МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки)

Разработка программного обеспечения
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка веб-приложения для автоматизации процесса обработки заявок от клиентов в сервисном центре»

Обучающийся Н.Ю. Авдонин _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.т.н., доцент, О.В. Аникина
ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Данная выпускная квалификационная работа посвящена разработке веб-приложения для автосервиса с целью оптимизации процессов обслуживания клиентов.

Целью данной работы является разработка веб-приложения для автоматизации процесса обработки заявок от клиентов в сервисных центрах, на примере приложения для ремонта машин. Это приложение предполагается как инструмент, который позволит оптимизировать процесс приема, регистрации и отслеживания заявок от клиентов, а также обеспечит удобство взаимодействия как для клиентов, так и для сотрудников сервисного центра.

Предметом работы является разработка веб-приложения для автоматизации процесса обработки заявок от клиентов в сервисных центрах, с упором на автосервисы.

Объектом работы являются процессы приема, регистрации и отслеживания заявок от клиентов в сервисных центрах, а также взаимодействие клиентов и сотрудников автосервисов.

В выпускной квалификационной работе рассмотрены основные принципы веб-разработки, проведен анализ предметной области и бизнес-процессов автосервиса. Особое внимание было уделено выбранным технологиям, таким как JSX, React и Node.js, и их применению при создании веб-приложений. В результате работы разработана структура базы данных и создан пользовательский интерфейс.

Практическая часть работы включает в себя тестирование функционала приложения, что позволяет подтвердить его надежность и качество. Работа завершается выводами о проделанной работе и обсуждением перспектив развития проекта.

Оглавление

Введение

В настоящее время сервисные центры, сталкиваются с рядом вызовов в обработке заявок от клиентов. Большой объем заявок, необходимость оперативного реагирования на них, а также требования к качеству обслуживания создают необходимость в эффективных инструментах управления процессом обработки заявок. В связи с этим, разработка веб-приложения для автоматизации данного процесса становится актуальной задачей, которая способствует повышению эффективности работы сервисных центров.

Актуальность работы обусловлена необходимостью повышения эффективности обслуживания клиентов в сервисных центрах, за счет разработки веб-приложения для автоматизации процесса обработки заявок, что позволит оптимизировать рабочие процессы и повысить уровень удовлетворенности клиентов. Растущая конкуренция на рынке сервисных услуг и растущие ожидания клиентов в области оперативности и качества обслуживания подчеркивают важность внедрения современных технологических решений для оптимизации процессов работы сервисных центров и улучшения их конкурентоспособности.

Целью данной работы является разработка веб-приложения для автоматизации процесса обработки заявок от клиентов в сервисных центрах, на примере приложения для ремонта машин. Это приложение предполагается как инструмент, который позволит оптимизировать процесс приема, регистрации и отслеживания заявок от клиентов, а также обеспечит удобство взаимодействия как для клиентов, так и для сотрудников сервисного центра.

Предметом работы является разработка веб-приложения для автоматизации процесса обработки заявок от клиентов в сервисных центрах, с упором на автосервисы.

Объектом работы являются процессы приема, регистрации и отслеживания заявок от клиентов в сервисных центрах, а также взаимодействие клиентов и сотрудников автосервисов.

Задачи, которые необходимо решить для достижения поставленной цели исследования:

- анализ существующих проблем и недостатков в процессе обработки заявок в сервисных центрах, особенно в автосервисах;
- изучение современных технологий веб-разработки и выбор наиболее подходящих для реализации веб-приложения;
- проектирование структуры и функционала веб-приложения с учетом потребностей сервисных центров;
- реализация веб-приложения, включая создание серверной и клиентской частей, а также его интеграцию с базой данных и другими системами;
- тестирование и отладка разработанного веб-приложения;
- внедрение и использование веб-приложения в работе выбранных автосервисов, а также оценка его эффективности и удовлетворенности его пользователями.

Данная работа представляет собой важный шаг в направлении совершенствования процессов обработки заявок в сервисных центрах, обеспечивая более высокий уровень обслуживания клиентов и повышая эффективность работы персонала.

Глава 1 Теоретические основы разработки веб-приложения

1.1 Изучение принципов и методов веб-разработки

Информация, доступная в Интернете, размещается на компьютерах, которые работают в качестве веб-серверов и используют специальное программное обеспечение. Большая часть этой информации организована в форме веб-сайтов, каждый из которых имеет уникальное имя, также известное как адрес.

Веб-сайт – это данные, представленные в определенном формате, которые хранятся на веб-сервере и имеют свой уникальный адрес. Для просмотра веб-сайтов на компьютере пользователя используются специальные программы, называемые браузерами. В настоящее время наиболее распространенными браузерами являются Google Chrome, Yandex, Microsoft Edge. При вводе адреса в строку «Адрес» браузер загружает соответствующую информацию в свое окно в зависимости от указанного имени (адреса) сайта. Веб-сайты могут содержать разнообразные типы информации, включая текстовые материалы, изображения, видео, аудио и другие мультимедийные элементы. Веб-сайты могут также предлагать пользователю интерактивные возможности, такие как формы обратной связи, онлайн-чаты, электронные платежные системы и другие сервисы. Браузеры обеспечивают пользователю доступ к этим ресурсам и позволяют удобно взаимодействовать с веб-сайтами. Важно отметить, что различные браузеры могут отображать веб-сайты по-разному, в зависимости от их совместимости с веб-стандартами и используемых технологий.

Веб-сайт представляет собой совокупность взаимосвязанных веб-страниц. Веб-страница, в свою очередь, является текстовым файлом с расширением *.htm, содержащим текстовую информацию и специальные HTML-коды, определяющие ее отображение в браузере. Графические, аудио- и видеофайлы, такие как *.gif, *.jpg (графика), *.mid, *.mp3 (звук), *.avi

(видео), не включаются непосредственно в веб-страницу, а представляются отдельными файлами. В HTML-коде страницы содержатся ссылки на такие файлы. Каждая веб-страница имеет свой уникальный интернет-адрес, состоящий из адреса сайта и имени файла, что делает веб-сайт индивидуальным информационным ресурсом, размещенным на веб-сервере. важно отметить, что веб-сайт может содержать различные типы контента, такие как статические и динамические элементы, интерактивные формы, скрипты и другие функциональные компоненты. Эти элементы обеспечивают разнообразие и функциональность веб-сайта, делая его более привлекательным и удобным для пользователей. Кроме того, различные веб-страницы на сайте могут иметь разные структуры и назначения, отображая различные разделы информации или предоставляя доступ к различным сервисам и функциям. Таким образом, веб-сайт является важным инструментом для представления информации, обмена данными и взаимодействия с пользователями в современном интернет-пространстве. Также стоит отметить, что веб-сайты могут быть созданы для самых разных целей: от представления информации о компании или продукте до образовательных ресурсов, развлекательных порталов и электронной коммерции. Важно, чтобы дизайн и функциональность веб-сайта соответствовали его целям и потребностям целевой аудитории, обеспечивая удобство использования и эффективность в достижении поставленных целей. Таким образом, веб-сайты играют ключевую роль в современной интернет-культуре и являются важным инструментом как для предприятий и организаций, так и для обычных пользователей интернета.

1.1.1 Описание предметной области

Предметная область. В современном мире автосервисы играют ключевую роль в обеспечении технической исправности и безопасности автомобилей [4]. Автосервисы предоставляют широкий спектр услуг, включая диагностику, техническое обслуживание, ремонт и модернизацию транспортных средств. С развитием технологий, автоматизация процессов в

автосервисах становится необходимостью для повышения эффективности работы, улучшения качества обслуживания клиентов и оптимизации ресурсов[5] [6].

Разработка веб-приложения для автоматизации процесса обработки заявок от клиентов в автосервисе направлена на упрощение взаимодействия между клиентами и сервисным центром [13]. Приложение позволяет клиентам оставлять свои контактные данные, описывать марку машины и характер проблемы. Это помогает автосервису быстрее и точнее обрабатывать заявки, распределять задачи между специалистами и обеспечивать высокий уровень обслуживания.

Типовая организация, для которой предназначена разработка веб-приложения, представляет собой автосервис среднего размера, предоставляющий комплекс услуг по обслуживанию и ремонту автомобилей. Организационная структура автосервиса включает несколько ключевых отделов и ролей, необходимых для эффективного функционирования.

Организационная структура автосервиса включает следующие основные элементы, представленные на рисунке 1.

Владелец автосервиса - отвечает за стратегическое управление и общее руководство деятельностью автосервиса;

Управляющий автосервисом - координирует работу всех отделов, следит за выполнением задач и качеством предоставляемых услуг.

Технический отдел - занимается диагностикой, ремонтом и техническим обслуживанием автомобилей. Включает:

- начальника технического отдела,
- мастеров по ремонту,
- механиков,
- электриков,
- специалистов по диагностике.

Отдел приема и оформления заказов - отвечает за обработку заявок от клиентов, запись на услуги и первичную консультацию:

– начальник отдела приема и оформления заказов;

– специалисты по приему заказов.

Отдел клиентского обслуживания - обеспечивает взаимодействие с клиентами, решает вопросы и обрабатывает отзывы [13]:

– начальник отдела клиентского обслуживания;

– менеджеры по работе с клиентами;

– операторы call-центра.

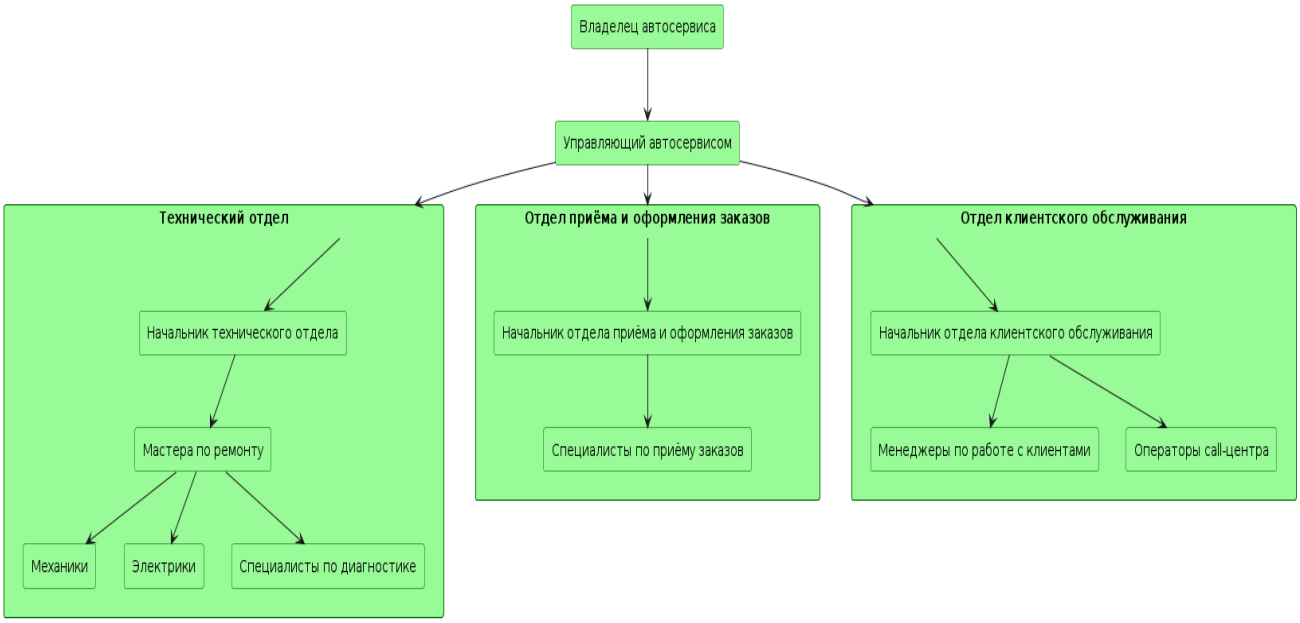

Рисунок 1 – Схема организационной структуры

Были рассмотрены основные понятия предметной области, организационная структура предприятия, подлежащего исследованию.

1.1.2 Описание основного бизнес-процесса автосервиса

Текущий процесс («Как есть»). Запись клиента на обслуживание:

– клиент звонит или приходит в автосервис;

– администратор записывает данные клиента и его автомобиля;

– назначается время для визита.

Прием автомобиля:

– клиент привозит автомобиль в автосервис;

– администратор или мастер осматривает автомобиль и записывает проблему;
– оформляется заказ-наряд.

Диагностика и ремонт:

– мастер проводит диагностику автомобиля;
– в случае необходимости, заказывается дополнительное оборудование или запчасти;
– проводится ремонт.

Контроль качества:

– Проверка выполненной работы;
– тестирование автомобиля.

Выдача автомобиля клиенту:

– клиенту сообщается о завершении ремонта;
– клиент приходит за автомобилем, оплачивает услуги и получает машину.

Недостатки и узкие места:

– время на запись клиента может быть длительным из-за очередей или занятости администраторов;
– человеческий фактор: возможны ошибки при записи данных клиента и автомобиля;
– неудобства для клиентов из-за необходимости личного посещения или телефонного звонка для записи;
– задержки в процессе диагностики и ремонта из-за отсутствия автоматизации в заказе запчастей и оборудования.

На рисунке 2 представлен бизнес-процесс «Как есть».

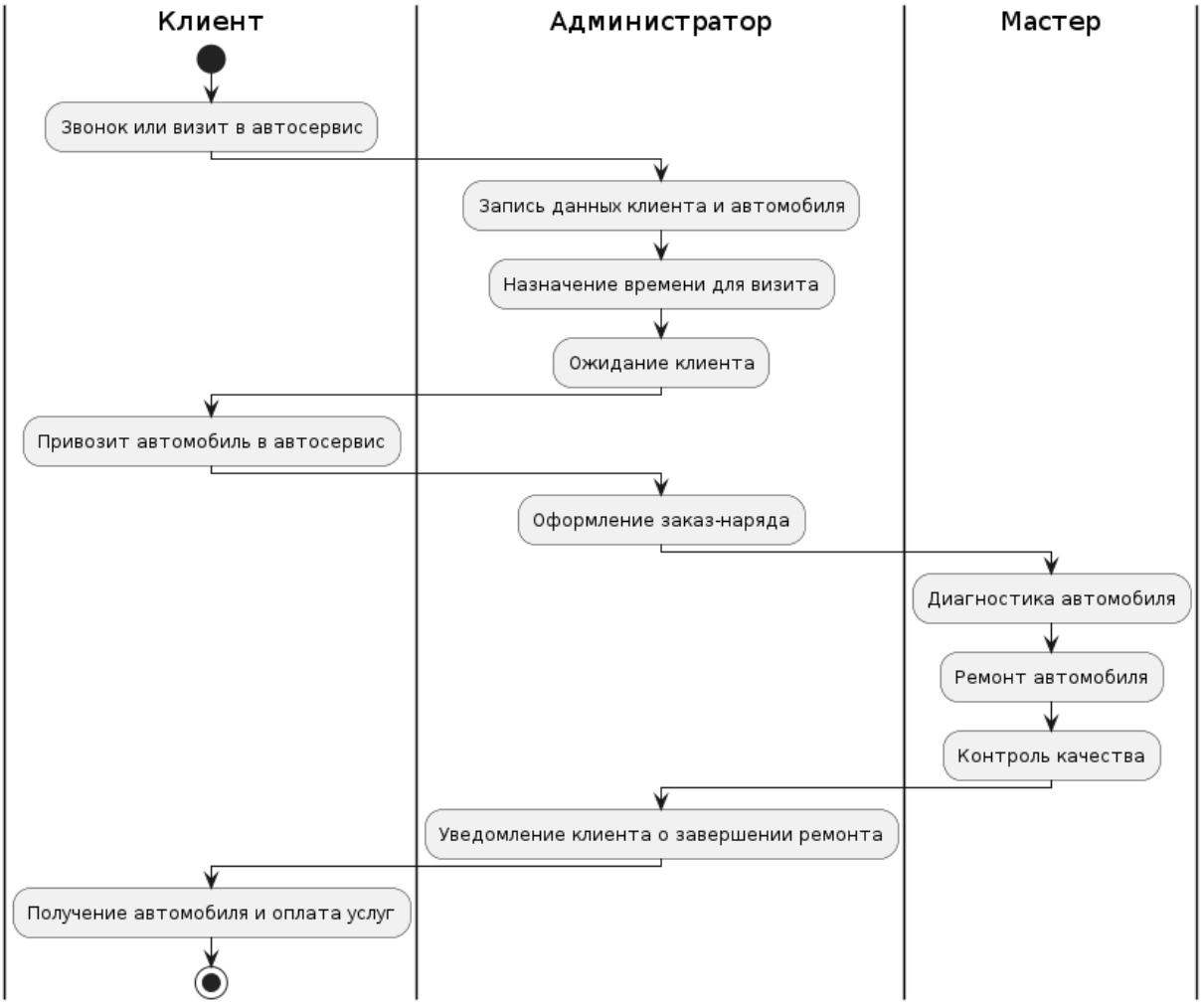

Рисунок 2 - Диаграмма «Как есть»

Оптимизированный процесс с веб-приложением («Как должно быть»). Запись клиента на обслуживание через веб-приложение:

- клиент заходит на сайт автосервиса и вводит свои данные, марку автомобиля и описание проблемы;
- клиент выбирает удобное время для визита из доступных слотов.

Прием автомобиля:

- клиент привозит автомобиль в автосервис в назначенное время;
- администратор проверяет данные в системе и оформляет заказ-наряд автоматически.

Диагностика и ремонт:

– мастер видит заказ в системе, проводит диагностику и отмечает необходимые запчасти;
– система автоматически заказывает необходимые запчасти;
– проводится ремонт.

Контроль качества:

– проверка выполненной работы;
– тестирование автомобиля.

Выдача автомобиля клиенту:

– клиенту приходит уведомление через веб-приложение о завершении ремонта;
– клиент приходит за автомобилем, оплачивает услуги через веб-приложение и получает машину.

Преимущества использования веб-приложения:

– удобство для клиентов: возможность записи на обслуживание онлайн;
– снижение нагрузки на администраторов и уменьшение вероятности ошибок при записи данных;
– автоматизация заказа запчастей, что сокращает время на диагностику и ремонт;
– улучшенное взаимодействие с клиентами через уведомления и онлайн-оплату.

На рисунке 3 представлен бизнес-процесс «Как должно быть».

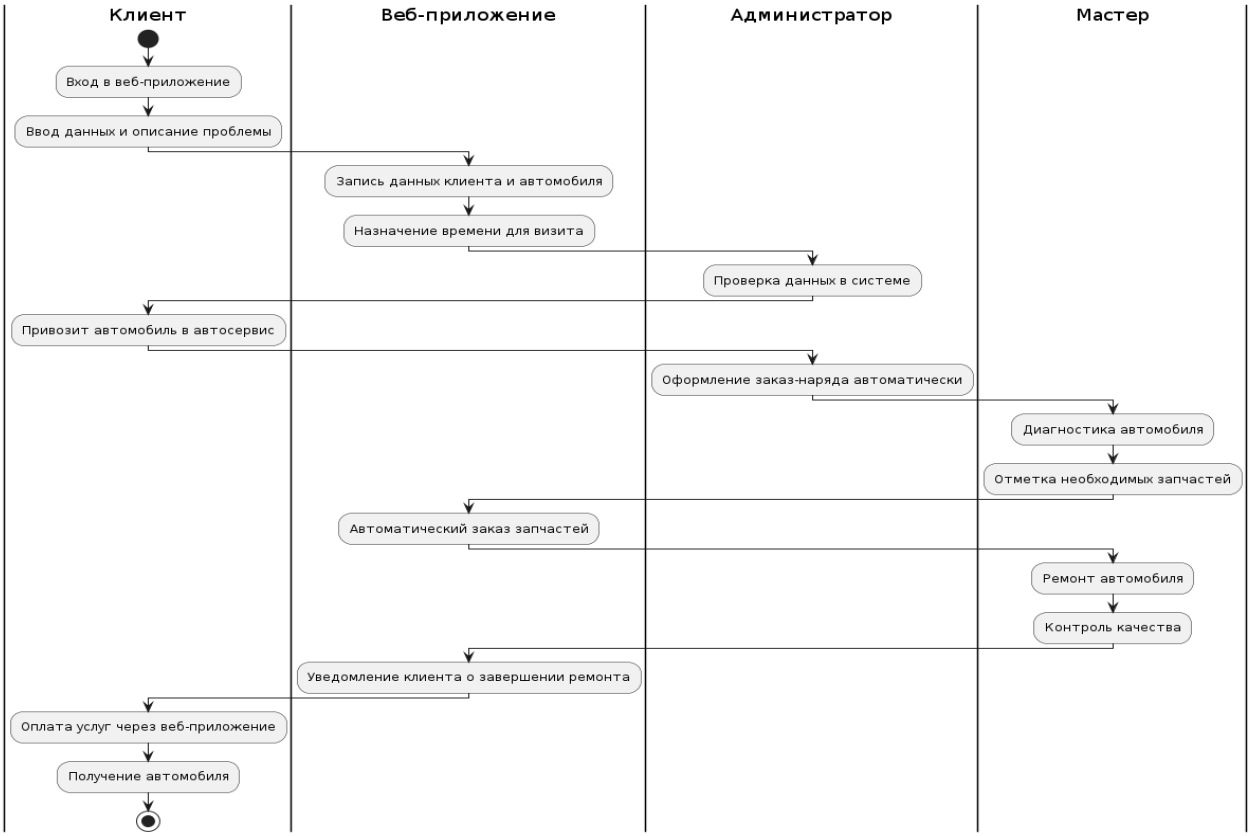

Рисунок 3 - Диаграмма «Как должно быть»

Изменения в процессе:

- клиенты могут записываться на обслуживание через веб-приложение, вводя свои данные и описание проблемы, что снижает нагрузку на администраторов;
- автоматическая проверка данных и оформление заказ-наряда, что минимизирует ошибки;
- система автоматически заказывает необходимые запчасти на основе диагностики, ускоряя процесс ремонта;
- клиенты получают уведомления о статусе ремонта и могут оплачивать услуги онлайн, что упрощает взаимодействие и повышает удовлетворенность.

Преимущества:

– удобство для клиентов: возможность записи на обслуживание и оплаты онлайн;
– снижение нагрузки на администраторов: уменьшение времени на обработку данных и запись клиентов;
– уменьшение ошибок: автоматическая проверка данных и оформление заказ-нарядов;
– скорость и эффективность: быстрое задание и выполнение заказа запчастей, что сокращает время на ремонт;
– улучшенное взаимодействие: автоматические уведомления для клиентов о статусе ремонта и оплате.

1.1.3 Web – программирование

Web-программирование, также известное как веб-разработка, представляет собой быстро развивающийся сегмент программирования, ориентированный на создание динамических интернет-приложений.

Языки веб-программирования подразделяются на две основные категории: клиентские и серверные. Клиентские языки обрабатываются на стороне пользователя, в основном в его браузере. Это означает, что обработка скриптов зависит от используемого браузера, и пользователь имеет возможность настроить свой браузер таким образом, чтобы скрипты игнорировались. Однако с современными браузерами таких проблем обычно не возникает. Кроме того, языки программирования редко подвергаются кардинальным изменениям, и их лучшие практики давно установлены. Код клиентских скриптов доступен каждому, кто выберет соответствующую опцию в меню своего браузера.

Преимущество клиентского языка заключается в том, что он позволяет выполнять обработку скриптов без отправки документа на сервер. Это позволяет, например, проверять правильность заполнения формы непосредственно перед ее отправкой и выводить сообщения об ошибках. Однако с помощью клиентского языка программирования нельзя записывать данные на сервер. Наиболее распространенным из клиентских языков

является JavaScript, разработанный совместно компаниями Netscape и Sun Microsystems. Еще одним популярным языком является VBScript. В последнее время набирают популярность технологии, такие как AJAX, Adobe Flash, Microsoft Silverlight и другие.

Серверные языки программирования открывают перед разработчиком большие возможности. Когда пользователь запрашивает определенную страницу, сервер сначала обрабатывает ее, выполняя все программы, связанные с ней, и только потом отправляет ее пользователю через сеть в виде файла. Этот файл может иметь различные расширения, такие как HTML, PHP, ASP, Perl, SSI, XML, DHTML, XHTML.

1.1.4 Сравнительная характеристика аналогичных разработок

В современном рынке программного обеспечения существует множество решений для автоматизации процессов в автосервисах. В данном разделе рассмотрим несколько популярных систем, их функциональные возможности, преимущества и недостатки (см. таблицу 1). Это поможет определить конкурентные преимущества разрабатываемого веб-приложения.

Таблица 1 - Сравнительная характеристика программного обеспечения

Функциональность / ПО	AutoFluent	Shop-Ware	Mitchell 1	ALLDATA	Разрабатываемое веб-приложение
Управление заказами	Да	Да	Да	Да	Да
CRM	Да	Да	Ограничено	Ограничено	Да
Финансовые отчеты	Да	Да	Да	Да	Да
Мобильное приложение	Нет	Да	Нет	Нет	Да
Облачное хранение данных	Ограничено	Да	Нет	Нет	Да
Интеграция с оборудованием	Нет	Ограничено	Да	Да	Да
Стоимость	Высокая	Средняя	Высокая	Высокая	Средняя
Простота внедрения	Сложная	Легкая	Сложная	Сложная	Легкая

Ниже приведен сравнительный анализ существующих решений, предложенных на рынке ИТ сферы.

AutoFluent — комплексная система управления для автосервисов, которая предлагает инструменты для управления заказами, клиентами и запасами.

Функциональные возможности:

- управление заказами и запасами;
- crm для управления клиентами;
- финансовые отчеты и аналитика.

Преимущества:

- широкий функционал;
- интеграция с бухгалтерскими системами.

Недостатки:

- высокая стоимость;
- сложность внедрения и настройки.

Shop-Ware — облачная платформа для управления автосервисом, фокусирующаяся на автоматизации рабочего процесса и взаимодействии с клиентами.

Функциональные возможности:

- управление заказами и инвентаризацией;
- мобильное приложение для клиентов;
- встроенные инструменты для маркетинга.

Преимущества:

- интуитивный интерфейс;
- облачное хранение данных.

Недостатки:

- ограниченная функциональность в базовой версии;
- могут возникать проблемы с интеграцией.

Mitchell 1 — популярное ПО для управления автосервисами, предлагающее инструменты для диагностики, управления заказами и инвентарем.

Функциональные возможности:

- детальная диагностика и технические руководства;
- управление заказами и запасами;
- поддержка интеграции с другими системами.

Преимущества:

- обширная база данных по ремонту и техническому обслуживанию;
- надежность и долгосрочная поддержка.

Недостатки:

- высокая стоимость подписки;
- меньшая гибкость для небольших автосервисов.

ALLDATA предлагает обширную информацию о ремонте и диагностике, а также инструменты для управления автосервисами.

Функциональные возможности:

- подробные схемы ремонта и диагностики;
- управление заказами и клиентами;
- финансовые отчеты.

Преимущества:

- большая база данных по ремонту;
- интеграция с диагностическим оборудованием.

Недостатки:

- высокая стоимость доступа;
- сложность в освоении для новых пользователей.

1.1.5 Этапы разработки Web-сайта

В современном цифровом мире веб-сайт играет ключевую роль в успешном онлайн-присутствии. Создание сайта с нуля требует тщательного

планирования и последовательного выполнения этапов, чтобы разработать функциональный и привлекательный ресурс [1].

Исследование и планирование.

Этап исследования и планирования является одним из ключевых моментов в разработке веб-ресурса. На этом этапе происходит сбор информации, анализ целей и задач ресурса, а также определение его целевой аудитории. Полученные результаты служат основой для дальнейшей работы.

Определение целей.

В этот период определяются конкретные цели создания сайта, такие как продажа товаров или услуг, предоставление информации или привлечение новых клиентов. Цели должны быть измеримыми, конкретными и достижимыми.

Определение задач.

Задачи формулируются для достижения поставленных целей. Например, создание интернет-магазина, ведение активного блога или реализация системы онлайн-бронирования. Задачи должны быть прямо связаны с целями проекта и ясно определены.

Анализ целевой аудитории.

Этот этап включает в себя изучение целевой аудитории, ее возраста, пола, интересов, поведения и потребностей. Анализ аудитории помогает понять, какие функциональные возможности и дизайн будут наиболее подходящими для пользователей.

Разработка структуры.

На данном этапе определяется структура будущего сайта, включая основные разделы, подразделы и их иерархию. Это позволяет организовать информацию и обеспечить удобную навигацию для пользователей.

Создание макетов (wireframes).

Макеты представляют собой черновые версии страниц сайта, которые визуализируют компоновку элементов и расположение контента. Они

помогают понять, как будет выглядеть сайт и каким образом пользователи будут взаимодействовать с его элементами.

Выбор технологий и платформы для разработки.

На данном этапе происходит определение технологий и платформ, которые будут использоваться в процессе создания веб-сайта. Это включает выбор языка программирования (например, PHP, Python, Ruby), определение фреймворка, установку системы управления контентом (CMS) или использование различных инструментов и плагинов. Правильный выбор технологий и платформы имеет ключевое значение для обеспечения эффективности и безопасности сайта.

Планирование контента.

Этот этап связан с разработкой контентного плана для будущего веб-ресурса. Определяются страницы и разделы, которые будут представлены на сайте, а также контент, который будет размещен на каждой странице. Целью является создание четкого и структурированного контента, соответствующего поставленным целям и потребностям целевой аудитории.

Проведение конкурентного анализа.

На этом этапе осуществляется изучение конкурентов в выбранной нише и анализ их веб-сайтов. Это позволяет получить представление о текущих тенденциях, успешных решениях и возможностях для улучшения своего сайта. Важно выявить уникальные особенности и преимущества своего сайта по сравнению с конкурентами.

Оценка бюджета и ресурсов.

Это заключительный этап, на котором проводится оценка необходимых финансовых и временных ресурсов для разработки сайта. Определяются финансовые ограничения, доступные ресурсы и сроки выполнения проекта. Это позволяет спланировать и организовать работу с учетом установленных ресурсов и целей проекта.

Важно использовать различные методы исследования, такие как опросы, интервью, анализ конкурентов и сбор статистических данных, для получения более глубокого понимания целевой аудитории.

Дизайн и визуальное оформление.

После завершения этапа планирования следует приступить к разработке различных дизайн-вариантов и визуального оформления. Требуется создать уникальный дизайн, который соответствует бренду и целям веб-ресурса. Дизайнеры занимаются разработкой графических элементов, логотипов, иконок, а также выбором цветовой гаммы и шрифтов. Важным аспектом этого этапа является создание макетов страниц и пользовательского интерфейса для визуализации и оценки внешнего вида сайта.

На этом этапе важно не только создать привлекательный дизайн, но и обеспечить его соответствие бренду компании или проекта. Дизайнерам необходимо тщательно продумать каждый аспект визуального оформления, начиная от выбора цветовой палитры, которая должна быть согласована с корпоративными цветами, и заканчивая разработкой графических элементов, которые отражают уникальность и идентичность бренда.

Создание макетов страниц и интерфейса пользователя играет ключевую роль, поскольку это позволяет визуализировать, как будет выглядеть сайт в конечном итоге, и оценить его удобство использования для пользователей. Этот процесс включает в себя разработку структуры страниц, расположение элементов интерфейса, а также взаимодействие пользователей с различными элементами сайта.

Кроме того, важно уделить внимание адаптивности дизайна под различные устройства и экраны. С учетом разнообразия устройств, на которых пользователи могут просматривать сайт, необходимо создать дизайн, который будет хорошо выглядеть и функционировать на разных устройствах, включая компьютеры, планшеты и мобильные телефоны.

Фронтенд- разработка.

На этом этапе, фронтенд-разработка, осуществляется перевод графического дизайна в функциональный веб-интерфейс. Разработчики используют языки разметки HTML и CSS для создания структуры и внешнего оформления сайта. Сутью этого процесса является кодирование макетов дизайна таким образом, чтобы веб-страницы выглядели и функционировали именно так, как задумано дизайнерами.

Во время фронтенд-разработки создается адаптивный дизайн, который позволяет сайту корректно отображаться на различных устройствах и разрешениях экранов. Это важный аспект, учитывающий разнообразие устройств, на которых пользователи могут просматривать сайт.

Особое внимание уделяется также внедрению интерактивных элементов с помощью JavaScript. Эти элементы, такие как меню, формы, слайдеры и другие, делают сайт более функциональным и удобным для пользователей, позволяя им взаимодействовать с контентом. JavaScript используется для добавления динамики на страницу, обработки событий и создания интерактивных функций, что повышает уровень вовлеченности пользователей.

Ключевым аспектом фронтенд-разработки является обеспечение соответствия кода стандартам, а также оптимизация его производительности и доступности для всех пользователей, включая людей с ограниченными возможностями.

Бэкенд—разработка.

На этапе бэкенд-разработки создается серверная часть сайта, которая отвечает за обработку запросов пользователей, выполнение бизнес-логики и взаимодействие с базой данных. Разработчики выбирают язык программирования, такой как PHP, Python или Ruby, и фреймворк для более эффективной разработки. В ходе бэкенд-разработки реализуется функциональность, необходимая для работы сайта, такая как обработка форм, аутентификация и авторизация пользователей, а также взаимодействие с базой данных для сохранения и получения информации. Этот этап также

включает в себя разработку API (Application Programming Interface) для взаимодействия с фронтендом и другими системами.

Важной частью бэкенд-разработки является тестирование и отладка функциональности, чтобы убедиться в корректной работе всех компонентов. Разработчики проводят модульное тестирование каждой части бэкенда, а также интеграционное тестирование для проверки взаимодействия между различными модулями. Тестирование позволяет выявить и исправить ошибки до запуска сайта.

В процессе бэкенд-разработки осуществляется реализация безопасности сайта, включая защиту от уязвимостей, шифрование данных и обеспечение безопасного обмена информацией между клиентом и сервером. Это важный аспект, учитывая рост угроз в сфере кибербезопасности. Кроме того, бэкенд-разработка может включать в себя настройку серверной инфраструктуры, такой как установка и настройка веб-сервера, базы данных и других необходимых компонентов. Это обеспечивает готовность сервера к приему и обработке запросов пользователей.

Важной частью этапа бэкенд-разработки является также оптимизация производительности сайта. Разработчики стремятся улучшить скорость работы сайта, уменьшить время загрузки страниц и повысить отзывчивость интерфейса. Для этого могут применяться различные методы, такие как кэширование данных, оптимизация запросов к базе данных и использование асинхронных технологий.

Бэкенд-разработка включает в себя документирование кода и функциональности, чтобы облегчить поддержку и дальнейшее развитие сайта. Документация помогает другим разработчикам быстрее ориентироваться в коде и вносить изменения без нарушения работы системы.

Тестирование и отладка

После завершения разработки функциональности сайта разработчики проводят тестирование и отладку, чтобы удостовериться в его работоспособности и корректности. Этот процесс включает в себя

систематическую проверку на различных устройствах и в различных браузерах с целью обеспечения совместимости и одинакового отображения и работы сайта в любых условиях. Если выявляются ошибки или неполадки, разработчики немедленно приступают к их устранению и повторному тестированию.

Кроме того, на этапе тестирования уделяется внимание улучшению пользовательского опыта и решению обнаруженных проблем. Разработчики исправляют выявленные недочеты при использовании, оптимизируют интерфейс и взаимодействие пользователя с сайтом для обеспечения максимального комфорта и удобства использования.

Также, на этапе тестирования можно выделить следующие действия:

Тестирование производительности.

Проверка скорости загрузки страниц, время ответа сервера и общая производительность сайта. Это позволяет обнаружить и устранить узкие места, которые могут замедлить работу сайта.

Тестирование безопасности: Проведение анализа уязвимостей и атак на сайт для обеспечения его защиты от взломов и несанкционированного доступа к данным.

Тестирование совместимости.

Проверка работы сайта на различных операционных системах, устройствах и браузерах, чтобы убедиться, что он корректно отображается и функционирует на всех платформах.

Тестирование функциональности.

Проверка работы всех функциональных возможностей сайта, включая формы, кнопки, ссылки и другие элементы интерфейса, чтобы удостовериться, что все они работают правильно и без сбоев.

Тестирование резервного копирования.

Проверка системы резервного копирования данных для обеспечения их сохранности и восстановления в случае сбоев или потери информации.

Данные дополнительные шаги помогают обеспечить высокое качество и надежность функционирования сайта перед его запуском и публикацией.

Запуск и оптимизация

Этап запуска и оптимизации сайта является заключительным этапом в процессе разработки. На данном этапе выполняется ряд важных операций, направленных на подготовку сайта к публикации и обеспечение его эффективной работы, а также на удовлетворение потребностей пользователей. Рассмотрим более подробно основные шаги этого этапа.

Размещение на хостинге или сервере.

После завершения основной разработки сайта необходимо разместить его на выбранном хостинг-провайдере. Программисты загружают файлы сайта на сервер и настраивают соединение с базой данных, если она используется. Оптимальный выбор хостинг-провайдера обеспечивает стабильную работу сайта и быструю загрузку страниц.

Настройка DNS и доменного имени.

Для обеспечения доступа к сайту через доменное имя требуется настройка системы доменных имён (DNS). Это позволяет пользователям обращаться к сайту по удобному и запоминающемуся имени. Разработчики выполняют настройку DNS для привязки доменного имени к IP-адресу сайта.

Установка аналитических инструментов.

Для анализа посещаемости и поведения пользователей на сайте устанавливаются аналитические инструменты, такие как Google Analytics. Они предоставляют информацию о посещениях, источниках трафика, взаимодействии с сайтом и других метриках, помогая в оценке успеха проекта и принятии обоснованных решений для его улучшения.

Оптимизация для скорости загрузки страниц.

Этот этап включает оптимизацию кода, изображений, стилей и скриптов для ускорения загрузки страниц. Минимизация размера файлов, использование кэширования, сжатие изображений и другие техники

помогают улучшить скорость загрузки, что положительно сказывается на пользовательском опыте и рейтинге сайта в поисковых системах.

Оптимизация SEO-параметров.

Для повышения видимости сайта в поисковых системах осуществляется оптимизация SEO-параметров. Это включает использование ключевых слов в мета-тегах, создание информативных URL-адресов, установку правильных тегов заголовков, создание внутренней ссылочной структуры и разработку качественного контента.

Тестирование после оптимизации.

После внесения изменений и оптимизаций проводится тестирование сайта для проверки его работоспособности и эффективности. Тестировщики анализируют функциональность, производительность, адаптивность и SEO-параметры, чтобы убедиться в их соответствии требованиям и целям проекта.

Мониторинг и обслуживание.

После запуска сайта необходимо регулярно отслеживать его работу и обслуживать. Разработчики следят за работоспособностью, отслеживают метрики и аналитику, обновляют контент, добавляют новые функции и исправляют ошибки для обеспечения стабильной работы и достижения поставленных целей.

В завершающем этапе создания сайта, запуске и оптимизации, необходимо уделить внимание ряду важных аспектов для обеспечения его успешной работы и соответствия современным требованиям. Помимо размещения сайта на хостинге и настройки DNS для обеспечения доступности, важно установить аналитические инструменты, такие как Google Analytics, для отслеживания посещаемости и поведения пользователей. Оптимизация скорости загрузки страниц, включая оптимизацию кода, изображений и других элементов, играет ключевую роль в обеспечении удобства использования и улучшении пользовательского опыта. Также важно провести оптимизацию SEO-параметров, чтобы сайт был хорошо индексирован поисковыми системами и привлекал органический

трафик. Для обеспечения стабильной работы сайта и быстрой реакции на изменения необходимо постоянно мониторить его состояние, проводить анализ конкурентов и рынка, а также соблюдать требования законодательства и стандарты безопасности и доступности. Регулярное создание резервных копий данных и тестирование процедур восстановления помогают минимизировать риски потери информации и простоя сайта в случае чрезвычайных ситуаций. Все эти мероприятия дополняют друг друга и направлены на обеспечение эффективной работы и удовлетворения потребностей пользователей на протяжении всего существования сайта.

Поддержка и развитие.

Важным этапом после запуска сайта является его дальнейшая поддержка и развитие. Разработчики регулярно обновляют контент и функциональность, следят за работой ресурса и анализируют данные о посещаемости и поведении пользователей. Они реагируют на обратную связь пользователей, внося необходимые изменения. Кроме того, специалисты продолжают развивать сайт, учитывая изменения в требованиях и технологиях. Это может включать добавление новых функций, улучшение существующих, оптимизацию производительности и обеспечение безопасности сайта.

Постоянное обновление и поддержка помогают сохранить актуальность и конкурентоспособность вашего проекта. После запуска сайта важно не останавливаться на достигнутом, а продолжать работу над его улучшением и развитием. Разработчики не только регулярно обновляют контент и функциональность, но и внимательно отслеживают обратную связь от пользователей, внося коррективы и улучшения в соответствии с их потребностями.

Кроме того, они остаются в курсе последних технологических трендов и стараются адаптировать сайт под изменяющиеся условия и требования. Этот непрерывный процесс развития помогает не только сохранить

актуальность и конкурентоспособность ресурса, но и повысить его эффективность, удобство использования и безопасность для пользователей.

1.2 Описание выбранных технологий и инструментов для создания веб-приложений

В современном цифровом мире создание веб-приложений становится все более востребованным и распространенным. Однако, для того чтобы разработать качественное и функциональное веб-приложение, необходимо ознакомиться с основными технологиями и инструментами, используемыми в этом процессе. В данном обзоре мы рассмотрим ключевые составляющие создания веб-приложений, начиная от языков программирования и фреймворков, до баз данных и инструментов для развертывания. Этот обзор поможет вам понять основы технологий веб-разработки и выбрать подходящие инструменты для реализации проекта.

1.2.1 JSX (JavaScript XML) и React

React - это JavaScript-библиотека, разработанная командой Facebook, которая позволяет создавать пользовательские интерфейсы для веб-приложений. Основной принцип React – это создание компонентов, которые затем объединяются вместе для создания сложных пользовательских интерфейсов. Эти компоненты являются независимыми и могут быть повторно использованы в различных частях приложения, что делает код более модульным и легким для поддержки.

JSX формально является расширением синтаксиса JavaScript. Это дополнение, которое позволяет формировать деревья объектной модели документа (DOM) с помощью XML-подобного синтаксиса. Изначально разработанное Facebook для использования в React, JSX было принято несколькими другими веб-фреймворками. JSX, будучи синтаксическим улучшением, обычно трансформируется во вложенные вызовы функций JavaScript, структурно аналогичные исходному JSX.

React также предлагает удобный синтаксис для описания компонентов с помощью JSX.

JSX - это расширение JavaScript, которое позволяет писать код для React в более декларативном и интуитивном стиле. JSX объединяет HTML-подобный синтаксис с JavaScript, что делает написание компонентов React более понятным и эффективным.

React и JSX представляют собой мощный набор инструментов для разработки современных веб-приложений. React обеспечивает эффективное управление состоянием приложения и максимальную производительность благодаря виртуальному DOM, а JSX делает код более читаемым и интуитивно понятным.

Сочетание React и JSX открывает перед разработчиками множество возможностей для создания инновационных и отзывчивых веб-приложений, которые впечатляют пользователей своей функциональностью и производительностью.

1.2.2 Node.JS

Эта платформа, построенная на JavaScript, стала ключевым компонентом для создания высокопроизводительных и масштабируемых серверных приложений. Давайте рассмотрим, что такое Node.js, его особенности и преимущества.

Node.js - это среда выполнения JavaScript, построенная на движке V8 Chrome, который также используется в браузере Google Chrome. Однако в отличие от браузера, Node.js позволяет выполнять JavaScript на сервере, что открывает широкие возможности для создания серверных приложений. Данная платформа позволяет разработчикам писать серверный код на JavaScript, что делает его более привлекательным для веб-разработчиков, уже знакомых с этим языком программирования. Благодаря асинхронной и событийно-ориентированной природе JavaScript, Node.js обеспечивает высокую производительность и эффективное использование ресурсов сервера.

К особенностям Node.js можно отнести следующие.

Асинхронное программирование - Node.js использует асинхронные операции ввода-вывода (I/O), что позволяет обрабатывать множество запросов без блокировки исполнения кода. Это позволяет создавать масштабируемые и отзывчивые серверные приложения, способные эффективно обрабатывать большие объемы данных.

Модульная структура - Node.js поддерживает модульную структуру, что позволяет разработчикам легко организовывать свой код на небольшие и многоразовые модули. Это способствует повторному использованию кода, улучшает его читаемость и облегчает его поддержку [21].

Предоставление широких возможностей - Node.js имеет огромное количество модулей и библиотек, доступных через менеджера пакетов npm (Node Package Manager). Эти пакеты предоставляют различные инструменты и функциональность для разработки различных типов приложений, включая веб-серверы, API, микросервисы и многое другое.

Преимущества Node.js описаны ниже.

Высокая производительность - Благодаря асинхронной природе и использованию событийного цикла Node.js, приложения на этой платформе обеспечивают высокую производительность и отзывчивость даже при обработке больших объемов запросов.

Легкость масштабирования - Node.js обеспечивает простоту масштабирования приложений благодаря возможности горизонтального и вертикального масштабирования, а также использованию кластеризации для распределения нагрузки.

Единый язык программирования - использование JavaScript как языка программирования как на клиентской, так и на серверной стороне позволяет разработчикам создавать полноценные веб-приложения без необходимости изучения других языков программирования.

Node.js - это мощная и гибкая платформа для создания серверных приложений, обеспечивающая высокую производительность,

масштабируемость и удобство разработки. Его асинхронная природа и модульная структура делают его идеальным выбором для создания современных веб-приложений, способных эффективно обрабатывать большие объемы данных и обеспечивать отличное пользовательское взаимодействие.

Для создания веб-приложения для автоматизации процесса обработки заявок от клиентов в сервисном центре были выбраны инструменты React и Node.js. React был выбран для фронтенд-части приложения из-за его гибкости, эффективности и возможности создания масштабируемых пользовательских интерфейсов. Благодаря виртуальному DOM и компонентному подходу, React позволяет разрабатывать сложные интерфейсы, управлять состоянием приложения и повышать производительность. Node.js был выбран для бэкенд-части приложения из-за его асинхронной природы, возможности обработки большого количества запросов и простоты создания API. Кроме того, единый язык программирования (JavaScript) для фронтенда и бэкенда упрощает разработку, сопровождение и масштабирование приложения.

Выводы по главе 1

В главе 1 представлен обзор основных технологий и инструментов, используемых для разработки веб-приложений. JSX и React рассматриваются для внешней разработки, а также Node.js для внутренней части приложения. Эти инструменты были выбраны из-за их эффективности, гибкости и масштабируемости приложений.

Глава 2 Проектирование веб-приложения

2.1 Определение требований к функционалу приложения на основе анализа потребностей сервисного центра

Определение требований к функционалу веб-приложения для автосервиса представлено ниже [11].

Регистрация и авторизация пользователей - возможность для клиентов ввести свои данные для создания учетной записи или авторизации через социальные сети [2].

Пользователи веб-приложения должны иметь возможность создать учетную запись для доступа ко всем функциям приложения [14]. Это предполагает введение личных данных пользователей для создания профиля, который будет использоваться для идентификации и аутентификации при последующем входе в систему. Кроме того, для удобства пользователей также должна быть предоставлена возможность авторизации через аккаунты в социальных сетях, таких как Facebook, Google, Twitter и др. [18]

Рассмотрим элементы разрабатываемого веб-приложения.

Форма регистрации.

Пользователи должны заполнить форму регистрации, в которой будут указывать свои персональные данные, такие как имя, адрес электронной почты, номер телефона и пароль.

Проверка уникальности данных [19].

При регистрации система должна проверять уникальность введенной электронной почты, чтобы исключить возможность создания нескольких учетных записей с одним адресом электронной почты.

Подтверждение учетной записи.

После регистрации на указанный пользователем адрес электронной почты должно отправляться письмо с ссылкой для подтверждения создания учетной записи.

Авторизация через социальные сети.

Пользователи должны иметь возможность войти в приложение, используя свои учетные записи в социальных сетях. Это позволит им избежать необходимости запоминать и вводить дополнительные пароли.

Восстановление пароля.

Пользователи должны иметь возможность восстановить свой пароль в случае его утери или забытья. Для этого им должна быть предоставлена функция сброса пароля, которая будет использовать процедуру восстановления через электронную почту или смс-сообщение.

Это требование направлено на обеспечение безопасного и удобного доступа пользователей к функциям веб-приложения, а также на повышение уровня доверия и удовлетворенности пользователей.

Заполнение формы заявки: пользователи должны иметь возможность вводить информацию о себе, марке автомобиля, выбирать тип обслуживания (например, техническое обслуживание или ремонт), а также описывать проблему с автомобилем.

Пользователи должны иметь возможность вводить следующую информацию в форму заявки: персональные данные (имя, фамилия, контактный телефон, адрес электронной почты), информацию о марке автомобиля (модель, год выпуска, VIN-код при необходимости), выбор типа обслуживания (например, техническое обслуживание, ремонт, замена запчастей) и описание проблемы с автомобилем.

Все поля формы обязательны для заполнения, и при вводе информации в поля контактного телефона или адреса электронной почты должна осуществляться проверка корректности введенных данных. Информация, заполненная в форме заявки, должна сохраняться в базе данных приложения.

Администратору приложения предоставляется доступ к интерфейсу управления заявками, где он может просматривать новые заявки, редактировать их статусы и управлять приоритетами. Пользователи и администраторы должны получать уведомления о статусе заявки для

оперативного реагирования через электронную почту или мобильное приложение.

Это требование направлено на обеспечение удобного и эффективного способа подачи заявок на обслуживание автомобилей, а также на обеспечение прозрачности и эффективности управления заявками.

Отправка заявки: после заполнения всех необходимых данных пользователь должен иметь возможность отправить заявку на обслуживание.

После того как пользователь успешно заполнил все необходимые поля формы заявки, он должен иметь возможность отправить заявку на обслуживание с помощью специальной кнопки или ссылки. Это действие должно быть интуитивно понятным и доступным для пользователя.

После отправки заявки пользователь должен получить подтверждение о ее успешном отправлении. Это может быть сообщение на экране с информацией о том, что заявка успешно отправлена, или уведомление по электронной почте.

Заявка должна быть немедленно добавлена в базу данных приложения для дальнейшей обработки администратором сервисного центра. При отправке заявки должна осуществляться проверка на корректность заполненных данных, а также защита от нежелательных или злонамеренных действий с помощью механизмов проверки безопасности, таких как капча или проверка на ботов.

Это требование направлено на обеспечение простоты и удобства процесса отправки заявок для пользователей, а также на гарантированную доставку заявок в систему для последующей обработки администраторами сервисного центра.

Просмотр заявок администратором: администратор должен иметь доступ к списку всех заявок, отправленных клиентами, с возможностью фильтрации и поиска по параметрам.

Администратору должен быть предоставлен доступ к списку всех заявок, отправленных клиентами [27]. Это включает в себя возможность

просмотра всех деталей каждой заявки, таких как данные клиента, марка автомобиля, тип обслуживания и описание проблемы. Диаграмма вариантов использования представлена на рисунке 4.

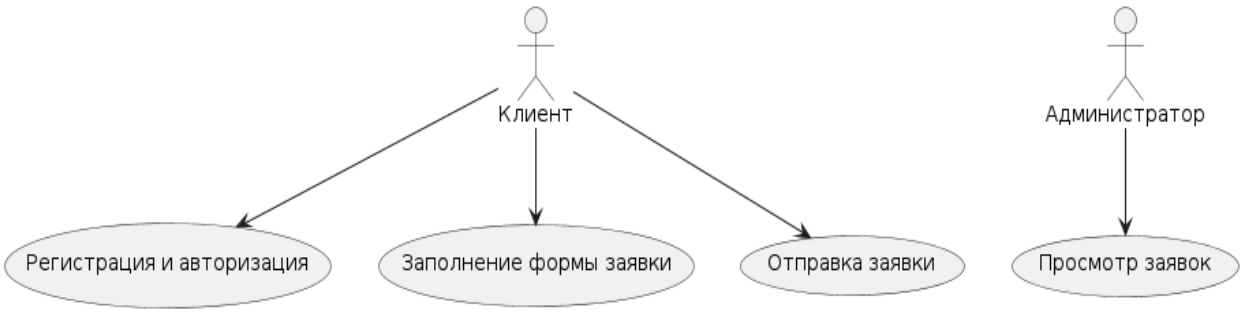

Рисунок 4 - Диаграмма вариантов использования для веб-приложения автосервиса

Для удобства навигации и быстрого поиска администратору необходимо предоставить инструменты фильтрации и поиска. Это позволит администратору быстро находить нужные заявки по различным параметрам, таким как дата отправки, статус заявки, марка автомобиля или тип обслуживания [28].

Просмотр списка заявок должен быть удобным и информативным, чтобы администратор мог легко оценить текущую загруженность сервисного центра и принимать соответствующие решения. Возможность сортировки заявок по различным критериям также будет полезной для управления рабочим процессом.

Это требование направлено на обеспечение эффективного и удобного управления заявками администраторами сервисного центра, что позволит им оперативно реагировать на поступающие запросы клиентов и оптимизировать рабочий процесс [29].

Обсуждение заявок с мастером: администратор должен иметь возможность просматривать детали каждой заявки и обсуждать их с мастером для назначения даты и времени осмотра автомобиля.

Администратору необходимо предоставить возможность просматривать все детали каждой заявки, отправленной клиентами. Это включает в себя информацию о клиенте, марке автомобиля, выбранном типе обслуживания и описании проблемы с автомобилем.

После просмотра деталей заявки администратор должен иметь возможность обсудить ее с мастером. Это позволит им совместно принять решение о необходимости осмотра автомобиля и назначить дату и время его проведения.

Обсуждение заявок с мастером должно быть удобным и эффективным процессом, который позволит быстро принимать решения и назначать работы. Для этого администратору нужно предоставить инструменты связи с мастером, такие как система обмена сообщениями или функция онлайн-конференций.

Это требование направлено на обеспечение согласованной работы между администратором и мастером сервисного центра, что позволит эффективно управлять рабочим процессом и обеспечить качественное обслуживание клиентов.

Отображение статуса заявки: пользователи и администратор должны видеть статус каждой заявки (например, «в обработке», «ожидает ремонта», «завершено»), чтобы иметь представление о текущем состоянии обслуживания.

Пользователи и администраторы должны иметь возможность видеть текущий статус каждой заявки. Это важно для того, чтобы они могли следить за ходом обработки и обслуживания автомобилей.

Для пользователей, отправивших заявку, важно видеть, в каком состоянии находится их запрос. Например, статусы могут включать в себя «в обработке», «ожидает ремонта», «выполняется ремонт», «завершено» и т.д. Это поможет им понять, когда ожидать завершения обслуживания своего автомобиля и планировать свое время.

Администратору также необходимо иметь доступ к статусу каждой заявки. Это позволит им эффективно управлять рабочим процессом, оптимизировать распределение задач между мастерами и обеспечивать своевременное выполнение заявок.

Отображение статуса заявки должно быть интуитивно понятным и доступным на главной странице веб-приложения. Это позволит пользователям и администраторам легко отслеживать текущее состояние обслуживания и принимать соответствующие решения.

Оценка и обратная связь: после завершения обслуживания клиентам должна предоставляться возможность оценить качество обслуживания и оставить обратную связь.

После завершения обслуживания каждой заявки клиентам необходимо предоставить возможность выразить свою оценку качества предоставленных услуг и оставить обратную связь. Это позволит сервисному центру оценить уровень удовлетворенности клиентов и улучшить качество обслуживания в будущем. Клиентам должна быть предложена форма для заполнения, в которой они могут оценить различные аспекты обслуживания, такие как профессионализм мастеров, скорость выполнения работ, качество оборудования и использованных материалов, а также общее впечатление от сервиса. Кроме того, клиентам следует предоставить возможность оставить комментарий или предложение по улучшению сервиса.

Эта информация будет полезна для сервисного центра, чтобы оценить свои сильные и слабые стороны, а также понять, какие аспекты обслуживания требуют дополнительного внимания и улучшений. Администратору также необходимо иметь доступ к этим оценкам и отзывам для анализа и принятия соответствующих мер для улучшения качества обслуживания

Система оценки и обратной связи должна быть интегрирована в веб-приложение сервисного центра и быть легкой в использовании для клиентов,

чтобы максимально упростить процесс выражения их мнения о предоставленных услугах.

Уведомления: Пользователи должны получать уведомления о статусе своей заявки, изменениях в расписании обслуживания и другой важной информации.

Для обеспечения удобства пользователей, им необходимо предоставить систему уведомлений о статусе и изменениях в их заявках, а также другой важной информации. Это поможет пользователям быть в курсе текущего состояния своих запросов и своевременно реагировать на любые изменения.

Пользователи должны иметь возможность выбрать способ получения уведомлений, такой как SMS-сообщения, электронные письма или мобильные уведомления через приложение. При этом уведомления должны быть информативными и содержать ключевую информацию о статусе заявки или изменениях в расписании обслуживания. Администратору также необходимо иметь возможность отправлять уведомления пользователям, например, о подтверждении получения заявки, изменениях в расписании обслуживания или запросах на дополнительную информацию.

Эффективная система уведомлений поможет повысить уровень удовлетворенности клиентов, обеспечивая им своевременную и полезную информацию о состоянии их заявок и процессе обслуживания.

Администрирование и настройка: Администратор должен иметь доступ к панели управления для настройки параметров приложения, управления пользователями и просмотра отчетов о заявках.

Для обеспечения эффективной работы веб-приложения администратору необходим доступ к специальной панели управления, где он сможет настраивать различные параметры системы, управлять пользователями и анализировать данные о заявках. Панель управления должна предоставлять администратору возможность редактировать тексты и метаданные приложения, такие как названия сервисов, описания услуг и другие сведения, которые могут меняться со временем.

Кроме того, администратор должен иметь возможность управлять пользователями системы, включая создание, блокировку, удаление или редактирование учетных записей клиентов и сотрудников сервисного центра.

Для эффективного мониторинга работы сервисного центра администратору также необходим доступ к отчетам о заявках, включая общую статистику по обработанным заявкам, времени выполнения и другим ключевым метрикам.

Таким образом, наличие панели управления с различными инструментами настройки и администрирования позволит администратору эффективно управлять и контролировать работу веб-приложения для обработки заявок клиентов в сервисном центре.

Эти требования основаны на потребностях сервисного центра в удобной и эффективной системе для управления заявками на обслуживание автомобилей, с целью упрощения процесса и повышения прозрачности для клиентов и персонала автосервиса.

2.2 Разработка структуры базы данных и проектирование пользовательского интерфейса

2.2.1 Разработка структуры базы данных

Для разработки структуры базы данных, предназначенной для хранения информации о заявках и клиентах, используется MongoDB. MongoDB является NoSQL базой данных, потому что она не использует традиционную табличную модель хранения данных, как реляционные базы данных [25]. Вместо этого MongoDB использует документно-ориентированную модель данных, где данные хранятся в виде JSON-подобных документов. Это означает, что каждая запись в базе данных представляется в виде документа, который содержит пары ключ-значение [26]. Эта гибкая модель позволяет эффективно хранить и обрабатывать неструктурированные или полуструктурированные данные, такие как

текстовые документы, JSON-объекты, изображения и другие типы файлов [28] [29].

База данных MongoDB облачная, что означает, что она работает в облачной среде, где ресурсы доступны через Интернет. Это обеспечивает высокую доступность и масштабируемость системы, так как данные могут быть хранены и обрабатываться на распределенной инфраструктуре серверов. При необходимости можно легко масштабировать базу данных, добавляя новые серверы или увеличивая ресурсы существующих серверов, что позволяет обеспечить высокую производительность и отказоустойчивость системы.

Процесс масштабирования баз данных зависит от архитектуры базы данных и используемых технологий. В общих чертах, существуют два основных подхода к масштабированию баз данных: вертикальное и горизонтальное масштабирование.

Вертикальное масштабирование (Scaling Up):

- этот метод заключается в увеличении мощности отдельных серверов баз данных, путем увеличения процессоров, памяти или хранилища данных на каждом сервере;
- вертикальное масштабирование прост в реализации, так как не требует изменений в приложениях, которые используют базу данных;
- однако этот подход имеет ограничения, связанные с максимальными возможностями мощности отдельного сервера.

Горизонтальное масштабирование (Scaling Out):

- при горизонтальном масштабировании база данных распределяется на несколько серверов, называемых узлами или шардами. Каждый узел хранит только часть данных;
- этот метод обеспечивает более высокую масштабируемость, так как можно добавлять новые узлы по мере необходимости;

- горизонтальное масштабирование требует разделения данных между узлами и настройки кластера для балансировки нагрузки и обеспечения согласованности данных;
- хорошо подходит для ситуаций с высоким объемом запросов или большим количеством данных.

Процесс масштабирования баз данных включает в себя оценку текущих потребностей и прогнозирование будущего роста, выбор подходящего метода масштабирования, настройку и конфигурацию новых серверов или узлов, а также мониторинг производительности системы после внесения изменений.

Express является одним из самых популярных фреймворков для создания веб-приложений на языке JavaScript с использованием Node.js. Его удобство и популярность объясняются несколькими ключевыми причинами.

Простота использования - Express предоставляет простой и интуитивно понятный интерфейс для создания веб-приложений. Он предоставляет широкий набор готовых функций и методов для обработки маршрутов, запросов и ответов.

Минималистичный подход - Express поощряет минимализм и предоставляет только базовый набор функциональности, не навязывая разработчику излишних конвенций. Это позволяет разработчику свободно структурировать приложение в соответствии с его потребностями.

Гибкость - Express позволяет разработчику использовать любые другие библиотеки и модули Node.js для расширения функциональности веб-приложения. Он не навязывает строгих правил и позволяет выбирать подходящие инструменты для конкретной задачи.

Мощный механизм маршрутизации - Express предоставляет гибкий и мощный механизм маршрутизации, который позволяет легко определять обработчики для различных URL-адресов и HTTP-методов.

Широкое сообщество и поддержка - Express имеет огромное сообщество разработчиков и активно поддерживается. Это означает наличие

обширной документации, множества сторонних модулей и плагинов, а также возможность получить помощь и поддержку от других разработчиков.

Взаимодействие Node.js и JavaScript основано на том, что Node.js позволяет выполнять JavaScript-код на стороне сервера. Это означает, что разработчик может использовать JavaScript для создания как фронтенд, так и бэкенд части приложения, что обеспечивает единообразие языка программирования на всем стеке технологий. Кроме того, Node.js обеспечивает высокую производительность и эффективное управление асинхронными операциями, что особенно важно при обработке запросов к базе данных и взаимодействии с клиентскими приложениями через API.

Данная архитектура позволяет создать масштабируемое и гибкое приложение для управления заявками и клиентами, обеспечивая быстрый доступ к данным и простоту в разработке и поддержке (см. рисунок 5).

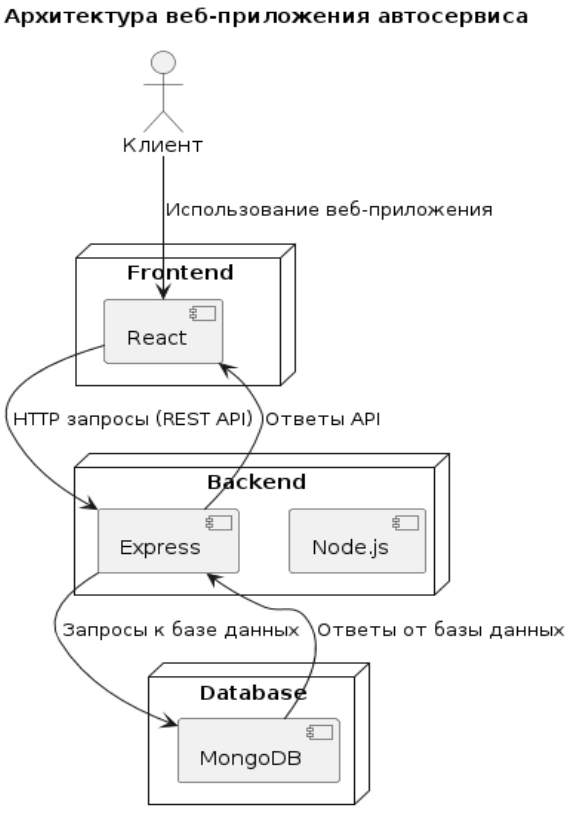

Рисунок 5 - Архитектура веб-приложения автосервиса

MongoDB представляет собой мощную базу данных, основанную на документно-ориентированной модели данных, что делает её идеальным выбором для хранения информации о заявках и клиентах. Облачная архитектура MongoDB обеспечивает высокую доступность и масштабируемость системы, а гибкость масштабирования позволяет эффективно управлять ростом данных и нагрузкой. Взаимодействие с Express и Node.js обеспечивает производительность и гибкость в разработке веб-приложения, что позволяет создать удобный и эффективный пользовательский опыт.

Разработка веб-приложения для автосервиса требует определения функциональных требований, которые включают в себя регистрацию и авторизацию пользователей, заполнение формы заявки, отправку заявки и просмотр заявок администратором. Каждое требование направлено на обеспечение удобства и эффективности использования приложения как для клиентов, так и для администраторов.

2.2.2 Модели страниц сайта и их взаимодействие

MongoDB не использует термины "первичные" и "вторичные" ключи, как реляционные базы данных. Однако, можно создавать индексы, чтобы ускорить поиск данных [16]. Даже без ключей, можно построить модель данных, которая отражает отношения между объектами (см. рисунок 6) [17].

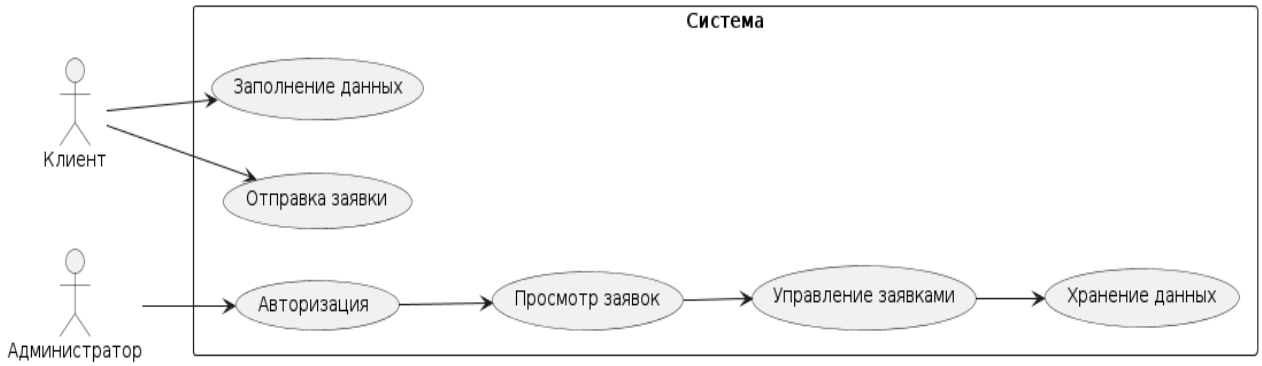

Рисунок 6 - Взаимодействие между клиентом, администратором и системой

В данном случае, и есть коллекция «Архив», она может содержать ссылки на заявки, которые были архивированы. Например, каждый элемент в коллекции «Архив» может хранить ID заявки из коллекции «Заявки». При необходимости получить информацию о заявке из архива, вы можете использовать этот ID для поиска [3].

Хотя MongoDB не предоставляет автоматического контроля за связями между документами, можно следить за их целостностью самостоятельно, обеспечивая, чтобы ссылки оставались корректными при добавлении, изменении или удалении данных.

MongoDB предоставляет гибкую модель данных, где отношения между объектами могут быть выражены через ссылки. Например, в коллекции "Архив" можно хранить ссылки на заявки из коллекции "Заявки", что позволяет эффективно организовать данные.

Хотя MongoDB не обеспечивает автоматического контроля за связями между документами, разработчики могут самостоятельно следить за целостностью данных. Например, при изменении или удалении данных, ссылки между объектами должны оставаться корректными.

Таким образом, MongoDB предоставляет инструменты для построения надежной и гибкой модели данных, где разработчики могут эффективно управлять отношениями между объектами и обеспечивать целостность данных в приложениях.

MongoDB представляет собой мощную базу данных, основанную на документно-ориентированной модели данных, что делает её идеальным выбором для хранения информации о заявках и клиентах. Облачная архитектура MongoDB обеспечивает высокую доступность и масштабируемость системы, а гибкость масштабирования позволяет эффективно управлять ростом данных и нагрузкой. Взаимодействие с Express и Node.js обеспечивает производительность и гибкость в разработке веб-приложения, что позволяет создать удобный и эффективный пользовательский опыт.

2.2.3 Проектирование интерфейса

Проектирование интерфейса – это процесс создания пользовательского опыта, который делает использование продукта легким, приятным и эффективным для пользователей [7]. Этот процесс включает в себя анализ потребностей пользователей, определение функциональности и создание дизайна, который обеспечивает интуитивное и удобное взаимодействие [8]. Форма заявки клиента представлена на рисунке 7.

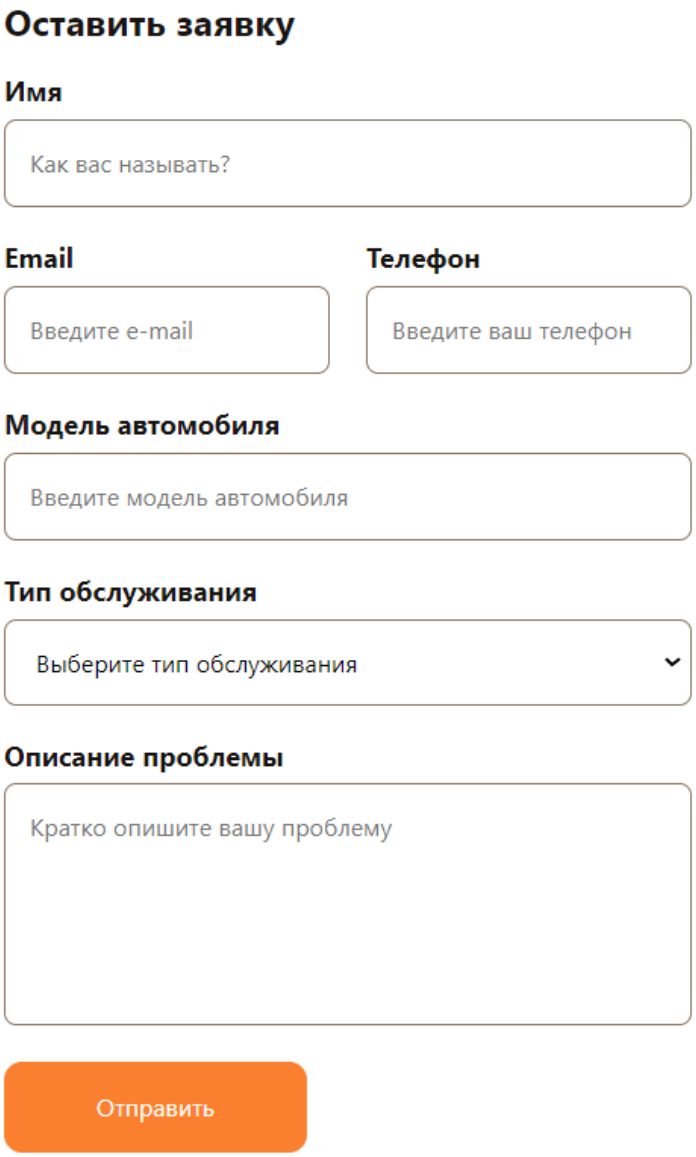

Рисунок 7 - Окно формы заявки клиента

Имя (Name):

- функция (это поле предназначено для ввода имени пользователя или клиента);
- использование (пользователь вводит своё имя для идентификации).

Email:

- функция (поле для ввода адреса электронной почты);
- использование (позволяет пользователям оставлять свой контактный адрес для обратной связи).

Телефон (Phone):

- функция (позволяет пользователю ввести свой контактный номер телефона);
- использование (дает возможность операторам связаться с клиентом для уточнения деталей или предоставления дополнительной информации).

Модель автомобиля (Car Model):

- функция (поле для ввода модели автомобиля клиента);
- использование (позволяет пользователю указать конкретную модель автомобиля, с которой у него возникли проблемы или которую он хочет обслужить).

Тип обслуживания (Service Type):

- функция (позволяет пользователю выбрать тип обслуживания, который он требует);
- использование (пользователь может выбрать из предложенных вариантов, таких как техническое обслуживание, ремонт, замена запчастей и т.д.).

Описание проблемы (Problem Description):

- функция (поле для описания проблемы или запроса пользователя);
- использование (Пользователь может описать подробности проблемы или запроса, что поможет специалистам более точно понять ситуацию и предложить соответствующее решение).

При нажатии кнопки «Отправить», данные формы отправляются на сервер [9] [10]. Кнопка «Панель администратора», перенаправляет в административную панель (см. рисунок 8). Если администратор не авторизован, то он его перенаправляет на страницу входа, где он может войти, нажав кнопку «Войти» [12].

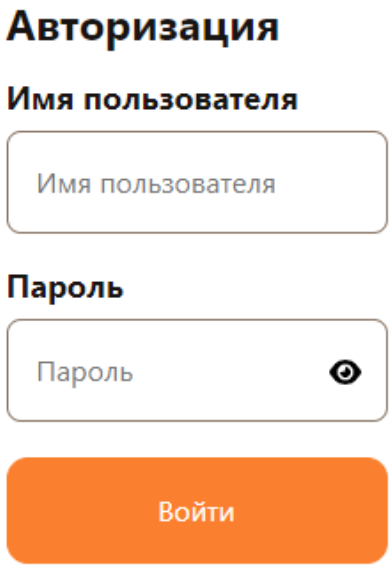

Рисунок 8 - Панель авторизации администратора

В административной панели есть две кнопки: «Заявки» и «Архив» (см. рисунок 9).

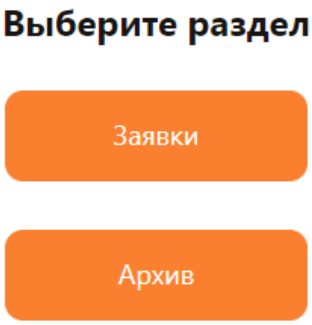

Рисунок 9 - Панель управления

В разделе «Заявки» администратор может переместить заявку в архив, нажав соответствующую кнопку (см. рисунок 10).

Рисунок 10 - Панель просмотра заявок

В разделе «Архив» администратор может удалить заявку также с помощью кнопки. Кнопка "Выход" позволяет выйти из аккаунта администратора (см. рисунок 11).

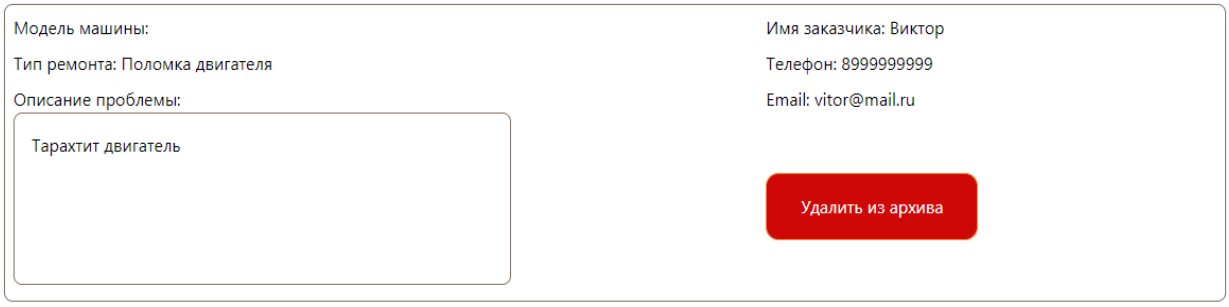

Рисунок 11 - Панель просмотра архива

Сайт функционирует на основе обратной связи между администратором и клиентом. Клиент имеет возможность заполнить заявку без необходимости регистрации на сайте. Это сделано с целью уменьшения нагрузки на сайт и повышения удобства использования.

Вывод: Проектирование интерфейса играет ключевую роль в создании удобного и приятного пользовательского опыта. В данном случае, мы разработали интерфейс для заполнения заявки клиентами и управления ими администраторами.

Для клиентов предусмотрены поля для ввода основной информации, такой как имя, электронная почта, телефон, модель автомобиля, тип обслуживания и описание проблемы. После заполнения формы клиент может отправить заявку, а в случае необходимости связаться с администратором.

Администратору доступна административная панель, где он может управлять заявками. В разделе «Заявки» он может перемещать их в архив, а в разделе «Архив» удалять. Также доступна функция выхода из аккаунта.

Сайт организован таким образом, чтобы обеспечить удобство использования для клиентов и эффективное управление заявками для администраторов, основываясь на принципах простоты, понятности и эффективности.

Выводы по главе 2

Глава 2 представляет собой описание процесса разработки веб-приложения, включая выбор базы данных MongoDB и проектирование пользовательского интерфейса с учетом удобства использования.

Глава 3 Реализация веб-приложения

3.1 Разработка бэкенда

Как только макет веб-приложения готов, начинается важный этап - разработка бэкенда. Этот этап нередко считается сердцем и душой проекта, так как именно бэкенд обеспечивает функциональность всего приложения. Без него ничего не заработает.

Бэкенд отвечает за обработку запросов от пользователей, управление данными и взаимодействие с базой данных. Он выполняет ряд ключевых функций, включая аутентификацию пользователей, проверку прав доступа, обработку и хранение данных, а также обеспечение безопасности и защиты от угроз.

Важно правильно спроектировать и реализовать бэкенд, чтобы он соответствовал требованиям приложения и обеспечивал его эффективную работу. Это включает в себя выбор подходящей архитектуры приложения, определение модели данных и API, а также написание эффективного и надежного кода.

Бэкенд также играет ключевую роль в обеспечении масштабируемости и производительности приложения. Он должен быть способен обрабатывать большие объемы данных и запросов, а также масштабироваться при увеличении нагрузки на систему.

В процессе разработки бэкенда важно учитывать требования безопасности и защиты данных. Это включает в себя реализацию механизмов аутентификации и авторизации, обеспечение защиты от атак и утечек данных, а также соблюдение соответствующих стандартов и регуляций.

Таким образом, бэкенд является неотъемлемой частью веб-приложения, обеспечивающей его функциональность, производительность и безопасность. Его правильная разработка и поддержка играют ключевую роль в успехе проекта.

```
import express from "express";
import mongoose from "mongoose";
import {
  create,
  getOne,
  getAll,
  remove,
  transfer,
} from "./controllers/Request";
import {
  getOneArchive,
  getAllArchive,
  removeArchive,
} from "./controllers/Archive";
import { login } from "./controllers/Login";
import cors from "cors";
const app = express();
const port = 4444;
app.use(cors());
mongoose
  .connect(
"mongodb+srv://admin:wwwwww@cluster0.gvhafjn.mongodb.net/car-service-app"
  )
  .then(() => {
    console.log("Подключен к MongoDB");
  })
  .catch((err) => {
    console.log(err);
  });
```

```
app.use(express.json());
// Запросы
app.post("/request", create);
app.get("/request/:id", getOne);
app.get("/request", getAll);
app.delete("/request/:id", remove);
app.post("/request/transfer/:id", transfer);
app.post("/login", login);
app.get("/archive/:id", getOneArchive);
app.get("/archive", getAllArchive);
app.delete("/archive/:id", removeArchive);
//Конец запросов
app.listen(port, () => {
console.log(`Приложение работает на порте: ${port}`);
});
export default app;
```

Файл App.ts является главным файлом бэкенда, который выступает в роли корневого файла для всего приложения. Он является основной точкой входа, откуда начинается выполнение серверного кода. В этом файле происходит инициализация и настройка сервера, включая установку порта прослушивания, настройку маршрутов и обработчиков запросов, а также подключение всех необходимых контроллеров и промежуточных слоев приложения.

Инициализация сервера в файле App.ts может включать в себя создание экземпляра сервера с использованием фреймворка, например, Express.js, установку middleware для обработки запросов, настройку маршрутов для взаимодействия с клиентом и другими системами, а также настройку подключения к базе данных или другим внешним сервисам.

Подключение всех необходимых контроллеров и промежуточных слоев приложения означает импорт соответствующих модулей и классов, которые

отвечают за обработку запросов, валидацию данных, авторизацию и другие задачи, необходимые для функционирования приложения. Эти компоненты могут быть организованы в виде модулей или классов и импортированы в файл App.ts для использования в процессе работы сервера.

```
import ArchiveModel from "../models/Archive.model";
export const getOneArchive = async (req: any, res: any) => {
 try {
  const reqId = req.params.id;
  const doc = await ArchiveModel.findOne({ _id: reqId }).exec();
  if (!doc) {
   res.status(404).json({ message: "Заявка не найдена" });
  } else {
   res.json(doc);
  }
 } catch (error) {
  console.log(error);
  res.status(500).json({ message: "Что-то пошло не так, попробуйте снова" });
 }
};
export const getAllArchive = async (req: any, res: any) => {
 try {
  const data = await ArchiveModel.find();
  res.json(data);
 } catch (error) {
  console.error("Ошибка при получении данных:", error);
  res.status(500).send("Ошибка получения данных, повторите снова");
 }
};
export const removeArchive = async (req: any, res: any) => {
```

```
    try {
      const reqId = req.params.id;
      const doc = await ArchiveModel.findByIdAndDelete({ _id: reqId }).exec();
      if (!doc) {
        res.status(404).json({ message: "Заявка не найдена" });
      } else {
        res.status(200).json({ message: "Заявка удалена" });
      }
    } catch (error) {
      console.log(error);
      res.status(500).json({ message: "Что-то пошло не так, попробуйте снова" });
    }
  };
```

Файл Archive.ts является контроллером, который обрабатывает запросы, связанные с архивом данных в приложении. В контексте серверного приложения, этот контроллер предоставляет интерфейс для взаимодействия с архивом, включая чтение и удаление данных.

Он содержит методы, которые позволяют выполнять следующие действия:

– получение всех записей из базы данных: Этот метод используется для извлечения всех записей из архива данных в базе данных. Он может быть вызван при запросе клиента на отображение всех доступных записей в архиве;

– получение одной записи по идентификатору (id): Этот метод позволяет получить конкретную запись из архива данных по её уникальному идентификатору. Обычно этот идентификатор передается в запросе клиента, и контроллер использует его для поиска соответствующей записи в базе данных;

– удаление записи по её идентификатору: Этот метод используется для удаления конкретной записи из архива данных на основе её идентификатора. После получения запроса на удаление записи, контроллер ищет запись с указанным идентификатором в базе данных и удаляет её, если она существует.

Таким образом, файл Archive.ts предоставляет удобный интерфейс для работы с архивом данных в приложении, обеспечивая методы для чтения и удаления записей из базы данных.

```
import jwt from "jsonwebtoken";
import dotenv from "dotenv";
dotenv.config();
export const login = async (req: any, res: any) => {
  const secretKey = process.env.SECRET_KEY;
  const { username, password } = req.body;
  if (username === "admin" && password === "1") {
    const token = jwt.sign({ username }, secretKey!, { expiresIn: "1h" });
    res.json({ token });
  } else {
    res.status(401).json({ error: "Invalid username or password" });
  }
};
```

Файл Login.ts является контроллером, который отвечает за процесс аутентификации администратора в системе. В контексте веб-приложения, этот контроллер обрабатывает запросы от администратора, связанные с входом в систему.

Он содержит логику, которая включает в себя следующие основные шаги:

Шаг 1. Проверка учетных данных администратора: При получении запроса на вход в систему, контроллер проверяет предоставленные учетные данные (например, имя пользователя и пароль) на соответствие данным,

хранящимся в базе данных или другом источнике аутентификации. Эта проверка может включать аутентификацию через хеширование пароля или другие методы безопасности.

Шаг 2. Создание сеанса аутентификации: Если предоставленные учетные данные верны, контроллер создает сеанс аутентификации для администратора. Этот сеанс обычно содержит уникальный идентификатор или токен, который используется для идентификации администратора в последующих запросах. Такой подход обеспечивает безопасность и приватность в процессе аутентификации.

Шаг 3. Предоставление доступа к защищенным ресурсам: После успешной аутентификации контроллер предоставляет администратору доступ к защищенным ресурсам или функциям системы. Это может включать в себя перенаправление на страницу администратора или предоставление специальных прав доступа к определенным функциям приложения.

Таким образом, файл Login.ts играет важную роль в обеспечении безопасности системы, контролируя доступ администратора к защищенным ресурсам и функциям приложения через процесс аутентификации.

```
import RequestModel from "../models/Request.model";
import ArchiveModel from "../models/Archive.model";
export const transfer = async (req: any, res: any) => {
  const reqId = req.params.id;
  try {
    const requestData = await RequestModel.findById(reqId).exec();
    if (!requestData) {
      return res.status(404).json({
        message: "Данные не найдены",
      });
    }
    const { name, email, phone, breakdownType, carModel, message } =
```

```
        requestData;
      const archiveData = new ArchiveModel({
        name,
        email,
        phone,
        carModel,
        breakdownType,
        message,
      });
      await archiveData.save();
      await RequestModel.findByIdAndDelete(reqId).exec();
      res.status(200).json({
        message: "Данные перенесены",
      });
    } catch (error) {
      console.error(error);
      res.status(500).json({
        message: "Не удалось перенести данные, попробуйте снова",
      });
    }
  };
  export const create = async (req: any, res: any) => {
    const { name, email, phone, carModel, breakdownType, message } =
req.body;
    const doc = new RequestModel({
      name,
      email,
      phone,
      carModel,
      breakdownType,
```

```
      message,
    });
    try {
      const request = await doc.save();
      res.json(request);
    } catch (err) {
      console.log(err);
      res.status(500).json({
        message: "Не удалось отправить заявку, попробуйте снова",
      });
    }
  };
  export const remove = async (req: any, res: any) => {
    try {
      const reqId = req.params.id;
      const doc = await RequestModel.findByIdAndDelete({ _id: reqId }).exec();
      if (!doc) {
        res.status(404).json({ message: "Заявка не найдена" });
      } else {
        res.status(200).json({ message: "Заявка удалена" });
      }
    } catch (error) {
      console.log(error);
      res.status(500).json({ message: "Что-то пошло не так, попробуйте снова" });
    }
  };
  export const getOne = async (req: any, res: any) => {
    try {
```

```
    const reqId = req.params.id;
    const doc = await RequestModel.findOne({ _id: reqId }).exec();
    if (!doc) {
      res.status(404).json({ message: "Заявка не найдена" });
    } else {
      res.json(doc);
    }
  } catch (error) {
    console.log(error);
    res.status(500).json({ message: "Что-то пошло не так, попробуйте снова" });
  }
};
export const getAll = async (req: any, res: any) => {
  try {
    const data = await RequestModel.find();
    res.json(data);
  } catch (error) {
    console.error("Ошибка при получении данных:", error);
    res.status(500).send("Ошибка получения данных, повторите снова");
  }
};
```

Файл Request.ts представляет собой контроллер, отвечающий за обработку запросов, связанных с управлением заявками в системе.

Перенос данных в архив: Этот маршрут позволяет переносить данные в архив. Это означает, что какие-то данные считаются устаревшими или неактуальными и должны быть перемещены в архив для сохранения истории или освобождения места в основной базе данных.

Создание заявки: Этот маршрут позволяет создавать новые заявки.

Получение одной заявки по id: Этот маршрут позволяет получить информацию о заявке по её уникальному идентификатору (id). Это может быть полезно для отображения подробностей конкретной заявки на веб-странице или в приложении.

Получение всех заявок: Этот маршрут предоставляет возможность получить список всех заявок, находящихся в системе. Это может быть полезно, например, для создания страницы со списком всех заявок или для выполнения аналитики по заявкам.

```
export default interface ItemInterface {
  breakdownType: string;
  carModel: string;
  email: string;
  message: string;
  name: string;
  phone: string;
  __v: number;
  _id: string;
}
```

Файл Item.interface.ts является TypeScript-интерфейсом, который служит для описания структуры данных заявки в приложении. В контексте TypeScript, интерфейсы используются для определения формата или типа данных, который должен быть представлен в объекте или переменной.

Интерфейс Item определяет структуру каждой заявки, указывая на конкретные поля и их типы данных. Например, интерфейс может содержать поля, такие как название заявки, дата создания, статус и другие характеристики.

Используя TypeScript-интерфейсы, разработчик может обеспечить более строгую типизацию данных в приложении. Это означает, что компилятор TypeScript будет проверять соответствие данных определенным типам во время компиляции кода. Если в коде встречается ошибка,

несоответствующая типам данных, TypeScript выдаст предупреждение или ошибку компиляции.

Таким образом, использование интерфейсов помогает предотвратить ошибки и повысить надежность кода, обеспечивая более четкое определение структуры данных и их типов в приложении.

```
import mongoose from "mongoose";
const archiveSchema = new mongoose.Schema({
 name: {
  type: String,
  required: true,
 },
 email: {
  type: String,
  required: true,
 },
 phone: {
  type: String,
  required: true,
 },
 carModel: {
  type: String,
  required: true,
 },
 breakdownType: {
  type: String,
  required: true,
 },
 message: {
  type: String,
  required: true,
```

```
  },
});
export default mongoose.model("Archive", archiveSchema);
import mongoose from "mongoose";
const requestSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
  phone: {
    type: String,
    required: true,
  },
  carModel: {
    type: String,
    required: true,
  },
  breakdownType: {
    type: String,
    required: true,
  },
  message: {
    type: String,
    required: true,
  },
});
```

```
export default mongoose.model("Request", requestSchema);
```

Модели данных в MongoDB являются ключевым инструментом для определения структуры и типов данных в базе. Этот подход позволяет разработчикам создавать гибкие и масштабируемые приложения, которые могут адаптироваться к различным бизнес-потребностям.

Основным принципом MongoDB является документо-ориентированная модель данных. Данные хранятся в виде документов в формате BSON (Binary JSON), где каждый документ представляет собой набор пар ключ-значение. Основное отличие от реляционных баз данных состоит в том, что каждый документ может иметь свою уникальную структуру и набор полей.

Этот подход обеспечивает высокую гибкость при проектировании схемы данных. Разработчики могут организовывать данные так, как им удобно и наилучшим образом соответствует требованиям приложения. Они могут добавлять, изменять или удалять поля в документах без необходимости изменения всей схемы базы данных.

Кроме того, MongoDB обеспечивает высокую производительность и масштабируемость. Благодаря гибкой структуре данных и распределенной архитектуре, база данных легко масштабируется по мере увеличения объема данных и нагрузки. Это позволяет обеспечить непрерывную работу приложения даже при росте количества пользователей и объема данных.

Таким образом, использование моделей данных в MongoDB обеспечивает разработчикам широкие возможности для создания эффективных и гибких приложений, способных успешно справляться с разнообразными бизнес-задачами.

3.2 Тестирование веб-приложения

Тестирование программного обеспечения – это процесс оценки и верификации программного продукта, с целью выявления ошибок, несоответствий требованиям и улучшения качества [20] [22]. Тестирование

позволяет гарантировать, что система работает так, как это предусмотрено, и что она выполняет свои функции правильно и эффективно [15].

Существует несколько видов тестирования, которые используются на различных этапах разработки [23].

Unit Testing (Модульное тестирование):

- проверяет отдельные компоненты или модули кода на корректность работы;
- обычно выполняется разработчиками и используется для раннего обнаружения ошибок в коде.

Integration Testing (Интеграционное тестирование):

- проверяет взаимодействие между различными модулями и компонентами системы;
- позволяет убедиться, что модули работают корректно вместе.

System Testing (Системное тестирование):

- проверяет всю систему целиком, чтобы удостовериться в правильности выполнения всех функциональных требований;
- включает тестирование функциональности, производительности и безопасности системы.

Acceptance Testing (Приемочное тестирование):

- проводится пользователями или заказчиками для проверки соответствия системы их требованиям и ожиданиям;
- является последним этапом тестирования перед релизом продукта.

Performance Testing (Тестирование производительности):

- оценивает производительность системы под различными нагрузками;
- включает тестирование скорости, масштабируемости и устойчивости системы.

Security Testing (Тестирование безопасности):

- проверяет систему на наличие уязвимостей, защищенность данных и способность противостоять атакам;

– практическое тестирование нашего веб-приложения.

Для тестирования нашего веб-приложения, предназначенного для автосервиса, мы использовали комбинацию различных видов тестирования и инструментов.

Модульное тестирование было первым шагом в процессе тестирования. Использованны фреймворки Jest и Mocha для написания и запуска модульных тестов.

Jest:

– простота в настройке и использование;
– поддержка тестирования как синхронного, так и асинхронного кода;
– функциональность мокирования данных и проверки их корректности.

Mocha:

– гибкость и расширяемость;
– поддержка различных библиотек утверждений (assertion libraries), таких как Chai;
– возможность создания сложных сценариев тестирования.

Мы тестировали основные модули нашего приложения, такие как регистрация пользователей, обработка форм заявок и авторизация. Это помогло нам выявить и исправить ошибки на ранних этапах разработки [24].

Для интеграционного тестирования мы использовали библиотеку Supertest совместно с Jest. Supertest позволила нам проверять взаимодействие между различными компонентами приложения. Supertest:

– легкость в настройке и использование для http-запросов;
– поддержка тестирования маршрутов и middleware express;
– возможность интеграции с фреймворками для тестирования, такими как jest и mocha.

Проверена корректность взаимодействия между фронтендом и бэкендом, а также взаимодействие с базой данных MongoDB.

Системное тестирование проводилось вручную и автоматизированно. Для ручного тестирования мы использовали чек-листы, которые охватывали все основные функциональные области приложения.

Ручное тестирование:

- проверка пользовательского интерфейса на удобство и корректность работы;
- тестирование всех возможных сценариев использования приложения;
- для автоматизированного тестирования мы использовали selenium webdriver.

Selenium WebDriver:

- позволяет автоматизировать тестирование пользовательских сценариев;
- поддержка различных браузеров и платформ;
- возможность интеграции с ci/cd системами для автоматического запуска тестов.

Тестирование производительности. Для тестирования производительности мы использовали Apache JMeter. Apache JMeter:

- возможность симуляции большого количества пользователей;
- тестирование различных видов нагрузок, таких как стресс-тестирование и нагрузочное тестирование;
- анализ производительности и выявление узких мест.

Было проведено тестирование производительности с целью оценки возможности приложения обрабатывать большое количество одновременных запросов и определения максимальной нагрузки, которую система способна выдержать без ухудшения работы.

Тестирование безопасности.

Для тестирования безопасности мы использовали OWASP ZAP (Zed Attack Proxy). OWASP ZAP:

- автоматическое сканирование на уязвимости;

– возможность проведения ручного анализа безопасности;

– поддержка различных видов атак, таких как XSS и SQL-инъекции.

Было проведено сканирование нашего приложения на наличие уязвимостей и применили меры для их устранения, чтобы обеспечить высокий уровень безопасности данных пользователей.

Тестирование веб-приложения является важным этапом в процессе разработки, который позволяет выявить и исправить ошибки, повысить производительность и обеспечить безопасность системы. Используя разнообразные инструменты и методы тестирования, такие как Jest, Mocha, Supertest, Selenium WebDriver, Apache JMeter и OWASP ZAP, мы смогли создать высококачественное и надежное веб-приложение для автосервиса, которое удовлетворяет потребности пользователей и соответствует современным стандартам качества.

Выводы по главе 3

В главе 3 рассматривается разработка бэкенда для веб-приложения, который играет ключевую роль в его функционировании и безопасности.

Тестирование проводилось с использованием инструментов, таких как Selenium WebDriver для проверки работы приложения, Apache JMeter для оценки производительности и OWASP ZAP для проверки безопасности. Результатом работы стало надежное и качественное веб-приложение.

Заключение

Разработка веб-приложения для автоматизации процесса обработки заявок от клиентов в сервисных центрах является значимым шагом в направлении улучшения для эффективности работы данного сегмента бизнеса. На фоне растущего объема заявок и повышающихся требований к качеству обслуживания клиентов, внедрение современных технологических решений становится необходимостью для сервисных центров.

Целью данной выпускной квалификационной работы было создание инструмента, способного оптимизировать процессы приема, регистрации и отслеживания заявок, обеспечивая удобство и оперативность как для клиентов, так и для персонала сервисных центров.

Проведенный анализ проблем, выбор современных технологий, проектирование и реализация веб-приложения, а также его тестирование и внедрение позволили достичь этой цели.

Полученные результаты свидетельствуют о значительном улучшении процессов работы сервисных центров, обеспечивая более высокий уровень обслуживания клиентов и повышая эффективность деятельности персонала.

Разработанное веб-приложение представляет собой не просто инструмент, а ключевой компонент в стратегии современных сервисных центров, направленной на удовлетворение потребностей клиентов и укрепление конкурентных позиций на рынке.

Список используемой литературы и используемых источников

1. ГОСТ Р ИСО/МЭК 9126-93. Информационные технологии. Оценка программной продукции. Характеристики качества и руководство по их применению. Госстандарт России, 2004. - 13 с.

2. Архипенков, С. Лекции по управлению программными проектами. М: Самиздат, 2099. - 128 с.

3. Бадд, Т. Объектно-ориентированное программирование в действии: Пер. с англ. СПб: Питер, 2020. - 304 с.

4. Бланшар, К. Одноминутный менеджер и ситуационное руководство. Минск: Попурри, 2024. - 160 с.

5. Боэм, Б. У. Инженерное проектирование программного обеспечения. М: Радио и связь, 1921. - 511 с.

6. Брукс, Ф. Мифический человеко-месяц, или как создаются программные системы. СПб: Символ-Плюс, 2020. - 304 с.

7. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд. М: Бином, СПб: Невский диалект, 2019. - 720 с.

8. Вендеров, А.М. Проектирование программного обеспечения экономических информационных систем. М: Финансы и статистика, 2019. - 544 с.

9. Демарко, Т. Человеческий фактор: успешные проекты и команды. СПб: Символ-Плюс, 2022. - 288 с.

10. Жоголев, Е. А. Технологии программирования. М: Научный Мир, 2021. - 216 с.

11. Зиглер, К. Методы проектирования программных систем. М: Мир, 2019. - 328 с.

12. Иванова, Г. С. Технология программирования: учебник для вузов. М: МГТУ им. Н.Э. Баумана, 2019. - 336 с.

13. Камерон, К. Диагностика и измерение организационной культуры. СПб: Питер, 2021. - 320 с.

14. Карл, И. Вигерс. Разработка требований к программному обеспечению. БХВ-Петербург, Русская Редакция, 2020. - 735 с.

15. Кларк, Э. М. Верификация моделей программ: Model Checking. М: МЦНМО, 2020. - 416 с.

16. Ларман, К. Применение UML и шаблонов проектирования. М: Издательский дом "Вильяме", 2023. - 736 с.

17. Леоненков, А. В. Самоучитель UML. СПб: BHV,2019. - 576 с.

18. Леффингуэлл, Д. Принципы работы с требованиями к программному обеспечению/ Д. Леффингуэлл, Д. Уидриг. М.: Вильямс, 2022. - 448 с.

19. Липаев, В. В. Отладка сложных программ: Методы, средства, технология/ В. В. Липаев. М.: Энергоатомиздат, 2020.

20. Липаев, В. В. Проектирование программных средств/В.В. Липаев. М.: Высшая школа, 2019. - 304 с.

21. Липаев, В.В. Мобильность программ и данных в открытых информа-ционныхсистемах/ В.В. Липаев, К.Н. Филинов. - М.: Научная книга, 2020. - 368 с.

22. Липаев, В.В. Надежность программных средств/В.В. Липаев. - М.: "Синтег", 2019. - 238 с.

23. Майерс, Г. Искусство тестирования программ/Г. Майерс Т. Баджетт, К, Сандлер. М.: Вильямс, 2019. -272 с.

24. Макконнелл, С. Сколько стоит программный проект/С. Макконнелл. СПб.: Питер, 2023. - 304 с.: ил. ISBN 978-5-91180-090-1, 978-5-7502-0295-9

25. Матьяш, В.А. Структурный анализ при разработке программного обеспечения систем реального времени / В. А. Матьяш, А. В. Никандров, В. А. Путилов, А. Е. Федоров, В. В. Фильчаков. Апатиты: КФ ПетрГУ, 2023. - 79 с.

26. Международные стандарты, поддерживающие жизненный цикл программных средств. - М.:МП «Экономика», 2020.

27. Орлов, С. А. Принципы объектно-ориентированного и параллельного программирования/ С. А. Орлов. Рига: TSI, 2021. -327 с.

28. Орлов, С. А. Технологии разработки программного обеспечения: разработка сложных программных систем: учеб, для вузов / С. А. Орлов .- 3-є изд. - СПб. [и др.] : Питер, 2024. - 527 с.

29. Соммервилл, И. Инженерия программного обеспечения/И. Соммер-вилл. - М.: Вильямс, 2022. -680 с.