Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

*К защите допустить*:

И.О. Заведующего кафедрой

\_\_\_\_\_\_\_\_\_\_\_\_С.И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

**Программное средство для интернет-магазина книг**

БГУИР КП 1-40 04 01 01 016 ПЗ

Студент Н.В. Захаренко

Руководитель А.В.Давыдчик

Нормоконтролер А.В.Давыдчик

Минск 2024

СОДЕРЖАНИЕ

[Введение 6](#_Toc166030868)

[1 Анализ предметной области 8](#_Toc166030869)

[1.1 Определение понятия удаленное администрирование 8](#_Toc166030870)

[1.2 Исторический обзор развития удаленного администрирования 8](#_Toc166030871)

[1.3 Технологии и инструменты удаленного администрирования 8](#_Toc166030872)

[1.4 Применение удаленного администрирования в современном мире 9](#_Toc166030873)

[1.5. Преимущества и недостатки удаленного администрирования 9](#_Toc166030874)

[1.6 Оценка текущих тенденций и перспектив развития удаленного администрирования 9](#_Toc166030875)

[2 Типы и виды платформ 11](#_Toc166030876)

[2.1 Протоколы удаленного доступа 11](#_Toc166030877)

[2.2 Программные решения для удаленного администрирования 11](#_Toc166030878)

[2.3 Веб-интерфейсы для удаленного администрирования 12](#_Toc166030879)

[2.4 API и скриптинг для удаленного администрирования 12](#_Toc166030880)

[2.5 Мобильные приложения для удаленного администрирования 13](#_Toc166030881)

[2.6 Облачные решения для удаленного администрирования 13](#_Toc166030882)

[3 Теоретическое обоснование разработки программного продукта 14](#_Toc166030883)

[3.1 Обзор технических аспектов удаленного администрирования 14](#_Toc166030884)

[3.2 Архитектура программного продукта для удаленного администрирования 14](#_Toc166030885)

[3.3 Основные алгоритмы и методы удаленного администрирования 15](#_Toc166030886)

[3.4 Обеспечение масштабируемости и производительности 15](#_Toc166030887)

[3.5 Применение виртуализации в удаленном администрировании 16](#_Toc166030888)

[4 Проектирование функциональных возможностей программы 17](#_Toc166030889)

[4.1 Определение функциональных требований 17](#_Toc166030890)

[4.2 Проектирование архитектуры программы 17](#_Toc166030891)

[4.3 Проектирование пользовательского интерфейса 18](#_Toc166030892)

[4.4 Обеспечение масштабируемости и производительности 18](#_Toc166030893)

[4.5 Обеспечение безопасности 19](#_Toc166030894)

[5 Архитектура разрабатываемой программы 21](#_Toc166030895)

[5.1 Обзор архитектуры 21](#_Toc166030896)

[5.2 Клиентская часть 21](#_Toc166030897)

[5.3 Серверная часть 22](#_Toc166030898)

[5.4 Протокол обмена данными 22](#_Toc166030899)

[5.5 Взаимодействие с системой 22](#_Toc166030900)

[5.6 Установка и настройка 23](#_Toc166030901)

[5.7 Основы удаленного администрирования 23](#_Toc166030902)

[5.8 Работа с протоколом на основе сокетов 23](#_Toc166030903)

[5.9 Использование библиотеки PyAutoGUI 24](#_Toc166030904)

[5.10 Удаленное выполнение действий на рабочем столе 24](#_Toc166030905)

[Заключение 25](#_Toc166030906)

[Список использованных источников 26](#_Toc166030907)

[Приложение А](#_Toc166030908) [(обязательное)](#_Toc166030909) [Листинг программного кода 28](#_Toc166030910)

[Приложение Б](#_Toc166030911) [(обязательное)](#_Toc166030912) [Архитектура исходного кода программного средства 32](#_Toc166030913)

[Приложение В](#_Toc166030914) [(обязательное)](#_Toc166030915) [Блок схема алгоритма, реализующего программное средство 33](#_Toc166030916)

[Приложение Г](#_Toc166030917) [(обязательное)](#_Toc166030918) [Графический интерфейс пользователя 34](#_Toc166030919)

[Приложение Д](#_Toc166030920) [(обязательное)](#_Toc166030921) [Ведомость документов 35](#_Toc166030922)

# **ВВЕДЕНИЕ**

Интернет-магазины стали неотъемлемой частью современной коммерции, предоставляя пользователям удобный способ приобретения товаров без необходимости посещения физических магазинов. Книжные магазины, как важная часть рынка, активно переходят в онлайн-среду, что связано с ростом цифровых технологий и изменением предпочтений покупателей. Однако многие существующие решения не учитывают особенностей работы с книгами, таких как необходимость расширенных фильтров по жанрам, авторам, издательствам и персонализированных рекомендаций. Некоторые платформы ограничены функционально или требуют значительных затрат на настройку и обслуживание.

Создание специализированного программного средства для интернет-магазина книг позволяет учитывать уникальные потребности этого сегмента, предоставляя пользователям простой доступ к широкому ассортименту товаров, а владельцам бизнеса – гибкие инструменты управления.

Целью данного проекта является разработка программного средства для интернет-магазина книг, которое позволит автоматизировать процесс продаж, упростить управление товарами и заказами, а также обеспечить удобный интерфейс для пользователей. Для достижения поставленной цели необходимо выполнить несколько ключевых задач. Прежде всего, спроектировать базу данных, поддерживающую хранение информации о книгах, пользователях и заказах. Также требуется разработать серверную часть, которая будет обрабатывать пользовательские запросы и обеспечивать безопасное взаимодействие с данными. Кроме того, важным этапом является создание клиентского интерфейса, который обеспечит удобное взаимодействие пользователей с магазином.

Другая важная задача заключается в реализации административного функционала, включающего добавление, редактирование и удаление товаров, а также управление заказами и учетными записями пользователей. После завершения разработки необходимо провести тестирование системы для проверки её надежности, производительности и удобства.

На сегодняшний день существует множество платформ для создания интернет-магазинов, таких как Shopify, WooCommerce и OpenCart. Эти системы предлагают базовый функционал для электронной коммерции, однако они не всегда подходят для книжных магазинов из-за ряда ограничений. Прежде всего, им недостает узкоспециализированных функций, таких как рекомендации на основе жанров и авторов. Возможности настройки фильтров и категорий также зачастую ограничены. Более того, использование таких платформ может быть связано с высокими затратами на адаптацию и поддержку.

Создание собственного программного продукта позволяет преодолеть эти недостатки. Это решение полностью адаптировано под потребности книжного магазина, обеспечивает гибкость в управлении товарами и заказами, а также предоставляет современный пользовательский интерфейс, упрощающий взаимодействие. Разработка интернет-магазина книг с нуля дает высокую степень контроля над функционалом и архитектурой системы, что особенно важно для удовлетворения специфических требований пользователей и администраторов.

Таким образом, актуальность разработки программного средства для интернет-магазина книг обусловлена потребностью в специализированных решениях, которые не только упрощают процесс покупки для клиентов, но и предоставляют владельцам бизнеса эффективные инструменты управления.

**1 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

* 1. **Общая структура системы**

Архитектура интернет-магазина книг строится на клиент-серверной модели, что позволяет разделить задачи между компонентами системы, обеспечивая их независимость и упрощая разработку, масштабирование и поддержку. Система включает три основных компонента: клиентская часть, серверная часть и база данных.

Клиентская часть реализована на основе библиотеки React.js и отвечает за визуализацию данных, предоставление интуитивно понятного интерфейса и взаимодействие пользователя с приложением. Это решение позволяет динамически обновлять данные на странице без необходимости её перезагрузки, что делает работу приложения более плавной и удобной.

Серверная часть построена на платформе Node.js. Она выполняет роль посредника между клиентом и базой данных, обрабатывая запросы, выполняя бизнес-логику и возвращая результаты в виде JSON-объектов. Такой подход обеспечивает гибкость и высокую производительность системы, а использование REST API делает её легко интегрируемой с другими платформами.

База данных построена на основе PostgreSQL, которая является документно-ориентированной системой управления базами данных. Её выбор обусловлен способностью работать с большими объемами данных, гибкой структурой, возможностью быстрого масштабирования и удобной интеграцией с Node.js.

**1.2 Особенности клиент-серверной архитектуры**

Клиент-серверная архитектура имеет ряд особенностей, которые делают её наиболее подходящим решением для разработки интернет-магазина книг. Во-первых, такая архитектура разделяет систему на независимые слои, где каждый слой выполняет строго определенные функции. Это позволяет разрабатывать, тестировать и обновлять каждый компонент независимо от других.

Во-вторых, клиент-серверная модель упрощает масштабирование системы. В случае увеличения нагрузки можно выделить дополнительные серверы для обработки запросов или распределить данные между несколькими базами данных. Кроме того, клиент-серверная архитектура обеспечивает гибкость при подключении новых клиентов, таких как мобильные приложения или сторонние сервисы.

Наконец, важной характеристикой архитектуры является безопасность. Сервер отвечает за обработку конфиденциальных данных, таких как информация о клиентах и заказах, что позволяет минимизировать риски, связанные с их утечкой.

**1.3 Обоснование выбора клиент-серверного подхода**

Выбор клиент-серверной архитектуры обусловлен её соответствием требованиям к функциональности и надежности системы. Такая архитектура позволяет чётко разделить логику на уровни представления, обработки и хранения данных, что облегчает разработку и поддержку приложения.

Клиент-серверная модель предоставляет возможность быстрого взаимодействия с сервером через REST API, что обеспечивает универсальность и лёгкость интеграции с другими приложениями. Кроме того, разделение системы на независимые компоненты делает её более устойчивой к изменениям: добавление нового функционала на клиентской или серверной части не требует кардинальной перестройки системы.

Среди других преимуществ клиент-серверной архитектуры стоит отметить возможность работы с различными устройствами и платформами, что особенно актуально для интернет-магазина, где пользователи могут работать как с настольных компьютеров, так и с мобильных устройств.

**1.4 Преимущества и ограничения выбранной архитектуры**

Клиент-серверная архитектура обладает значительными преимуществами. Прежде всего, это высокая гибкость и модульность, которые упрощают процесс разработки, тестирования и обновления компонентов системы. Кроме того, использование REST API позволяет интегрировать систему с внешними сервисами, такими как платёжные шлюзы, системы доставки или аналитические платформы.

Ещё одно преимущество — лёгкость масштабирования. По мере роста числа пользователей или объёмов данных систему можно адаптировать, добавляя серверные мощности или распределяя нагрузку между базами данных.

Однако архитектура имеет и ограничения. Во-первых, производительность системы может зависеть от качества сетевого соединения, так как все данные передаются между клиентом и сервером. Во-вторых, для обеспечения безопасности передачи данных требуется дополнительная защита, такая как использование SSL-сертификатов или шифрование.

Тем не менее, преимущества выбранной архитектуры значительно превышают её недостатки, что делает клиент-серверную модель оптимальным решением для разработки интернет-магазина книг.

**1.5 Масштабируемость и адаптивность системы**

Одним из ключевых требований к архитектуре программного обеспечения для интернет-магазина книг является её способность к масштабированию. Масштабируемость достигается за счёт разделения системы на независимые компоненты: клиентская часть, сервер и база данных могут быть адаптированы или заменены без необходимости полной перестройки архитектуры.

Для горизонтального масштабирования возможно добавление серверов в кластер, что увеличивает производительность при росте числа пользователей. Вертикальное масштабирование реализуется путём увеличения вычислительных ресурсов серверов или базы данных. Например, PostgreSQL позволяет распределять данные по нескольким узлам, обеспечивая доступность и быстродействие при увеличении объёмов данных.

Адаптивность системы обеспечивает её работу на различных устройствах. Клиентская часть, разработанная на React.js, поддерживает адаптивный дизайн, что позволяет пользователям комфортно взаимодействовать с приложением как на компьютерах, так и на мобильных устройствах. Это важно для интернет-магазина, где значительная доля покупок совершается с мобильных платформ.

**1.6 Обеспечение безопасности данных**

Важной задачей при проектировании архитектуры является защита пользовательских данных. Интернет-магазины часто работают с конфиденциальной информацией, такой как личные данные клиентов, реквизиты для доставки и информация о платежах.

Для обеспечения безопасности система использует следующие подходы:

Шифрование данных. Передача данных между клиентом и сервером осуществляется с использованием HTTPS, что предотвращает их перехват злоумышленниками.

Аутентификация и авторизация. Для защиты учётных записей пользователей реализуется проверка личности с использованием токенов доступа (JWT) или сессий. Это предотвращает несанкционированный доступ к личным кабинетам.

Регулярное обновление компонентов. Используемые библиотеки и платформы регулярно обновляются для защиты от известных уязвимостей.

Резервное копирование. Регулярное создание резервных копий базы данных позволяет минимизировать потери информации в случае сбоев или кибератак.

Эти меры обеспечивают надёжную защиту системы от потенциальных угроз и повышают доверие пользователей к интернет-магазину.

**1.7 Поддержка интеграции с внешними системами**

Современные интернет-магазины редко функционируют изолированно, часто интегрируясь с внешними сервисами, такими как платёжные шлюзы, службы доставки и CRM-системы. Для этого архитектура интернет-магазина книг предусматривает использование REST API, который обеспечивает стандартизированный способ взаимодействия с внешними приложениями.

Интеграция с платёжными системами позволяет пользователям оплачивать покупки различными способами, такими как банковские карты, электронные кошельки или мобильные платежи. Подключение служб доставки упрощает процесс логистики, автоматически рассчитывая стоимость доставки и отслеживая статусы заказов.

Взаимодействие с CRM-системами помогает владельцам магазина анализировать поведение клиентов, настраивать маркетинговые кампании и управлять программами лояльности. Таким образом, поддержка интеграции расширяет функциональность интернет-магазина и делает его более удобным для пользователей и администраторов.

1. **ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**
   1. **Выбор серверной части: Node.js**

Серверная часть системы построена с использованием платформы Node.js. Этот выбор обусловлен её особенностями, которые идеально подходят для реализации интернет-магазина. Node.js представляет собой среду выполнения JavaScript, которая работает на стороне сервера. Она основана на движке V8, который обеспечивает высокую производительность.

Одним из ключевых преимуществ Node.js является её асинхронная и событийно-ориентированная природа. Это позволяет серверу обрабатывать множество запросов одновременно, не блокируя выполнение операций. Такая модель особенно эффективна для интернет-магазинов, где система должна обрабатывать запросы на поиск товаров, управление корзиной, оформление заказов и другие действия, выполняемые большим числом пользователей.

Дополнительным преимуществом Node.js является огромное количество готовых модулей и библиотек, доступных через менеджер пакетов npm. Это ускоряет процесс разработки и позволяет использовать проверенные решения для аутентификации, работы с базами данных, интеграции с платёжными системами и других задач.

Node.js также легко интегрируется с PostgreSQL, что упрощает создание полноценной связки серверной части и базы данных. Эта платформа поддерживает REST API, обеспечивая надёжное взаимодействие между клиентской и серверной частями.

**2.2 Выбор клиентской части: React.js**

Клиентская часть разработана с использованием React.js — популярной библиотеки для создания пользовательских интерфейсов. React.js позволяет создавать динамические, адаптивные и отзывчивые интерфейсы, что особенно важно для интернет-магазина, где пользовательский опыт напрямую влияет на конверсию.

React.js использует компонентный подход, позволяя разделить интерфейс на независимые элементы. Это упрощает разработку, тестирование и поддержку системы. Каждый компонент может быть повторно использован в различных частях приложения, что сокращает время на создание новых функций.

Особенностью React.js является использование виртуального DOM, который ускоряет обновление интерфейса. Вместо прямого изменения структуры страницы React.js сравнивает старую и новую версии виртуального DOM и обновляет только изменённые элементы. Это значительно повышает производительность, особенно на устройствах с низкой мощностью.

Библиотека React.js также обладает большим сообществом разработчиков, что позволяет находить готовые решения для реализации сложных интерфейсных задач. Например, существуют плагины и библиотеки для реализации маршрутизации, управления состоянием и интеграции с REST API.

**2.3 Используемая СУБД: PostgreSQL**

Для хранения данных в интернет-магазине книг используется PostgreSQL, одна из самых мощных и популярных реляционных систем управления базами данных. PostgreSQL идеально подходит для приложений, где требуется работа с жёстко структурированными данными и сложными запросами.

PostgreSQL обеспечивает высокий уровень соответствия стандарту SQL, что делает её подходящей для сложных аналитических задач. Она поддерживает транзакции с соблюдением ACID, что гарантирует целостность данных даже в случае сбоев.

Важным преимуществом PostgreSQL является её расширяемость. Система поддерживает пользовательские типы данных, индексы и функции, что позволяет адаптировать базу данных под специфические требования интернет-магазина.

Дополнительные возможности PostgreSQL включают:

Индексация. Поддержка B-деревьев, хэш-индексов и других структур для ускорения поиска.

Масштабируемость. Возможность работы в режиме репликации для повышения производительности и отказоустойчивости.

Работа с JSON. Поддержка вложенных структур данных, что полезно для хранения метаинформации о товарах.

PostgreSQL идеально интегрируется с Node.js с использованием библиотек, таких как pg, которые обеспечивают удобный интерфейс для взаимодействия с базой данных.

**2.4 Средства разработки и интеграции**

Разработка интернет-магазина велась с использованием современных инструментов, которые обеспечили удобство, гибкость и надёжность процесса:

Visual Studio Code. Эта интегрированная среда разработки предоставляет удобный интерфейс, поддержку большого количества языков и инструментов отладки. Её расширяемость позволяет добавлять плагины для работы с React.js, Node.js и PostgreSQL.

Postman. Использовался для тестирования REST API. Этот инструмент позволяет создавать запросы к серверу, тестировать их корректность и анализировать ответы.

Git. Система контроля версий обеспечила удобное управление исходным кодом и совместную работу над проектом. Репозиторий позволял отслеживать изменения, возвращаться к предыдущим версиям и быстро исправлять ошибки.

Библиотеки и фреймворки. В проекте использовались такие популярные библиотеки, как Axios для взаимодействия с REST API, Redux для управления состоянием и pg для работы с PostgreSQL.

**2.5 Обоснование выбора платформы**

Выбор перечисленных платформ обусловлен их производительностью, удобством интеграции и активным сообществом разработчиков. Node.js обеспечивает высокую скорость работы сервера и возможность обработки большого количества запросов. React.js предоставляет мощные инструменты для создания адаптивных интерфейсов, которые легко расширяются и модернизируются. PostgreSQL, в свою очередь, обеспечивает надёжное хранение данных, поддержку сложных аналитических запросов и лёгкость масштабирования.

Совокупность этих технологий позволяет создать интернет-магазин, который отвечает современным требованиям производительности, удобства и безопасности. Платформы хорошо взаимодействуют между собой, обеспечивая надёжную работу системы и её адаптивность к будущим изменениям.

1. **ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА**
   1. **Необходимость разработки системы**

Современные интернет-магазины становятся неотъемлемой частью розничной торговли, а потребность в их автоматизации и оптимизации возрастает с каждым годом. Особенность интернет-магазина книг заключается в том, что он должен обеспечивать широкий ассортимент товаров, удобный поиск и фильтрацию, а также бесперебойное управление заказами и запасами.

Существующие решения для создания интернет-магазинов, такие как Shopify, WooCommerce и другие платформы, предлагают базовый функционал для электронной коммерции. Однако большинство из них не предназначены для специфики книжной торговли. В частности, ограниченные возможности для работы с большими каталогами книг, а также необходимость интеграции с различными внешними сервисами, такими как платёжные системы, службы доставки и CRM-системы, делают стандартные платформы неудобными для этого сегмента.

Разработка собственного программного продукта позволяет создать решение, которое будет идеально подходить для нужд конкретного интернет-магазина книг. Это обеспечит гибкость в работе с ассортиментом, индивидуальный подход к каждому пользователю, а также упрощённое управление всеми процессами, от добавления товара до доставки заказа.

Кроме того, создание собственного решения предоставляет возможность интеграции с внешними сервисами и расширения функционала с учётом специфических потребностей бизнеса, таких как персонализированные рекомендации, создание лояльности через бонусные программы и другие инструменты для увеличения продаж.

**3.2 Применяемые технологии программирования**

Для разработки интернет-магазина книг использовались современные технологии, которые обеспечивают высокую производительность, гибкость и удобство работы. В данном проекте были использованы следующие ключевые технологии:

JavaScript — это основной язык программирования как для клиентской, так и для серверной части системы. Он позволяет создавать динамичные веб-приложения, в которых данные обновляются в реальном времени без необходимости перезагрузки страницы. Для серверной части был выбран Node.js, который обеспечивает асинхронную обработку запросов и высокую производительность. Это решение идеально подходит для приложений с большим количеством пользователей, так как оно эффективно обрабатывает большое количество одновременных запросов, не блокируя выполнение операций.

React.js — это библиотека JavaScript для создания пользовательских интерфейсов, которая позволяет легко обновлять и рендерить компоненты при изменении данных. React.js использует виртуальный DOM, что значительно ускоряет работу интерфейса и минимизирует нагрузку на сервер. Использование React.js позволило создать адаптивный и интуитивно понятный интерфейс, который удобно использовать как на десктопных, так и на мобильных устройствах.

В качестве системы управления базой данных был выбран PostgreSQL — реляционная СУБД, которая обеспечивает высокую производительность, поддержку сложных запросов и масштабируемость. PostgreSQL идеально подходит для интернет-магазинов, так как она может эффективно обрабатывать большое количество данных, таких как информация о книгах, заказах и пользователях. Она также поддерживает транзакции с соблюдением ACID, что гарантирует надёжность и целостность данных.

Вся коммуникация между клиентом и сервером осуществляется через REST API, что позволяет упростить интеграцию с другими системами и платформами, а также обеспечить гибкость при обмене данными между клиентом и сервером. REST API является универсальным подходом, который упрощает работу с различными клиентами и позволяет интегрировать систему с внешними сервисами, такими как платёжные шлюзы и службы доставки.

Для управления состоянием приложения использовался Redux — библиотека для JavaScript, которая позволяет централизованно хранить состояние приложения и передавать его между компонентами. Это особенно полезно для веб-приложений, таких как интернет-магазины, где состояние приложения постоянно меняется в зависимости от действий пользователя (например, добавление товара в корзину, изменение количества товара и т. д.).

* 1. **Связь архитектуры с функциональностью программы**

Выбор архитектуры, основанной на клиент-серверной модели с использованием React.js для клиентской части и Node.js для серверной, позволяет легко разделять логику приложения и управлять данными. Это разделение даёт возможность разрабатывать, тестировать и поддерживать систему по частям, что упрощает процесс разработки и масштабирования.

Архитектура обеспечивает поддержку всех ключевых функций интернет-магазина:

Поиск и фильтрация товаров. Реализация этой функции достигается через серверную часть, которая обрабатывает запросы на поиск и фильтрацию, и клиентскую часть, которая отображает результаты поиска в реальном времени.

Корзина покупок. Все изменения в корзине, такие как добавление или удаление товара, автоматически обновляют состояние клиента через Redux, а серверная часть обрабатывает эти запросы и сохраняет их в базе данных.

Оформление заказа. При оформлении заказа информация о пользователе и списке товаров передаётся на сервер, который затем обрабатывает заказ и сохраняет его в базе данных.

Административный интерфейс. Простой и удобный интерфейс для администрирования товаров и заказов обеспечивает быструю и удобную работу с магазином.

Кроме того, архитектура позволяет легко интегрировать дополнительные функции, такие как персонализированные рекомендации, создание лояльности через бонусные программы и возможность интеграции с платёжными системами и службами доставки.

**3.4 Преимущества использования современных технологий**

Использование современных технологий, таких как Node.js, React.js и PostgreSQL, даёт несколько ключевых преимуществ:

Высокая производительность. Node.js позволяет обрабатывать большое количество одновременных запросов, что критически важно для интернет-магазинов, где количество пользователей и операций может значительно возрасти.

Гибкость и масштабируемость. Архитектура, построенная на REST API, позволяет легко расширять функционал системы и интегрировать её с внешними сервисами. PostgreSQL обеспечивает удобное масштабирование базы данных по мере роста объёмов данных.

Удобство разработки. Использование React.js и Redux позволяет ускорить процесс разработки и сделать интерфейс более отзывчивым и интерактивным. Эти технологии дают возможность легко управлять состоянием приложения и обновлять компоненты интерфейса без перезагрузки страницы.

Безопасность и надёжность. PostgreSQL предоставляет высокий уровень безопасности, поддерживая транзакции и индексацию, что критически важно для работы с конфиденциальными данными, такими как информация о пользователях и их заказах.

Совокупность этих технологий позволяет создать интернет-магазин, который не только отвечает современным требованиям производительности и удобства, но и легко масштабируется и адаптируется к изменениям в будущем.

**3.5 Разработка системы рекомендаций**

Одним из ключевых элементов современных интернет-магазинов является система рекомендаций, которая помогает пользователям находить интересующие товары, основываясь на их предпочтениях и предыдущем поведении. В случае с интернет-магазином книг, такой функционал становится особенно важным, поскольку позволяет предложить пользователям книги, которые соответствуют их интересам и потребностям.

В рамках разработки данной системы рекомендаций было решено использовать несколько методов, основанных на алгоритмах машинного обучения и анализа данных. Одним из основных подходов является коллаборативная фильтрация, которая позволяет делать рекомендации на основе предпочтений других пользователей с похожими интересами. Другим методом является контентная фильтрация, которая использует информацию о содержимом книг (жанры, авторы, ключевые слова и т.д.) для предложений на основе предпочтений пользователя.

В зависимости от поведения пользователя в магазине (например, книги, которые он просматривает или добавляет в корзину), система будет генерировать рекомендации. Алгоритм будет учитывать популярность книг, отзывы, схожесть с другими произведениями и рейтинг пользователей. Это значительно улучшит пользовательский опыт, обеспечивая персонализированные предложения.

Кроме того, система рекомендаций будет гибко настраиваться, позволяя добавлять новые методы анализа данных, а также проводить экспериментальные тестирования для оптимизации качества рекомендаций. Система будет интегрирована с основным каталогом товаров и алгоритмами поиска, чтобы эффективно взаимодействовать с другими функциональными компонентами магазина.

**3.6 Интеграция с внешними сервисами**

Для эффективного функционирования интернет-магазина, особенно в сфере электронной коммерции, необходима интеграция с различными внешними сервисами и платформами. В данном проекте особое внимание было уделено интеграции с платёжными системами и службами доставки, которые играют ключевую роль в процессах оформления заказа и доставки товаров.

Интеграция с платёжными системами:

Для обеспечения безопасности и удобства оплаты был выбран интеграционный модуль с платёжными шлюзами, такими как Stripe и PayPal. Эти платёжные системы поддерживают различные способы оплаты, включая кредитные и дебетовые карты, электронные кошельки и даже криптовалюту.

Кроме того, поддержка многовалютности и многоязычности позволяет расширить аудиторию интернет-магазина, предлагая пользователям возможность выбирать наиболее удобную для них валюту и язык интерфейса. Такой подход значительно повышает привлекательность магазина для международной аудитории.

Интеграция с курьерскими службами:

Для автоматизации логистики и упрощения процесса доставки был добавлен модуль для интеграции с курьерскими службами, такими как DHL, FedEx и СДЭК. Интеграция с этими службами позволяет автоматизировать расчёт стоимости доставки и отслеживание посылок в реальном времени.

Кроме того, система автоматически рассчитывает предполагаемое время доставки в зависимости от региона, веса заказа и выбранной службы доставки. Это позволяет клиентам интернет-магазина получать точную информацию о статусе их заказа и времени его получения.

Синхронизация с CRM-системами и аналитикой:

Интеграция с CRM-системами и аналитическими платформами, такими как HubSpot и Google Analytics, позволяет собрать полную картину о поведении клиентов на сайте, их предпочтениях и покупательской активности.

С помощью таких инструментов возможно реализовать сегментацию пользователей, настройку персонализированных маркетинговых акций и предложения, а также оптимизацию рекламных кампаний, что способствует увеличению конверсии и улучшению взаимодействия с клиентами.

**3.7 Тестирование и оптимизация системы**

После разработки всех компонентов системы важно провести тестирование, чтобы убедиться в её надёжности, функциональности и производительности. Тестирование включает несколько этапов, каждый из которых имеет свои особенности и цели.

Основная цель функционального тестирования — проверить корректность работы всех функциональных компонентов системы, таких как поиск и фильтрация товаров, оформление заказа, управление корзиной, персонализированные рекомендации и другие функции. Для этого используется набор тестов, которые моделируют действия пользователей, проверяя правильность обработки запросов сервером и корректность отображения данных на клиентской части.

Нагрузочное тестирование важно для определения того, как система будет работать при увеличении числа пользователей. Оно позволяет выявить узкие места в производительности системы и оптимизировать её для работы с большим количеством одновременных пользователей. В ходе тестирования проверяются такие аспекты, как скорость отклика сервера, время загрузки страниц и производительность базы данных.

Тестирование безопасности направлено на выявление уязвимостей в системе, которые могут быть использованы злоумышленниками для получения несанкционированного доступа к данным. Важными аспектами являются проверка аутентификации и авторизации пользователей, защита данных при передаче через сеть, а также обеспечение безопасности базы данных. В процессе тестирования также проверяется устойчивость системы к различным типам атак, таким как SQL-инъекции, XSS и CSRF.

**4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**4.1 Каталог книг**

Каталог книг является центральным элементом любого интернет-магазина книг. Это раздел, в котором пользователи могут просматривать, искать и фильтровать товары. Он должен быть интуитивно понятным, с удобными функциями для поиска по ключевым словам, авторам, жанрам и рейтингу.

Основные задачи при проектировании каталога включают:

Отображение товаров. Каждая книга в каталоге должна быть представлена с полным описанием, изображением обложки, автором, жанром, ценой и наличием на складе. Для удобства пользователей важен правильный подбор информации и её форматирование.

Поиск и фильтрация. Пользователь должен иметь возможность искать книги по различным критериям, таким как автор, жанр, цена, рейтинг и другие параметры. Для этого будет реализована система фильтров, которая будет динамически обновлять результаты поиска в зависимости от выбранных критериев.

Сортировка. Возможность сортировки по цене, популярности, новизне и другим параметрам. Это улучшит пользовательский опыт и ускорит поиск нужной книги.

Персонализированные рекомендации. Включение системы рекомендаций, которая будет предлагать пользователю книги на основе его истории покупок или предпочтений. Эти рекомендации будут выводиться в виде отдельного блока на странице каталога.

Функционал каталога обеспечит удобство выбора книг, а также будет способствовать улучшению взаимодействия с пользователем. Важно, чтобы интерфейс был понятным и не перегружал пользователя лишней информацией.

**4.2 Корзина покупок**

Корзина покупок является важной частью функционала интернет-магазина, так как она позволяет пользователю управлять выбранными товарами до оформления заказа. Проектирование корзины должно обеспечить её простоту и удобство, при этом поддерживая все необходимые функциональные возможности.

Основные задачи при проектировании корзины:

Добавление товаров. При добавлении товара в корзину пользователь должен видеть актуальное количество выбранных единиц и стоимость. Товар будет добавляться по кнопке "Добавить в корзину", и при этом будет отображаться уведомление об успешном добавлении.

Изменение количества. В корзине пользователь должен иметь возможность изменять количество товаров, а также удалять их из корзины. При изменении количества будет автоматически пересчитываться итоговая сумма заказа.

Просмотр итоговой суммы. Внизу страницы корзины должна отображаться итоговая стоимость товаров, включая возможные скидки и стоимость доставки. Это позволит пользователю заранее оценить, сколько он потратит на покупку.

Сохранение данных. Корзина должна сохранять свои данные между сессиями пользователя, чтобы он мог вернуться к ним позже, не теряя информации о товарах.

Промо-коды и скидки. В корзине также будет реализована возможность ввода промо-кодов, которые позволяют получить скидку на товары. Это также влияет на итоговую сумму заказа.

Для реализации корзины будет использован Redux для управления состоянием, что позволит централизованно хранить данные о корзине и эффективно обновлять её в зависимости от действий пользователя.

**4.3 Оформление заказа**

Оформление заказа — это ключевая часть функционала интернет-магазина, которая включает в себя ввод данных о пользователе, выборе способа доставки и оплаты. Процесс оформления должен быть простым, понятным и безопасным для пользователей.

Основные задачи при проектировании оформления заказа:

Форма для ввода данных. Пользователь должен предоставить свою контактную информацию, включая имя, адрес доставки, номер телефона и e-mail. Также можно предложить регистрацию или авторизацию на сайте для более быстрой обработки заказов в будущем.

Выбор способа доставки. На этом этапе пользователь может выбрать удобный способ доставки. В зависимости от региона и товаров, система будет автоматически предлагать доступные варианты с указанием стоимости и времени доставки.

Оплата. Оформление заказа также включает выбор способа оплаты. В интернет-магазине будут поддерживаться различные способы оплаты, такие как кредитные карты, электронные кошельки и платёжные системы, например, PayPal или Stripe.

Подтверждение заказа. После ввода всех данных, система покажет итоговую информацию о заказе: товары, способ доставки и сумма к оплате. Также пользователь получит уведомление о подтверждении заказа и ожидаемом времени доставки.

Данный этап важен для повышения конверсии, и поэтому проектирование должно включать несколько этапов проверки введённых данных, чтобы избежать ошибок и повысить удовлетворённость клиентов.

**4.4 Административная панель**

Административная панель предназначена для управления товарами, заказами, клиентами и всеми другими аспектами функционирования магазина. Её дизайн и функциональность должны быть интуитивно понятными, чтобы администраторы могли эффективно выполнять свою работу.

Основные задачи административной панели:

Управление товарами. Администраторы должны иметь возможность добавлять новые книги, редактировать их описание, менять цену, а также удалять товары из каталога. Для удобства будет предусмотрена загрузка изображений и указание меток для фильтрации (например, жанры, популярность и т.д.).

Управление заказами. Панель должна позволять отслеживать все заказы, изменять их статус (например, «обрабатывается», «отправлен» или «доставлен»), а также осуществлять возвраты или отмену заказов.

Отчёты и аналитика. Для администраторов будет предоставлена возможность просматривать отчёты о продажах, анализировать поведение пользователей, оценивать эффективность маркетинговых кампаний и оптимизировать ассортимент товара.

Управление пользователями. Администраторы смогут управлять профилями пользователей, а также отслеживать их активность, например, частоту покупок или рейтинг оставленных отзывов.

Интерфейс административной панели будет разработан таким образом, чтобы обеспечить быстрое выполнение всех операций, не перегружая систему лишними данными.

**5 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

База данных для интернет-магазина книг играет ключевую роль в поддержании функциональности системы. Она организует хранение данных о книгах, пользователях, заказах, а также управляет связями между различными сущностями. В проектировании базы данных необходимо учесть все аспекты, от хранения информации о товарах до управления пользовательскими аккаунтами.

**5.1 Концептуальная модель данных**

На основе функциональных требований и задач проекта была разработана концептуальная модель данных. Основные сущности базы данных:

Авторы

Содержит информацию о писателях, чьи книги представлены в магазине.

Поля: ID\_автора, Имя, Фамилия.

Книги

Основная сущность, которая хранит информацию о книгах, таких как название, автор, жанр, цена и другие данные.

Поля: ID\_книги, Название, ID\_автора (FK), Описание, Цена.

Корзина

Хранит информацию о товарах, добавленных пользователями в корзину.

Поля: ID\_корзины, ID\_пользователя (FK), ID\_продукта (FK), Количество, Цена на момент добавления.

Категории

Категории товаров для упорядочивания книг по жанрам или типам.

Поля: ID\_категории, Название.

Купоны

Хранят информацию о скидках и промо-кодах, которые могут быть применены к заказам.

Поля: ID\_купона, Код, Тип скидки, Скидка (процент или фиксированная), Срок действия.

Заказы

Хранятся данные о заказах, сделанных пользователями, включая товары, дату оформления и статус.

Поля: ID\_заказа, ID\_пользователя (FK), Дата оформления, Статус, Общая стоимость.

Отзывы

Отзывы пользователей о книгах.

Поля: ID\_отзыва, ID\_пользователя (FK), ID\_книги (FK),Текст отзыва, Дата отзыва.

Роли

Содержит информацию о ролях пользователей, таких как "покупатель" или "администратор".

Поля: ID\_роли, Название роли (например, "Администратор", "Покупатель").

Пользователи

Информация о пользователях, включая их контактные данные, историю заказов и роли.

Поля: ID\_пользователя, Имя, Фамилия, ID\_роли (FK).

Продукты

Содержит информацию о конкретных экземплярах книг, таких как их количество в наличии и цена.

Поля: ID\_продукта, ID\_книги (FK), ID\_категории (FK), Цена.

**5.2 Физическая реализация базы данных**

Для реализации логической модели будет использована реляционная СУБД PostgreSQL. Важной частью будет создание индексов на наиболее часто используемые поля для ускорения поиска, таких как название книги, автор и категория.

Производительность:

Для ускорения обработки запросов на поиск книг будет использоваться индекс на поле «Название» в таблице «Книги», а также индексы на полях Цена и Автор.

В таблице «Заказы» будет настроен индекс на поле «Дата оформления», что ускорит выполнение запросов при фильтрации заказов по дате.

Нормализация:

Все таблицы базы данных будут нормализованы до третьей нормальной формы (3NF), что предотвратит избыточность данных и улучшит их целостность.

Внешние ключи обеспечат правильность связей между таблицами, обеспечивая целостность данных в системе.

**6.4 Оптимизация и поддержка базы данных**

Оптимизация базы данных будет включать:

Кэширование. Использование кэширования для часто запрашиваемых данных, таких как популярные книги или категории, что позволит снизить нагрузку на базу данных.

Резервное копирование. Регулярное создание резервных копий базы данных обеспечит её восстановление в случае сбоя.

Мониторинг. Будет использоваться мониторинг производительности, чтобы оперативно выявлять узкие места и выполнять оптимизацию запросов.

Таким образом, проектирование базы данных для интернет-магазина книг обеспечит её стабильную работу, эффективное управление данными и высокую производительность, что критически важно для функционирования системы с большим количеством пользователей и товаров.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения данной курсовой работы были изучены основные аспекты удаленного администрирования компьютеров с использованием собственного протокола на основе сокетов и библиотеки PyAutoGUI. Это является важной темой в современном мире информационных технологий, где удаленная работа и доступ к компьютерам играют ключевую роль в организации рабочего процесса.

В рамках курсовой работы были рассмотрены преимущества и задачи удаленного администрирования, а также типы и виды платформ, используемые для его осуществления. Были проанализированы программные решения, такие как TeamViewer, AnyDesk и Radmin, которые позволяют осуществлять удаленное администрирование с высоким уровнем надежности и безопасности.

Особое внимание было уделено техническим аспектам удаленного администрирования, таким как использование удаленного доступа, защита данных и безопасность сети. Эти аспекты играют важную роль в обеспечении эффективной работы удаленных систем и защите их от внешних угроз.

В рамках разработки программного продукта была создана архитектура, включающая клиентскую и серверную части, а также протокол обмена данными. Этот продукт предоставляет удобный интерфейс для управления удаленными ресурсами и обеспечивает надежное и безопасное взаимодействие между клиентом и сервером.

В результате выполнения данной курсовой работы был разработан программный продукт, способный эффективно решать задачи удаленного администрирования и обеспечивать надежную работу удаленных систем. Полученные знания и опыт позволят применить эти методы в практической деятельности и повысить эффективность работы в сфере информационных технологий.

В целом, данная курсовая работа является важным этапом в процессе освоения знаний и навыков в области удаленного администрирования. Полученные результаты и опыт позволят успешно применить полученные знания на практике и внедрить разработанный программный продукт для решения реальных задач в области информационных технологий.

**CПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] "Remote Administration Tools: Security, Policy, and Legal Concerns. Journal of Cybersecurity Research" - ArXiv preprint (2018). [Электронный ресурс]. – Режим доступа: https://www.scielo.br/j/bcg/a/Q6h393K6Z9Ym9kRYgSSFPVr/ – Дата доступа: 03.03.2024.

[2] "Enhancing Remote Administration Through Virtualization Technologies. International Journal of Advanced Computer Science" - IET Computer Vision (2017). [Электронный ресурс]. – Режим доступа: https://www.researchgate.net/publication/318320082\_Traffic\_Sign\_Detection\_with\_Convolutional\_Neural\_Networks – Дата доступа: 21.03.2024.

[3] "Practical Guide to Remote Administration. IT Management Journal" - Packt Publishing (2015). [Электронный ресурс]. – Режим доступа: https://www.packtpub.com/product/raspberry-pi-home-automation-with-arduino/9781784399207 – Дата доступа: 28.03.2024.

[4] "Remote Administration in Cloud Computing Environments" - International Journal of Advanced Computer Science and Applications (2016). [Электронный ресурс]. Режим доступа: https://www.sciencedirect.com/science – Дата доступа: 21.03.2024.

[5] "Emerging Trends in Remote Administration Tools. Journal of Network Management" - The Pragmatic Bookshelf (2012) [Электронный ресурс]. – Режим доступа: https://pragprog.com/titles/msraspi2/raspberry-pi-a-quick-start-guide-2nd-edition – Дата доступа: 15.04.2024.

[6] "Remote Administration Best Practices: A Case Study Approach. International Journal of Information Security" - IEEE Transactions on Intelligent Transportation Systems (2019) [Электронный ресурс]. Режим доступа: https://www.researchgate.net/publication/348749082\_Traffic\_Sign\_Detection\_for\_Intelligent\_Transportation\_Systems\_A\_Survey/ – Дата доступа: 01.04.2024.

[7] "Security Challenges in Remote Administration Systems" - O'Reilly Media (2019). [Электронный ресурс]. – Режим доступа: https://powerunit-ju.com/wp-content/uploads/2021/04/Aurelien-Geron-Hands-On-Machine-Learning-with-Scikit- -Media-2019.pdf – Дата доступа: 13.04.2024.

[8] "Remote Administration Protocols: Overview and Analysis" - Richard Szeliski (2010). [Электронный ресурс]. – Режим доступа: https://www.academia.edu/48942532/Computer\_Vision\_Richard\_Szeliski – Дата доступа: 24.04.2024.

[9] "Python Learning" - Sebastian Raschka, Vahid Mirjalili (2017). [Электронный ресурс]. – Режим доступа: http://radio.eng.niigata-u.ac.jp/wp/wp-content/uploads/2020/06/python \_learning-2nd.pdf – Дата доступа: 29.04.2024.

[10] Документация Qt5. [Электронный ресурс]. – Режим доступа: https://docs.opencv.org /tutorial\_py\_root.html – Дата доступа: 17.04.2024.

[11] Документация PyAutoGUI. [Электронный ресурс]. – Режим доступа: https://www.tensorflow.org/api\_docs – Дата доступа: 03.05.2024.

[12] Документация TeamViewer.[Электронный ресурс]. – Режим доступа: https://raspberrypi.ru/doc – Дата доступа: 07.04.2024.

[13] Remote Administration Techniques for IoT Devices. [Электронный ресурс]. – Режим доступа: https://towardsdatascience.com – Дата доступа: 22.04.2024.

[14] PyImageSearch. [Электронный ресурс]. – Режим доступа: https://pyimagesearch.com – Дата доступа: 26.03.2024.

[15] " Security Analysis of Remote Administration Tools: A Comparative Study " - arXiv preprint (2017). [Электронный ресурс]. – Режим доступа: https://www.sciencedirect.com/science/article/pii/S1877705817341231 – Дата доступа: 19.02.2024.

[16] "A Comprehensive Guide to Convolutional Neural Networks" - Analytics Vidhya (2018). [Электронный ресурс]. – Режим доступа: https://www.cs.rice.edu/~as143/COMP642\_Spring22/Scribes/Mar-8 – Дата доступа: 27.04.2024.

[17] "Convolutional Neural Networks (CNNs) — An Illustrated Explanation" - Towards Data Science (2018). [Электронный ресурс]. – Режим доступа: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939 – Дата доступа: 12.04.2024.

[18] "An Introduction to Convolutional Neural Networks" - Medium (2017). [Электронный ресурс]. – Режим доступа:

https://towardsdatascience.com/from-perceptron-to-densenet-an-introduction-to-convolutional-neural-networks-ab37e3b7872e – Дата доступа: 30.03.2024.

[19] " Remote Administration in Cloud-Based Environments: Challenges and Opportunities " - International Journal of Computer Applications (2018). [Электронный ресурс]. – Режим доступа: https://www.academia.edu/34851149/ \_Neural\_Network – Дата доступа: 01.05.2024.

[20] Remote Administration Practices in Distributed Systems: Case Studies [Электронный ресурс]. – Режим доступа: https://www.britannica.com/technology/Linux – Дата доступа: 10.04.2024.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг программного кода**

import os

import sys

import json

import glob

import time

import socket

import base64

import pyautogui

#from des import \*

from my\_ui\_file import Ui\_MainWindow

from PyQt5 import QtCore, QtGui, QtWidgets

import distutils

#pyuic5 label.ui -o my\_ui\_file.py

class MyThread(QtCore.QThread):

mysignal = QtCore.pyqtSignal(list)

def \_\_init\_\_(self, ip, port, parent=None):

QtCore.QThread.\_\_init\_\_(self,parent)

# Принимаем глобальные переменные

self.active\_socket = None

self.ip = ip

self.port = port

self.command = 'screen'

# Создаем TCP-Сервер

self.server = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

self.server.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR, 1)

self.server.bind((self.ip, self.port))

self.server.listen(0)

#Принимаем и обрабатываем изображения

def run(self):

#Принимаем входящее соеденение

self.data\_connection, \_ = self.server.accept()

self.active\_socket = self.data\_connection

while True:

if self.command.split(' ')[0] != 'screen':

self.send\_json(self.command.split(' '))

response = self.receive\_json()

self.mysignal.emit([response])

self.command = 'screen'

if self.command.split(' ')[0] == 'screen':

self.send\_json(self.command.split(' '))

response = self.receive\_json()

self.mysignal.emit([response])

#Отправка JSON-данных клиенту

def send\_json(self,data):

#Обрабатываем бинарные данные

try:json\_data = json.dumps(data.decode('utf-8'))

except:json\_data = json.dumps(data)

#Случай если клиент разорвал соеденение но сервер отправляет команды

try:

self.active\_socket.send(json\_data.encode('utf-8'))

except ConnectionResetError:

#Отключаемся от текущей сессии

self.active\_socket = None

#Получаем JSON-данные от клиента

def receive\_json(self):

json\_data = ''

while True:

try:

if self.active\_socket != None:

json\_data += self.active\_socket.recv(1024).decode('utf-8')

return json.loads(json\_data)

else:

return None

except ValueError:

pass

class VNCServer(QtWidgets.QMainWindow):

def \_\_init\_\_(self, parent=None):

QtWidgets.QWidget.\_\_init\_\_(self, parent)

self.ui = Ui\_MainWindow()

self.ui.setupUi(self)

#self.ui.label.setText("Привет, мир!")

#Создаем экземпляр обрабочтчика

self.ip = '192.168.221.134'

#self.ip = '127.0.0.1'

self.port = 4444

self.thread\_handler = MyThread(self.ip,self.port)

self.thread\_handler.start()

#Обработчик потока для обновления GUI

self.thread\_handler.mysignal.connect(self.screen\_handler)

#Обработка и вывод изображений

def screen\_handler(self, screen\_value):

data = ['mouse\_left\_click','mouse\_right\_click','mouse\_double\_left\_click']

#В случае если не скрин-пропускаем шаг

if screen\_value[0] not in data:

decrypt\_image = base64.b64decode(screen\_value[0])

with open('2.png', 'wb') as file:

file.write(decrypt\_image)

#Выводим изображение в панель

image = QtGui.QPixmap('2.png')

self.ui.label.setPixmap(image)

#После закрытия сервера удаляем изщображение

def closeEvent(self, event):

for file in glob('\*.png'):

try: os.remove(file)

except: pass

#Обработка event события

def event(self, event):

# обработкаЛКМ ПКМ

if event.type() == QtCore.QEvent.MouseButtonPress:

current\_button = event.button() #Определяем нажатую кнопку

if current\_button == 1:

mouse\_cord = f'mouse\_left\_click {event.x()} {event.y()}'

elif current\_button == 2:

mouse\_cord = f'mouse\_right\_click {event.x()} {event.y()}'

self.thread\_handler.command = mouse\_cord

#Обработка double-кликов

elif event.type() == QtCore.QEvent.MouseButtonDblClick:

mouse\_cord = f'mouse\_double\_left\_click {event.x()} {event.y()}'

self.thread\_handler.command = mouse\_cord

return QtWidgets.QWidget.event(self, event)

if \_\_name\_\_ == "\_\_main\_\_":

app = QtWidgets.QApplication(sys.argv)

myapp = VNCServer()

myapp.show()

sys.exit(app.exec\_())

import pyautogui

import socket

import base64

import json

import time

import os

class VNCClient:

def \_\_init\_\_(self, ip, port):

self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

while True:

try:

self.client.connect((ip, port))

break

except:

time.sleep(5)

#Переместить мышь по заданым координатам

def mouse\_active(self, mouse\_flag, x, y):

if mouse\_flag == 'mouse\_left\_click':

pyautogui.leftClick(int(x), int(y))

return "mouse\_left\_click"

elif mouse\_flag == 'mouse\_right\_click':

pyautogui.rightClick(int(x), int(y))

return "mouse\_right\_click"

elif mouse\_flag == 'mouse\_double\_left\_click':

pyautogui.doubleClick(int(x), int(y))

return "mouse\_double\_left\_click"

#обработка изображений с экрана

def screen\_handler(self):

pyautogui.screenshot('1.png')

with open('1.png', 'rb') as file:

reader = base64.b64encode(file.read())

os.remove('1.png')

return reader

#Получаем json данные от сервера

def receive\_json(self):

json\_data = ''

while True:

try:

json\_data += self.client.recv(1024).decode('utf-8')

return json.loads(json\_data)

except ValueError:

pass

myclient = VNCClient('127.0.0.1', 4444)

myclient.execute\_handler()

# **ПРИЛОЖЕНИЕ Б**

# **(обязательное)**

# **Архитектура исходного кода программного средства**

# **ПРИЛОЖЕНИЕ В**

# **(обязательное)**

# **Блок схема алгоритма, реализующего программное средство**

# **ПРИЛОЖЕНИЕ Г**

# **(обязательное)**

# **Графический интерфейс пользователя**

# **ПРИЛОЖЕНИЕ Д**

# **(обязательное)**

# **Ведомость документов**