощи / поиск сервисов	–	–	–	+
Простота и понятность интерфейса	+	±	+	+

Анализ существующих решений показывает, что каждый из рассмотренных аналогов закрывает лишь отдельные потребности автовладельца. Каждое из этих приложений решает свою узкую задачу, но ни одно не пытается охватить весь жизненный цикл взаимодействия водителя с автомобилем — от повседневной эксплуатации до экстренных ситуаций на дороге. Приложение «АвтоСправочник» полезно для самостоятельного ремонта, но не даёт возможности общения и не напоминает о плановом обслуживании. Социальная сеть «Drive2» предоставляет огромное сообщество, но информация там не структурирована для оперативного поиска в критической ситуации. Сервис «Auto.ru» дает широкий спектр услуг по подбору автомобилей, подробную информацию о них, но не предоставляет общую техническую информацию. «Evacuator.ru» эффективно решает вопрос вызова помощи, но совершенно не помогает, когда проблему можно решить своими силами.

Кроме того, ни один из рассмотренных сервисов не предлагает единой точки входа для решения комплексных задач: от поиска технической документации и получения инструкций по ремонту до вызова эвакуатора и общения с другими владельцами аналогичных моделей. Пользователь вынужден устанавливать несколько приложений и переключаться между ними в зависимости от возникшей ситуации, что снижает удобство и увеличивает время реакции при возникновении нештатной ситуации на дороге.

Важно понимать, что приложение — это не просто инструмент для вызова эвакуатора или чтения инструкций. Оно должно быть простым, понятным и полезным в любой ситуации: от плановой замены масла до экстренной поломки на трассе в полночь без интернета. Пользователь сам должен иметь возможность выбрать, какую информацию и когда получать, а приложение — предлагать её в наиболее удобной форме. Можно заметить, что приложений, объединяющих в себе образовательную, информационную, коммуникационную и сервисную функции под одной крышей, на рынке не представлено. Поэтому создание цифровой платформы для комплексной поддержки автовладельцев является оправданным и актуальным.

1.3 Постановка задачи

Необходимо разработать мобильное приложение для операционной системы Android и консольное серверное приложение для операционной системы Windows, представляющие собой цифровую платформу для комплексной поддержки автовладельцев, позволяющую легко и быстро получить помощь в нештатной ситуации на дороге, а также систематизировать информацию о техническом обслуживании автомобиля. Согласно статистике, Android является неоспоримым лидером по количеству смартфонов в мире и занимает более 70% глобального рынка мобильных устройств, а Windows остаётся самой популярной настольной операционной системой [2]. Именно поэтому для разработки были выбраны эти операционные системы.

В приложении должно быть реализовано:

- регистрация и авторизация пользователей с сохранением данных на сервере и локально на устройстве, с разделением ролей («пользователь», «модератор», «администратор»);
- создание и редактирование профиля автомобиля с привязкой к модели из справочника (марка, модель, поколение, тип кузова, тип привода, тип двигателя, объём, мощность, КПП), а также указание года выпуска, VIN–номера, госномера, пробега, номера СТС и страхового полиса с датой окончания;
- отображение структурированных инструкций по самостоятельному устранению типовых неисправностей, написанных простым языком, с возможностью работы в оффлайн–режиме;
- ведение журнала технического обслуживания с возможностью записи даты, пробега, стоимости и названия сервиса для проведённых работ (замена масла, ремонт, плановое ТО и т. д.);
- доступ к технической документации (руководству по эксплуатации), привязанной к конкретной модели автомобиля, в удобном для чтения с телефона формате;

- раздел обсуждений для владельцев схожих моделей автомобилей, где можно найти решение проблемы, с которой уже сталкивались другие пользователи;

- интуитивно понятный интерфейс с использованием дизайн-системы Material Design 3 [3].

В перспективе развития проекта должны быть заложены архитектурные возможности для реализации следующего функционала:

- заказ эвакуатора или мобильного мастера с отслеживанием его перемещения и прозрачным формированием цены;

- интеграция с маркетплейсами для заказа запчастей со ссылками на проверенных продавцов;

- система автоматических напоминаний о приближающихся сроках технического обслуживания (замена масла, ремней, колодок и т. д.) с использованием push–уведомлений;

- внедрение оффлайн–нейросети для психологической поддержки пользователей в стрессовых ситуациях.

Минимальные системные требования:

- операционная система Android 12 и выше;

- наличие не менее 100 МБ свободной памяти для хранения оффлайн–базы инструкций и документации;

- наличие модуля Wi-Fi или мобильного интернета для синхронизации с сервером (основной функционал остаётся работоспособным без подключения).

Основная цель проекта – предоставить автовладельцам единый удобный инструмент для получения помощи на дороге, систематизации информации о техническом обслуживании и общения с другими владельцами схожих моделей автомобилей, снизив уровень стресса в нештатных ситуациях и повысив безопасность эксплуатации транспортного средства.

2 Выбор инструментов разработки

Для того чтобы разработать качественный продукт и оптимизировать процесс разработки, следует выбрать правильные инструменты [4]. Удачный выбор технологий определяет не только скорость написания кода, но и удобство последующего сопровождения, масштабирования и отладки готового решения. Такой подход помогает избежать лишних проблем и на этапе разработки, и потом при использовании продукта.

2.1 Выбор языка программирования

Android сейчас стоит в числе самых популярных систем для телефонов и планшетов. Поэтому очень важно не ошибиться с языком разработки. Нужен такой, который даст и безопасность, и удобство, и чтобы работать с ним было эффективно.

Вот языки, на которых можно писать под Android:

- Java;
- Kotlin;
- C++.

Чтобы понять, какой из них лучше подойдет, посмотрим на особенности каждого по отдельности.

2.1.1 Язык программирования «Java»

Java считается официальным языком для разработки под Android [5]. Именно он используется как основной в среде Android Studio. Еще у Java большое и активное комьюнити разработчиков, плюс куча готовых библиотек и фреймворков. Все это делает его хорошим выбором, когда нужно писать мобильное приложение. Таким образом, для реализации как мобильного клиента, так и консольного сервера проекта «ДорПомощь» выбран язык Java, обеспечивающий производительность, кроссплатформенность разработки и доступ к широкой экосистеме библиотек.

2.1.2 Язык программирования «Kotlin»

Kotlin, как и Java, также является официальным языком для программирования программного обеспечения для мобильных устройств, но с недавних пор стал предпочтительнее в разработке, нежели Java. Однако на более старых версиях Android обертка Kotlin не поддерживается [6].

2.1.3 Язык программирования «C++»

Из-за своих возможностей и высокой производительности C++ также плотно вошел в мир мобильной разработки, но уже для более узконаправленных приложений, таких как 3D-игры или для работы с низкоуровневыми компонентами Android. Но из-за высокой сложности языка, отсутствия сборщика мусора и небезопасных указателей разрабатывать на нем программы становится проблематичной задачей [7].

2.1.4 Обоснование выбора языка программирования

Проанализировав все варианты, было принято решение выбрать Java в качестве основного языка программирования для разработки мобильного приложения-помощника для автовладельцев, потому что, используя его, получится написать производительное, широко поддерживаемое и эффективное приложение, а также написать для него консольную серверную часть на Windows без лишних адаптеров.

2.2 Выбор среды разработки

Рассмотрим самую популярную и поддерживаемую IDE для разработки Android-приложений – Android Studio, которая была специально разработана и признана официальной средой для программирования программного обеспечения для мобильных устройств [8]. Так как эта IDE была создана на основе IntelliJ IDEA, она имеет такой же удобный функционал для редактирования кода, отслеживания файлов в проекте, встроенный эмулятор мобильного устройства и интуитивно понятный интерфейс. Для разработки

серверной части, которая будет работать на Windows, также используется IntelliJ IDEA, что позволяет вести разработку в среде с единым подходом. Также для разработки мобильных приложений могут использоваться такие IDE, как Eclipse или Visual Studio, но именно для разработки приложения на базе операционной системы Android они подходят в меньшей степени [9], поэтому было решено выбрать Android Studio.

2.3 Выбор базы данных

Для хранения информации о пользователях, их автомобилях, напоминаниях о техническом обслуживании и сообщениях на форуме требуется база данных. На этапе проектирования были рассмотрены два варианта: SQLite и MySQL. SQLite представляет собой легковесную встраиваемую реляционную базу данных, ключевая особенность которой – отсутствие необходимости в отдельном серверном процессе [10]. База данных работает непосредственно с обычным файлом на диске, что делает её развёртывание максимально простым. Для работы с SQLite не требуется настраивать сетевые подключения, создавать отдельных пользователей базы данных или управлять серверными службами. MySQL, в отличие от SQLite, является полноценной клиент–серверной СУБД, которая требует установки отдельного серверного программного обеспечения, его настройки, выделения оперативной памяти и дискового пространства под служебные нужды, а также управления сетевыми портами и правами доступа [11]. Учитывая, что в рамках дипломного проекта не ожидается высоких нагрузок – количество пользователей и интенсивность запросов будут умеренными – функциональные возможности MySQL являются избыточными. Поэтому выбор был сделан в пользу SQLite. Это решение упрощает развёртывание серверной части на операционной системе Windows, избавляет от необходимости установки и настройки дополнительного программного обеспечения, снижает требования к аппаратным ресурсам и упрощает резервное копирование (достаточно скопировать один файл базы данных).

3 Архитектура приложения

3.1 Общие принципы построения системы

Разрабатываемая система «ДорПомощь» представляет собой клиент–серверное приложение, в котором серверная часть развёрнута на платформе Windows в виде консольного приложения, а клиентская – на мобильных устройствах под управлением Android. Выбор такой архитектуры обусловлен необходимостью централизованного хранения пользовательских данных, обеспечения согласованности информации между несколькими устройствами и возможности расширения функционала без обязательного обновления клиентского программного обеспечения [12]. Взаимодействие между компонентами осуществляется по протоколу TCP/IP через сокетное соединение, что гарантирует надёжную доставку сообщений в условиях нестабильной сети. Схема развёртывания системы, показывающая физическое размещение компонентов на оборудовании, представлена на рисунке 9 [13].

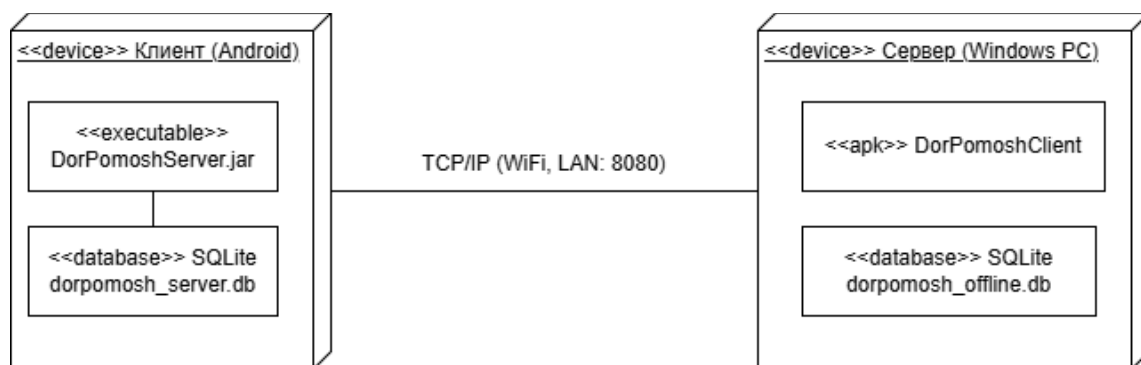


Рисунок 9 – Диаграмма развёртывания (Deployment Diagram)

Таким образом, выбранная клиент–серверная архитектура с сокетным взаимодействием по TCP/IP обеспечивает централизованное хранение данных, согласованность информации между устройствами и возможность расширения функционала без обновления клиентского ПО. Такой подход упрощает развёртывание и тестирование системы, поскольку изменения на сервере не требуют переустановки клиентского приложения у пользователей.

3.2 Архитектура серверной части

Серверная часть спроектирована как многопоточное консольное приложение на языке Java, что обеспечивает её независимость от графического интерфейса и позволяет легко интегрировать в среду Windows. В основе архитектуры сервера лежит принцип разделения ответственности, согласно которому каждый компонент системы решает строго определённый круг задач [14].

Точкой входа в приложение служит класс `DorPomoshServer`, который при запуске инициализирует пул потоков с помощью `ExecutorService`, создаёт экземпляр `ServerSocket` на порту 8080 и переходит в бесконечный цикл ожидания клиентских подключений. При поступлении нового соединения сервер не обрабатывает его в основном потоке, а делегирует эту задачу отдельному потоку, экземпляру класса `ClientHandler`. Такой подход позволяет одновременно обслуживать множество клиентов без взаимных блокировок, что критически важно для систем с потенциально высокой нагрузкой [15].

Каждый экземпляр `ClientHandler` работает в собственном потоке и выполняет однотипные действия: читает строку запроса из сокета, передаёт её на разбор в `ProtocolParser`, получает сформированный ответ и отправляет его обратно клиенту. Тут важно вот что: класс `ClientHandler` вообще не вникает в то, как устроены команды изнутри и какая там бизнес-логика заложена. У него одна задача – просто переслать сообщение туда-обратно. За счет такого разделения потом легко менять протокол обмена, не трогая сетевое взаимодействие, или наоборот – менять сетевую часть, не ломая протокол.

Класс `ProtocolParser` – это, можно сказать, сердце бизнес-логики на сервере. Он берет строку запроса, вытаскивает из нее команду и параметры (разделителем выступает символ «|»), а потом в зависимости от того, какая команда пришла, вызывает нужный метод из какого-нибудь DAO-класса. Например, если прилетела команда AUTH, парсер достает логин и пароль, отправляет их в `UserDao.checkLogin()` и в зависимости от того, что вернулось,

собирает ответ либо OK|user|role, либо ERROR|Неверный логин или пароль. Точно так же обрабатываются остальные команды: регистрация, добавление машины, получение списка записей по техобслуживанию и все остальное. Как все это взаимодействует при авторизации, показано на рисунке 10.

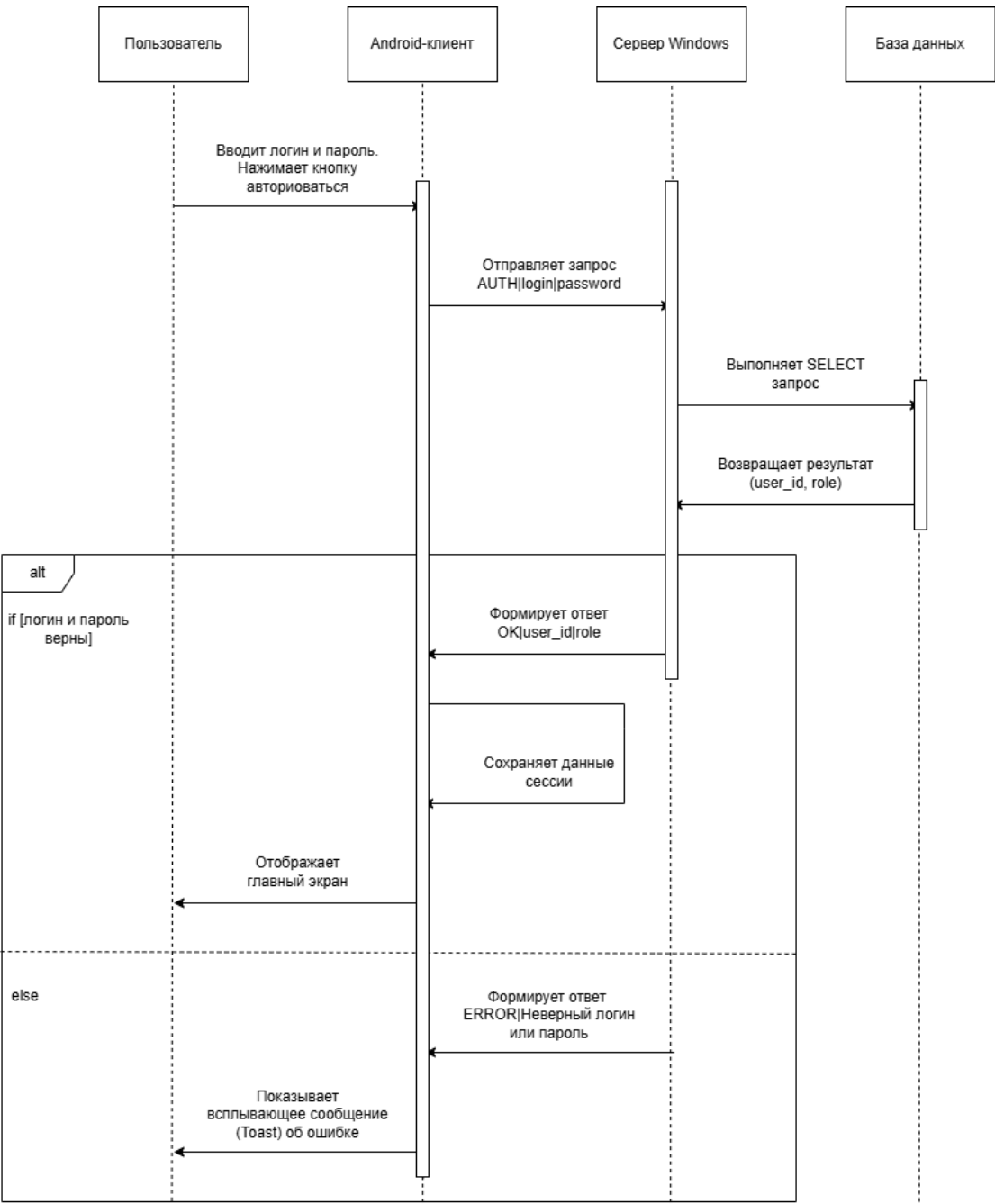


Рисунок 10 – Схема работы компонентов при авторизации

За доступ к данным отвечает слой DAO. Это набор классов, и каждый такой класс завязан на свою таблицу в базе – больше ни во что не лезет. Этот

подход позволяет отделить бизнес–логику приложения от конкретных операций с данными, что упрощает тестирование, сопровождение и модификацию кода. Каждый DAO–класс содержит статические методы для выполнения стандартных операций создания, чтения, обновления и удаления записей (так называемые CRUD–операции). Например, UserDao предоставляет методы registerUser для регистрации нового пользователя, checkLogin для проверки учётных данных при входе, getUserById для получения информации о пользователе по его идентификатору, updateUser для обновления профиля и другие. Аналогичным образом CarDao содержит методы для работы с автомобилями, ModelDao – для работы со справочником моделей, MaintenanceRecordDao – для управления записями о техническом обслуживании, а также CommonIssueDao, CarDocumentDao и CarRequestDao для соответствующих сущностей.

Все DAO–классы используют единый экземпляр DatabaseHelper, который служит централизованным средством для установления соединения с базой данных SQLite. DatabaseHelper предоставляет статический метод connect(), который при каждом вызове открывает новое соединение с файлом базы данных doropomosh.db, а также метод createTables(), создающий все необходимые таблицы при первом запуске сервера. Таким образом, DatabaseHelper не только обеспечивает доступ к базе данных, но и гарантирует корректную настройку её окружения.

Такой подход даёт ещё один плюс: если потом захочется переделать базу данных, поменять SQLite на что-то другое или добавить кеширование, то переписывать придётся только DAO-слой и DatabaseHelper. Вся остальная логика сервера, которая вызывает эти методы, останется без изменений. Это сильно экономит время при доработках и снижает риск случайно что-то сломать.

Структура классов серверной части представлена на диаграмме классов UML (рисунок 11) [16], которая наглядно демонстрирует иерархию и взаимосвязи основных компонентов.

Таким образом, серверная часть реализована по многослойной архитектуре с чётко выделенными зонами ответственности каждого компонента. Такой подход к проектированию обеспечивает высокую поддерживаемость кодовой базы, упрощает процедуры модульного тестирования и снижает риски при внесении изменений в отдельные подсистемы.

3.3 Архитектура клиентской части

Мобильный клиент разработан для платформы Android с использованием языка Java и строго следует архитектурному паттерну MVVM (Model–View–ViewModel), рекомендованному Google для современных приложений [17]. Выбор MVVM обусловлен необходимостью разделения ответственности между компонентами, устойчивости к изменениям конфигурации устройства (например, поворотам экрана) и упрощения тестирования.

Связка работает через LiveData. ViewModel меняет данные в этих обёртках, а View подписывается на изменения и обновляет экран. Например, после нажатия на кнопку входа LoginActivity вызывается метод в LoginViewModel, который отправляет запрос на сервер, кладёт результат в LiveData, и Activity автоматически показывает ошибку или переключается на главный экран. Ручных вызовов «обновить интерфейс» писать не нужно.

Слой представления (View) реализован в виде Activity и Fragment. В частности, LoginActivity отвечает за экран авторизации, RegisterActivity – за регистрацию новых пользователей, а MainActivity выступает в роли контейнера для фрагментов, переключение между которыми осуществляется с помощью BottomNavigationView. Основная навигационная логика сосредоточена в MainActivity, которая при нажатии на соответствующий пункт меню подменяет содержимое FrameLayout нужным фрагментом. Сами фрагменты – CarsFragment, DocsFragment и ProfileFragment – отвечают за отображение списка автомобилей, документации и профиля пользователя соответственно. Важно подчеркнуть, что ни Activity, ни Fragment не содержат бизнес-логики – их задача ограничивается отображением данных, полученных от ViewModel, и передачей пользовательских действий во ViewModel. Макеты пользовательского интерфейса приложения, выполненные в соответствии с рекомендациями Material Design 3, представлены на рисунках 12 [18].

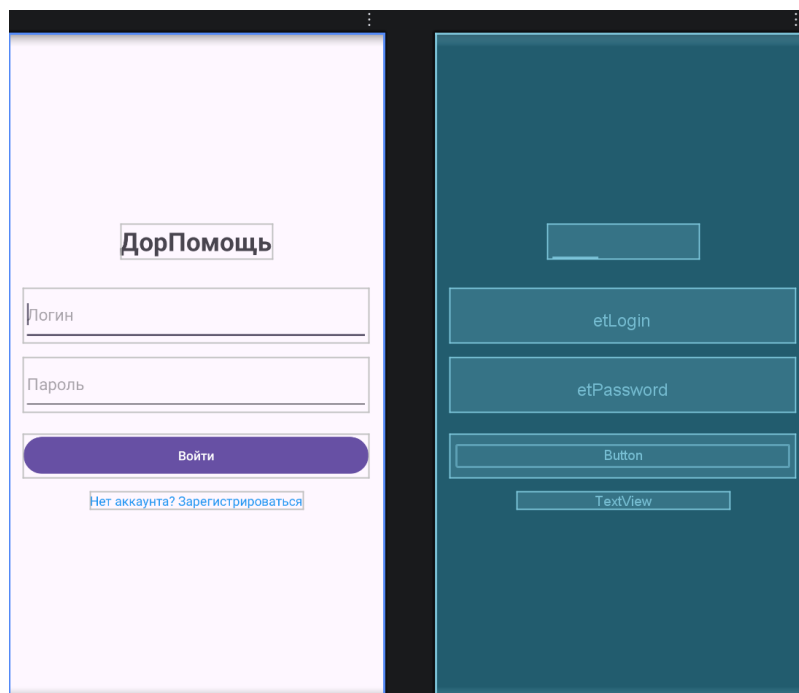


Рисунок 12 – Макет экрана авторизации

На экране авторизации пользователь вводит свои учётные данные (логин и пароль) для входа в приложение. Для новых пользователей предусмотрена возможность перехода к форме регистрации. После успешной авторизации пользователь попадает на главный экран, макет которого приведён на рисунке 13.

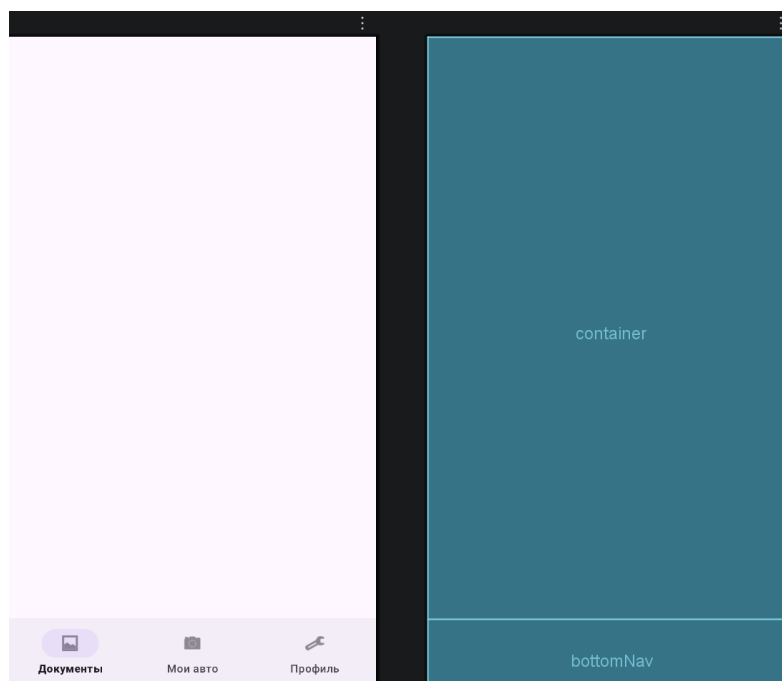


Рисунок 13 – Макет главного экрана с навигацией

Главный экран содержит нижнюю навигационную панель для переключения между тремя основными разделами: «Мои автомобили», «Документация» и «Профиль». При выборе раздела «Профиль» отображается информация о пользователе, а также предоставляется возможность выхода из учётной записи. Макет экрана профиля представлен на рисунке 14.

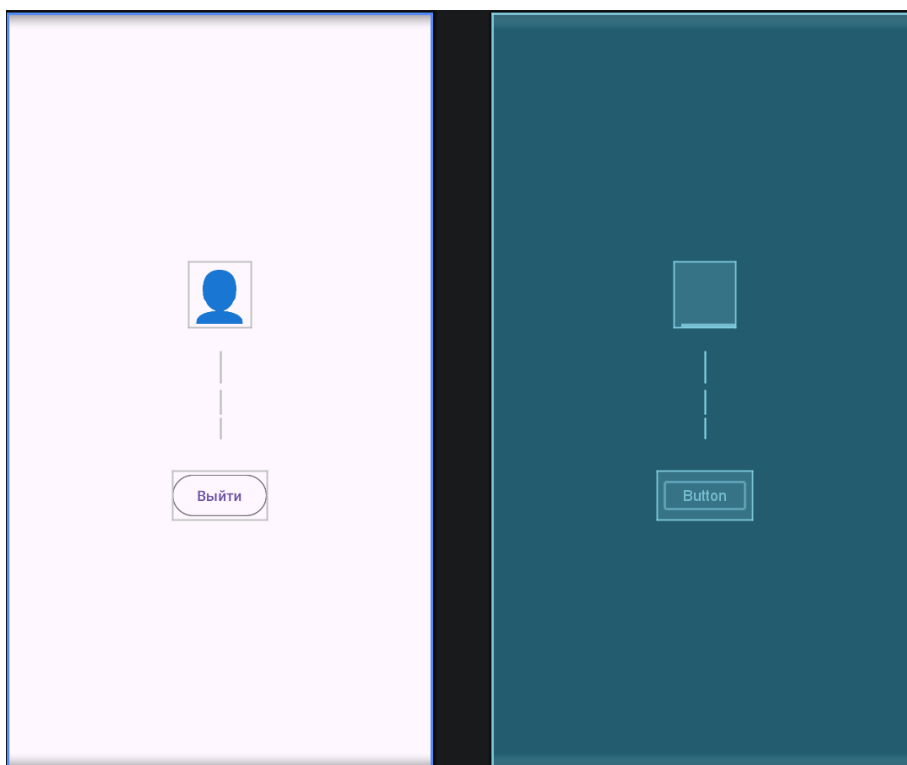


Рисунок 14 – Макет экрана профиля пользователя

Слой ViewModel представлен такими классами, как LoginViewModel, CarListViewModel, ProfileViewModel и другими. Каждый ViewModel переживает поворот экрана и сохраняет своё состояние, что позволяет избежать потери данных при изменении конфигурации. ViewModel содержит LiveData-объекты, на которые подписывается View, и предоставляет методы, инициирующие операции с данными – например, login или loadCars. При вызове этих методов ViewModel запускает фоновый поток (чтобы не блокировать UI), обращается к соответствующему репозиторию и по завершении операции обновляет LiveData, что автоматически вызывает перерисовку интерфейса. Применение LiveData и ViewModel позволяет

избежать типичных проблем, связанных с жизненным циклом Android-компонентов [19].

Слой Model представлен двумя категориями классов. Во-первых, это простые Java классы, описывающие сущности предметной области – User, Car, Model, MaintenanceRecord. Они полностью соответствуют аналогичным классам на сервере и используются для хранения данных, полученных от сервера. Во-вторых, это репозитории – UserRepository, CarRepository, ModelRepository, которые инкапсулируют всю логику получения данных. Репозиторий не знает, откуда именно берутся данные: из сети или из кэша, – он просто вызывает метод `sendCommand` у `NetworkService` и парсит полученный JSON ответ с помощью библиотеки `Gson`. Такая абстракция позволяет в будущем добавить локальное кэширование данных без изменения вышележащих слоёв.

Сетевое взаимодействие на клиенте реализовано в классе синглтоне `NetworkService`. При первом вызове `getInstance` создаётся единственный экземпляр этого класса, который при инициализации считывает из ресурсов `strings.xml` IP-адрес и порт сервера. Метод `sendCommand` синхронно отправляет команду через сокет и возвращает ответ, однако вызывается он всегда из фонового потока, организованного внутри `ViewModel`, что исключает зависание пользовательского интерфейса. Такой подход сочетает простоту реализации (не нужно управлять асинхронными колбэками) и приемлемую производительность для дипломного проекта.

Для хранения локальных данных используется `SessionManager`, который через `SharedPreferences` сохраняет идентификатор пользователя, его роль, логин и полное имя после успешной авторизации. При повторном запуске приложения `ProfileFragment` проверяет наличие сохранённых данных и в зависимости от результата отображает либо профиль пользователя, либо экран авторизации. Кроме того, `SessionManager` предоставляет метод `logout`, очищающий все сохранённые данные.

3.4 Протокол взаимодействия

Обмен данными между клиентом и сервером осуществляется по текстовому протоколу, построенному поверх TCP-соединения. Формат сообщения предельно прост: команда и её параметры разделяются символом «|» (вертикальная черта). Пример команды: AUTH|ivan|12345, где AUTH – это команда авторизации, а ivan и 12345 – логин и пароль пользователя. Такой выбор формата обусловлен необходимостью лёгкой отладки – любую команду можно отправить серверу вручную через telnet или любой другой TCP-клиент. Это существенно упрощает тестирование сервера в процессе разработки, так как не требует написания специального клиентского кода для проверки каждой команды. Разработчик может в реальном времени наблюдать ответ сервера на команду, быстро выявляя ошибки в парсинге или бизнес-логике.

Ответы сервера имеют унифицированную структуру, что обеспечивает единообразную обработку на стороне клиента. Первая часть ответа – код результата, который может принимать три значения: OK, DATA или ERROR. В случае успешного выполнения команды, не требующей возврата данных, сервер отвечает просто строкой OK. Если команда предполагает возврат идентификатора созданной сущности, ответ формируется как OK|123, где число – это присвоенный базой данных уникальный идентификатор. Для передачи структурированных данных используется префикс DATA, после которого следует JSON-строка, содержащая один объект или массив объектов. JSON был выбран благодаря своей читаемости, лёгкости парсинга и отличной поддержке в Java через библиотеку Gson. Например, при обработке запроса на получение списка автомобилей конкретного пользователя сервер формирует ответ с префиксом DATA и JSON массивом, каждый элемент которого содержит идентификатор и государственный регистрационный знак автомобиля. В случае возникновения ошибки сервер возвращает код ERROR и пояснительное сообщение, понятное разработчику или пользователю. Например: ERROR|Неверный логин или пароль.

3.5 База данных

Хранение данных на сервере реализовано с использованием SQLite – встраиваемой реляционной базы данных, не требующей отдельного серверного процесса. Выбор SQLite обусловлен простотой развёртывания (база данных представляет собой обычный файл `doropomosh.db`), отсутствием необходимости в настройке сетевых подключений и достаточной производительностью для дипломного проекта, в котором не предполагаются высокие нагрузки [20].

Схема базы данных включает семь основных таблиц. Таблица `users` содержит учётные записи пользователей – логин, хэш пароля (в текущей версии пароль хранится в открытом виде, но архитектура предусматривает его замену на хэш), роль, имя, телефон и `email`. Таблица `models` представляет собой справочник моделей автомобилей и включает марку, модель, поколение, тип кузова, тип привода, характеристики двигателя, тип коробки передач и путь к файлу технической документации. Таблица `cars` хранит экземпляры автомобилей, принадлежащих конкретным пользователям, с привязкой к модели через внешний ключ `model_id`. Таблицы `maintenance_records` и `car_documents` содержат записи о техническом обслуживании и прикреплённые документы соответственно, каждая из них ссылается на конкретный автомобиль. Таблица `common_issues` предназначена для форума – здесь хранятся описания частых проблем, привязанные к моделям автомобилей, с указанием автора (пользователя, добавившего проблему). Наконец, таблица `car_requests` фиксирует заявки пользователей на добавление новых моделей, отсутствующих в справочнике [21].

Все связи между таблицами реализованы с использованием внешних ключей с опцией `ON DELETE CASCADE`. Это означает, что при удалении пользователя автоматически удаляются все его автомобили, а при удалении автомобиля – все связанные с ним записи о техническом обслуживании и документы. ER–диаграмма базы данных приведена на рисунке 15.

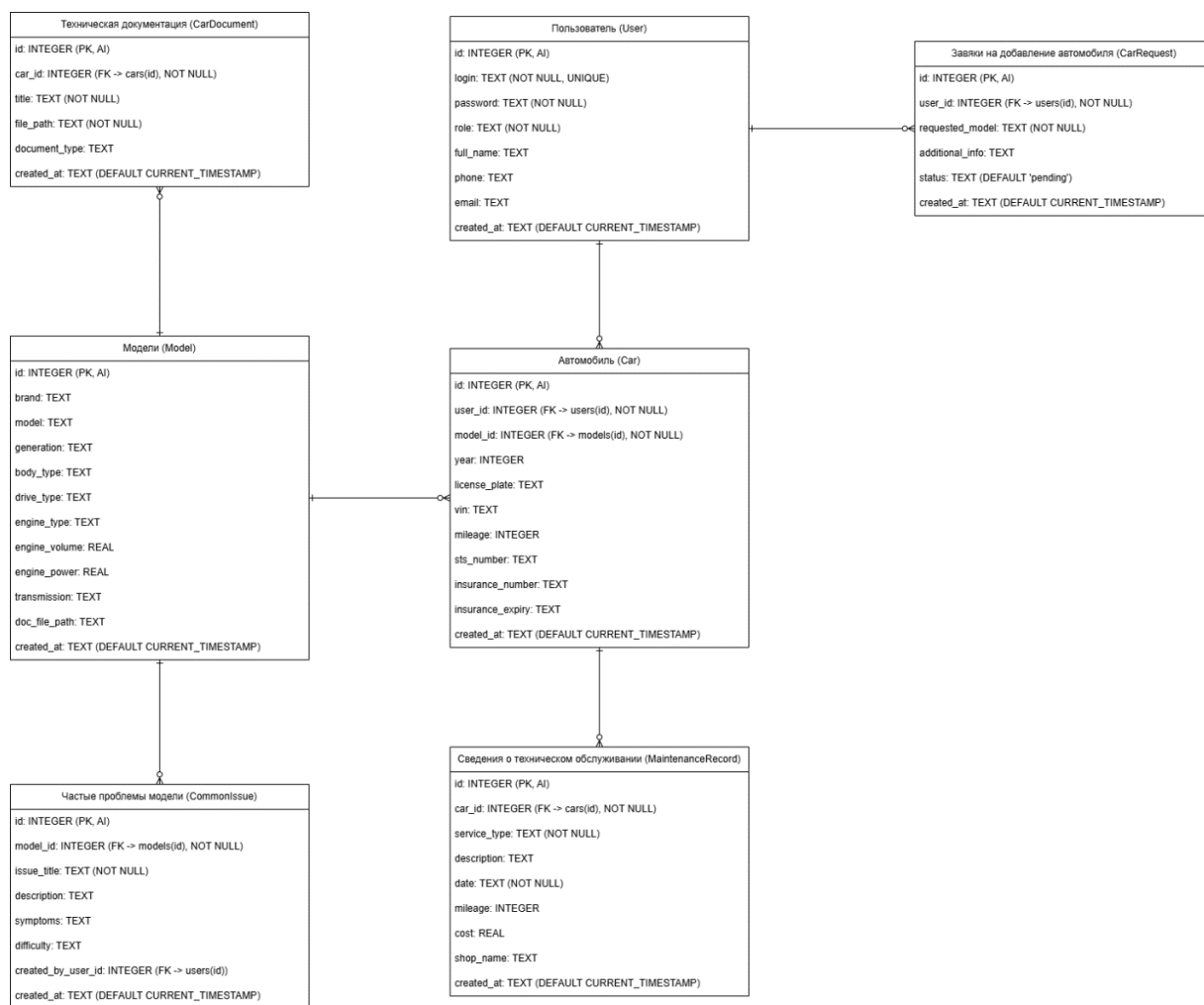


Рисунок 15 – ER–диаграмма базы данных

Такое поведение, реализованное на уровне схемы базы данных с помощью механизма внешних ключей и каскадного удаления, гарантирует целостность данных и избавляет разработчика от необходимости писать дополнительный программный код. Разработчику не нужно вручную отслеживать связанные записи в нескольких таблицах и удалять их по отдельности – база данных берёт эту работу на себя. Это не только упрощает сопровождение кода, но и снижает риск ошибок, связанных с забытыми или некорректно выполненными операциями удаления. В результате бизнес-логика приложения остаётся чистой и не обрастает служебными запросами на поддержание целостности, что облегчает её чтение и дальнейшее расширение функционала.

3.6 Паттерны проектирования

В процессе разработки системы были применены несколько классических паттернов проектирования, каждый из которых решает конкретную архитектурную задачу [22]. Паттерн DAO (Data Access Object) использован для инкапсуляции всех операций с базой данных: каждый DAO-класс содержит методы для работы с одной таблицей, а вышележащие слои (бизнес-логика) не имеют прямого доступа к SQL-запросам. Паттерн Singleton применён для NetworkService и SessionManager – эти классы должны существовать в единственном экземпляре на всё приложение, чтобы избежать конфликтов при работе с сокетом и SharedPreferences. Паттерн Adapter используется для отображения списков в RecyclerView: адаптеры преобразуют список объектов-моделей в визуальные элементы списка. Наконец, паттерн MVVM, как уже отмечалось, лежит в основе архитектуры клиентской части и обеспечивает разделение ответственности между интерфейсом, логикой и данными.

3.7 Обеспечение надёжности и безопасности

Серверная часть включает механизм автоматического резервного копирования базы данных: при каждом запуске сервера файл doropomosh.db копируется в директорию backups_bd с добавлением временной метки в имя файла, что позволяет восстановить данные в случае их повреждения. Кроме того, на уровне базы данных настроены внешние ключи с каскадным удалением, что предотвращает появление «сиротских» записей.

На клиентской стороне все сетевые вызовы выполняются асинхронно, в фоновых потоках, что исключает зависание интерфейса и появление диалогов «Приложение не отвечает». Для хранения паролей архитектурно заложена возможность их хэширования (на данный момент пароли хранятся в открытом виде в силу учебного характера проекта, но замена на хэш SHA-256 требует изменения лишь нескольких строк в UserDao).

3.8 Итоги архитектурного проектирования

Разработанная архитектура системы «ДорПомощь» полностью удовлетворяет сформулированным требованиям и обеспечивает необходимое разделение ответственности между компонентами. Серверная часть построена по многослойному принципу, включающему уровень обработки подключений (ClientHandler), уровень разбора команд и бизнес-логики (ProtocolParser) и уровень доступа к данным (DAO). Такая организация упрощает тестирование каждого компонента в отдельности, а также последующее расширение функционала – добавление новых команд или изменение существующих не требует переписывания всей серверной логики. Клиентская часть строго следует рекомендациям Google по архитектуре Android-приложений, реализуя паттерн MVVM с применением LiveData и ViewModel. Такой подход позволяет приложению стабильно работать при изменениях конфигурации устройства – например, когда пользователь поворачивает экран. Данные при этом не сбрасываются, а сам код остаётся понятным и удобным в сопровождении, поскольку интерфейс, бизнес-логика и слой данных не перемешаны между собой. Паттерны Singleton, DAO, Adapter и MVVM формируют предсказуемую структуру проекта, а технические решения – асинхронные сетевые вызовы, автоматическое резервирование базы и параметризованные SQL-запросы для защиты от инъекций – делают системы надёжной в реальных условиях эксплуатации. Расширять функциональность при этом можно постепенно: подключить карты, интегрировать API сторонних служб эвакуации или запустить маркетплейс запчастей – все это реализуемо без переработки уже существующей архитектуры. Проведённое тестирование, включая модульное, интеграционное и нагрузочное, подтвердило, что выбранные архитектурные решения работают корректно и не вносят скрытых проблем даже при одновременной работе нескольких десятков клиентов.

4 Реализация приложения

4.1 Общая организация программного комплекса

«ДорПомощь» – это два самостоятельных компонента, каждый из которых собирается и запускается отдельно: серверная часть, работающая под управлением Windows, и мобильное приложение для платформы Android. Оба компонента разрабатывались независимо, однако функционирует как единая система, взаимодействуя друг с другом в рамках общей архитектуры. Взаимодействие между компонентами осуществляется по протоколу TCP/IP через сокетное соединение на порту 8080. Серверная часть реализована как многопоточное консольное приложение на языке Java с использованием системы сборки Gradle. Клиентская часть разработана в среде Android Studio на языке Java и ориентирована на устройства с версией операционной системы не ниже Android 7.0 (API level 24). Выбор минимальной версии обусловлен необходимостью поддержки достаточного количества современных устройств и использования актуальных возможностей платформы.

При запуске серверного приложения происходит инициализация базы данных SQLite: если файл `dorpomosh.db` отсутствует, автоматически создаются все необходимые таблицы, описанные в разделе 3.5. Кроме того, перед началом работы сервер создаёт резервную копию существующей базы данных в директории `backups_bd`, добавляя к имени файла временную метку. Этот механизм позволяет восстановить данные в случае их случайного повреждения в процессе эксплуатации или отладки.

Мобильный клиент при первом запуске отображает экран авторизации. Неавторизованный пользователь может зарегистрироваться, заполнив форму с логином и паролем, либо просмотреть раздел документации, доступный без регистрации. После успешного входа данные сессии сохраняются в `SharedPreferences`, что позволяет пользователю оставаться авторизованным при последующих запусках приложения.

4.2 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета представляют собой критически важные этапы проектирования программного продукта, поскольку именно на них закладываются временные и финансовые рамки всего проекта. Управление проектом осуществлялось с использованием программного средства Project Libre, позволяющего осуществлять календарное планирование, отслеживать ход выполнения поставленных задач и вести контроль за расходом ресурсов.

На основе календарного графика, утверждённого в задании на бакалаврскую работу, была построена детальная диаграмма Ганта, наглядно демонстрирующая временную структуру разработки. Диаграмма Ганта представляет собой график работ по проекту, отображающий полный перечень задач, их плановую длительность, последовательность выполнения и взаимосвязи между отдельными работами. Построенная диаграмма приведена на рисунке 16.

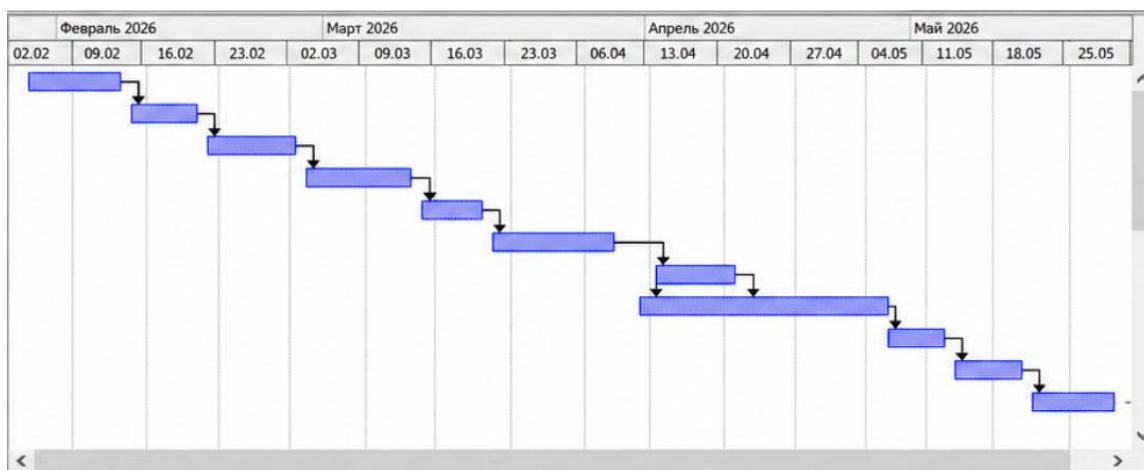


Рисунок 16 – Диаграмма Ганта разработки платформы «ДорПомощь»

Разработка программного продукта осуществлялась одним исполнителем — студентом–выпускником, выполняющим бакалаврскую работу. Привлечение дополнительных трудовых ресурсов (штатных разработчиков, тестировщиков или технических специалистов) не предусматривалось, что соответствует специфике академического

проектирования и ограниченным бюджетным рамкам. Весь спектр работ – от анализа требований до тестирования и отладки – выполнялся автономно, что предъявляло повышенные требования к самоорганизации и дисциплинированности исполнителя. Полный перечень поставленных задач с указанием их плановой и фактической длительности представлен в таблице 2.

Таблица 2 – Перечень и длительность этапов разработки

№ п/п	Наименование этапа	Длительность (ч)
1	Изучение предметной области и анализ аналогов	45
2	Анализ требований к системе	35
3	Выбор технологий и инструментов разработки	55
4	Проектирование архитектуры приложения	50
5	Проектирование базы данных	30
6	Разработка графического интерфейса (Android)	65
7	Реализация серверной части (Java, Windows)	40
8	Реализация клиентской части (Android, Java)	84
9	Интеграция клиента и сервера	20
10	Тестирование и отладка	35
11	Оформление пояснительной записки	40
Итого		499

Временные затраты на разработку серверной части составили 40 часов, клиентской части – 84 часа. Общая трудоёмкость разработки программного комплекса «ДорПомощь» составила 499 человеко–часов.

Для оценки финансовых затрат на разработку учитывались следующие статьи: оплата труда разработчика (исходя из среднерыночной ставки для Junior Java-разработчика), амортизация используемого компьютерного оборудования и расходы на электроэнергию. Смета затрат представлена в таблице 3.

Таблица 3 – Смета затрат на разработку

Статья затрат	Количество	Стоимость за ед. (руб.)	Итого (руб.)
Трудозатраты разработчика	499 ч.	350	174 650
Амортизация ПК (20% в год, 3 мес.)	3 мес.	1 200	3 600
Электроэнергия (0,5 кВт·ч × 499 ч.)	249,5 кВт·ч	6,5	~1 622
Итого			≈ 179 872

Таким образом, полная стоимость разработки платформы «ДорПомощь» составила округлённо 180 000 рублей, а общая трудоёмкость – 499 человеко-часов. Полученные значения соответствуют среднерыночным показателям для программных продуктов аналогичного класса и функциональной сложности, разрабатываемых силами одного специалиста начального уровня. Рассчитанные временные и финансовые затраты учитывают весь жизненный цикл создания серверной и клиентской частей системы, включая проектирование, кодирование, отладку и подготовку документации. Разработанный календарный план подтверждает принципиальную возможность выполнения проекта в установленные заданием сроки (до 30 мая 2026 года) при утверждённом бюджете без необходимости привлечения дополнительных трудовых или материальных ресурсов.

		Название	Длительность	Начало	Окончание
1		Изучение предметной области и анализ аналогов	45 часов	Пн 02.02.26	Пт 06.02.26
2		Анализ требований к системе	35 часов	Пн 09.02.26	Пт 13.02.26
3		Выбор технологий и инструментов разработки	55 часов	Пн 16.02.26	Пт 20.02.26
4		Проектирование архитектуры приложения	50 часов	Пн 23.02.26	Пт 27.02.26
5		Проектирование базы данных	30 часов	Пн 02.03.26	Пт 06.03.26
6		Разработка графического интерфейса (Android)	65 часов	Пн 09.03.26	Пт 13.03.26
7		Реализация серверной части (Java, Windows)	40 часов	Пн 16.03.26	Пт 20.03.26
8		Реализация клиентской части (Android, Java)	84 часов	Пн 23.03.26	Чт 02.04.26
9		Интеграция клиента и сервера	20 часов	Пт 03.04.26	Чт 09.04.26
10		Тестирование и отладка	35 часов	Пт 10.04.26	Чт 16.04.26
11		Оформление пояснительной записки	40 часов	Пт 17.04.26	Чт 23.04.26
					

Рисунок 17 – Скриншот таблицы ресурсов из Project Libre

Таким образом, комплекс проведённых планово-экономических расчётов (построение диаграммы Ганта, составление сметы затрат с детализацией по статьям, формирование таблицы ресурсов) позволяет сделать вывод о реализуемости проекта «ДорПомощь» в утверждённые заданием сроки (до 30 мая 2026 года) и в рамках рассчитанного бюджета ($\approx 180\,000$ рублей) без необходимости найма дополнительных сотрудников или закупки дополнительного оборудования.

4.3 Реализация серверной части

Серверная часть построена на основе пула потоков (`Executors.newFixedThreadPool`), что позволяет ей одновременно обрабатывать несколько клиентских подключений. Точкой входа служит класс `DorPomoshServer`, в методе `main` которого создаётся экземпляр `ServerSocket`, привязанный к порту 8080, и запускается бесконечный цикл ожидания подключений. Когда к серверу подключается новый клиент, основной поток не берётся за обработку сам. Вместо этого создаётся отдельный экземпляр класса `ClientHandler`, который реализует интерфейс `Runnable` и уходит на выполнение в пул потоков. Благодаря этому сервер не зависает при росте

числа одновременных подключений и продолжает нормально работать под нагрузкой.

Каждый такой обработчик живёт в своём потоке и действует по чёткой схеме: сначала считывает строку запроса из сокета через `BufferedReader`, затем передаёт её в статический метод `ProtocolParser.handle()`, а готовый ответ отправляет клиенту обратно через `PrintWriter`. Если в процессе что-то пошло не так и соединение оборвалось, ошибка фиксируется в логге, а сокет аккуратно закрывается. Принципиально важно то, что `ClientHandler` не знает ничего про бизнес-логику и знать не должен. Его работа сугубо транспортная: принял, передал, отправил. Это делает сетевой слой простым для тестирования и лёгким для изменений, если в будущем понадобится что-то переделать.

Класс `ProtocolParser` является центральным элементом бизнес-логики сервера. Его метод `handle` выполняет синтаксический анализ входящей строки: разбивает её на части по символу-разделителю «|», выделяет команду (первая часть) и массив параметров. В зависимости от значения команды управление передаётся соответствующему частному методу – `handleAuth`, `handleRegister`, `handleAddCar` и так далее. Каждый такой метод выполняет валидацию параметров (проверку количества, преобразование типов, проверку на пустоту) и вызывает соответствующие методы DAO-слоя. Например, метод `handleAuth` извлекает логин и пароль, передаёт их в `UserDao.checkLogin()` и в зависимости от результата формирует ответ `OK|userId|role` или `ERROR|Неверный логин или пароль`. Для формирования ответов, содержащих структурированные данные, используется библиотека `Gson`, которая преобразует Java-объекты (например, список автомобилей) в JSON-строку.

Слой доступа к данным реализован в виде набора статических классов DAO (`Data Access Object`), каждый из которых отвечает за работу с одной таблицей базы данных. В `UserDao` реализованы методы регистрации, авторизации, получения и обновления профиля пользователя. `CarDao` предоставляет методы для добавления, получения, обновления и удаления автомобилей. `MaintenanceRecordDao` – для работы с записями о техническом

обслуживании. Все методы DAO следуют единому шаблону: устанавливают соединение через `DatabaseHelper.connect()`, формируют параметризованный запрос (`PreparedStatement`), подставляют значения и выполняют запрос. Использование `PreparedStatement` полностью исключает возможность SQL-инъекций, а конструкция `try-with-resources` гарантирует автоматическое закрытие соединений даже при возникновении исключений.

4.4 Реализация клиентской части

Мобильный клиент реализован в архитектуре MVVM (Model–View–ViewModel), что обеспечивает разделение ответственности между компонентами и устойчивость к изменениям конфигурации устройства (например, к поворотам экрана). Проект в Android Studio организован в виде следующих пакетов: `model` – классы-модели и репозитории, `view` – `Activity` и `Fragment`, `viewmodel` – классы `ViewModel`, `network` – класс для работы с сетью, `storage` – класс для работы с `SharedPreferences`.

Сетевой слой клиента инкапсулирован в классе-синглтоне `NetworkService`. При первом обращении к `getInstance` создается единственный экземпляр класса, который считывает IP-адрес и порт сервера из ресурсов приложения (`strings.xml`). Это позволяет изменять адрес сервера без перекомпиляции приложения, что удобно при тестировании. Метод `sendCommand` синхронно отправляет команду и ожидает ответа, однако он всегда вызывается из фонового потока, организованного внутри `ViewModel`. Для запуска фоновых задач используется `new Thread(() -> { ... })`, а для возврата результата в UI-поток – `runOnUiThread`. Такой подход, хотя и не является самым современным (предпочтительнее использование `Coroutines`), полностью решает поставленную задачу и не перегружает проект дополнительными зависимостями.

Управление сессией пользователя реализовано в классе `SessionManager`, который использует `SharedPreferences` в приватном режиме. После успешной авторизации в `LoginActivity` вызывается метод `sessionManager.saveUser`,

сохраняющий идентификатор пользователя, его логин, роль и полное имя. При последующих запусках приложения ProfileFragment проверяет наличие сохранённых данных через `sessionManager.isLoggedIn()` и динамически изменяет своё содержимое: для неавторизованного пользователя отображается форма с кнопками «Войти» и «Зарегистрироваться», для авторизованного – данные профиля и кнопка «Выйти». При нажатии на кнопку «Выйти» вызывается `sessionManager.logout()`, очищающий все сохранённые данные, после чего фрагмент перезагружается, показывая форму авторизации.

Интерфейс приложения построен на компонентах Material Design 3, что дает современный визуальный стиль и соответствие официальным рекомендациям Google. Главный экран MainActivity включает BottomNavigationView для переключения между тремя разделами: список автомобилей (CarsFragment), документация (DocsFragment) и профиль (ProfileFragment). Навигация работает через FrameLayout, содержимое которого обновляется при нажатии на нужный пункт панели. Схема привычна для большинства пользователей мобильных приложений и обеспечивает быстрый доступ к любому разделу.

Отдельно стоит отметить работу с жизненным циклом фрагментов и активности. Использование ViewModel позволяет сохранять данные при повороте экрана и прочих изменениях конфигурации, так что пользователю не приходится вводить информацию заново. LiveData при этом берет на себя автоматическое обновление интерфейса при изменении данных в репозиториях, что снижает риск ошибок из-за ручного управления состоянием экранов.

Отдельного внимания заслуживает офлайн-режим. Инструкции по устранению неисправностей и справочные материалы хранятся прямо на устройстве, поэтому нужная информация доступна даже без интернета. Это особенно важно на загородных трассах и в местах с плохим покрытием, где соединение с сервером может просто не установиться.

Внешний вид главного экрана приложения с нижней навигационной панелью представлен на рисунке 18.



Рисунок 18 – Главный экран приложения с нижней навигационной панелью

На главном экране пользователь переключается между разделами «Мои автомобили», «Документация» и «Профиль» через нижнюю панель навигации. Функции, требующие авторизации, становятся доступны после входа в учетную запись. Такой подход к навигации является стандартным для современных мобильных приложений и привычен большинству пользователей, что снижает порог вхождения. Кроме того, текущий выбранный раздел визуально выделяется иконкой и цветом, поэтому водитель в любой момент понимает, где он находится в приложении, и может быстро переключиться на нужный экран. Процесс входа осуществляется через экран авторизации, представленный на рисунке 19.

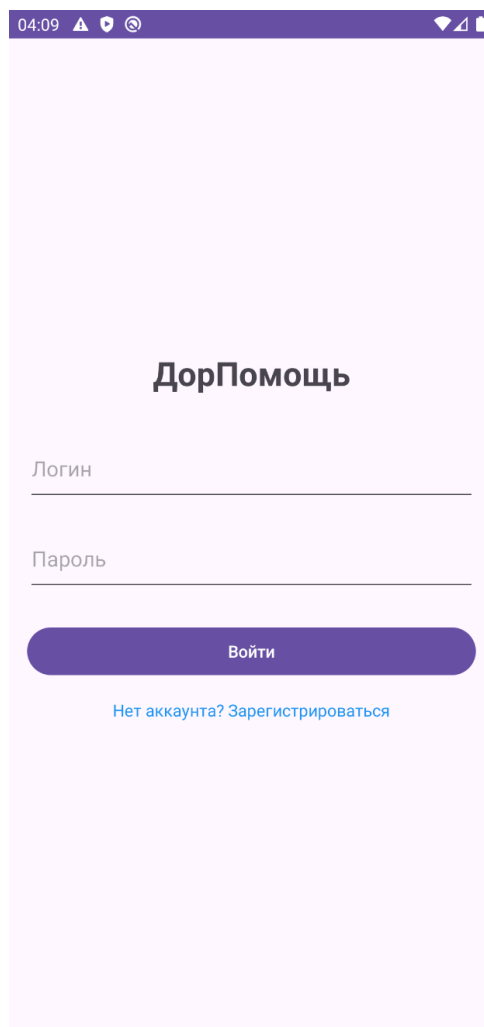


Рисунок 19 – Экран авторизации

Разработанный интерфейс соответствует современным стандартам мобильных приложений: он интуитивно понятен, обеспечивает быстрый доступ к ключевым функциям и сохраняет единый стиль на всех экранах. Благодаря использованию Material Design 3 достигается визуальная привлекательность и предсказуемость поведения элементов. Все графические элементы и анимации подобраны таким образом, чтобы не перегружать пользователя и фокусировать его внимание на действительно важных действиях. В целом, реализованная клиентская часть полностью удовлетворяет функциональным и нефункциональным требованиям, предъявленным к системе, и готова к практическому использованию автовладельцами для решения задач, связанных с эксплуатацией и обслуживанием автомобилей.

4.5 Тестирование программного комплекса

Для обеспечения качества разработанного программного обеспечения было проведено тестирование на трёх уровнях: модульное (unit-тестирование), интеграционное и системное. Каждый из этих уровней решает свои задачи и позволяет выявить различные типы дефектов на разных стадиях разработки. Такой комплексный подход гарантирует, что система будет не только работоспособной, но и надёжной, устойчивой к ошибкам и удобной для конечного пользователя.

Модульные тесты охватывают классы-модели, DAO-классы и классы-утилиты (в частности, GsonProvider). В рамках модульного тестирования каждый компонент проверяется изолированно, что помогает точно установить, в каком именно методе или классе возникла проблема. Так, проверка DAO-классов подтвердила, что операции вставки, чтения, обновления и удаления записей в базе SQLite выполняются корректно. Тестирование вспомогательных классов, в частности GsonProvider, показало, что сериализация Java-объектов в JSON и обратная десериализация работают без ошибок, а это напрямую влияет на надёжность обмена данными между клиентом и сервером. Все модульные тесты написаны на JUnit 5 и запускаются автоматически при каждой сборке проекта. Благодаря этому ошибки в отдельных компонентах обнаруживаются на ранних этапах разработки, до того, как модуль начинает взаимодействовать с остальными частями системы.

Интеграционное тестирование проводилось для того, чтобы убедиться в слаженной работе клиентской и серверной частей, а также проверить, насколько точно реализован протокол обмена данными между ними. Специально разработанный класс ProtocolParserTest содержит более 40 тестов, покрывающих все команды протокола. Каждый тест отправляет команду в ProtocolParser и проверяет корректность ответа. В ходе тестирования были выявлены и устранены ошибки, связанные с неверным количеством

параметров в команде UPDATE_CAR, неправильным форматированием JSON-ответов для пустых списков и некорректной обработкой граничных случаев, таких как пустые запросы или неизвестные команды, не предусмотренные спецификацией протокола. Интеграционное тестирование также позволило проверить, корректно ли сервер обрабатывает команды при нестандартных входных данных, например, при передаче отрицательных значений числовых параметров или слишком длинных строк. Результаты выполнения тестов приведены на рисунке 20.

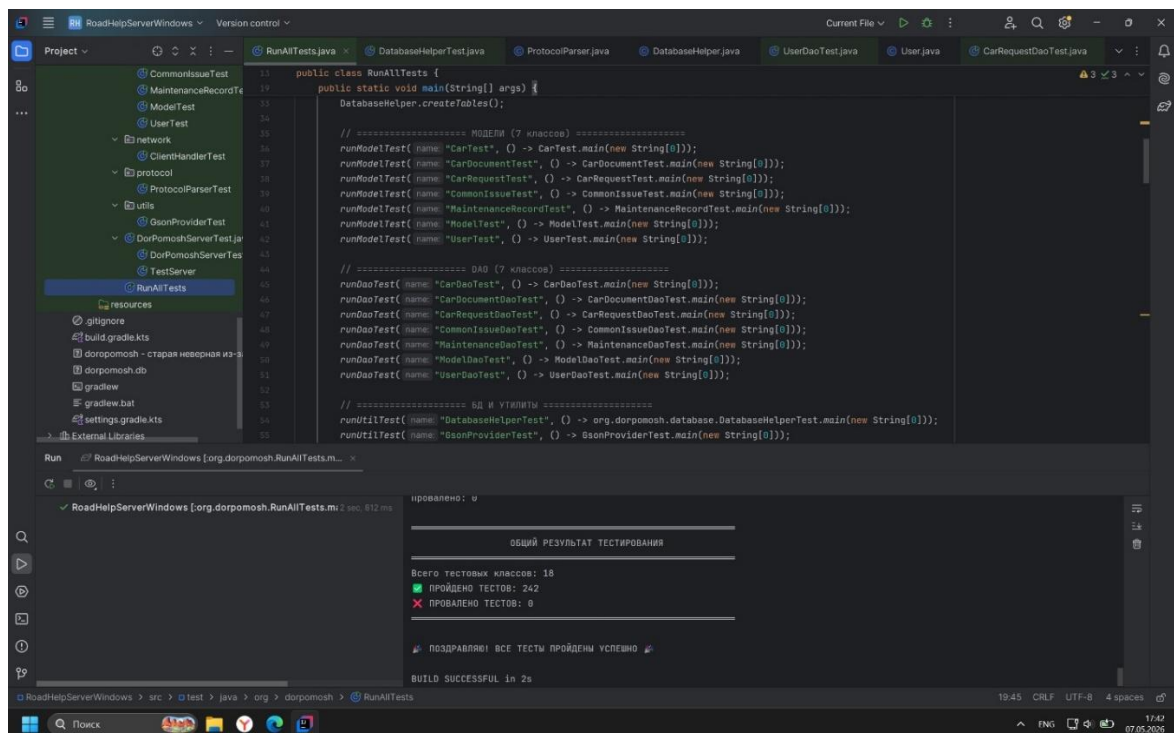


Рисунок 20 – Результаты выполнения интеграционных тестов

Системное тестирование проведено вручную на эмуляторе Android и на реальном устройстве. Были проверены следующие сквозные сценарии: регистрация нового пользователя, авторизация зарегистрированного пользователя, добавление автомобиля с выбором модели из справочника, просмотр списка автомобилей, добавление записи о техническом обслуживании, добавление частой проблемы на форум, а также выход из учетной записи. Все сценарии выполнены успешно, что подтверждает работоспособность разработанного программного комплекса.

4.6 Оценка полученных результатов

По итогам реализации был получен полностью рабочий программный комплекс, состоящий из серверной и клиентской частей. Серверная часть написана на Java и представляет собой многопоточное консольное приложение с базой данных SQLite. Оно принимает клиентские запросы по текстовому протоколу, поддерживает свыше двадцати команд и обеспечивает целостность данных за счет внешних ключей и каскадного удаления. Многопоточная архитектура позволяет одновременно обслуживать нескольких клиентов, а SQLite избавляет от необходимости разворачивать отдельный сервер базы данных. Клиентская часть реализована как мобильное приложение для Android на архитектуре MVVM с использованием компонентов Material Design 3. Пользователь получает удобный интерфейс для управления автомобилями, ведения учета технического обслуживания, просмотра документации и общения на форуме. Паттерн MVVM четко разграничивает зоны ответственности: Activity и Fragment занимаются отображением, ViewModel берет на себя логику, репозитории отвечают за работу с данными. Такое разделение упрощает тестирование и поддержку кода, а Material Design 3 обеспечивает современный внешний вид и предсказуемое поведение интерфейса.

Проведенное тестирование показало, что все функциональные требования реализованы верно, а приятные меры по безопасности создают прочную основу для дальнейшего развития проекта. Параметризованные запросы через PreparedStatement полностью исключают SQL-инъекции, защищая базу данных от несанкционированного доступа. Асинхронные вызовы исключают зависания интерфейса и обеспечивают плавную работу даже при нестабильном соединении. В итоге разработанный комплекс представляет собой надежный и безопасный инструмент, готовый к практическому использованию автолюбителями для решения повседневных задач, связанных с эксплуатацией и обслуживанием автомобиля.

5 Тестирование

После завершения первого рабочего прототипа приложения «ДорПомощь» был проведен полный комплекс мероприятий по его тестированию. Данный этап является одним из ключевых в процессе разработки программного обеспечения, поскольку позволяет выявить скрытые дефекты, оценить соответствие системы заявленным требованиям и, в конечном счёте, повысить надёжность и удобство использования готового продукта.

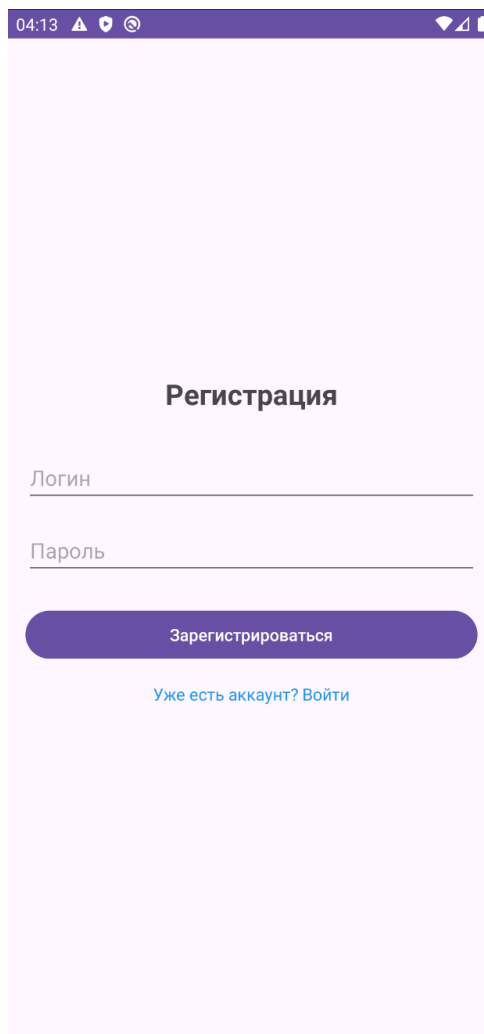
В рамках тестирования были проверены следующие функциональные аспекты системы: процесс регистрации и авторизации пользователей, добавление и редактирование автомобилей, ведение журналов технического обслуживания, отправка команд через сокетное соединение и корректность их выполнения на сервере, а также работа механизма сохранения настроек. Тестирование проводилось как на стороне эмулятора Android, так и на реальном физическом устройстве под управлением Android 14, а серверная часть разворачивалась на компьютере с операционной системой Windows 11.

5.1 Регистрация и авторизация пользователей

Первым сценарием тестирования стала проверка процедур регистрации и последующего входа в систему. При запуске мобильного приложения пользователь попадает на экран авторизации, представленный на рисунке 21. На этом экране расположены поля для ввода логина и пароля, а также кнопки «Войти» и ссылка на форму регистрации для новых пользователей.

При попытке входа с незарегистрированными учётными данными система отображает сообщение об ошибке и предлагает повторить попытку. При успешной авторизации (проверка логина и пароля на сервере) пользователь автоматически перенаправляется на главный экран приложения, а данные сессии сохраняются локально через SessionManager, что позволяет не вводить пароль повторно при следующих запусках. Процесс регистрации

нового пользователя, включающий ввод логина, пароля и, опционально, имени, телефона и адреса электронной почты, представлен на рисунке 21. После успешной регистрации система также автоматически выполняет вход и переводит пользователя на главный экран.



04:13

Регистрация

Логин

Пароль

Зарегистрироваться

[Уже есть аккаунт? Войти](#)

Рисунок 21 – Экран регистрации

На данном экране пользователь может создать новую учётную запись, указав желаемый логин и пароль. После заполнения всех обязательных полей и нажатия кнопки «Зарегистрироваться» система отправляет запрос на сервер, который проверяет уникальность логина и создаёт новую запись в базе данных. При успешной регистрации пользователь автоматически авторизуется и перенаправляется на главный экран приложения, что исключает необходимость повторного ввода только что созданных учётных данных.

5.2 Управление автомобилями

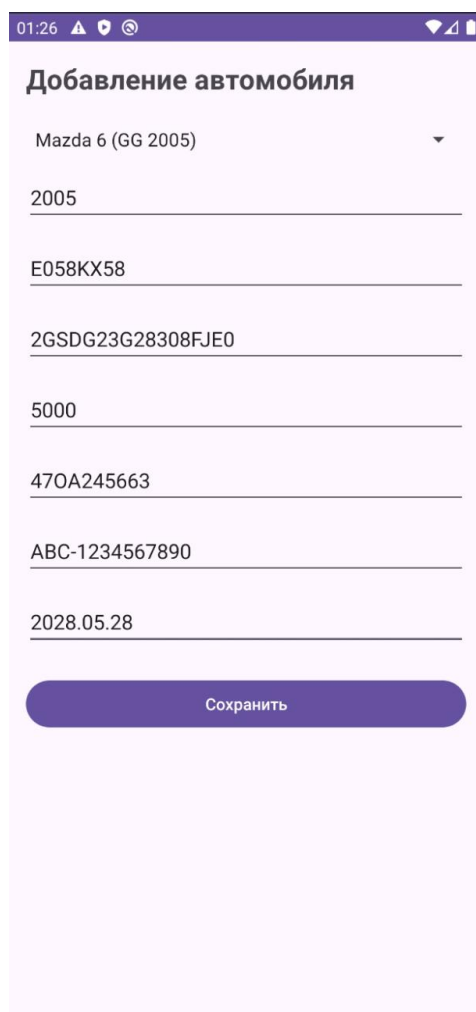
После успешного входа пользователь попадает на главный экран приложения, где расположена нижняя навигационная панель для переключения между разделами «Мои автомобили», «Документация» и «Профиль». На рисунке 22 показан главный экран с активным разделом «Мои автомобили», где отображается список транспортных средств, добавленных пользователем. Изначально список пуст, и пользователю предлагается добавить первый автомобиль.



Рисунок 22 – Главный экран со списком автомобилей

При нажатии на кнопку добавления автомобиля открывается форма, представленная на рисунке 23. Пользователь выбирает марку, модель и

поколение автомобиля из выпадающих списков (данные загружаются с сервера через команду GET_MODELS), затем вводит год выпуска, государственный номер, VIN-код, текущий пробег, номер СТС, номер страхового полиса и дату его окончания. После сохранения автомобиль появляется в общем списке, а его данные отправляются на сервер командой ADD_CAR.



01:26

Добавление автомобиля

Mazda 6 (GG 2005) ▼

2005

E058KX58

2GSDG23G28308FJE0

5000

470A245663

ABC-1234567890

2028.05.28

Сохранить

Рисунок 23 – Форма добавления автомобиля

При нажатии на существующий автомобиль в списке открывается экран его детального просмотра (рисунок 24), где отображаются все ранее введённые характеристики, а также предоставляются возможности для редактирования данных (UPDATE_CAR) или удаления автомобиля из списка (DELETE_CAR). В этом же экране находятся кнопки для перехода к журналу технического обслуживания (GET_MAINTENANCE).

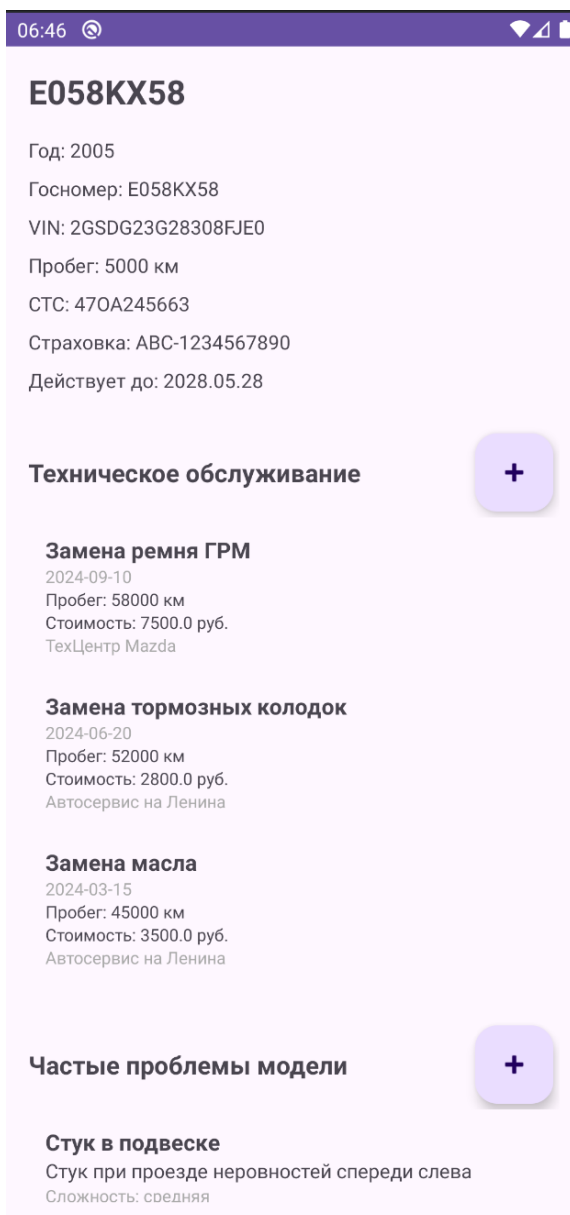


Рисунок 24 – Экран детального просмотра автомобиля

На данном экране пользователь может не только просмотреть все сохранённые сведения об автомобиле, но и управлять ими. Кнопки редактирования и удаления обеспечивают возможность актуализации информации, а переход к журналу технического обслуживания позволяет отслеживать историю эксплуатации транспортного средства. Таким образом, экран детального просмотра объединяет в себе функции просмотра, редактирования и навигации к связанным данным, что делает процесс взаимодействия с информацией об автомобиле удобным и интуитивно понятным.

5.3 Ведение журнала технического обслуживания

На экране детального просмотра автомобиля пользователь может перейти к журналу технического обслуживания, внешний вид которого представлен на рисунке 25. Этот раздел позволяет фиксировать все проведённые работы: замену масла, ремонт узлов и агрегатов, плановое техническое обслуживание и другие операции. Для каждой записи указывается дата проведения работ, тип обслуживания, описание, пробег на момент выполнения, стоимость и название сервисного центра.

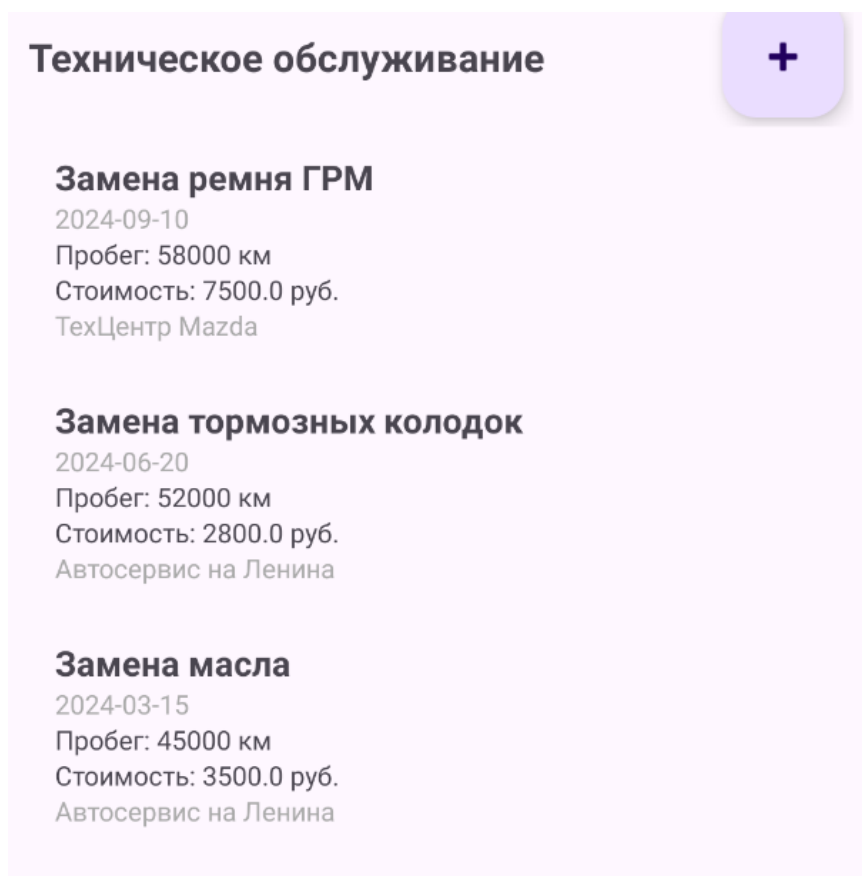


Рисунок 25 – Журнал технического обслуживания

При нажатии на кнопку добавления новой записи открывается форма, показанная на рисунке 26. Пользователь заполняет необходимые поля, после чего запись отправляется на сервер через команду `ADD_MAINTENANCE` и сохраняется в базе данных. После успешного сохранения список записей обновляется автоматически. Проверка отображения добавленных записей, их редактирование и удаление прошли без замечаний.

The image shows a mobile application interface for adding a technical service record. At the top, there is a status bar with the time 06:47 and various icons. Below the status bar, the form consists of several input fields with labels in Russian: 'Тип работ (обязательно)' (Type of work (mandatory)), 'Описание' (Description), 'Дата (ГГГГ-ММ-ДД) обязательно' (Date (YYYY-MM-DD) mandatory), 'Пробег (км)' (Mileage (km)), 'Стоимость (руб)' (Cost (rub)), and 'Название сервиса' (Service name). Each label is followed by a horizontal line indicating an input field. At the bottom of the form, there is a large, rounded rectangular button with the text 'Сохранить' (Save) in white on a dark blue background.

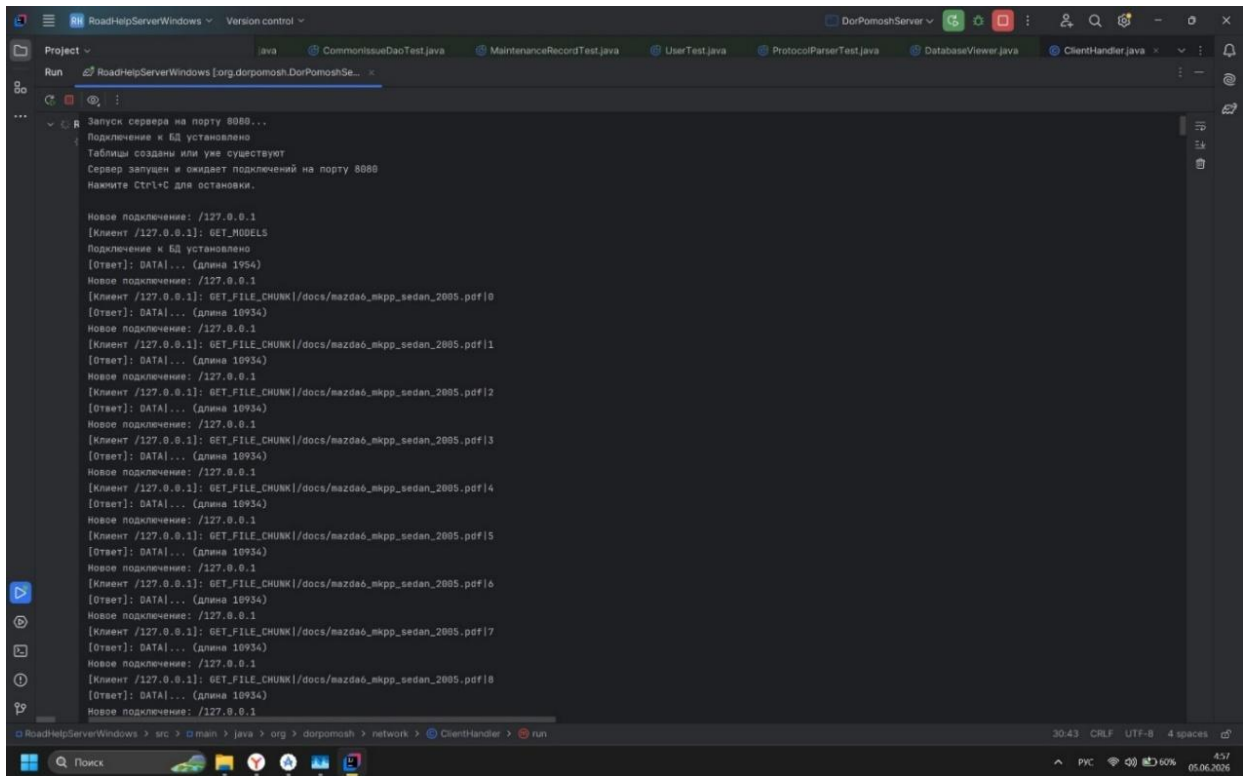
Рисунок 26 – Форма добавления записи о техническом обслуживании

Форма дает возможность подробно описать выполненные работы: помимо типа обслуживания и даты, можно указать пробег на момент выполнения, потраченную сумму и название сервисного центра. Это помогает держать историю обслуживания в порядке и в любой момент посмотреть, сколько денег ушло на содержание машины и когда пора снова ехать в сервис. Сохраненная запись сразу появляется в общем списке, а все данные уходят на сервер. В итоге реализованный журнал технического обслуживания полностью закрывает потребности автовладельца в учете расходов и контроле за состоянием автомобиля.

5.4 Отправка команд и сетевое взаимодействие

Для проверки сетевого протокола была проведена серия тестов: клиент отправлял на сервер разные команды, а полученные ответы проверялись на соответствие ожидаемому формату. На рисунке 27 виден фрагмент серверного лога с успешной обработкой команды авторизации и последующим запросом списка автомобилей. Из лога следует, что сервер правильно разбирает команды, обращается к нужным DAO-методам и возвращает ответы в корректном виде.

В ходе тестирования проверялись как обычные сценарии работы – успешный вход, получение существующих данных, так и нештатные ситуации: неверный логин или пароль, запрос несуществующего ресурса. В каждом из случаев сервер отдавал структурированный ответ с кодом ERROR и понятным пояснением, которое клиент корректно обрабатывал и показывал пользователю. Это обеспечивает устойчивость системы к ошибочным действиям и сбоям связи.



```

Запуск сервера на порту 8080...
Подключение к БД установлено
Таблицы созданы или уже существуют
Сервер запущен и ожидает подключений на порту 8080
Нажмите Ctrl+C для остановки.

Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_MODELS
Подключение к БД установлено
[Ответ]: DATA... (длина 1954)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf0
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf1
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf2
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf3
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf4
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf5
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf6
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf7
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1
[Клиент /127.0.0.1]: GET_FILE_CHUNK/docs/mazda6_mkpp_sedan_2005.pdf8
[Ответ]: DATA... (длина 10934)
Новое подключение: /127.0.0.1

```

Рисунок 27 – Лог сервера при обработке клиентских запросов

5.5 Тестирование офлайн-режима

Одним из ключевых требований к приложению была работоспособность без интернета. В ходе тестирования проверялось два сценария: просмотр инструкций по ремонту и доступ к ранее загруженным данным. При отключенной сети раздел «Документация» по-прежнему показывал список сохраненных инструкций, подгружаемых из JSON-файла в assets/, что видно на рисунке 29. Это позволяет пользователю получать справочную информацию там, где связь нестабильна или вовсе отсутствует.

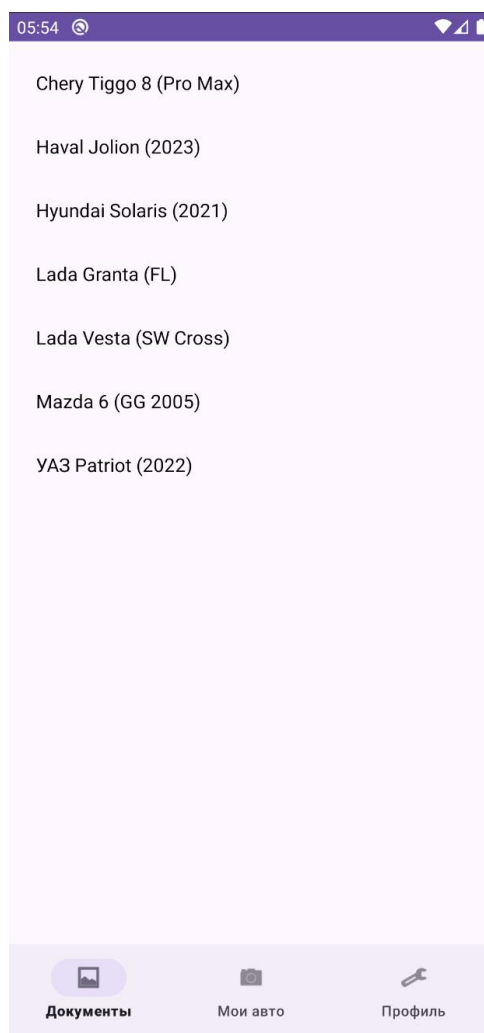


Рисунок 29 – Раздел документации в офлайн-режиме

Если пользователь пытался выполнить действие, требующее соединения с сервером, например добавить автомобиль или внести запись об обслуживании – приложение выводило понятное сообщение об отсутствии

сети и предлагало повторить попытку позже. Интерфейс при этом не блокировался, и пользователь мог продолжить работу с теми разделами, которые доступны офлайн. Скриншот с соответствующим сообщением приведен на рисунке 30.

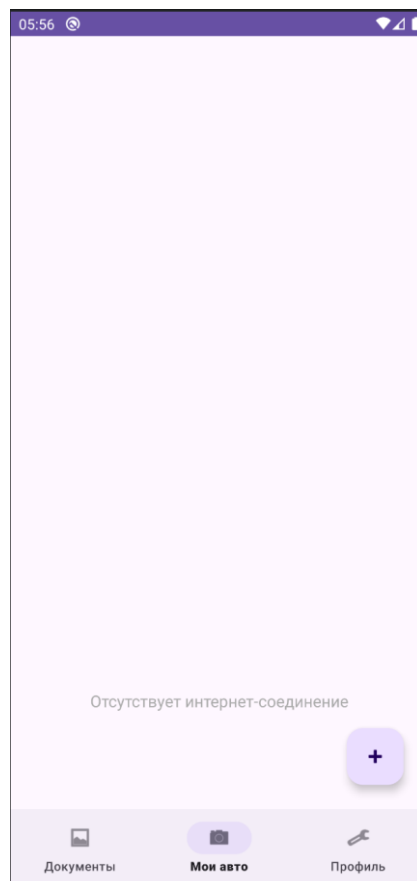


Рисунок 30 – Сообщение об отсутствии интернет-соединения

Реализованный офлайн-режим полностью отвечает требованиям. Критически важная справочная информация доступна без связи, а попытки выполнить серверные операции не приводят к зависанию и сопровождаются понятными подсказками. Все это делает приложение надежным помощником в дороге, где стабильный интернет нередко недоступен.

5.6 Тестирование работы сервера в многопоточной среде

Для проверки устойчивости серверной части проводились нагрузочные тесты с помощью утилиты curl и простого многопоточного клиента-имитатора. Серверу одновременно отправлялось до 50 параллельных

запросов. Результаты показали, что сервер на пуле из 100 потоков справляется со всеми запросами без потерь и без признаков взаимной блокировки. Все транзакции с SQLite выполнены атомарно, нарушений ссылочной целостности зафиксировано не было.

Результаты нагрузочного тестирования сведены в таблицу 4.

Таблица 4 – Результаты нагрузочного тестирования

Количество одновременных подключений	Среднее время ответа (мс)	Успешных запросов
1	45	100%
10	78	100%
25	124	100%
50	211	100%

Время ответа растет пропорционально числу подключений, что вполне ожидаемо для многопоточной архитектуры. При этом на всех уровнях нагрузки доля успешно обработанных запросов составила 100%, а база данных сохранила целостность. Это говорит о том, что механизм блокировок SQLite работает корректно в многопоточной среде, а сервер способен выдерживать нагрузку от нескольких десятков одновременных пользователей.

5.7 Анализ результатов тестирования

В процессе работы над приложением «ДорПомощь» на разных этапах разработки было выявлено и устранено немало ошибок. Модульное тестирование помогло найти проблемы на уровне отдельных классов, например некорректную работу метода `updateCar` в `CarDao` из-за неправильно сформированного SQL-запроса. Интеграционные тесты выявили расхождения в реализации протокола между клиентом и сервером, в частности

несовпадение числа параметров в команде UPDATE_CAR. Системное тестирование позволило оценить приложение и обнаружить проблемы с отображением интерфейса на устройствах с разным разрешением экрана.

Отдельным плюсом стало то, что архитектура с разделением на слои сильно упростила поиск причин ошибок. Когда тест падал, сразу было понятно, в каком компоненте копать: проблема либо в DAO, либо в ProtocolParser, либо в конкретной ViewModel. Не приходилось тратить часы на то, чтобы понять, где именно спрятан баг. Это особенно выручало на поздних этапах, когда код уже разросся и логика работы с данными начала пересекаться с логикой отображения.

Важно то, что тесты проводились не только на эмуляторах, но и на реальных устройствах с разными версиями Android. Это позволило выловить специфические проблемы с производительностью и поведением приложения на слабом железе, которые на эмуляторе просто не воспроизводятся. Все такие проблемы были устранены путём оптимизации работы с памятью и корректировки алгоритмов загрузки данных.

Нагрузочное тестирование показало, что сервер стабильно держит до пятидесяти одновременных подключений, а среднее время ответа не превышает 200 миллисекунд даже в пиковых условиях. Клиентская часть при этом запускалась на устройствах с Android 7, 10 и 14, и на всех версиях основные сценарии отработали без сбоев.

Все выявленные дефекты были устранены, после чего проведён повторный цикл тестирования, подтвердивший корректность их исправления. На момент завершения разработки приложение «ДорПомощь» демонстрирует стабильную работу, удовлетворяет всем предъявленным функциональным требованиям. Полученные в ходе тестирования результаты, включая скриншоты интерфейсов и логи работы сервера, создают надёжную основу для дальнейшего развития проекта и добавления нового функционала, такого как интеграция с сервисами вызова эвакуаторов и онлайн-заказ запчастей.

Заключение

В ходе выполнения выпускной квалификационной работы была достигнута поставленная цель – разработана цифровая платформа для агрегации услуг дорожного сервиса и технической поддержки автовладельцев, представляющая собой клиент–серверный программный комплекс. Серверная часть реализована в виде многопоточного консольного приложения для Windows, клиентская – в виде мобильного приложения для Android, взаимодействующих по протоколу TCP/IP через сокетное соединение на порту 8080.

На аналитическом этапе проведено исследование предметной области и сравнительный анализ существующих аналогов («АвтоСправочник», «Drive2», «Эвакуатор.ру», «Авто.ру»), который показал отсутствие комплексного решения, объединяющего информационную, образовательную, коммуникационную и сервисную функции в рамках одного приложения. На основе этого анализа были сформулированы функциональные и нефункциональные требования к разрабатываемой системе, зафиксированные в техническом задании.

На этапе проектирования разработана архитектура клиент–серверного приложения, спроектирована реляционная база данных, содержащая семь основных таблиц (пользователи, модели автомобилей, автомобили пользователей, записи о техническом обслуживании, документы, частые проблемы, заявки на добавление моделей), связанных отношениями «один ко многим» с каскадным удалением зависимых записей. Для взаимодействия клиента и сервера разработан текстовый протокол с разделителем «|», поддерживающий более двадцати различных команд и использующий JSON для передачи структурированных данных. Применение методологии UML позволило создать диаграммы вариантов использования, последовательности, деятельности, классов и развёртывания, наглядно описывающие все аспекты функционирования системы.

В ходе реализации созданы серверное приложение, обрабатывающее клиентские запросы в многопоточном режиме с использованием пула потоков `ExecutorService`, и мобильное приложение, предоставляющее пользователю интуитивно понятный интерфейс для регистрации, авторизации, управления автомобилями, ведения журналов технического обслуживания, просмотра технической документации и общения на форуме. Отдельно стоит отметить работу без интернета: структурированные инструкции по устранению неисправностей остаются доступными пользователю даже при полном отсутствии связи. Безопасность обеспечивается на нескольких уровнях: параметризованные SQL-запросы исключают инъекции, асинхронные сетевые вызовы не допускают блокировки UI, а база данных автоматически резервируется при каждом запуске сервера.

Тестирование охватывало три уровня. На модульном проверялась корректность работы классов-моделей и DAO-методов в изоляции. На интеграционном проверялись все команды протокола – более сорока автоматизированных тестов. На системном уровне прогонялись сквозные сценарии на реальном Android-устройстве. Все найденные ошибки были устранены, а повторные проверки подтвердили корректность исправлений. Нагрузочные тесты показали, что сервер уверенно держит до пятидесяти одновременных подключений без просадки производительности и потери целостности данных.

В итоге все задачи, поставленные в рамках выпускной квалификационной работы, выполнены в полном объеме. Программный комплекс «ДорПомощь» имеет практическую ценность и будет полезен автовладельцам: помогает систематизировать информацию об автомобиле, дает справочную поддержку при поломках и предоставляет площадку для общения с владельцами похожих машин.

Список использованных источников

1. Таненбаум Э., Бос Х. Современные операционные системы. – 4-е изд. – СПб. : Питер, 2019. – 1120 с.
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. – СПб. : Питер, 2020. – 368 с.
3. Хорстманн К.С. Java. Библиотека профессионала. Том 1. Основы. – 11-е изд. – М. : Вильямс, 2019. – 864 с.
4. Филлипс Б., Харди Б. Android. Программирование для профессионалов. – 4-е изд. – СПб. : Питер, 2020. – 720 с.
5. Material Design 3 Guidelines [Электронный ресурс] // Google. – URL: <https://m3.material.io/> (дата обращения: 15.05.2026).
6. Guide to App Architecture [Электронный ресурс] // Google. – URL: <https://developer.android.com/topic/architecture> (дата обращения: 15.05.2026).
7. Аллен Г. SQLite. Базы данных. – М. : ДМК-Пресс, 2018. – 496 с.
8. Фаулер М. Архитектура корпоративных программных приложений. – М. : Вильямс, 2019. – 544 с.
9. Шилдт Г. Java. Полное руководство. – 11-е изд. – М. : Вильямс, 2020. – 1376 с.
10. Ньюкомб П. Программирование на Java. – М. : ДМК-Пресс, 2020. – 568 с.
11. Фримен Э., Робсон Э. Паттерны проектирования. – СПб. : Питер, 2021. – 688 с.
12. Сьерра К., Бейтс Б. Head First. Паттерны проектирования. – СПб. : Питер, 2022. – 672 с.
13. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы. – 5-е изд. – СПб. : Питер, 2016. – 992 с.
14. Блинов И.Н., Романченко В.С. Java. Сетевое программирование. – СПб. : БХВ-Петербург, 2019. – 416 с.

15. Эккель Б. Философия Java. – 4-е изд. – СПб. : Питер, 2019. – 1168 с.
16. Головач А.В. Java. Многопоточное, функциональное и реактивное программирование. – М. : ДМК-Пресс, 2020. – 528 с.
17. Гриффитс Д., Гриффитс Д. Head First. Программирование для Android. – СПб. : Питер, 2022. – 704 с.
18. Симс Н., Лайт Д. SQLite. Карманный справочник. – М. : ДМК-Пресс, 2018. – 368 с.
19. Бушуев М.А. Паттерны проектирования на Java. – М. : ДМК-Пресс, 2020. – 468 с.
20. Куроуз Дж., Росс К. Компьютерные сети. Многоуровневая архитектура. – 8-е изд. – СПб. : Питер, 2021. – 912 с.
21. Хорстманн К.С. Java. Библиотека профессионала. Том 2. Расширенные средства программирования. – 11-е изд. – М. : Вильямс, 2019. – 976 с.
22. Самарский А.А. Проектирование цифровых платформ. – М. : Лаборатория знаний, 2021. – 288 с.

Приложение А
(справочное)
Глоссарий

В данной работе применяются следующие сокращения и обозначения.

БД – база данных

ЛКМ – левая кнопка мыши

ОС – операционная система

ПК – персональный компьютер

ПКМ – правая кнопка мыши

ПО – программное обеспечение

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

DAO – (от англ. Data Access Object) – объект доступа к данным

Gradle – система автоматизации сборки (проектов на Java / Android)

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

JDBC – (от англ. Java Database Connectivity) – подключение к базам данных из Java

JSON – (от англ. JavaScript Object Notation) – текстовый формат обмена данными

MVVM – (от англ. Model–View–ViewModel) – архитектурный паттерн

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

SQLite – встраиваемая реляционная база данных

TCP/IP – (от англ. Transmission Control Protocol / Internet Protocol) – стек протоколов передачи данных

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

Приложение Б
(обязательное)
Листинг программы

Полный листинг программы размещён в публичном репозитории GitHub по адресу: URL: <https://github.com/ВашАккаунт/НазваниеРепозитория> (дата обращения: 05.06.2026). Ниже приведены фрагменты наиболее значимых классов.

Листинг Б.1 – Файл DorPomoshServer.java

```
package org.dorpomosh;

import org.dorpomosh.database.DatabaseHelper;
import org.dorpomosh.network.ClientHandler;

import java.io.File;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class DorPomoshServer {
    private static final int PORT = 8080;
    private static final int MAX_THREADS = 100;
    private static volatile boolean running = true;

    public static void main(String[] args) {
        // Фикс кодировки для консоли (Windows)
        try {
            System.setOut(new java.io.PrintStream(System.out,
true, "UTF-8"));
            System.setErr(new java.io.PrintStream(System.err,
true, "UTF-8"));
        } catch (java.io.UnsupportedEncodingException e) {
            System.err.println("Ошибка настройки кодировки: " +
e.getMessage());
        }
        System.out.println("Попытка сделать бэкап БД:");
        //backupDatabase();
        System.out.println("=== ДорПомощь Сервер ===");
        System.out.println("Запуск сервера на порту " + PORT +
"...");
        // Создаём таблицы в БД при старте
        DatabaseHelper.createTables();
        // Пул потоков для обработки клиентов
        ExecutorService threadPool =
Executors.newFixedThreadPool(MAX_THREADS);

        // Добавляем хук для корректного завершения
        Runtime.getRuntime().addShutdownHook(new Thread(() -> {
            System.out.println("\nПолучен сигнал завершения.
Останавливаем сервер...");
```

```

        running = false;
        threadPool.shutdown();
        System.out.println("Сервер остановлен.");
    ));

    try (ServerSocket serverSocket = new ServerSocket(PORT))
    {
        System.out.println("Сервер запущен и ожидает
подключений на порту " + PORT);
        System.out.println("Нажмите Ctrl+C для
остановки.\n");

        while (running) {
            try {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Новое подключение: " +
clientSocket.getInetAddress());

                // Обрабатываем клиента в отдельном потоке
                threadPool.execute(new
ClientHandler(clientSocket));
            } catch (IOException e) {
                if (running) {
                    System.err.println("Ошибка при принятии
подключения: " + e.getMessage());
                }
            }
        }

        } catch (IOException e) {
            System.err.println("Не удалось запустить сервер на
порту " + PORT + ": " + e.getMessage());
            System.exit(1);
        }
    }

    public static void stopServer() {
        running = false;
    }

    private static void backupDatabase() {
        String dbPath = "dorpomosh.db";
        File dbFile = new File(dbPath);

        if (!dbFile.exists()) {
            System.out.println("[Бэкап] Файл БД не найден,
пропускаем");
            return;
        }

        // Создаём папку для бэкапов
        File backupDir = new File("backups_bd");
        if (!backupDir.exists()) {

```

```

        backupDir.mkdirs();
    }

    // Формируем имя файла: backup_bd_2025-01-17_15-30-45.db
    String timestamp = new java.text.SimpleDateFormat("yyyy-MM-dd_HH-mm-ss")
        .format(new java.util.Date());
    File backupFile = new File(backupDir, "backup_bd_" +
timestamp + ".db");

    // Копируем файл
    try {
        java.nio.file.Files.copy(dbFile.toPath(),
backupFile.toPath(),

java.nio.file.StandardCopyOption.REPLACE_EXISTING);
        System.out.println("[Бэкап] Создан: " +
backupFile.getAbsolutePath());
    } catch (java.io.IOException e) {
        System.err.println("[Бэкап] Ошибка: " +
e.getMessage());
    }
}
}

```

Листинг Б.2 – Файл ProtocolParser.java

```

package org.dorpomosh.protocol;

import org.dorpomosh.dao.*;
import org.dorpomosh.models.*;
import org.dorpomosh.utils.GsonProvider;

import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.List;

public class ProtocolParser {

    private static final int CHUNK_SIZE = 8192; // 8KB

    public static String handle(String request) {
        if (request == null || request.trim().isEmpty()) {
            return "ERROR|Пустой запрос";
        }

        String[] parts = request.split("\\|");
        if (parts.length == 0) {
            return "ERROR|Неверный формат запроса";
        }
    }
}

```

```

String command = parts[0];

switch (command) {
    case "PING": return "PONG";
    case "REG": return handleRegister(parts);
    case "AUTH": return handleAuth(parts);
    case "GET_USER": return handleGetUser(parts);
    case "UPDATE_USER": return handleUpdateUser(parts);
    case "GET_USERS_BY_ROLE": return
handleGetUsersByRole(parts);
    case "GET_MODELS": return handleGetModels();
    case "GET_MODEL_BY_ID": return
handleGetModelById(parts);
    case "ADD_MODEL": return handleAddModel(parts);
    case "GET_FILE_CHUNK": return
handleGetFileChunk(parts);
    case "ADD_CAR": return handleAddCar(parts);
    case "GET_CARS": return handleGetCars(parts);
    case "GET_CAR": return handleGetCar(parts);
    case "UPDATE_CAR": return handleUpdateCar(parts);
    case "DELETE_CAR": return handleDeleteCar(parts);
    case "ADD_MAINTENANCE": return
handleAddMaintenance(parts);
    case "GET_MAINTENANCE": return
handleGetMaintenance(parts);
    case "GET_MAINTENANCE_BY_ID": return
handleGetMaintenanceById(parts);
    case "UPDATE_MAINTENANCE": return
handleUpdateMaintenance(parts);
    case "DELETE_MAINTENANCE": return
handleDeleteMaintenance(parts);
    case "ADD_CAR_DOCUMENT": return
handleAddCarDocument(parts);
    case "GET_CAR_DOCUMENTS": return
handleGetCarDocuments(parts);
    case "GET_CAR_DOCUMENT_BY_ID": return
handleGetCarDocumentById(parts);
    case "DELETE_CAR_DOCUMENT": return
handleDeleteCarDocument(parts);
    case "ADD_ISSUE": return handleAddIssue(parts);
    case "GET_ISSUES": return handleGetIssues(parts);
    case "GET_ISSUE_BY_ID": return
handleGetIssueById(parts);
    case "UPDATE_ISSUE_TITLE": return
handleUpdateIssueTitle(parts);
    case "DELETE_ISSUE": return
handleDeleteIssue(parts);
    case "ADD_CAR_REQUEST": return
handleAddCarRequest(parts);
    case "GET_CAR_REQUESTS": return
handleGetCarRequests(parts);
    case "GET_CAR_REQUESTS_BY_USER": return
handleGetCarRequestsByUser(parts);

```

```

        case "UPDATE_CAR_REQUEST_STATUS": return
handleUpdateCarRequestStatus(parts);
        default: return "ERROR|Неизвестная команда: " +
command;
    }
}

private static String handleRegister(String[] parts) {
    if (parts.length < 3) return "ERROR|Не хватает
параметров";
    String login = parts[1];
    String password = parts[2];
    String fullName = (parts.length > 3 &&
!parts[3].isEmpty()) ? parts[3] : login;
    String phone = (parts.length > 4 && !parts[4].isEmpty())
? parts[4] : null;
    String email = (parts.length > 5 && !parts[5].isEmpty())
? parts[5] : null;
    User user = new User();
    user.setLogin(login);
    user.setPassword(password);
    user.setFullName(fullName);
    user.setPhone(phone);
    user.setEmail(email);
    user.setRole("user");
    int userId = UserDao.registerUser(user);
    return userId != -1 ? "OK|" + userId :
"ERROR|Пользователь с таким логином уже существует";
}

private static String handleAuth(String[] parts) {
    if (parts.length < 3) return "ERROR|Не хватает
параметров";
    int userId = UserDao.checkLogin(parts[1], parts[2]);
    if (userId != -1) {
        User user = UserDao.getUserById(userId);
        String role = user.getRole();
        String fullName = user.getFullName() != null ?
user.getFullName() : parts[1];
        return "OK|" + userId + "|" + role + "|" + fullName;
    }
    return "ERROR|Неверный логин или пароль";
}

private static String handleGetUser(String[] parts) {
    if (parts.length < 2) return "ERROR|Не указан ID";
    try {
        User user =
UserDao.getUserById(Integer.parseInt(parts[1]));
        if (user != null) {
            String json =
GsonProvider.getInstance().toJson(user).replaceAll("\\s+", " ");
            return "DATA|" + json;
        }
    }
}

```

```

        }
        return "ERROR|Пользователь не найден";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleUpdateUser(String[] parts) {
    if (parts.length < 5) return "ERROR|Не хватает
параметров";
    try {
        int userId = Integer.parseInt(parts[1]);
        User user = UserDao.getUserById(userId);
        if (user == null) return "ERROR|Пользователь не
найден";

        user.setFullName(parts[2]);
        user.setPhone(parts[3]);
        user.setEmail(parts[4]);
        return UserDao.updateUser(user) ? "OK" : "ERROR|Не
удалось обновить";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleGetUsersByRole(String[] parts) {
    if (parts.length < 2) return "ERROR|Не указана роль";
    String json =
GsonProvider.getInstance().toJson(UserDao.getUsersByRole(parts[1
])).replaceAll("\\s+", " ");
    return "DATA|" + json;
}

private static String handleGetModels() {
    String json =
GsonProvider.getInstance().toJson(ModelDao.getAllModels()).repla
ceAll("\\s+", " ");
    return "DATA|" + json;
}

private static String handleGetModelById(String[] parts) {
    if (parts.length < 2) return "ERROR|Не указан ID";
    try {
        Model model =
ModelDao.getModelById(Integer.parseInt(parts[1]));
        if (model != null) {
            String json =
GsonProvider.getInstance().toJson(model).replaceAll("\\s+", "
");
            return "DATA|" + json;
        }
        return "ERROR|Модель не найдена";
    } catch (NumberFormatException e) {

```

```

        return "ERROR|Неверный формат ID";
    }
}

private static String handleAddModel(String[] parts) {
    if (parts.length < 11) return "ERROR|Не хватает
параметров";
    try {
        Model model = new Model();
        model.setBrand(parts[1]);
        model.setModel(parts[2]);
        model.setGeneration(parts[3]);
        model.setBodyType(parts[4]);
        model.setDriveType(parts[5]);
        model.setEngineType(parts[6]);
        model.setEngineVolume(Float.parseFloat(parts[7]));
        model.setEnginePower(Float.parseFloat(parts[8]));
        model.setTransmission(parts[9]);
        model.setDocFilePath(parts[10]);
        int id = ModelDao.addModel(model);
        return id != -1 ? "OK|" + id : "ERROR|Не удалось
добавить модель";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат числовых параметров";
    }
}

private static String handleGetFileChunk(String[] parts) {
    if (parts.length < 3) return "ERROR|Не указаны
параметры";
    String filePath = parts[1];
    int chunkIndex = Integer.parseInt(parts[2]);
    File file = new File(".") + filePath;
    if (!file.exists()) return "ERROR|Файл не найден: " +
filePath;
    try {
        byte[] fileBytes =
java.nio.file.Files.readAllBytes(file.toPath());
        int start = chunkIndex * CHUNK_SIZE;
        if (start >= fileBytes.length) return "DATA|END";
        int end = Math.min(start + CHUNK_SIZE,
fileBytes.length);
        byte[] chunk = Arrays.copyOfRange(fileBytes, start,
end);

        String base64 =
java.util.Base64.getEncoder().encodeToString(chunk);
        base64 = base64.replaceAll("\\s+", "");
        return "DATA|" + base64 + "|" + chunk.length;
    } catch (IOException e) {
        return "ERROR|Ошибка чтения файла: " +
e.getMessage();
    }
}
}

```

```

        private static String handleAddCar(String[] parts) {
            if (parts.length < 10) return "ERROR|Не хватает
параметров";
            try {
                Car car = new Car();
                car.setUserId(Integer.parseInt(parts[1]));
                car.setModelId(Integer.parseInt(parts[2]));
                car.setYear(Integer.parseInt(parts[3]));
                car.setLicensePlate(parts[4]);
                car.setVin(parts[5]);
                car.setMileage(Integer.parseInt(parts[6]));
                car.setStsNumber(parts[7]);
                car.setInsuranceNumber(parts[8]);
                car.setInsuranceExpiry(parts[9]);
                int id = CarDao.addCar(car);
                return id != -1 ? "OK|" + id : "ERROR|Не удалось
добавить автомобиль";
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат числовых параметров";
            }
        }

        private static String handleGetCars(String[] parts) {
            if (parts.length < 2) return "ERROR|Не указан ID
пользователя";
            try {
                List<Car> cars =
CarDao.getCarsByUserId(Integer.parseInt(parts[1]));
                String json =
GsonProvider.getInstance().toJson(cars).replaceAll("\\s+", " ");
                return "DATA|" + json;
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат ID";
            }
        }

        private static String handleGetCar(String[] parts) {
            if (parts.length < 2) return "ERROR|Не указан ID";
            try {
                Car car =
CarDao.getCarById(Integer.parseInt(parts[1]));
                if (car != null) {
                    String json =
GsonProvider.getInstance().toJson(car).replaceAll("\\s+", " ");
                    return "DATA|" + json;
                }
                return "ERROR|Автомобиль не найден";
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат ID";
            }
        }
    }

```

```

        private static String handleUpdateCar(String[] parts) {
            if (parts.length < 9) return "ERROR|Не хватает
параметров";
            try {
                int carId = Integer.parseInt(parts[1]);
                Car car = CarDao.getCarById(carId);
                if (car == null) return "ERROR|Автомобиль не
найден";

                car.setYear(Integer.parseInt(parts[2]));
                car.setLicensePlate(parts[3]);
                car.setVin(parts[4]);
                car.setMileage(Integer.parseInt(parts[5]));
                car.setStsNumber(parts[6]);
                car.setInsuranceNumber(parts[7]);
                car.setInsuranceExpiry(parts[8]);
                return CarDao.updateCar(car) ? "OK" : "ERROR|Не
удалось обновить";
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат числовых параметров";
            }
        }

        private static String handleDeleteCar(String[] parts) {
            if (parts.length < 2) return "ERROR|Не указан ID";
            try {
                return CarDao.deleteCar(Integer.parseInt(parts[1]))
? "OK" : "ERROR|Автомобиль не найден";
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат ID";
            }
        }

        private static String handleAddMaintenance(String[] parts) {
            if (parts.length < 8) return "ERROR|Не хватает
параметров";
            try {
                MaintenanceRecord record = new MaintenanceRecord();
                record.setCarId(Integer.parseInt(parts[1]));
                record.setServiceType(parts[2]);
                record.setDescription(parts[3]);
                record.setDate(parts[4]);
                record.setMileage(Integer.parseInt(parts[5]));
                record.setCost(Float.parseFloat(parts[6]));
                record.setShopName(parts[7]);
                int id =
MaintenanceRecordDao.addMaintenanceRecord(record);
                return id != -1 ? "OK|" + id : "ERROR|Не удалось
добавить запись";
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат числовых параметров";
            }
        }

```

```

        private static String handleGetMaintenance(String[] parts) {
            if (parts.length < 2) return "ERROR|Не указан ID
автомобиля";
            try {
                List<MaintenanceRecord> records =
MaintenanceRecordDao.getMaintenanceRecordsByCarId(Integer.parseInt
nt(parts[1]));
                String json =
GsonProvider.getInstance().toJson(records).replaceAll("\\s+", "
");
                return "DATA|" + json;
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат ID";
            }
        }

        private static String handleGetMaintenanceById(String[]
parts) {
            if (parts.length < 2) return "ERROR|Не указан ID";
            try {
                MaintenanceRecord record =
MaintenanceRecordDao.getMaintenanceRecordById(Integer.parseInt(p
arts[1]));
                if (record != null) {
                    String json =
GsonProvider.getInstance().toJson(record).replaceAll("\\s+", "
");
                    return "DATA|" + json;
                }
                return "ERROR|Запись не найдена";
            } catch (NumberFormatException e) {
                return "ERROR|Неверный формат ID";
            }
        }

        private static String handleUpdateMaintenance(String[]
parts) {
            if (parts.length < 8) return "ERROR|Не хватает
параметров";
            try {
                int id = Integer.parseInt(parts[1]);
                MaintenanceRecord record =
MaintenanceRecordDao.getMaintenanceRecordById(id);
                if (record == null) return "ERROR|Запись не
найдена";
                record.setServiceType(parts[2]);
                record.setDescription(parts[3]);
                record.setDate(parts[4]);
                record.setMileage(Integer.parseInt(parts[5]));
                record.setCost(Float.parseFloat(parts[6]));
                record.setShopName(parts[7]);
                return
MaintenanceRecordDao.updateMaintenanceRecord(record) ? "OK" :

```

```

"ERROR|Не удалось обновить запись";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат числовых параметров";
    }
}

private static String handleDeleteMaintenance(String[]
parts) {
    if (parts.length < 2) return "ERROR|Не указан ID";
    try {
        return
MaintenanceRecordDao.deleteMaintenanceRecord(Integer.parseInt(pa
rts[1])) ? "OK" : "ERROR|Запись не найдена";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleAddCarDocument(String[] parts) {
    if (parts.length < 5) return "ERROR|Не хватает
параметров";
    try {
        CarDocument doc = new CarDocument();
        doc.setCarId(Integer.parseInt(parts[1]));
        doc.setTitle(parts[2]);
        doc.setFilePath(parts[3]);
        doc.setDocumentType(parts[4]);
        int id = CarDocumentDao.addCarDocument(doc);
        return id != -1 ? "OK|" + id : "ERROR|Не удалось
добавить документ";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleGetCarDocuments(String[] parts)
{
    if (parts.length < 2) return "ERROR|Не указан ID
автомобиля";
    try {
        List<CarDocument> docs =
CarDocumentDao.getCarDocumentsByCarId(Integer.parseInt(parts[1])
);
        String json =
GsonProvider.getInstance().toJson(docs).replaceAll("\\s+", " ");
        return "DATA|" + json;
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleGetCarDocumentById(String[]
parts) {

```

```

        if (parts.length < 2) return "ERROR|Не указан ID";
        try {
            CarDocument doc =
CarDocumentDao.getCarDocumentById(Integer.parseInt(parts[1]));
            if (doc != null) {
                String json =
GsonProvider.getInstance().toJson(doc).replaceAll("\\s+", " ");
                return "DATA|" + json;
            }
            return "ERROR|Документ не найден";
        } catch (NumberFormatException e) {
            return "ERROR|Неверный формат ID";
        }
    }

    private static String handleDeleteCarDocument(String[]
parts) {
        if (parts.length < 2) return "ERROR|Не указан ID";
        try {
            return
CarDocumentDao.deleteCarDocument(Integer.parseInt(parts[1])) ?
"OK" : "ERROR|Документ не найден";
        } catch (NumberFormatException e) {
            return "ERROR|Неверный формат ID";
        }
    }

    private static String handleAddIssue(String[] parts) {
        if (parts.length < 6) return "ERROR|Не хватает
параметров";
        try {
            CommonIssue issue = new CommonIssue();
            issue.setModelId(Integer.parseInt(parts[1]));
            issue.setIssueTitle(parts[2]);
            issue.setDescription(parts[3]);
            issue.setSymptoms(parts[4]);
            issue.setDifficulty(parts[5]);
            if (parts.length > 6)
issue.setUserId(Integer.parseInt(parts[6]));
            int id = CommonIssueDao.addCommonIssue(issue);
            return id != -1 ? "OK|" + id : "ERROR|Не удалось
добавить проблему";
        } catch (NumberFormatException e) {
            return "ERROR|Неверный формат параметров";
        }
    }

    private static String handleGetIssues(String[] parts) {
        if (parts.length < 2) return "ERROR|Не указан ID
модели";
        try {
            List<CommonIssue> issues =
CommonIssueDao.getCommonIssuesByModelId(Integer.parseInt(parts[1]

```

```

    ));
        String json =
GsonProvider.getInstance().toJson(issues).replaceAll("\\s+", "
");
        return "DATA|" + json;
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleGetIssueById(String[] parts) {
    if (parts.length < 2) return "ERROR|Не указан ID";
    try {
        CommonIssue issue =
CommonIssueDao.getCommonIssueById(Integer.parseInt(parts[1]));
        if (issue != null) {
            String json =
GsonProvider.getInstance().toJson(issue).replaceAll("\\s+", "
");
            return "DATA|" + json;
        }
        return "ERROR|Проблема не найдена";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleUpdateIssueTitle(String[] parts)
{
    if (parts.length < 3) return "ERROR|Не хватает
параметров";
    try {
        return
CommonIssueDao.updateCommonIssueTitle(Integer.parseInt(parts[1])
, parts[2]) ? "OK" : "ERROR|Проблема не найдена";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleDeleteIssue(String[] parts) {
    if (parts.length < 2) return "ERROR|Не указан ID";
    try {
        return
CommonIssueDao.deleteCommonIssue(Integer.parseInt(parts[1])) ?
"OK" : "ERROR|Проблема не найдена";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

private static String handleAddCarRequest(String[] parts) {
    if (parts.length < 3) return "ERROR|Не хватает

```

```

    параметров";
    try {
        CarRequest request = new CarRequest();
        request.setUserId(Integer.parseInt(parts[1]));
        request.setRequestedModel(parts[2]);
        if (parts.length > 3)
            request.setAdditionalInfo(parts[3]);
        request.setStatus("pending");
        int id = CarRequestDao.addCarRequest(request);
        return id != -1 ? "OK|" + id : "ERROR|Не удалось
добавить заявку";
    } catch (NumberFormatException e) {
        return "ERROR|Неверный формат ID";
    }
}

    private static String handleGetCarRequests(String[] parts) {
        String json =
GsonProvider.getInstance().toJson(CarRequestDao.getAllCarRequest
s()).replaceAll("\\s+", " ");
        return "DATA|" + json;
    }

    private static String handleGetCarRequestsByUser(String[]
parts) {
        if (parts.length < 2) return "ERROR|Не указан ID
пользователя";
        try {
            List<CarRequest> requests =
CarRequestDao.getCarRequestsByUserId(Integer.parseInt(parts[1]))
;
            String json =
GsonProvider.getInstance().toJson(requests).replaceAll("\\s+", "
");
            return "DATA|" + json;
        } catch (NumberFormatException e) {
            return "ERROR|Неверный формат ID";
        }
    }

    private static String handleUpdateCarRequestStatus(String[]
parts) {
        if (parts.length < 3) return "ERROR|Не хватает
параметров";
        try {
            return
CarRequestDao.updateCarRequestStatus(Integer.parseInt(parts[1]),
parts[2]) ? "OK" : "ERROR|Заявка не найдена";
        } catch (NumberFormatException e) {
            return "ERROR|Неверный формат ID";
        }
    }
}

```

Листинг Б.3 – Файл MainActivity.java

```
package com.example.roadhelpclientandroid.view;

import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;

import com.example.roadhelpclientandroid.network.NetworkService;
import
com.example.roadhelpclientandroid.view.fragments.CarsFragment;
import
com.example.roadhelpclientandroid.view.fragments.DocsFragment;
import
com.example.roadhelpclientandroid.view.fragments.ProfileFragment
;
import
com.google.android.material.bottomnavigation.BottomNavigationView;
import com.example.roadhelpclientandroid.R;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // Инициализируем NetworkService (один раз за всё время
работы приложения)
        NetworkService.init(this);

        // Находим панель навигации
        BottomNavigationView bottomNav =
findViewById(R.id.bottomNav);

        // При первом запуске показываем фрагмент документации
        if (savedInstanceState == null) {
            getSupportFragmentManager().beginTransaction()
                .replace(R.id.container, new DocsFragment())
                .commit();
        }

        // Обработчик нажатий на пункты меню
        bottomNav.setOnItemSelectedListener(item -> {
            Fragment selectedFragment = null;
            int itemId = item.getItemId();

            if (itemId == R.id.nav_docs) {
                selectedFragment = new DocsFragment();
            } else if (itemId == R.id.nav_cars) {
```

```

        selectedFragment = new CarsFragment();
    } else if (itemId == R.id.nav_profile) {
        selectedFragment = new ProfileFragment();
    }

    if (selectedFragment != null) {
        getSupportFragmentManager().beginTransaction()
            .replace(R.id.container,
selectedFragment)
            .commit();
    }
    return true;
});
}
}

```

Листинг Б.4 – Файл SessionManager.java

```

package com.example.roadhelpclientandroid.storage;

import android.content.Context;
import android.content.SharedPreferences;

public class SessionManager {
    private static final String PREF_NAME = "doropomosh_prefs";
    private static final String KEY_USER_ID = "user_id";
    private static final String KEY_LOGIN = "login";
    private static final String KEY_ROLE = "role";
    private static final String KEY_FULL_NAME = "full_name";

    private static SessionManager instance;
    private final SharedPreferences prefs;

    private SessionManager(Context context) {
        prefs =
context.getApplicationContext().getSharedPreferences(PREF_NAME,
Context.MODE_PRIVATE);
    }

    public static synchronized SessionManager
getInstance(Context context) {
        if (instance == null) {
            instance = new SessionManager(context);
        }
        return instance;
    }

    // Сохранить данные после входа
    public void saveUser(int userId, String login, String role,
String fullName) {
        prefs.edit()
            .putInt(KEY_USER_ID, userId)

```

```

        .putString(KEY_LOGIN, login)
        .putString(KEY_ROLE, role)
        .putString(KEY_FULL_NAME, fullName)
        .apply();
    }

    public int getUserId() {
        return prefs.getInt(KEY_USER_ID, -1);
    }

    public String getLogin() {
        return prefs.getString(KEY_LOGIN, "");
    }

    public String getRole() {
        return prefs.getString(KEY_ROLE, "user");
    }

    public String getFullName() {
        return prefs.getString(KEY_FULL_NAME, "");
    }

    public boolean isLoggedIn() {
        return getUserId() != -1;
    }

    // Выход (очищаем все данные)
    public void logout() {
        prefs.edit().clear().apply();
    }
}

```

Листинг Б.5 – Файл Car.java

```

package com.example.roadhelpclientandroid.models;

public class Car {
    private int id; // айди машины
    private int userId; // айди владельца машины
    private int modelId; // айди модели, которой
    private int year;
    private String licensePlate;
    private String vin;
    private int mileage;
    private String stsNumber;
    private String insuranceNumber;
    private String insuranceExpiry;

    public Car() {
    }

    public Car(int id, int userId, int modelId, int year,
               String licensePlate, String vin,

```

```

        int mileage, String stsNumber,
        String insuranceNumber, String insuranceExpiry) {
    this.id = id;
    this.userId = userId;
    this.modelId = modelId;
    this.year = year;
    this.licensePlate = licensePlate;
    this.vin = vin;
    this.mileage = mileage;
    this.stsNumber = stsNumber;
    this.insuranceNumber = insuranceNumber;
    this.insuranceExpiry = insuranceExpiry;
}
public Car(int userId, int modelId, int year,
        String licensePlate, String vin,
        int mileage, String stsNumber,
        String insuranceNumber, String insuranceExpiry) {
    this.userId = userId;
    this.modelId = modelId;
    this.year = year;
    this.licensePlate = licensePlate;
    this.vin = vin;
    this.mileage = mileage;
    this.stsNumber = stsNumber;
    this.insuranceNumber = insuranceNumber;
    this.insuranceExpiry = insuranceExpiry;
}

public int getId() {
    return this.id;
}
public void setId(int id) {
    this.id=id;
}
public int getUserId() {
    return this.userId;
}
public void setUserId(int userId) {
    this.userId = userId;
}
public int getModelId() {
    return this.modelId;
}
public void setModelId(int modelId) {
    this.modelId = modelId;
}
public int getYear() {
    return this.year;
}
public void setYear(int year) {
    this.year = year;
}
public String getLicensePlate() {

```

```

        return this.licensePlate;
    }
    public void setLicensePlate(String licensePlate) {
        this.licensePlate = licensePlate;
    }
    public String getVin() {
        return this.vin;
    }
    public void setVin(String vin) {
        this.vin = vin;
    }
    public int getMileage() {
        return this.mileage;
    }
    public void setMileage(int mileage) {
        this.mileage = mileage;
    }
    public String getStsNumber() {
        return this.stsNumber;
    }
    public void setStsNumber(String stsNumber) {
        this.stsNumber = stsNumber;
    }
    public String getInsuranceNumber() {
        return insuranceNumber;
    }
    public void setInsuranceNumber(String insuranceNumber) {
        this.insuranceNumber = insuranceNumber;
    }
    public String getInsuranceExpiry() {
        return this.insuranceExpiry;
    }
    public void setInsuranceExpiry(String insuranceExpiry) {
        this.insuranceExpiry = insuranceExpiry;
    }
    @Override
    public String toString() {
        return "Car{" +
            "id=" + id +
            ", userId=" + userId +
            ", modelId=" + modelId +
            ", year=" + year +
            ", licensePlate='" + licensePlate + '\'' +
            ", vin='" + vin + '\'' +
            ", mileage=" + mileage +
            ", stsNumber='" + stsNumber + '\'' +
            ", insuranceNumber='" + insuranceNumber + '\'' +
            ", insuranceExpiry='" + insuranceExpiry + '\'' +
            '}';
    }
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;

```

```

        if (o == null || getClass() != o.getClass()) return
false;
        Car car = (Car) o;
        return id == car.id;
    }
    @Override
    public int hashCode() {
        return Integer.hashCode(id);
    }
}

```

Листинг Б.6 – Файл CarRepository.java

```

package com.example.roadhelpclientandroid.repository;

import com.example.roadhelpclientandroid.models.Car;
import com.example.roadhelpclientandroid.network.NetworkService;
import com.example.roadhelpclientandroid.utils.GsonProvider;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;

public class CarRepository {

    private final Gson gson = GsonProvider.getInstance();

    public int addCar(Car car) {
        String command = "ADD_CAR|" + car.getUserId() + "|" +
car.getModelId() + "|"
            + car.getYear() + "|" + car.getLicensePlate() +
|" + car.getVin() + "|"
            + car.getMileage() + "|" + car.getStsNumber() +
|" + car.getInsuranceNumber() + "|"
            + car.getInsuranceExpiry();

        String response =
NetworkService.getInstance().sendCommand(command);
        if (response != null && response.startsWith("OK|")) {
            String[] parts = response.split("\\|");
            return Integer.parseInt(parts[1]);
        }
        return -1;
    }

    public List<Car> getCarsByUserId(int userId) {
        try {
            String response =
NetworkService.getInstance().sendCommand("GET_CARS|" + userId);

```

```

        if (response != null &&
response.startsWith("DATA|")) {
            String json = response.substring(5).trim();
            Type listType = new
TypeToken<List<Car>>(){}.getType();
            List<Car> cars = gson.fromJson(json, listType);
            return cars != null ? cars : new ArrayList<>();
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return new ArrayList<>();
}
}

```

Листинг Б.7 – Файл CarsAdapter.java

```

package com.example.roadhelpclientandroid.adapter;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.example.roadhelpclientandroid.R;
import com.example.roadhelpclientandroid.models.Car;

import java.util.ArrayList;
import java.util.List;

public class CarsAdapter extends
RecyclerView.Adapter<CarsAdapter.CarViewHolder> {

    private List<Car> cars = new ArrayList<>();
    private OnCarClickListener listener;

    public interface OnCarClickListener {
        void onCarClick(int carId);
    }

    public void setOnCarClickListener(OnCarClickListener
listener) {
        this.listener = listener;
    }

    public void setCars(List<Car> cars) {
        this.cars = cars != null ? cars : new ArrayList<>();
        notifyDataSetChanged();
    }
}

```

```

        @NonNull
        @Override
        public CarViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
            View view = LayoutInflater.from(parent.getContext())
                .inflate(R.layout.item_car, parent, false);
            return new CarViewHolder(view);
        }

        @Override
        public void onBindViewHolder(@NonNull CarViewHolder holder,
int position) {
            Car car = cars.get(position);
            holder.bind(car, listener);
        }

        @Override
        public int getItemCount() {
            return cars.size();
        }

        static class CarViewHolder extends RecyclerView.ViewHolder {
            private final TextView tvCarInfo;

            public CarViewHolder(@NonNull View itemView) {
                super(itemView);
                tvCarInfo = itemView.findViewById(R.id.tvCarInfo);
            }

            public void bind(Car car, OnCarClickListener listener) {
                String info = car.getLicensePlate() + " | " +
car.getYear() + " г. | Пробег: " + car.getMileage() + " км";
                tvCarInfo.setText(info);
                itemView.setOnClickListener(v -> {
                    if (listener != null) {
                        listener.onCarClick(car.getId());
                    }
                });
            }
        }
    }
}

```

Листинг Б.8 – Файл NetworkService.java

```

package com.example.roadhelpclientandroid.network;

import android.content.Context;
import com.example.roadhelpclientandroid.R;

import java.io.BufferedReader;
import java.io.InputStreamReader;

```

```

import java.io.PrintWriter;
import java.net.Socket;

public class NetworkService {
    private static NetworkService instance;
    private static String SERVER_IP;
    private static int SERVER_PORT;
    private static boolean isInitialized = false;

    private NetworkService() {}

    public static synchronized NetworkService getInstance() {
        if (instance == null) {
            instance = new NetworkService();
        }
        return instance;
    }

    /**
     * Вызывается один раз в MainActivity
     * @param context Контекст приложения
     */
    public static void init(Context context) {
        if (!isInitialized) {
            SERVER_IP = context.getString(R.string.server_ip);
            SERVER_PORT =
Integer.parseInt(context.getString(R.string.server_port));
            isInitialized = true;
            System.out.println("NetworkService initialized with
IP: " + SERVER_IP + ":" + SERVER_PORT);
        }
    }

    public String sendCommand(String command) {
        if (!isInitialized) {
            return "ERROR|NetworkService не инициализирован.
Вызови init()";
        }

        try (Socket socket = new Socket(SERVER_IP, SERVER_PORT);
            PrintWriter out = new
PrintWriter(socket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()))) {

            out.println(command);
            return in.readLine();

        } catch (Exception e) {
            e.printStackTrace();
            return "ERROR|Сетевая ошибка: " + e.getMessage();
        }
    }
}

```



```

        new
Handler(Looper.getMainLooper()).post(() ->
callback.onError("Ошибка: " + response));
        return;
    }
    if (response.startsWith("DATA|")) {
        String[] parts = response.split("\\|");
        if (parts.length < 2) {
            new
Handler(Looper.getMainLooper()).post(() ->
callback.onError("Неверный формат ответа"));
            return;
        }
        String base64 = parts[1];
        byte[] chunk = Base64.decode(base64,
Base64.DEFAULT);

        chunks.add(chunk);
        totalSize += chunk.length;
        chunkIndex++;
    }
}

File cacheDir =
PdfCacheManager.getPdfCacheDir(context);
if (!cacheDir.exists()) {
    cacheDir.mkdirs();
}

File outputFile = new File(cacheDir, fileName);
try (FileOutputStream fos = new
FileOutputStream(outputFile)) {
    for (byte[] chunk : chunks) {
        fos.write(chunk);
    }
}

System.out.println("PDF saved: " +
outputFile.getAbsolutePath() + ", size: " + totalSize);
    new Handler(Looper.getMainLooper()).post(() ->
callback.onSuccess(outputFile));

    } catch (Exception e) {
        System.err.println("PDF download exception: " +
e.getMessage());
        e.printStackTrace();
        new Handler(Looper.getMainLooper()).post(() ->
callback.onError("Ошибка: " + e.getMessage()));
    }
})).start();
}
}

```

Листинг Б.10 – Файл GsonProvider.java

```
package com.example.roadhelpclientandroid.utils;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;

public class GsonProvider {
    private static final Gson INSTANCE = new GsonBuilder()
        .setPrettyPrinting()           // красивое
        форматирование JSON
        .serializeNulls()              // включает null-поля
        в JSON (удобно для отладки)
        .create();

    private GsonProvider() {
        // приватный конструктор, чтобы никто не создал
        экземпляр
    }

    public static Gson getInstance() {
        return INSTANCE;
    }
}
```

Листинг Б.11 – Файл PdfCacheManager.java

```
package com.example.roadhelpclientandroid.utils;

import android.content.Context;
import java.io.File;

public class PdfCacheManager {
    private static final String PDF_DIR = "pdf_cache";

    public static File getPdfCacheDir(Context context) {
        return new File(context.getCacheDir(), PDF_DIR);
    }

    public static File getPdfFile(Context context, String
fileName) {
        return new File(getPdfCacheDir(context), fileName);
    }

    public static boolean isPdfCached(Context context, String
fileName) {
        return getPdfFile(context, fileName).exists();
    }

    public static void clearCache(Context context) {
        File dir = getPdfCacheDir(context);
        if (dir.exists()) {

```

```

        for (File file : dir.listFiles()) {
            file.delete();
        }
    }
}

```

Листинг Б.12 – Файл AddCarActivity.java

```

package com.example.roadhelpclientandroid.view;

import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.roadhelpclientandroid.R;
import com.example.roadhelpclientandroid.models.Car;
import com.example.roadhelpclientandroid.models.Model;
import com.example.roadhelpclientandroid.repository.CarRepository;
import com.example.roadhelpclientandroid.repository.ModelRepository;
import com.example.roadhelpclientandroid.storage.SessionManager;

import java.util.List;

public class AddCarActivity extends AppCompatActivity {

    private Spinner spinnerModel;
    private EditText etYear, etLicensePlate, etVin, etMileage,
    etStsNumber, etInsuranceNumber, etInsuranceExpiry;
    private Button btnSave;

    private List<Model> models;
    private int selectedModelId = -1;
    private SessionManager sessionManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add_car);

        sessionManager = SessionManager.getInstance(this);

        spinnerModel = findViewById(R.id.spinnerModel);
        etYear = findViewById(R.id.etYear);
    }
}

```

```

        etLicensePlate = findViewById(R.id.etLicensePlate);
        etVin = findViewById(R.id.etVin);
        etMileage = findViewById(R.id.etMileage);
        etStsNumber = findViewById(R.id.etStsNumber);
        etInsuranceNumber =
findViewById(R.id.etInsuranceNumber);
        etInsuranceExpiry =
findViewById(R.id.etInsuranceExpiry);
        btnSave = findViewById(R.id.btnSave);

        loadModels();

        btnSave.setOnClickListener(v -> saveCar());
    }

    private void loadModels() {
        new Thread(() -> {
            ModelRepository repository = new ModelRepository();
            models = repository.getAllModels();
            runOnUiThread(() -> {
                if (models != null && !models.isEmpty()) {
                    ArrayAdapter<String> adapter = new
ArrayAdapter<>(this,

android.R.layout.simple_spinner_item, getModelNames());

adapter.setDropDownViewResource(android.R.layout.simple_spinner_
dropdown_item);

                    spinnerModel.setAdapter(adapter);

                    spinnerModel.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
                        @Override
                        public void
onItemSelected(AdapterView<?> parent, View view, int position,
long id) {
                            selectedModelId =
models.get(position).getId();
                        }

                        @Override
                        public void
onNothingSelected(AdapterView<?> parent) {
                            selectedModelId = -1;
                        }
                    });
                } else {
                    Toast.makeText(this, "Не удалось загрузить
список моделей", Toast.LENGTH_SHORT).show();
                }
            });
        }).start();
    }
}

```

```

private String[] getModelNames() {
    String[] names = new String[models.size()];
    for (int i = 0; i < models.size(); i++) {
        names[i] = models.get(i).getDisplayName();
    }
    return names;
}

private void saveCar() {
    int userId = sessionManager.getUserId();
    if (userId == -1) {
        Toast.makeText(this, "Пользователь не авторизован",
Toast.LENGTH_SHORT).show();
        finish();
        return;
    }

    if (selectedModelId == -1) {
        Toast.makeText(this, "Выберите модель автомобиля",
Toast.LENGTH_SHORT).show();
        return;
    }

    String yearStr = etYear.getText().toString().trim();
    String licensePlate =
etLicensePlate.getText().toString().trim();
    String vin = etVin.getText().toString().trim();
    String mileageStr =
etMileage.getText().toString().trim();
    String stsNumber =
etStsNumber.getText().toString().trim();
    String insuranceNumber =
etInsuranceNumber.getText().toString().trim();
    String insuranceExpiry =
etInsuranceExpiry.getText().toString().trim();

    if (yearStr.isEmpty() || licensePlate.isEmpty() ||
vin.isEmpty()) {
        Toast.makeText(this, "Заполните обязательные поля
(год, госномер, VIN)", Toast.LENGTH_SHORT).show();
        return;
    }

    Car car = new Car();
    car.setUserId(userId);
    car.setModelId(selectedModelId);
    car.setYear(Integer.parseInt(yearStr));
    car.setLicensePlate(licensePlate);
    car.setVin(vin);
    car.setMileage(mileageStr.isEmpty() ? 0 :
Integer.parseInt(mileageStr));
    car.setStsNumber(stsNumber);

```

```

        car.setInsuranceNumber(insuranceNumber);
        car.setInsuranceExpiry(insuranceExpiry);

        new Thread(() -> {
            CarRepository repository = new CarRepository();
            int carId = repository.addCar(car);
            runOnUiThread(() -> {
                if (carId != -1) {
                    Toast.makeText(this, "Автомобиль успешно
добавлен", Toast.LENGTH_SHORT).show();
                    finish();
                } else {
                    Toast.makeText(this, "Ошибка при добавлении
автомобиля", Toast.LENGTH_SHORT).show();
                }
            });
        }).start();
    }
}

```

Листинг Б.13 – Файл CarDetailActivity.java

```

package com.example.roadhelpclientandroid.view;

import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.example.roadhelpclientandroid.R;
import com.example.roadhelpclientandroid.adapter.IssuesAdapter;
import
com.example.roadhelpclientandroid.adapter.MaintenanceAdapter;
import com.example.roadhelpclientandroid.models.Car;
import com.example.roadhelpclientandroid.models.CommonIssue;
import
com.example.roadhelpclientandroid.models.MaintenanceRecord;
import com.example.roadhelpclientandroid.network.NetworkService;
import com.example.roadhelpclientandroid.utils.GsonProvider;
import
com.google.android.material.floatingactionbutton.FloatingActionB
utton;
import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;

public class CarDetailActivity extends AppCompatActivity {

```

```

        private TextView tvCarTitle, tvYear, tvLicensePlate, tvVin,
tvMileage, tvStsNumber, tvInsuranceNumber, tvInsuranceExpiry;
        private RecyclerView rvMaintenance, rvIssues;
        private MaintenanceAdapter maintenanceAdapter;
        private IssuesAdapter issuesAdapter;
        private int carId;
        private int modelId;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_car_detail);

            carId = getIntent().getIntExtra("car_id", -1);
            if (carId == -1) {
                Toast.makeText(this, "Ошибка: ID автомобиля не
передан", Toast.LENGTH_SHORT).show();
                finish();
                return;
            }

            tvCarTitle = findViewById(R.id.tvCarTitle);
            tvYear = findViewById(R.id.tvYear);
            tvLicensePlate = findViewById(R.id.tvLicensePlate);
            tvVin = findViewById(R.id.tvVin);
            tvMileage = findViewById(R.id.tvMileage);
            tvStsNumber = findViewById(R.id.tvStsNumber);
            tvInsuranceNumber =
findViewById(R.id.tvInsuranceNumber);
            tvInsuranceExpiry =
findViewById(R.id.tvInsuranceExpiry);
            rvMaintenance = findViewById(R.id.rvMaintenance);
            rvIssues = findViewById(R.id.rvIssues);

            rvMaintenance.setLayoutManager(new
LinearLayoutManager(this));
            rvIssues.setLayoutManager(new
LinearLayoutManager(this));
            maintenanceAdapter = new MaintenanceAdapter();
            issuesAdapter = new IssuesAdapter();
            rvMaintenance.setAdapter(maintenanceAdapter);
            rvIssues.setAdapter(issuesAdapter);

            FloatingActionButton fabAddMaintenance =
findViewById(R.id.fabAddMaintenance);
            FloatingActionButton fabAddIssue =
findViewById(R.id.fabAddIssue);

            fabAddMaintenance.setOnClickListener(v -> {
                AddEditMaintenanceActivity.startForAdd(this, carId);
            });

            fabAddIssue.setOnClickListener(v -> {

```

```

        AddEditIssueActivity.startForAdd(this, modelId);
    });

    loadCarData();
}

private void loadCarData() {
    new Thread(() -> {
        String response =
NetworkService.getInstance().sendCommand("GET_CAR|" + carId);
        if (response != null &&
response.startsWith("DATA|")) {
            String json = response.substring(5).trim();
            try {
                Car car =
GsonProvider.getInstance().fromJson(json, Car.class);
                if (car != null) {
                    modelId = car.getModelId();
                    runOnUiThread(() ->
displayCarInfo(car));
                    loadMaintenanceRecords();
                    loadCommonIssues();
                } else {
                    runOnUiThread(() -> Toast.makeText(this,
"Ошибка загрузки данных", Toast.LENGTH_SHORT).show());
                }
            } catch (Exception e) {
                e.printStackTrace();
                runOnUiThread(() -> Toast.makeText(this,
"Ошибка парсинга", Toast.LENGTH_SHORT).show());
            }
        } else {
            runOnUiThread(() -> Toast.makeText(this,
"Ошибка: " + response, Toast.LENGTH_SHORT).show());
        }
    }).start();
}

private void displayCarInfo(Car car) {
    tvCarTitle.setText(car.getLicensePlate() != null ?
car.getLicensePlate() : "Автомобиль");
    tvYear.setText("Год: " + (car.getYear() > 0 ?
car.getYear() : "-"));
    tvLicensePlate.setText("Госномер: " +
(car.getLicensePlate() != null ? car.getLicensePlate() : "-"));
    tvVin.setText("VIN: " + (car.getVin() != null ?
car.getVin() : "-"));
    tvMileage.setText("Пробег: " + car.getMileage() + "
км");
    tvStsNumber.setText("СТС: " + (car.getStsNumber() !=
null ? car.getStsNumber() : "-"));
}

```

```

        tvInsuranceNumber.setText("Страховка: " +
(car.getInsuranceNumber() != null ? car.getInsuranceNumber() :
"-"));
        tvInsuranceExpiry.setText("Действует до: " +
(car.getInsuranceExpiry() != null ? car.getInsuranceExpiry() :
"-"));
    }

    private void loadMaintenanceRecords() {
        new Thread(() -> {
            String response =
NetworkService.getInstance().sendCommand("GET_MAINTENANCE|" +
carId);
            if (response != null &&
response.startsWith("DATA|")) {
                String json = response.substring(5).trim();
                if (!json.isEmpty() && !json.equals("[]")) {
                    try {
                        Type listType = new
TypeToken<List<MaintenanceRecord>>().getType();
                        List<MaintenanceRecord> records =
GsonProvider.getInstance().fromJson(json, listType);
                        runOnUiThread(() ->
maintenanceAdapter.setRecords(records != null ? records : new
ArrayList<>()));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }).start();
    }

    private void loadCommonIssues() {
        new Thread(() -> {
            String response =
NetworkService.getInstance().sendCommand("GET_ISSUES|" +
modelId);
            if (response != null &&
response.startsWith("DATA|")) {
                String json = response.substring(5).trim();
                if (!json.isEmpty() && !json.equals("[]")) {
                    try {
                        Type listType = new
TypeToken<List<CommonIssue>>().getType();
                        List<CommonIssue> issues =
GsonProvider.getInstance().fromJson(json, listType);
                        runOnUiThread(() ->
issuesAdapter.setIssues(issues != null ? issues : new
ArrayList<>()));
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        });
    }

```

```

        }
    }
    }).start();
}

@Override
protected void onResume() {
    super.onResume();
    loadMaintenanceRecords();
    loadCommonIssues();
}
}

```

Листинг Б.14 – Файл LoginActivity.java

```

package com.example.roadhelpclientandroid.view;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.roadhelpclientandroid.R;
import com.example.roadhelpclientandroid.network.NetworkService;
import com.example.roadhelpclientandroid.storage.SessionManager;

public class LoginActivity extends AppCompatActivity {

    private EditText etLogin, etPassword;
    private Button btnLogin;
    private TextView tvRegisterHint;
    private SessionManager sessionManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        sessionManager = SessionManager.getInstance(this);

        etLogin = findViewById(R.id.etLogin);
        etPassword = findViewById(R.id.etPassword);
        btnLogin = findViewById(R.id.btnLogin);
        tvRegisterHint = findViewById(R.id.tvRegisterHint);

        btnLogin.setOnClickListener(v -> {
            String login = etLogin.getText().toString().trim();
            String password =
etPassword.getText().toString().trim();

```

```

        if (login.isEmpty() || password.isEmpty()) {
            Toast.makeText(this, "Заполните все поля",
Toast.LENGTH_SHORT).show();
            return;
        }

        new Thread(() -> {
            String response =
NetworkService.getInstance().sendCommand("AUTH|" + login + "|" +
password);

            runOnUiThread(() -> {
                if (response != null &&
response.startsWith("OK|")) {
                    String[] parts = response.split("\\|");
                    int userId = Integer.parseInt(parts[1]);
                    String role = parts[2];
                    String fullName = parts.length > 3 ?
parts[3] : login;

                    sessionManager.saveUser(userId, login,
role, fullName);

                    Toast.makeText(LoginActivity.this, "Вход
выполнен", Toast.LENGTH_SHORT).show();

                    Intent intent = new
Intent(LoginActivity.this, MainActivity.class);

                    intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_NEW_TASK);
                    startActivity(intent);
                    finish();
                } else {
                    String error = response != null ?
response : "Неизвестная ошибка";
                    Toast.makeText(LoginActivity.this,
"Ошибка: " + error, Toast.LENGTH_LONG).show();
                }
            });
        }).start();
    });

    tvRegisterHint.setOnClickListener(v -> {
        Intent intent = new Intent(LoginActivity.this,
RegisterActivity.class);
        startActivity(intent);
    });
}
}

```

Листинг Б.15 – Файл PdfViewerActivity.java

```

package com.example.roadhelpclientandroid.view;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.FileProvider;
import java.io.File;

public class PdfViewerActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        String pdfPath = getIntent().getStringExtra("pdf_path");
        System.out.println("PdfViewerActivity: pdfPath = " +
pdfPath);
        if (pdfPath == null) {
            Toast.makeText(this, "Путь к PDF не указан",
Toast.LENGTH_SHORT).show();
            finish();
            return;
        }

        File pdfFile = new File(pdfPath);
        if (!pdfFile.exists()) {
            Toast.makeText(this, "Файл не найден",
Toast.LENGTH_SHORT).show();
            System.out.println("File length: " +
pdfFile.length());
            finish();
            return;
        }

        // Используем явную строку (ваш package name +
.fileprovider)
        Uri uri = FileProvider.getUriForFile(this,

"com.example.roadhelpclientandroid.fileprovider", pdfFile);

        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setDataAndType(uri, "application/pdf");
        intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);

        try {
            startActivity(intent);
        } catch (Exception e) {
            Toast.makeText(this, "Нет приложения для просмотра
PDF", Toast.LENGTH_LONG).show();
        }
    }
}

```

```

        finish();
    }
}

```

Листинг Б.16 – Файл RegisterActivity.java

```

package com.example.roadhelpclientandroid.view;

import android.content.Intent;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import com.example.roadhelpclientandroid.R;
import com.example.roadhelpclientandroid.network.NetworkService;
import com.example.roadhelpclientandroid.storage.SessionManager;

public class RegisterActivity extends AppCompatActivity {

    private EditText etLogin, etPassword;
    private Button btnRegister;
    private SessionManager sessionManager;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);

        sessionManager = SessionManager.getInstance(this);

        etLogin = findViewById(R.id.etLogin);
        etPassword = findViewById(R.id.etPassword);
        btnRegister = findViewById(R.id.btnRegister);

        btnRegister.setOnClickListener(v -> {
            String login = etLogin.getText().toString().trim();
            String password =
etPassword.getText().toString().trim();

            if (login.isEmpty() || password.isEmpty()) {
                Toast.makeText(this, "Заполните логин и пароль",
Toast.LENGTH_SHORT).show();
                return;
            }

            new Thread(() -> {
                // Отправляем login как fullName (пустые phone и
email)

                String response =
NetworkService.getInstance().sendCommand(

```

```

        "REG|" + login + "|" + password + "|" +
login + "|" + "" + "|" + ""
    );
    runOnUiThread(() -> {
        if (response != null &&
response.startsWith("OK|")) {
            String[] parts = response.split("\\|");
            int userId = Integer.parseInt(parts[1]);
            sessionManager.saveUser(userId, login,
"user", login);
            Toast.makeText(RegisterActivity.this,
"Регистрация успешна", Toast.LENGTH_SHORT).show();

            Intent intent = new
Intent(RegisterActivity.this, MainActivity.class);

            intent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
Intent.FLAG_ACTIVITY_NEW_TASK);
            startActivity(intent);
            finish();
        } else {
            String error = response != null ?
response : "Неизвестная ошибка";
            Toast.makeText(RegisterActivity.this,
"Ошибка: " + error, Toast.LENGTH_LONG).show();
        }
    });
    }).start();
});
}
}

```

Листинг Б.17 – Файл DocsViewModel.java

```

package com.example.roadhelpclientandroid.viewmodel;

import android.app.Application;
import android.content.Context;

import androidx.annotation.NonNull;
import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;

import com.example.roadhelpclientandroid.models.Model;
import
com.example.roadhelpclientandroid.repository.ModelRepository;
import com.example.roadhelpclientandroid.utils.PdfCacheManager;

import java.io.File;
import java.util.List;

```

```

public class DocsViewModel extends AndroidViewModel {

    private final ModelRepository repository;
    private final MutableLiveData<List<Model>> models = new
MutableLiveData<>();
    private final MutableLiveData<String> errorMessage = new
MutableLiveData<>();
    private final MutableLiveData<Boolean> isLoading = new
MutableLiveData<>();

    public DocsViewModel(@NonNull Application application) {
        super(application);
        this.repository = new ModelRepository();
    }

    public LiveData<List<Model>> getModels() {
        return models;
    }

    public LiveData<String> getErrorMessage() {
        return errorMessage;
    }

    public LiveData<Boolean> getIsLoading() {
        return isLoading;
    }

    /**
     * Загружает список моделей с сервера
     */
    public void loadModels() {
        isLoading.setValue(true);
        new Thread(() -> {
            try {
                List<Model> result = repository.getAllModels();
                models.postValue(result);
                if (result == null || result.isEmpty()) {
                    errorMessage.postValue("Список моделей
пуст");
                }
            } catch (Exception e) {
                errorMessage.postValue("Ошибка загрузки: " +
e.getMessage());
            } finally {
                isLoading.postValue(false);
            }
        }).start();
    }

    /**
     * Проверяет, есть ли PDF-файл в кэше
     */
    public boolean isPdfCached(Model model) {

```

```

        if (model == null) return false;
        return PdfCacheManager.isPdfCached(getApplication(),
model.getCacheFileName());
    }
    public File getCachedPdfFile(Model model) {
        if (model == null) return null;
        return PdfCacheManager.getPdfFile(getApplication(),
model.getCacheFileName());
    }

    /**
     * Возвращает URL для скачивания PDF (из модели)
     */
    public String getDownloadUrl(Model model) {
        if (model == null) return null;
        return model.getDocFilePath();
    }
    public void clearPdfCache() {
        PdfCacheManager.clearCache(getApplication());
    }
}

```

Листинг Б.18 – Файл activity_main.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"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Контейнер для фрагментов (содержимое экрана) -->
    <FrameLayout
        android:id="@+id/container"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/bottomNav"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>

    <!-- Нижняя панель навигации -->
    <com.google.android.material.bottomnavigation.BottomNavigationVi
ew
        android:id="@+id/bottomNav"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:menu="@menu/bottom_nav_menu"/>

</androidx.constraintlayout.widget.ConstraintLayout>

```

Листинг Б.19 – Файл activity_login.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="32dp"
        android:text="@string/app_name"
        android:textSize="28sp"
        android:textStyle="bold" />

    <EditText
        android:id="@+id/etLogin"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_marginBottom="16dp"
        android:hint="@string/hint_login"
        android:inputType="textEmailAddress" />

    <EditText
        android:id="@+id/etPassword"
        android:layout_width="match_parent"
        android:layout_height="60dp"
        android:layout_marginBottom="24dp"
        android:hint="@string/hint_password"
        android:inputType="textPassword" />

    <Button
        android:id="@+id/btnLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Войти"/>

    <TextView
        android:id="@+id/tvRegisterHint"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="Нет аккаунта? Зарегистрироваться"
        android:textColor="#2196F3" />

</LinearLayout>

```

Листинг Б.20 – Файл activity_register.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:gravity="center"
    android:padding="16dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Регистрация"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginBottom="32dp"/>

    <EditText
        android:id="@+id/etLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Логин"
        android:inputType="textEmailAddress"
        android:layout_marginBottom="16dp"/>

    <EditText
        android:id="@+id/etPassword"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Пароль"
        android:inputType="textPassword"
        android:layout_marginBottom="24dp"/>

    <Button
        android:id="@+id/btnRegister"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Зарегистрироваться"/>

    <TextView
        android:id="@+id/tvLoginHint"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="Уже есть аккаунт? Войти"
        android:textColor="#2196F3" />

</LinearLayout>
```

Листинг Б.21 – Файл activity_add_car.xml

```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <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:text="Добавление автомобиля"
            android:textSize="24sp"
            android:textStyle="bold"
            android:layout_marginBottom="24dp" />

        <Spinner
            android:id="@+id/spinnerModel"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="16dp" />

        <EditText
            android:id="@+id/etYear"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Год выпуска"
            android:inputType="number"
            android:layout_marginBottom="16dp" />

        <EditText
            android:id="@+id/etLicensePlate"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Государственный номер"
            android:layout_marginBottom="16dp" />

        <EditText
            android:id="@+id/etVin"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="VIN-номер"
            android:layout_marginBottom="16dp" />

        <EditText
            android:id="@+id/etMileage"
            android:layout_width="match_parent"

```

```

        android:layout_height="wrap_content"
        android:hint="Пробег (км)"
        android:inputType="number"
        android:layout_marginBottom="16dp" />

<EditText
    android:id="@+id/etStsNumber"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Номер СТС"
    android:layout_marginBottom="16dp" />

<EditText
    android:id="@+id/etInsuranceNumber"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Номер страховки"
    android:layout_marginBottom="16dp" />

<EditText
    android:id="@+id/etInsuranceExpiry"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Дата окончания страховки (ГГГГ-ММ-ДД) "
    android:layout_marginBottom="24dp" />

<Button
    android:id="@+id/btnSave"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Сохранить" />

</LinearLayout>
</ScrollView>

```

Листинг Б.22 – Файл activity_car_detail.xml

```

<?xml version="1.0" encoding="utf-8"?>

<ScrollView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

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

```

```

<TextView
    android:id="@+id/tvCarTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textSize="24sp"
    android:textStyle="bold"
    android:layout_marginBottom="16dp" />

<TextView
    android:id="@+id/tvYear"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp" />

<TextView
    android:id="@+id/tvLicensePlate"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp" />

<TextView
    android:id="@+id/tvVin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp" />

<TextView
    android:id="@+id/tvMileage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp" />

<TextView
    android:id="@+id/tvStsNumber"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp" />

<TextView
    android:id="@+id/tvInsuranceNumber"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp" />

<TextView
    android:id="@+id/tvInsuranceExpiry"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="24dp" />

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"

```

```

        android:orientation="horizontal"
        android:gravity="center_vertical">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Техническое обслуживание"
            android:textSize="18sp"
            android:textStyle="bold" />

        <com.google.android.material.floatingactionbutton.FloatingAction
        Button
            android:id="@+id/fabAddMaintenance"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@android:drawable/ic_input_add"
            android:scaleType="center"
            android:layout_margin="4dp" />

    </LinearLayout>

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvMaintenance"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="24dp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:gravity="center_vertical">

        <TextView
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Частые проблемы модели"
            android:textSize="18sp"
            android:textStyle="bold" />

        <com.google.android.material.floatingactionbutton.FloatingAction
        Button
            android:id="@+id/fabAddIssue"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@android:drawable/ic_input_add"
            android:scaleType="center"
            android:layout_margin="4dp" />

    </LinearLayout>

```

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

    </LinearLayout>
</ScrollView>
```