

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ВЕБ-ПРИЛОЖЕНИЕ МИНИ-CRM УЧЕТА И
АНАЛИЗА ДЕЯТЕЛЬНОСТИ КОМПАНИИ С
ВИЗУАЛИЗАЦИЕЙ ДАННЫХ И СИСТЕМОЙ
ОТЧЕТНОСТИ

Студент Шатин Данила Дмитриевич (ФИО полностью)
(подпись, дата) 25.05.26

Руководитель Гурин Е.И. (фамилия, инициалы)
(подпись, дата) 26.05.26

Нормоконтролёр Бычков А.С. (фамилия, инициалы)
(подпись, дата) 26.05.26

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

Заведующий кафедрой М.А. Митрохин (фамилия, инициалы)
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от
08 июня 2026 г. № 20)

Секретарь ГЭК Е.И. Гурин (фамилия, инициалы)
(подпись)

Пенза 2026

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»



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

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Шатин Данила Дмитриевич гр. 22ВВП1 факультета ФВТ

2. Тема работы: Веб-приложение мини-CRM учета и анализа деятельности компании с визуализацией данных и системой отчетности

Тема утверждена приказом ПГУ № 347/о от " 7 " апреля 2026 г.

3. Руководитель бакалаврской работы Гурин Е.И.

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

Разработать мини-CRM, использующую web-технологии. Язык программирования python, среда разработки Visual Studio Code. Программа должна обладать интуитивно понятным интерфейсом в Мессенджере и использовать следующие технологии: Flask, HTML, CSS, Сетевое программирование, Chart.js, SMTP.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие информационные системы, выделить их достоинства и недостатки. Выявить потребности пользователей. Выбрать язык программирования, среду разработки, СУБД, хостинг которые подойдут для создания мини-CRM. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал библиотек для разработки информационной системы на основе Flask и HTML.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	07.04.2026 – 10.04.2026	
2	Анализ требований	32	11.04.2026 – 14.04.2026	
3	Создание макета графического интерфейса	32	15.04.2026 – 18.04.2026	
4	Выбор технологий, применяемых при проектировании и разработке	40	19.04.2026 – 23.04.2026	
5	Реализация графического интерфейса	66	24.05.2026 – 01.05.2026	
6	Разработка основной части программы	70	02.05.2026 – 10.05.2026	
7	Оформление пояснительной записки	54	11.05.2026 – 21.05.2026	

Дата выдачи задания "7" апреля 2026 г.

Руководитель бакалаврской работы  07.04.2026 Гурин Е.И.
(подпись, дата)

Задание к исполнению принял студент  07.04.2026 Шатин Д.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета  07.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 110 листов, 17 рисунков, 21 таблица, 16 источников и 3 приложения

МИНИ-CRM, WEB-ПРИЛОЖЕНИЕ, АВТОСЕРВИС, УЧЕТ ЗАЯВОК, БАЗА ДАННЫХ, FLASK, SQLITE, PYTHON, BOOTSTRAP, CHART.JS

Цель работы – разработка веб-приложения мини-CRM «АвтоТрейдСервис» для автоматизации учета клиентов, автомобилей, заявок и анализа деятельности автосервисной компании с визуализацией данных и системой отчетности.

Для достижения поставленной цели было разработано веб-приложение, обеспечивающее выполнение следующих задач:

- ведение клиентской базы;
- учет автомобилей клиентов;
- регистрация и обработка заявок на обслуживание;
- назначение ответственных сотрудников;
- хранение сведений об услугах;
- автоматический расчет стоимости работ;
- отображение статистики и формирование отчетных графиков.

В результате проведенной работы разработана мини-CRM с веб-интерфейсом на основе Flask и Bootstrap, реализованная на языке Python. Для хранения данных применена СУБД SQLite. В приложении реализованы средства учета клиентов, автомобилей и заявок, а также инструменты визуализации данных с использованием библиотеки Chart.js.

					ПГУ 1.090301.26.001 ПЗ			
Изм	Лист	№ докум.	Подпись	Дата	Веб-приложение мини-CRM учета и анализа деятельности компании с визуализацией данных и системой отчетности Пояснительная записка	Литера	Лист	Листов
Разраб.	Шатин Д.Д.	1860	25.05.20				2	85
Пров.	Гурин Е.И.	2	25.05.20					
Н. Контр.	Бычков А.С.	26.05.20				ФВТ, каф. ВТ гр. 22ВВП1		
Утв.	Митрохин М.А.	26.05.20						

Содержание

Перечень сокращений и обращений.....	6
Введение	7
1 CRM «АвтоТрейдСервис» для учета и анализа деятельности компании с визуализацией данных и системой отчетности	10
1.1 Анализ предметной области.....	10
1.1.1 Актуальность разработки	10
1.1.2 Анализ существующих аналогов	17
1.1.3 Обоснование актуальности разработки программы	22
1.1.4 Анализ требований к работе и постановка задачи	28
1.2 Выбор технологий и средств разработки приложения	33
1.2.1 Выбор языка программирования и средств разработки	33
1.2.2 Выбор СУБД	38
1.2.3 Выбор среды разработки	43
1.3 Проектирование информационной системы.....	47
1.3.1 Архитектура приложения	47
1.3.2 Проектирование базы данных	53
1.3.3 Разработка приложения	58
1.3.4 Использование приложения	67
1.3.5 Тестирование и отладка	75
2 Безопасность жизнедеятельности	78
Заключение.....	80
Список использованных источников.....	83
Приложение А – UML-диаграммы	86
Приложение Б – Листинги программы.....	92
Приложение В - Презентация	104

Перечень сокращений и обозначений

В данной работе применяют следующие обозначения и сокращения

АС – автоматизированная система

БД – база данных

Десктопное приложение - программа, предназначенная для установки и запуска на персональном компьютере или ноутбуке

Многопоточность - способность программы выполнять несколько потоков (threads) одновременно

Рефакторинг кода - процесс улучшения внутренней структуры программного кода без изменения его внешнего поведения

СУБД – система управления базами данных

API (Application Programming Interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

FK (Foreign key) – внешний ключ, понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных

GUI (Graphical User Interface) – графический интерфейс пользователя

ID (Identifier) – уникальный номер объекта

IDE (Integrated development environment) – интегрированная среда разработки

PK (Primary key) – первичный ключ, понятие теории реляционных баз данных, относящееся к ограничениям целостности базы данных

SQL (Structured Query Language) – язык структурированных запросов

UML (Unified Modeling Language) – унифицированный язык моделирования

Введение

В современных условиях важной областью деятельности сервисных компаний стало информационное обеспечение, связанное со сбором, хранением, обработкой и анализом данных, необходимых для принятия управленческих решений. Организации, работающие с клиентскими обращениями, ежедневно выполняют большое количество операций:

- регистрируют клиентов;
- уточняют сведения об объектах обслуживания;
- оформляют заявки;
- контролируют статус выполнения работ;
- подготавливают отчетность.

Эти процессы могут выполняться вручную, но при увеличении количества обращений такой подход снижает скорость работы и повышает вероятность ошибок.

Развитие информационных систем прошло путь от простых бумажных журналов и локальных таблиц до специализированных CRM-систем, обеспечивающих централизованное хранение клиентских данных и контроль бизнес-процессов. Для автосервисных компаний особенно важны системы, позволяющие учитывать клиентов, автомобили, заявки на обслуживание, перечень услуг и результаты выполненных работ. В небольшой организации такая система должна быть не избыточной, а компактной, понятной и доступной для повседневного использования.

В автосервисной деятельности значительная часть управленческих задач связана с обработкой заявок. Клиент обращается в сервис, сотрудник фиксирует сведения об автомобиле, описывает неисправность, выбирает услуги, рассчитывает стоимость, назначает ответственного мастера и отслеживает состояние работ. Если эти данные хранятся в разных файлах или бумажных журналах, руководителю сложно быстро оценить загрузку сервиса, количество активных заявок, популярность услуг и финансовый результат за период. В связи

					ПГУ 1.090301.26.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

с этим возникает потребность в разработке мини-CRM, объединяющей учетные и аналитические функции.

Актуальность выбранной темы исследования заключается в:

- необходимости повышения качества учета клиентов, автомобилей и заявок автосервисной компании;
- потребности в сокращении времени обработки клиентских обращений за счет автоматизации типовых операций;
- необходимости формирования отчетности и визуализации данных для управленческого анализа;
- снижении риска ошибок, связанных с ручным вводом, дублированием и потерей информации;
- возможности создания компактного программного продукта, не требующего сложной инфраструктуры и дорогостоящего внедрения;
- повышении эффективности деятельности компании за счет применения современных веб-технологий, базы данных и инструментов визуализации.

Объект исследования — деятельность автосервисной компании «АвтоТрейдСервис» по учету клиентов, автомобилей, заявок и результатов оказания услуг.

Предмет исследования — процессы проектирования и разработки веб-приложения мини-CRM для автоматизации учета и анализа деятельности автосервисной компании.

Цель исследования — разработать веб-приложение мини-CRM «АвтоТрейдСервис» (далее – «мини-CRM») для учета клиентов, автомобилей, заявок, анализа деятельности компании, визуализации данных и формирования отчетности.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- «Анализ предметной области». Изучить особенности учета клиентов, автомобилей и заявок в автосервисной компании, определить основные проблемы ручного и разрозненного учета;

- «Анализ аналогов». Рассмотреть существующие программные решения для автоматизации работы с клиентами и заявками, выявить их преимущества и ограничения;
- «Выбор технологий». Обосновать выбор языка программирования, СУБД, среды разработки и дополнительных средств создания веб-приложения;
- «Проектирование системы». Разработать архитектуру приложения, логическую и физическую модель базы данных, определить основные программные модули;
- «Разработка приложения». Реализовать мини-CRM с использованием Python, Flask, SQLite, HTML, CSS, Bootstrap и Chart.js;
- «Тестирование системы». Проверить работоспособность основных функций приложения: учет клиентов, автомобилей, заявок, услуг и формирование отчетов.

При создании выпускной квалификационной работы применялись методы анализа, сравнения, моделирования, проектирования информационных систем, структурного описания данных, метод аналогии, системный подход, а также методы функционального тестирования программного обеспечения.

Выпускная квалификационная работа состоит из введения, трех глав, заключения, списка использованных источников и приложения. Во введении формулируются актуальность, цель и задачи исследования. В Главе 1 проводится анализ предметной области, рассматриваются существующие аналоги и формулируется постановка задачи. В Главе 2 обосновывается выбор технологий и средств разработки. В Главе 3 выполняются проектирование информационной системы, разработка приложения, описание его использования, тестирование, анализ вопросов безопасности жизнедеятельности. В заключении подводятся итоги выполненной работы.

В работе имеется 21 таблица, 17 иллюстраций и 16 библиографических источников.

1 CRM «АвтоТрейдСервис» для учета и анализа деятельности компании с визуализацией данных и системой отчетности

1.1 Анализ предметной области

1.1.1 Актуальность разработки

Объектом разработки в ВКР является веб-приложение мини-CRM «АвтоТрейдСервис» для учета и анализа деятельности автосервисной компании «АвтоТрейдСервис» (далее — Предприятие). Предприятие относится к сфере сервисного обслуживания автомобилей и выполняет типовые для такого бизнеса операции:

- прием обращений клиентов;
- оформление заявок;
- учет автомобилей;
- назначение сотрудников;
- контроль статусов ремонта;
- хранение истории обслуживания;
- подготовку отчетности.

Автосервис как вид деятельности тесно связан с постоянным потоком обращений. Клиент может записаться на диагностику, оставить автомобиль на ремонт, заказать замену расходных материалов, уточнить состояние работ или получить итоговый расчет. Каждое такое действие требует фиксации данных. Если учет ведется вручную, часть сведений быстро теряется:

- администратор записал заявку в блокнот;
- мастер уточнил проблему устно;
- руководитель получил итоговую сумму уже после закрытия смены.

В небольшой компании это некоторое время кажется допустимым. Но при росте числа клиентов такой подход начинает мешать работе.

В современных условиях информационные системы становятся важным инструментом управления малым и средним бизнесом. Они позволяют не только хранить сведения, но и связывать отдельные операции в единый управляемый

процесс: от первого обращения клиента до формирования отчета по выполненным работам [1, 15]. Для автосервиса это особенно важно, потому что качество обслуживания зависит не только от ремонта автомобиля, но и от точности клиентского учета, скорости обработки заявки и прозрачности статусов.

Мини-CRM в такой задаче выступает не как сложная корпоративная система, а как компактный программный продукт, закрывающий основные потребности сервисной компании. Его назначение — убрать разрозненные записи, сократить ручной поиск информации, дать руководителю простую аналитику и повысить управляемость заявок. При этом система не должна быть перегружена лишними функциями. Для малого автосервиса важны понятный интерфейс, быстрый запуск, невысокие требования к оборудованию и возможность работать с базовыми сущностями:

- клиентами;
- автомобилями;
- заявками;
- услугами;
- сотрудниками;
- отчетами.

Основные проблемы ручного и разрозненного учета в автосервисной компании приведены в Таблице 1.1.1.

Таблица 1.1.1 — Основные проблемы учета заявок и клиентов в автосервисной компании

Проблема	Проявление в работе автосервиса	Последствие для управления
Разрозненное хранение данных	Клиенты, автомобили и заявки фиксируются в разных файлах или бумажных журналах	Увеличивается время поиска сведений
Отсутствие единой истории клиента	Информация о прошлых обращениях не связана с новым заказом	Сложнее оценить лояльность клиента и повторные обращения
Слабый контроль статусов	Руководитель не видит, какие заявки новые, какие находятся в работе и какие завершены	Возникают задержки и спорные ситуации

Проблема	Проявление в работе автосервиса	Последствие для управления
Ручная подготовка отчетов	Итоги по заявкам и услугам собираются вручную	Повышается риск ошибок в отчетности
Низкая прозрачность загрузки сотрудников	Назначение мастеров ведется без общей картины по заявкам	Сложнее распределять нагрузку
Ограниченная аналитика	Нет быстрых графиков по заявкам, услугам и результатам работы	Руководитель принимает решения без полной картины

Из Таблицы 1.1.1 видно, что главная проблема состоит не в отсутствии отдельных записей, а в отсутствии единого информационного контура. Данные вроде бы есть, но они не связаны между собой. Клиент хранится отдельно, автомобиль — отдельно, заявка — в другом файле, а итоговая сумма попадает в отчет только после ручной обработки. Из-за этого сотрудники тратят время не на обслуживание клиента, а на поиск и уточнение информации.

Для решения этой задачи требуется веб-приложение, которое объединяет учетные операции и аналитические функции. Система должна позволять администратору быстро создать карточку клиента, связать ее с автомобилем, оформить заявку, указать услугу и назначить ответственного сотрудника. Руководитель, в свою очередь, должен получать сводные показатели:

- количество заявок;
- распределение по статусам;
- популярные услуги;
- динамику обращений;
- итоговые суммы.

Такой подход соответствует общему направлению цифрового развития бизнеса, где информационная система становится средством повышения качества управления, а не просто электронным хранилищем данных [2]. Проблемный контур учета при отсутствии мини-CRM приведен на Рисунке 1.1.1.

Рисунок 1.1.1 показывает, что при ручной обработке данные проходят через несколько слабых мест. На каждом этапе возможны задержка, неполная запись или потеря сведений. Например, администратор может не указать точную модель автомобиля, мастер может не получить полное описание неисправности, а руководитель увидит проблему только после закрытия смены. В системе мини-

CRM эти операции должны выполняться через единую базу данных, что снижает зависимость от устных договоренностей и отдельных файлов.



Рисунок 1.1.1.1 — Проблемный контур учета клиентов и заявок при ручной обработке данных

Актуальность разработки также связана с тем, что готовые CRM-системы часто избыточны для небольшого автосервиса. Универсальные продукты имеют широкий набор модулей:

- продажи;
- маркетинг;
- рассылки;
- телефония;
- сложные воронки;

- интеграции с внешними сервисами.

Для малого предприятия такой объем функций может быть лишним. Сотрудникам нужен инструмент, который решает ежедневные задачи без долгого обучения. Избыточность интерфейса в этом случае снижает полезность продукта: система вроде бы мощная, но пользователь возвращается к привычной таблице, потому что так быстрее.

В то же время простая электронная таблица не решает задачу полноценно. Она помогает хранить строки данных, но плохо поддерживает связи между сущностями. Клиент может иметь несколько автомобилей, один автомобиль — несколько заявок, заявка — несколько услуг, а услуга должна попадать в отчет. Для таких связей требуется база данных и приложение, которое правильно обрабатывает операции добавления, изменения, удаления и поиска записей. Основы проектирования баз данных позволяют выстроить такую структуру без дублирования информации и с сохранением целостности данных [3].

Разработка мини-CRM «АвтоТрейдСервис» также важна с позиции отчетности. Руководителю автосервиса нужно быстро видеть состояние работы:

- сколько заявок открыто;
- сколько заявок завершено;
- какие услуги наиболее востребованы;
- кто из сотрудников загружен сильнее;
- как меняется поток обращений.

Без автоматизации эти сведения собираются вручную. В лучшем случае это занимает дополнительное время, в худшем — руководитель получает неполную картину. Веб-приложение с модулем визуализации данных позволяет перейти от набора отдельных записей к управленческим показателям.

Важное значение имеет и доступность выбранного формата приложения. Веб-интерфейс удобен для пользователей, потому что не требует сложной установки на каждое рабочее место. Сотрудник открывает систему в браузере, проходит авторизацию и работает с нужным разделом. Для учебного и прикладного проекта такой подход позволяет наглядно показать работу

					ПГУ 1.090301.26.001 ПЗ	Лист
						14
Изм.	Лист	№ докум.	Подпись	Дата		

клиентской части, серверной части и базы данных. Разработка с применением Python и Flask хорошо подходит для таких задач. Flask позволяет:

- быстро создать серверную часть;
- связать маршруты с HTML-шаблонами;
- организовать обработку данных [4].

С точки зрения интерфейса мини-CRM должна быть не только рабочей, но и понятной. Пользователь автосервиса не обязан быть специалистом в ИТ. Ему нужно быстро найти клиента, открыть заявку, поменять статус и получить отчет. Поэтому в проекте важны простые формы, четкие кнопки, визуальные карточки и таблицы. Использование Bootstrap позволяет оформить интерфейс в едином стиле и ускорить разработку адаптивных страниц [5]. Для графиков и отчетных элементов подходит Chart.js, поскольку библиотека дает возможность строить диаграммы прямо на страницах веб-приложения [6].

Актуальность разработки можно рассмотреть через несколько групп факторов:

- организационный фактор. Автосервис работает с повторяющимися действиями: запись клиента, прием автомобиля, оформление заявки, контроль выполнения, закрытие работ. При ручном учете эти действия разорваны между сотрудниками, а мини-CRM объединяет их в единый рабочий процесс;

- информационный фактор. Данные о клиентах, автомобилях и ремонтах имеют ценность только тогда, когда они доступны быстро и связаны между собой. Единая база данных позволяет хранить историю обслуживания и использовать ее при новых обращениях;

- управленческий фактор. Руководителю нужны не только записи, но и показатели. Система отчетности и визуализации помогает оценить поток заявок, популярность услуг, загруженность сотрудников и результаты работы за период;

- экономический фактор. Даже небольшое сокращение времени на поиск клиента, оформление заявки и подготовку отчета дает накопленный эффект. Сотрудники меньше времени тратят на ручную обработку и больше внимания уделяют клиентскому сервису;

– технический фактор. Стек Python, Flask, SQLite, Bootstrap и Chart.js позволяет создать приложение без избыточной сложности. Оно будет прозрачным по архитектуре, понятным для сопровождения и достаточным для задач ВКР.

Для создаваемой системы особенно важно сочетание простоты и достаточной функциональности. Если приложение будет слишком простым, оно превратится в набор форм без ценности для управления. Если сделать его слишком сложным, возрастут сроки разработки и риск ошибок. Поэтому в проекте выбирается промежуточный вариант — мини-CRM, включающая:

- базовые справочники;
- заявки;
- статусы;
- отчеты;
- графики.

Такой продукт показывает полноценную работу информационной системы, но остается реалистичным для разработки.

В учебной и инженерной части работа также имеет значение. Разработка мини-CRM позволяет показать полный цикл создания программного продукта:

- анализ предметной области;
- постановку задачи;
- выбор технологий;
- проектирование архитектуры;
- проектирование базы данных;
- программную реализацию;
- описание пользовательских сценариев;
- тестирование.

Такой состав хорошо соответствует требованиям к созданию автоматизированной системы, где важны стадии анализа, проектирования, реализации и проверки результата [7].

Итоговая актуальность разработки мини-CRM «АвтоТрейдСервис» заключается в необходимости создать простое, доступное и наглядное веб-

приложение для автосервисной компании. Оно должно заменить разрозненный ручной учет, обеспечить хранение данных в единой базе, ускорить обработку заявок и дать руководителю инструменты анализа. За счет этого система решает не отдельную техническую задачу, а более широкую задачу повышения управляемости сервисной деятельности Предприятия.

1.1.2 Анализ существующих аналогов

Развитие цифровых сервисов в сфере малого и среднего бизнеса привело к активному распространению CRM-систем, предназначенных для автоматизации работы с клиентами, заявками и внутренними процессами компании. Особенно заметно это в сервисных организациях, где важны скорость обработки обращений, прозрачность статусов и сохранение истории взаимодействия с клиентом. Автосервисные компании также постепенно переходят от бумажного учета и электронных таблиц к специализированным информационным системам, позволяющим централизованно хранить данные и контролировать выполнение работ.

Для разработки мини-CRM «АвтоТрейдСервис» необходимо рассмотреть существующие решения, определить их достоинства и недостатки, а также выявить функции, которые действительно востребованы в условиях небольшой сервисной компании. Анализ аналогов позволяет избежать избыточности системы и выбрать рациональный набор возможностей будущего приложения [1].

Все рассматриваемые решения условно можно разделить на три группы:

- универсальные CRM-системы;
- специализированные системы для автосервисов;
- простые средства учета на базе электронных таблиц.

Универсальные CRM-системы ориентированы на широкий круг компаний и обычно содержат большое количество модулей:

- продажи;
- маркетинг;
- телефония;
- управление задачами;

- документооборот;
- аналитику;
- интеграции с внешними сервисами.

Такие решения подходят крупным организациям, но для небольшого автосервиса часто оказываются избыточными. Пользователь получает сложный интерфейс и множество функций, которые фактически не используются.

Специализированные системы для автосервисов создаются с учетом особенностей сервисного обслуживания автомобилей. В них обычно предусмотрены карточки автомобилей, учет выполненных работ, формирование заказ-нарядов, работа с клиентской базой и контроль статусов ремонта. Подобные продукты ближе к задачам рассматриваемой предметной области, однако многие из них ориентированы на коммерческое внедрение и содержат платные модули, сложную настройку или зависимость от облачной инфраструктуры [8].

Третья группа — электронные таблицы и локальные файлы учета. Такой вариант часто используется небольшими компаниями на раннем этапе работы. Его достоинство заключается в простоте и отсутствии затрат на внедрение. Однако по мере роста количества клиентов и заявок появляются серьезные ограничения, например:

- данные начинают дублироваться;
- становится сложно контролировать историю ремонта;
- поиск информации занимает больше времени;
- отчеты приходится формировать вручную.

Для определения сильных и слабых сторон существующих решений проведем сравнительный анализ нескольких популярных продуктов и подходов. Результаты сравнения приведены в Таблице 1.1.2.1.

Из Таблицы 1.1.2.1 видно, что существующие решения либо перегружены дополнительными функциями, либо не обеспечивают полноценного учета и аналитики. Для небольшого автосервиса важен баланс между функциональностью и простотой использования. Именно этот баланс предполагается реализовать в мини-CRM «АвтоТрейдСервис».

Таблица 1.1.2.1 — Сравнение существующих решений для учета клиентов и заявок автосервиса

Решение	Основные возможности	Преимущества	Недостатки
YCLIENTS[16]	Онлайн-запись, клиентская база, расписание, уведомления	Современный интерфейс, облачный доступ	Избыточность функций для малого автосервиса
АвтоДилер Онлайн	Учет ремонтов, заказ-наряды, история обслуживания	Специализация под автосервис	Высокая зависимость от коммерческой платформы
Универсальные CRM	Работа с клиентами и сделками	Гибкость настройки	Сложность интерфейса и высокая нагрузка на пользователя
Электронные таблицы	Хранение записей вручную	Простота использования	Нет единой структуры и аналитики
Разрабатываемая мини-CRM «АвтоТрейдСервис»	Учет клиентов, автомобилей, заявок и отчетность	Простота, адаптация под задачи автосервиса	Ограниченная масштабируемость

С точки зрения архитектуры большинство современных CRM-систем используют веб-технологии. Такой подход позволяет сотрудникам работать через браузер без установки тяжелого программного обеспечения на каждое рабочее место. Кроме того, веб-приложение проще обновлять и сопровождать. По этой причине для разрабатываемой системы также выбран веб-формат реализации.

При анализе существующих аналогов особое внимание следует уделить пользовательскому интерфейсу. В сервисной компании сотрудники работают в условиях постоянного потока клиентов и не могут тратить много времени на изучение сложного программного обеспечения. Поэтому современные CRM-системы стремятся использовать:

- компактные формы;
- понятные таблицы;
- цветовые статусы;
- карточки клиентов;
- быстрый поиск;
- панели аналитики.

Интерфейс мини-CRM «АвтоТрейдСервис» также должен строиться по этим принципам. Пользователь должен быстро ориентироваться в системе, видеть ключевые показатели и выполнять основные действия без лишних переходов между страницами. Bootstrap как библиотека пользовательского интерфейса позволяет реализовать подобный подход достаточно быстро и при этом сохранить единый стиль страниц [5].

Для систем учета и управления важна не только работа с данными, но и возможность их анализа. Многие современные CRM-продукты содержат встроенные графики и панели мониторинга. Это позволяет руководителю оценивать:

- поток клиентов;
- динамику заявок;
- загрузку сотрудников;
- популярность услуг;
- финансовые показатели.

Пример распределения функциональных возможностей современных CRM-систем приведен на Рисунке 1.1.2.1.

Из Рисунка 1.1.2.1 видно, что современные CRM-системы строятся вокруг единого центра обработки данных. Все модули связаны между собой и работают через общую информационную среду. Это позволяет хранить историю взаимодействия с клиентом, контролировать состояние заявок и автоматически формировать отчеты.

При анализе аналогов важно учитывать не только функциональность, но и сложность сопровождения и поделучения системы. Коммерческие продукты часто требуют:

- постоянной подписки;
- внешнего сервера;
- обновлений платформы;
- подключения дополнительных модулей;
- обучения сотрудников.

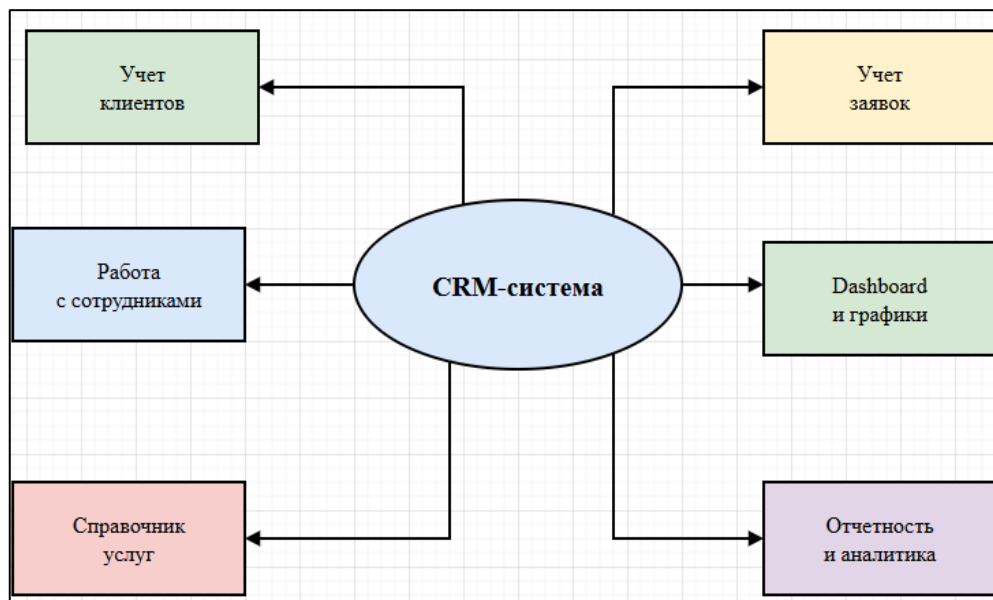


Рисунок 1.1.2.1 — Распределение функциональных возможностей современных CRM-систем

Для проекта и небольшого предприятия такой подход не всегда рационален. Поэтому разрабатываемая мини-CRM должна быть:

- автономной;
- понятной;
- простой в развертывании;
- удобной для демонстрации;
- прозрачной по архитектуре.

С технической точки зрения современные веб-приложения для управления данными обычно строятся по клиент-серверной модели. Серверная часть обрабатывает запросы пользователя, выполняет операции с базой данных и возвращает готовые страницы интерфейса. Для реализации такой архитектуры хорошо подходит Flask — легковесный Python-фреймворк для веб-разработки [4]. Его преимуществом является простота структуры проекта, возможность быстро создавать маршруты и удобная интеграция с HTML-шаблонами.

Для хранения данных в небольших системах часто используется SQLite. Такая СУБД не требует отдельного сервера базы данных и позволяет хранить всю структуру приложения в одном локальном файле. Для мини-CRM малого

автосервиса этого достаточно, поскольку система не рассчитана на высокую промышленную нагрузку [9].

В результате проведенного анализа можно сделать несколько выводов:

- Универсальные CRM-системы обладают широкими возможностями, но часто перегружены функциями, не востребованными в небольшом автосервисе;
- Специализированные решения для сервисных компаний ближе к предметной области, однако обычно ориентированы на коммерческое внедрение и зависят от внешней инфраструктуры;
- Простые электронные таблицы удобны только на начальном этапе работы и не обеспечивают полноценного учета, аналитики и контроля связей между данными.

Следовательно, разработка мини-CRM «АвтоТрейдСервис» является обоснованной задачей. Создаваемое приложение должно объединить основные возможности современных CRM-систем, но при этом сохранить простоту интерфейса, прозрачную структуру и удобство работы для сотрудников автосервисной компании [1, 4].

1.1.3 Обоснование актуальности разработки программы

Разработка веб-приложения мини-CRM «АвтоТрейдСервис» связана с потребностью автосервисных компаний в более точном и удобном учете клиентов, автомобилей, заявок и результатов выполненных работ. Для малой сервисной компании особенно важны простота внедрения, понятный интерфейс и возможность быстро получать управленческую информацию без сложной настройки. Именно поэтому создаваемая система ориентирована не на широкий набор лишних модулей, а на базовые процессы автосервиса:

- регистрацию клиента;
- учет автомобиля;
- оформление заявки;
- контроль статуса;
- расчет общей стоимости услуг;
- получение отчетов.

История развития подобных решений связана с общим переходом бизнеса от бумажного учета к цифровым системам. Сначала многие компании использовали журналы регистрации, затем электронные таблицы, позже — локальные базы данных и CRM-системы. На каждом этапе повышалась скорость обработки информации, но вместе с этим росли требования к качеству данных. Если раньше было достаточно записать фамилию клиента и номер телефона, то сейчас сервисной компании важно видеть полную историю:

- какой автомобиль обслуживался;
- какие услуги выполнялись;
- какие неисправности повторялись;
- кто был ответственным мастером;
- как менялась сумма заказа.

Для автосервиса такая история имеет прямое прикладное значение. Один клиент может обслуживать несколько автомобилей, один автомобиль может иметь несколько заявок, а каждая заявка — включать разные услуги. При ручном учете эти связи быстро становятся неудобными. Сотруднику приходится искать сведения в разных местах, сверять даты, уточнять прошлые обращения. В базе данных эти отношения можно выстроить значительно аккуратнее:

- клиент связан с автомобилем;
- автомобиль — с заявкой;
- заявка — с услугами и сотрудником.

Такой подход соответствует базовым принципам проектирования информационных систем и баз данных [1, 3].

Актуальность разработки усиливается тем, что автосервисная компания работает в условиях постоянного контакта с клиентом. Клиент ожидает, что его обращение не потеряется, статус ремонта будет понятен, а история обслуживания сохранится. Если сотрудник не может быстро найти предыдущую заявку или уточнить выполненные работы, доверие к сервису снижается. Даже хороший технический ремонт может восприниматься хуже, если учет и коммуникация организованы слабо.

Создаваемое приложение должно закрывать эту проблему за счет единого рабочего пространства. Администратор получает возможность быстро внести клиента и автомобиль, мастер — видеть назначенную заявку, руководитель — контролировать показатели по заявкам и услугам. Такая структура делает работу более прозрачной и снижает зависимость от памяти отдельных сотрудников. Основные группы пользователей мини-CRM «АвтоТрейдСервис» приведены в Таблице 1.1.3.1.

Таблица 1.1.3.1 — Группы пользователей мини-CRM «АвтоТрейдСервис» и их потребности

Группа пользователей	Основные задачи	Потребность в системе
Администратор	Прием клиентов, регистрация заявок, поиск данных	Быстрое оформление обращения и доступ к карточке клиента
Мастер-приемщик	Уточнение неисправности, передача заявки в работу	Просмотр заявки, автомобиля и перечня услуг
Сотрудник сервиса	Выполнение работ, изменение состояния заявки	Понятный статус и привязка работ к конкретному заказу
Руководитель	Контроль работы сервиса, анализ заявок и отчетов	Сводные показатели, графики и отчеты по деятельности
Клиент	Получение услуги и прозрачного результата	Сокращение времени ожидания и сохранение истории обслуживания

Из Таблицы 1.1.3.1 видно, что система нужна не одному пользователю, а нескольким участникам процесса. Для администратора важна скорость ввода данных. Для мастера — полнота информации по автомобилю и заявке. Для руководителя — отчетность и аналитика. Поэтому мини-CRM должна быть не просто журналом обращений, а компактной информационной системой, объединяющей учетные и аналитические функции.

Отдельно следует отметить востребованность визуализации данных. В обычной таблице можно увидеть список заявок, но трудно быстро понять, растет ли нагрузка, какие услуги наиболее популярны, сколько заявок находится в работе и как меняется поток обращений. Графики и диаграммы позволяют быстрее увидеть тенденции. Поэтому в системе «АвтоТрейдСервис» предусмотрена отчетная панель с визуальными показателями. Для их вывода

целесообразно использовать Chart.js, так как эта библиотека позволяет строить графики прямо в веб-интерфейсе [6]. Расчетные данные для построения графика динамики роста количества заявок автосервиса приведены в Таблице 1.1.3.2.

Динамика роста количества заявок автосервиса при использовании мини-CRM приведена на Рисунке 1.1.3.1.

Таблица 1.1.3.2 — Расчетные данные для построения графика динамики роста количества заявок автосервиса

Месяц	Количество заявок при ручном учете	Количество заявок при использовании мини-CRM	Прирост, заявок
Январь	105	105	0
Февраль	108	116	8
Март	111	129	18
Апрель	109	143	34
Май	112	158	46
Июнь	114	171	57

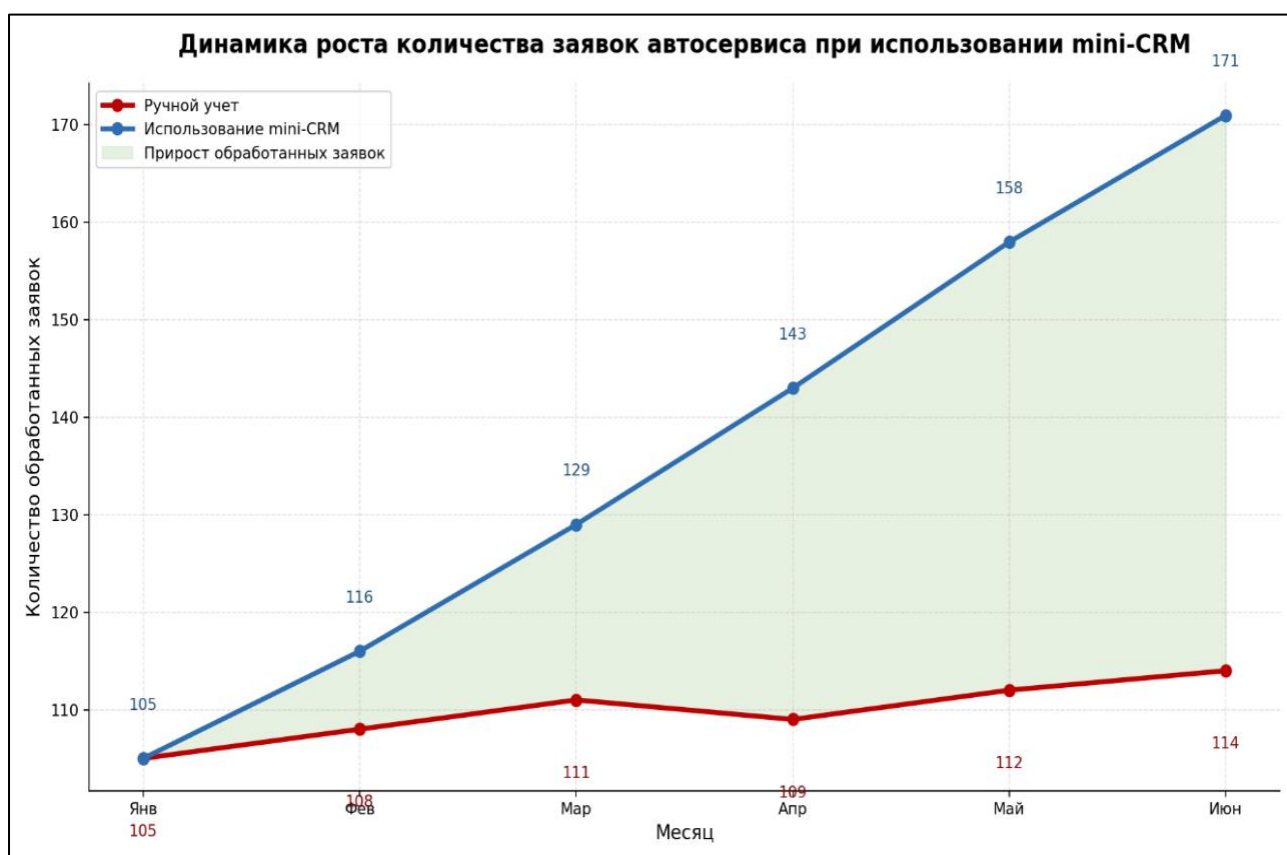


Рисунок 1.1.3.1 — Динамика роста количества заявок автосервиса при использовании мини-CRM

Рисунок 1.1.3.1 показывает ожидаемое изменение обработки заявок при переходе от ручного учета к мини-CRM. При ручной работе рост ограничен, потому что администратор и руководитель тратят время на поиск сведений, сверку записей и подготовку отчетов. При использовании системы часть этих действий становится быстрее:

- данные сохраняются в единой базе;
- заявка получает статус;
- отчетные показатели формируются автоматически.

Такой эффект особенно заметен не в первый день внедрения, а через несколько месяцев, когда в системе накапливается история клиентов и автомобилей.

Разработка программы актуальна также из-за роста требований к управляемости сервисных процессов. Автосервис не может рассматриваться только как место выполнения технических работ. Это бизнес, в котором важны клиентская база, скорость обслуживания, повторные обращения, качество записи, контроль сотрудников и своевременная отчетность. Информационная система помогает связать эти стороны в один управляемый процесс [2, 14].

При этом мини-CRM не должна заменять профессиональные знания сотрудников. Ее назначение другое — поддерживать их работу. Система не принимает решение за мастера, но дает ему сведения о клиенте и автомобиле. Она не управляет бизнесом вместо руководителя, но предоставляет показатели для анализа. Она не отменяет личное общение с клиентом, но снижает риск потерять важную информацию после разговора.

Востребованность создаваемого продукта определяется несколькими факторами:

- рост числа обращений. Чем больше клиентов обслуживает автосервис, тем сложнее вести записи вручную. При увеличении потока даже небольшие ошибки начинают повторяться и влиять на качество работы;
- хранение истории. Клиентская история помогает лучше понимать повторные обращения, предыдущие неисправности и выполненные услуги;

– потребность в отчетности. Руководителю нужны быстрые сведения о количестве заявок, завершенных работах, популярных услугах и загрузке сотрудников. Без автоматизации такие отчеты требуют отдельной ручной подготовки;

– ограниченность ресурсов малого предприятия. Небольшой автосервис не всегда может внедрить сложную коммерческую платформу. Компактное веб-приложение позволяет решить основные задачи без избыточных затрат;

– простота доступа. Веб-интерфейс удобен для сотрудников, потому что работа выполняется через браузер. Это снижает требования к рабочему месту и упрощает сопровождение приложения.

С технической стороны разработка мини-CRM хорошо согласуется с выбранным стеком. Python подходит для быстрой реализации серверной части, Flask позволяет организовать маршруты и обработку запросов, SQLite обеспечивает хранение данных без отдельного сервера, Bootstrap упрощает создание аккуратного интерфейса, а Chart.js закрывает задачу визуализации показателей [4, 5]. Такой набор технологий достаточен для компактной системы учета и анализа деятельности автосервиса.

Разрабатываемая программа должна включать несколько ключевых модулей. Первый модуль — учет клиентов. В нем сохраняются ФИО, телефон, электронная почта и дополнительные сведения. Второй модуль — учет автомобилей, связанных с клиентами. Здесь фиксируются марка, модель, год выпуска и государственный номер. Третий модуль — заявки на обслуживание. Он должен хранить описание проблемы, статус, дату создания, назначенного сотрудника и итоговую стоимость. Четвертый модуль — услуги, используемые при оформлении заявки. Пятый модуль — отчетность и визуализация.

Такая структура позволяет показать полноценный цикл работы системы. Сначала создается клиент, затем добавляется автомобиль, после этого оформляется заявка и выбираются услуги. По мере выполнения работ статус меняется. В конце руководитель видит отчетные данные. Это делает приложение цельным, а не набором отдельных экранов.

Актуальность разработки программы также подтверждается тем, что существующие коммерческие решения не всегда подходят небольшим компаниям. Они могут требовать оплаты подписки, сложного внедрения, настройки внешних сервисов и обучения персонала. Для малого автосервиса важнее другой подход:

- минимальный набор нужных функций;
- простые формы;
- локальная база данных;
- понятные отчеты.

В этом случае мини-CRM становится не заменой крупной платформы, а более подходящим инструментом для конкретных условий.

В результате разработка мини-CRM «АвтоТрейдСервис» обоснована с организационной, технической и управленческой сторон. Система нужна для повышения качества учета, сокращения ручной работы, сохранения истории обслуживания и формирования наглядной отчетности. Ее применение позволит Предприятию быстрее обрабатывать заявки, лучше контролировать состояние работ и получать данные для принятия управленческих решений.

1.1.4 Анализ требований к работе и постановка задачи

Разработка мини-CRM «АвтоТрейдСервис» требует предварительного определения требований к будущей системе. На этом этапе важно описать, какие задачи должна решать программа, какие пользователи будут с ней работать, какие данные необходимо хранить и какие ограничения нужно учитывать при проектировании [11]. Без такой фиксации приложение легко превратить в набор отдельных форм, которые технически работают, но не образуют единую систему учета.

Основная потребность Предприятия состоит в совершенствовании обработки клиентских заявок. В текущей схеме небольшого автосервиса значительная часть операций может выполняться вручную:

- клиент обращается к администратору;
- данные заносятся в таблицу или журнал;

- информация об автомобиле уточняется отдельно;
- статус ремонта передается устно;
- итоговый отчет формируется после ручного подсчета.

Такая схема приемлема только при малом числе обращений. При росте нагрузки она создает задержки, повышает риск ошибок и затрудняет контроль работы сотрудников.

Проектируемая система должна устранить эти недостатки за счет единой базы данных и веб-интерфейса. Пользователь вводит сведения один раз, после чего они становятся доступными в связанных разделах приложения. Клиент связывается с автомобилем, автомобиль — с заявкой, заявка — с услугами и сотрудником. Такой подход снижает дублирование информации и делает учет более надежным. При проектировании базы данных важно заранее определить сущности, их атрибуты и связи, чтобы обеспечить целостность хранения сведений [3].

Требования к мини-CRM можно разделить на функциональные и технические. Функциональные требования описывают, что именно должна делать система. Технические требования определяют качество работы:

- удобство интерфейса;
- простоту запуска;
- скорость обработки;
- надежность хранения;
- понятность сопровождения.

Оба блока одинаково важны. Если приложение имеет нужные функции, но неудобно для пользователя, сотрудники будут избегать его применения. Если интерфейс удобный, но отсутствуют важные операции, система не решит задачу учета.

1.1.4.1 Функциональные требования.

Функциональные требования к мини-CRM «АвтоТрейдСервис» приведены в Таблице 1.1.4.1.

					ПГУ 1.090301.26.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

Из Таблицы 1.4.1 видно, что система должна закрывать полный цикл работы с обращением клиента. Цикл начинается с регистрации клиента и автомобиля, продолжается созданием заявки и завершается анализом итоговых показателей. Важным условием является связь между разделами. Например, заявка не должна существовать отдельно от клиента и автомобиля, потому что иначе теряется история обслуживания.

Таблица 1.1.4.1 — Функциональные требования к мини-CRM «АвтоТрейдСервис»

Группа требований	Содержание требования	Назначение
Авторизация	Вход пользователя по логину и паролю	Ограничение доступа к системе
Учет клиентов	Добавление, просмотр, редактирование и удаление карточек клиентов	Ведение клиентской базы
Учет автомобилей	Хранение сведений о марке, модели, годе выпуска и номере автомобиля	Привязка заявок к конкретному автомобилю
Учет заявок	Создание заявки, назначение статуса, сотрудника и перечня работ	Управление сервисным процессом
Справочник услуг	Хранение перечня услуг и ориентировочной стоимости	Упрощение оформления заявки
Отчетность	Формирование сводных данных по заявкам, услугам и статусам	Поддержка управленческого анализа
Визуализация	Отображение графиков и диаграмм на панели управления	Быстрая оценка состояния работы
Поиск и фильтрация	Поиск клиентов, автомобилей и заявок по основным полям	Сокращение времени обработки данных

1.1.4.2 Технические требования.

Технические требования к системе включают несколько положений. Интерфейс должен быть простым и понятным, чтобы сотрудник мог выполнять основные действия без долгого обучения. Приложение должно работать через браузер, поскольку такой вариант снижает требования к рабочему месту и упрощает доступ к системе. Хранение данных предполагается организовать в SQLite, так как эта СУБД подходит для небольшого приложения и не требует отдельного сервера базы данных [9]. Серверная часть реализуется на Flask, что

позволяет построить веб-приложение с понятной структурой маршрутов и шаблонов [4].

Особое значение имеет визуальная отчетность. Руководитель автосервиса не должен вручную подсчитывать количество заявок за период или искать популярные услуги в таблицах. Система должна предоставлять сводные показатели и графики. Для этого в интерфейс целесообразно включить панель управления, на которой отображаются количество заявок, распределение по статусам, динамика обращений и структура услуг. Такой подход соответствует современному использованию информационных систем в управлении бизнес-процессами [2].

Одной из задач разработки является сокращение времени обработки клиентской заявки. Чтобы это показать, можно сравнить ручной способ учета и работу через мини-CRM по основным операциям. Расчетные данные для сравнения времени обработки клиентской заявки приведены в Таблице 1.1.4.2.

Сравнение времени обработки клиентской заявки до и после внедрения системы приведено на Рисунке 1.1.4.1.

Диаграмма вариантов использования приложения приведена на Рисунке 1.1.4.2.

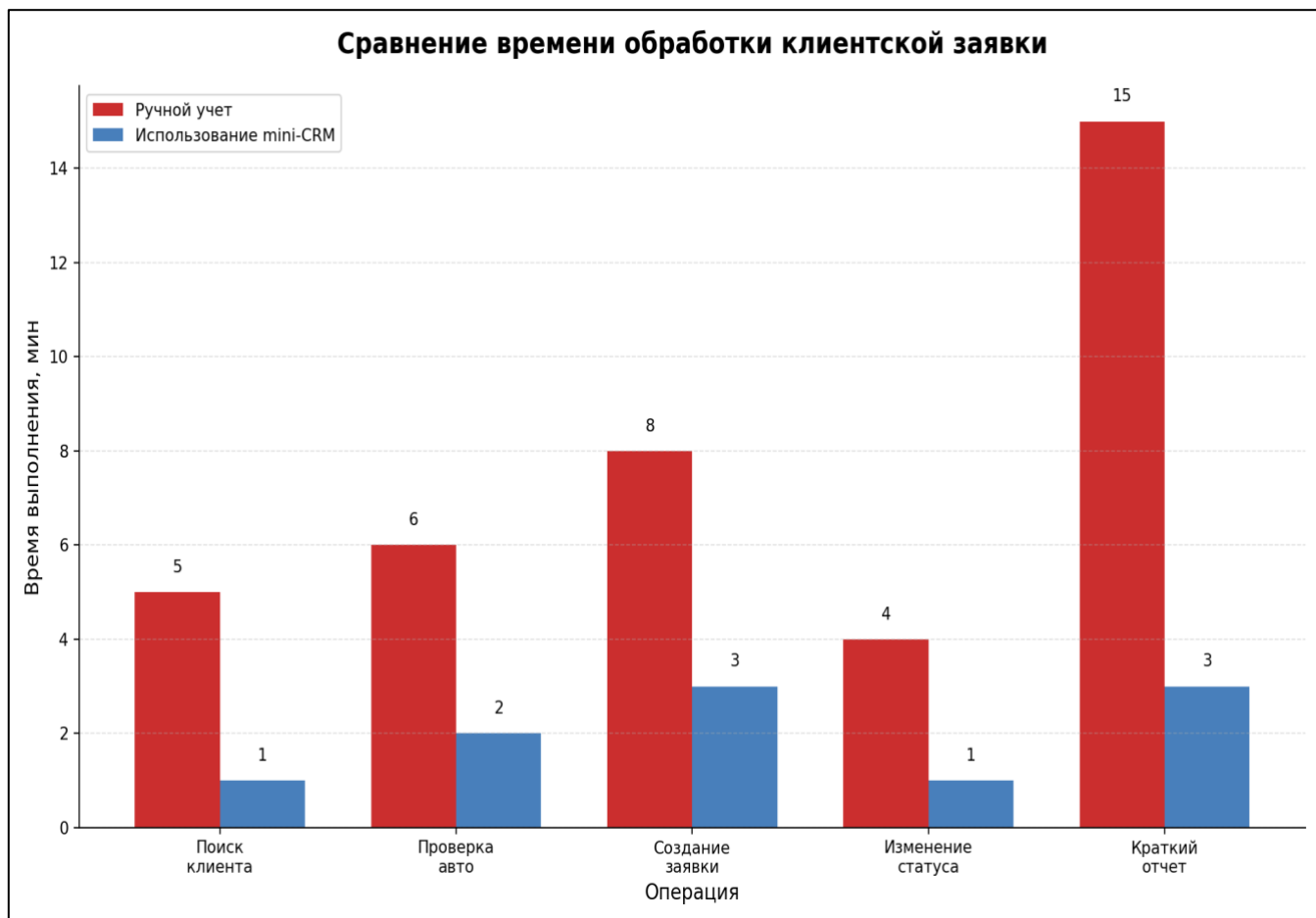


Рисунок 1.1.4.1 — Сравнение времени обработки клиентской заявки до и после внедрения системы

Таблица 1.1.4.2 — Расчетные данные для сравнения времени обработки клиентской заявки

Операция	Время при ручном учете, мин	Время при использовании мини-CRM, мин	Сокращение времени, мин
Поиск клиента	5	1	4
Проверка автомобиля	6	2	4
Создание заявки	8	3	5
Изменение статуса	4	1	3
Подготовка краткого отчета	15	3	12

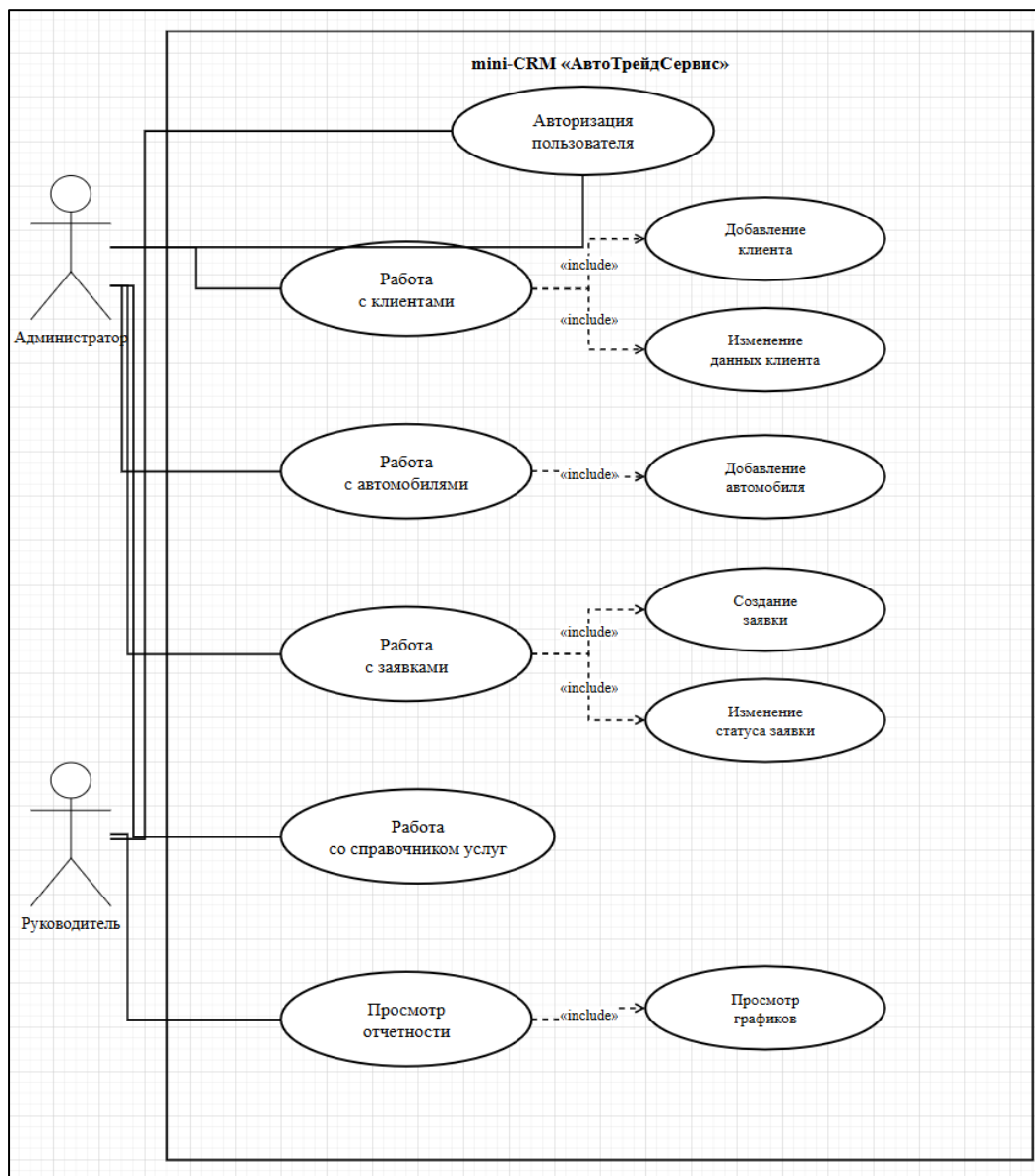


Рисунок 1.1.4.2 — Диаграмма вариантов использования приложения с правами обычного пользователя

1.2 Выбор технологий и средств разработки приложения

1.2.1 Выбор языка программирования и средств разработки

Выбор технологий разработки определяет, насколько быстро и надежно может быть создана мини-CRM «АвтоТрейдСервис». Для такого приложения важны не только возможности языка программирования, но и простота сопровождения, удобство работы с базой данных, наличие средств для разработки веб-интерфейса и построения графиков. Система должна быть компактной,

понятной по структуре и достаточной для реализации учета клиентов, автомобилей, заявок, услуг, сотрудников и отчетности.

В качестве основных вариантов языка программирования рассмотрены C++, JavaScript и Python. Эти языки широко применяются в программной разработке, но их пригодность для создаваемого веб-приложения различается. C++ обеспечивает высокую производительность, но требует больше времени на разработку прикладной веб-системы и менее удобен для быстрого создания интерфейсов и серверной логики. JavaScript активно используется в веб-разработке, особенно на стороне клиента, но для создания серверной части требует отдельного выбора среды выполнения и набора дополнительных библиотек. Python хорошо подходит для компактных веб-приложений, имеет простой синтаксис и развитую экосистему библиотек [10]. Сравнение языков программирования приведено в Таблице 1.2.1.1.

Таблица 1.2.1.1 — Сравнение языков программирования для разработки мини-CRM «АвтоТрейдСервис»

Язык программирования	Функциональность	Распространенность	Интеграция	Подходимость для продукта работы	Итоговая оценка
Python	5	5	5	5	20
JavaScript	4	5	4	4	17
C++	5	4	3	2	14

По результатам сравнения наибольшую итоговую оценку получает Python. Его преимущество для мини-CRM связано с простотой написания серверной части, хорошей читаемостью кода, большим количеством библиотек и удобной интеграцией с веб-фреймворками. Для проекта, где требуется быстро реализовать рабочее приложение с базой данных, формами, маршрутами и отчетами, Python выглядит наиболее рациональным выбором. Расчетные данные для визуального сравнения языков программирования приведены в Таблице 1.2.1.2.

Таблица 1.2.1.2 — Расчетные данные для графика сравнения характеристик языков программирования, данные — смоделированные

Критерий	Python	JavaScript	C++
Функциональность	5	4	5
Распространенность	5	5	4
Интеграция	5	4	3
Подходимость для продукта работы	5	4	2

Сравнение характеристик выбранных технологий разработки приведено на Рисунке 1.2.1.1.

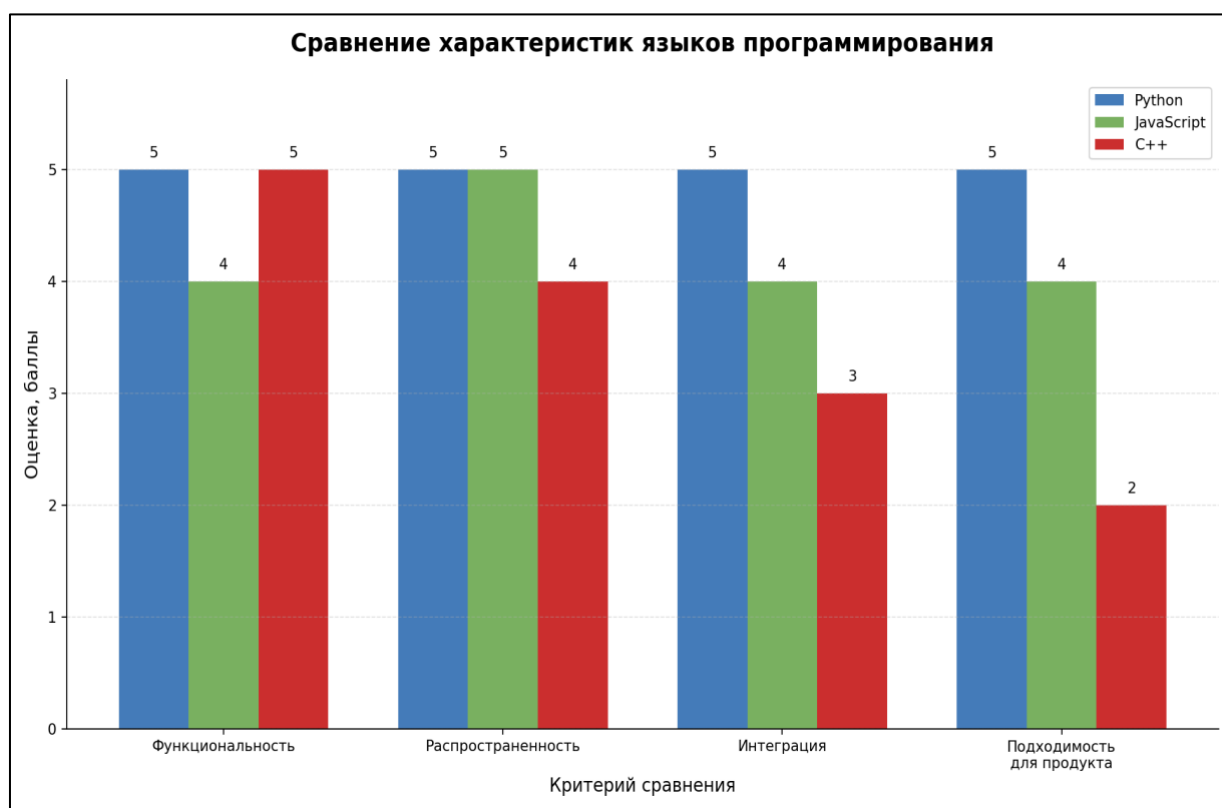


Рисунок 1.2.1.1 — Сравнение характеристик выбранных технологий разработки

Рисунок 1.2.1.1 показывает, что Python имеет наиболее равномерные и высокие оценки по всем критериям. JavaScript также хорошо подходит для веб-разработки, но в создаваемой системе он будет использоваться прежде всего на стороне интерфейса и визуализации. C++ сохраняет сильные позиции по производительности, но для мини-CRM его преимущества не являются определяющими. Система не выполняет сложных вычислений в реальном времени и не требует низкоуровневого управления памятью, поэтому высокая производительность C++ не компенсирует рост трудоемкости разработки.

Для серверной части выбран Flask. Это легковесный веб-фреймворк Python, который позволяет создавать маршруты, обрабатывать HTTP-запросы, работать с шаблонами и передавать данные между сервером и страницами интерфейса [4]. Для мини-CRM такой подход удобен, потому что структура приложения остается прозрачной. Основные маршруты можно связать с разделами системы:

- главная панель;
- клиенты;
- автомобили;
- заявки;
- услуги;
- сотрудники;
- отчеты.

Flask также хорошо подходит для поэтапной разработки. Сначала можно создать базовые маршруты и страницы, затем подключить базу данных, после этого добавить формы, обработку действий пользователя и визуализацию. Это снижает риск ошибок, потому что каждая часть системы проверяется отдельно. Для небольшого проекта такое свойство особенно важно: приложение должно быть понятным не только при запуске, но и при дальнейшем изменении.

Клиентская часть разрабатывается с использованием HTML, CSS и Bootstrap. HTML отвечает за структуру страниц, CSS — за внешний вид, Bootstrap — за готовые компоненты интерфейса:

- таблицы;
- кнопки;
- формы;
- карточки;
- навигационные панели;
- модальные окна [5].

За счет этого можно создать аккуратный интерфейс без избыточного ручного написания стилей. Для сотрудников автосервиса это важно, потому что интерфейс должен быть простым и быстро читаемым.

Для отображения данных применяется шаблонизатор Jinja2, который используется вместе с Flask. Он позволяет вставлять данные из Python-кода в HTML-страницы, формировать таблицы заявок, выводить карточки клиентов и передавать значения в графики. Например, серверная часть получает из базы количество заявок по статусам, а шаблон отображает эти данные в панели управления. Такая связка делает приложение цельным: данные не просто хранятся, а сразу используются в интерфейсе.

Для визуализации выбран Chart.js. Эта библиотека применяется для построения диаграмм и графиков в браузере [6]. В мини-CRM она будет использоваться для отображения распределения заявок по статусам, динамики обращений и структуры услуг. Для руководителя это удобнее обычных таблиц: график позволяет быстрее увидеть изменение нагрузки и оценить результаты работы сервиса.

Использование выбранных средств разработки позволяет построить приложение с разделением на несколько уровней:

- уровень представления — HTML-шаблоны, Bootstrap-компоненты, таблицы, формы и графики;
- уровень обработки запросов — маршруты Flask, которые принимают действия пользователя и возвращают нужные страницы;
- уровень данных — SQLite-база, где хранятся клиенты, автомобили, заявки, услуги и сотрудники;
- уровень визуализации — Chart.js, получающий подготовленные значения и отображающий их в виде диаграмм.

Такое разделение упрощает сопровождение системы. Если нужно изменить внешний вид таблицы, не требуется переписывать базу данных. Если меняется структура отчета, можно доработать маршрут и шаблон, не затрагивая остальные разделы. Этот подход соответствует общим принципам разработки информационных систем, где важны разделение функций, надежность хранения и понятная организация программных компонентов [11, 12].

Выбор Python и Flask также связан с доступностью документации и большим числом примеров. Это снижает риск технических затруднений при реализации. В случае возникновения ошибки проще найти решение, проверить работу маршрута, обработку формы или соединение с базой данных. Для создаваемой мини-CRM это значимое преимущество, потому что проект должен быть не только разработан, но и протестирован.

В итоге для разработки мини-CRM «АвтоТрейдСервис» выбран следующий набор средств:

- Python как основной язык программирования;
- Flask как серверный веб-фреймворк;
- HTML/CSS и Bootstrap для интерфейса;
- Jinja2 для шаблонов;
- Chart.js для визуализации данных.

Такой стек обеспечивает достаточную функциональность, быструю разработку, понятную архитектуру и хорошую пригодность для создания компактной CRM-системы автосервисной компании.

1.2.2 Выбор СУБД

Для мини-CRM «АвтоТрейдСервис» база данных является центральным элементом системы. В ней должны храниться сведения о клиентах, автомобилях, заявках, услугах, сотрудниках и статусах выполнения работ. От правильного выбора СУБД зависит надежность хранения данных, удобство разработки, скорость запуска приложения и сложность дальнейшего сопровождения.

Система управления базами данных должна соответствовать назначению разрабатываемого приложения. Мини-CRM предназначена для небольшой автосервисной компании, поэтому к ней не предъявляются требования промышленной нагрузки, распределенной обработки данных или сложного администрирования. Приоритетными становятся другие параметры:

- простая установка;
- понятная структура;
- достаточная надежность;

- поддержка SQL;
- удобная интеграция с Python-приложением.

Для выбора СУБД рассмотрены три варианта: PostgreSQL, MS SQL Server и SQLite. Все они относятся к распространенным реляционным системам управления базами данных и позволяют хранить структурированные сведения в таблицах. При этом их назначение и уровень сложности различаются.

PostgreSQL — мощная объектно-реляционная СУБД с открытым исходным кодом. Она подходит для крупных веб-сервисов, корпоративных систем и проектов, где важны масштабируемость, расширенные типы данных, сложные запросы и высокая надежность. PostgreSQL активно используется в информационных системах, где база данных работает как отдельный серверный компонент. Для мини-CRM такая СУБД может быть применена, но ее возможности в рассматриваемом проекте будут использованы лишь частично.

MS SQL Server — промышленная СУБД компании Microsoft. Она хорошо подходит для корпоративной среды, особенно если организация уже использует продукты Microsoft. Эта система обладает развитыми средствами администрирования, безопасности, резервного копирования и аналитики. При этом для небольшого приложения она может оказаться избыточной. Использование MS SQL Server обычно требует более сложной настройки и большего внимания к инфраструктуре.

SQLite — компактная встраиваемая реляционная СУБД. Ее ключевая особенность заключается в том, что база данных хранится в одном файле и не требует отдельного серверного процесса. Для небольших веб-приложений, прототипов, локальных систем учета и приложений с умеренной нагрузкой это значительное преимущество. SQLite поддерживает SQL, транзакции и основные механизмы работы с таблицами, но при этом остается простой в развертывании [9]. Сравнение СУБД для разработки мини-CRM «АвтоТрейдСервис» приведено в Таблице 1.2.2.1.

Таблица 12.2.1 — Сравнение СУБД для разработки мини-CRM
«АвтоТрейдСервис»

СУБД	Простота внедрения	Функциональность	Интеграция с Python	Подходимость для продукта работы	Итоговая оценка
SQLite	5	4	5	5	19
PostgreSQL	3	5	5	4	17
MS SQL Server	2	5	4	3	14

По результатам сравнения наибольшую итоговую оценку получает SQLite. Это не означает, что она превосходит PostgreSQL или MS SQL Server во всех задачах. Ее преимущество проявляется именно в условиях создаваемого приложения:

- компактная система;
- умеренный объем данных;
- отсутствие необходимости в отдельном сервере базы данных;
- быстрый запуск.

Для мини-CRM «АвтоТрейдСервис» SQLite закрывает основные требования. В базе данных будут храниться связанные таблицы, например clients, cars, service_requests, services, employees и users. Каждая таблица имеет первичный ключ, а связи между таблицами могут быть организованы через внешние ключи. Такой подход достаточен для учета клиентов, автомобилей и заявок. При правильном проектировании база сохранит целостность данных и позволит формировать отчеты.

Важно учитывать, что выбор SQLite снижает сложность развертывания системы. Для работы приложения не нужно устанавливать отдельный сервер БД, настраивать учетные записи администратора базы, открывать сетевые порты или управлять службой СУБД. Файл базы данных создается вместе с приложением и подключается из Python-кода. Это удобно для небольшого проекта и упрощает проверку работоспособности системы.

PostgreSQL имеет значительные преимущества при увеличении нагрузки. Если мини-CRM в дальнейшем будет развиваться до многопользовательской

системы с удаленным доступом, большим числом филиалов и сложной отчетностью, переход на PostgreSQL может стать оправданным. Она лучше подходит для одновременной работы большого числа пользователей и более гибкого управления правами доступа. В рассматриваемой версии приложения такие возможности не являются обязательными.

MS SQL Server также обладает сильными сторонами, но в данном случае они не определяют выбор. Эта СУБД рациональна для корпоративной инфраструктуры, где уже используются решения Microsoft, требуется интеграция с доменной средой, сложные регламенты безопасности и централизованное администрирование. Для мини-CRM небольшой автосервисной компании такой уровень инфраструктуры усложняет разработку без значимого выигрыша.

Критерии выбора СУБД объяснены несколькими параметрами:

- простота запуска. Система должна быстро разворачиваться и запускаться без сложной настройки серверного окружения;
- достаточность функциональности. СУБД должна поддерживать таблицы, связи, запросы, фильтрацию и хранение справочных данных;
- совместимость с Python. Работа с базой должна быть удобной из серверной части Flask-приложения;
- надежность хранения. Данные клиентов, автомобилей и заявок должны сохраняться в структурированном виде;
- возможность дальнейшего развития. При необходимости структура базы должна допускать добавление новых таблиц и полей.

SQLite соответствует этим требованиям. Она встроена в стандартные возможности Python, что делает подключение базы простым и не требует установки дополнительных серверных компонентов. Для Flask-приложения это удобно:

- обработчик маршрута может открыть соединение с файлом базы;
- выполнить SQL-запрос;
- получить результат;
- передать результат в HTML-шаблон.

С точки зрения проектирования данных SQLite позволяет реализовать все ключевые сущности системы. Таблица клиентов будет хранить ФИО, телефон, электронную почту и примечание. Таблица автомобилей — марку, модель, год выпуска, государственный номер и ссылку на клиента. Таблица заявок — дату создания, описание неисправности, статус, автомобиль, сотрудника и итоговую стоимость. Таблица услуг — наименование и цену. При необходимости может использоваться связующая таблица между заявками и услугами, если одна заявка включает несколько работ.

Такая структура хорошо подходит для реляционной модели. Данные не дублируются в каждой заявке, а связываются через идентификаторы. Например, сведения о клиенте хранятся один раз, а все его автомобили и заявки ссылаются на эту запись. Это снижает риск расхождений и упрощает обновление информации. Подобный подход является базовым для проектирования надежных баз данных [3].

SQLite также подходит для отчетности. С помощью SQL-запросов можно получить количество заявок по статусам, сумму работ за период, перечень популярных услуг, число обращений по каждому клиенту или загрузку сотрудников. Эти данные затем передаются в шаблоны и отображаются в таблицах или графиках. Для руководителя автосервиса такой набор отчетов достаточен, чтобы оценивать текущую деятельность.

Выбранная СУБД должна быть согласована с архитектурой приложения. Поскольку серверная часть создается на Flask, база данных будет использоваться через Python-модуль sqlite3 или через вспомогательный слой доступа к данным. Второй вариант удобнее при расширении проекта, но для компактного приложения достаточно прямой работы с SQL-запросами при аккуратном разделении функций. Главное — не смешивать все операции в одном месте, а выделять отдельные функции для получения, добавления, изменения и удаления записей.

Ограничения SQLite также нужно учитывать. Эта СУБД не предназначена для больших распределенных систем и высокой одновременной записи

множества пользователей. Если приложение будет использоваться большим числом сотрудников одновременно или хранить значительный объем данных за много лет, может потребоваться переход на более мощную СУБД. Но для текущего объема задач мини-CRM эти ограничения не являются критическими.

Важным фактором является переносимость. Файл SQLite можно легко скопировать вместе с приложением, использовать для резервного сохранения или перенести на другой компьютер. Для небольшого предприятия это удобно: не требуется отдельный специалист по администрированию БД, а процесс развертывания остается понятным. При этом базовые требования к хранению данных сохраняются.

В результате для разработки мини-CRM «АвтоТрейдСервис» выбирается SQLite. Она обеспечивает лучший баланс между простотой, достаточной функциональностью и пригодностью для небольшого веб-приложения. PostgreSQL и MS SQL Server сохраняют значение как более мощные альтернативы, но для рассматриваемой системы их применение усложнило бы проект без необходимости. SQLite позволяет сосредоточиться на главной задаче — реализации учета клиентов, автомобилей, заявок и отчетности в понятной и рабочей мини-CRM.

1.2.3 Выбор среды разработки

Для разработки мини-CRM «АвтоТрейдСервис» необходимо выбрать среду, которая обеспечит удобную работу с Python-кодом, HTML-шаблонами, CSS-файлами, JavaScript и структурой Flask-приложения. Среда разработки должна быть простой в установке, достаточно легкой, поддерживать расширения и позволять быстро находить ошибки в коде.

В качестве вариантов рассмотрены Visual Studio Code, PyCharm и Sublime Text. Эти инструменты широко применяются разработчиками и подходят для работы с веб-проектами. При этом их возможности и удобство для создаваемого приложения различаются. Сравнение сред разработки приведено в Таблице 1.2.3.1.

Таблица 1.2.3.1 — Сравнение сред разработки для мини-CRM
«АвтоТрейдСервис»

Среда разработки	Поддержка Python	Поддержка веб-файлов	Удобство настройки	Расширяемость	Итоговая оценка
Visual Studio Code	5	5	5	5	20
PyCharm	5	4	3	4	16
Sublime Text	3	4	5	3	15

По результатам сравнения наиболее подходящей средой разработки является Visual Studio Code. Она сочетает легкость редактора и возможности полноценной среды разработки. Для проекта на Flask это особенно удобно. В одном рабочем окне можно редактировать:

- Python-файлы;
- HTML-шаблоны;
- CSS-стили;
- JavaScript-код;
- файлы базы данных.

Visual Studio Code поддерживает расширения для Python, Flask, HTML, CSS, JavaScript, SQLite и Git. За счет этого разработчик может настроить рабочую среду под конкретный проект, не устанавливая лишние компоненты. Важное преимущество — встроенный терминал. Через него можно запускать Flask-приложение, устанавливать зависимости, создавать виртуальное окружение и выполнять команды проверки.

PyCharm также хорошо подходит для Python-разработки. Он имеет развитые средства анализа кода, отладки и управления проектом. Но для небольшого веб-приложения эта среда может быть слишком тяжелой. Кроме того, часть расширенных функций доступна в профессиональной версии. Для мини-CRM, где важны скорость работы и простота настройки, Visual Studio Code выглядит более удобным решением.

Sublime Text отличается высокой скоростью и простым интерфейсом. Он хорошо подходит для редактирования файлов, но требует дополнительной

настройки для полноценной работы с Python-проектом. В нем меньше встроенных средств для запуска, отладки и управления окружением. Поэтому Sublime Text можно рассматривать как легкий редактор, но не как основной инструмент разработки создаваемой системы.

Выбор Visual Studio Code связан с особенностями проекта. Мини-CRM «АвтоТрейдСервис» включает несколько типов файлов:

- серверный код;
- шаблоны страниц;
- стили;
- скрипты;
- файл базы данных.

Среда разработки должна одинаково удобно работать со всеми этими компонентами. Visual Studio Code позволяет открыть проект как единую папку, видеть структуру каталогов и быстро переходить между файлами.

Для Flask-приложения это имеет практическое значение. Например, маршрут в файле `app.py` может передавать данные в шаблон `requests.html`, а этот шаблон использует стили из папки `static`. Если среда разработки хорошо показывает структуру проекта, разработчик быстрее понимает связь между компонентами и меньше ошибается при изменении кода.

К основным преимуществам Visual Studio Code для мини-CRM относятся:

- Поддержка Python. Расширение Python обеспечивает подсветку синтаксиса, проверку ошибок, автодополнение и запуск кода;
- Удобство работы с Flask. Встроенный терминал позволяет запускать сервер разработки и проверять изменения без переключения между программами;
- Работа с веб-файлами. HTML, CSS и JavaScript поддерживаются напрямую, что важно для разработки интерфейса;
- Расширения для SQLite. Можно просматривать структуру базы данных и выполнять SQL-запросы прямо из среды;
- Простая настройка. Программа быстро устанавливается и не требует сложной подготовки рабочего места;

– Низкая нагрузка на компьютер. Среда достаточно легкая по сравнению с крупными IDE.

Для создаваемого приложения также важно, что Visual Studio Code хорошо подходит для пошаговой разработки. Сначала можно создать структуру проекта и главный файл приложения. Затем — добавить шаблоны, подключить базу данных, реализовать формы и отчеты. Каждый этап можно запускать и проверять через встроенный терминал. Такой порядок снижает риск накопления ошибок.

Среда разработки должна поддерживать работу с виртуальным окружением Python. Это необходимо для изоляции зависимостей проекта. Виртуальное окружение позволяет установить Flask и другие библиотеки только для данного приложения, не влияя на остальные проекты на компьютере. Visual Studio Code умеет обнаруживать такие окружения и использовать выбранный интерпретатор Python.

Еще одно преимущество — удобная навигация по коду. При разработке мини-CRM потребуется работать с маршрутами, функциями обработки форм, SQL-запросами и шаблонами. Быстрый поиск по файлам помогает находить нужный обработчик, название таблицы или переменную. Для небольшого проекта это кажется мелочью, но на практике ускоряет разработку и отладку.

Visual Studio Code также удобен для работы с Bootstrap и Chart.js. HTML-страницы можно редактировать с подсветкой синтаксиса, а JavaScript-код для графиков — проверять в том же проекте. Это особенно важно для разделов dashboard и отчетности, где данные из Flask передаются в шаблон и затем используются для построения диаграмм [5, 6].

С точки зрения сопровождения проекта выбранная среда также подходит. Код мини-CRM должен оставаться понятным:

- отдельные папки для шаблонов, статических файлов и базы данных;
- отдельные функции для работы с таблицами;
- аккуратные маршруты для основных страниц.

Visual Studio Code помогает поддерживать такую структуру за счет дерева проекта, поиска и расширений форматирования.

В результате основной средой разработки для мини-CRM «АвтоТрейдСервис» выбран Visual Studio Code. Он обеспечивает лучший баланс между удобством, скоростью, поддержкой Python и веб-технологий. PyCharm остается сильной альтернативой для более крупных Python-проектов, а Sublime Text — быстрым редактором кода. Но для создаваемого Flask-приложения Visual Studio Code наиболее рационален, поскольку позволяет быстро разрабатывать, запускать и проверять все компоненты системы в одном рабочем пространстве.

1.3 Проектирование информационной системы

1.3.1 Архитектура приложения

Архитектура мини-CRM «АвтоТрейдСервис» строится по принципу веб-приложения с разделением на клиентскую часть, серверную часть и базу данных. Такой подход позволяет отделить пользовательский интерфейс от обработки запросов и хранения данных. Пользователь работает с системой через браузер, серверная часть на Flask принимает запросы, выполняет проверку данных, обращается к SQLite и возвращает пользователю готовую HTML-страницу. Основные архитектурные компоненты приложения приведены в Таблице 1.3.1.1.

Таблица 1.3.1.1 — Основные компоненты архитектуры мини-CRM «АвтоТрейдСервис»

Компонент	Назначение	Используемые средства
Клиентская часть	Отображение страниц, форм, таблиц и графиков	HTML, CSS, Bootstrap, Chart.js
Серверная часть	Обработка маршрутов, форм, авторизации и бизнес-операций	Python, Flask
Шаблоны	Формирование страниц на основе данных приложения	Jinja2
База данных	Хранение клиентов, автомобилей, заявок, услуг и сотрудников	SQLite
Отчетный модуль	Подготовка сводных данных и визуальных показателей	SQL-запросы, Chart.js

Мини-CRM разворачивается на рабочем компьютере или локальном сервере автосервисной компании. Пользовательский доступ выполняется через браузер.

Сервер Flask обрабатывает запросы и взаимодействует с файлом SQLite. Такая схема подходит для компактного приложения, поскольку снижает требования к инфраструктуре и не требует отдельного сервера БД [4, 9]. Диаграмма развертывания приложения приведена на Рисунке 1.3.1.2.

Архитектура приложения приведена на Рисунке 1.3.1.1.

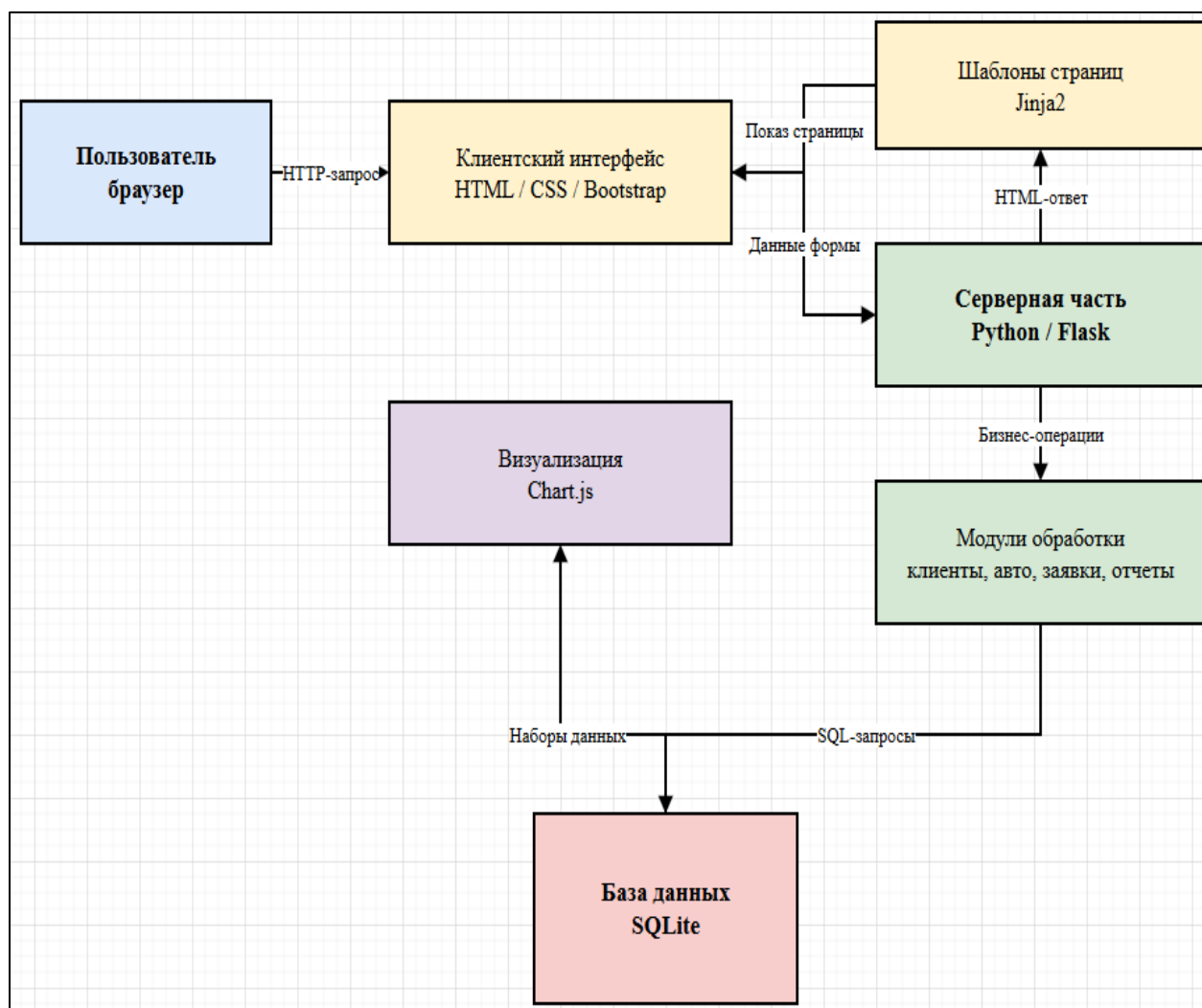


Рисунок 1.3.1.1 — Архитектура приложения

При создании заявки пользователь проходит типовую последовательность действий:

- открывает страницу заявок;
- заполняет форму;
- отправляет данные;
- сервер выполняет проверку и сохраняет запись в базе.

Затем приложение возвращает обновленный список заявок. Такая последовательность отражает взаимодействие представления, серверной обработки и базы данных. Диаграмма последовательности приведена на Рисунке 1.3.1.3.

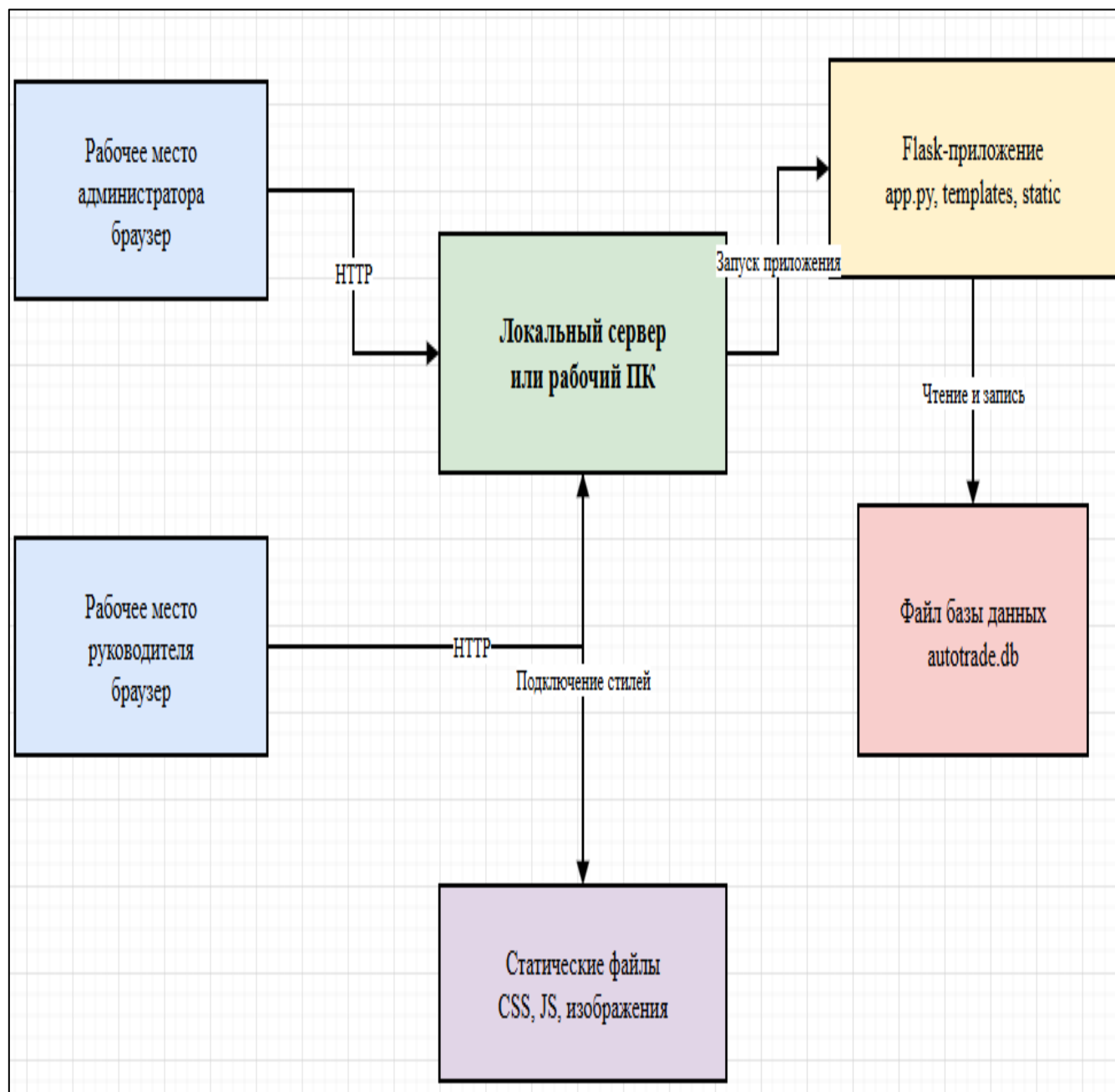


Рисунок 1.3.1.2 — Диаграмма развертывания приложения

Классовая структура приложения отражает основные сущности предметной области. Для мини-CRM ключевыми являются пользователь, клиент, автомобиль, заявка, услуга и сотрудник. Связи между ними определяют будущую структуру

базы данных и маршрутов приложения. Например, клиент может иметь несколько автомобилей, а автомобиль — несколько заявок.

Диаграмма классов приведена на Рисунке 1.3.1.4.

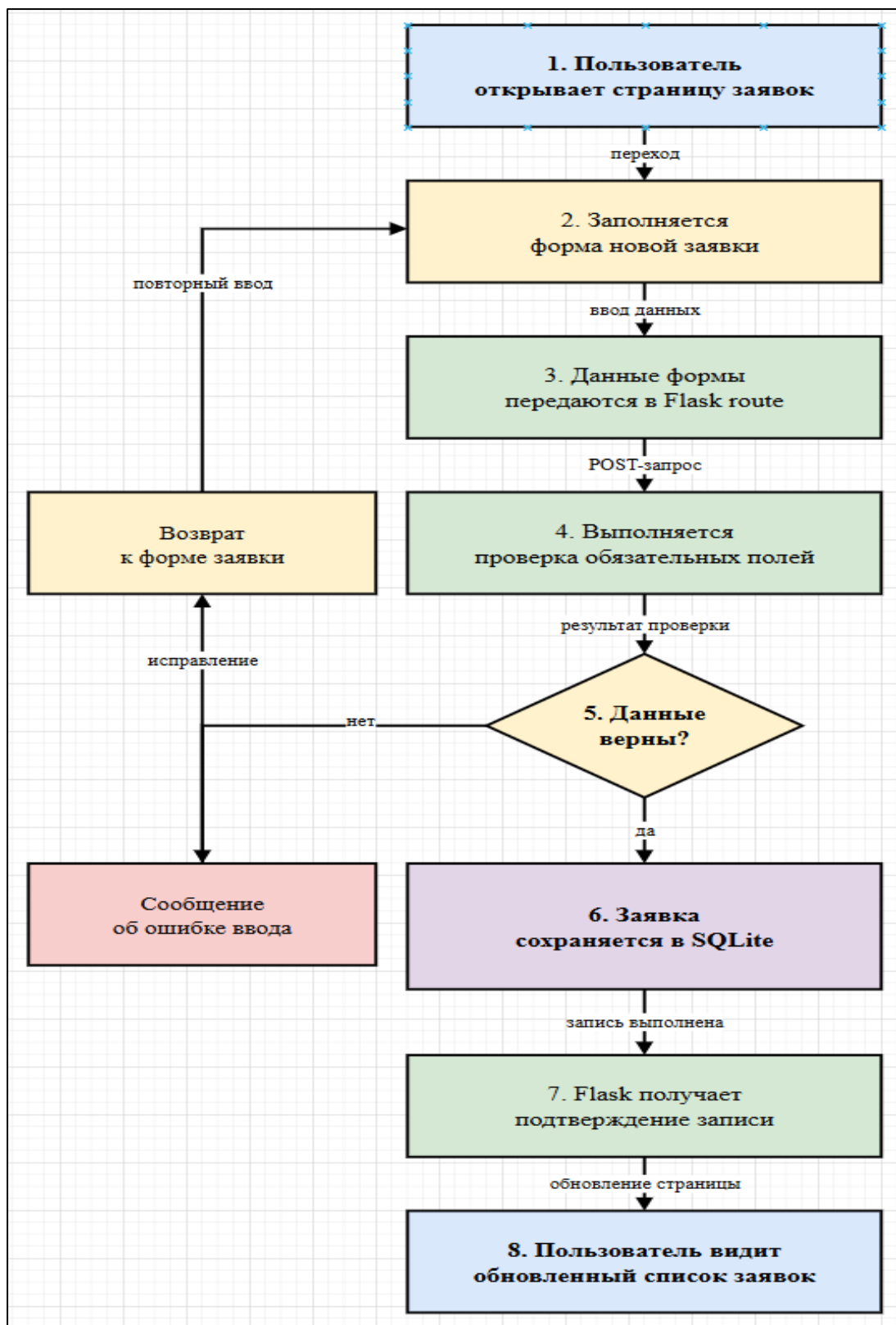


Рисунок 1.3.1.3 — Диаграмма последовательности создания заявки

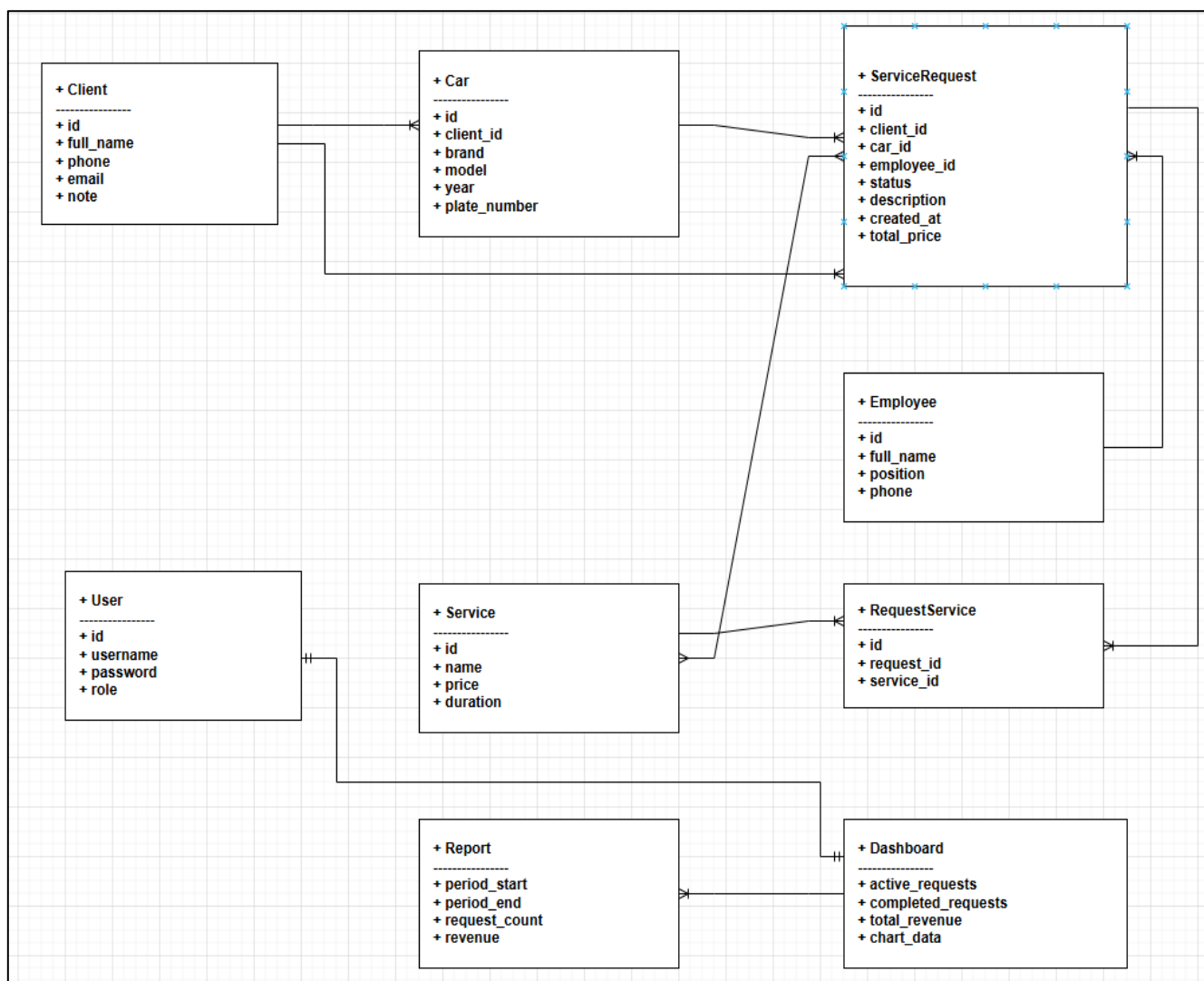


Рисунок 1.3.1.4 — Диаграмма классов мини-CRM «АвтоТрейдСервис»

Работа пользователя в системе начинается с авторизации. После входа он выбирает нужный раздел:

- клиенты;
- автомобили;
- заявки;
- услуги;
- отчеты.

При создании заявки система проверяет заполнение обязательных полей, сохраняет данные и обновляет список. Если данные заполнены неверно,

пользователь получает сообщение об ошибке и возвращается к форме. Диаграмма деятельности приведена на Рисунке 1.3.1.5.

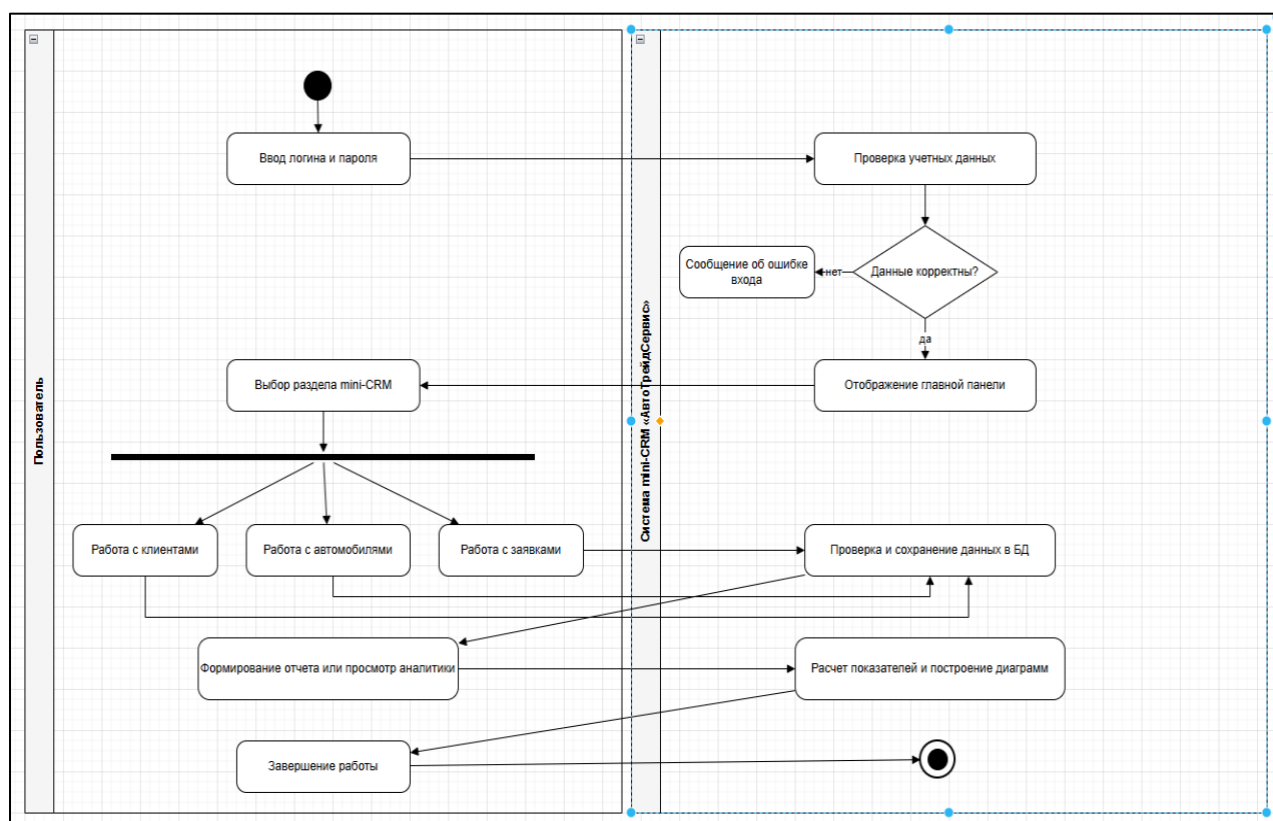


Рисунок 1.3.1.5 — Диаграмма деятельности пользователя

Разработанная архитектура обеспечивает достаточную простоту и целостность системы. Клиентская часть отвечает за ввод и просмотр данных, Flask выполняет обработку запросов, SQLite хранит информацию, а Chart.js используется для вывода аналитики. Такое построение позволяет реализовать мини-CRM без избыточной инфраструктуры и при этом сохранить все основные функции информационной системы:

- учет;
- обработку;
- хранение;
- визуальное представление данных [3, 11].

1.3.2 Проектирование базы данных

Проектирование базы данных мини-CRM «АвтоТрейдСервис» выполняется после определения архитектуры приложения. База данных должна хранить сведения о клиентах, автомобилях, заявках, услугах, сотрудниках и пользователях системы. Для разрабатываемого приложения выбрана реляционная модель, так как данные имеют устойчивую табличную структуру и хорошо описываются через связи «один ко многим» и «многие ко многим» [3].

Основная задача проектирования — исключить дублирование данных и обеспечить правильную связь между объектами. Например, клиент может иметь несколько автомобилей, каждый автомобиль может быть связан с несколькими заявками, а одна заявка может включать несколько услуг. Если эти данные хранить в одной общей таблице, быстро появятся повторы, ошибки и сложности при изменении сведений. Поэтому база проектируется как набор связанных таблиц. Перечень таблиц базы данных приведен в Таблице 1.3.2.1.

Таблица 1.3.2.1 — Перечень таблиц базы данных

Таблица	Назначение	Основные поля
users	Хранение учетных записей пользователей	id, username, password, role
employees	Хранение данных сотрудников	id, full_name, position, phone
clients	Хранение клиентской базы	id, full_name, phone, email, note
cars	Хранение автомобилей клиентов	id, client_id, brand, model, year, plate_number
services	Хранение справочника услуг	id, name, price, duration
service_requests	Хранение заявок на обслуживание	id, client_id, car_id, employee_id, status, description, created_at, total_price
request_services	Связь заявок и услуг	id, request_id, service_id

Из Таблицы 1.3.2.1 видно, что структура базы данных покрывает основные процессы автосервисной компании. Таблица clients отвечает за учет клиентов, cars — за автомобили, service_requests — за заявки, services — за перечень работ. Связующая таблица request_services нужна для ситуации, когда одна заявка включает несколько услуг. Например, в одном обращении клиент может заказать диагностику, замену масла и проверку тормозной системы.

Ниже приводятся атрибуты таблиц БД CRM.

Таблица 1.3.2.2 — Атрибуты таблицы users

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор учетной записи	PK	1
username	TEXT	Логин для входа в систему		admin
password	TEXT	Пароль пользователя		admin123
role	TEXT	Роль пользователя в системе		Администратор

Таблица 1.3.2.3 — Атрибуты таблицы employees

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор сотрудника	PK	1
full_name	TEXT	ФИО сотрудника		Иванов Сергей Петрович
position	TEXT	Должность сотрудника		Мастер-приемщик
phone	TEXT	Контактный телефон сотрудника		+7 900 111-22-33

Таблица 1.3.2.4 — Атрибуты таблицы clients

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор клиента	PK	1
full_name	TEXT	ФИО клиента		Петров Андрей Викторович
phone	TEXT	Контактный телефон клиента		+7 900 222-33-44
email	TEXT	Электронная почта клиента		petrov@mail.ru
note	TEXT	Примечание по клиенту		Постоянный клиент

Таблица 1.3.2.5 — Атрибуты таблицы cars

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор автомобиля	PK	1
client_id	INTEGER	Идентификатор владельца автомобиля	FK	1
brand	TEXT	Марка автомобиля		Toyota
model	TEXT	Модель автомобиля		Camry
year	INTEGER	Год выпуска автомобиля		2018
plate_number	TEXT	Государственный номер автомобиля		A123BC77

Таблица 1.3.2.6 — Атрибуты таблицы services

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор услуги	PK	1
name	TEXT	Наименование услуги		Замена масла
price	REAL	Стоимость услуги		2500
duration	INTEGER	Плановая длительность выполнения, мин		60

Таблица 1.3.2.7— Атрибуты таблицы service_requests

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор заявки	PK	1
client_id	INTEGER	Идентификатор клиента	FK	1
car_id	INTEGER	Идентификатор автомобиля	FK	1
employee_id	INTEGER	Идентификатор ответственного сотрудника	FK	2
status	TEXT	Статус выполнения заявки		В работе
description	TEXT	Описание проблемы или работ		Диагностика двигателя
created_at	TEXT	Дата создания заявки		2026-05-09
total_price	REAL	Итоговая стоимость заявки		7500

Таблица 1.3.2.8— Атрибуты таблицы request_services

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор записи связи	PK	1
request_id	INTEGER	Идентификатор заявки	FK	1
service_id	INTEGER	Идентификатор услуги	FK	3

При проектировании используются первичные и внешние ключи. Первичный ключ однозначно определяет запись в таблице. Внешний ключ связывает таблицы между собой. Например, поле client_id в таблице cars указывает, какому клиенту принадлежит автомобиль. Это позволяет избежать повторного хранения данных клиента в каждой записи автомобиля. Логическая модель базы данных приведена на Рисунке 1.3.2.1.

На Рисунке 1.3.2.1 показано, что центральной таблицей выступает service_requests. Она связывает клиента, автомобиль, сотрудника и набор услуг. Такая структура отражает основной процесс работы автосервиса:

- клиент обращается в сервис;
- автомобиль принимается на обслуживание;
- сотрудник выполняет работу;
- заявка получает статус и итоговую стоимость.

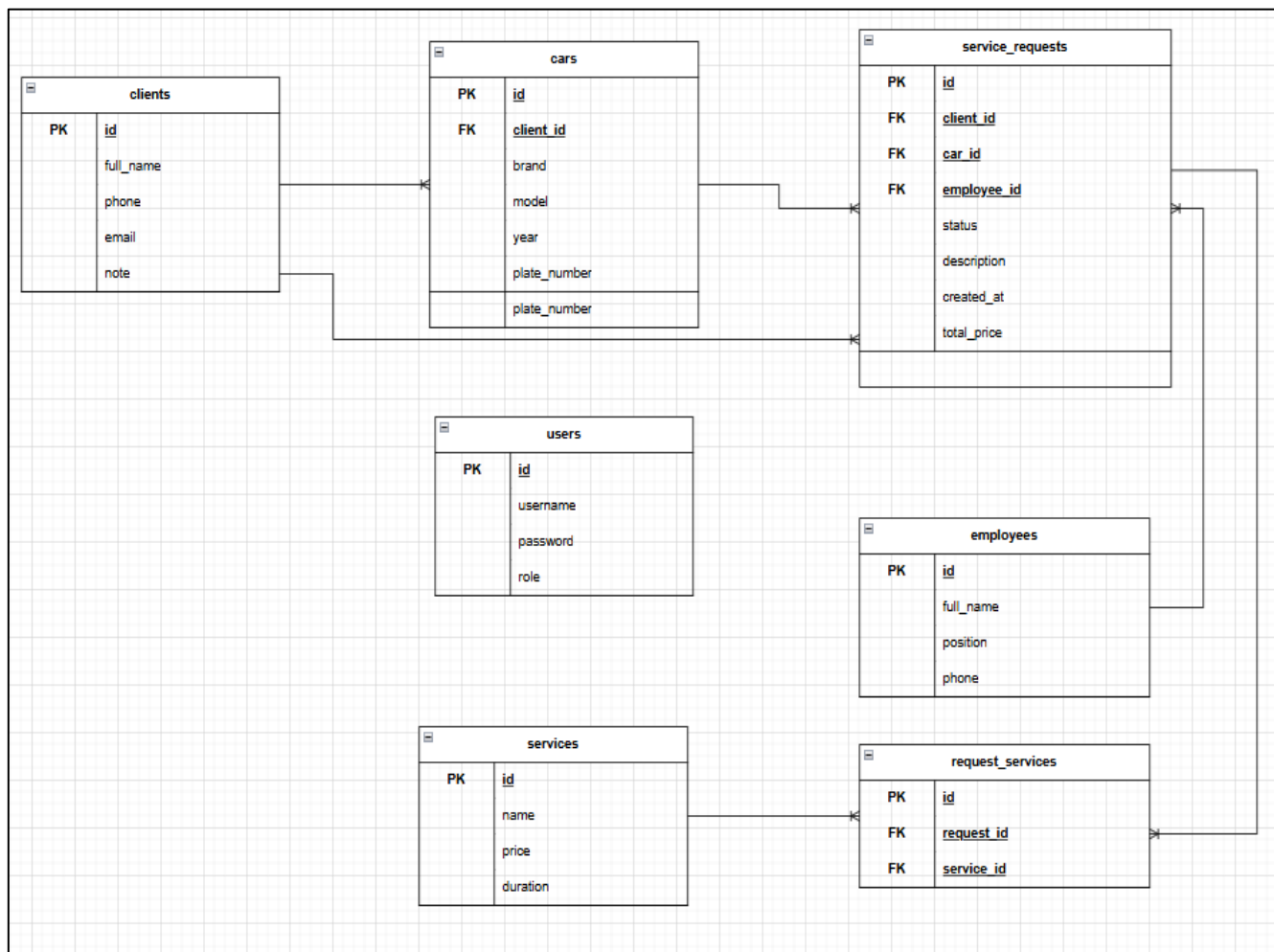


Рисунок 1.3.2.1 — Логическая модель базы данных

Физическая модель базы данных приведена на Рисунке 1.3.2.2.

Физическая модель уточняет типы данных и ограничения. Для SQLite применяются типы INTEGER, TEXT и REAL. Поля id используются как первичные ключи. Поля client_id, car_id, employee_id, request_id и service_id

работают как внешние ключи. Такой набор типов достаточен для хранения всех данных мини-CRM [9].

При проектировании базы данных учитываются требования целостности. Заявка должна быть связана с существующим клиентом и автомобилем. Услуга должна существовать в справочнике до включения в заявку. Это снижает риск появления «пустых» записей, которые невозможно использовать в отчетах.

Важным элементом является таблица request_services. Она решает задачу связи «многие ко многим» между заявками и услугами. Одна заявка может включать несколько услуг, и одна услуга может встречаться во многих заявках. Без связующей таблицы пришлось бы либо хранить несколько услуг в одном текстовом поле, либо создавать отдельные повторяющиеся поля, что нарушило бы структуру базы.

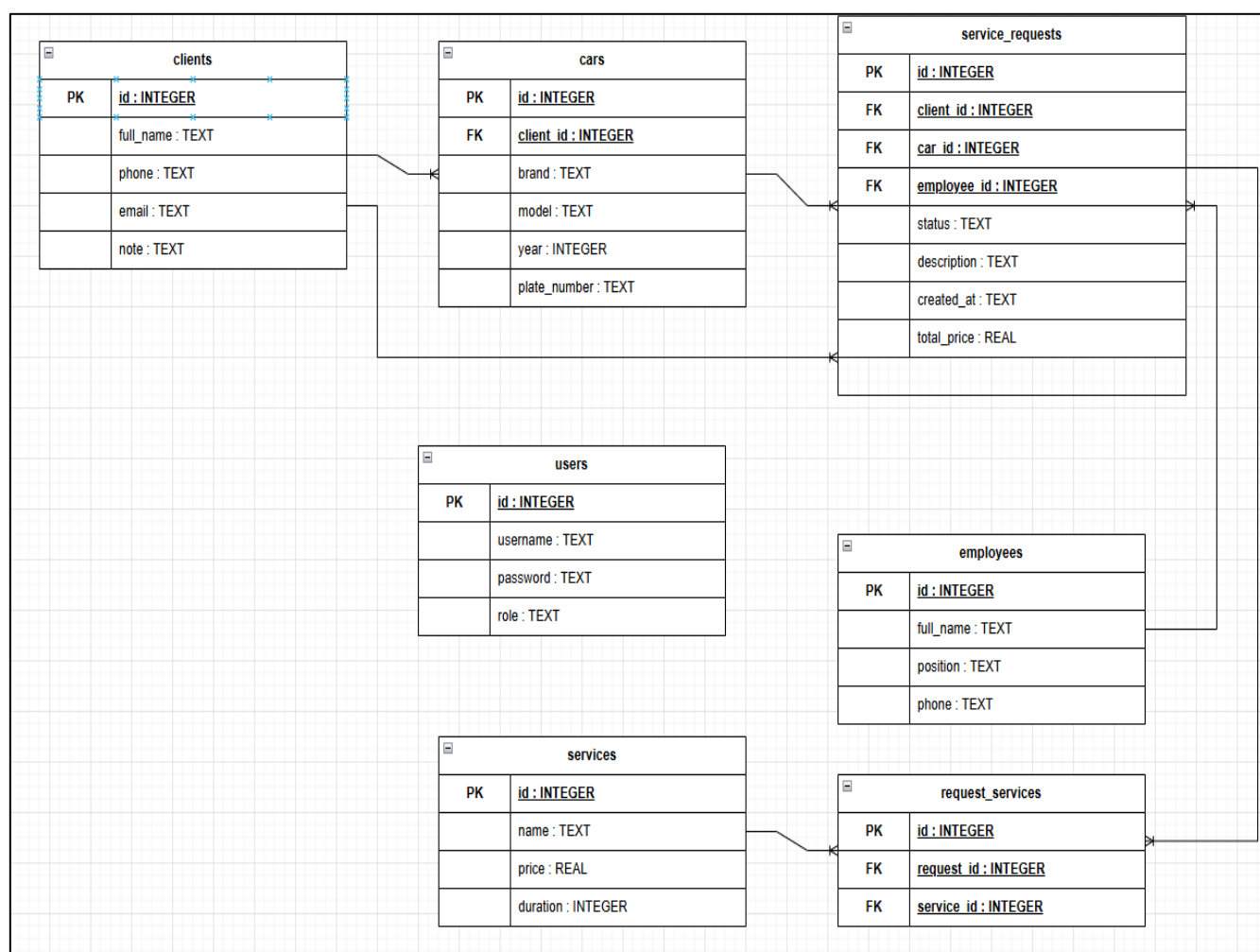


Рисунок 1.3.2.2 — Физическая модель базы данных

Выбранная структура также позволяет строить отчеты. Например, количество заявок по статусам можно получить из service_requests. Популярность услуг определяется через связь request_services и services. Данные по клиенту и автомобилю извлекаются через объединение таблиц clients, cars и service_requests. За счет этого база становится не только хранилищем, но и источником аналитики для панели управления.

В результате спроектированная база данных соответствует назначению мини-CRM «АвтоТрейдСервис». Она поддерживает учет клиентов, автомобилей, заявок, услуг и сотрудников, обеспечивает связи между сущностями и позволяет формировать отчетные показатели. Такая структура достаточна для реализации основных функций приложения и дальнейшего расширения системы при необходимости.

1.3.3 Разработка приложения

Разработка мини-CRM «АвтоТрейдСервис» выполняется после проектирования архитектуры и базы данных. На этом этапе создается рабочая структура веб-приложения, реализуются таблицы базы данных, маршруты Flask, HTML-шаблоны, интерфейсные формы и отчетная панель. Основная задача раздела — показать, каким образом спроектированная информационная система превращается в программный продукт. Разрабатываемое приложение имеет файловую структуру, приведенную в Таблице 1.3.3.1.

Таблица 1.3.3.1 — Основные файлы мини-CRM

Файл или каталог	Назначение
app.py	Основной файл Flask-приложения
autotrade.db	Файл базы данных SQLite
templates/base.html	Базовый шаблон интерфейса
templates/dashboard.html	Главная панель управления
templates/clients.html	Страница работы с клиентами

Файл или каталог	Назначение
templates/cars.html	Страница работы с автомобилями
templates/requests.html	Страница работы с заявками
templates/reports.html	Страница отчетности
static/style.css	Пользовательские стили интерфейса

Сначала создается база данных SQLite. Она содержит таблицы users, employees, clients, cars, services, service_requests и request_services. Эти таблицы соответствуют физической модели, описанной в Разделе 1.3.2. Создание базы выполняется при первом запуске приложения.

После запуска приложение автоматически создает файл autotrade.db и наполняет справочники начальными данными. Это упрощает проверку работы системы. Сразу доступны:

- сотрудники;
- услуги;
- клиенты;
- автомобили.

Пользователь может открыть главную панель, добавить клиента, зарегистрировать автомобиль, создать заявку и посмотреть отчетные графики. Основные маршруты приложения приведены в Таблице 1.3.3.2.

Таблица 1.3.3.2 — Основные маршруты приложения CRM

Маршрут	Метод	Назначение
/	GET	Отображение панели управления
/clients	GET, POST	Просмотр и добавление клиентов
/cars	GET, POST	Просмотр и добавление автомобилей
/requests	GET, POST	Просмотр и создание заявок
/reports	GET	Отображение отчетов и графиков

В разработанном приложении маршрут / отвечает за панель управления. На этой странице выводятся четыре основных показателя:

- количество клиентов;
- количество автомобилей;
- количество заявок;
- суммарная стоимость работ.

Кроме того, на панели отображается диаграмма заявок по статусам. Такой экран удобен для руководителя, потому что дает краткое состояние работы автосервиса без перехода в отдельные таблицы.

Маршрут `/clients` реализует работу с клиентами. При GET-запросе система показывает список клиентов. При POST-запросе данные из формы сохраняются в таблицу `clients`. Аналогично устроен маршрут `/cars`, где автомобиль связывается с владельцем через поле `client_id`. Это обеспечивает сохранение отношения между клиентом и автомобилем.

Маршрут `/requests` является центральным для системы, так как через него создаются заявки на обслуживание. При добавлении заявки пользователь выбирает клиента, автомобиль, сотрудника, статус и перечень услуг. Стоимость заявки рассчитывается на основе выбранных услуг. После сохранения запись попадает в таблицу `service_requests`, а выбранные услуги — в связующую таблицу `request_services`.

Отчетный маршрут `/reports` формирует данные для графиков. Первый график показывает распределение заявок по статусам. Второй график показывает популярность услуг. Для подготовки этих данных используются SQL-запросы с группировкой. Затем значения передаются в HTML-шаблон и отображаются с помощью Chart.js [6].

Реализация приложения соответствует выбранной архитектуре. Flask выполняет роль серверной части, SQLite хранит данные, HTML-шаблоны отвечают за интерфейс, Bootstrap обеспечивает оформление страниц, а Chart.js строит диаграммы [4, 5, 9]. Такой набор средств позволяет получить компактную, но полноценную мини-CRM для учета и анализа деятельности автосервисной компании.

Файл app.py является главным файлом приложения, который запускается для его старта. Ниже приводится код данного файла.

Файл app.py

```
from flask import Flask, render_template, request,
redirect, url_for
import sqlite3
from datetime import datetime

app = Flask(__name__)
DATABASE = "autotrade.db"

def get_db():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn

def init_db():
    conn = get_db()
    cur = conn.cursor()

    cur.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL,
    password TEXT NOT NULL,
    role TEXT NOT NULL
)
""")

    cur.execute("""
CREATE TABLE IF NOT EXISTS employees (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    position TEXT,
    phone TEXT
)
""")

    cur.execute("""
CREATE TABLE IF NOT EXISTS clients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    phone TEXT NOT NULL,
    email TEXT,
    note TEXT
)
""")
```

```

cur.execute("""
CREATE TABLE IF NOT EXISTS cars (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    brand TEXT NOT NULL,
    model TEXT NOT NULL,
    year INTEGER,
    plate_number TEXT,
    FOREIGN KEY (client_id) REFERENCES clients(id)
)
""")

```

```

cur.execute("""
CREATE TABLE IF NOT EXISTS services (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL NOT NULL,
    duration INTEGER
)
""")

```

```

cur.execute("""
CREATE TABLE IF NOT EXISTS service_requests (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    car_id INTEGER NOT NULL,
    employee_id INTEGER,
    status TEXT NOT NULL,
    description TEXT,
    created_at TEXT,
    total_price REAL,
    FOREIGN KEY (client_id) REFERENCES clients(id),
    FOREIGN KEY (car_id) REFERENCES cars(id),
    FOREIGN KEY (employee_id) REFERENCES employees(id)
)
""")

```

```

cur.execute("""
CREATE TABLE IF NOT EXISTS request_services (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_id INTEGER NOT NULL,
    service_id INTEGER NOT NULL,
    FOREIGN KEY (request_id) REFERENCES
service_requests(id),
    FOREIGN KEY (service_id) REFERENCES services(id)
)
""")

```

```

cur.execute("SELECT COUNT(*) FROM employees")
if cur.fetchone()[0] == 0:
    cur.executemany(
        "INSERT INTO employees (full_name, position,
phone) VALUES (?, ?, ?)",
        [
            ("Иванов Сергей Петрович", "Мастер-
приемщик", "+7 900 111-22-33"),
            ("Кузнецов Алексей Игоревич",
"Автомеханик", "+7 900 222-33-44"),
            ("Смирнова Анна Олеговна", "Администратор",
"+7 900 333-44-55")
        ]
    )

cur.execute("SELECT COUNT(*) FROM services")
if cur.fetchone()[0] == 0:
    cur.executemany(
        "INSERT INTO services (name, price, duration)
VALUES (?, ?, ?)",
        [
            ("Диагностика двигателя", 2500, 60),
            ("Замена масла", 1800, 40),
            ("Замена тормозных колодок", 3500, 90),
            ("Компьютерная диагностика", 2200, 45),
            ("Проверка подвески", 2000, 50)
        ]
    )

cur.execute("SELECT COUNT(*) FROM clients")
if cur.fetchone()[0] == 0:
    cur.executemany(
        "INSERT INTO clients (full_name, phone, email,
note) VALUES (?, ?, ?, ?)",
        [
            ("Петров Андрей Викторович", "+7 900 444-
55-66", "petrov@mail.ru", "Постоянный клиент"),
            ("Соколова Мария Павловна", "+7 900 555-66-
77", "sokolova@mail.ru", "Запись по рекомендации")
        ]
    )
    cur.executemany(
        "INSERT INTO cars (client_id, brand, model,
year, plate_number) VALUES (?, ?, ?, ?, ?)",
        [
            (1, "Toyota", "Camry", 2018, "A123BC77"),
            (2, "Kia", "Rio", 2020, "M456OP77")
        ]
    )

```

```

        ]
    )

    conn.commit()
    conn.close()

@app.route("/")
def dashboard():
    conn = get_db()
    total_clients = conn.execute("SELECT COUNT(*) FROM
clients").fetchone()[0]
    total_cars = conn.execute("SELECT COUNT(*) FROM
cars").fetchone()[0]
    total_requests = conn.execute("SELECT COUNT(*) FROM
service_requests").fetchone()[0]
    total_revenue = conn.execute(
        "SELECT COALESCE(SUM(total_price), 0) FROM
service_requests"
    ).fetchone()[0]

    status_rows = conn.execute("""
        SELECT status, COUNT(*) AS count
        FROM service_requests
        GROUP BY status
    """).fetchall()

    status_labels = [row["status"] for row in status_rows]
    status_values = [row["count"] for row in status_rows]

    conn.close()
    return render_template(
        "dashboard.html",
        total_clients=total_clients,
        total_cars=total_cars,
        total_requests=total_requests,
        total_revenue=total_revenue,
        status_labels=status_labels,
        status_values=status_values
    )

@app.route("/clients", methods=["GET", "POST"])
def clients():
    conn = get_db()

    if request.method == "POST":
        conn.execute(

```

```

        "INSERT INTO clients (full_name, phone, email,
note) VALUES (?, ?, ?, ?)",
        (
            request.form["full_name"],
            request.form["phone"],
            request.form.get("email", ""),
            request.form.get("note", "")
        )
    )
    conn.commit()
    return redirect(url_for("clients"))

    clients_list = conn.execute("SELECT * FROM clients
ORDER BY id DESC").fetchall()
    conn.close()
    return render_template("clients.html",
clients=clients_list)

@app.route("/cars", methods=["GET", "POST"])
def cars():
    conn = get_db()

    if request.method == "POST":
        conn.execute(
            "INSERT INTO cars (client_id, brand, model,
year, plate_number) VALUES (?, ?, ?, ?, ?)",
            (
                request.form["client_id"],
                request.form["brand"],
                request.form["model"],
                request.form.get("year") or None,
                request.form.get("plate_number", "")
            )
        )
        conn.commit()
        return redirect(url_for("cars"))

    cars_list = conn.execute("""
SELECT cars.*, clients.full_name AS client_name
FROM cars
JOIN clients ON clients.id = cars.client_id
ORDER BY cars.id DESC
""").fetchall()

    clients_list = conn.execute("SELECT * FROM clients
ORDER BY full_name").fetchall()
    conn.close()

```

```

        return render_template("cars.html", cars=cars_list,
clients=clients_list)

@app.route("/requests", methods=["GET", "POST"])
def requests_page():
    conn = get_db()

    if request.method == "POST":
        service_ids = request.form.getlist("service_ids")
        total_price = 0

        for service_id in service_ids:
            service = conn.execute(
                "SELECT price FROM services WHERE id = ?",
                (service_id,)
            ).fetchone()
            if service:
                total_price += service["price"]

        cur = conn.execute(
            """
            INSERT INTO service_requests
            (client_id, car_id, employee_id, status,
description, created_at, total_price)
            VALUES (?, ?, ?, ?, ?, ?, ?)
            """,
            (
                request.form["client_id"],
                request.form["car_id"],
                request.form.get("employee_id") or None,
                request.form["status"],
                request.form.get("description", ""),
                datetime.now().strftime("%Y-%m-%d"),
                total_price
            )
        )

        request_id = cur.lastrowid
        for service_id in service_ids:
            conn.execute(
                "INSERT INTO request_services (request_id,
service_id) VALUES (?, ?)",
                (request_id, service_id)
            )

        conn.commit()
        return redirect(url_for("requests_page"))

```

```

        requests_list = conn.execute("""
            SELECT
                service_requests.*,
                clients.full_name AS client_name,
                cars.brand || ' ' || cars.model || ' (' ||
cars.plate_number || ')' AS car_name,
                employees.full_name AS employee_name
            FROM service_requests
            JOIN      clients      ON      clients.id      =
service_requests.client_id
            JOIN cars ON cars.id = service_requests.car_id
            LEFT JOIN      employees      ON      employees.id      =
service_requests.employee_id
            ORDER BY service_requests.id DESC
        """).fetchall()

        clients_list = conn.execute("SELECT * FROM clients
ORDER BY full_name").fetchall()
        cars_list = conn.execute("SELECT * FROM cars ORDER BY
brand").fetchall()
        employees_list = conn.execute("SELECT * FROM employees
ORDER BY full_name").fetchall()
        services_list = conn.execute("SELECT * FROM services
ORDER BY name").fetchall()

        conn.close()
        return render_template(
            "requests.html",
            requests=requests_list,
            clients=clients_list,
            cars=cars_list,
            employees=employees_list,
            services=services_list
        )

@app.route("/reports")
def reports():
    conn = get_db()

    status_rows = conn.execute("""
        SELECT status, COUNT(*) AS count
        FROM service_requests
        GROUP BY status
    """).fetchall()

    service_rows = conn.execute("""
        SELECT services.name, COUNT(request_services.id) AS
count
    """).fetchall()

```

```

        FROM request_services
        JOIN      services      ON      services.id      =
request_services.service_id
        GROUP BY services.name
        ORDER BY count DESC
        """).fetchall()

    conn.close()
    return render_template(
        "reports.html",
        status_labels=[row["status"]      for      row      in
status_rows],
        status_values=[row["count"]      for      row      in
status_rows],
        service_labels=[row["name"]      for      row      in
service_rows],
        service_values=[row["count"]      for      row      in
service_rows]
    )
if __name__ == "__main__":
    init_db()
    app.run(debug=True)

```

Остальные файлы мини-CRM приведены в Приложении Б.

1.3.4 Использование приложения

Разработанное веб-приложение мини-CRM «АвтоТрейдСервис» предназначено для ежедневной работы сотрудников автосервисной компании с клиентами, автомобилями, заявками и отчетами. Пользователь взаимодействует с системой через браузер. После запуска Flask-приложения открывается главная страница, с которой доступны основные разделы:

- панель управления;
- клиенты;
- автомобили;
- заявки;
- отчеты.

Работа пользователя строится по следующей последовательности. Сначала сотрудник открывает систему, затем переходит в нужный раздел, выполняет учетную операцию и получает обновленные данные. Такой порядок снижает количество лишних действий и делает приложение понятным для пользователя

без специальной технической подготовки. Основные разделы интерфейса мини-CRM приведены в Таблице 1.3.4.1.

Таблица 1.3.4.1 — Основные разделы интерфейса мини-CRM

Раздел приложения	Назначение	Основные действия пользователя
Панель управления	Краткий просмотр ключевых показателей	Просмотр количества клиентов, автомобилей, заявок и выручки
Клиенты	Ведение клиентской базы	Добавление и просмотр клиентов
Автомобили	Учет автомобилей клиентов	Добавление автомобиля и привязка к клиенту
Заявки	Учет обращений на обслуживание	Создание заявки, выбор услуг, статуса и сотрудника
Отчеты	Анализ деятельности автосервиса	Просмотр графиков по заявкам и услугам

Первый экран приложения — панель управления. На ней пользователь видит сводные показатели:

- число клиентов;
- число автомобилей;
- число заявок;
- статус заявок;
- общую сумму выполненных работ.

Также отображается диаграмма заявок по статусам. Этот экран нужен руководителю и администратору для быстрой оценки текущего состояния работы автосервиса.

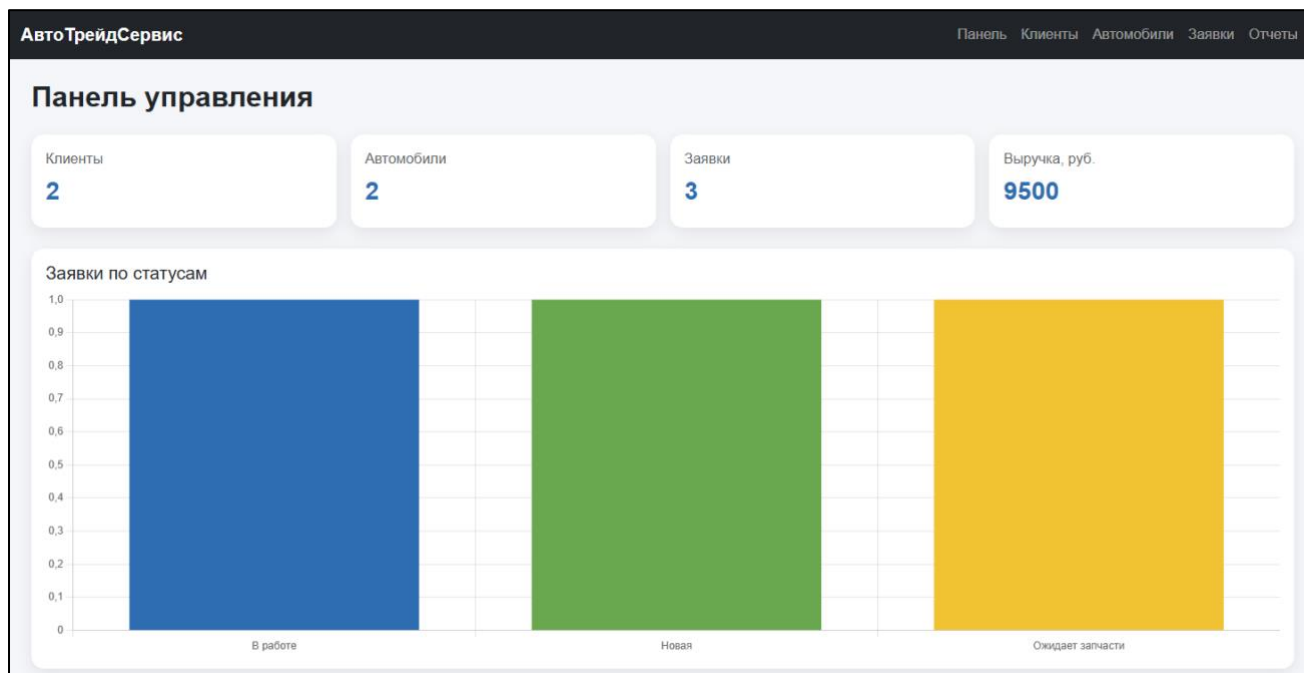


Рисунок 3.4.1 — Главная панель управления мини-CRM «АвтоТрейдСервис»

После просмотра панели управления пользователь может перейти в раздел клиентов. Здесь отображается таблица всех клиентов и форма добавления новой записи. Для регистрации клиента указываются ФИО, телефон, электронная почта и примечание. После отправки формы запись сохраняется в базе данных и появляется в таблице.

АвтоТрейдСервис | Панель | Клиенты | Автомобили | Заявки | Отчеты

Клиенты

! Вы пропустили это поле.

ID	ФИО	Телефон	Email	Примечание
2	Соколова Мария Павловна	+7 900 555-66-77	sokolova@mail.ru	Запись по рекомендации
1	Петров Андрей Викторович	+7 900 444-55-66	petrov@mail.ru	Постоянный клиент

Рисунок 3.4.2 — Форма учета клиентов

Следующий сценарий — добавление автомобиля. В разделе «Автомобили» пользователь выбирает клиента из списка, указывает марку, модель, год выпуска

и государственный номер. Связь автомобиля с клиентом создается через поле client_id, что позволяет в дальнейшем видеть, какие автомобили принадлежат конкретному клиенту. Такая связь особенно важна, если один клиент обслуживает несколько автомобилей.

АвтоТрейдСервис

ПанельКлиентыАвтомобилиЗаявкиОтчеты

Автомобили

Петров Андрей Викторович

Марка

Модель

Год

Госномер

+

Вы пропустили это поле.

ID	Клиент	Марка	Модель	Год	Госномер
2	Соколова Мария Павловна	Kia	Rio	2020	M456OP77
1	Петров Андрей Викторович	Toyota	Camry	2018	A123BC77

Рисунок 3.4.3 — Форма учета автомобилей клиентов

Основная рабочая операция системы — создание заявки. В разделе «Заявки» пользователь выбирает клиента, автомобиль, сотрудника, статус и перечень услуг. Система автоматически рассчитывает итоговую стоимость заявки на основе выбранных услуг. После сохранения заявка появляется в общем списке. Для администратора это основной инструмент учета текущих обращений.

АвтоТрейдСервис

ПанельКлиентыАвтомобилиЗаявкиОтчеты

Заявки

Петров Андрей Виктор

Kia Rio

Иванов Сергей Петров

Новая

Диагностика двигателя
Замена масла
Замена тормозных колод
Компьютерная диагности

Описание

+

ID	Клиент	Автомобиль	Сотрудник	Статус	Дата	Сумма
3	Петров Андрей Викторович	Kia Rio (M456OP77)	Кузнецов Алексей Игоревич	Ожидает запчасти	2026-05-10	3500.0
2	Соколова Мария Павловна	Toyota Camry (A123BC77)	Кузнецов Алексей Игоревич	В работе	2026-05-10	3500.0
1	Петров Андрей Викторович	Kia Rio (M456OP77)	Иванов Сергей Петрович	Новая	2026-05-10	2500.0

Рисунок 3.4.4 — Форма создания заявки на обслуживание автомобиля

В заявке используется набор статусов:

- «Новая»;
- «В работе»;
- «Ожидает запчастей»;
- «Завершена».

Такой подход помогает контролировать состояние обслуживания без устных уточнений. Руководитель может быстро понять, сколько заявок еще не принято в работу, сколько находится в ремонте и сколько уже закрыто.

Для анализа работы применяется раздел отчетов. Он содержит диаграммы по заявкам и услугам. Первый график показывает распределение заявок по статусам. Второй график отражает популярность услуг. Эти данные помогают выявить наиболее частые виды работ и оценить текущую нагрузку на сервис.

Расчетные

данные для анализа активности по модулям приложения приведены в Таблице 1.3.4.2.

Таблица 1.3.4.2 — Расчетные данные для графика распределения пользовательской активности по модулям мини-CRM

Модуль приложения	Количество действий за период	Доля активности, %
Заявки	120	40
Клиенты	72	24
Автомобили	54	18
Отчеты	36	12
Панель управления	18	6

Распределение пользовательской активности по модулям мини-CRM приведено на Рисунке 1.3.4.5.

Рисунок 1.3.4.5 показывает, что основная активность пользователя сосредоточена в модуле заявок. Это соответствует назначению системы: главная рабочая операция автосервиса связана с приемом, обработкой и контролем заявок. Разделы клиентов и автомобилей используются как вспомогательные, но

обязательные элементы учета. Отчеты применяются реже, поскольку они нужны в первую очередь для управленческого анализа.

Типовой сценарий работы администратора включает несколько шагов:

- сотрудник включает приложение в браузере и попадает на главный экран;
- при обращении нового клиента сотрудник открывает раздел «Клиенты» и добавляет карточку клиента;
- после этого в разделе «Автомобили» создается запись об автомобиле и выполняется привязка к клиенту;
- далее в разделе «Заявки» оформляется обращение на обслуживание, выбираются услуги, статус и ответственный сотрудник;
- после сохранения заявка появляется в таблице, а показатели панели управления обновляются;
- руководитель переходит в раздел «Отчеты» и анализирует состояние заявок и популярность услуг.

					ПГУ 1.090301.26.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

Распределение пользовательской активности по модулям mini-CRM

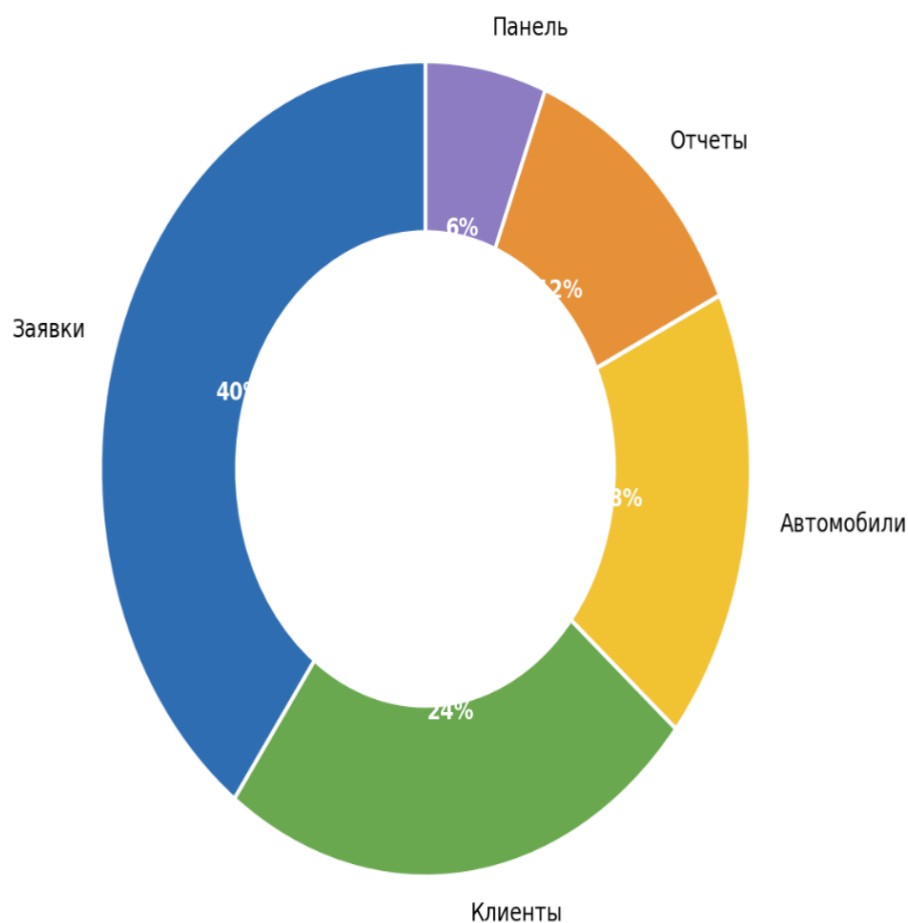


Рисунок 1.3.4.5 — Распределение пользовательской активности по модулям мини-CRM

Такая последовательность охватывает полный цикл работы мини-CRM: от первичного обращения клиента до получения отчетной информации. Все данные сохраняются в SQLite и используются повторно при создании новых записей. Это снижает дублирование сведений и делает работу сотрудников более упорядоченной.

Использование приложения показывает, что разработанная система выполняет основные функции учета и анализа деятельности автосервисной компании. Пользователь может вести клиентскую базу, хранить сведения об автомобилях, создавать заявки, выбирать услуги и просматривать отчетность.

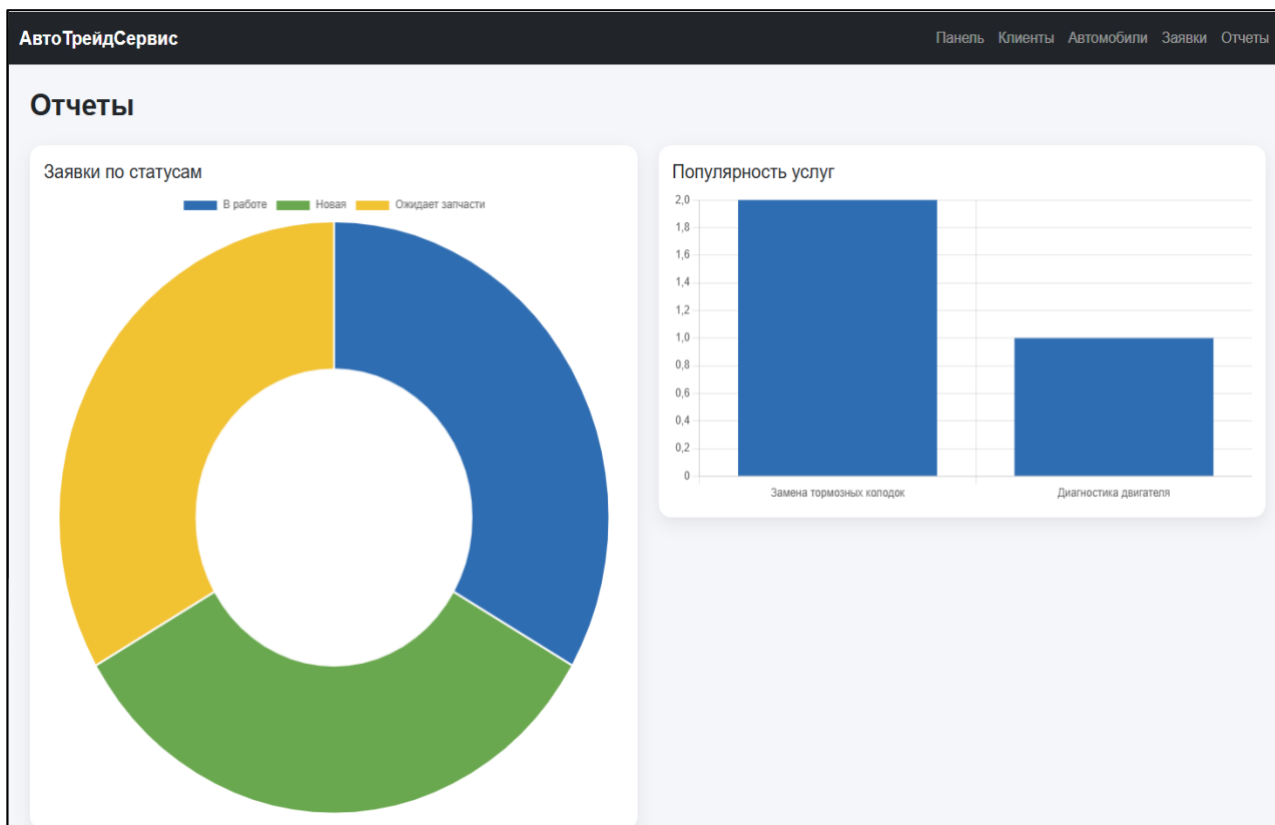


Рисунок 1.3.4.6 — Раздел отчетности мини-CRM «АвтоТрейдСервис»

1.3.5 Тестирование и отладка

Тестирование разработанного приложения мини-CRM «АвтоТрейдСервис» выполняется для проверки работоспособности основных функций системы. На этом этапе необходимо убедиться, что приложение правильно запускается, создает базу данных, отображает страницы интерфейса, сохраняет записи и формирует отчетные показатели. Проверка особенно важна для систем учета, потому что ошибка в сохранении или отображении данных может привести к неверной отчетности [12].

Тестирование проводилось по функциональным сценариям. Такой подход позволяет проверить приложение не только на уровне отдельных файлов, но и с позиции пользователя. Сотрудник автосервиса открывает страницу, вводит данные, сохраняет запись и ожидает, что информация появится в таблице. Руководитель открывает раздел отчетов и ожидает, что графики будут построены на основе сохраненных заявок. Если эти действия выполняются правильно, система готова к использованию.

Основные задачи тестирования:

- проверить запуск Flask-приложения;
- убедиться в создании базы данных SQLite;
- проверить отображение главной панели управления;
- протестировать добавление клиентов, автомобилей и заявок;
- проверить расчет итоговой стоимости заявки;
- убедиться в корректном отображении отчетных графиков;
- проверить обработку неполных и ошибочных данных;
- оценить удобство переходов между разделами.

Тестовые сценарии проверки мини-CRM «АвтоТрейдСервис» приведены в Таблице 1.3.5.1.

По итогам тестирования проверяются как основные учетные функции, так и вспомогательные операции. Особое внимание уделяется созданию заявки, потому что именно этот сценарий связывает несколько таблиц:

- клиента;
- автомобиль;
- сотрудника;
- услугу;
- саму заявку.

Если этот сценарий работает правильно, можно считать, что база данных и серверная часть взаимодействуют согласованно.

Таблица 1.3.5.1 — Тестовые сценарии проверки мини-CRM

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
Запуск приложения	Проверка старта Flask-сервера	Запустить команду <code>python app.py</code> , открыть <code>http://127.0.0.1:5000</code>	Приложение открывается в браузере
Создание базы данных	Проверка автоматического создания SQLite-файла	Удалить <code>autotrade.db</code> , снова запустить приложение	Файл базы создается автоматически

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
Открытие панели управления	Проверка главной страницы	Перейти на главную страницу приложения	Отображаются карточки показателей и график
Добавление клиента	Проверка формы клиентов	Открыть «Клиенты», заполнить ФИО и телефон, нажать кнопку добавления	Новый клиент появляется в таблице
Добавление автомобиля	Проверка привязки автомобиля к клиенту	Открыть «Автомобили», выбрать клиента, указать марку и модель	Автомобиль сохраняется и отображается с владельцем
Создание заявки	Проверка основного бизнес-сценария	Открыть «Заявки», выбрать клиента, автомобиль, услугу и статус	Заявка создается и появляется в списке
Расчет стоимости заявки	Проверка расчета суммы по услугам	При создании заявки выбрать одну или несколько услуг	Итоговая стоимость соответствует сумме выбранных услуг
Отображение статуса	Проверка статуса заявки	Создать заявку со статусом «В работе»	В таблице заявок отображается выбранный статус
Отчет по статусам	Проверка графика заявок по статусам	Создать несколько заявок с разными статусами, открыть «Отчеты»	Диаграмма отображает распределение заявок
Отчет по услугам	Проверка графика популярности услуг	Создать заявки с разными услугами, открыть «Отчеты»	График показывает количество использований услуг
Обязательные поля клиента	Проверка валидации формы	Попытаться добавить клиента без ФИО или телефона	Браузер не отправляет форму без обязательных полей
Навигация по разделам	Проверка переходов интерфейса	Перейти по ссылкам «Панель», «Клиенты», «Автомобили», «Заявки», «Отчеты»	Все разделы открываются без ошибки
Повторный запуск	Проверка сохранения данных	Добавить записи, закрыть сервер, запустить приложение снова	Ранее добавленные данные сохраняются
Проверка пустых отчетов	Проверка работы отчетности без заявок	Открыть отчеты при отсутствии заявок	Страница открывается без сбоя
Проверка отображения таблиц	Проверка интерфейса списков	Открыть разделы клиентов, автомобилей и заявок	Таблицы отображаются в читаемом виде

Отладка приложения выполнялась поэтапно. Сначала проверялся запуск сервера и корректность маршрутов. Затем проверялись формы добавления

клиентов и автомобилей. После этого тестировалось создание заявки, расчет стоимости и запись выбранных услуг в связующую таблицу. На последнем этапе проверялась отчетность, поскольку графики зависят от уже накопленных данных.

При отладке могли возникать типовые ошибки:

- неверное имя маршрута в шаблоне;
- отсутствие нужного поля в HTML-форме;
- неправильное имя таблицы или столбца в SQL-запросе;
- попытка сохранить заявку без связанного клиента или автомобиля;
- ошибка передачи данных из Flask в Chart.js;
- несоответствие типов данных в форме и базе.

Для устранения таких ошибок применялась последовательная проверка:

- сначала просматривался текст ошибки Flask;
- затем проверялся соответствующий маршрут;
- далее проверялись SQL-запрос и HTML-шаблон.

Этот способ удобен для небольшого веб-приложения, потому что каждая операция имеет понятный путь: форма → маршрут → база данных → шаблон.

Отдельно проверялась работа отчетов. График заявок по статусам строится на основе группировки записей в таблице `service_requests`. График популярности услуг использует связь `request_services` и `services`. Если связующая таблица заполняется неправильно, отчет по услугам будет пустым или неполным. Поэтому тестирование создания заявки выполнялось вместе с проверкой отчетной страницы.

Важной частью тестирования является проверка сохранности данных. После добавления клиентов, автомобилей и заявок приложение перезапускалось. При повторном открытии страницы ранее созданные записи должны оставаться в таблицах. Это подтверждает, что данные сохраняются в SQLite, а не существуют только во время работы серверного процесса [9].

Проверка пользовательского интерфейса показала, что основные элементы системы доступны из верхнего меню. Таблицы имеют понятную структуру, формы размещены перед списками, а отчетные графики находятся в отдельном

разделе. Такой вариант подходит для ежедневного применения, так как пользователь не тратит время на поиск основных функций.

Результаты тестирования подтверждают, что мини-CRM выполняет заявленные функции. Приложение запускается, создает базу данных, сохраняет клиентов и автомобили, формирует заявки, рассчитывает стоимость работ и отображает отчетные графики. Выявленные в процессе отладки ошибки относятся к типовым ошибкам разработки веб-приложений и устраняются путем проверки маршрутов, форм и SQL-запросов.

					ПГУ 1.090301.26.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		

2 Безопасность жизнедеятельности

Разработка и использование мини-CRM «АвтоТрейдСервис» связаны с работой за персональным компьютером. Поэтому при эксплуатации системы необходимо учитывать требования безопасности жизнедеятельности, относящиеся к организации рабочего места пользователя, снижению зрительной нагрузки, предупреждению утомления и обеспечению электробезопасности. В Разделе 2 образца ВКР вопросы охраны труда также рассматриваются через условия работы пользователя с вычислительной техникой, поэтому для данного проекта применяется сходный подход.

Основными пользователями системы являются администратор, мастер-приемщик и руководитель автосервиса. Их работа предполагает ввод данных, просмотр таблиц, создание заявок и анализ отчетов. При длительной работе с ПК на сотрудника могут воздействовать опасные и вредные факторы:

- зрительное напряжение при длительном просмотре экрана;
- статическая нагрузка на мышцы спины, шеи и плечевого пояса;
- утомление кистей при работе с клавиатурой и мышью;
- нервно-эмоциональное напряжение при обработке клиентских обращений;
- неблагоприятные параметры микроклимата;
- риск поражения электрическим током при неисправности оборудования.

Для снижения зрительной нагрузки рабочее место должно быть организовано так, чтобы экран находился на расстоянии примерно 50–70 см от глаз пользователя. Верхняя граница монитора должна располагаться на уровне глаз или немного ниже. Необходимо исключить блики на экране от окон и светильников. Освещение должно быть равномерным, без резких перепадов яркости между экраном и окружающим пространством.

Для профилактики статической нагрузки рабочее место должно включать устойчивый стол, регулируемый стул, свободное пространство для ног и удобное расположение клавиатуры. Спина пользователя должна иметь опору, а

предплечья — располагаться примерно горизонтально. Нежелательно размещать монитор слишком высоко или сбоку, потому что это приводит к постоянному напряжению шеи.

При работе с мини-CRM важно также учитывать эргономику интерфейса. Разработанное приложение использует крупные таблицы, понятные формы и контрастные элементы Bootstrap. Это снижает вероятность ошибок при вводе данных и уменьшает зрительное утомление. Цветовые акценты применяются умеренно: они помогают различать показатели, но не перегружают экран.

Режим труда и отдыха имеет большое значение. При длительной работе за компьютером рекомендуется делать короткие перерывы, менять положение тела, выполнять простые упражнения для глаз и кистей рук. Такие меры особенно важны для администратора, который в течение смены постоянно работает с клиентскими заявками.

Электробезопасность обеспечивается использованием исправного оборудования, заземленных розеток, сетевых фильтров и отсутствием поврежденных кабелей. Запрещается работать с техникой при признаках перегрева, запахе гари, искрении или нарушении изоляции. Оборудование должно подключаться так, чтобы провода не находились в проходах и не создавали риск спотыкания.

Санитарные требования к рабочим местам и факторам производственной среды регулируются санитарными правилами и нормами, включая СанПиН 1.2.3685-21 [13]. При организации работы с приложением необходимо соблюдать эти требования в части освещенности, микроклимата, шума и условий труда пользователей ПК.

В результате безопасное применение мини-CRM «АвтоТрейдСервис» обеспечивается не только исправной работой программы, но и правильной организацией рабочего места. Соблюдение требований охраны труда снижает утомляемость сотрудников, уменьшает риск ошибок при вводе данных и повышает устойчивость работы автосервисной компании.

Заключение

В ходе выполнения выпускной квалификационной работы была рассмотрена задача автоматизации учета и анализа деятельности автосервисной компании. Результатом работы стала разработка веб-приложения мини-CRM «АвтоТрейдСервис», предназначенного для учета клиентов, автомобилей, заявок, визуализации данных и формирования отчетности.

Основной целью работы было разработать веб-приложение мини-CRM «АвтоТрейдСервис» для учета клиентов, автомобилей, заявок, анализа деятельности компании, визуализации данных и формирования отчетности.

Результаты выпускной квалификационной работы последовательно представлены в ее главах.

В Главе 1 был проведен анализ предметной области, обоснована актуальность разработки, рассмотрены существующие аналоги CRM-систем и сформулирована постановка задачи на разработку приложения.

В Главе 2 был выполнен выбор технологий и средств разработки. Для реализации системы выбраны Python, Flask, SQLite, HTML, CSS, Bootstrap, Chart.js и среда разработки Visual Studio Code.

В Главе 3 были выполнены проектирование архитектуры приложения, проектирование базы данных, разработка программного продукта, описание пользовательских сценариев, тестирование, рассмотрение вопросов безопасности жизнедеятельности.

В работе выполнено решение поставленных задач.

Задача «Анализ предметной области». Были изучены особенности работы автосервисной компании, включая учет клиентов, автомобилей, заявок, услуг и статусов выполнения работ. Установлено, что ручной и разрозненный учет приводит к потерям времени, затрудняет подготовку отчетности и снижает прозрачность управления заявками.

Задача «Анализ аналогов». Были рассмотрены универсальные CRM-системы, специализированные решения для автосервисов и простые средства

учета на базе электронных таблиц. По результатам анализа сделан вывод, что для небольшой автосервисной компании целесообразна разработка компактной мини-CRM, не перегруженной лишними функциями.

Задача «Выбор технологий». Для серверной части выбран язык Python и фреймворк Flask, что позволило создать понятную структуру веб-приложения. Для хранения данных выбрана SQLite, так как она не требует отдельного сервера и подходит для небольшого проекта. Для интерфейса применены Bootstrap и Chart.js, обеспечившие аккуратное оформление страниц и визуализацию отчетных данных.

Задача «Проектирование системы». Была разработана архитектура приложения, включающая клиентскую часть, серверную часть и базу данных. Также были построены диаграммы архитектуры, развертывания, последовательности, классов и деятельности. Для базы данных спроектированы таблицы users, employees, clients, cars, services, service_requests и request_services.

Задача «Разработка приложения». Было реализовано веб-приложение мини-CRM «АвтоТрейдСервис» с разделами «Панель управления», «Клиенты», «Автомобили», «Заявки» и «Отчеты». Приложение позволяет добавлять клиентов, регистрировать автомобили, создавать заявки, выбирать услуги, рассчитывать стоимость работ и просматривать отчетные графики.

Задача «Тестирование системы». Были проверены запуск приложения, создание базы данных, отображение страниц, добавление клиентов, автомобилей и заявок, расчет стоимости, сохранение данных и построение отчетов. Тестирование подтвердило работоспособность основных функций системы.

Практическая ценность работы заключается в создании компактного веб-приложения, которое может применяться небольшой автосервисной компанией для организации клиентского и заявочного учета. Система повышает прозрачность работы, ускоряет обработку обращений и дает руководителю инструменты анализа деятельности.

На основании результатов проведенного исследования можно сформулировать следующие практические рекомендации:

- в дальнейшем целесообразно добавить систему ролей, чтобы разграничить права администратора, мастера и руководителя;
- перспективным направлением развития является добавление модуля учета запасных частей и складских остатков;
- для повышения удобства работы можно реализовать экспорт отчетов в PDF или Excel;
- при расширении количества пользователей возможно перенести базу данных с SQLite на PostgreSQL;
- для повышения качества клиентского сервиса можно добавить уведомления о статусе заявки через электронную почту или мессенджер.

Из представленных результатов следует — поставленная цель работы достигнута, а задачи, сформулированные во введении, решены в полном объеме.

Список использованных источников

1. Зараменских, Е. П. Информационные системы в бизнесе : учебник и практикум / Е. П. Зараменских. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 470 с. — ISBN 978-5-534-17537-0. — Текст : электронный // Образовательная платформа Юрайт. — URL: <https://urait.ru/bcode/566519> (дата обращения: 09.05.2026).

2. Моргунов, А. Ф. Информационные технологии в менеджменте : учебник для вузов / А. Ф. Моргунов. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2024. — 380 с. — ISBN 978-5-534-19417-3. — Текст : электронный // Образовательная платформа Юрайт. — URL: <https://urait.ru/bcode/556437> (дата обращения: 09.05.2026).

3. Волк, В. К. Базы данных. Проектирование, программирование, управление и администрирование : учебник для вузов / В. К. Волк. — Санкт-Петербург : Лань, 2025. — 244 с. — ISBN 978-5-507-53648-1. — Текст : электронный // ЭБС Лань. — URL: <https://lanbook.com/catalog/informatika/bazy-dannykh-proektirovanie-programmirovanie-upravlenie-i-administrirovanie/> (дата обращения: 09.05.2026).

4. Flask Documentation. — Текст : электронный. — URL: <https://flask.palletsprojects.com/> (дата обращения: 09.05.2026).

5. Bootstrap 5 Documentation. — Текст : электронный. — URL: <https://getbootstrap.com/docs/5.3/gettingstarted/> (дата обращения: 09.05.2026).

6. Chart.js Documentation. — Текст : электронный. — URL: <https://www.chartjs.org/docs/latest/> (дата обращения: 09.05.2026).

7. ГОСТ Р 59793-2021. Информационные технологии. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания. — Москва : Стандартинформ, 2021. — Текст : электронный. — URL: <https://docs.cntd.ru/document/1200181349> (дата обращения: 09.05.2026).

8. АвтоДилер Онлайн : программа для СТО и автоматизации учета в автосервисе. — Текст : электронный. — URL: <https://autodealer.ru/> (дата обращения: 09.05.2026).

9. SQLite Documentation. — Текст : электронный. — URL: <https://sqlite.org/docs.html> (дата обращения: 09.05.2026).

10. Гаврилов, М. В. Информатика и информационные технологии : учебник для вузов / М. В. Гаврилов, В. А. Климов. — 5-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2024. — 355 с. — ISBN 978-5-534-15819-9. — Текст : электронный // Образовательная платформа Юрайт. — URL: <https://urait.ru/bcode/535560> (дата обращения: 09.05.2026).

11. Башарина, О. Ю. Управление информационными системами : учебное пособие / О. Ю. Башарина, С. В. Бегичева, Е. В. Буценко [и др.]. — Екатеринбург : УрГЭУ, 2023. — 115 с. — Текст : электронный // ЭБС Лань. — URL: <https://e.lanbook.com/book/444044> (дата обращения: 09.05.2026).

12. Богатырев, В. А. Информационные системы и технологии. Теория надежности : учебное пособие для вузов / В. А. Богатырев. — 2-е изд. — Москва : Издательство Юрайт, 2026. — 366 с. — ISBN 978-5-534-15951-6. — Текст : электронный // Образовательная платформа Юрайт. — URL: <https://urait.ru/bcode/590557> (дата обращения: 09.05.2026).

13. Постановление Главного государственного санитарного врача Российской Федерации от 28.01.2021 № 2 «Об утверждении санитарных правил и норм СанПиН 1.2.3685-21» : с изменениями на 30.12.2022. — Текст : электронный. — URL: <https://docs.cntd.ru/document/573500115> (дата обращения: 09.05.2026).

14. Кожевникова, Г. П. Информационные системы и технологии в маркетинге : учебное пособие для вузов / Г. П. Кожевникова, Б. Е. Одинцов. — Москва : Издательство Юрайт, 2024. — 444 с. — ISBN 978-5-534-07447-5. — Текст : электронный // Образовательная платформа Юрайт. — URL: <https://urait.ru/bcode/536501> (дата обращения: 09.05.2026).

15. Чеботков, М. С. Сравнительный анализ традиционной и цифровой моделей бизнеса в контексте трансформации малого и среднего предпринимательства / М. С. Чеботков, А. А. Антонова // Journal of Monetary Economics and Management. — 2025. — № 12. — С. 268–274. — DOI: 10.26118/2782-4586.2025.62.61.033.

16. YCLIENTS : программа для онлайн-записи и автоматизации автосервиса. — Текст : электронный. — URL: <https://www.yclients.com/auto-service> (дата обращения: 09.05.2026).

					ПГУ 1.090301.26.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

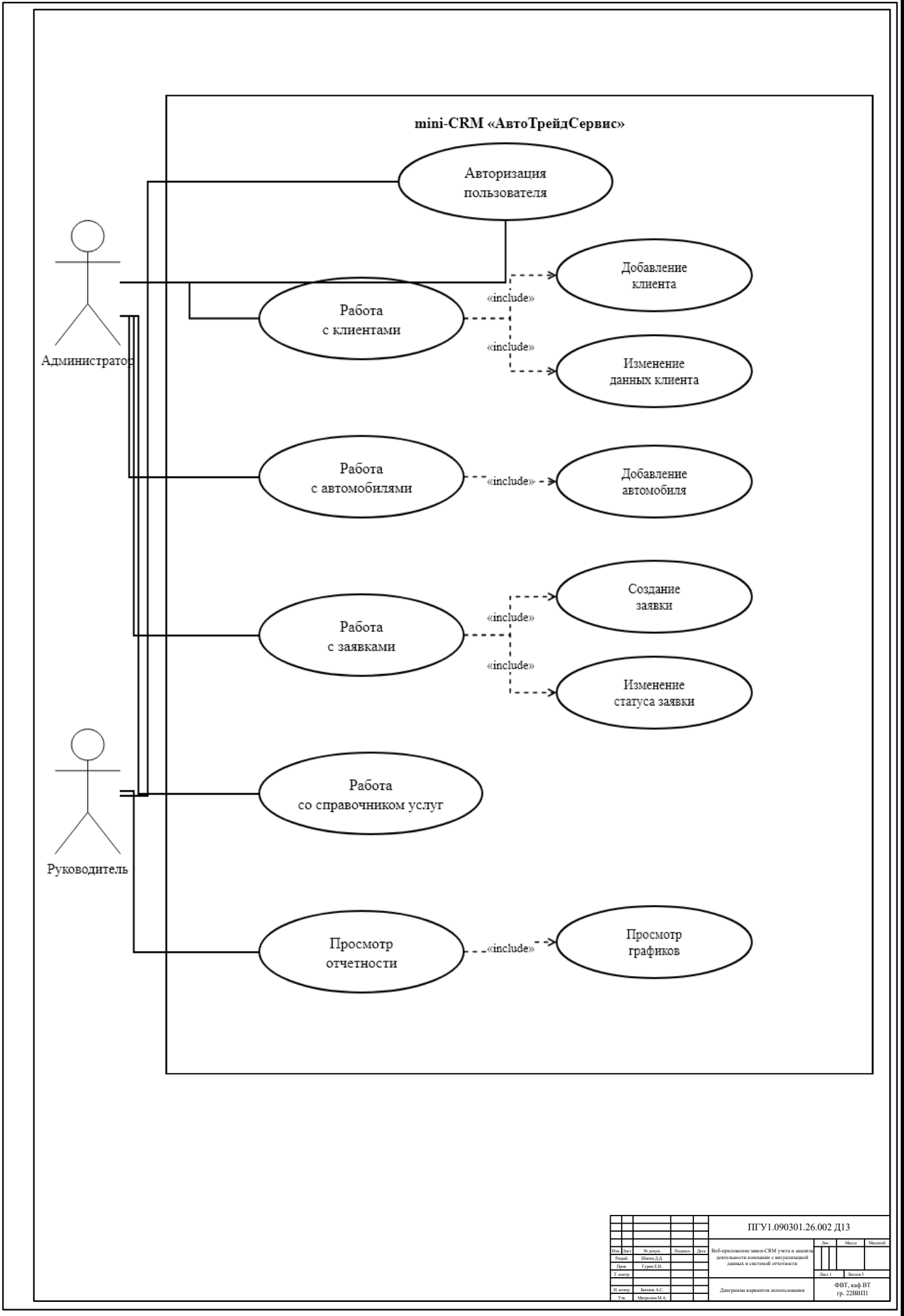
Приложение А

(обязательное)

Мини-CRM «АвтоТрейдСервис» для учета и анализа деятельности компании с
визуализацией данных и системой отчетности

UML-диаграммы

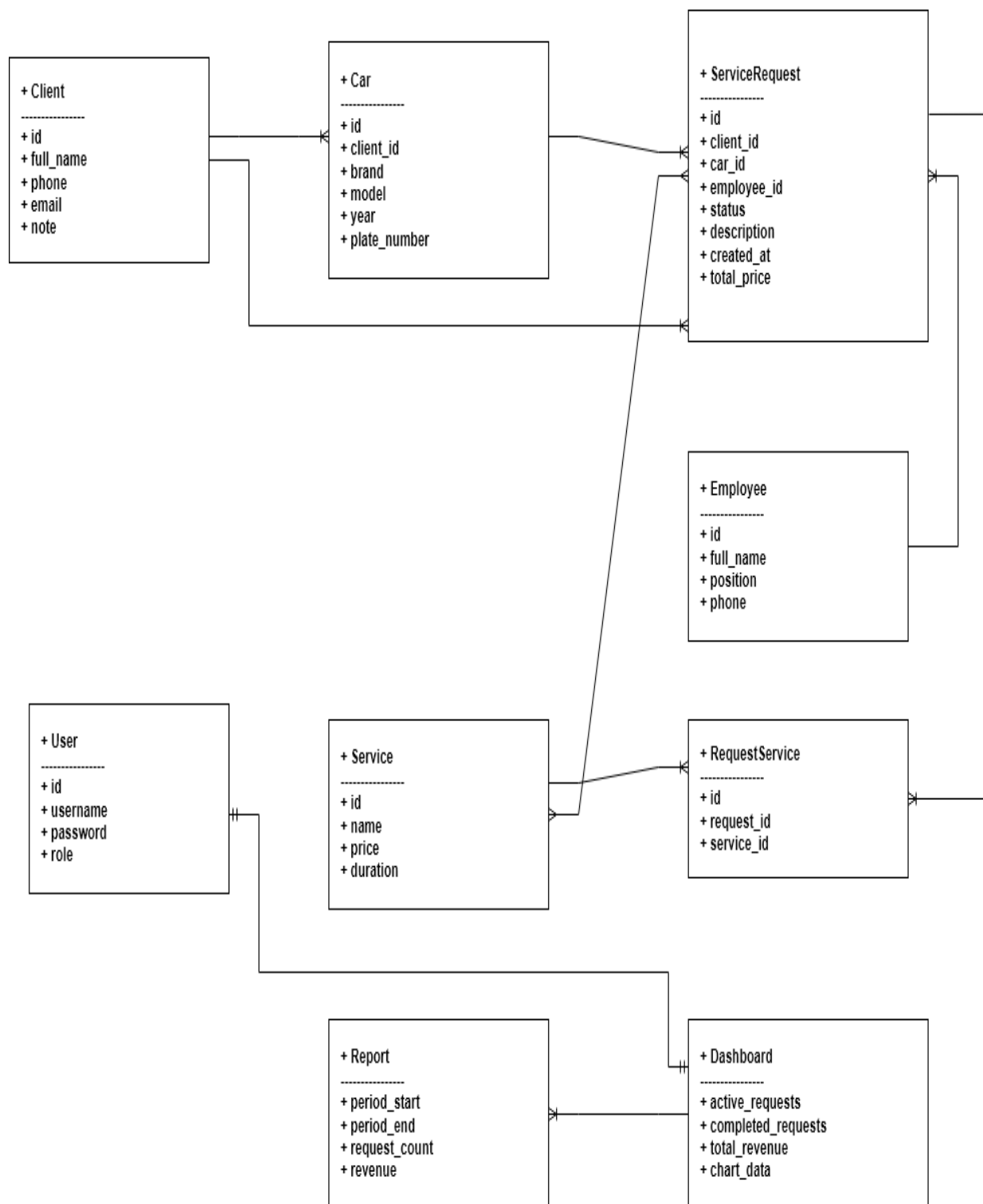
					ПГУ 1.090301.26.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		



ПГУ 1.090301.26.002 Д13									
Имя	Фамилия	Инициалы	Пол	Дата	Вед-приложение мини-СРМ учета и анализ деятельности компании с визуализацией данных и системой отчетности				
Фамилия	Инициалы	Пол	Дата						
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ



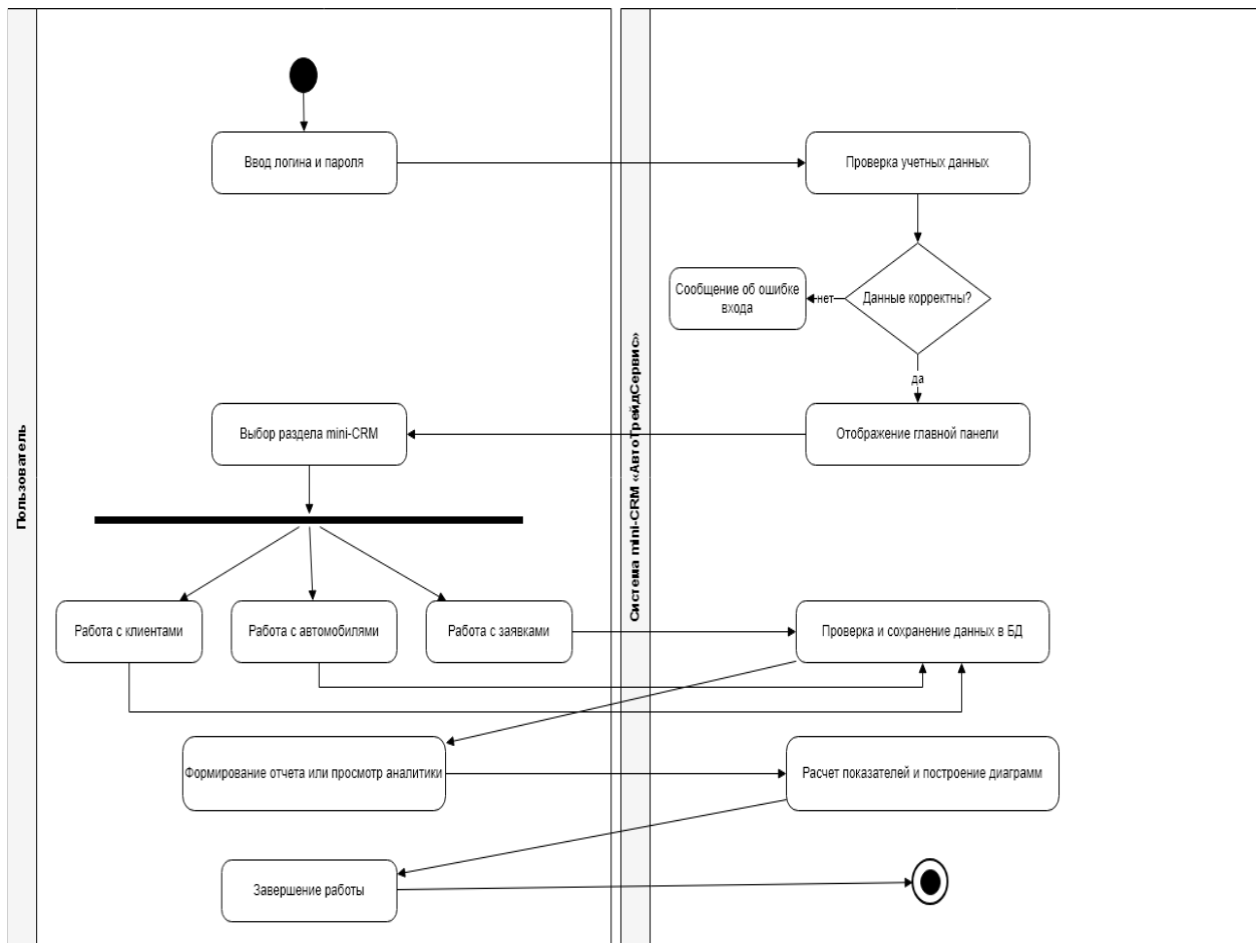
ПГУ 1.090301.26.002 Д11									
Имя	Фамилия	Инициалы	Пол	Дата	Вед-применение: учет и анализ деятельности компании с использованием данных и системной отчетности				
Фамилия	Имя	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					
Имя	Фамилия	Инициалы	Пол	Дата					

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

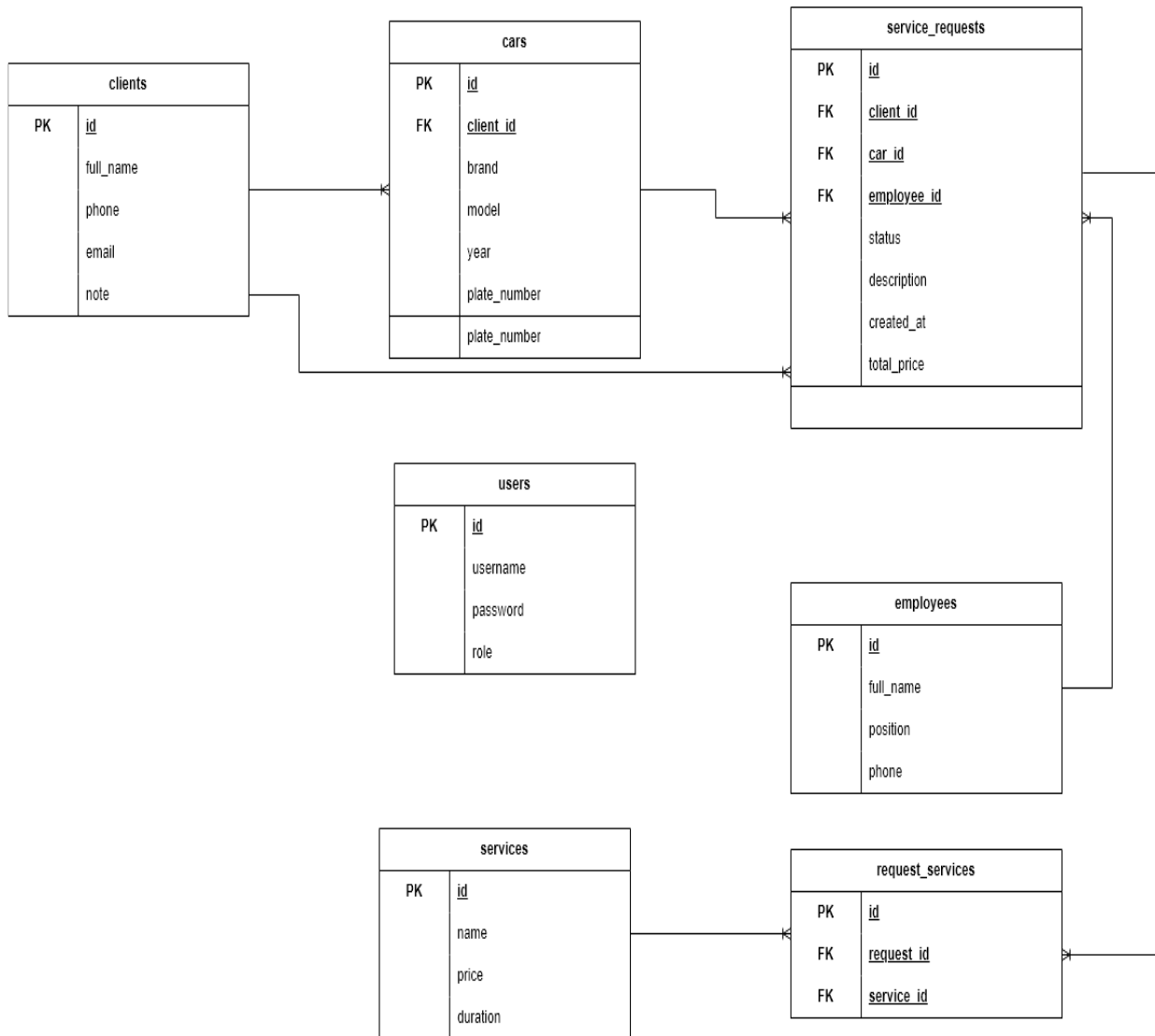
90



ПГУ 1.090301.26.002 Д17					Лист		
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб
Имя	Фамилия	Имя	Фамилия	Дата	Лист	Место	Масштаб

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ



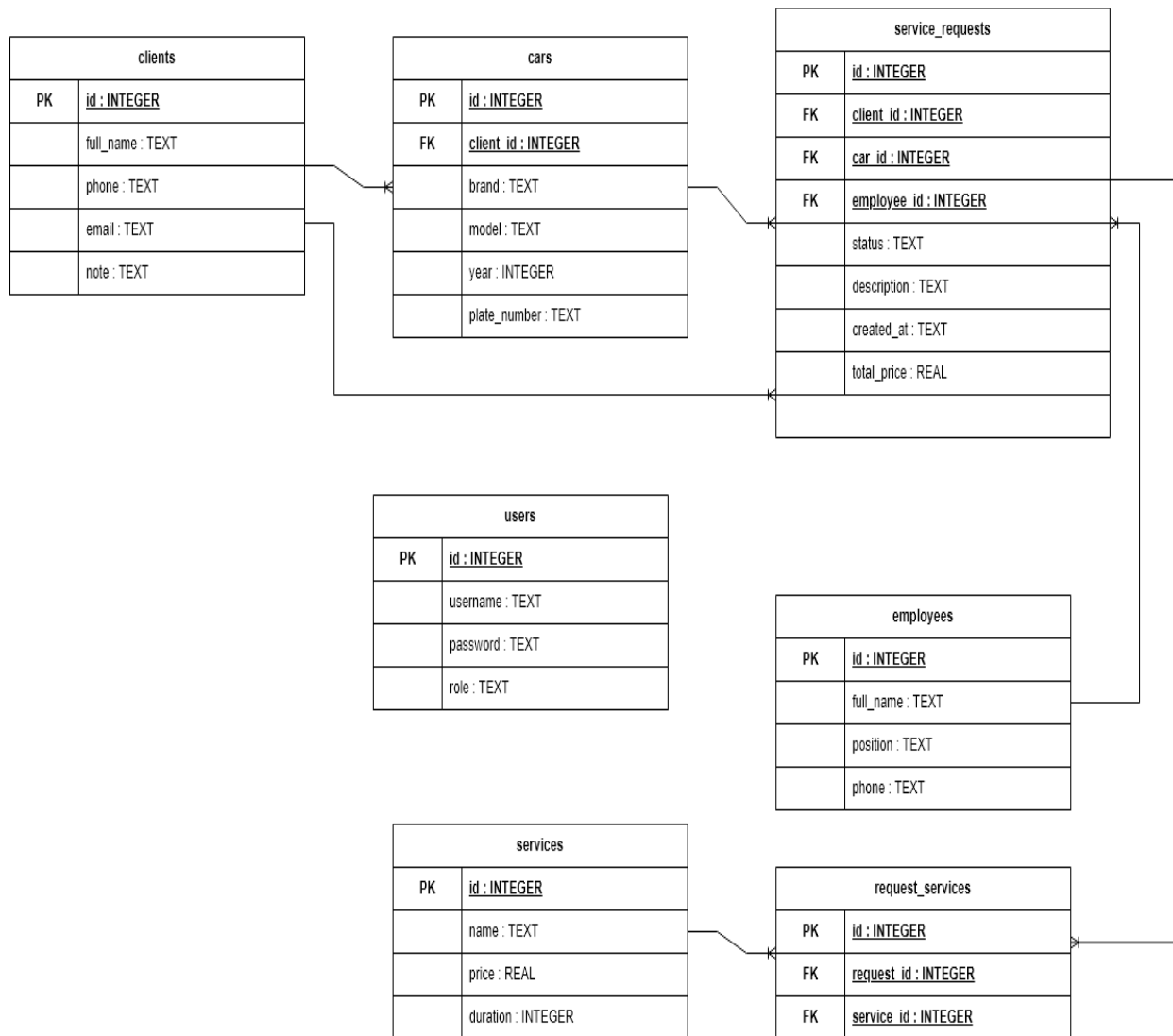
					ПГУ1.090301.26.002 Д6		
Имя	Фамилия	Место	Должность	Дата	Должность	Место	Место
Решен	Павлов Д.Д.						
Созд	Суров Е.И.						
Т. автор							
И. автор	Павлов А.С.						
Упр.	Морозов М.А.						
					Логическая модель БД		
					ФБТ, каф. БТ гр. 22БВ0111		

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

92



ПГУ 1.090301.26.002 Д7									
Имя	Пол	№ докум.	Подпись	Дата	Веб-приложение мнги-СRM учета и анализа деятельности компании с визуализацией данных и системой отчетности				
Фамилия	Имя	Ф.И.О.	Подпись	Дата					
Имя	Пол	№ докум.	Подпись	Дата					
Имя	Пол	№ докум.	Подпись	Дата					
Имя	Пол	№ докум.	Подпись	Дата					
Имя	Пол	№ докум.	Подпись	Дата					
Имя	Пол	№ докум.	Подпись	Дата					
Имя	Пол	№ докум.	Подпись	Дата					
Имя	Пол	№ докум.	Подпись	Дата					

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Приложение Б

(обязательное)

мини-CRM «АвтоТрейдСервис»

Листинги программы

Файл app.py:

```
from flask import Flask, render_template, request, redirect, url_for
import sqlite3
from datetime import datetime

app = Flask(__name__)
DATABASE = "autotrade.db"

def get_db():
    conn = sqlite3.connect(DATABASE)
    conn.row_factory = sqlite3.Row
    return conn

def init_db():
    conn = get_db()
    cur = conn.cursor()

    cur.execute("""
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT NOT NULL,
    password TEXT NOT NULL,
    role TEXT NOT NULL
)
""")

    cur.execute("""
CREATE TABLE IF NOT EXISTS employees (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    position TEXT,
    phone TEXT
)
""")

    cur.execute("""
CREATE TABLE IF NOT EXISTS clients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    full_name TEXT NOT NULL,
    phone TEXT NOT NULL,
    email TEXT,
    note TEXT
)
""")

    cur.execute("""
CREATE TABLE IF NOT EXISTS cars (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    brand TEXT NOT NULL,
    model TEXT NOT NULL,
```

					ПГУ 1.090301.26.001 ПЗ	Лист
						94
Изм.	Лист	№ докум.	Подпись	Дата		

```

        year INTEGER,
        plate_number TEXT,
        FOREIGN KEY (client_id) REFERENCES clients(id)
    )
    """)

cur.execute("""
CREATE TABLE IF NOT EXISTS services (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL NOT NULL,
    duration INTEGER
)
""")

cur.execute("""
CREATE TABLE IF NOT EXISTS service_requests (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    client_id INTEGER NOT NULL,
    car_id INTEGER NOT NULL,
    employee_id INTEGER,
    status TEXT NOT NULL,
    description TEXT,
    created_at TEXT,
    total_price REAL,
    FOREIGN KEY (client_id) REFERENCES clients(id),
    FOREIGN KEY (car_id) REFERENCES cars(id),
    FOREIGN KEY (employee_id) REFERENCES employees(id)
)
""")

cur.execute("""
CREATE TABLE IF NOT EXISTS request_services (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    request_id INTEGER NOT NULL,
    service_id INTEGER NOT NULL,
    FOREIGN KEY (request_id) REFERENCES service_requests(id),
    FOREIGN KEY (service_id) REFERENCES services(id)
)
""")

cur.execute("SELECT COUNT(*) FROM employees")
if cur.fetchone()[0] == 0:
    cur.executemany(
        "INSERT INTO employees (full_name, position, phone) VALUES (?, ?, ?)",
        [
            ("Иванов Сергей Петрович", "Мастер-приемщик", "+7 900 111-22-33"),
            ("Кузнецов Алексей Игоревич", "Автомеханик", "+7 900 222-33-44"),
            ("Смирнова Анна Олеговна", "Администратор", "+7 900 333-44-55")
        ]
    )

cur.execute("SELECT COUNT(*) FROM services")
if cur.fetchone()[0] == 0:
    cur.executemany(
        "INSERT INTO services (name, price, duration) VALUES (?, ?, ?)",
        [
            ("Диагностика двигателя", 2500, 60),
            ("Замена масла", 1800, 40),
            ("Замена тормозных колодок", 3500, 90),
            ("Компьютерная диагностика", 2200, 45),
            ("Проверка подвески", 2000, 50)
        ]
    )

```

```

    )

    cur.execute("SELECT COUNT(*) FROM clients")
    if cur.fetchone()[0] == 0:
        cur.executemany(
            "INSERT INTO clients (full_name, phone, email, note) VALUES (?, ?, ?,
?)",
            [
                ("Петров Андрей Викторович", "+7 900 444-55-66", "petrov@mail.ru",
"Постоянный клиент"),
                ("Соколова Мария Павловна", "+7 900 555-66-77",
"sokolova@mail.ru", "Запись по рекомендации")
            ]
        )

        cur.executemany(
            "INSERT INTO cars (client_id, brand, model, year, plate_number) VALUES
(?, ?, ?, ?, ?)",
            [
                (1, "Toyota", "Camry", 2018, "A123BC77"),
                (2, "Kia", "Rio", 2020, "M456OP77")
            ]
        )

    conn.commit()
    conn.close()

@app.route("/")
def dashboard():
    conn = get_db()
    total_clients = conn.execute("SELECT COUNT(*) FROM clients").fetchone()[0]
    total_cars = conn.execute("SELECT COUNT(*) FROM cars").fetchone()[0]
    total_requests = conn.execute("SELECT COUNT(*) FROM
service_requests").fetchone()[0]
    total_revenue = conn.execute(
        "SELECT COALESCE(SUM(total_price), 0) FROM service_requests"
    ).fetchone()[0]

    status_rows = conn.execute("""
        SELECT status, COUNT(*) AS count
        FROM service_requests
        GROUP BY status
    """).fetchall()

    status_labels = [row["status"] for row in status_rows]
    status_values = [row["count"] for row in status_rows]

    conn.close()
    return render_template(
        "dashboard.html",
        total_clients=total_clients,
        total_cars=total_cars,
        total_requests=total_requests,
        total_revenue=total_revenue,
        status_values=status_values
    )

@app.route("/clients", methods=["GET", "POST"])
def clients():
    conn = get_db()

    if request.method == "POST":

```

```

        conn.execute(
            "INSERT INTO clients (full_name, phone, email, note) VALUES (?, ?, ?,
?)",
            (
                request.form["full_name"],
                request.form["phone"],
                request.form.get("email", ""),
                request.form.get("note", "")
            )
        )
        conn.commit()
        return redirect(url_for("clients"))

    clients_list = conn.execute("SELECT * FROM clients ORDER BY id
DESC").fetchall()
    conn.close()
    return render_template("clients.html", clients=clients_list)

@app.route("/cars", methods=["GET", "POST"])
def cars():
    conn = get_db()

    if request.method == "POST":
        conn.execute(
            "INSERT INTO cars (client_id, brand, model, year, plate_number) VALUES
(?, ?, ?, ?, ?)",
            (
                request.form["client_id"],
                request.form["brand"],
                request.form["model"],
                request.form.get("year") or None,
                request.form.get("plate_number", "")
            )
        )
        conn.commit()
        return redirect(url_for("cars"))

    cars_list = conn.execute("""
        SELECT cars.*, clients.full_name AS client_name
        FROM cars
        JOIN clients ON clients.id = cars.client_id
        ORDER BY cars.id DESC
    """).fetchall()

    clients_list = conn.execute("SELECT * FROM clients ORDER BY
full_name").fetchall()
    conn.close()
    return render_template("cars.html", cars=cars_list, clients=clients_list)

@app.route("/requests", methods=["GET", "POST"])
def requests_page():
    conn = get_db()

    if request.method == "POST":
        service_ids = request.form.getlist("service_ids")
        total_price = 0

        for service_id in service_ids:
            service = conn.execute(
                "SELECT price FROM services WHERE id = ?",
                (service_id,)
            )

```

					ПГУ 1.090301.26.001 ПЗ	Лист
						97
Изм.	Лист	№ докум.	Подпись	Дата		

```

        ).fetchone()
        if service:
            total_price += service["price"]

    cur = conn.execute(
        """
        INSERT INTO service_requests
        (client_id, car_id, employee_id, status, description, created_at,
total_price)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?)
        """,
        (
            request.form["client_id"],
            request.form["car_id"],
            request.form.get("employee_id") or None,
            request.form["status"],
            request.form.get("description", ""),
            datetime.now().strftime("%Y-%m-%d"),
            total_price
        )
    )

    request_id = cur.lastrowid
    for service_id in service_ids:
        conn.execute(
            "INSERT INTO request_services (request_id, service_id) VALUES (?,
?",
            (request_id, service_id)
        )

    conn.commit()
    return redirect(url_for("requests_page"))

requests_list = conn.execute("""
SELECT
    service_requests.*,
    clients.full_name AS client_name,
    cars.brand || ' ' || cars.model || ' (' || cars.plate_number || ')' AS
car_name,
    employees.full_name AS employee_name
FROM service_requests
JOIN clients ON clients.id = service_requests.client_id
JOIN cars ON cars.id = service_requests.car_id
LEFT JOIN employees ON employees.id = service_requests.employee_id
ORDER BY service_requests.id DESC
""").fetchall()

clients_list = conn.execute("SELECT * FROM clients ORDER BY
full_name").fetchall()
cars_list = conn.execute("SELECT * FROM cars ORDER BY brand").fetchall()
employees_list = conn.execute("SELECT * FROM employees ORDER BY
full_name").fetchall()
services_list = conn.execute("SELECT * FROM services ORDER BY
name").fetchall()

conn.close()
return render_template(
    "requests.html",
    requests=requests_list,
    clients=clients_list,
    cars=cars_list,
    employees=employees_list,
    services=services_list

```

```

)

@app.route("/reports")
def reports():
    conn = get_db()

    status_rows = conn.execute("""
        SELECT status, COUNT(*) AS count
        FROM service_requests
        GROUP BY status
    """).fetchall()

    service_rows = conn.execute("""
        SELECT services.name, COUNT(request_services.id) AS count
        FROM request_services
        JOIN services ON services.id = request_services.service_id
        GROUP BY services.name
        ORDER BY count DESC
    """).fetchall()

    conn.close()
    return render_template(
        "reports.html",
        status_labels=[row["status"] for row in status_rows],
        service_labels=[row["name"] for row in service_rows],
        service_values=[row["count"] for row in service_rows]
    )

if __name__ == "__main__":
    init_db()
    app.run(debug=True)

```

Файл base.html:

```

<!doctype html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>АвтоТрейдСервис</title>
    <link
href="https://cdn.jsdelivrivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">
    <link href="{ url_for('static', filename='style.css') }" rel="stylesheet">
    <script src="https://cdn.jsdelivrivr.net/npm/chart.js"></script>
</head>
<body>
<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-4">
    <div class="container-fluid">
        <a class="navbar-brand" href="/">АвтоТрейдСервис</a>
        <div class="navbar-nav">
            <a class="nav-link" href="/">Панель</a>
            <a class="nav-link" href="/clients">Клиенты</a>
            <a class="nav-link" href="/requests">Заявки</a>
            <a class="nav-link" href="/reports">Отчеты</a>
        </div>
    </div>
</nav>

<main class="container-fluid px-4">

```

					ПГУ 1.090301.26.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпись	Дата		

```

    {% block content %}{% endblock %}
</main>
</body>
</html>

```

Файл dashboard.html:

```

{% extends "base.html" %}
{% block content %}
<h2 class="mb-4">Панель управления</h2>

<div class="row g-3 mb-4">
  <div class="col-md-3">
    <div class="card stat-card">
      <div class="card-body">
        <div class="text-muted">Клиенты</div>
        <h3>{{ total_clients }}</h3>
      </div>
    </div>
  </div>
  <div class="col-md-3">
    <div class="card stat-card">
      <div class="card-body">
        <div class="text-muted">Автомобили</div>
        <h3>{{ total_cars }}</h3>
      </div>
    </div>
  </div>
  <div class="col-md-3">
    <div class="card stat-card">
      <div class="card-body">
        <div class="text-muted">Заявки</div>
        <h3>{{ total_requests }}</h3>
      </div>
    </div>
  </div>
  <div class="col-md-3">
    <div class="card stat-card">
      <div class="card-body">
        <div class="text-muted">Выручка, руб.</div>
        <h3>{{ "%.0f"|format(total_revenue) }}</h3>
      </div>
    </div>
  </div>
</div>

<div class="card">
  <div class="card-body">
    <h5>Заявки по статусам</h5>
    <canvas id="statusChart" height="90"></canvas>
  </div>
</div>

<script>
const statusLabels = {{ status_labels|tojson }};
const statusValues = {{ status_values|tojson }};

new Chart(document.getElementById("statusChart"), {
  type: "bar",
  data: {
    labels: statusLabels,
    datasets: [{

```

					ПГУ 1.090301.26.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

```

        label: "Количество заявок",
        data: statusValues,
        backgroundColor: ["#2F6DB2", "#6AA84F", "#F1C232", "#C00000"]
    }
},
options: {
    responsive: true,
    plugins: { legend: { display: false } }
}
});
</script>
{% endblock %}

```

Файл clients.html:

```

{% extends "base.html" %}
{% block content %}
<h2 class="mb-4">Клиенты</h2>

<div class="card mb-4">
    <div class="card-body">
        <form method="post" class="row g-3">
            <div class="col-md-3">
                <input class="form-control" name="full_name" placeholder="ФИО
клиента" required>
            </div>
            <div class="col-md-2">
                <input class="form-control" name="phone" placeholder="Телефон"
required>
            </div>
            <div class="col-md-3">
                <input class="form-control" name="email" placeholder="Email">
            </div>
            <div class="col-md-3">
                <input class="form-control" name="note" placeholder="Примечание">
            </div>
            <div class="col-md-1">
                <button class="btn btn-primary w-100">+</button>
            </div>
        </form>
    </div>
</div>

<table class="table table-bordered table-hover bg-white">
    <thead>
        <tr>
            <th>ID</th>
            <th>ФИО</th>
            <th>Телефон</th>
            <th>Email</th>
            <th>Примечание</th>
        </tr>
    </thead>
    <tbody>
        {% for client in clients %}
            <tr>
                <td>{{ client.id }}</td>
                <td>{{ client.full_name }}</td>
                <td>{{ client.phone }}</td>
                <td>{{ client.email }}</td>
            </tr>
        {% endfor %}
    </tbody>
</table>

```

```

        </tr>
    {% endfor %}
</tbody>
</table>
{% endblock %}

```

Файл cars.html:

```

{% extends "base.html" %}
{% block content %}
<h2 class="mb-4">Автомобили</h2>

<div class="card mb-4">
    <div class="card-body">
        <form method="post" class="row g-3">
            <div class="col-md-3">
                <select class="form-select" name="client_id" required>
                    {% for client in clients %}
                        <option value="{{ client.id }}">{{ client.full_name
                    }}</option>
                    {% endfor %}
                </select>
            </div>
            <div class="col-md-2">
                <input class="form-control" name="brand" placeholder="Марка"
required>
            </div>
            <div class="col-md-2">
                <input class="form-control" name="model" placeholder="Модель"
required>
            </div>
            <div class="col-md-2">
                <input class="form-control" name="year" placeholder="Год">
            </div>
            <div class="col-md-2">
                <input class="form-control" name="plate_number"
placeholder="Госномер">
            </div>
            <div class="col-md-1">
                <button class="btn btn-primary w-100">+</button>
            </div>
        </form>
    </div>
</div>

<table class="table table-bordered table-hover bg-white">
    <thead>
        <tr>
            <th>ID</th>
            <th>Клиент</th>
            <th>Марка</th>
            <th>Год</th>
            <th>Госномер</th>
        </tr>
    </thead>
    <tbody>
        {% for car in cars %}
            <tr>
                <td>{{ car.id }}</td>
                <td>{{ car.client_name }}</td>
            </tr>
        </tbody>
    </table>

```

```

        <td>{{ car.brand }}</td>
        <td>{{ car.model }}</td>
        <td>{{ car.year }}</td>
        <td>{{ car.plate_number }}</td>
    </tr>
    {% endfor %}
</tbody>
</table>
{% endblock %}

```

Файл requests.html:

```

{% extends "base.html" %}
{% block content %}
<h2 class="mb-4">Заявки</h2>

<div class="card mb-4">
    <div class="card-body">
        <form method="post" class="row g-3">
            <div class="col-md-2">
                <select class="form-select" name="client_id" required>
                    {% for client in clients %}
                        <option value="{{ client.id }}">{{ client.full_name }}</option>
                    {% endfor %}
                </select>
            </div>
            <div class="col-md-2">
                <select class="form-select" name="car_id" required>
                    {% for car in cars %}
                        <option value="{{ car.id }}">{{ car.brand }} {{ car.model }}</option>
                    {% endfor %}
                </select>
            </div>
            <div class="col-md-2">
                <select class="form-select" name="employee_id">
                    {% for employee in employees %}
                        <option value="{{ employee.id }}">{{ employee.full_name }}</option>
                    {% endfor %}
                </select>
            </div>
            <div class="col-md-2">
                <select class="form-select" name="status">
                    <option>Новая</option>
                    <option>В работе</option>
                    <option>Ожидает запчастей</option>
                    <option>Завершена</option>
                </select>
            </div>
            <div class="col-md-2">
                <select class="form-select" name="service_ids" multiple>
                    {% for service in services %}
                        <option value="{{ service.id }}">{{ service.name }}</option>
                    {% endfor %}
                </select>
            </div>
            <div class="col-md-1">
                <input class="form-control" name="description"
placeholder="Описание">
            </div>
        </form>
    </div>
</div>

```

```

        <div class="col-md-1">
            <button class="btn btn-primary w-100">+</button>
        </div>
    </form>
</div>
</div>

<table class="table table-bordered table-hover bg-white">
    <thead>
        <tr>
            <th>ID</th>
            <th>Клиент</th>
            <th>Автомобиль</th>
            <th>Сотрудник</th>
            <th>Статус</th>
            <th>Дата</th>
            <th>Сумма</th>
        </tr>
    </thead>
    <tbody>
        {% for item in requests %}
            <tr>
                <td>{{ item.id }}</td>
                <td>{{ item.client_name }}</td>
                <td>{{ item.car_name }}</td>
                <td>{{ item.employee_name }}</td>
                <td>{{ item.status }}</td>
                <td>{{ item.created_at }}</td>
                <td>{{ item.total_price }}</td>
            </tr>
        {% endfor %}
    </tbody>
</table>
{% endblock %}

```

Файл reports.html:

```

{% extends "base.html" %}
{% block content %}
<h2 class="mb-4">Отчеты</h2>

<div class="row g-4">
    <div class="col-md-6">
        <div class="card">
            <div class="card-body">
                <h5>Заявки по статусам</h5>
                <canvas id="statusChart"></canvas>
            </div>
        </div>
    </div>
    <div class="col-md-6">
        <div class="card">
            <div class="card-body">
                <h5>Популярность услуг</h5>
                <canvas id="serviceChart"></canvas>
            </div>
        </div>
    </div>
</div>

<script>
const statusLabels = {{ status_labels|tojson }};

```

```

const statusValues = {{ status_values|tojson }};
const serviceLabels = {{ service_labels|tojson }};
const serviceValues = {{ service_values|tojson }};

new Chart(document.getElementById("statusChart"), {
  type: "doughnut",
  data: {
    labels: statusLabels,
    datasets: [{
      data: statusValues,
      backgroundColor: ["#2F6DB2", "#6AA84F", "#F1C232", "#C00000"]
    }]
  }
});

new Chart(document.getElementById("serviceChart"), {
  type: "bar",
  data: {
    labels: serviceLabels,
    datasets: [{
      label: "Количество",
      data: serviceValues,
      backgroundColor: "#2F6DB2"
    }]
  },
  options: {
    plugins: { legend: { display: false } }
  }
});
</script>
{% endblock %}

```

Файл style.css:

```

body {
  background: #f4f6f9;
  font-family: Arial, sans-serif;
}

.navbar-brand {
  font-weight: 700;
}

.card {
  border: none;
  border-radius: 14px;
  box-shadow: 0 6px 18px rgba(0, 0, 0, 0.08);
}

.stat-card h3 {
  margin-top: 8px;
  font-weight: 700;
  color: #2F6DB2;
}

.table {
  border-radius: 10px;
  overflow: hidden;
}

h2 {
  font-weight: 700;
}

```

					ПГУ 1.090301.26.001 ПЗ	Лист
						105
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(обязательное)

Мини-CRM «АвтоТрейдСервис» для учета и анализа деятельности компании с визуализацией данных и системой отчетности

Презентация

БАКАЛАВРСКАЯ РАБОТА

мини-CRM
АвтоТрейдСервис

Веб-приложение мини-CRM учета и анализа деятельности компании с визуализацией данных и системой отчетности

Автор: студент группы 22ВВП1 Шатин Данила Дмитриевич
Руководитель: д.т.н., профессор кафедры «Вычислительная техника» Гурин Евгений Иванович
Пенза, 2026

ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

1

Актуальность разработки

Проблемы ручного и разрозненного учета

данные клиентов, автомобилей и заявок хранятся отдельно
заявки трудно быстро искать и контролировать
история обслуживания не всегда связана с клиентом
отчеты и показатели собираются вручную
руководитель не видит полную картину работы

Решение: единая мини-CRM связывает учетные операции и аналитику.

ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

```
graph TD; A[Обращение клиента] --> B[1. Прием обращения]; B --> C[Запись в журнал или таблицу]; C --> D[2. Фиксация данных]; D --> E[Передача сведений мастеру устно]; E --> F[3. Передача задачи]; F --> G[Статус заявки контролируется вручную]; G --> H[4. Контроль работ]; H --> I[Отчет собирается после смены]; I --> J[5. Ручная отчетность]; J --> K[Риски: потери данных, ошибки, задержки];
```

2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

106

Цель и задачи работы

Цель: разработать веб-приложение мини-CRM «АвтоТрейдСервис» для учета клиентов, автомобилей, заявок, анализа деятельности компании, визуализации данных и формирования отчетности.

Объект исследования

деятельность автосервисной компании по учету клиентов, автомобилей, заявок и результатов оказания услуг

Предмет исследования

проектирование и разработка веб-приложения мини-CRM для автоматизации учета и анализа деятельности

- 1 проанализировать предметную область и существующие аналоги
- 2 обосновать выбор языка программирования, СУБД и средств разработки
- 3 спроектировать архитектуру приложения и модель базы данных
- 4 реализовать мини-CRM на Python, Flask, SQLite, Bootstrap и Chart.js
- 5 проверить работу основных функций: клиенты, автомобили, заявки, услуги и отчеты

ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

3

Анализ существующих аналогов

YCLIENTS

онлайн-запись, клиентская база, расписание, уведомления

АвтоДилер Онлайн

учет ремонтов, заказ-наряды, история обслуживания

Универсальные CRM

клиенты, сделки, задачи, маркетинг и интеграции

Таблицы

простые записи без связей и автоматической аналитики

Вывод после анализа

готовые CRM-системы часто перегружены лишними модулями
специализированные решения зависят от коммерческой платформы
электронные таблицы не обеспечивают целостную историю и отчеты
для ВКР выбран компактный вариант: учет клиентов, автомобилей, заявок, услуг и отчетности

ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

4

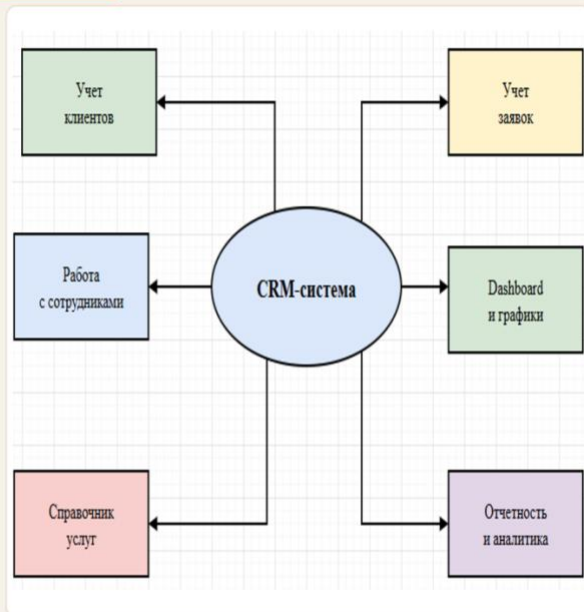
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

107

Требования и основные модули



Функциональные требования

авторизация пользователя
ведение клиентской базы
учет автомобилей клиентов
создание и обработка заявок
справочник услуг и расчет стоимости
формирование отчетов и графиков

Клиент → автомобиль → заявка → услуги → отчетность

ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

5

Стек технологий

Python

основная логика приложения

Flask

маршруты, формы и серверная часть

SQLite

локальное хранение данных

HTML / CSS / Bootstrap

страницы, таблицы и адаптивный интерфейс

Chart.js

диаграммы и отчетные графики

Visual Studio Code

разработка и проверка проекта

Выбранный стек позволяет создать понятное веб-приложение без отдельного сервера баз данных и без сложной инфраструктуры.

ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

6

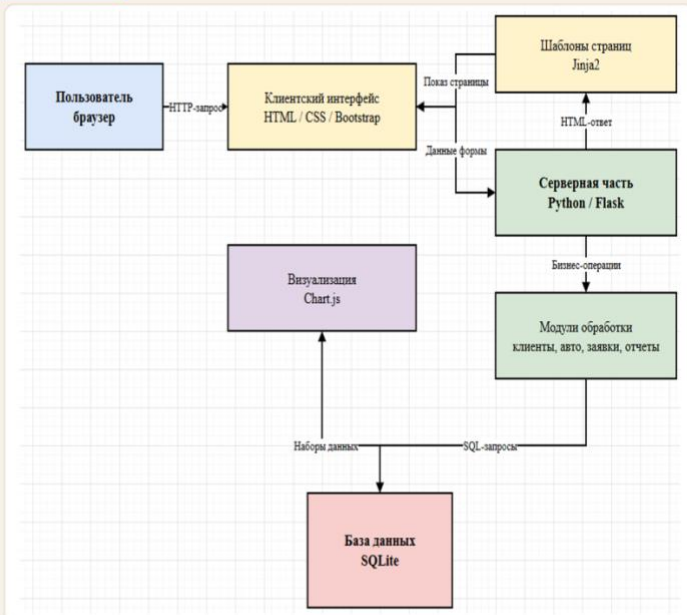
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

108

Архитектура приложения



ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

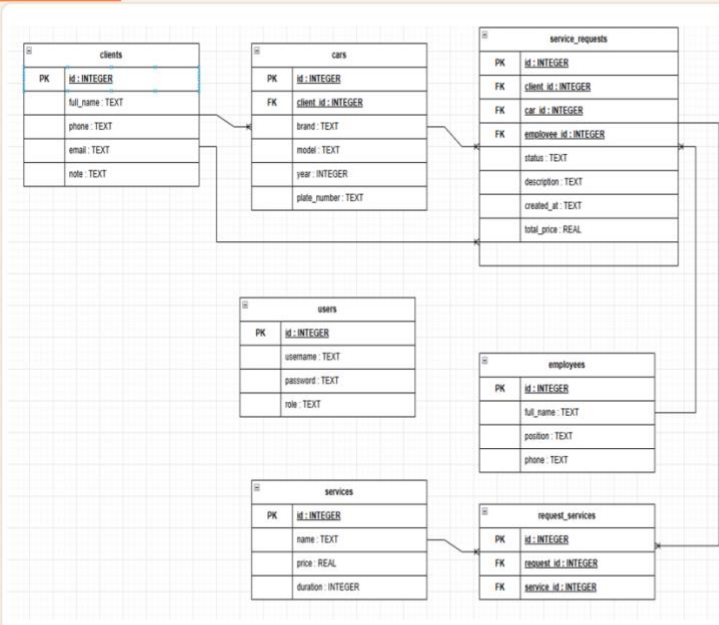
Ключевой принцип

пользователь работает через браузер
Flask принимает запросы и обрабатывает формы
Jinja2 формирует HTML-страницы
SQLite хранит клиентов, автомобили, заявки и услуги
Chart.js отображает графики на панели отчетов

Разделение: интерфейс · логика · данные

7

Проектирование базы данных



ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

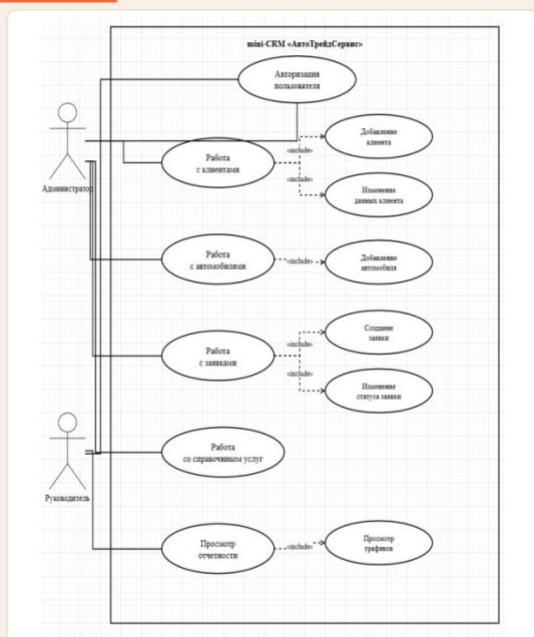
Основные сущности

users — пользователи и авторизация
clients — клиентская база
cars — автомобили клиентов
employees — сотрудники
services — справочник услуг
service_requests — заявки
request_services — связь заявок и услуг

Внешние ключи обеспечивают целостность

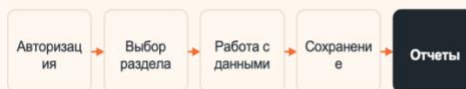
8

Пользовательские сценарии



ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

Работа пользователя в системе

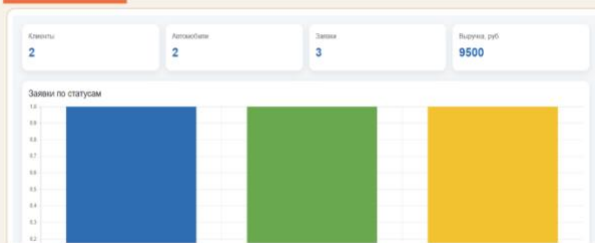
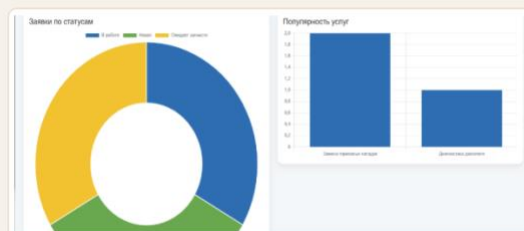


администратор регистрирует клиента и автомобиль
сотрудник оформляет заявку и выбирает услуги
ответственный меняет статус выполнения работ
руководитель просматривает отчетные показатели

Основной сценарий: оформление заявки

9

Пользовательский интерфейс и аналитика

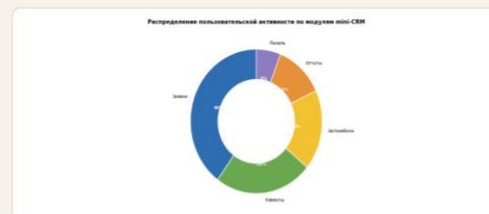


АвтоТрейдСервис

Заявки

Петров Андрей Витальевич, Kia Rio (B4550P77), Иванов Сергей Петрович, Новая, Диагностика двигателя, Замена масла, Замена тормозных колодок, Колесный балансировщик

ID	Клиент	Автомобиль	Сотрудник	Статус	Дата	Сумма
3	Петров Андрей Витальевич	Kia Rio (B4550P77)	Кузнецов Алексей Игоревич	Ожидает запчасти	2026-05-10	3500.0
2	Сидорова Мария Павловна	Toyota Camry (A123BC77)	Кузнецов Алексей Игоревич	В работе	2026-05-10	3500.0
1	Петров Андрей Витальевич	Kia Rio (B4550P77)	Иванов Сергей Петрович	Новая	2026-05-10	2500.0



Интерфейс ориентирован на быстрые действия: таблицы, формы, поиск, статусы заявок и наглядные графики.

ПГУ · ВКР · мини-CRM «АвтоТрейдСервис»

10

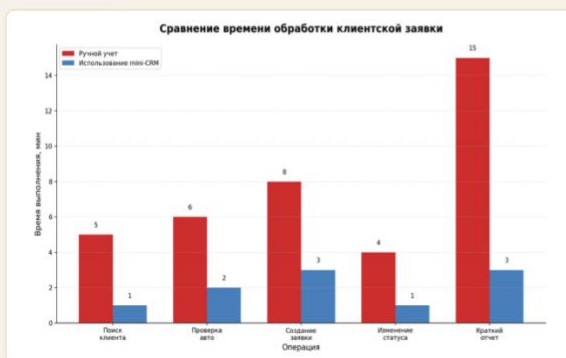
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.26.001 ПЗ

Лист

110

Тестирование и результат работы



Проверенные сценарии

запуск Flask-приложения и создание базы SQLite
добавление клиентов, автомобилей и заявок
расчет итоговой стоимости работ
сохранность данных после перезапуска
отображение графиков и отчетных показателей
обработка неполных и ошибочных данных

Результат: приложение запускается, сохраняет данные, формирует заявки, рассчитывает стоимость работ и отображает отчетные графики.

Итоги ВКР и перспективы развития

В ходе работы:

выполнен анализ предметной области и существующих решений
спроектирована архитектура веб-приложения
разработана структура базы данных SQLite
реализованы модули клиентов, автомобилей, заявок, услуг и отчетности
создан пользовательский веб-интерфейс
проведено функциональное тестирование

Перспективы развития

разграничение ролей пользователей
экспорт отчетов в Excel/PDF
резервное копирование базы данных
расширение аналитики по сотрудникам и услугам
развертывание на локальном сервере компании

Цель выпускной квалификационной работы достигнута, поставленные задачи выполнены.

**СПАСИБО
ЗА ВНИМАНИЕ!**

ПГУ - ВКР - мини-CRM «АвтоТрейдСервис»

13

					ПГУ 1.090301.26.001 ПЗ	Лист
						112
Изм.	Лист	№ докум.	Подпись	Дата		