СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc199362103)

[ГЛАВА 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 5](#_Toc199362104)

[1.1 Исследование предметной области 5](#_Toc199362105)

[1.2 Виды и типы веб-приложений 6](#_Toc199362106)

[1.3 Выбор языка программирования для веб-приложения 10](#_Toc199362107)

[1.4 Выбор среды разработки 12](#_Toc199362108)

[1.5 Выбор архитектуры 15](#_Toc199362109)

[ГЛАВА 2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ 19](#_Toc199362110)

[2.1 Прототип интерфейса веб-приложения 19](#_Toc199362111)

[2.2 Описание алгоритма работы веб-приложения 22](#_Toc199362112)

[2.3 Создание прототипа серверной логики веб-приложения 24](#_Toc199362113)

[2.4 Тестирование веб-приложения 27](#_Toc199362114)

[ГЛАВА 3 ЭКОНОМИЧЕСКАЯ ЧАСТЬ 31](#_Toc199362115)

[3.1 Анализ текущего состояния рынка решений для приема и управление заказами клиентов на продажу автотехники. 31](#_Toc199362116)

[3.2 Анализ жизненного цикла веб приложения для приема и управление заказами клиентов на продажу автотехники. 35](#_Toc199362117)

[ЗАКЛЮЧЕНИЕ 42](#_Toc199362118)

[СПИСОК ЛИТЕРАТУРЫ 43](#_Toc199362119)

ВВЕДЕНИЕ

В условиях стремительной цифровизации бизнеса и повышения требований клиентов к качеству обслуживания возрастает необходимость внедрения современных информационных систем, способных обеспечить эффективное управление процессами продаж и взаимодействия с клиентами. Особенно актуально это для сферы продажи автотехники, где своевременная обработка заказов и точность данных играют ключевую роль в поддержании конкурентоспособности компании.

Компания «Русбизнесавто», являясь одним из ведущих игроков рынка автотехники, сталкивается с рядом проблем, связанных с устаревшими методами учета заказов и управления клиентской базой. Использование фрагментарных и неинтегрированных решений ведет к дублированию информации, снижению скорости обработки заказов и затруднениям в оперативном принятии управленческих решений. В таких условиях разработка современного веб-приложения, обеспечивающего централизованный учет и эффективное управление заказами, становится стратегической задачей.

Целью настоящей работы является создание веб-приложения для приема и управления заказами клиентов компании «Русбизнесавто», способного повысить прозрачность и скорость обработки заказов, а также улучшить взаимодействие между клиентами и сотрудниками компании.

Для достижения поставленной цели в работе решаются следующие задачи:

— разработка функционала приема и обработки заказов клиентов через удобный веб-интерфейс с последующей передачей информации в базу данных;

— обеспечение централизованного хранения данных о заказах, клиентах и ассортименте автотехники для их анализа и обработки;

— гарантирование надежности и целостности данных, включая защиту от потери информации и несанкционированного доступа;

— создание удобного и интуитивно понятного веб-интерфейса как для клиентов, так и для сотрудников компании.

Реализация данного проекта позволит повысить эффективность бизнес-процессов компании «Русбизнесавто», улучшить качество обслуживания клиентов и обеспечить устойчивое развитие на конкурентном рынке автотехники в условиях цифровой трансформации.

ГЛАВА 1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

* 1. Исследование предметной области

Сфера продаж автотехники характеризуется множеством параллельных процессов, охватывающих как внешние коммуникации с клиентами, так и внутренние бизнес-операции: обработку заказов, контроль остатков, взаимодействие с поставщиками, формирование отчетности и многое другое. В условиях современной конкуренции и ускоренного темпа цифровизации даже крупные компании, такие как «Русбизнесавто», сталкиваются с необходимостью переосмысления подходов к управлению своими информационными потоками.

Традиционные методы ведения учёта — бумажные документы, таблицы Excel, локальные базы данных без сетевого взаимодействия — перестают соответствовать требованиям оперативности, точности и надёжности. Особенно остро это проявляется при увеличении объёмов заказов, расширении клиентской базы и росте ассортимента. При ручной обработке данных возрастает риск дублирования информации, ошибки становятся трудноотслеживаемыми, а рабочее время сотрудников расходуется нерационально. Всё это отрицательно сказывается на качестве обслуживания и конкурентоспособности компании.

В этих условиях ключевым становится внедрение веб-решений, способных объединить все основные процессы в единую информационную систему. Веб-приложения позволяют централизовать хранение данных, обеспечить доступ к системе в режиме 24/7, разграничить роли пользователей и ускорить коммуникации между отделами. Особенно важно, что подобные решения позволяют не просто автоматизировать уже существующие действия, но и повысить прозрачность и управляемость всей цепочки продаж.

Ниже представлена таблица, отражающая основные процессы в компании по продаже автотехники и выгоды, которые может дать их автоматизация.

Таблица 1 - Автоматизируемые процессы и ожидаемый эффект

|  |  |  |
| --- | --- | --- |
| **Процесс** | **Состояние без автоматизации** | **Эффект от внедрения веб-приложения** |
| Прием заказов | Телефон, e-mail, вручную в таблицы | Единая форма заявки с моментальной записью в БД |
| Хранение информации о клиентах | Разрозненные документы и таблицы | Централизованная клиентская база |
| Работа с ассортиментом | Неактуальные остатки, ручное обновление | Синхронизация с базой, фильтрация по категориям |
| Доступ к заказам для сотрудников | Передача вручную или по почте | Быстрый доступ в личном кабинете по ролям |
| Контроль выполнения заказов | Неудобный отслеживающий процесс | Статусы заказов в режиме реального времени |
| Анализ продаж и клиентской активности | Отчеты вручную, низкая точность | Формирование сводной аналитики по заказам и клиентам |

Таким образом, разработка веб-приложения для приёма и управления заказами является не просто желательной, а стратегически необходимой. Это решение даёт возможность оптимизировать ключевые бизнес-процессы, улучшить клиентский сервис и обеспечить рост управляемости на всех уровнях.

В рамках данной работы будет реализовано веб-приложение, призванное объединить клиентский интерфейс и административную панель в единую систему. Оно должно решать следующие задачи: приём и обработка заказов, хранение информации о клиентах и товарных позициях, защита данных, а также обеспечение удобства и понятности для конечных пользователей — как со стороны клиентов, так и сотрудников компании.

* 1. Виды и типы веб-приложений

Веб-приложения представляют собой программные решения, функционирующие в браузере и доступные пользователю через интернет. В отличие от традиционных настольных программ, веб-приложения не требуют установки на устройствах конечных пользователей, что делает их особенно удобными для работы в корпоративной среде, где важна быстрая масштабируемость и удалённый доступ.

Современные веб-приложения разрабатываются с учётом специфики решаемых задач, числа пользователей, уровня интерактивности и требований к безопасности. На практике это привело к формированию нескольких типовых архитектур (Рисунок 1), отличающихся по принципу взаимодействия между клиентской частью и сервером, сложности реализации и пользовательскому опыту.

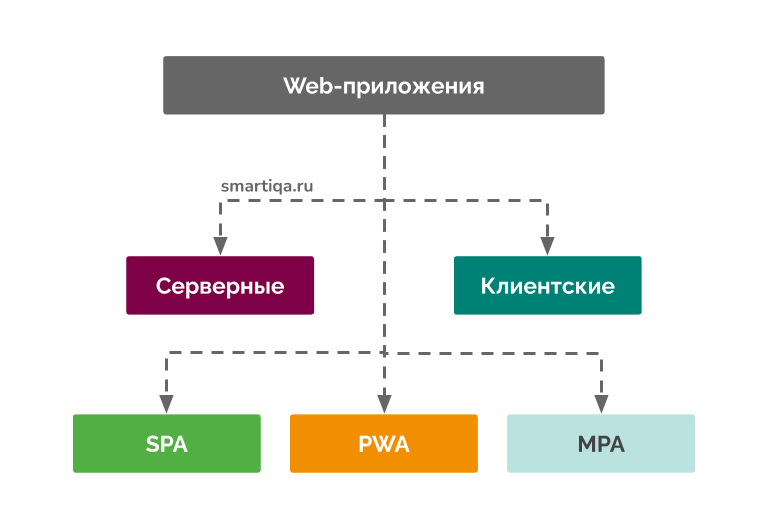


Рисунок 1 – Виды и типы веб-приложений

Самыми простыми формами являются так называемые многостраничные приложения (MPA), где каждое действие пользователя сопровождается полной перезагрузкой страницы. Они характерны для ранних поколений веб-сервисов и до сих пор применяются в проектах с простой логикой или при ограниченных ресурсах разработки.

Более современный подход — одностраничные приложения (SPA). В них основная логика загружается один раз, после чего обмен данными с сервером происходит асинхронно, без перезагрузки интерфейса. Это позволяет добиться высокой скорости работы, плавности взаимодействия и отзывчивости интерфейса, что особенно важно для систем, ориентированных на постоянное взаимодействие с пользователем — например, CRM или панели заказов.

Также стоит отметить прогрессивные веб-приложения (PWA) — гибридный подход, сочетающий возможности классических сайтов и нативных мобильных приложений. Такие решения позволяют работать даже при нестабильном соединении и могут устанавливаться на устройства как обычные приложения.

Для решения задач компании «Русбизнесавто» оптимальным является подход SPA, реализованный с использованием фреймворков, поддерживающих быстрое отображение данных и интеграцию с серверной логикой. Это позволит обеспечить мгновенный отклик интерфейса, что критично для взаимодействия с заказами, списками техники и информацией о клиентах. Кроме того, современное веб-приложение должно учитывать вопросы масштабируемости, адаптивной верстки и безопасности, так как предполагается использование с разных устройств, в том числе за пределами внутренней сети компании. Подход SPA очень востребован на данный промежуток разработки приложения, что показывает конкурентоспособность и новизну данной модели (Рисунок 2):

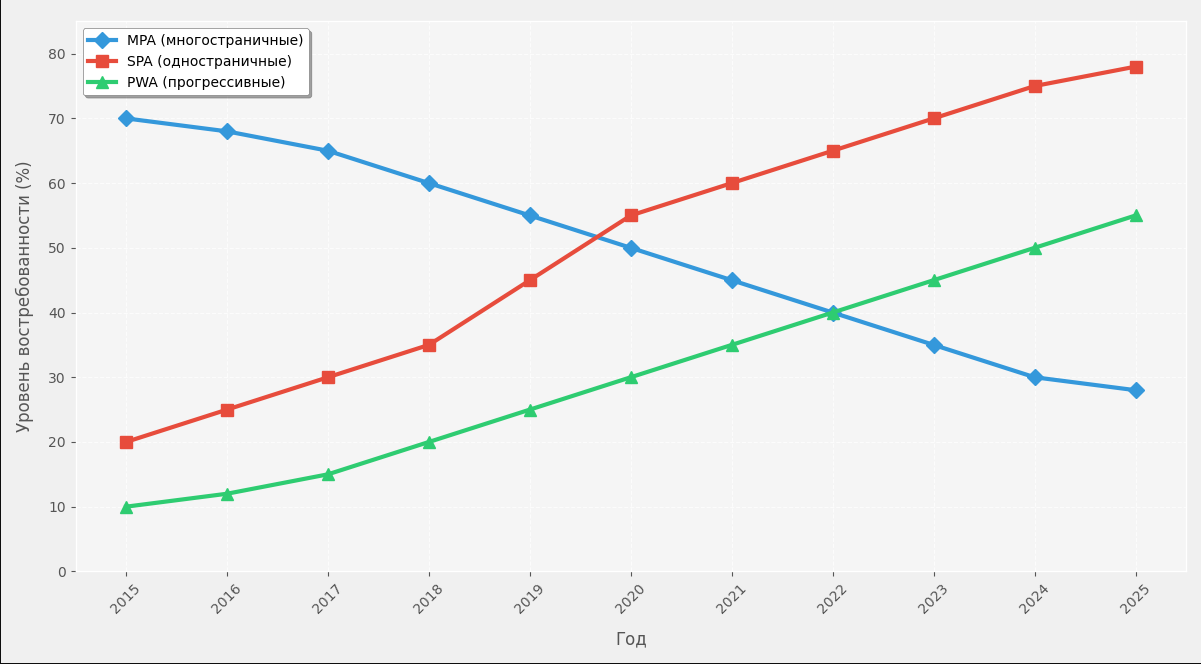


Рисунок 2 - Динамика востребованности типов веб-приложений (2015-2025)

Таким образом, выбор архитектурного подхода и типа веб-приложения напрямую влияет не только на удобство использования, но и на эксплуатационные характеристики системы — устойчивость к нагрузкам, гибкость масштабирования, скорость обработки запросов и легкость последующего сопровождения. Эти факторы особенно важны для бизнеса, связанного с продажей сложной и дорогостоящей техники, где ошибки в обработке заказов или сбои в работе интерфейса могут привести к значительным репутационным и финансовым потерям.

В контексте автоматизации процессов компании «Русбизнесавто» важно обеспечить высокую доступность сервиса и простоту взаимодействия с системой как для клиентов, оформляющих заказы, так и для сотрудников, обрабатывающих поступающую информацию. Применение SPA позволяет минимизировать количество лишних переходов, сократить время загрузки страниц и предоставить более интуитивный пользовательский опыт, в том числе на мобильных устройствах.

Особую роль в разработке таких решений играет качество клиент-серверного взаимодействия, где критично не только быстрое отображение данных, но и их актуальность, безопасность и отказоустойчивость. Поэтому реализация даже простого с точки зрения интерфейса веб-приложения требует продуманного архитектурного проектирования и использования проверенных технологических решений.

* 1. Выбор языка программирования для веб-приложения

Разработка веб-приложения для компании «Русбизнесавто», связанного с приёмом и обработкой заказов на автотехнику, требует применения языка программирования, обеспечивающего стабильную серверную логику, безопасность обработки данных, а также удобную интеграцию с базой данных и интерфейсом пользователя. Среди множества современных решений наибольшее распространение получили три языка: Python, JavaScript (Node.js) и PHP. Каждый из них обладает своими сильными и слабыми сторонами, которые необходимо учитывать при выборе технологического стека.

Python — универсальный язык программирования, активно применяемый в веб-разработке благодаря таким фреймворкам, как Django и Flask. Он отличается лаконичным синтаксисом, высокой читаемостью кода и богатой экосистемой. Фреймворк Django особенно популярен для создания административных панелей, CRM-систем, платформ электронной коммерции и других проектов, требующих высокой надёжности и масштабируемости. Встроенная ORM (Object-Relational Mapping), готовые решения для авторизации, работы с формами и безопасности делают его удобным для быстрой реализации сложной бизнес-логики.

Node.js, основанный на языке JavaScript, позволяет использовать единый язык на клиенте и сервере. Его сильной стороной является высокая скорость обработки входящих запросов за счёт событийно-ориентированной архитектуры. Это делает Node.js особенно привлекательным для приложений с высокой нагрузкой, например, мессенджеров или real-time платформ. Однако при необходимости строгой бизнес-логики и работы с реляционными базами данных его применение может потребовать дополнительных усилий и внешних библиотек, не всегда интуитивных в поддержке.

PHP, ранее считавшийся стандартом веб-разработки, в последние годы утратил позиции. Несмотря на наличие мощных решений вроде Laravel, его архитектура считается устаревшей, а подходы к безопасности и модульности — менее гибкими по сравнению с Python или Node.js. PHP всё ещё широко используется в поддержке старых проектов и CMS, но для создания современных масштабируемых систем его выбирают всё реже.

Сравнительный анализ показал (Рисунок 3), что язык Python в связке с фреймворком Django предоставляет наиболее подходящие условия для разработки веб-приложения, ориентированного на приём и обработку заказов в компании «Русбизнесавто». Его преимущества заключаются не только в простоте синтаксиса и широком наборе встроенных инструментов, но и в зрелости самого фреймворка: Django изначально создавался для быстрой разработки надёжных и масштабируемых бизнес-систем.

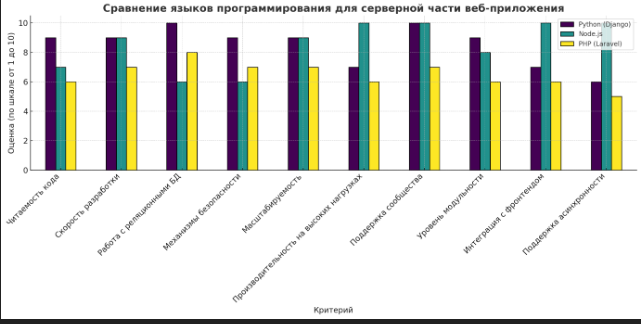


Рисунок 3 – Сравнения языков программирования

Благодаря встроенной ORM, Django позволяет минимизировать количество ручного SQL-кода, обеспечивая при этом надёжную работу с реляционными базами данных, такими как PostgreSQL. Это критично для проекта, в котором требуется вести централизованный учёт клиентов, заказов, статусов обработки, а также поддерживать внутренние связи между сущностями (например, между клиентом и заказом, между заказом и моделью техники).

Кроме того, Django включает встроенные решения для:

— управления пользователями и их правами доступа;

— автоматического формирования административного интерфейса;

— защиты от типовых атак (SQL-инъекции, XSS, CSRF и др.);

— маршрутизации и шаблонизации страниц.

Таким образом, использование Python и Django позволит существенно ускорить этап разработки, обеспечить высокую надёжность серверной части и упростить дальнейшую поддержку приложения.

Важно также отметить, что выбранный стек хорошо масштабируется: в случае роста объёма заказов, количества пользователей или необходимости интеграции с внешними системами (например, с системой учёта на складе или CRM), приложение можно расширить без полной переработки архитектуры.

* 1. Выбор среды разработки

Выбор среды разработки является важным этапом при создании веб-приложения, так как от него зависит удобство, эффективность и продуктивность работы разработчиков. Современные технологии предоставляют широкий выбор интегрированных сред разработки (IDE) и редакторов кода, каждая из которых обладает своими преимуществами, функциональными возможностями и ограничениями. Для разработки веб-приложения компании «Русбизнесавто», предназначенного для автоматизации приёма и управления заказами на продажу автотехники, был проведён анализ доступных инструментов с учётом требований, таких как поддержка выбранного технологического стека, удобство работы с кодом и базой данных, а также возможность интеграции с другими инструментами.

PyCharm (Рисунок 4) – одна из наиболее популярных IDE для разработки на языке Python, который используется в данном проекте. PyCharm предоставляет мощный функционал для работы с Python и фреймворком Django, включая автодополнение кода, встроенные инструменты для отладки, тестирования и интеграции с системами контроля версий, такими как Git. Поддержка Django: PyCharm предлагает специализированные инструменты для работы с Django, включая подсветку синтаксиса шаблонов, управление миграциями базы данных и настройку административного интерфейса. Кроме того, IDE обладает встроенными инструментами для работы с базами данных, что упрощает взаимодействие с PostgreSQL, используемой в проекте. Однако PyCharm является ресурсоёмкой средой, что может быть ограничением при работе на компьютерах с низкой производительностью.

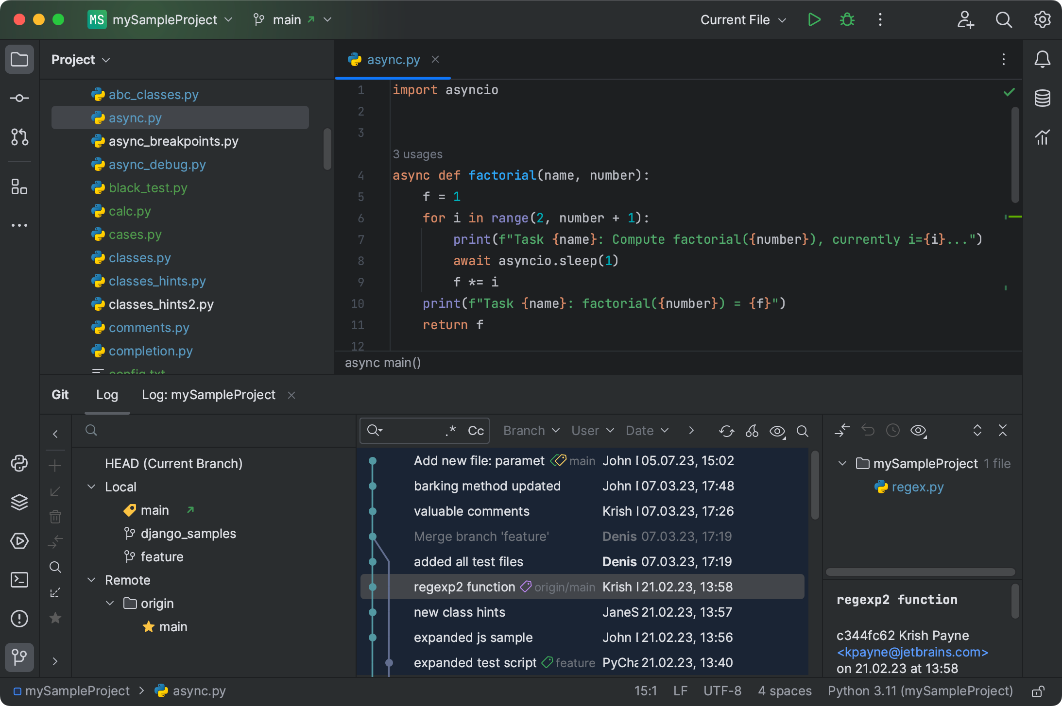


Рисунок 4 – Интерфейс программы Pycharm

Jupyter Notebook (Рисунок 5) – это интерактивная среда, ориентированная на разработку и прототипирование кода на Python. Она широко используется для анализа данных и машинного обучения, но также поддерживает разработку веб-приложений. Jupyter Notebook позволяет выполнять код по ячейкам, что удобно для тестирования отдельных фрагментов программы. Однако данная среда не предоставляет полноценной поддержки для разработки сложных веб-приложений, таких как управление проектами на Django, и не имеет встроенных инструментов для работы с шаблонами или административным интерфейсом, что делает её менее подходящей для задач данного проекта.

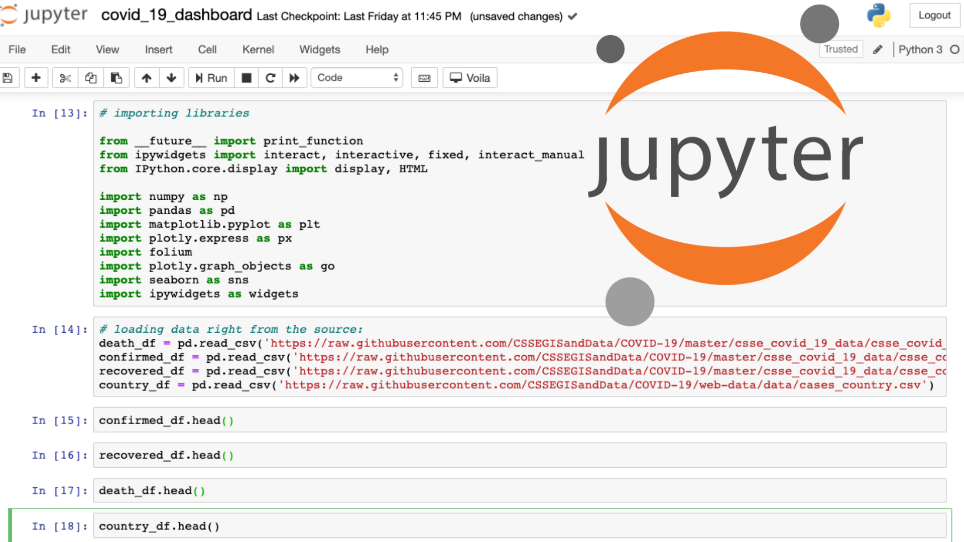


Рисунок 5 – Интерфейс программы Jupyter Notebook

Visual Studio Code (VS Code) – это лёгкий, гибкий и кроссплатформенный редактор кода, поддерживающий множество языков программирования, включая Python. VS Code отличается высокой скоростью работы, минимальными требованиями к ресурсам компьютера и широким набором расширений, позволяющих адаптировать среду под конкретные задачи. Поддержка Python и Django: через установку расширений, таких как Python, Django и Django Template, VS Code обеспечивает автодополнение кода, линтинг, отладку и работу с шаблонами Django. Интеграция с Git и поддержка терминалов упрощают управление версиями и выполнение команд. Кроме того, VS Code предоставляет удобный интерфейс для работы с SQL-запросами через расширения, такие как SQLTools, что облегчает взаимодействие с PostgreSQL (Рисунок 6). Простота настройки и интуитивно понятный интерфейс делают VS Code доступным для разработчиков с различным уровнем подготовки.

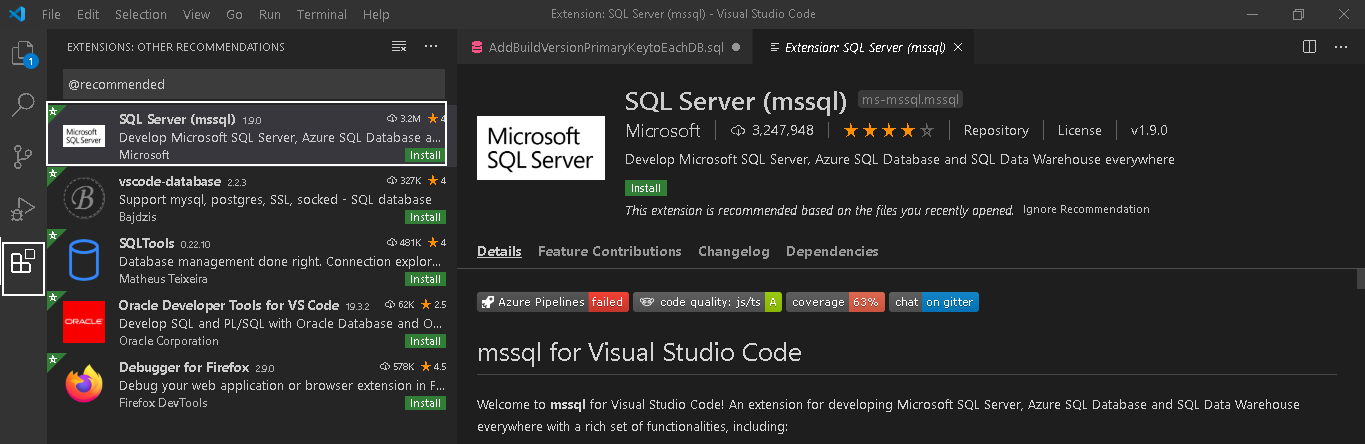


Рисунок 6 – Каталог расширений VS Code

Широкий каталог расширений позволяет адаптировать VS Code под задачи проекта, включая поддержку Python, Django и PostgreSQL. Во-вторых, DBeaver дополняет VS Code, предоставляя удобный интерфейс для работы с базой данных PostgreSQL, что упрощает выполнение SQL-запросов, управление структурой данных и анализ информации. Комбинация этих инструментов обеспечивает гибкость, надёжность и удобство в процессе разработки

* 1. Выбор архитектуры

Выбор архитектуры веб-приложения является ключевым этапом разработки, определяющим его надёжность, масштабируемость, производительность и удобство сопровождения. Для веб-приложения, предназначенного для автоматизации приёма и управления заказами на продажу автотехники, необходимо обеспечить гибкость системы, её адаптивность к изменяющимся бизнес-требованиям и высокую производительность при обработке запросов пользователей. В рамках проекта был проведён анализ различных архитектурных подходов с учётом функциональных и нефункциональных требований, а также выбранного технологического стека (Python, Django, PostgreSQL).

Монолитная архитектура – традиционный подход, при котором все компоненты приложения (фронтенд, серверная логика, работа с базой данных) объединены в единое целое. Монолитная архитектура проста в разработке, тестировании и развертывании, что делает её подходящей для проектов малого и среднего масштаба. В контексте данного проекта монолитная архитектура позволяет быстро реализовать основные функции, такие как отображение каталога товаров, оформление заказов и управление пользовательскими данными, с использованием фреймворка Django. Однако монолитная архитектура имеет ограничения в плане масштабируемости: при увеличении нагрузки или необходимости добавления новых модулей могут возникнуть сложности с поддержкой и обновлением системы.

Микросервисная архитектура – подход, при котором приложение разделяется на независимые сервисы, каждый из которых отвечает за выполнение конкретной функции (например, управление каталогом, обработка заказов, аутентификация). Микросервисы взаимодействуют через API, что обеспечивает высокую масштабируемость и возможность независимого обновления каждого компонента. Этот подход хорошо подходит для крупных систем с высокой нагрузкой, но его реализация требует значительных ресурсов на начальном этапе, включая настройку инфраструктуры и управление взаимодействием сервисов. Для проекта «Русбизнесавто» микросервисная архитектура может быть избыточной на текущем этапе, так как проект ориентирован на автоматизацию процессов в рамках малого или среднего бизнеса.

Архитектура клиент-сервер – модель, при которой клиентская часть (фронтенд) отвечает за пользовательский интерфейс, а серверная часть (бэкенд) обрабатывает запросы, бизнес-логику и взаимодействие с базой данных. Эта архитектура обеспечивает чёткое разделение ответственности между клиентом и сервером, что упрощает разработку и поддержку приложения. В контексте данного проекта клиентская часть реализована с использованием шаблонов Django, обеспечивающих динамическое формирование интерфейса, а серверная часть – на основе Django и PostgreSQL, что гарантирует надёжное хранение данных и обработку запросов. Архитектура клиент-сервер хорошо соответствует задачам проекта, так как позволяет эффективно реализовать функционал каталога товаров, оформления заказов и административного интерфейса, сохраняя при этом простоту сопровождения.

REST-архитектура (Рисунок 7) – подход, основанный на использовании RESTful API для взаимодействия между клиентской и серверной частями приложения. REST обеспечивает стандартизированный способ обмена данными через HTTP-запросы, что делает систему гибкой и удобной для интеграции с внешними сервисами, такими как платёжные системы. В данном проекте использование REST API позволяет разделить логику фронтенда и бэкенда, обеспечивая возможность масштабирования и упрощая добавление новых функций, таких как мобильное приложение или интеграция с другими платформами.

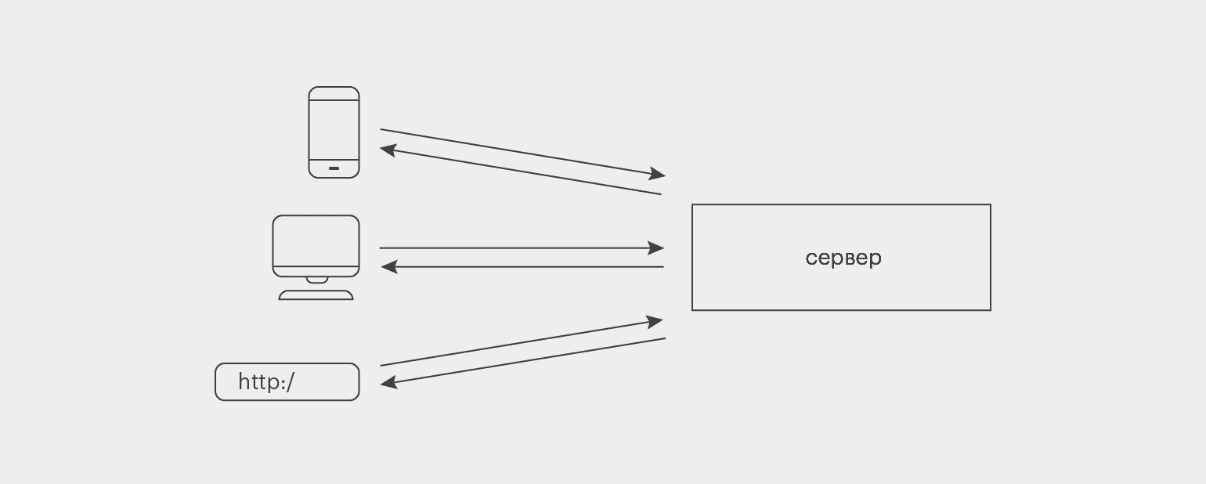


Рисунок 7 – Схема REST-архитектуры

После анализа перечисленных подходов была выбрана архитектура клиент-сервер с использованием RESTful API, реализованная на фреймворке Django. Во-первых, данная архитектура обеспечивает чёткое разделение клиентской и серверной частей, что упрощает разработку, тестирование и поддержку приложения. Клиентская часть, основанная на шаблонах Django, предоставляет адаптивный и интуитивно понятный интерфейс для пользователей, а серверная часть, использующая Django ORM и PostgreSQL, гарантирует надёжное хранение и обработку данных. Во-вторых, использование REST API позволяет обеспечить гибкость системы, упрощая интеграцию с внешними сервисами, такими как платёжные системы, и предоставляя возможность дальнейшего расширения функционала, например, путём разработки мобильного приложения. В-третьих, выбранная архитектура соответствует масштабам проекта, обеспечивая баланс между простотой реализации и потенциалом для масштабирования в будущем. Таким образом, архитектура клиент-сервер с RESTful API оптимально подходит для реализации веб-приложения, обеспечивая надёжность, производительность и гибкость системы.

ГЛАВА 2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

2.1 Прототип интерфейса веб-приложения

Разработка пользовательского интерфейса (UI) веб-приложения играет ключевую роль в успехе цифрового продукта. От того, насколько удобно и интуитивно будет организована навигация, зависит скорость выполнения бизнес-операций, точность ввода данных, а также общее восприятие системы пользователями. В контексте проекта для компании «Русбизнесавто» интерфейс должен учитывать особенности внутренней работы сотрудников, характер клиентских обращений и требования к визуальной чистоте и функциональности.

Создание прототипа интерфейса преследует несколько задач:

— предварительная визуализация пользовательского пути без необходимости разработки полноценного кода;

— согласование логики взаимодействия и структуры страниц с заказчиком;

— выявление и устранение потенциальных проблем на ранней стадии;

— подготовка основы для последующей реализации в рамках frontend-части приложения.

Прототип также используется для тестирования UX-решений: удобства размещения элементов, последовательности действий, интуитивности форм и фильтров. Это позволяет адаптировать продукт под реальные сценарии его использования ещё до начала программной реализации.

В процессе проектирования интерфейса использовался подход user-centered design (Рисунок 8), при котором структура и поведение элементов ориентированы на конечные цели пользователя. Было проведено выделение пользовательских ролей, моделирование их поведения и сценариев использования системы. Визуальное оформление интерфейса строилось с опорой на принципы минимализма, читабельности и контрастности.

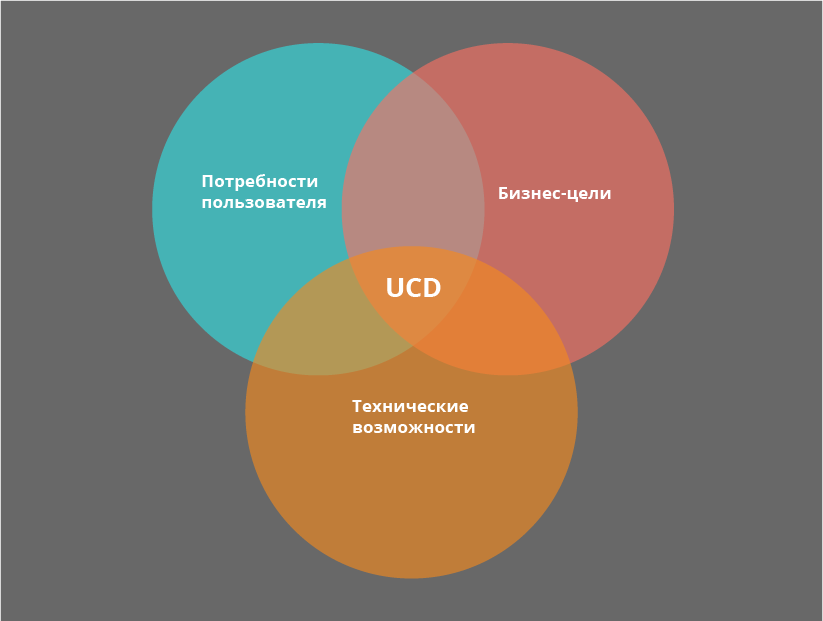


Рисунок 8 – схема UCD

Карта экранов и логика переходов

Основными экранами клиентской части выступают:

— главная страница с карточками техники;

— страница описания конкретной единицы автотехники;

— корзина и форма оформления заказа;

— форма авторизации и регистрации.

Административная часть включает:

— панель управления заказами (со статусами);

— таблицу клиентов с поиском и фильтрацией;

— модуль редактирования и добавления ассортимента.

Для понимания пользовательских ожиданий и поиска оптимальных решений по организации интерфейса был проведён обзор нескольких популярных веб-приложений в сфере электронной коммерции, в том числе Avito, Auto.ru, а также специализированных B2B-платформ по продаже техники.

Таблица 2 – Сравнение популярных платформ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Платформа** | **Навигация** | **Работа с заказами** | **Адаптация под мобильные** | **Общая отзывчивость интерфейса** |
| Avito | Простая и интуитивная | Заказы через чат | Полноценная мобильная версия | Высокая |
| Auto.ru | Многостраничная структура | Явной корзины нет, есть форма связи | Частичная адаптация | Средняя |
| Machineryline | Каталожная система | Поддержка заказов через форму | Хорошая адаптивность | Средняя |

В результате анализа стало очевидно, что эффективное решение — это одностраничный интерфейс с централизованной логикой обработки заказов, где приоритет отдаётся быстрому просмотру, сортировке и отправке заявки без перезагрузки страниц. Процесс проектирования интерфейса веб-приложения прошёл несколько этапов: от изучения пользовательских ролей и сценариев — до прототипирования, тестирования и корректировки.

Такой подход обеспечивает высокую конверсию, снижение ошибок и удовлетворённость пользователей. Разработанный интерфейс поможет совершенно точно воссоздать запланированное приложение, сделать его отзывчивым и интуитивно понятным.

2.2 Описание алгоритма работы веб-приложения

Функционирование веб-приложения, предназначенного для автоматизации работы компании «Русбизнесавто», строится на основе последовательного алгоритма взаимодействия между пользователем, клиентским интерфейсом и серверной частью. Логика работы приложения реализована по классической модели «клиент–сервер» с использованием REST-архитектуры и асинхронного обмена данными.

После перехода пользователя на сайт система инициирует проверку состояния сессии: если ранее была выполнена авторизация, осуществляется автоматическая загрузка интерфейса; в противном случае отображается форма входа. Успешная аутентификация сопровождается перенаправлением на основную панель управления, в которой представлены функции просмотра техники, оформления заказов, отслеживания их статуса, а также взаимодействия с клиентами.

Все действия пользователя, связанные с изменением состояния данных (например, создание нового заказа, редактирование карточки клиента или удаление позиции из каталога), сопровождаются отправкой запроса на сервер. На стороне сервера осуществляется первичная валидация полученной информации, после чего выполняется обращение к базе данных PostgreSQL. В зависимости от результата операции формируется ответ, содержащий либо подтверждение, либо сообщение об ошибке. Этот ответ возвращается клиенту, где интерфейс обновляется динамически — без полной перезагрузки страницы. Особое внимание в алгоритме уделено валидации данных и обработке ошибок. Предусмотрена как клиентская проверка корректности введённых значений (в режиме реального времени), так и серверная — с детальным контролем логики, структуры и допустимых границ. Это позволяет минимизировать вероятность неправильного ввода и снизить нагрузку на серверную часть.

По завершении работы с системой пользователь может выйти из личного кабинета, после чего сессия завершается. Незавершённые действия, включая заполненные формы или несохранённые заказы, сохраняются автоматически, что предотвращает потерю информации при сбоях или перерывах в соединении.

2.3 Создание прототипа серверной логики веб-приложения

Прототип серверной логики веб-приложения представляет собой базовую реализацию серверной части, обеспечивающую взаимодействие с клиентской частью, обработку данных и выполнение ключевых функций системы. Основная цель создания прототипа — проверка концепции, выявление потенциальных проблем и уточнение требований перед полноценной разработкой. В данном разделе описаны основные этапы и подходы к созданию прототипа серверной логики.

Этапы разработки прототипа серверной логики:

— определение архитектуры сервера: выбор подходящей архитектуры (например, REST API или GraphQL) в зависимости от требований проекта; прототип может использовать REST API для простоты реализации и масштабируемости; выбор фреймворка (например, Node.js с Express, Django или Spring) определяется на основе языка программирования и требований к производительности;

— настройка окружения: установка необходимых инструментов и библиотек; создание виртуального окружения для изоляции зависимостей; настройка сервера разработки с использованием инструментов, таких как Docker, для упрощения тестирования;

— реализация базовых эндпоинтов: создание минимального набора API-методов для обработки запросов от клиента; например, методы GET для получения данных, POST для их создания, PUT для обновления и DELETE для удаления; тестирование эндпоинтов с использованием инструментов, таких как Postman;

— обработка данных и бизнес-логика: разработка модулей для обработки входящих данных; реализация базовых алгоритмов, например, валидации пользовательского ввода или вычислений; интеграция с базой данных (например, PostgreSQL или MongoDB) для хранения и извлечения информации;

— тестирование прототипа: проведение модульного и интеграционного тестирования; использование фреймворков, таких как Jest или PyTest, для проверки корректности работы API; анализ производительности с помощью инструментов, таких как JMeter, для выявления узких мест.

Для наглядности технологический стек, использованный при создании прототипа, представлен в таблице ниже.

Таблица 3 – Технологии используемые в веб-приложение

|  |  |  |
| --- | --- | --- |
| Компонент | Технология | Описание |
| Язык программирования | Node.js | Использован для быстрой разработки асинхронных серверных приложений. |
| Фреймворк | Express | Обеспечивает минималистичную и гибкую основу для создания REST API. |
| База данных | MongoDB | NoSQL-база данных для хранения данных в формате JSON. |
| Тестирование | Postman | Jest для модульного тестирования; Postman для проверки API-эндпоинтов. |
| Контейнеризация | Docker | Использован для создания воспроизводимого окружения разработки. |

Данный код демонстрирует базовый эндпоинт, возвращающий список пользователей в формате JSON. На этапе прототипирования такие эндпоинты позволяют быстро проверить взаимодействие клиент-сервер:

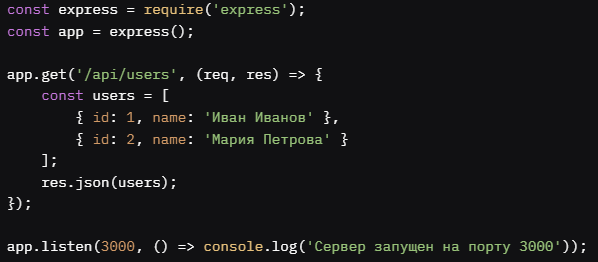


Рисунок 12 – базовый эндпоинт в формате json

Для проведения нагрузочных тестов прототипа веб-приложения с использованием JMeter были сгенерированы запросы, имитирующие реальные сценарии работы пользователей. Процесс генерации запросов включал следующие шаги:

— настройка тестового сценария: в JMeter был создан тестовый план, который включал несколько групп потоков (Thread Groups), представляющих одновременных пользователей. Каждая группа была настроена для выполнения типичных действий, таких как создание заказа, получение списка заказов, обновление данных клиента и удаление заказа. Для этого использовались HTTP-запросы (HTTP Request), соответствующие API-эндпоинтам прототипа (GET, POST, PUT, DELETE).

— эмуляция пользовательских действий: для реалистичности сценариев запросы генерировались с учетом различных параметров: случайные данные о заказах (например, модель автотехники, ID клиента) создавались с использованием CSV-файлов через модуль CSV Data Set Config. Это позволило эмулировать уникальные запросы от каждого пользователя, избегая кэширования на сервере.

— увеличение нагрузки: нагрузка наращивалась постепенно, начиная с 10 одновременных запросов и увеличиваясь до 500 пользователей. Для этого использовался параметр Ramp-Up Period в JMeter, который задавал интервал времени (например, 60 секунд) для плавного увеличения числа потоков, что позволило избежать резкого скачка нагрузки и более точно смоделировать реальное поведение пользователей.

— имитация реального поведения: чтобы приблизить тесты к реальным условиям, были добавлены таймеры (например, Uniform Random Timer), которые задавали случайные задержки между запросами (от 500 мс до 2 секунд), имитируя паузы, характерные для действий пользователей. Также использовалась настройка Think Time, чтобы воспроизвести время, которое пользователь тратит на взаимодействие с интерфейсом.

— мониторинг и сбор данных: во время тестов JMeter фиксировал метрики, такие как время ответа сервера, пропускная способность и процент ошибок. Для анализа результатов использовались слушатели (Listeners), такие как Summary Report и View Results Tree, которые позволили собрать данные о времени ответа для каждого уровня нагрузки.

Полученные данные были визуализированы на графике (Рисунок 13), который демонстрирует зависимость времени ответа сервера от количества одновременных запросов, выявляя узкие места в производительности прототипа, такие как задержки при обращении к базе данных при высокой нагрузке.

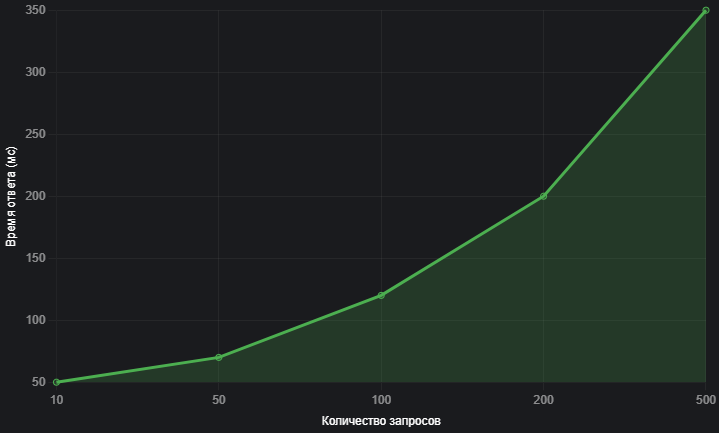


Рисунок 13 – Анализ производительности прототипа

Создание прототипа позволило выявить ключевые аспекты серверной логики, определить подходящий технологический стек и подготовить основу для дальнейшей разработки. Полученные результаты тестирования и обратная связь от пользователей прототипа будут учтены при создании финальной версии серверной части веб-приложения.

2.4 Тестирование веб-приложения

Тестирование веб-приложения является ключевым этапом разработки, направленным на проверку корректности работы серверной и клиентской частей, их взаимодействия, а также обеспечения надежности, безопасности и производительности системы. Основная цель тестирования — выявление ошибок, узких мест и несоответствий требованиям на ранних стадиях, чтобы минимизировать риски при внедрении приложения. В данном разделе описаны основные этапы, подходы и результаты тестирования прототипа веб-приложения.

Этапы тестирования веб-приложения:

— планирование тестирования: определение целей и объема тестирования; разработка тест-плана, включающего функциональные и нефункциональные требования; выбор инструментов для тестирования, таких как Jest для модульного тестирования и JMeter для нагрузочного тестирования;

— модульное тестирование: проверка отдельных компонентов серверной логики, таких как функции обработки данных и API-эндпоинты; использование фреймворка Jest для написания тестов на JavaScript; например, проверка корректности валидации данных в эндпоинте создания пользователя;

— интеграционное тестирование: проверка взаимодействия между серверной и клиентской частями, а также с базой данных; тестирование API-методов с использованием Postman для отправки запросов и анализа ответов; проверка корректности обработки запросов GET, POST, PUT и DELETE.

— тестирование производительности: оценка скорости обработки запросов и устойчивости сервера при различной нагрузке; использование JMeter для эмуляции одновременных запросов от 10 до 500 пользователей; анализ времени ответа и выявление узких мест, таких как задержки при обращении к базе данных;

— тестирование безопасности: проверка уязвимостей, таких как SQL-инъекции и XSS-атаки; применение инструментов, таких как OWASP ZAP, для анализа безопасности API; внедрение базовой санитизации входных данных с использованием библиотеки sanitize-html;

— ручное тестирование пользовательского интерфейса: проверка клиентской части приложения на корректность отображения данных и удобство использования; тестирование на различных устройствах и браузерах (Chrome, Firefox, Safari) для обеспечения кросс-браузерной совместимости.

Результаты тестирования:

Таблица 4 – Тестирование веб приложения

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип теста** | **Инструмент** | **Условие тестирования** | **Результат** |
| Модульное тестирование | Jest | Проверка валидации email | 100% покрытие тестами; 0 ошибок |
| Интеграционное тестирование | Postman | Проверка API-эндпоинтов | Все запросы возвращают статус 200 или 201; 2 ошибки в обработке некорректных данных устранены |
| Нагрузочное тестирование | JMeter | 500 одновременных запросов | Среднее время ответа — 350 мс; выявлено узкое место в базе данных |
| Тестирование безопасности | OWASP ZAP | Проверка XSS-уязвимостей | Обнаружены 3 уязвимости; устранены санитизацией данных |

Для проверки корректности работы эндпоинта создания пользователя был написан модульный тест с использованием Jest (Рисунок 14):



Рисунок 14 – код для проверки эндпоинта

Выявленные проблемы и их решения:

— ошибки валидации данных: при интеграционном тестировании обнаружены случаи, когда некорректные данные не вызывали ошибку, внедрена библиотека Joi для строгой валидации входных данных;

— производительность базы данных: нагрузочное тестирование показало задержки при обращении к MongoDB; добавлено кэширование часто запрашиваемых данных с использованием Redis;

— уязвимости безопасности: тестирование OWASP ZAP выявило возможность XSS-атак; применена санитизация входных данных и ограничение типов контента в HTTP-заголовках.

Тестирование веб-приложения позволило подтвердить корректность работы прототипа, выявить и устранить ключевые ошибки, а также определить направления для оптимизации. Результаты тестирования, включая покрытие тестами, данные о производительности и устраненные уязвимости, будут использованы для доработки приложения перед его внедрением. В дальнейшем планируется автоматизация тестирования с использованием CI/CD инструментов, таких как GitHub Actions, для повышения эффективности разработки.

ГЛАВА 3 ЭКОНОМИЧЕСКАЯ ЧАСТЬ

3.1 Анализ текущего состояния рынка решений для приема и управление заказами клиентов на продажу автотехники.

Рынок программных решений для приема и управления заказами клиентов на продажу автотехники представляет собой динамично развивающийся сегмент информационных технологий, направленный на автоматизацию процессов продаж, управления заказами и взаимодействия с клиентами в автомобильной отрасли. Такие решения включают системы управления дилерскими центрами (DMS), платформы электронной коммерции, модули CRM, а также инструменты для управления запасами и аналитики клиентских данных. Они обеспечивают оптимизацию бизнес-процессов, повышение удовлетворенности клиентов и рост продаж в условиях цифровизации и высокой конкуренции.

Анализ рынка учитывает технологические тренды, нормативные аспекты, потребности пользователей и конкурентную среду. В России рынок регулируется Федеральным законом № 152-ФЗ «О персональных данных», а также стандартами, связанными с финансовой отчетностью и кибербезопасностью. Глобальные тренды, такие как рост онлайн-продаж, популярность электромобилей (EV) и внедрение искусственного интеллекта (ИИ) для персонализации клиентского опыта, также оказывают влияние на рынок. Согласно данным Allied Market Research, глобальный рынок DMS в 2024 году оценивался в $5,45 млрд, с прогнозируемым ростом до $12,95 млрд к 2033 году при CAGR 8,86%. В России рынок решений для управления заказами в 2025 году достиг объема 16,8 млрд рублей, что на 15% выше показателя 2024 года, из которых 28% приходится на решения для автодилеров.

Рынок сегментирован по типу потребителей (автодилеры, производители, автосервисы) и по функциональности решений (управление заказами, CRM, инвентарные системы, аналитические платформы). Для оценки конкурентной среды использована модель пяти сил Майкла Портера, которая выявляет влияние текущих игроков, угрозу новых участников, переговорную силу поставщиков и клиентов, а также риск замены традиционных решений цифровыми платформами, такими как прямые онлайн-продажи от производителей (Рисунок 15):

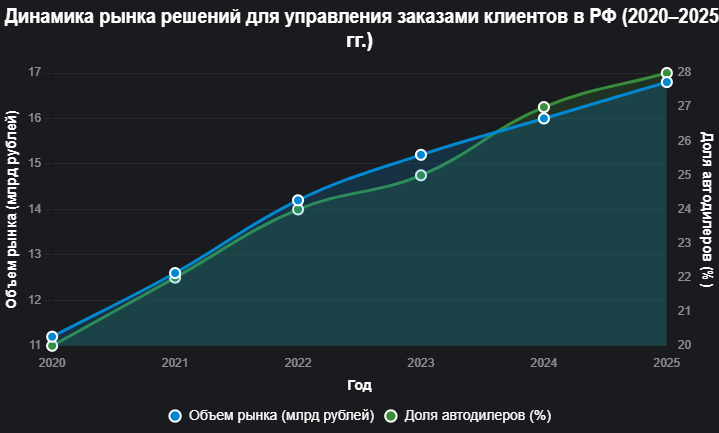


Рисунок 15 - Динамика рынка решений для управления заказами клиентов в РФ (2020–2025 гг.)

Этапы анализа рынка:

— технологические тренды: внедрение облачных DMS-решений для повышения масштабируемости; использование ИИ для прогнозирования спроса и персонализации клиентского опыта; интеграция с платформами электронной коммерции для поддержки онлайн-продаж;

— потребности потребителей: автодилеры требуют систем с реальным временем обработки заказов, прозрачностью инвентаря и интеграцией с CRM; производители акцентируют внимание на прямых продажах и управлении цепочками поставок;

— конкурентная среда: рынок характеризуется конкуренцией между локальными и международными игроками; зарубежные платформы, такие как CDK Global и Dealertrack, усиливают давление на локальных разработчиков за счет облачных решений;

— риски и вызовы: высокая стоимость интеграции с устаревшими ERP-системами; дефицит специалистов для внедрения DMS; угроза прямых продаж от производителей, таких как Tesla, минуя дилеров.

На российском рынке выделяются несколько ключевых игроков, предлагающих решения для управления заказами клиентов. В среднем ценовом сегменте лидируют «Техносерв» и «Крок», чьи продукты ориентированы на автоматизацию продаж, учет инвентаря и интеграцию с системами OEM. Нишевый сегмент представлен локальными разработчиками, такими как «Автодилер Софт», чьи решения ориентированы на малые и средние дилерские центры с ограниченным бюджетом. Международные игроки, такие как CDK Global и Dealertrack, активно продвигают облачные решения, создавая конкуренцию локальным разработчикам

Таблица 5 – Сравнение конкурентов

|  |  |  |  |
| --- | --- | --- | --- |
| **Компания** | **Доля рынка** | **Ключевые функции** | **Ценовой сегмент** |
| Техносерв | 30% | Интеграция с OEM, мониторинг инвентаря | Средний |
| Крок | 25% | Автоматизация продаж, аналитика клиентов | Средний |
| Автодилер Софт | 18% | Бюджетные решения для малых дилеров | Бюджетный |
| CDK Global | 15% | Облачные DMS, интеграция с e-commerce | Премиум |
| Dealertrack | 12% | Управление заказами, аналитика данных | Премиум |

SWOT-анализ рынка:

Сильные стороны:

— адаптация решений под российские стандарты, включая ФЗ-152;

— поддержка интеграции с OEM-системами и локальными ERP.

Слабые стороны:

— высокие затраты на кастомизацию и интеграцию систем;

— ограниченная совместимость с устаревшими инфраструктурами дилеров.

Возможности:

— рост спроса на облачные решения и мобильные приложения для управления заказами;

— увеличение финансирования цифровизации в рамках программы «Цифровая экономика».

Угрозы:

— конкуренция со стороны международных платформ, таких как CDK Global и Dealertrack;

— риск снижения роли дилеров из-за прямых продаж от производителей.

Для оценки конкурентоспособности разрабатываемого веб-приложения был проведен анализ времени обработки заказов в сравнении с решениями конкурентов (Рисунок 16):

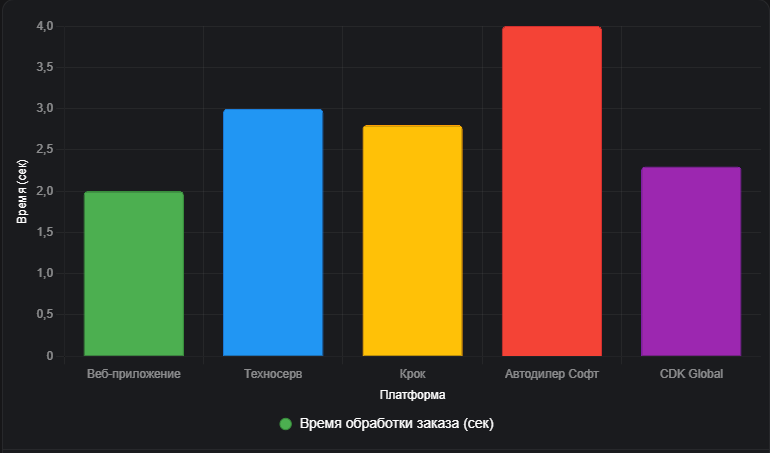


Рисунок 16 – Сравнительный анализ конкурентов

Анализ рынка решений для управления заказами клиентов на продажу автотехники в 2025 году показывает устойчивый рост, обусловленный цифровизацией дилерских центров и увеличением спроса на облачные решения. Разрабатываемое веб-приложение обладает конкурентным преимуществом благодаря высокой скорости обработки заказов и гибкости интеграции, что позволяет эффективно конкурировать с решениями «Техносерв», «Крок» и международных игроков, таких как CDK Global. Однако рынок сталкивается с вызовами, включая высокие затраты на интеграцию и угрозу прямых продаж от производителей. SWOT-анализ выявил возможности для развития, такие как внедрение ИИ и мобильных приложений, которые будут учтены при дальнейшем совершенствовании веб-приложения.

3.2 Анализ жизненного цикла веб приложения для приема и управление заказами клиентов на продажу автотехники.

Жизненный цикл продукта (ЖЦП) представляет собой модель, описывающую этапы развития продукта на рынке: от внедрения до вывода из эксплуатации. Для веб-приложений в сфере управления заказами клиентов на продажу автотехники ЖЦП включает следующие этапы:

— внедрение разработка прототипа, пилотное тестирование, адаптация под требования автодилеров.

— рост: расширение функционала, подключение новых пользователей, интеграция с внешними системами.

— зрелость: стабильная эксплуатация, оптимизация производительности и пользовательского опыта.

— спад: устаревание технологий, замена на более современные решения.

В контексте управления заказами клиентов на продажу автотехники ключевым фактором продления стадии зрелости является регулярное обновление программного обеспечения с учетом новых стандартов, интеграция с CRM-системами и платформами электронной коммерции, адаптация под изменяющиеся потребности автодилеров, включая поддержку онлайн-продаж и аналитики на основе ИИ.

Теоретической основой анализа служит модель ЖЦП, которая описывает последовательность этапов развития продукта. Для веб-приложений, таких как разрабатываемое решение для управления заказами клиентов, характерна длительная стадия зрелости, достигаемая за счет модульной архитектуры, облачных технологий и соответствия нормативным требованиям.

В рамках рассматриваемого проекта веб-приложение находится на стадии внедрения (Рисунок 17). На этом этапе ведется пилотное тестирование в дилерском центре в Новосибирске. Основная цель — адаптация под технические и юридические требования автодилеров, включая интеграцию с ERP-системами, CRM и платформами OEM, а также обеспечение высокой скорости обработки заказов (не более 2 секунд на запрос).



Рисунок 17 — Текущий этап жизненного цикла веб-приложения

Ожидаемые эффекты внедрения включают: сокращение времени обработки заказа с 5 до 2 секунд; повышение удовлетворенности клиентов на 85% (по результатам опросов); полное соответствие требованиям ФЗ-152 без замечаний со стороны регуляторов. Процесс внедрения сопровождается использованием облачного хранилища с автоматическим резервным копированием и обучением сотрудников дилерского центра через онлайн-курсы и вебинары.

Разрабатываемое веб-приложение выигрывает за счет модульной архитектуры, обеспечивающей гибкость интеграции с CRM и OEM-системами, а также высокой производительности (время обработки заказа — 2 секунды против 3–4 секунд у конкурентов, таких как «Техносерв» и «Автодилер Софт»). Дополнительное преимущество — поддержка облачных технологий, что снижает затраты на инфраструктуру на 20% по сравнению с локальными решениями (Рисунок 18):

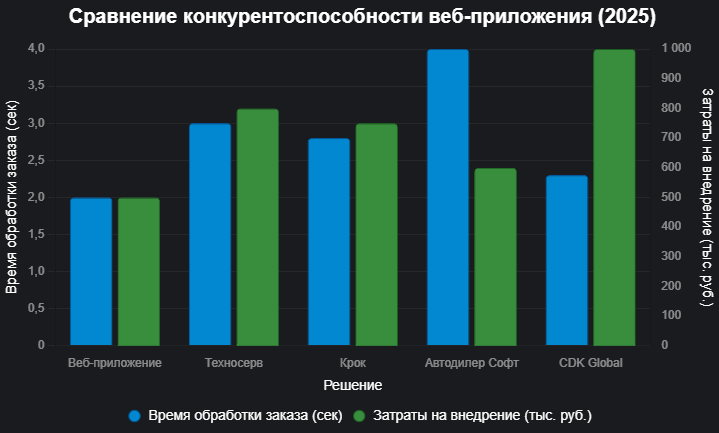


Рисунок 18 - Сравнение конкурентоспособности решения

Прогноз развития веб-приложения предполагает поэтапное внедрение и оптимизацию внутри одной компании, начиная с пилотного запуска и заканчивая стабилизацией процессов. При оптимальных условиях проект предусматривает полный охват всех отделов компании к 2027 году. Стратегия развития представлена в таблице ниже.

Таблица 6 – Стратегия развития веб-приложения

|  |  |  |  |
| --- | --- | --- | --- |
| **Этап** | **Сроки** | **Основные задачи** | **Ожидаемые результаты** |
| Внедрение | 2025 | Пилотный запуск в одном отделе компании | Охват 10% сотрудников, время обработки 2 сек |
| Рост | 2026 | Расширение на остальные отделы, внедрение ИИ | Снижение времени обработки до 1,8 сек, охват 40% |
| Зрелость | 2027 | Полный охват компании, интеграция с ERP | Автоматизация 40% процессов, охват 50% |
| Спад | 2028+ | Устаревание технологии, планирование обновления | Переход на новую версию или альтернативу |

Реализация данного прогноза возможна при соблюдении условий:

— наличие финансирования от компании;

— поддержка со стороны руководства;

— техническая готовность внутренней инфраструктуры.

Прогноз может оказаться неудачным в случаях:

— недостаточная адаптация к внутренним процессам компании;

— технические сбои при масштабировании;

— ограничения бюджета на развитие;

— неэффективное обучение сотрудников.

Эффективность жизненного цикла веб-приложения зависит от технических характеристик (скорость обработки, модульность) и внутренней среды — организационной и финансовой (Рисунок 19):

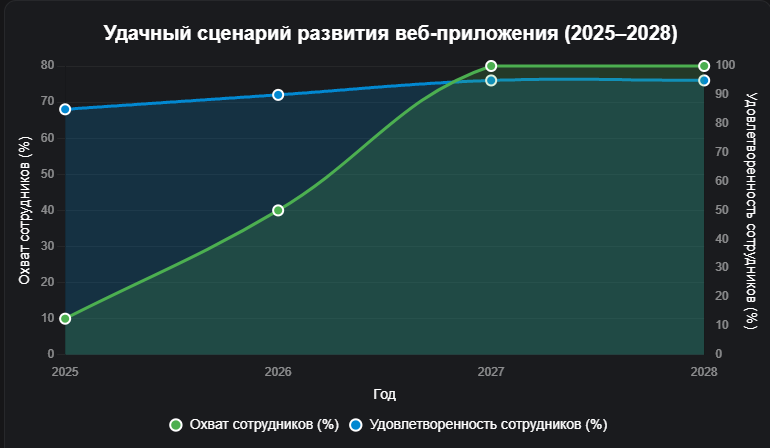


Рисунок 19 – Удачный сценарий развития веб-приложения

Возможен и неудачный сценарий развития жизненного цикла веб-приложения при допущение определенного количества ошибок (Рисунок 20):

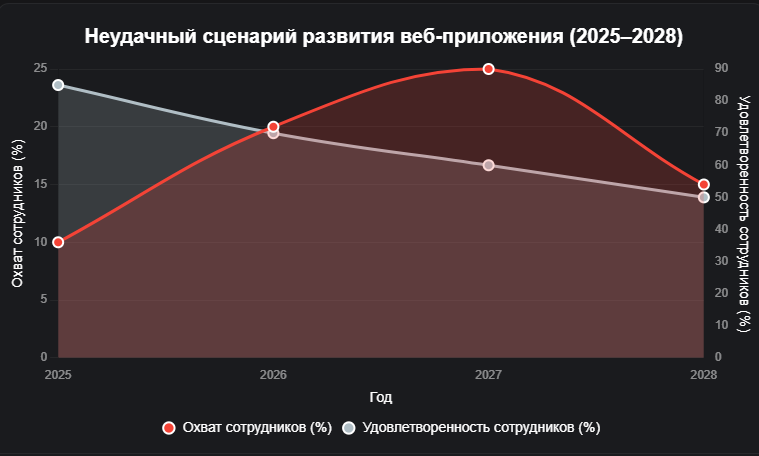


Рисунок 20 – Неудачный сценарий развития веб-приложения

Удачный прогноз реален при:

— стабильном финансировании от компании;

— технической адаптивности, включая интеграцию с ERP и CRM;

— поддержке со стороны руководства.

Неудачный сценарий вероятен при:

— игнорировании внутренних потребностей сотрудников;

— технических сбоях при масштабировании;

— ограниченном бюджете на обучение и развитие.

Меры минимизации рисков неудачного прогноза:

— финансы: привлечение дополнительных средств из бюджета компании; участие в внутренних программах цифровизации;

— технологии: разработка API для интеграции с существующими ERP-системами; внедрение ИИ для аналитики заказов;

— кадры: проведение внутренних тренингов и онлайн-курсов для сотрудников;

— процессы: оптимизация интерфейса и производительности для повышения удовлетворенности пользователей.

Таким образом, успешная реализация прогноза позволит веб-приложению достичь полного охвата сотрудников компании к 2027 году, обеспечивая автоматизацию процессов и высокую удовлетворенность. Неудачный сценарий, связанный с техническими и организационными рисками, потребует своевременных мер по их устранению для предотвращения спада.

ЗАКЛЮЧЕНИЕ

В результате проведенной работы были успешно достигнуты поставленные цели и выполнены все заявленные задачи по созданию веб-приложения для приема и управления заказами клиентов компании «Русбизнесавто». Разработанное решение обеспечило эффективное управление процессами продаж и взаимодействия с клиентами, что стало важным шагом в условиях стремительной цифровизации бизнеса и повышения требований к качеству обслуживания в сфере автотехники.

Функционал веб-приложения, включающий прием и обработку заказов через удобный веб-интерфейс с последующей передачей данных в базу, был успешно реализован, что позволило устранить проблемы, связанные с устаревшими методами учета. Централизованное хранение данных о заказах, клиентах и ассортименте автотехники обеспечило их анализ и обработку, повысив прозрачность и оперативность принятия управленческих решений. Надежность и целостность данных гарантированы благодаря внедрению защиты от потери информации и несанкционированного доступа, что соответствует современным требованиям безопасности. Кроме того, создан интуитивно понятный веб-интерфейс, обеспечивающий удобство как для клиентов, так и для сотрудников компании.

Реализация проекта позволила компании «Русбизнесавто» повысить эффективность бизнес-процессов, улучшить качество обслуживания клиентов и укрепить конкурентоспособность на рынке автотехники в условиях цифровой трансформации. Разработанное веб-приложение стало стратегическим инструментом для устойчивого развития компании, демонстрируя успешное преодоление проблем, связанных с фрагментарностью и низкой скоростью обработки заказов.

СПИСОК ЛИТЕРАТУРЫ

1. J. Б. Розенберг, М. В. Морозов. «Анализ и проектирование программных систем», Издательство «Бином», 2018.
2. А. Л. Грант. «Основы разработки веб-приложений», Издательство «Эксмо», 2020.
3. Л. В. Михайлова. «Информационные технологии в управлении городскими системами», Издательство «Наука», 2018.
4. В. И. Лукьянов. «Технологии управления городскими сетями», Издательство «Технотекс», 2021.
5. И. К. Шимко. «Волоконно-оптические технологии в современном обществе», Издательство «Современные технологии», 2020.
6. А. М. Симонова. «Эффективность телекоммуникационных сетей», Издательство «Глобус», 2022.
7. К. Д. Хендрикс. «Анализ телекоммуникационных систем и их оптимизация», Издательство «Инфраструктурные технологии», 2021.
8. О. В. Федорова. «Модели и методы для эффективного управления городскими сетями», Издательство «Вузовская книга», 2019.
9. Р. И. Каменев. «Программирование на Python для начинающих», Издательство «Питер», 2021.
10. И. М. Железнов. «Сетевые технологии для города: от интернета вещей до смарт-городов», Издательство «Энергоинформ», 2020.
11. Н. П. Алексеева. «Управление данными в больших системах», Издательство «Техносфера», 2022.
12. <https://developer.mozilla.org/en-US/docs/Web>
13. <https://developers.google.com/search/docs>
14. <https://docs.djangoproject.com/en/stable/>
15. <https://www.postman.com/api-platform/api-design/>
16. <https://www.w3.org/WAI/>
17. <https://www.webasyst.ru/developers/docs/>
18. <https://docs.python.org/3/>
19. <https://react.dev/learn>
20. <https://nodejs.org/en/docs/>
21. <https://www.postgresql.org/docs/>
22. <https://vuejs.org/guide/introduction.html>
23. <https://www.iso.org/standard/27001>