Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Институт информационных технологий

Факультет компьютерных технологий

Кафедра информационных систем и технологий

|  |
| --- |
| *К защите допустить*: |
| Заведующий кафедрой ИСиТ |
| \_\_\_\_\_\_\_\_\_\_\_\_А.И. Парамонов |

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту

на тему

**программное средство автоматизации**

**бизнес-процессов информационного отдела организации**

БГУИР ДП 1-40 01 01 01 040 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент |  | А.В. Готовчиц |
| Руководитель |  | А.С. Корзан |
| Консультанты: |  |  |
| *от кафедры ИСиТ* |  | А.Г. Савенко |
| *по экономической части* |  | В.Г. Горовой |
|  |  |  |
| Нормоконтролер |  | А.С. Шелягович |
|  |  |  |
| Рецензент |  |  |

Минск 2025

**Р Е Ф Е Р А Т**

ПРОГРАММНОЕ СРЕДСТВО АВТОМАТИЗАЦИИ БИЗНЕС-ПРОЦЕССОВ ИНФОРМАЦИОННОГО ОТДЕЛА ОРГАНИЗАЦИИ: дипломный проект / А.В. Готовчиц. – Минск: БГУИР, 2025, – п.з. – 89с., чертежей (плакатов) – 6 л. формата А1.

Объектом проектирования является программное средство для автоматизации бизнес-процессов информационного отдела организации.

Целью данного дипломного проекта является разработка программного средства, которое позволяет автоматизировать ключевые бизнес-процессы информационного отдела, включая планирование задач, управление видеоконференциями, отправку заявок на доступ к системам, обработку пользовательских запросов и управление доступными сервисами.

Для достижения поставленной цели был реализован комплексный подход, включающий проектирование архитектуры системы, разработку базы данных, создание пользовательского интерфейса и интеграцию с внешними сервисами. Программное средство реализовано с использованием современных технологий, таких как Python, Django, JavaScript, Bootstrap и PostgreSQL, что обеспечивает его масштабируемость, надежность и удобство в эксплуатации.

В процессе выполнения проекта были изучены современные подходы к автоматизации бизнес-процессов, проведен анализ аналогичных решений, а также сформулированы и обоснованы требования к разрабатываемой системе.

Программное средство протестировано с использованием методов белого и черного ящика. Тестирование подтвердило его соответствие заявленным требованиям по функциональности, надежности и производительности. Обнаруженные в процессе тестирования ошибки были исправлены, что повысило стабильность и удобство использования системы.

В разделе технико-экономического обоснования выполнены расчеты затрат на разработку и эксплуатацию программного средства. Результаты расчетов подтверждают экономическую целесообразность разработки и внедрения системы, что позволит значительно сократить время обработки запросов и повысить эффективность работы сотрудников информационного отдела.

Разработанное программное средство полностью готово к внедрению в эксплуатацию и обеспечивает повышение эффективности управления бизнес-процессами, снижая трудозатраты и минимизируя риски, связанные с человеческим фактором.

**СОДЕРЖАНИЕ**

[1 Анализ предметной области 7](#_Toc185362119)

[1.1 Обзор существующих аналогов 7](#_Toc185362120)

[1.2 Выбор и обоснование языков программирования, фреймворков, библиотек, СУБД для разработки программного средства 11](#_Toc185362121)

[1.3 Постановка задачи на дипломное проектирование 21](#_Toc185362122)

[2 Моделирование предметной области 24](#_Toc185362123)

[2.1 Разработка информационной модели базы данных 24](#_Toc185362124)

[2.2 Разработка диаграммы классов 25](#_Toc185362125)

[2.3 Разработка диаграммы вариантов использования 28](#_Toc185362126)

[2.4 Спецификация требований 31](#_Toc185362127)

[3 Проектирование и разработка программного средства 35](#_Toc185362128)

[3.1 Проектирование и разработка базы данных 35](#_Toc185362129)

[3.2 Разработка функциональной модели 40](#_Toc185362130)

[3.3 Разработка диаграммы развертывания 43](#_Toc185362131)

[3.4 Проектирование и разработка алгоритма программного средства 45](#_Toc185362132)

[3.5 Управление процессом разработки с помощью контроля версий 48](#_Toc185362133)

[4 Тестирование программного средства 50](#_Toc185362134)

[4.1 Выбор и обоснование видов тестирования 50](#_Toc185362135)

[4.2 Результаты тестирования 51](#_Toc185362136)

[4.3 Вывод тестирования 54](#_Toc185362137)

[5 Руководство по эксплуатации программного средства 56](#_Toc185362138)

[6 Технико-экономическое обоснование разработки программного средства автоматизации бизнес-процессов информационного отдела организации 65](#_Toc185362139)

[6.1 Характеристика программного средства 65](#_Toc185362140)

[6.2 Расчет инвестиций в разработку программного средства 65](#_Toc185362141)

[6.3 Расчет экономического эффекта от использования программного средства 68](#_Toc185362142)

[6.4 Расчет показателей экономической эффективности разработки и использования программного средства в организации 71](#_Toc185362143)

[Заключение 73](#_Toc185362144)

[Список использованных источников 74](#_Toc185362145)

[Приложение а (обязательное) исходный код программного средства 75](#_Toc185362146)

**ВВЕДЕНИЕ**

Современные организации сталкиваются с необходимостью автоматизации бизнес-процессов для повышения эффективности работы и оптимизации использования ресурсов. Информационные отделы играют ключевую роль в поддержке таких процессов, обеспечивая доступ к данным и инфраструктуре. В последние годы активно развиваются программные решения для управления задачами, планирования, мониторинга и взаимодействия сотрудников, что способствует снижению человеческого фактора и ускорению выполнения операций.

Целью данного дипломного проектирования является разработка программного средства для автоматизации бизнес-процессов информационного отдела организации. Разрабатываемое решение должно обеспечить удобное планирование задач, управление запросами пользователей, организацию видеоконференций и централизованный доступ ко всем предоставляемым услугам.

Основные принципы проектирования включают использование современных технологий разработки, таких как Python, Django, JavaScript, Bootstrap, PostgreSQL. Процесс проектирования базируется на принципах модульности, масштабируемости и эргономичности пользовательского интерфейса.

Пояснительная записка к проекту состоит из нескольких разделов, каждый из которых решает конкретные задачи. Первый раздел посвящен анализу предметной области и включает обзор аналогичных программных средств, а также обоснование выбора технологий для разработки. Второй раздел описывает моделирование предметной области, включая информационную модель базы данных и спецификацию требований к системе. В третьем разделе рассматривается проектирование системы: разработаны диаграммы базы данных, классов и развертывания, а также представлены схемы работы приложения. Четвертый раздел содержит результаты тестирования разработанного программного средства. В пятом разделе представлено руководство по эксплуатации системы. Заключительный, шестой раздел включает технико-экономическое обоснование разработки.

Данный дипломный проект выполнен мной лично, проверен на заимствования в системе «Antiplagius», оригинальность пояснительной записки составляет 91%.[1]

1. **АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

## **1.1 Описание предметно области**

Бизнес-процессы, протекающие в информационном отделе, охватывают широкий спектр задач, связанных с обеспечением стабильного функционирования корпоративной инфраструктуры, управлением сервисами и оборудованием, а также взаимодействием с сотрудниками организации. Основные процессы включают обработку заявок пользователей на доступ к корпоративным системам, координацию и планирование событий через встроенные инструменты календаря, решение технических проблем, которые могут возникнуть в процессе работы, и ведение учёта материально-технических ресурсов.

Пользователи информационного отдела регулярно взаимодействуют с отделом для выполнения своих рабочих задач. Например, они запрашивают доступ к определённым сервисам, таким как корпоративные системы документооборота, аналитики или видеоконференций. Также пользователи подают заявки на устранение технических проблем, связанных с работой их оборудования или программного обеспечения. Помимо этого, сотрудники организации через календарь могут планировать события, такие как встречи и совещания, выбирая время и место проведения.

Администраторы информационного отдела, в свою очередь, координируют работу всех этих процессов. Они обрабатывают запросы пользователей, предоставляют права доступа, управляют перечнем доступных сервисов и контролируют их состояние. Кроме того, администраторы отслеживают состояние оборудования, распределяют задачи по устранению неисправностей и фиксируют статус выполнения заявок. Учет оборудования включает такие параметры, как тип устройства, марка, модель, инвентарный номер, кабинет нахождения и ответственное лицо.

Автоматизация данных бизнес-процессов позволяет упростить их выполнение, устранить ручные ошибки, повысить прозрачность взаимодействия между пользователями и администраторами, а также сократить время, затрачиваемое на выполнение рутинных операций. Объектом автоматизации выступают все ключевые процессы информационного отдела, начиная от управления доступом и заявками и заканчивая учетом оборудования и планированием мероприятий.

Таким образом, предметной областью данной разработки является деятельность информационного отдела, включающая управление корпоративными сервисами, обработку заявок, учет материально-технических ресурсов и решение технических проблем. Создание программного средства для автоматизации этих процессов направлено на улучшение организации работы, оптимизацию использования ресурсов и повышение эффективности взаимодействия сотрудников.

## **1.2 Обоснование актуальности разработки программного средства**

Актуальность разработки программного средства для автоматизации бизнес-процессов информационного отдела обусловлена необходимостью повышения эффективности работы современных организаций. В условиях растущей цифровизации бизнес-процессов информационные отделы сталкиваются с увеличением объёмов данных, усложнением инфраструктуры и возрастанием требований к оперативности и качеству предоставляемых услуг. Это создаёт потребность в инструментах, которые обеспечивают автоматизацию рутинных задач, упрощают взаимодействие между сотрудниками и администраторами и минимизируют человеческий фактор при обработке данных.

Информационные отделы организаций выполняют ключевые функции, связанные с обеспечением доступа сотрудников к корпоративным сервисам, планированием событий, решением технических проблем и учётом материально-технических ресурсов. Ручная обработка заявок, отсутствие единого инструмента для управления процессами и дублирование данных приводят к задержкам, ошибкам и повышенной нагрузке на сотрудников отдела.

Разрабатываемое программное средство решает эти проблемы, предоставляя пользователям удобный интерфейс для подачи заявок, планирования событий и взаимодействия с информационным отделом. Для администраторов создаются инструменты управления сервисами, заявками, событиями и оборудованием, что позволяет быстрее реагировать на запросы и улучшать контроль за выполнением задач.

Дополнительно актуальность разработки обусловлена требованиями к обеспечению прозрачности и отслеживаемость процессов. Система фиксирует все действия пользователей и администраторов, включая историю заявок, изменения статусов и управление ресурсами, что позволяет вести полноценный аудит процессов и предотвращать нарушения.

С экономической точки зрения автоматизация снижает затраты на обработку данных, минимизирует риски ошибок, сокращает время выполнения задач и оптимизирует использование человеческих ресурсов. Это особенно важно для организаций, где информационные отделы обслуживают большое количество сотрудников и управляют сложной инфраструктурой.

Таким образом, разработка программного средства для автоматизации бизнес-процессов информационного отдела является актуальной задачей, которая отвечает современным требованиям к организации и управлению цифровыми процессами. Внедрение такой системы позволит повысить производительность сотрудников, улучшить качество обслуживания и обеспечить стабильное функционирование корпоративной инфраструктуры.

## **1.3 Анализ методов и технологий**

Разработка программного средства автоматизации бизнес-процессов информационного отдела базируется на современных технологиях, которые обеспечивают надёжность, производительность и удобство использования. Основой системы является клиент-серверная архитектура, разделяющая пользовательский интерфейс и серверную логику для гибкости, масштабируемости и удобства сопровождения.

Клиент-серверная архитектура обеспечивает разделение пользовательского интерфейса и серверной логики, что позволяет повысить гибкость и масштабируемость системы, а также упростить её сопровождение. Использование валидации и обработки данных гарантирует корректность информации, поступающей от пользователей, до её сохранения или дальнейшей обработки. Применение RESTful API обеспечивает стандартизированное взаимодействие между клиентом и сервером, упрощая передачу данных и интеграцию с другими системами. В системе реализовано разделение прав доступа, что позволяет разграничить функционал для пользователей и администраторов, обеспечивая безопасность и контроль. Для работы с данными используется метод объектно-реляционного отображения (ORM), что упрощает взаимодействие с базой данных, позволяя работать с объектами вместо написания сложных SQL-запросов. Шифрование паролей пользователей с применением алгоритма PBKDF2 обеспечивает надёжную защиту конфиденциальной информации.

Для серверной части используется Django, мощный веб-фреймворк на языке Python, который предоставляет встроенные инструменты для работы с ORM, аутентификацией пользователей, защитой от CSRF и XSS-атак. В качестве системы управления базами данных выбрана PostgreSQL, обеспечивающая надёжное хранение данных, поддержку сложных запросов и транзакций. Клиентская часть разрабатывается с использованием HTML, CSS и Bootstrap, которые обеспечивают создание адаптивного и удобного интерфейса. Bootstrap используется для ускорения разработки благодаря набору готовых компонентов и инструментов. Для взаимодействия клиентской и серверной частей применяется HTTP-протокол с передачей данных в формате JSON. Управление проектом и версионный контроль кода осуществляется с помощью Git, а среда разработки Visual Studio Code обеспечивает удобство написания, отладки и тестирования программного кода.

## **1.4 Анализ существующих аналогов**

Для автоматизации бизнес-процессов и управления информационными потоками часто используются системы, предназначенные для планирования, управления проектами, командного взаимодействия и документооборота. Среди них выделяются такие продукты, как Jira, Atlassian Confluence и Taiga, которые широко применяются в различных организациях для управления задачами и проектами. Рассмотрим их более подробно, выявив их преимущества и недостатки.

Jira – это система для управления проектами и задачами, разработанная компанией Atlassian. Она особенно популярна среди команд, использующих методологии Agile (например, Scrum и Kanban). Jira предоставляет множество инструментов для управления рабочими процессами, мониторинга выполнения задач и анализа производительности команд.

Преимущества работы с Jira:

1. Гибкость настройки Jira позволяет настроить рабочие процессы, статусы задач, роли участников и разрешения для различных типов проектов. Это делает её удобной для команд с различными потребностями.
2. Поддержка Agile в Jira предоставляет встроенные инструменты для работы по Scrum и Kanban, такие как доски задач, спринты и отчеты по производительности (burn-down и burn-up диаграммы).
3. Интеграция с другими инструментами Atlassian. Jira легко интегрируется с Confluence (для документации), Bitbucket (для управления репозиториями кода) и другими продуктами Atlassian, что делает её частью комплексной экосистемы для управления разработкой программного обеспечения.
4. Jira поддерживает множество плагинов и расширений, что позволяет добавлять функциональность для специфических нужд, включая отчеты, автоматизацию задач и интеграцию с другими системами.

Недостатки работы с Jira:

1. Сложность для небольших команд. Для небольших организаций или команд с ограниченными ресурсами Jira может оказаться слишком сложной и избыточной по функционалу. Множество настроек и возможностей требуют времени для освоения и внедрения.
2. Стоимость. Jira является платной системой, и при расширении команды или добавлении новых функций затраты на её использование могут значительно возрасти.
3. Потребность в обучении. Интерфейс Jira может быть сложным для новых пользователей, особенно для тех, кто не имеет опыта работы с Agile или подобными системами управления проектами.

Пример работы с Jira представлен на рисунке 1.1.

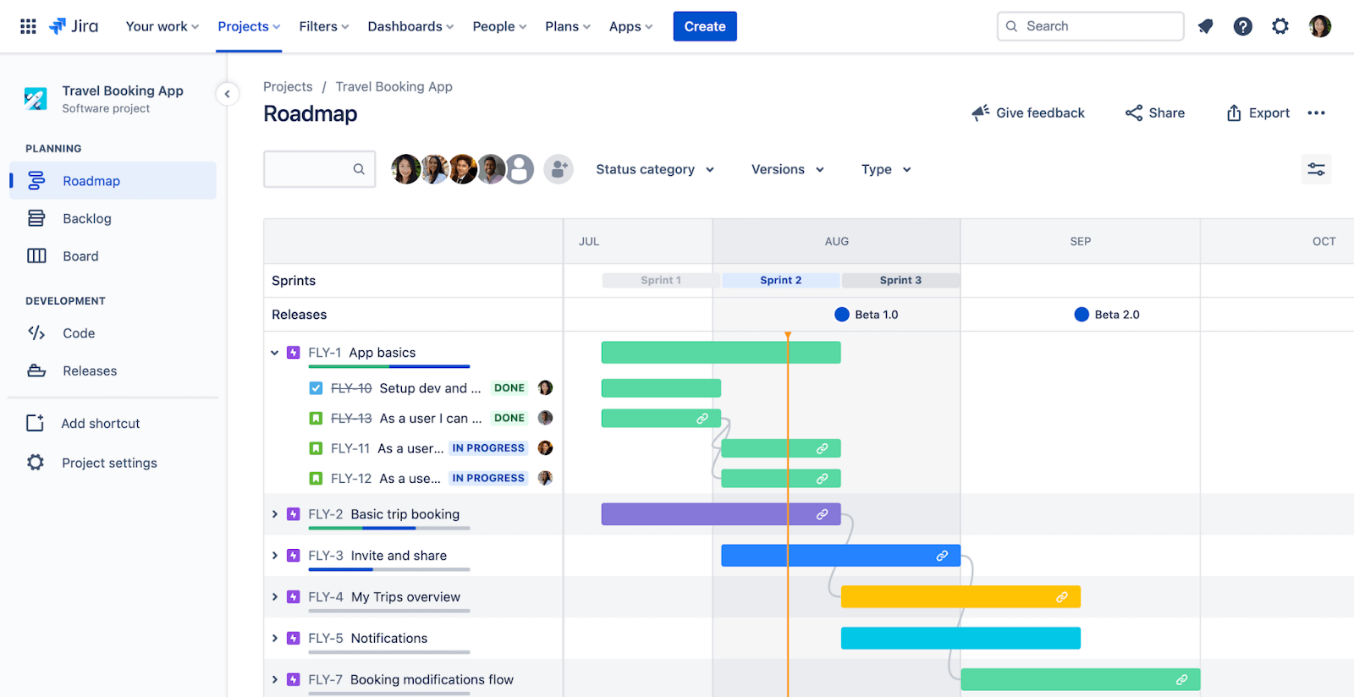


Рисунок 1.1 – Страница с проектами программного средства Jira.

Atlassian Confluence – это система для совместной работы и управления знаниями, предназначенная для создания, организации и управления документацией. Она тесно интегрирована с Jira и используется для ведения проектной документации, разработки спецификаций и совместной работы команд.

Преимущества:

1. Совместное редактирование. Confluence позволяет одновременно редактировать документы нескольким пользователям, что облегчает совместную работу над проектами.
2. Интеграция с Jira. Сильная интеграция с Jira позволяет легко связывать задачи с соответствующей документацией и отслеживать взаимосвязи между различными элементами проекта.
3. Шаблоны документации. Confluence предоставляет множество готовых шаблонов для документации, что упрощает ведение рабочих процессов и структурирование информации.
4. Контроль версий. Confluence поддерживает версионность документов, что позволяет отслеживать изменения и возвращаться к предыдущим версиям, если это необходимо.

Недостатки:

1. Сложность использования для начинающих: Confluence может быть сложным для новых пользователей, особенно для тех, кто не знаком с системами управления знаниями. Освоение всех функций и возможностей может занять время.
2. Отсутствие продвинутых возможностей для совместной работы. хотя Confluence отлично подходит для документации, он не всегда может предложить те же инструменты совместной работы, которые предлагают более специализированные системы управления проектами и коммуникацией.
3. Высокая стоимость продукта. Как и другие продукты Atlassian, Confluence является платным решением, и затраты могут увеличиваться с ростом команды или необходимостью дополнительных функций.

Пример работы с Atlassian Confluence представлен на рисунке 1.2.

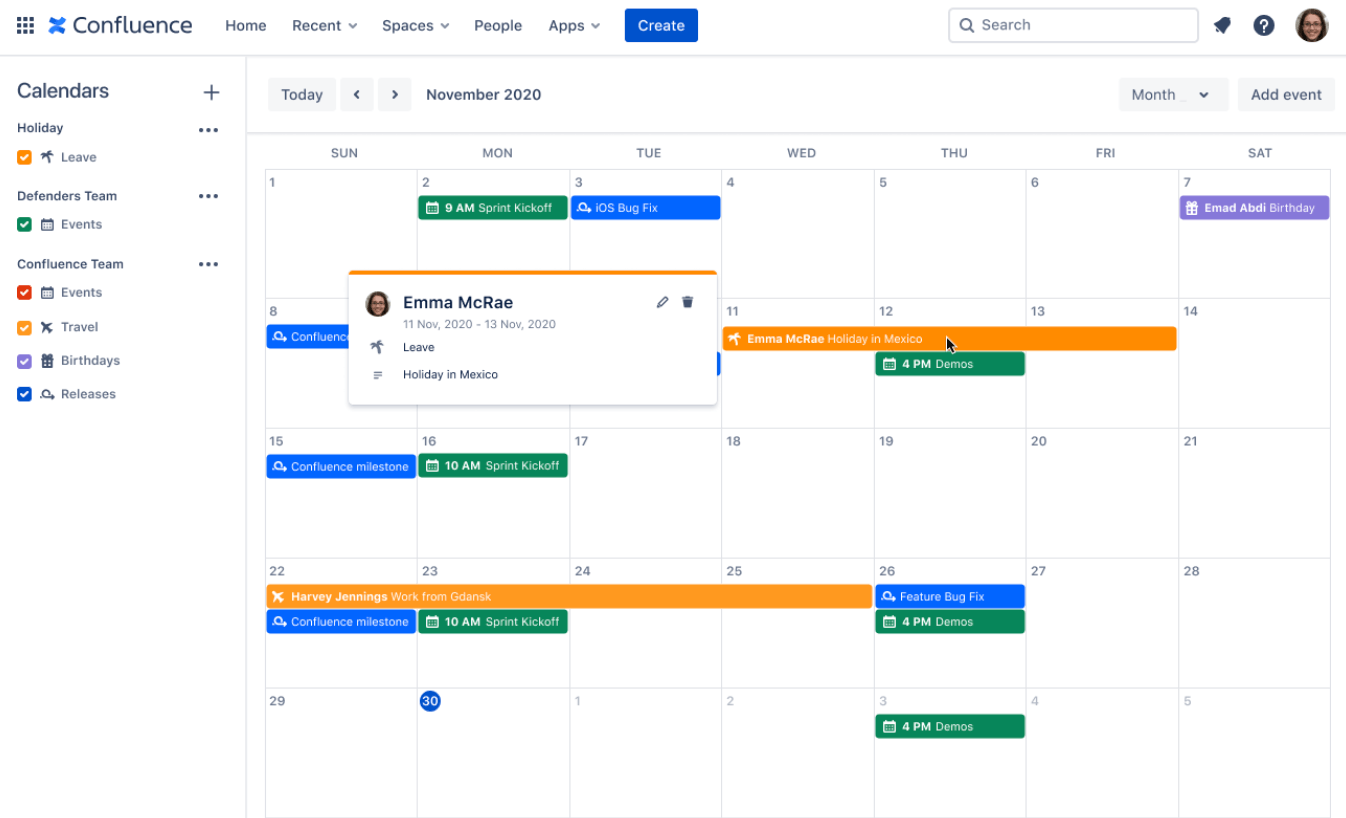


Рисунок 1.2 – Страница работы с календарем программного средства Atlassian Confluence.

Taiga – это платформа с открытым исходным кодом для управления проектами, ориентированная на команды, работающие по методологиям Agile. Она поддерживает Kanban и Scrum, а также предоставляет инструменты для отслеживания задач, планирования спринтов и ведения беклогов. Taiga – это платформа с открытым исходным кодом для управления проектами, ориентированная на команды, работающие по методологиям Agile. Она поддерживает Kanban и Scrum, а также предоставляет инструменты для отслеживания задач, планирования спринтов и ведения беклогов. В системе предусмотрена интегрированная система для управления задачами и ошибками, позволяющая настраивать типы задач, их приоритеты и критичность. Каждая задача может быть связана с определенными эпиками или пользовательскими историями.

Преимущества работы с Taiga:

1. Открытый исходный код. Taiga является бесплатной для небольших команд и доступной для развертывания на собственных серверах, что позволяет полностью контролировать данные и настройку системы.
2. Простота использования. В отличие от Jira, Taiga предлагает более простой и интуитивный интерфейс, что делает её удобной для небольших команд или стартапов, которые только начинают внедрять Agile.
3. Гибкость и кастомизация. несмотря на простоту, Taiga предлагает достаточный уровень гибкости для настройки рабочих процессов и управления задачами в зависимости от требований команды.
4. Интеграция с Git: Taiga поддерживает интеграцию с системами контроля версий, такими как GitLab и GitHub, что удобно для команд разработчиков программного обеспечения.

Недостатки:

1. Ограниченный функционал. В сравнении с Jira, Taiga предлагает меньше возможностей для кастомизации и расширения. Она больше ориентирована на небольшие команды и может не удовлетворить потребности крупных организаций с комплексными процессами.
2. Ограниченные отчеты и аналитика. Taiga предоставляет ограниченные инструменты для анализа производительности команды и генерации отчетов, что может быть недостатком для менеджеров, ориентированных на метрики.
3. Отсутствие комплексной поддержки. поскольку Taiga является open-source решением, поддержка может быть ограничена, и пользователи должны полагаться на сообщество или свои собственные ресурсы для решения технических проблем.

Пример работы с Taiga представлен на рисунке 1.3.

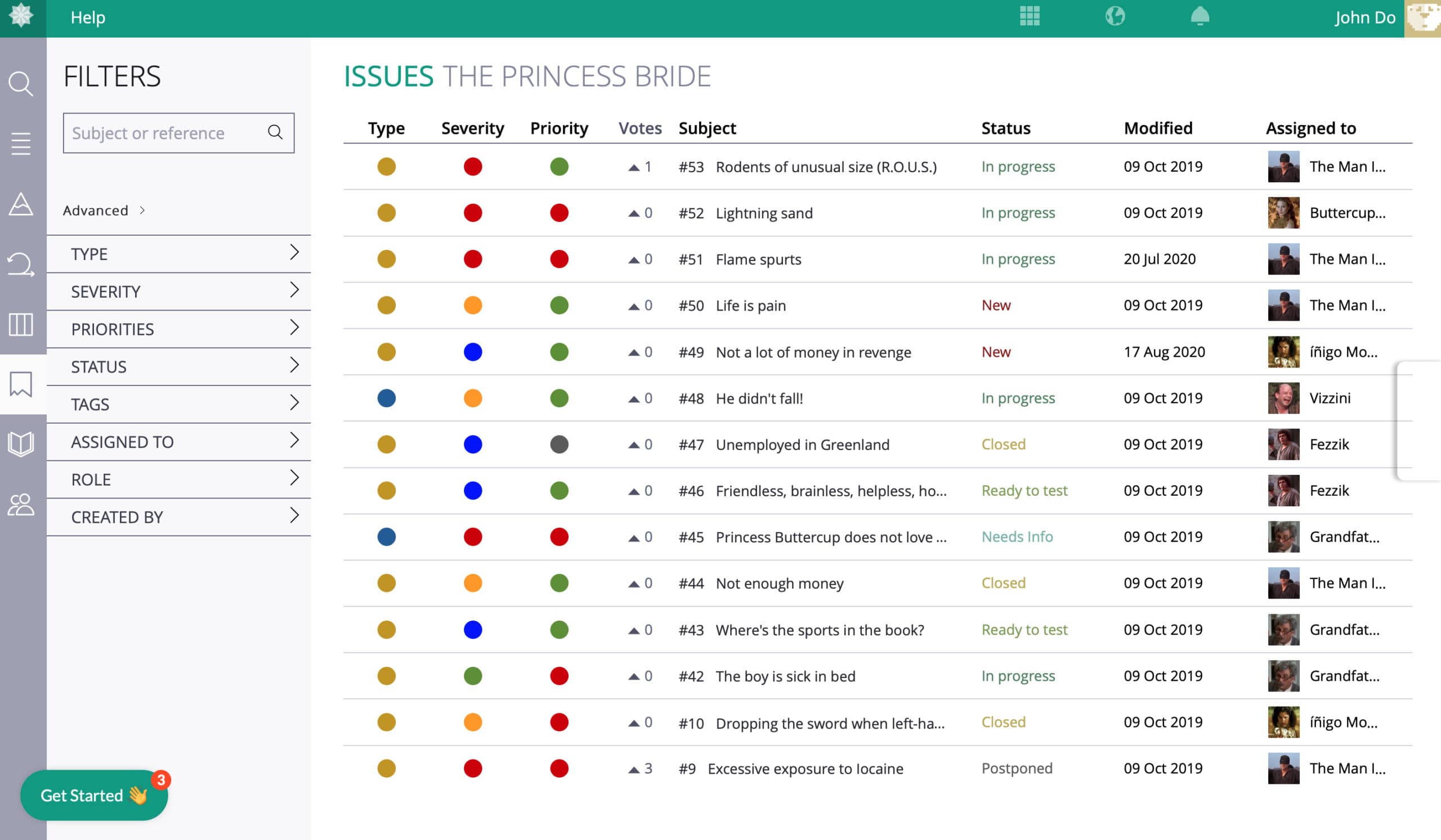


Рисунок 1.3 – Страница работы с проблемами в программном средстве Taiga.

Каждое из рассмотренных решений обладает своими сильными и слабыми сторонами. Jira подходит для крупных команд и сложных проектов, требующих детализированного управления задачами и отчетности. Confluence обеспечивает эффективное управление знаниями и документацией, особенно в связке с Jira. Taiga, с другой стороны, является отличным выбором для небольших команд и стартапов, которые нуждаются в простом и доступном инструменте для управления проектами по методологии Agile.

* 1. **Выбор и обоснование языков программирования, фреймворков, библиотек, СУБД для разработки программного средства**

Для разработки программного средства автоматизации бизнес-процессов информационного отдела организации выбран стек технологий, включающий Python, Django, JavaScript, Bootstrap, PostgreSQL. Рассмотрим обоснование выбора каждого из этих компонентов, исходя из их преимуществ, соответствия поставленным задачам и широкого применения в современном программировании.

Python – это один из самых популярных языков программирования в мире, известный своей простотой, гибкостью и широкими возможностями для разработки различных типов приложений. В рамках данного проекта Python был выбран для реализации серверной части программного средства автоматизации бизнес-процессов информационного отдела. Рассмотрим подробнее причины этого выбора, его преимущества и недостатки, а также особенности работы с языком в контексте проекта.

Python является высокоуровневым интерпретируемым языком программирования с поддержкой нескольких парадигм, таких как объектно-ориентированное программирование, функциональное программирование и императивное программирование. Его популярность обусловлена легкостью освоения благодаря читаемому синтаксису, который делает код понятным даже для начинающих разработчиков. Это особенно важно для командной разработки и поддержки проектов, так как снижает порог вхождения для новых участников команды и ускоряет процесс разработки.

Преимущества Python:

1. Простота и читаемость кода. Синтаксис Python интуитивно понятен и минималистичен, что делает код легко читаемым и сопровождаемым. Это позволяет разработчикам сосредоточиться на решении задач, а не на технических деталях. Простота языка ускоряет написание программ, что важно при разработке комплексных бизнес-систем, требующих быстрой адаптации и изменений.
2. Широкий набор библиотек и фреймворков. Одной из ключевых причин выбора Python является его мощная экосистема библиотек, которые упрощают разработку и расширяют возможности языка. Для веб-разработки активно используются такие фреймворки, как Django, которые позволяют быстро создавать веб-приложения и API. В данном проекте Python будет использоваться для обработки запросов от фронтенд-части и взаимодействия с базой данных PostgreSQL через ORM-библиотеки, такие как SQLAlchemy.
3. Универсальность. Python может использоваться для различных задач: от создания веб-приложений до автоматизации рабочих процессов и анализа данных. Это делает его универсальным инструментом для разработки бизнес-решений, где необходимо решать широкий спектр задач, связанных с автоматизацией и обработкой данных. В контексте данного проекта Python будет отвечать за обработку заявок пользователей, управление календарем видеоконференций и генерацию отчетов.
4. Активное сообщество и документация. Python имеет огромное сообщество разработчиков, что обеспечивает доступ к множеству готовых решений и поддержке. Это помогает в случае возникновения проблем и ускоряет процесс разработки, так как множество задач уже решено и задокументировано.
5. Интеграция с другими системами. Python легко интегрируется с различными базами данных, системами управления доступами и другими внешними сервисами. В данном проекте Python будет взаимодействовать с PostgreSQL для работы с данными, а также обеспечивать интеграцию с электронной почтой и календарями для автоматизации планирования и уведомлений.

Недостатки Python:

1. Скорость выполнения. Python является интерпретируемым языком, что делает его медленнее по сравнению с компилируемыми языками, такими как C++ или Java. Для высоконагруженных приложений, требующих максимальной производительности, Python может быть недостаточно быстрым. Однако для большинства задач, связанных с автоматизацией бизнес-процессов, производительности Python вполне достаточно.
2. Ограниченная многозадачность. Одной из известных проблем Python является наличие глобальной блокировки интерпретатора, которая ограничивает одновременное выполнение потоков. Это может стать узким местом в многопоточных приложениях, где требуется параллельное выполнение задач. Тем не менее, для веб-приложений и API, использующих асинхронную обработку запросов, это ограничение не критично.
3. Необходимость использования сторонних библиотек. Несмотря на богатую экосистему, разработчикам часто приходится полагаться на сторонние библиотеки для реализации определенных функций. Это может создавать сложности с совместимостью при обновлениях или возникновении ошибок в этих библиотеках. Во избежание проблем важно тщательно выбирать библиотеки, активно поддерживаемые сообществом.
4. Ресурсоемкость. Python потребляет больше памяти и ресурсов по сравнению с некоторыми другими языками, особенно при работе с большими объемами данных или сложными операциями. Это может стать проблемой для приложений, развернутых на серверах с ограниченными ресурсами, однако современные серверные решения позволяют минимизировать это влияние.

Особенности использования Python в данном проекте: Python будет использоваться для разработки серверной части системы, которая будет обеспечивать обработку пользовательских запросов, взаимодействие с базой данных и генерацию уведомлений. Серверная часть приложения будет реализована с использованием Django, что позволит быстро развернуть REST API для связи с фронтенд-частью. Python будет взаимодействовать с PostgreSQL для хранения данных о задачах, видеоконференциях, заявках на доступы и проблемах пользователей. Интеграция с системами электронной почты позволит автоматизировать процесс уведомлений и рассылки приглашений на видеоконференции.

Важной частью проекта станет обработка данных и их представление в виде отчетов. Для этого Python предоставляет множество инструментов, таких как Pandas для анализа данных и Matplotlib для визуализации. Эти библиотеки позволяют гибко обрабатывать и представлять данные, что важно для контроля и оценки эффективности работы информационного отдела.

Django – это высокоуровневый веб-фреймворк на языке Python, который позволяет быстро и эффективно создавать сложные веб-приложения. Django был разработан для того, чтобы автоматизировать рутинные задачи веб-разработки, предоставляя разработчикам инструменты для создания надежных, безопасных и масштабируемых приложений. В рамках данного проекта Django выбран для реализации серверной части системы автоматизации бизнес-процессов информационного отдела. Рассмотрим подробнее особенности Django, его преимущества и недостатки, а также его роль в проекте.

Django является фреймворком с принципом "batteries included" (всё включено), что означает, что он поставляется с набором готовых решений для наиболее распространённых задач веб-разработки. Это позволяет разработчикам сосредоточиться на бизнес-логике приложения, а не на создании базовой инфраструктуры. Django отлично подходит для создания сложных корпоративных систем, таких как автоматизация бизнес-процессов, благодаря своей гибкости и расширяемости.

Преимущества Django:

1. Быстрая разработка. Django изначально разрабатывался для ускорения процесса создания веб-приложений. Он предоставляет мощные встроенные инструменты для работы с базами данных, управления аутентификацией пользователей, администрирования и многого другого. Это значительно ускоряет разработку серверной части приложения. В рамках данного проекта это позволит быстро реализовать функционал обработки заявок, управления пользователями и интеграции с базой данных PostgreSQL.
2. Модульная архитектура. Django поддерживает модель MVC (Model-View-Controller), что делает код структурированным и легко сопровождаемым. Это важно при разработке сложных систем, таких как автоматизация бизнес-процессов, где требуется четкое разделение логики данных, представления и контроля над процессами. В данном проекте модель данных будет отвечать за управление информацией о задачах, заявках, видеоконференциях и пользователях, в то время как контроллеры будут обрабатывать пользовательские запросы и взаимодействие с API.
3. Безопасность. Безопасность является одним из приоритетов Django. Он включает встроенные механизмы защиты от наиболее распространенных веб-угроз, таких как SQL-инъекции, межсайтовый скриптинг (XSS), атаки с подделкой межсайтовых запросов (CSRF) и другие. В данном проекте, где требуется обработка конфиденциальной информации, такой как заявки на доступ и запросы на устранение проблем, это обеспечивает высокий уровень защиты данных.
4. Встроенная панель администратора. Django предоставляет мощную встроенную панель администратора, которая позволяет управлять данными и пользователями через удобный веб-интерфейс. Это значительно упрощает процесс управления системой для сотрудников отдела, так как администраторы смогут контролировать доступы, обрабатывать заявки и просматривать отчеты без необходимости разрабатывать отдельные инструменты для администрирования.
5. Масштабируемость и производительность. Django поддерживает масштабируемость и позволяет работать с большими объемами данных. В проекте, направленном на автоматизацию бизнес-процессов, это важно, так как с ростом организации и увеличением объема заявок система должна быть способна выдерживать возросшую нагрузку. Django также легко интегрируется с такими решениями, как Docker и Kubernetes, что упрощает развертывание и управление масштабируемыми приложениями.
6. Совместимость с различными базами данных. Django поддерживает работу с множеством реляционных баз данных, включая PostgreSQL, MySQL и SQLite. В данном проекте будет использоваться PostgreSQL, которая обеспечивает надежную работу с большими объемами данных и поддерживает сложные транзакции. Django предоставляет ORM (Object-Relational Mapping), что упрощает работу с базой данных, позволяя писать запросы на уровне Python без необходимости напрямую использовать SQL.

Недостатки Django:

1. Монотонность и шаблонность. Django предоставляет очень строгую структуру и набор инструментов, что иногда ограничивает гибкость разработчиков. В рамках небольших проектов, где требуется больше свободы в реализации, использование Django может показаться излишним, так как фреймворк навязывает свои подходы к организации кода и процессов. В больших системах это, однако, скорее преимущество, так как строгая структура упрощает поддержание проекта.
2. Производительность для высоконагруженных систем. Хотя Django хорошо масштабируется, его производительность может быть ниже, чем у некоторых более легковесных фреймворков, таких как Flask, если не оптимизировать приложение. В проектах с высокими требованиями к производительности может потребоваться дополнительная настройка и использование дополнительных инструментов для повышения скорости работы.
3. Крутая кривая обучения для новичков. Для начинающих разработчиков Django может показаться сложным из-за большого количества встроенных инструментов и компонентов. Однако в контексте данного проекта, где требуется высокая функциональность, это компенсируется тем, что фреймворк предоставляет готовые решения для большинства задач.

Django будет использоваться для реализации серверной части системы, обеспечивая взаимодействие между клиентской частью, написанной на JavaScript с использованием React, и базой данных PostgreSQL. Django будет отвечать за обработку заявок на доступы, видеоконференции и запросы на устранение проблем, а также за управление пользователями и правами доступа.

С помощью встроенной ORM Django обеспечит удобное взаимодействие с PostgreSQL, автоматизируя процесс создания, чтения, обновления и удаления данных. Это позволит быстро обрабатывать запросы пользователей и генерировать отчеты о выполненных задачах и заявках.

Кроме того, Django обеспечит безопасную работу системы благодаря встроенным механизмам защиты от распространенных атак и уязвимостей. Панель администратора Django предоставит сотрудникам информационного отдела простой инструмент для управления системой, позволяя контролировать заявки, задачи и доступы.

Также Django легко интегрируется с технологиями контейнеризации, такими как Docker, что позволит развернуть приложение в контейнерах, обеспечивая гибкость и простоту управления окружением разработки, тестирования и продакшена.

JavaScript – это основной язык программирования, используемый для разработки клиентской части (frontend) веб-приложений. В рамках данного проекта JavaScript выбран для создания интерактивного и динамичного пользовательского интерфейса, что позволяет эффективно взаимодействовать с сервером, обрабатывать запросы пользователей и обновлять данные на страницах без перезагрузки. Рассмотрим подробнее особенности JavaScript, его преимущества и недостатки, а также причины выбора для автоматизации бизнес-процессов информационного отдела.

JavaScript – это высокоуровневый, динамически типизированный язык программирования, изначально разработанный для работы с веб-страницами в браузере. Со временем его применение значительно расширилось, и теперь JavaScript используется не только на стороне клиента, но и на сервере (благодаря таким средам, как Node.js). В данном проекте JavaScript будет применяться для создания интерактивных элементов пользовательского интерфейса и взаимодействия с серверной частью, написанной на Python, через API.

Преимущества JavaScript:

1. Асинхронность и скорость. Одним из ключевых преимуществ JavaScript является его способность выполнять асинхронные операции, такие как отправка запросов к серверу без блокировки интерфейса пользователя. Это особенно важно для веб-приложений, где нужно быстро обрабатывать запросы и обновлять интерфейс. В контексте данного проекта это позволит пользователям отправлять заявки на доступы, видеоконференции и запросы на устранение проблем без необходимости перезагрузки страницы.
2. Интерактивность и динамичность. JavaScript позволяет создавать динамичные веб-страницы с возможностью изменения содержимого без полной перезагрузки. Это позволяет реализовать удобные формы для отправки заявок и планирования видеоконференций, а также предоставлять пользователям актуальную информацию о состоянии задач и сервисов в режиме реального времени.
3. Кросс браузерная поддержка. JavaScript поддерживается всеми современными браузерами, что делает его идеальным выбором для создания веб-приложений, которые будут работать на различных устройствах и платформах. Это важно в рамках проекта, так как пользователи могут работать с системой как с настольных компьютеров, так и с мобильных устройств.
4. Широкая экосистема. JavaScript обладает огромной экосистемой библиотек и фреймворков, которые могут значительно упростить разработку. Например, использование библиотек для работы с формами, валидацией данных и маршрутизацией помогает ускорить процесс создания интерфейсов.

Недостатки JavaScript:

1. Отсутствие строгой типизации. JavaScript является динамически типизированным языком, что означает отсутствие контроля типов на этапе компиляции. Это может привести к ошибкам, которые выявляются только на этапе выполнения программы, что увеличивает риск непредсказуемого поведения кода. Для решения этой проблемы в проекте можно использовать TypeScript, который добавляет статическую типизацию поверх JavaScript и улучшает качество кода.
2. Проблемы с производительностью. При работе с большими объемами данных или сложными операциями JavaScript может замедлять работу интерфейса, особенно если активно используется манипуляция с DOM (Document Object Model).
3. Разнообразие реализаций в браузерах. Несмотря на стандартизацию JavaScript, его реализация в различных браузерах может немного отличаться, что требует дополнительного тестирования и корректировки кода для обеспечения кросс браузерной совместимости. Однако современные библиотеки и инструменты, такие как Babel, помогают решать эту проблему, обеспечивая поддержку старых версий браузеров.
4. Сложность отладки. В отличие от языков с компиляцией, таких как C++ или Java, JavaScript выполняется в браузере, и отладка может быть сложнее из-за динамической природы языка. Для решения этой проблемы разработчики могут использовать встроенные инструменты отладки браузеров, такие как Chrome DevTools, которые позволяют инспектировать и отслеживать выполнение кода в реальном времени.

Особенности использования JavaScript в данном проекте: JavaScript будет играть ключевую роль в создании динамичного интерфейса для взаимодействия пользователей с системой. Используя React, приложение сможет обновлять данные о задачах, заявках и видеоконференциях в реальном времени, обеспечивая удобный и отзывчивый интерфейс. Асинхронная работа с сервером позволит пользователям отправлять запросы и получать данные без необходимости перезагрузки страницы, что улучшит общую производительность системы и повысит удовлетворенность пользователей.

Также, JavaScript обеспечит интеграцию с внешними сервисами, такими как календарь для планирования видеоконференций, и позволит реализовать уведомления о предстоящих событиях и изменениях в статусах заявок. Для отправки HTTP-запросов к серверу будет использована библиотека Axios.

Bootstrap – это популярный CSS-фреймворк для разработки адаптивных и современных веб-интерфейсов.

Основные преимущества:

1. Готовые компоненты и стили, что ускоряет разработку и позволяет сосредоточиться на функциональности, а не на дизайне.
2. Поддержка адаптивной верстки, что делает веб-приложения доступными на различных устройствах, включая мобильные.
3. Совместимость с React и JavaScript, что упрощает интеграцию.

Основные недостатки:

1. Однообразие дизайна: Использование стандартных компонентов Bootstrap может привести к однотипности веб-приложений, так как многие сайты используют одни и те же шаблоны и стили. Для уникального дизайна часто требуется значительная кастомизация CSS.
2. Тяжелый CSS-код: По умолчанию Bootstrap включает много неиспользуемого CSS-кода, что может замедлять загрузку страниц. Разработчикам часто приходится вручную удалять ненужные стили или применять специальные инструменты для оптимизации.
3. Ограниченная гибкость: хотя Bootstrap подходит для базового дизайна, при создании сложных пользовательских интерфейсов он может не обладать достаточной гибкостью, требуя больше ручного редактирования стилей.

PostgreSQL – это мощная объектно-реляционная система управления базами данных (СУБД) с открытым исходным кодом, которая широко используется для хранения и обработки больших объемов данных в сложных корпоративных приложениях. В рамках данного проекта PostgreSQL выбран в качестве основной базы данных для хранения информации о задачах, заявках на доступ, видеоконференциях и проблемах пользователей. Рассмотрим подробнее причины выбора PostgreSQL, её преимущества и недостатки, а также её роль в системе автоматизации бизнес-процессов информационного отдела. Это одна из самых стабильных и гибких СУБД, которая поддерживает множество расширенных функций, таких как транзакции, индексы, полнотекстовый поиск, хранение и обработка больших данных. Благодаря своей расширяемости и надежности PostgreSQL подходит для сложных бизнес-приложений, где важно не только эффективно управлять данными, но и обеспечивать их целостность, безопасность и производительность.

Преимущества PostgreSQL:

1. Надежность и целостность данных. PostgreSQL полностью соответствует стандарту ACID (Atomicity, Consistency, Isolation, Durability), что гарантирует надежность выполнения транзакций и целостность данных. Это особенно важно в данном проекте, где система будет управлять заявками и данными о видеоконференциях, и требуется, чтобы данные оставались неизменными и корректными даже при сбоях или отказах.
2. Поддержка сложных запросов и индексов. PostgreSQL поддерживает сложные SQL-запросы, включая вложенные запросы, объединения таблиц и агрегатные функции. Это позволяет реализовать сложную логику обработки данных, необходимую для генерации отчетов и аналитики по задачам и заявкам. Кроме того, PostgreSQL предоставляет возможность создания индексов для ускорения выполнения запросов, что позволяет поддерживать высокую производительность даже при работе с большими объемами данных.
3. Масштабируемость. PostgreSQL хорошо масштабируется как по вертикали, так и по горизонтали, что делает её подходящей для систем с увеличивающимися объемами данных и числами пользователей. В данном проекте это особенно важно, поскольку с ростом компании и увеличением количества задач и заявок система должна оставаться производительной и эффективно обрабатывать все запросы.
4. Поддержка JSON и NoSQL-функций. Хотя PostgreSQL является реляционной СУБД, она также поддерживает хранение и обработку данных в формате JSON. Это позволяет использовать её как гибридное решение, объединяя реляционную структуру данных с динамическими JSON-документами. Это может быть полезным в случае, если в системе потребуется хранить нестандартные или полуструктурированные данные, например, для хранения метаданных заявок или результатов видеоконференций.[9]
5. Расширяемость. PostgreSQL поддерживает расширения, которые позволяют добавлять новые функции без изменения исходного кода. Одним из таких популярных расширений является PostGIS, который добавляет поддержку работы с географическими данными. В данном проекте расширяемость PostgreSQL позволяет легко адаптировать систему к новым требованиям, таким как интеграция с внешними сервисами или добавление специфических функций обработки данных.
6. Открытый исходный код. PostgreSQL является полностью бесплатной и открытой системой, что снижает затраты на разработку и развертывание системы. Она активно развивается сообществом, что обеспечивает её актуальность и безопасность.

Недостатки PostgreSQL:

1. Сложность настройки и администрирования. Несмотря на свои возможности, PostgreSQL требует глубоких знаний для настройки и оптимизации производительности. Например, при работе с большими объемами данных необходимо настраивать параметры кэширования, индексации и разбиения таблиц на партиции для повышения скорости обработки запросов. В рамках данного проекта могут потребоваться дополнительные усилия для правильной настройки системы, чтобы обеспечить её максимальную производительность и надежность.
2. Требовательность к ресурсам. PostgreSQL требует значительных ресурсов для работы с большими объемами данных, особенно при выполнении сложных аналитических запросов и обработке транзакций. Это может стать проблемой на серверах с ограниченными вычислительными мощностями.
3. Медлительность при выполнении простых запросов. В некоторых случаях PostgreSQL может уступать другим СУБД, таким как MySQL, при выполнении простых запросов или операций на небольших объемах данных. Это связано с тем, что PostgreSQL изначально ориентирован на более сложные транзакции и обработку данных. Для оптимизации производительности можно использовать индексы и кэширование.

В рамках системы автоматизации бизнес-процессов информационного отдела PostgreSQL будет использоваться для хранения данных о задачах, заявках на доступ, видеоконференциях и проблемах пользователей. Система будет обрабатывать и хранить большие объемы информации, и PostgreSQL обеспечит высокую надежность, целостность и производительность при работе с этими данными.

Одной из ключевых задач PostgreSQL будет обеспечение быстрого и надежного выполнения запросов на обработку заявок и видеоконференций. Пользователи должны иметь возможность отправлять заявки и отслеживать их статус в режиме реального времени, а система – мгновенно обновлять информацию. Для этого будут использоваться индексы и оптимизированные SQL-запросы, которые позволят ускорить обработку данных.

PostgreSQL также обеспечит безопасное хранение данных и защиту от потерь. Встроенная поддержка транзакций гарантирует, что все изменения данных будут сохраняться в системе корректно, даже в случае сбоя. Это особенно важно для обработки заявок, где требуется гарантировать целостность данных и их корректную обработку.

Кроме того, PostgreSQL будет интегрирована с серверной частью, реализованной на Django. Django использует встроенную ORM (Object-Relational Mapping), которая позволяет взаимодействовать с базой данных PostgreSQL на уровне Python-кода, что упрощает работу с данными и ускоряет процесс разработки.

## **Укрупненная спецификация требований**

На основе проанализированных аналогов и выявленных недостатков существующих решений сформирована укрупнённая спецификация требований к разрабатываемому программному средству. Основная задача заключается в автоматизации ключевых бизнес-процессов информационного отдела, включая управление расписанием, обработку заявок на доступ к корпоративным системам, решение проблем пользователей и предоставление информации о доступных сервисах.

Программное средство должно обеспечивать управление расписанием, предоставляя пользователям возможность создавать, редактировать и удалять события, такие как встречи, совещания и видеоконференции. Встроенный календарь должен поддерживать отправку напоминаний, привязку событий к конкретным сотрудникам и визуализацию для удобства восприятия.

Одной из ключевых функций является обработка заявок на доступ к корпоративным системам. Пользователи смогут подать заявку, указав необходимый сервис или уровень доступа. Система должна автоматически распределять заявки для рассмотрения ответственными лицами, отслеживать их статусы (ожидание, утверждено, отклонено) и сохранять историю изменений для дальнейшего анализа.

Для решения пользовательских проблем система будет поддерживать регистрацию и обработку инцидентов. Пользователи смогут создавать заявки, описывая суть проблемы, которая автоматически передаётся администраторам. Система должна фиксировать статус выполнения заявки, время её закрытия и обеспечивать уведомление пользователей о прогрессе.

Кроме того, программное средство должно включать модуль управления корпоративными сервисами. Этот модуль отображает актуальную информацию о доступных сервисах, их статусе (активен, временно недоступен, отключён) и обновлениях. Пользователи должны иметь возможность просматривать эту информацию в удобной форме, а администраторы – обновлять её при необходимости.

Входными данными системы являются пользовательские данные (имя, должность, контакты), информация о заявках на доступ и инциденты, параметры событий календаря и данные о корпоративных сервисах. Выходные данные включают уведомления о предстоящих событиях, статусы заявок и инцидентов, списки доступных сервисов и отчёты о действиях пользователей, которые помогут анализировать эффективность работы отдела.

Программное средство разрабатывается с учётом функциональных и нефункциональных требований. Оно должно быть интуитивно понятным для конечных пользователей, обеспечивать высокую производительность и надёжность. Особое внимание уделяется безопасности данных, интеграции с существующими корпоративными системами, а также возможности масштабирования для учёта роста организации и её потребностей.

Таким образом, разработка призвана устранить текущие недостатки в управлении бизнес-процессами информационного отдела, повысить эффективность работы сотрудников, улучшить координацию между отделами и обеспечить удобство выполнения рутинных операций.

1. **МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ** 
   1. **Разработка функциональной модели**

Диаграмма A-0 для системы автоматизации бизнес-процессов пользователей и администраторов отражает общий контекст работы системы, объединяя основные функции и их взаимодействие. Основная цель системы – автоматизировать ключевые процессы, связанные с управлением заявками, событиями, доступами и учётом оборудования, а также упростить взаимодействие между пользователями и администраторами.

Входные данные системы формируют основу её работы. К ним относятся информация о пользователях, включающая их ФИО, подразделение, адрес электронной почты и пароль, а также запросы пользователей. Такие запросы могут включать заявки на доступ к сервисам, создание событий в корпоративном календаре, подачу заявок на устранение проблем, а также обновление учётной информации об оборудовании.

Результаты работы системы представляют собой преобразованные данные и итоговые действия. Среди них: статусы заявок (например, одобрено или отклонено), обновлённые списки доступных сервисов, корректные записи в календаре событий и учёт оборудования, отражающий текущее состояние активов компании.

Работа системы регулируется управляющими факторами, такими как политики безопасности организации, правила обработки заявок в зависимости от их приоритетности, а также внутренние регламенты работы с оборудованием и сервисами. Эти факторы обеспечивают соблюдение требований безопасности, прозрачность обработки запросов и упрощают контроль за выполнением задач.

На уровне диаграммы A-0 система представлена как единый функциональный блок, который включает все описанные выше элементы и процессы. Эта диаграмма является отправной точкой для создания более детализированных диаграмм, которые подробно описывают отдельные функции, такие как управление заявками, планирование событий или учёт оборудования. Каждая из этих функций может быть детализирована до подфункций, что позволяет глубже понять логику работы системы и её элементов.

Контекстная диаграмма позволяет наглядно отразить, как информация и действия проходят через систему, преобразуюсь в результаты, соответствующие требованиям и ожиданиям пользователей и администраторов. Она является основой для дальнейшего проектирования системы. Контекстная диаграмма представлена на рисунке 2.1.

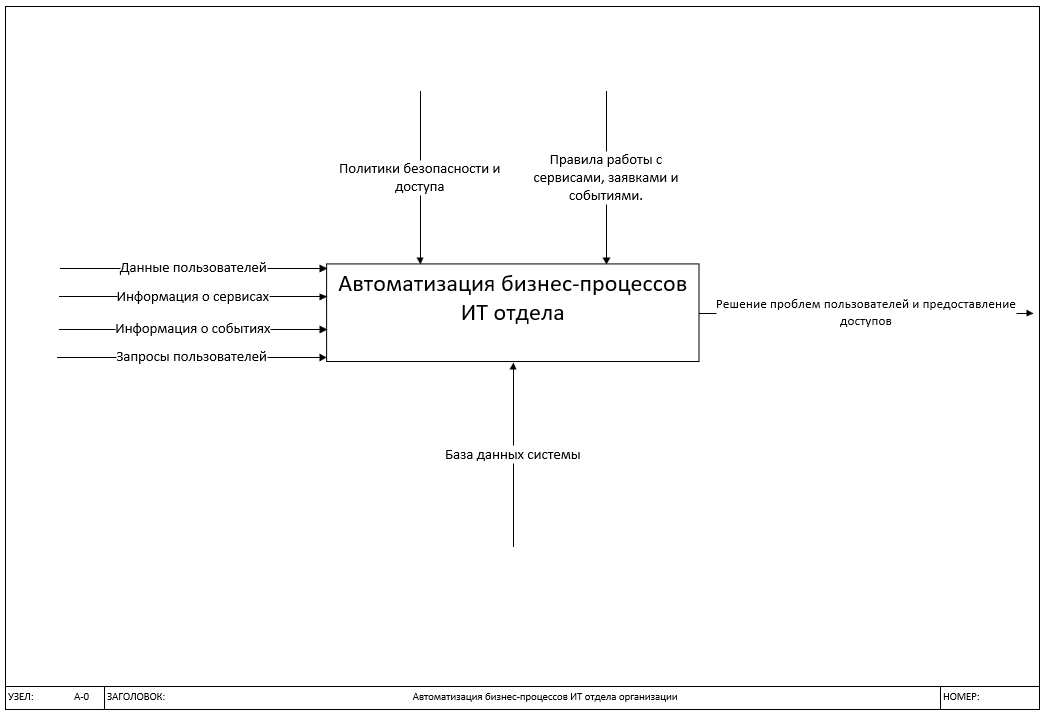


Рисунок 2.1 – Контекстная диаграмма

Диаграмма IDEF0 демонстрирует ключевые процессы и функции системы автоматизации бизнес-процессов информационного отдела, ориентированной на пользователей и администраторов. Главная функция, представленная на уровне A0, называется «Автоматизация бизнес-процессов пользователей и администраторов». Она объединяет цели системы, такие как управление сервисами, заявками на доступ, календарём событий, обработкой проблем и учётом оборудования. Система направлена на упрощение взаимодействия между сотрудниками и администраторами, оптимизацию задач и контроль за их выполнением.

Входными данными для системы являются запросы от пользователей, информация о событиях и проблемах, данные об оборудовании и информация о пользователях. Управляющими факторами выступают политики безопасности, организационные правила и приоритеты обработки данных. Механизмами являются база данных, интерфейс веб-приложения, сервер уведомлений и действия администраторов. Выходными данными являются статусы заявок и проблем, обновлённые события в календаре, добавленные сервисы и актуализированный учёт техники.

Функции и их описание на детализированном уровне A0 включают:

1. Регистрация и аутентификация пользователей. Эта функция обеспечивает вход пользователей и администраторов в систему. Входными данными являются ФИО, название отдела, почта и пароль при регистрации, а также почта и пароль для входа. Управляющими факторами служат политики безопасности и правила валидации данных. Результатом выполнения функции является создание новой учётной записи или предоставление доступа к системе. Механизмами являются база данных для хранения данных пользователей и интерфейс веб-приложения, где происходит ввод данных.
2. Изменение списка сервисов. Функция отвечает за выбор сервисов пользователями. Входные данные включают заявки на доступ и выбор сервисов со стороны пользователей. Управляющими факторами выступают политики доступа, правила предоставления прав и организационные приоритеты. На выходе формируются обновлённые статусы заявок, информация о выбранных сервисах и предоставленные права доступа. Механизмами функции являются действия администратора, база данных заявок и интерфейс системы, который отображает статусы для пользователей.
3. Управление календарём и событиями. Эта функция позволяет пользователям и администраторам планировать и управлять событиями. Пользователи могут создавать встречи, указывая дату, время, место и название события, а администраторы управляют всеми записями, включая удаление событий. Управляющими факторами являются график работы сотрудников и правила синхронизации времени. Выходные данные – это список событий с учётом всех обновлений. Механизмами функции выступают календарь системы, база данных событий и интерфейс для взаимодействия с пользователем.
4. Работа с сервисами, доступами, проблемами. Функция включает в себя отправку заявок на устранение проблем пользователями, запрос на доступы и ведение учёта оборудования администраторами. Входными данными являются запросы на устранение проблем с заголовком, описанием и категорией, а также информация об оборудовании, такая как тип, марка, инвентарный номер и ответственное лицо. Управляющими факторами служат приоритеты задач и организационные стандарты учёта. Выходные данные включают статусы проблем (выполненные, в ожидании, отклонённые) и актуальный список техники. Механизмами выступают база данных для хранения информации, интерфейс для подачи заявок и действия администраторов.

Таким образом, диаграмма IDEF0 на уровне A0 детализирует, как каждая функция системы взаимодействует с входными данными, управляется различными факторами и использует доступные механизмы для формирования требуемых результатов. Общая система автоматизации позволяет пользователям и администраторам эффективно управлять сервисами, доступами, событиями, проблемами и учётом оборудования. В результате достигается оптимизация рабочих процессов, повышение прозрачности задач и улучшение координации между сотрудниками и администраторами. Диаграмма декомпозиции представлена на рисунке 2.2.

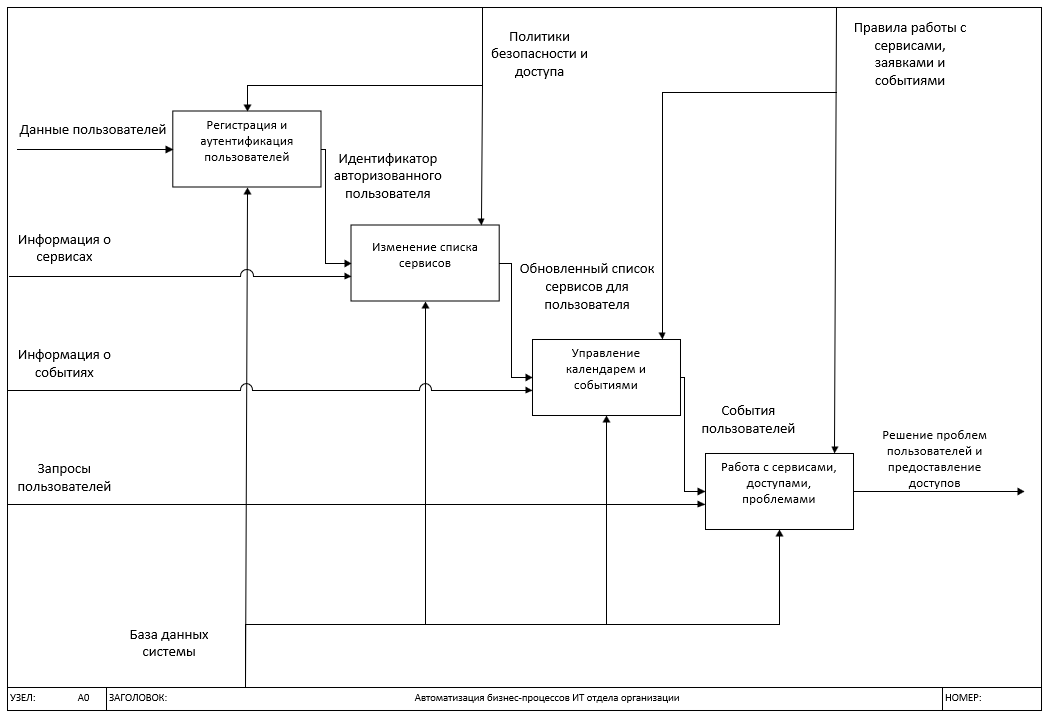


Рисунок 2.2 – Диаграмма декомпозиции

## **Разработка диаграммы последовательности**

Диаграмма последовательности для процесса подачи заявки на доступ отражает взаимодействие между участниками системы и демонстрирует последовательность их действий. Основные участники диаграммы: пользователь (представленный человечком), веб-интерфейс, сервер и база данных. Каждый из этих участников выполняет свою часть работы, чтобы заявка была успешно подана и сохранена в системе.

Процесс начинается с действия пользователя, который взаимодействует с веб-интерфейсом. Пользователь заходит в раздел подачи заявок, где отображается форма для ввода данных. В этой форме пользователь выбирает нужный сервис из списка, может добавить комментарий (если требуется) и подтверждает отправку заявки, нажимая кнопку.

После нажатия кнопки веб-интерфейс формирует запрос к серверу. Этот запрос содержит данные, введенные пользователем, включая идентификатор выбранного сервиса и дополнительные комментарии. Сервер, получив запрос, начинает процесс обработки данных.

Если все проверки пройдены успешно, сервер формирует новую запись заявки и отправляет запрос на сохранение этой записи в базу данных. База данных принимает запрос и создает новую запись, которая включает идентификатор пользователя, идентификатор сервиса, дату подачи заявки и начальный статус (например, «Ожидает обработки»). После успешного сохранения записи база данных возвращает серверу подтверждение.

Сервер, получив подтверждение, завершает обработку заявки и формирует ответ для веб-интерфейса. Веб-интерфейс обновляется, показывая пользователю сообщение о том, что заявка успешно отправлена. Пользователь может видеть свою заявку в обновленном списке заявок, где отображаются данные о поданных заявках, включая название сервиса, дату подачи и текущий статус.

Диаграмма последовательности подробно отражает каждую из этих операций, начиная от взаимодействия пользователя с интерфейсом и заканчивая подтверждением успешной отправки заявки. Логика построения диаграммы позволяет визуально представить все взаимодействия между участниками процесса и убедиться, что система работает корректно на каждом этапе. Диаграмма последовательности показана на рисунке 2.3.

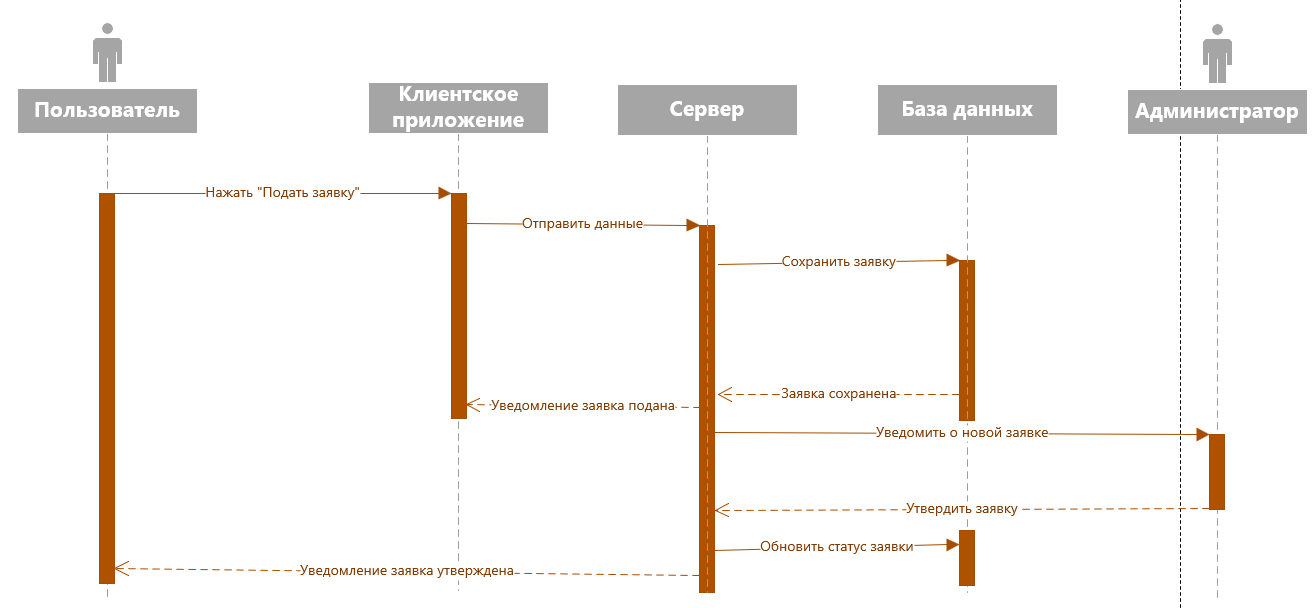


Рисунок 2.3 – Диаграмма последовательности

## **Разработка диаграммы состояний**

Диаграмма состояний для системы автоматизации бизнес-процессов иллюстрирует различные состояния заявки и возможные переходы между ними. Она позволяет наглядно представить, как изменяется статус заявки на доступ, событие или решение проблемы в зависимости от действий пользователя или администратора.

Процесс начинается с начального состояния, обозначенного чёрным кругом, которое символизирует старт процесса – момент, когда пользователь инициирует взаимодействие с системой. Первым состоянием становится "Создание заявки", где пользователь вводит данные, такие как выбор сервиса, описание проблемы или параметры события. Это состояние отражает этап подготовки информации для обработки.

После отправки заявки она переходит в состояние "Ожидание обработки". Здесь заявка зарегистрирована, но ещё не рассмотрена администратором. Это состояние особенно важно, так как оно показывает, что заявка успешно сохранена в системе, но действия по её обработке не начались.

Далее заявка может перейти в состояние "На рассмотрении", когда администратор начинает работать с ней. Здесь система фиксирует, что заявка активна, и начинает отображать, кто и когда приступил к её обработке. Это позволяет отслеживать, какой администратор ответственен за выполнение задачи.

В зависимости от решения администратора, заявка может перейти в одно из нескольких конечных состояний. Например, в случае одобрения доступа или успешного решения проблемы заявка переходит в состояние "Одобрена" или "Выполнена". Если заявка не соответствует требованиям или проблема не может быть решена, то она попадает в состояние "Отклонена".

Конечное состояние обозначается двойным чёрным кругом, который указывает на завершение жизненного цикла заявки. Это может быть либо успешное выполнение, либо отказ, в зависимости от обстоятельств.

Диаграмма состояний также может включать дополнительные стрелки, которые отображают возвратные переходы. Например, если в процессе рассмотрения администратор запросил уточнения, заявка может вернуться из состояния "На рассмотрении" обратно в "Создана". Такие переходы делают диаграмму более гибкой и отражают реальные сценарии работы с заявками.

Эта диаграмма помогает разработчику и пользователю системы понять логику обработки заявок, взаимодействие между состояниями и возможные сценарии использования. Она позволяет выявить недочёты в процессах и, при необходимости, оптимизировать переходы между состояниями. Диаграмма состояний показана на рисунке 2.4.



Рисунок 2.4 – Диаграмма состояний

## **Разработка диаграммы вариантов использования**

Диаграмма вариантов использования для системы автоматизации бизнес-процессов информационного отдела позволяет наглядно представить взаимодействие акторов (пользователей и администраторов) с системой и отобразить сценарии, которые они могут выполнять. Основными актерами здесь являются Пользователь и Администратор, каждый из которых взаимодействует с системой по своим задачам и ролям. Варианты использования описывают функции системы, которые доступны для выполнения пользователями, а также показывают связи между сценариями и актерами. При этом в диаграмме отражаются отношения include (включение) и extend (расширение), которые помогают детализировать обязательные и дополнительные шаги выполнения функций.

В системе пользователь может выполнить процесс регистрации или входа в систему, чтобы получить доступ к основным функциям. При входе пользователь вводит учётные данные, такие как почта и пароль, которые проверяются системой. Отношение include связывает «Проверку учетных данных» с процессом входа, так как это обязательное действие, без которого пользователь не сможет продолжить работу в системе. После успешного входа пользователь получает доступ к трём основным вкладкам: Сервисы, Календарь, Доступы и Проблемы.

На вкладке Сервисы пользователь может выбрать, какие сервисы будут отображаться на его странице. Это действие не требует дополнительных проверок, но позволяет пользователю настроить интерфейс системы под свои потребности.

В разделе Календарь пользователь может добавлять события, задавая время, место и название мероприятия. Добавление события обязательно включает процесс Проверки доступности времени, который не допускает конфликтов по времени с уже существующими мероприятиями. Это отношение include, поскольку без проверки система не позволит создать событие. Дополнительно действие Добавления события может расширяться функцией Отправка уведомлений участникам, которая выполняется только после успешного создания события. Отношение extend здесь показывает, что уведомления отправляются при необходимости и являются необязательным шагом.

В разделе Доступы пользователь может подать заявку на предоставление доступа к конкретной системе или сервису. Этот процесс включает Валидацию данных заявки, где система проверяет корректность введённых пользователем данных, чтобы избежать ошибок или неполных заявок. Отношение include здесь отображает обязательность этого шага. После успешной подачи заявки выполняется Отправка уведомления администратору, которое расширяет основной сценарий и показывает, что администратор будет проинформирован о новой заявке. Это действие обозначается как extend, так как оно выполняется дополнительно при определённых условиях.

В разделе Проблемы пользователь может создать заявку на устранение проблемы, указывая заголовок, описание и категорию. Система обрабатывает эту информацию и фиксирует время отправки. Данный процесс также может включать Валидацию данных, чтобы убедиться, что все обязательные поля заполнены корректно. В зависимости от результата заявки отправляются уведомления администратору, что является расширением основного сценария и обозначается отношением extend.

Администратор взаимодействует с системой в более широком контексте и имеет доступ к функциям управления. В разделе Сервисы администратор добавляет или редактирует сервисы, указывая их названия и ссылки. Этот процесс может включать Проверку уникальности сервиса, чтобы избежать дублирования записей в базе данных. Отношение include здесь отображает обязательный шаг проверки.

В разделе Календарь администратор имеет возможность не только добавлять события, но и удалять уже существующие мероприятия любого пользователя. Добавление события, как и в случае с пользователем, обязательно включает проверку доступности времени и может расширяться функцией Отправка уведомлений участникам, которые получают информацию о созданном мероприятии.

В разделе Доступы администратор обрабатывает заявки, поданные пользователями. Основной сценарий Обработка заявок на доступ включает обязательный шаг Просмотр списка заявок, так как администратор не может обработать заявки, не изучив их. Далее процесс расширяется функцией Назначение прав доступа, которая выполняется только в случае одобрения заявки. Отношение extend здесь указывает на дополнительный шаг, который зависит от решения администратора.

Раздел Проблемы позволяет администратору управлять запросами на устранение проблем, отправленными пользователями. Обработка запроса обязательно включает Просмотр деталей проблемы, а расширение происходит на этапе Назначение статуса проблемы. Статус может быть назначен как «In Progress», «Complete» или «Reject», что выполняется по усмотрению администратора.

В разделе Учет администратор управляет учётом компьютерной техники, добавляя записи с указанием типа оборудования, модели, инвентарного номера и других данных. Этот процесс включает Проверку уникальности инвентарного номера как обязательный шаг, а при успешном добавлении записи может расширяться функцией Отправка уведомления ответственному лицу, который получает информацию о новой записи в учётной системе.

Таким образом, диаграмма вариантов использования демонстрирует, как пользователь и администратор взаимодействуют с системой для выполнения своих задач. Отношения include помогают отобразить обязательные шаги, без которых сценарий не может быть завершён, тогда как extend показывает дополнительные действия, выполняемые при определённых условиях. Диаграмма позволяет структурировать систему на уровне взаимодействия с пользователями, выявляя ключевые функции, их взаимосвязи и зависимости, что обеспечивает более точное представление требований к системе. Диаграмма вариантов использования показана на рисунке 2.5.



Рисунок 2.5 – диаграмма вариантов использования

* 1. **Разработка информационной модели базы данных**

Диаграмма IDEF1X наглядно отражает ключевые сущности системы и их связи, включая такие таблицы, как Users, Services, CalendarEvents, AccessRequests, Problems, Equipment и другие, обеспечивающие функциональность управления бизнес-процессами.

Таблица Users – центральная сущность, содержащая информацию обо всех пользователях системы. Связана с таблицей Roles через поле role\_id, что позволяет указывать роли пользователя (например, администратор, пользователь). Также связана с таблицами CalendarEvents, AccessRequests, Problems и Equipment, обеспечивая их взаимодействие.

Таблица Departments хранит информацию о структурных подразделениях организации, таких как название отдела и описание. Эта таблица связана с таблицей Users через поле department\_id, указывая, в каком отделе работает каждый пользователь.

Таблица Roles содержит перечень доступных ролей в системе, таких как администратор или пользователь. Она позволяет добавлять гибкость в разграничении прав.

Таблица Services представляет перечень доступных сервисов. Связь с таблицей UserServices позволяет пользователям выбирать, какие сервисы отображать на своей странице.

Таблица CalendarEvents хранит события, создаваемые пользователями. Администратор имеет возможность редактировать или удалять события любого пользователя.

Таблица AccessRequests отвечает за управление заявками на доступ к системам. Она связана с Users и Services, позволяя отслеживать заявки, их статус и предоставленные права.

Таблица Problems фиксирует заявки пользователей на устранение проблем. Связана с Users для определения автора заявки и позволяет администраторам управлять их статусом.

Таблица Equipment используется для учета техники. Она связана с ответственными лицами через responsible\_person, указывая ответственного пользователя.

Таблица UserServices реализует связь многие-ко-многим между таблицами Users и Services. Она позволяет пользователям выбирать и сохранять список сервисов, которые будут отображаться на их странице.

Таблица AdminsActions хранит логи действий администраторов, таких как одобрение заявок, редактирование событий или управление сервисами. Эта таблица помогает вести аудит и предоставляет возможность отслеживать историю изменений в системе. Связь с таблицей Users (администраторов) позволяет определить, кто именно выполнил то или иное действие.

Диаграмма IDEF1X охватывает все ключевые элементы базы данных, обеспечивая структурированное хранение информации и позволяя эффективно реализовать функции автоматизации бизнес-процессов информационного отдела. Эти связи и ограничения помогают поддерживать целостность данных, их актуальность и упрощают доступ к нужной информации, что критически важно для успешного функционирования системы. Диаграмма IDEF1X показана на рисунке 2.6.

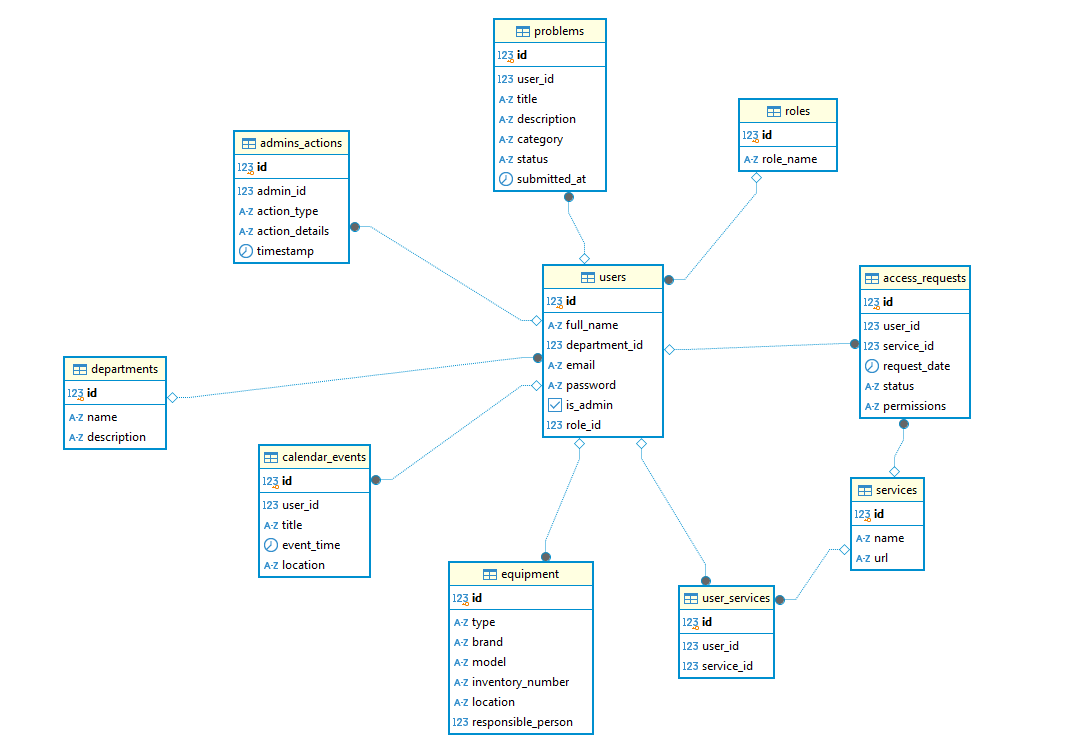


Рисунок 2.6 – Диаграмма IDEF1X

## **2.6 Спецификация требований**

Спецификация требований – это документ, описывающий функциональные и нефункциональные требования к разрабатываемому программному обеспечению. В данном проекте, посвященном автоматизации бизнес-процессов информационного отдела организации, спецификация требований определяет ключевые задачи и параметры работы системы, которые обеспечат эффективную реализацию всех бизнес-процессов.

Функциональные требования:

1. Система должна поддерживать регистрацию пользователей с возможностью указания их роли (администратор, сотрудник, гость). Администратор должен иметь возможность назначать роли пользователям, управлять правами доступа, а также блокировать или удалять учетные записи. Роли определяют, какие функции системы доступны пользователю, например, администраторы могут создавать задачи и управлять заявками, а сотрудники и гости – только отправлять заявки и получать уведомления.
2. Система должна предоставлять интерфейс для подачи заявок на доступ к корпоративным системам. Пользователи могут выбирать, к каким системам им необходим доступ, указывать тип доступа (чтение, запись, полные права) и отправлять заявку на рассмотрение администратору. Администраторы должны иметь возможность одобрять или отклонять заявки, а пользователи – отслеживать статус заявок в режиме реального времени.
3. Пользователи должны иметь возможность отправлять заявки на решение технических проблем, с которыми они сталкиваются при работе с корпоративными системами или оборудованием. В заявке указывается описание проблемы, приоритет, а также возможность прикрепления файлов (скриншотов или логов). Администраторы должны назначать ответственных сотрудников для решения проблемы и отслеживать её статус.
4. Система должна поддерживать календарь для планирования видеоконференций. Пользователи могут бронировать время для встреч, указывать участников и отправлять им приглашения. Функционал видеоконференций включает возможность отправки напоминаний перед началом встречи, управление участниками и предоставление ссылки на видео встречу.
5. Система должна поддерживать возможность создания проектов и задач, назначать их сотрудникам, а также отслеживать выполнение задач. Администраторы могут создавать задачи с указанием сроков, приоритета и ответственных сотрудников. Система должна поддерживать различные статусы задач (например, новая, в процессе, завершена) и предоставлять пользователям возможность фильтрации задач по приоритетам, срокам и статусам.
6. На отдельной странице системы пользователи должны видеть список доступных корпоративных сервисов, таких как CRM, ERP-системы, электронная почта и другие. Администраторы должны обновлять информацию о доступности сервисов, указывая их статус (в работе, временно недоступен). Пользователи должны получать уведомления о недоступности сервисов или об изменении их статуса.
7. Система должна предоставлять возможность отправки уведомлений пользователям. Уведомления должны информировать о новых заявках, изменениях статуса задач или приближающихся видеоконференциях. Пользователи могут получать уведомления через интерфейс системы или по электронной почте, в зависимости от настроек.
8. Система должна поддерживать возможность генерации отчетов по заявкам, задачам и видеоконференциям. Отчеты должны включать информацию о завершенных и текущих задачах, успешности обработки заявок, количестве видеоконференций, а также метрики производительности сотрудников (например, количество решенных проблем).

Нефункциональные требования

1. Система должна обрабатывать запросы пользователей с минимальными задержками. Время отклика не должно превышать 2 секунд на типовые операции, такие как создание заявки, обновление задачи или изменение статуса заявки. Для обеспечения быстрой работы при большом количестве пользователей и заявок должны использоваться индексы для ключевых полей в базе данных.
2. Система должна быть надежной и устойчивой к сбоям. Все данные о задачах, заявках и пользователях должны сохраняться в базе данных с возможностью восстановления после аварийных ситуаций. Для этого необходимо реализовать регулярное резервное копирование базы данных. Система должна поддерживать отказоустойчивую архитектуру, чтобы минимизировать простои при технических сбоях.
3. Система должна обеспечивать защиту от несанкционированного доступа и атак. Для этого необходимо реализовать многофакторную аутентификацию для пользователей, а также ограничение прав доступа в зависимости от роли. Все данные должны передаваться через зашифрованные соединения (например, через HTTPS). Действия пользователей должны логироваться для последующего аудита.
4. Система должна быть масштабируемой, чтобы поддерживать рост числа пользователей и объема данных без ухудшения производительности. Это достигается за счет горизонтального и вертикального масштабирования сервера, а также использования контейнеризации и оркестрации через Docker и Kubernetes для гибкого управления инфраструктурой.
5. Система должна корректно работать во всех современных веб-браузерах (Google Chrome, Mozilla Firefox, Safari и Microsoft Edge). Она также должна поддерживать кроссплатформенность, обеспечивая доступ через различные операционные системы, включая Windows, macOS и Linux. Также необходима интеграция с внешними системами, такими как корпоративная электронная почта и календарные сервисы.
6. Интерфейс системы должен быть интуитивно понятным и удобным для пользователя. Это особенно важно для сотрудников, которые не обладают глубокими техническими знаниями. Адаптивный дизайн должен обеспечивать корректное отображение на экранах различных размеров, включая мобильные устройства.

Серверная часть системы разрабатывается на языке Python с использованием фреймворка Django, который обеспечивает высокую производительность, надежность и масштабируемость. Django предоставляет удобные инструменты для работы с базами данных, создания REST API и реализации сложной бизнес-логики. Это позволяет эффективно управлять запросами пользователей, обработкой данных и взаимодействием с клиентской частью. Bootstrap упрощает создание интуитивно понятного интерфейса благодаря встроенным компонентам и стилям, что ускоряет разработку и делает взаимодействие пользователя с системой более удобным.

Клиентская часть реализуется с использованием JavaScript и фреймворка Bootstrap. Этот выбор обеспечивает адаптивный дизайн и удобство использования интерфейса на различных устройствах, включая мобильные.

Хранение данных осуществляется в реляционной базе данных PostgreSQL, которая обеспечивает высокую надежность, целостность данных и возможность обработки большого объема информации. PostgreSQL поддерживает продвинутые функции, такие как индексы, триггеры и транзакции, что делает её подходящей для сложных систем с высокой нагрузкой.

Такой выбор технологий и подход к разработке обеспечивают соответствие системы всем функциональным и нефункциональным требованиям, гарантируя её производительность, гибкость и устойчивость к изменениям в будущем.

# 3 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

## **3.1 Проектирование и разработки архитектуры программного средства**

Архитектура разрабатываемой системы автоматизации бизнес-процессов построена на основе клиент-серверной модели, которая обеспечивает четкое разделение функций между серверной и клиентской частями. Такое решение упрощает разработку, поддержку и масштабирование системы.

Серверная часть реализована на Python с использованием фреймворка Django, который позволяет эффективно управлять бизнес-логикой и взаимодействием с базой данных. Django предоставляет встроенные механизмы для построения REST API, что обеспечивает надежное и безопасное взаимодействие между клиентом и сервером. Также серверная часть отвечает за проверку прав доступа, обработку данных пользователей, управление заявками, событиями в календаре и информацией о доступных сервисах. Для безопасности данных используются защищенные соединения HTTPS.

Клиентская часть системы представляет собой веб-приложение, созданное с использованием JavaScript и библиотеки Bootstrap. Этот подход позволяет разработать интуитивно понятный и адаптивный интерфейс, поддерживающий работу на устройствах с различными размерами экранов. Все взаимодействие пользователя с системой происходит через веб-интерфейс, который обращается к API серверной части для выполнения запросов.

Данные системы хранятся в реляционной базе данных PostgreSQL, которая обеспечивает надежное и производительное управление информацией. Структура базы данных включает таблицы для хранения данных о пользователях, заявках, событиях, сервисах и задачах. Оптимизированные запросы и индексация ключевых полей позволяют ускорить работу с большими объемами данных и обеспечить высокую производительность системы.

Для интеграции с внешними сервисами, такими как электронная почта, реализованы механизмы подключения через стандартные протоколы (например, SMTP для отправки уведомлений). Это дает возможность информировать пользователей о статусах заявок, изменениях в задачах или приближающихся событиях, а также отправлять напоминания.

Система поддерживает модульность, что позволяет независимо разрабатывать и тестировать основные функции: управление заявками, событиями в календаре, решением технических проблем и предоставлением информации о сервисах. Такой подход облегчает сопровождение и развитие системы.

Проектирование архитектуры учитывает современные требования к надежности и производительности. Для защиты данных реализованы регулярные резервные копирования базы данных, а логирование действий пользователей позволяет отслеживать ключевые операции. В условиях увеличения нагрузки серверная архитектура позволяет гибко масштабировать систему за счет увеличения мощности серверного оборудования.

Таким образом, архитектура программного средства полностью соответствует функциональным и нефункциональным требованиям, обеспечивая стабильную и удобную работу системы в рамках автоматизации бизнес-процессов.

## **3.2 Проектирование и разработка алгоритма программного средства**

В данном разделе подробно рассматривается процесс создания алгоритма для автоматизации бизнес-процессов информационного отдела организации. На первом этапе была проведена тщательная аналитика текущих рабочих процессов, чтобы выявить их слабые места и определить этапы, которые подлежат автоматизации. Результатом аналитического исследования стала разработка детализированной блок-схемы, которая отражает последовательность действий, альтернативные сценарии и ключевые решения, необходимые для успешного выполнения целей автоматизации.

Алгоритм начинает свою работу с обработки входных данных, поступающих из различных источников. Эти данные проходят обязательный этап валидации для проверки их корректности и соответствия установленным требованиям системы. Валидация необходима для предотвращения ошибок и обеспечения целостности данных на всех этапах работы программы.[7]

После успешного входа в систему пользователю предоставляется возможность выбора одного из доступных разделов, каждый из которых решает конкретную задачу. В случае выбора раздела "Сервисы", пользователь может просмотреть и обновить список доступных для него сервисов, которые уже заранее добавлены администратором. Этот процесс позволяет каждому пользователю индивидуализировать интерфейс и отобразить только те инструменты, которые действительно нужны для его работы.

Если пользователь выбирает раздел "Календарь", ему предоставляется возможность планировать и создавать события, такие как совещания, встречи или видеоконференции. При создании события пользователь указывает название, место проведения и время. После добавления событие сохраняется в календаре и становится доступным для просмотра и редактирования при необходимости.

Выбор раздела "Доступы" инициирует процесс подачи заявок на получение доступа к определённым сервисам или системам. Пользователь формирует заявку, указывая необходимый сервис, после чего она передаётся администратору. Администратор, в свою очередь, проверяет полномочия пользователя и принимает решение о предоставлении или отказе в доступе. Этот этап позволяет чётко регламентировать предоставление прав доступа и контролировать безопасность корпоративных систем.

При работе с разделом "Проблемы", пользователь имеет возможность сообщить о возникшей проблеме или неисправности. Для этого он создаёт заявку, где указывает заголовок, описание проблемы и категорию. После отправки заявки администратор приступает к её решению. Когда проблема устраняется, статус заявки обновляется, позволяя пользователю отслеживать её выполнение. В случае нарушения сроков выполнения заявки, система фиксирует это для дальнейшего анализа эффективности работы администрации.

Таким образом, разработанный алгоритм охватывает ключевые задачи, с которыми сталкиваются пользователи и администраторы информационного отдела. Его структура позволяет автоматизировать рутинные процессы, такие как управление сервисами, планирование событий, обработка заявок на доступ и решение проблем. Алгоритм работы программного средства представлен на рисунке 3.1.



Рисунок 3.1 – Алгоритм работы программного средства

## **Разработка диаграммы классов**

Диаграмма классов системы автоматизации бизнес-процессов отображает основные сущности, их атрибуты, методы и взаимодействие между ними, что позволяет эффективно организовать управление пользователями, сервисами, календарем, доступами, проблемами и учетом оборудования. Основные классы включают User, Admin, Service, CalendarEvent, AccessRequest, Issue, и Equipment, каждый из которых отражает ключевые аспекты бизнес-логики системы.

Класс User представляет пользователей системы, храня информацию, такую как имя, фамилия, отдел, электронная почта и пароль. Пользователь может выполнять регистрацию и вход в систему, что позволяет ему получить доступ к функциональности. Этот класс связан с несколькими другими сущностями: AccessRequest (для отправки заявок на доступ), Issue (для отправки проблем), CalendarEvent (для планирования событий) и Service (для выбора доступных сервисов).

Класс Admin, наследующий свойства и методы от User, добавляет возможности управления: администратор может создавать новые сервисы, управлять календарными событиями пользователей, обрабатывать заявки на доступы, а также управлять проблемами и учетными записями оборудования. Это делает Admin центральной фигурой в системе, ответственной за координацию всех процессов.

Класс Service предоставляет информацию о доступных в системе сервисах, включая их название и ссылку. Пользователи могут выбирать сервисы для отображения на своей странице, а администраторы добавлять или редактировать эти сервисы.

Класс CalendarEvent позволяет пользователям и администраторам создавать события с указанием названия, времени и места проведения. События связаны с пользователями, которые их создали, обеспечивая возможность персонализации.

Класс AccessRequest описывает заявки на доступ, отправленные пользователями. Каждая заявка включает название сервиса, дату подачи и статус (например, "Ожидает", "Выполнено"). Этот класс связан с User (отправитель заявки) и обрабатывается Admin, который определяет права доступа.

Класс Issue представляет заявки на устранение проблем, включающие заголовок, описание, категорию, статус и время подачи. Эти заявки также связаны с User, который их отправляет, и Admin, который обновляет статус (например, "В процессе", "Завершено").

Класс Equipment отвечает за учет оборудования в организации. Он хранит данные о типе, марке, модели, инвентарном номере, местоположении (кабинете) и ответственном лице. Администратор управляет этим классом, добавляя и редактируя записи.

Диаграмма классов демонстрирует связи между сущностями: наследование между Admin и User, ассоциации между User и такими классами, как Service, CalendarEvent, AccessRequest и Issue, а также зависимости между Admin и всеми процессами системы. Эта структура обеспечивает централизованное управление данными и удобство для пользователей и администраторов.

В контексте данного проекта диаграмма классов играет ключевую роль в проектировании базы данных и архитектуры программного обеспечения, а также обеспечивает четкое представление о взаимодействии объектов внутри системы.

Диаграмма классов показывает, какие классы существуют в системе и как они взаимодействуют друг с другом. Она позволяет структурировать такие сущности, как пользователи, заявки, сервисы, события и проблемы. На основе этой диаграммы можно спроектировать и нормализовать структуру базы данных, которая уже включает таблицы users, services, user\_services, calendar\_events, access\_requests и equipment. Каждая таблица в базе данных соответствует классу, который содержит атрибуты (поля таблицы) и методы, обеспечивающие выполнение необходимых операций с данными. Это позволяет чётко описать, какие действия можно выполнять с каждой сущностью и как они взаимодействуют друг с другом в рамках системы.

Методы классов определяют операции, которые можно выполнять с данными. Например, класс "Заявка на доступ" может иметь методы для создания, обновления статуса или отмены заявки, а класс "Календарь" включает методы для добавления или удаления событий. Такое представление помогает описать функционал системы на уровне кода и уточнить, какие процессы и бизнес-логика будут реализованы на этапе разработки. Разработка на React для фронтенда и PostgreSQL для базы данных требует логичной и структурированной модели данных, которая заранее проектируется с помощью диаграммы классов. Это помогает эффективно организовать код в JavaScript, взаимодействие компонентов интерфейса с базой данных и управление данными на клиентской стороне.

Диаграмма классов также служит инструментом для оптимизации кода и структуры программы. Она помогает продумать, как организованы объекты системы и их взаимодействия, что позволяет сократить дублирование кода и обеспечить его читаемость и расширяемость. Разработка такой диаграммы упрощает интеграцию всех компонентов системы: интерфейса, базы данных и бизнес-логики, а также ускоряет процесс внесения изменений или добавления нового функционала в будущем.

В диаграмме классов отображаются связи между сущностями, которые можно представить через различные виды отношений. Ассоциации показывают взаимодействие между классами, например, пользователь отправляет заявки на доступ или запросы на устранение проблем. Композиция указывает на более тесную связь, когда один класс неразрывно включает в себя другой, как в случае событий внутри календаря. Агрегация позволяет отразить отношение, когда одна сущность содержит в себе другие, но при этом они могут существовать независимо, как администратор управляет пользователями, сервисами и заявками. Наследование позволяет описать иерархическую структуру классов, когда администратор является расширением пользователя с дополнительными правами и методами.

Диаграмма классов выполняет роль документации для разработчиков, показывая структуру и логику системы в наглядной форме. Она упрощает коммуникацию в команде и помогает всем участникам проекта понимать, как устроена система на уровне данных и объектов. Благодаря диаграмме классов, процесс разработки становится более прозрачным и контролируемым, что особенно важно для реализации проекта автоматизации бизнес-процессов информационного отдела.



Рисунок 2.5 – Диаграмма классов

## **Разработка диаграммы развертывания**

Диаграмма развертывания (Deployment Diagram) предназначена для наглядного представления физического размещения компонентов системы на различных узлах, таких как клиентские рабочие станции, серверы приложений, базы данных и другие ресурсы. Она демонстрирует, как программные компоненты системы распределены по физическим узлам, как они взаимодействуют друг с другом и как обеспечивается передача данных. В системе автоматизации бизнес-процессов информационного отдела диаграмма развертывания отражает архитектуру проекта, описывает размещение компонентов и их связи для достижения требуемого функционала.

В проекте основными узлами развертывания являются клиентская рабочая станция, сервер приложений, сервер базы данных и почтовый сервер.

Клиентская рабочая станция представляет собой устройство пользователя, на котором открыт веб-браузер для взаимодействия с системой. Веб-интерфейс системы разработан с использованием React и Bootstrap, что обеспечивает удобство работы, современный дизайн и адаптивность интерфейса под разные разрешения экранов. С помощью веб-интерфейса пользователи выполняют задачи, такие как отправка заявок на доступ, создание событий в календаре, выбор отображаемых сервисов и отправка запросов на устранение проблем. Администраторы используют тот же интерфейс для управления сервисами, доступами, календарём, пользователями и учётом компьютерного оборудования. Клиентская рабочая станция отправляет HTTP-запросы на сервер приложений, получая обратно результаты в виде обновлённых страниц или данных.

Сервер приложений является центральным узлом, на котором развернуты все компоненты бизнес-логики системы. Этот сервер отвечает за обработку всех входящих HTTP-запросов от клиентских рабочих станций. Сервер обрабатывает операции, такие как регистрация и аутентификация пользователей, создание и обновление заявок, планирование событий в календаре, обновление статусов проблем, управление данными о сервисах и учёте оборудования. Вся бизнес-логика реализована в виде серверного приложения, написанного на Python с использованием веб-фреймворка Django. Сервер также генерирует и отправляет уведомления пользователям через подключённый почтовый сервер, чтобы оперативно информировать о статусах заявок, событиях в календаре или других изменениях.

На сервере приложений размещён механизм взаимодействия с базой данных. При поступлении запроса на получение или сохранение информации сервер отправляет SQL-запросы на сервер базы данных, где хранятся все необходимые данные для функционирования системы. Между сервером приложений и сервером базы данных используется защищённый канал передачи данных для обеспечения безопасности информации.

Сервер базы данных представляет собой отдельный узел, на котором развернута база данных PostgreSQL. База данных содержит все таблицы и данные, используемые в системе, включая информацию о пользователях, заявках, сервисах, календарных событиях, проблемах и учёте оборудования. Сервер приложений взаимодействует с базой данных для выполнения операций записи, чтения, обновления и удаления данных. Структура базы данных спроектирована таким образом, чтобы обеспечить целостность, надёжность и производительность системы даже при большом количестве пользователей и запросов.

Почтовый сервер обеспечивает формирование и отправку уведомлений пользователям и администраторам системы. При изменении статуса заявки, добавлении нового события в календарь или возникновении других важных событий сервер приложений генерирует уведомления и передаёт их почтовому серверу. Почтовый сервер отправляет письма на электронные адреса пользователей, указывая необходимую информацию о статусах и обновлениях. Это позволяет сотрудникам и администраторам оперативно получать актуальную информацию, что повышает прозрачность и эффективность работы.

Таким образом, диаграмма развертывания в проекте демонстрирует, как клиентские рабочие станции взаимодействуют с сервером приложений по HTTP-протоколу, а сервер приложений обрабатывает запросы, взаимодействует с базой данных для хранения и получения информации, а также с почтовым сервером для отправки уведомлений. Разделение системы на клиентскую часть, сервер приложений и сервер базы данных обеспечивает масштабируемость системы, надёжное хранение данных и быструю обработку запросов. Такое распределение компонентов позволяет эффективно организовать работу информационного отдела, обеспечивая стабильное выполнение задач, минимизацию рутинных операций и повышение продуктивности сотрудников. Диаграмма развертывания показана на рисунке 2.4



Рисунок 2.4 – Диаграмма развертывания

## **3.****5 Проектирование и разработка базы данных**

Для успешной автоматизации бизнес-процессов информационного отдела необходимо определить, как данные будут структурированы и взаимодействовать в рамках системы, а также какие требования к системе предъявляются с точки зрения функциональности и производительности.

Информационная модель базы данных определяет структуру хранения данных в системе. Для данного проекта, который включает автоматизацию бизнес-процессов информационного отдела, разрабатываемая база данных будет поддерживать управление задачами, видеоконференциями, заявками на доступы и проблемы, а также предоставлять информацию о доступных сервисах.

Таблица Users хранит информацию о пользователях системы, которые взаимодействуют с программой. Каждая запись в таблице представляет собой одного пользователя и содержит его уникальный идентификатор (id), имя и фамилию, адрес электронной почты, пароль для входа и роль пользователя (например, администратор, сотрудник или гость). Также хранится дата регистрации пользователя в системе. Эти данные необходимы для идентификации пользователей, управления их доступом и правами в системе.

Таблица 2.1 – Таблица «Users»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| Full\_name | VARCHAR | ФИО пользователя |
| Email | VARCHAR | Почта |
| Password | VARCHAR | Пароль |
| Is\_admin | VARCHAR | Проверка на права администратора |
| Role\_id | INT, FOREIGN KEY references Roles(id) | Внешний ключ |
| Department\_id | INT, FOREIGN KEY references Departments(id) | Внешний ключ |

Таблица Roles отвечает за управление ролями пользователей. Каждая запись в этой таблице содержит уникальный идентификатор роли, название роли и её описание. Система позволяет назначать пользователям различные роли, такие как администратор, который управляет всеми функциями, или гость с ограниченным доступом. Это позволяет организовать иерархию прав доступа и контролировать, какие действия могут выполнять пользователи в зависимости от их роли.

Таблица 2.2 – Таблица «Roles»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| Role\_name | VARCHAR | Название роли |
| Description | TEXT | Описание |

AccessRequests содержит информацию о заявках на доступ к различным корпоративным системам. В каждой записи хранится информация о пользователе, который отправил заявку, название системы, тип запрашиваемого доступа (например, чтение или запись), статус заявки (ожидается, одобрена или отклонена) и дата подачи заявки. Эта таблица позволяет отслеживать и управлять процессом предоставления доступа к системам, улучшая контроль и безопасность.

Таблица 2.3 – Таблица «AccessRequests»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| User\_id | INT, FOREIGN KEY references Users(id) | Внешний ключ |
| Service\_id | INT, FOREIGN KEY references Services(id) | Внешний ключ |
| Request\_date | TIMESTAMP | Дата представления |
| Status | VARCHAR | Статус запроса |
| Permissions | VARCHAR | Права, выданные пользователю для доступов к сервисам |

Problems хранит данные о заявках пользователей на устранение проблем с системами или оборудованием. Каждая заявка включает описание проблемы, её приоритет, статус (например, в работе или решено) и ответственного сотрудника, который назначен для её решения.

Таблица 2.4 – Таблица «Problems»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| User\_id | INT, FOREIGN KEY references Users(id) | Внешний ключ |
| Title | VARCHAR | Заголовок |
| Description | VARCHAR | Описание проблемы |
| Category | VARCHAR | Категория проблемы |
| Status | VARCHAR | Статус исполнения |
| Submitted\_at | VARCHAR | Дата отправки |

Таблица Services используется для Эта таблица играет ключевую роль в организации рабочего процесса и контроле за выполнением задач.

Таблица 2.5 – Таблица «Services»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| Name | VARCHAR | Название сервиса |
| Url | TEXT | Ссылка на сервис |

Таблица Equipment предназначена для учета и управления компьютерной техникой и другим оборудованием в организации. В каждой записи хранится информация о конкретном оборудовании, включая его тип, марку, модель, инвентарный номер, место нахождения и ответственное лицо, а также дополнительные данные, необходимые для контроля и обслуживания оборудования. Таблица Equipment позволяет систематизировать учет оборудования, связывать его с ответственными пользователями и отслеживать всю необходимую информацию для эффективного управления техническими ресурсами организации.

Таблица 2.6 – Таблица «Equipment»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| Type | VARCHAR | Тип оборудования |
| Brand | VARCHAR | Бренд компьютера |
| Model | VARCHAR | Модель комьютера |
| Inventory Number | VARCHAR | Инвентарный номер |
| Location | VARCHAR | Место нахождения или кабинет |
| Responsible\_person | INT, FOREIGN KEY references Users(id) | Лицо ответственное за технику |

Таблица Conferences отвечает за хранение информации о видеоконференциях, которые планируются и проводятся в рамках системы. Записи содержат данные о дате и времени конференции, её организаторе, теме и ссылке на встречу. Это позволяет пользователям планировать и участвовать в видеоконференциях, а также получать напоминания о предстоящих событиях.

Таблица 2.7 – Таблица «Departments»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| Name | VARCHAR | Название отдела |
| Description | Text | Описание |

Таблица Admin\_actions предназначена для фиксации действий администраторов в системе. В этой таблице хранятся записи обо всех значимых операциях, выполняемых администраторами, включая изменения данных пользователей, управление заявками, оборудованием и сервисами. Это позволяет вести аудит действий администраторов, отслеживать изменения. Таблица Admin\_actions играет ключевую роль в обеспечении аудита и прозрачности работы администраторов. Она фиксирует все действия, выполняемые в системе, связывает их с конкретными пользователями и сущностями, что позволяет эффективно контролировать и анализировать изменения.

Таблица 2.8 – Таблица «Admin\_actions»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| Admin\_id | INT, FOREIGN KEY references Users(id) | Внешний ключ |
| Action\_type | VARCHAR | Тип события |
| Action\_details | VARCHAR | Детали события |
| Timestamp | TIMESTAMP | Время |

Таблица Calendar\_events предназначена для хранения событий в календаре, которые создаются пользователями или администраторами. В каждой записи содержатся детали мероприятия, включая название события, дату и время проведения, место и организатора.

Таблица 2.9 – Таблица «Calendar\_events»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| User\_id | INT, FOREIGN KEY references Users(id) | Внешний ключ |
| Title | TEXT | Описание сервиса |
| Event\_time | VARCHAR | Время события |
| location | VARCHAR | Место проведения |

Таблица User\_services предназначена для управления связями между пользователями и сервисами, которые они используют. Она хранит информацию о том, какие сервисы доступны каждому пользователю.

Таблица 2.10 – Таблица «User\_services»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| Id | INT | Идентификатор, первичный ключ |
| User\_id | INT, FOREIGN KEY references User(id) | Внешний ключ |
| Service\_id | INT, FOREIGN KEY references Services(id) | Внешний ключ |

**3.6 Управление процессом разработки с помощью контроля версий**

Управление процессом разработки является ключевым аспектом успешной реализации любого программного обеспечения. В рамках данного проекта мы применяем систему контроля версий (СКВ), чтобы обеспечить надежное и эффективное управление кодовой базой, а также совместную работу команды разработчиков. Основная цель использования контроля версий заключается в том, чтобы отслеживать изменения в коде, упростить процесс совместной работы и обеспечить возможность восстановления предыдущих версий.

Для нашего проекта была выбрана система контроля версий Git, которая является одной из наиболее популярных и мощных систем, предоставляющих широкий набор возможностей для управления проектами. Git поддерживает распределенный подход, что позволяет каждому разработчику иметь полную копию репозитория на своем локальном компьютере, обеспечивая тем самым высокую скорость работы и независимость от центрального сервера.[8]

Использование Git имеет множество преимуществ. Во-первых, система позволяет отслеживать все изменения в коде, что помогает разработчикам видеть историю изменений и понимать, кто и когда вносил правки. Во-вторых, Git поддерживает возможность создания веток, что позволяет разработчикам работать над новыми функциями или исправлениями в изолированном окружении, не нарушая основную кодовую базу. После завершения работы ветка может быть слита обратно в основную ветку, что минимизирует риск возникновения конфликтов. В-третьих, Git облегчает совместную работу команды, позволяя разработчикам одновременно работать над одним и тем же проектом, а изменения синхронизировать через удаленные репозитории, такие как GitHub или GitLab. Это обеспечивает удобный доступ ко всем изменениям. Наконец, каждое изменение в коде хранится в репозитории, что позволяет восстанавливать предыдущие версии в случае необходимости и обеспечивает защиту от потери данных.

В нашем проекте процесс разработки с использованием Git будет организован следующим образом: сначала создается новый репозиторий Git на платформе GitHub, где будет размещен код проекта. Каждый разработчик клонирует этот репозиторий на свою локальную машину для работы. После внесения изменений разработчики будут делать коммиты с информативными сообщениями, описывающими суть изменений. Это позволит другим членам команды легко понимать, какие изменения были внесены. После завершения работы над задачей разработчик отправляет свою ветку в удаленный репозиторий и создает Pull Request (PR), который позволяет остальным участникам команды просмотреть изменения, оставить комментарии и предложить улучшения. После одобрения PR ветка будет слита с основной.

Для предотвращения конфликтов и упрощения процесса слияния, разработчики будут регулярно обновлять свои ветки, сливая изменения из основной ветки. Для работы с Git будут использоваться инструменты, такие как Git Bash для выполнения команд Git. Кроме того, интеграция с IDE, такими как Visual Studio Code, будет поддерживать встроенные функции работы с Git.

Для успешной работы с системой контроля версий команда разработчиков будет обеспечена документацией и обучающими материалами по использованию Git. Это будет включать основные команды и их применение, руководства по работе с ветками и слияниями, а также примеры типичных рабочих процессов в Git.

Использование системы контроля версий Git в нашем проекте позволит эффективно управлять процессом разработки, обеспечить высокое качество кода и минимизировать риски, связанные с ошибками. Это создаст удобные условия для работы команды, повысит продуктивность и упростит взаимодействие между разработчиками.

Каждый раз, когда в основной ветке происходит изменение, CI/CD-процессы могут автоматически запускать тесты, проверяя, что новый код не нарушает существующую функциональность. Это не только экономит время, но и позволяет команде быстро обнаруживать и устранять проблемы, что в конечном итоге повышает надежность и стабильность приложения.

Важно также упомянуть о потенциальных рисках, связанных с использованием системы контроля версий. Например, неправильно выполненное слияние может привести к конфликтам в коде, которые могут быть трудными для разрешения. Поэтому важно регулярно проводить код-ревью и обсуждать принципы работы с ветками. Создание четких правил и стандартов по использованию Git поможет минимизировать риски и повысить качество командной работы.

В заключение, интеграция системы контроля версий Git в процесс разработки нашего проекта не только обеспечит надежное управление кодом, но и создаст устойчивую и эффективную командную среду. Это, в свою очередь, будет способствовать успешной реализации проекта и достижению поставленных целей, что является критически важным в условиях современного быстро меняющегося мира разработки программного обеспечения.

1. **ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА**

## **Выбор и обоснование видов тестирования**

Тестирование программного средства – это ключевой этап разработки, позволяющий убедиться в его работоспособности, надежности и соответствии заданным требованиям. В рамках данного проекта для автоматизации бизнес-процессов информационного отдела были выбраны наиболее подходящие виды тестирования, основываясь на характеристиках приложения, предполагаемых условиях эксплуатации и основных рисках, связанных с его использованием.

В первую очередь было применено функциональное тестирование. Этот вид тестирования направлен на проверку правильности выполнения всех функций приложения: создания и управления задачами, отправки запросов на доступ, обработки сообщений о проблемах, организации видеоконференций и работы с базой данных. Особое внимание уделялось корректности реализации заявленных функций, так как их надежность напрямую влияет на удобство работы пользователей.

Тестирование методом белого ящика использовалось для проверки внутренней логики работы ключевых функций системы. Были проанализированы алгоритмы обработки данных в календаре, формирования запросов к базе данных, а также процедуры отправки уведомлений. Это позволило убедиться, что код работает корректно, алгоритмы оптимизированы, а ошибки в логике отсутствуют.

Тестирование методом черного ящика проводилось с целью проверки функциональности системы без учета ее внутренней структуры. Пользовательские сценарии, включающие создание задач, отправку запросов и настройку видеоконференций, проверялись исключительно на основе входных данных и ожидаемого результата. Такой подход позволил оценить удобство использования интерфейса и соответствие функций требованиям.

Кроме того, тестирование пользовательского интерфейса (UI) играло важную роль. Поскольку система предназначена для сотрудников, обладающих различными уровнями компьютерной грамотности, проверка эргономичности интерфейса и интуитивной навигации стала приоритетной задачей. Это позволило выявить и устранить потенциальные неудобства в использовании приложения, чтобы минимизировать время на обучение персонала работе с системой.

Для обеспечения стабильности работы приложения в реальных условиях было проведено тестирование производительности. Оно включало моделирование нагрузки при большом количестве одновременных пользователей, чтобы убедиться, что система сохраняет высокую скорость отклика и стабильную работу даже при пиковых значениях запросов.

Интеграционное тестирование использовалось для проверки корректного взаимодействия всех компонентов приложения. В рамках данного проекта особое внимание уделялось взаимодействию с базой данных, обеспечивающей хранение информации о задачах, запросах и событиях, а также с внешними сервисами для отправки уведомлений и организации видеоконференций.[9]

Безопасность системы также находилась в центре внимания. Для этого проводились тесты, направленные на проверку правильности реализации механизмов аутентификации и авторизации, а также защиты данных от несанкционированного доступа. Проверялась безопасность работы с конфиденциальной информацией, особенно в части передачи данных между клиентом и сервером.

Выбор этих видов тестирования был обусловлен сложностью и функциональным наполнением приложения, а также необходимостью обеспечения его надежности и удобства в реальной эксплуатации. Такой подход позволил систематически выявлять и устранять недостатки на всех этапах проверки, обеспечивая соответствие системы заявленным требованиям.

## **Результаты тестирования**

В ходе тестирования программного средства, автоматизирующего бизнес-процессы информационного отдела, проводилась проверка ключевых функций и системных характеристик. Основной целью тестирования было удостовериться в соответствии работы приложения функциональным и нефункциональным требованиям, заявленным на этапе проектирования.

Метод белого ящика помог обнаружить проблемы в логике обработки временных данных для задач календаря и алгоритме массовой отправки уведомлений. В частности, некорректное отображение времени было связано с неправильной обработкой временных зон. Анализ исходного кода позволил быстро найти и исправить причину ошибки.

Метод черного ящика выявил недочеты, которые могли быть неочевидны на уровне кода, но заметны пользователям. Например, тестирование интерфейса показало, что при большом объеме данных интерфейс мог работать с задержками. Это позволило провести доработки, которые улучшили пользовательский опыт.

Все тесты выполнялись по заранее составленным сценариям, включающим описание тестовых случаев, ожидаемых результатов и критериев успешного выполнения. Основные направления тестирования охватывали функциональность, производительность, стабильность, безопасность и удобство использования.

Тестирование проводилось в среде, максимально приближенной к предполагаемым условиям эксплуатации. Для тестирования использовались как единичные запросы, так и сценарии с массовым одновременным доступом. Тестовые случаи и результаты приведены в таблице 4.1.

Таблица 4.1 – Таблица результатов тестирования программы

|  |  |  |
| --- | --- | --- |
| Название | Тестовый случай пройден? | Комментарии |
| 1 | 2 | 3 |
| Тест №1: Проверка создания задачи в календаре | Да | Пользователь успешно создает задачу, она отображается корректно в календаре. |
| Тест №2: Тестирование отправки запроса на доступ | Да | Пользователь успешно создает задачу, она отображается корректно в календаре. |
| Тест №3: Организация видеоконференции | Да | Видеоконференция создается, участники получают уведомления и подключаются. |
| Тест №4: Чтение и запись данных в базу | Да | Все операции с базой данных выполняются корректно, данные сохраняются и отображаются. |
| Тест №5: Тестирование под нагрузкой | Да | Приложение сохраняет работоспособность при 500 одновременных запросах. |
| Тест №6: Тестирование авторизации пользователей | Да | Доступ предоставляется только авторизованным пользователям. |
| Тест №7: Тестирование интерфейса при нестандартных разрешениях экрана | Да | Интерфейс адаптируется под различные разрешения без нарушений в отображении. |

В процессе тестирования были выявлены несколько ошибок, которые могли повлиять на стабильность и функциональность приложения. Одна из наиболее заметных проблем касалась обработки временных данных в календаре. При создании повторяющихся задач происходило некорректное отображение времени для пользователей, находящихся в разных часовых поясах. Это могло привести к путанице в расписании и неудобству в использовании системы. Для решения данной проблемы был доработан алгоритм синхронизации временных зон. После внесения изменений время для всех пользователей стало отображаться корректно, вне зависимости от их местоположения.

Еще одна проблема, обнаруженная во время тестирования, касалась массовой отправки уведомлений. В некоторых случаях письма не доходили до всех адресатов, что создавало риски несвоевременного информирования участников о задачах или событиях. Для устранения данной проблемы был пересмотрен алгоритм отправки сообщений: оптимизирована работа с очередями и реализована дополнительная проверка успешности доставки уведомлений. Это обеспечило надежную коммуникацию между системой и пользователями.

Также во время тестирования производительности было замечено, что при отображении большого объема данных интерфейс системы реагировал с задержками. Это могло стать критичным при интенсивной работе пользователей с базой данных. Для решения данной задачи был внедрен механизм кэширования наиболее часто запрашиваемой информации. Этот подход значительно повысил скорость отклика интерфейса и улучшил общий пользовательский опыт.

После устранения всех обнаруженных дефектов были проведены повторные тесты, которые подтвердили корректность внесенных изменений. Результаты показали, что приложение стабильно функционирует даже при высокой нагрузке и активно использует ресурсы системы для обеспечения надежной работы.

Тестирование также позволило убедиться в высокой степени безопасности программного средства. Механизмы авторизации и аутентификации были проверены на возможность несанкционированного доступа. Проверки показали, что система успешно блокирует попытки входа без наличия учетных данных, а конфиденциальные данные защищены от утечек.

Особое внимание было уделено пользовательскому интерфейсу, чтобы убедиться в его удобстве и адаптивности. Интерфейс корректно отображается на устройствах с различными разрешениями экранов и обеспечивает интуитивно понятное взаимодействие с системой. Это особенно важно для пользователей, работающих с приложением на разных устройствах.

## **Вывод тестирования**

Результаты тестирования подтверждают, что разработанное программное средство для автоматизации бизнес-процессов информационного отдела соответствует всем заявленным требованиям и демонстрирует высокую степень готовности к использованию в реальной среде.

Выводы по результатам тестирования, выполненного методами белого и черного ящика, подтверждают, что оба подхода взаимодополняемы и необходимы для обеспечения высокого качества программного средства.

Метод белого ящика позволил углубленно проанализировать внутреннюю работу системы, выявить и устранить ошибки в алгоритмах и логике. Метод черного ящика, в свою очередь, сосредоточился на тестировании функций с точки зрения конечного пользователя, помог оценить удобство использования и корректность выполнения задач.

Тем не менее, при тестировании производительности и стабильности метод черного ящика показал, что требуется дополнительная работа для оптимизации системы при высоких нагрузках. Метод белого ящика подтвердил эти выводы, указав на узкие места в реализации запросов и работе с системными ресурсами.

Оба подхода в совокупности обеспечили всестороннее тестирование программного средства, выявили его сильные стороны и области для дальнейших улучшений.

Во время тестирования функциональных возможностей системы было подтверждено, что все основные функции, включая управление задачами в календаре, отправку запросов на доступ, обработку сообщений о проблемах и организацию видеоконференций, работают корректно. Реализованные механизмы взаимодействия с базой данных обеспечивают надежное хранение, обновление и предоставление данных, что подтверждено успешным прохождением всех тестовых сценариев.

Производительность системы также находится на высоком уровне. Тесты под нагрузкой показали, что приложение сохраняет стабильность работы и оперативно обрабатывает запросы даже при значительном количестве одновременных пользователей. Это позволяет использовать программное средство в условиях интенсивной эксплуатации без риска снижения его эффективности.

Безопасность системы была протестирована с учетом сценариев, предполагающих возможные угрозы. Программа успешно справилась с задачей защиты конфиденциальных данных, реализовав эффективные механизмы аутентификации, авторизации и управления правами доступа. Эти аспекты особенно важны для обеспечения сохранности информации, обрабатываемой в информационном отделе организации.

Проведенная проверка интерфейса подтвердила его интуитивность и адаптивность. Пользовательский интерфейс корректно отображается на устройствах с различными разрешениями экранов, что делает систему удобной для работы с компьютеров и мобильных устройств. Логика интерфейса и простота выполнения основных операций получили положительные отзывы от тестировщиков, что подтверждает ориентацию на удобство пользователя.

Таким образом, программное средство прошло полный цикл тестирования, включающий функциональные, нагрузочные, интеграционные и пользовательские проверки. Обнаруженные в процессе тестирования ошибки были оперативно исправлены, а их устранение подтверждено повторным тестированием.

На основании проведенных тестов можно сделать вывод, что разработанная система полностью готова к внедрению. Она соответствует предъявляемым функциональным и техническим требованиям, отличается надежностью, безопасностью и высокой производительностью. Внедрение данного программного средства позволит существенно оптимизировать работу информационного отдела, снизить трудозатраты и повысить эффективность бизнес-процессов.

1. **РУКОВОДСТВО ПО ЭКСПЛУАТАЦИИ ПРОГРАММНОГО СРЕДСТВА**

В программном средстве автоматизации бизнес-процессов информационного отдела организации есть две группы пользователей: пользователь и администратор. Пользователю, в зависимости от его роли, предоставляется определённый набор функций и элементов интерфейса, позволяющий осуществлять те или иные действия в системе.

При входе в систему пользователь попадает на главную страницу со всеми сервисами. При нажатии на сервис пользователь перенаправляется на страницу с адресом сервиса.

Вид главной страницы программного средства с сервисами представлен на рисунке 5.1.

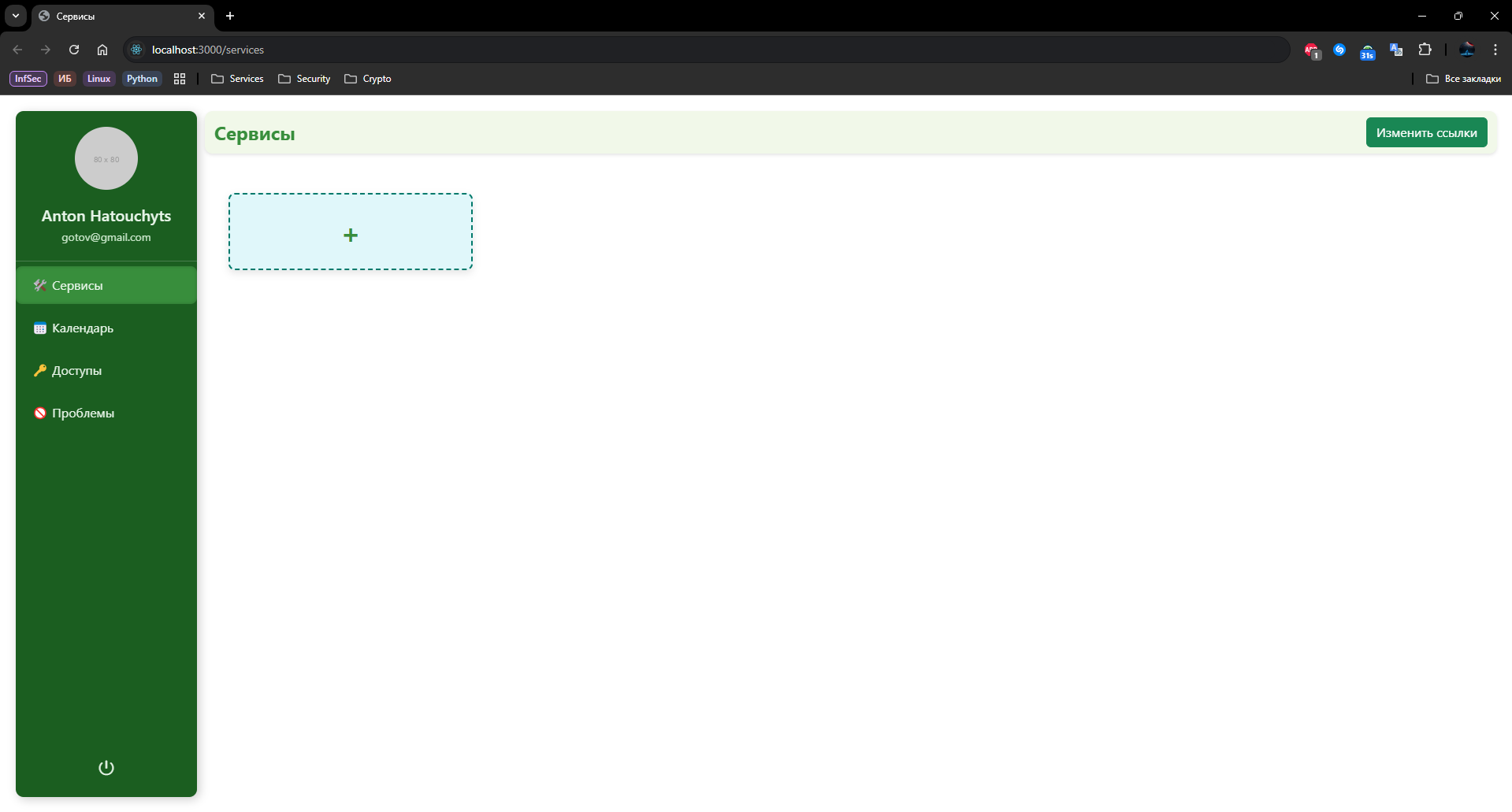


Рисунок 5.1 – Главная страница c сервисами

Чтобы добавить сервисы на главную страницу, пользователю необходимо нажать на кнопку с плюсом, после чего появиться диалоговое окно с возможность выбрать сервисы, которые пользователю необходимы. Вид диалогового окна с сервисами показан на рисунке 5.2.

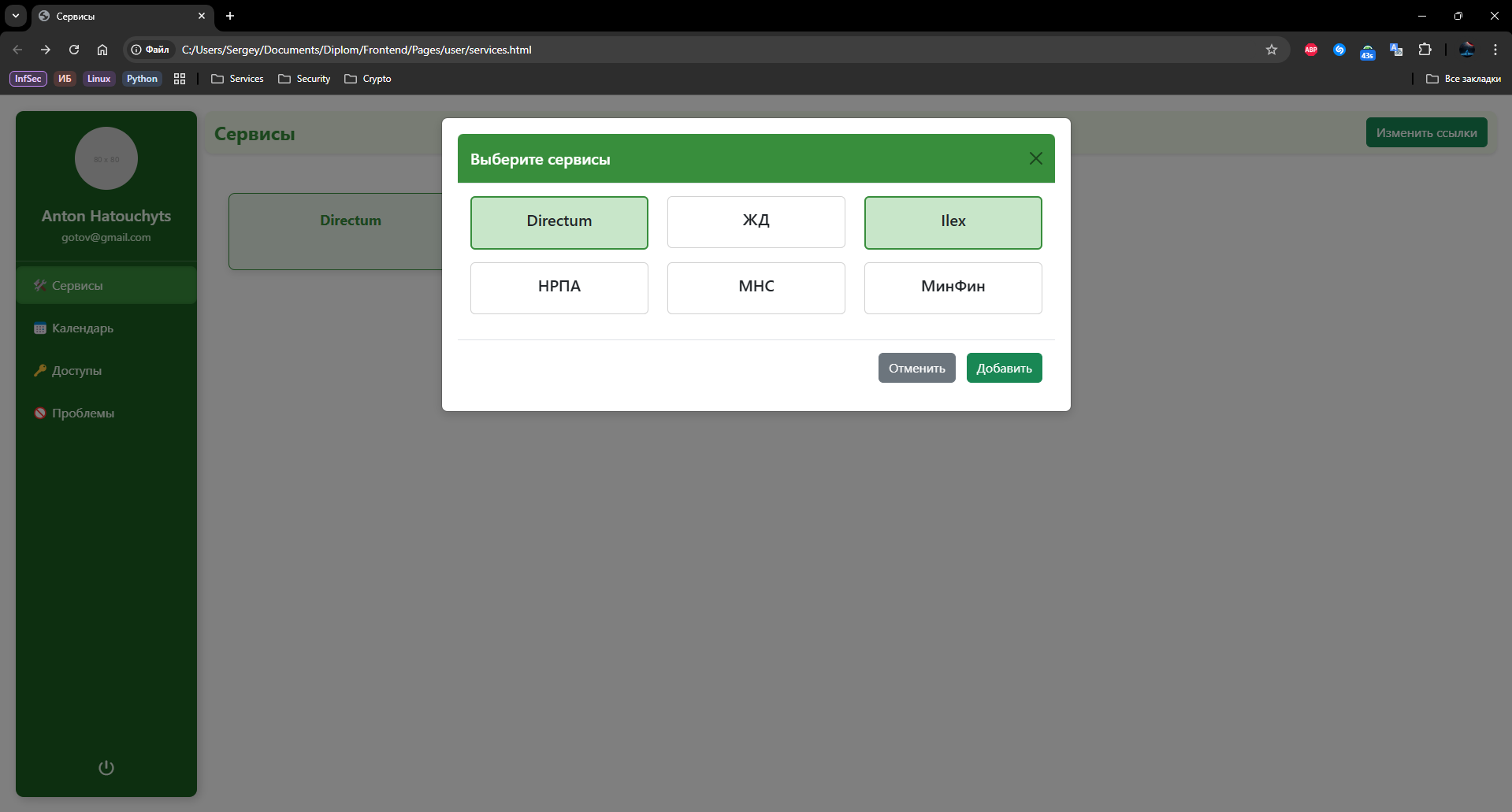


Рисунок 5.2 – Диалоговое окно с сервисами

Страница, предоставляющая пользователю доступ к календарю, представлена на рисунке 5.3.

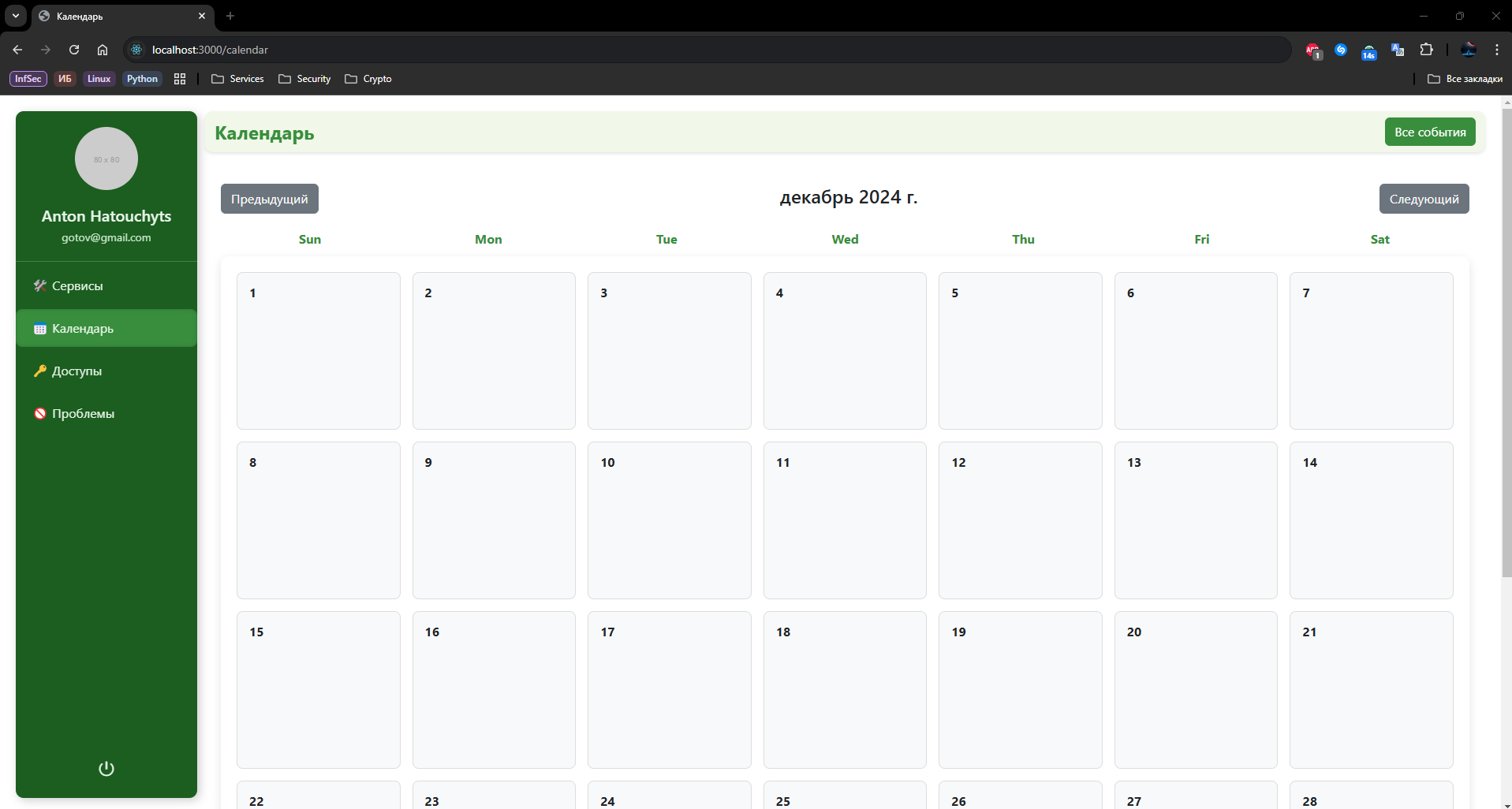


Рисунок 5.3 – Страница с календарем

Для того чтобы пользователь мог создавать событие, ему нужно нажать на плитку с нужным ему днем два раза, после чего для его появиться диалоговое окно с созданием события. Диалоговое окно, предоставляющая пользователю доступ к созданию события, представлена на рисунке 5.4.

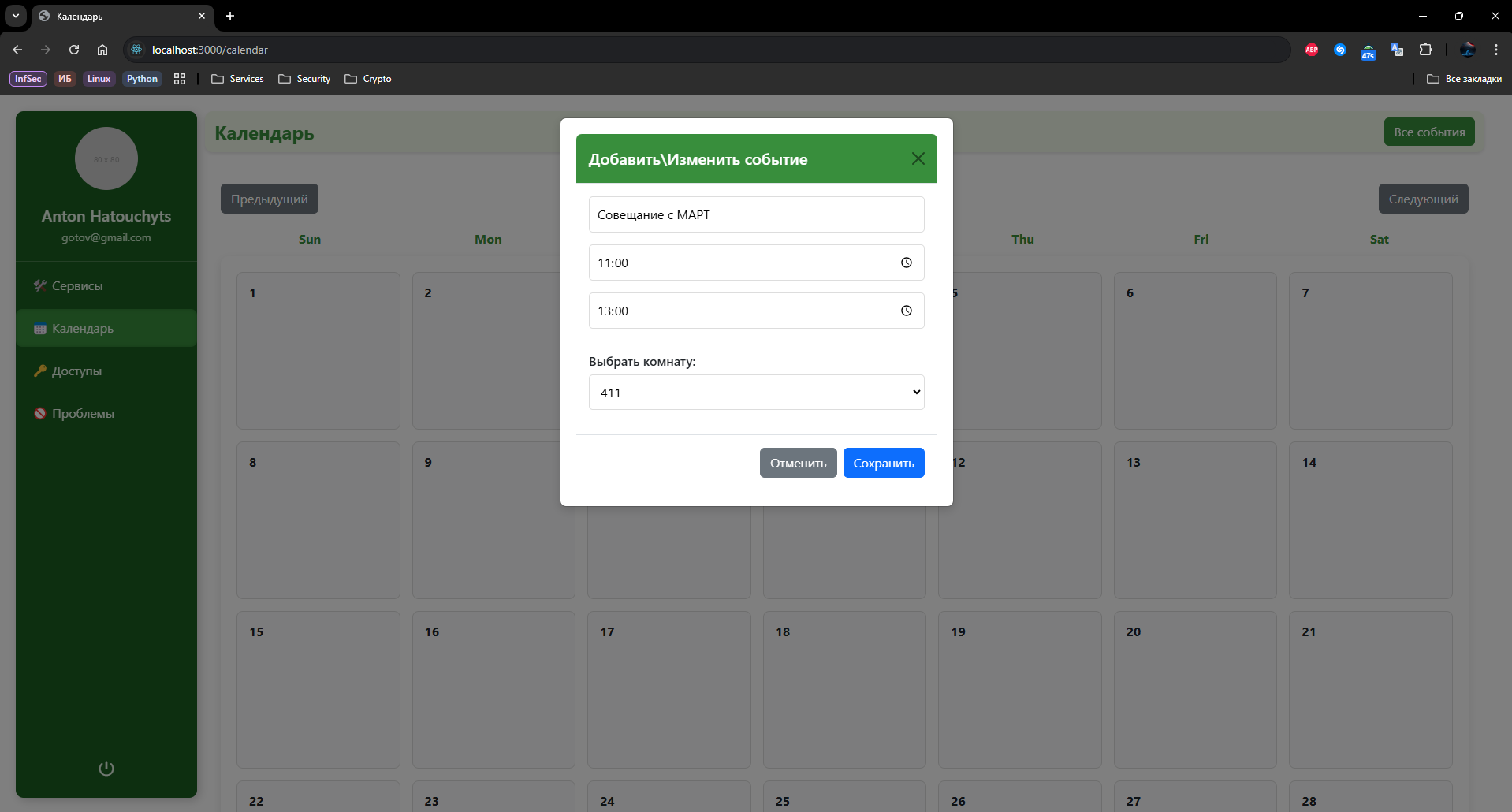


Рисунок 5.4 – Диалоговое окно с созданием события

У пользователя есть возможность отправить заявку на доступ к определенному сервису, а также распечатать заявку. Внешний вид страницы с доступами показан на рисунке 5.5.

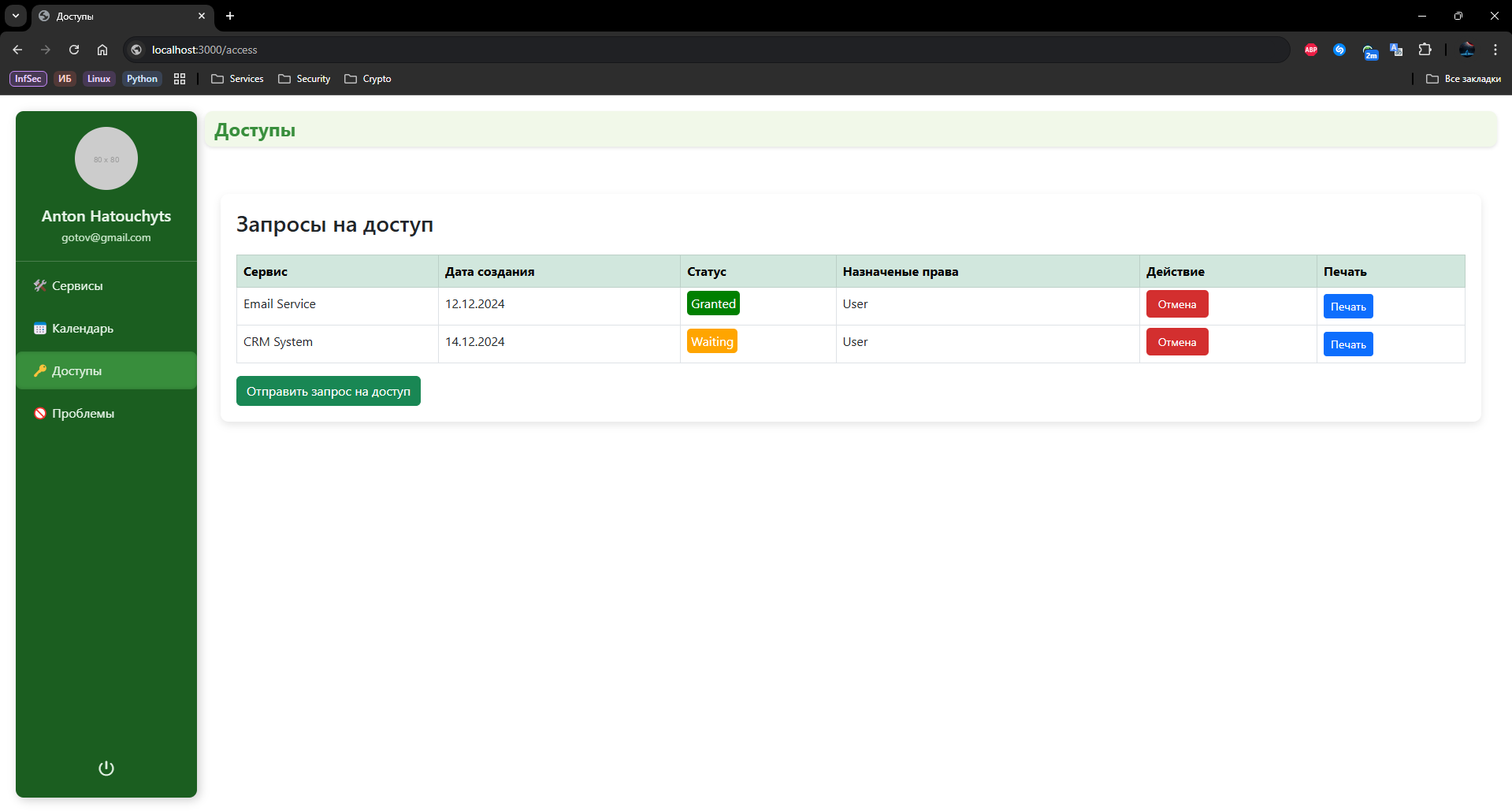


Рисунок 5.5 – Страница с запрошенными доступами пользователя

В случае возникновения у пользователя какой-либо неполадки, он может отравить заявку для получения технической поддержки администратору. Внешний вид страницы для отправки заявки представлен на рисунке 5.6.

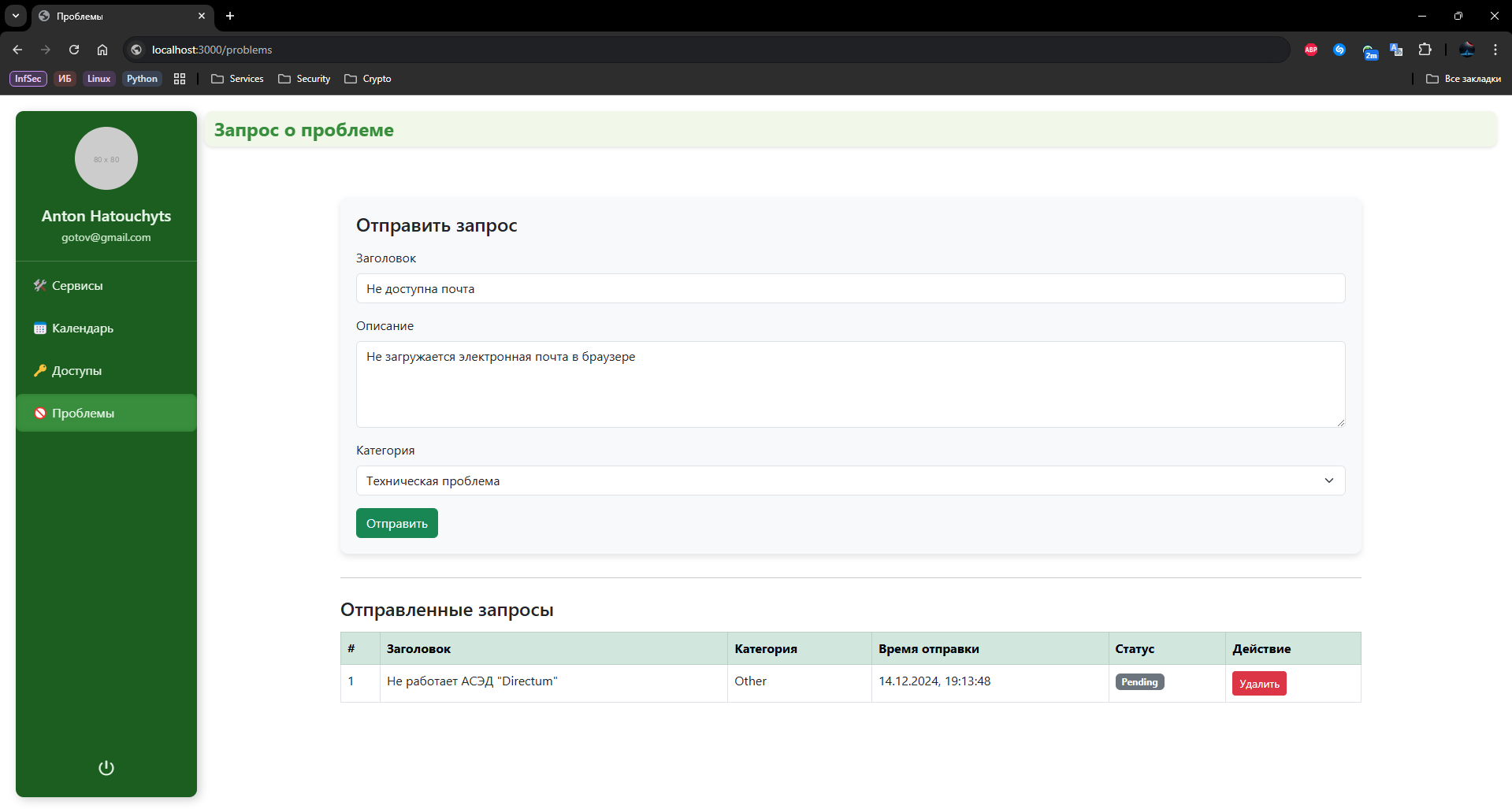


Рисунок 5.6 – Страница с запросами на устранение проблемы

Для того, чтобы иметь возможность программным средством, пользователю необходимо войти в систему.

Внешний вид окна авторизации представлен на рисунке 5.7.

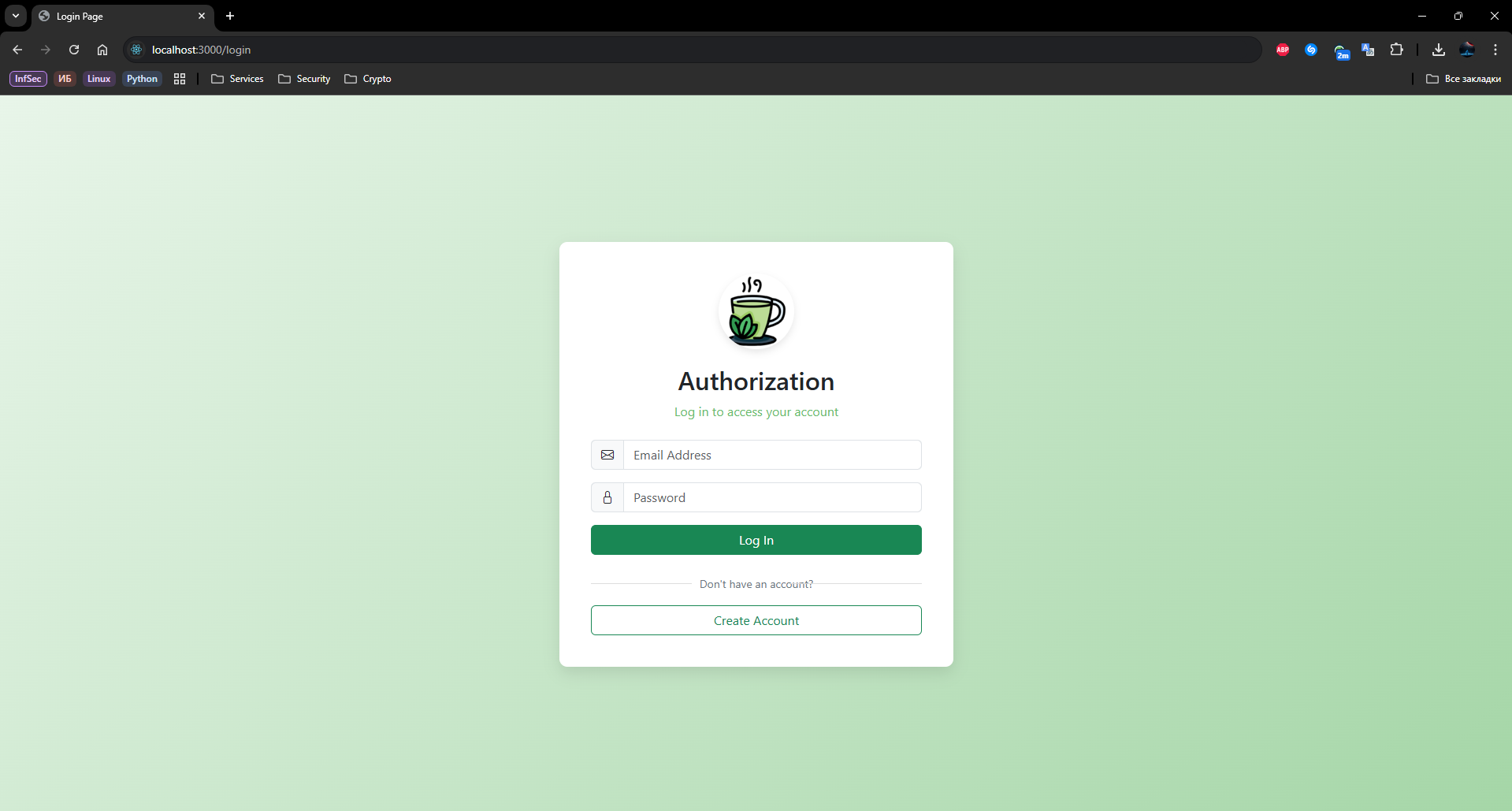


Рисунок 5.7 – Страница авторизации

При нажатии на кнопку «Войти», если логин пользователя найден в базе данных и введён правильный пароль, то пользователь будет перенаправлен на главную страницу с сервисами.

Если пользователь ещё не имеет аккаунта в системе, ему будет предложено перейти на окно регистрации и пройти её. При успешном прохождении регистрации, произойдёт тоже самое, что и при успешной авторизации.

Страница для регистрации отображена на рисунке 5.8.

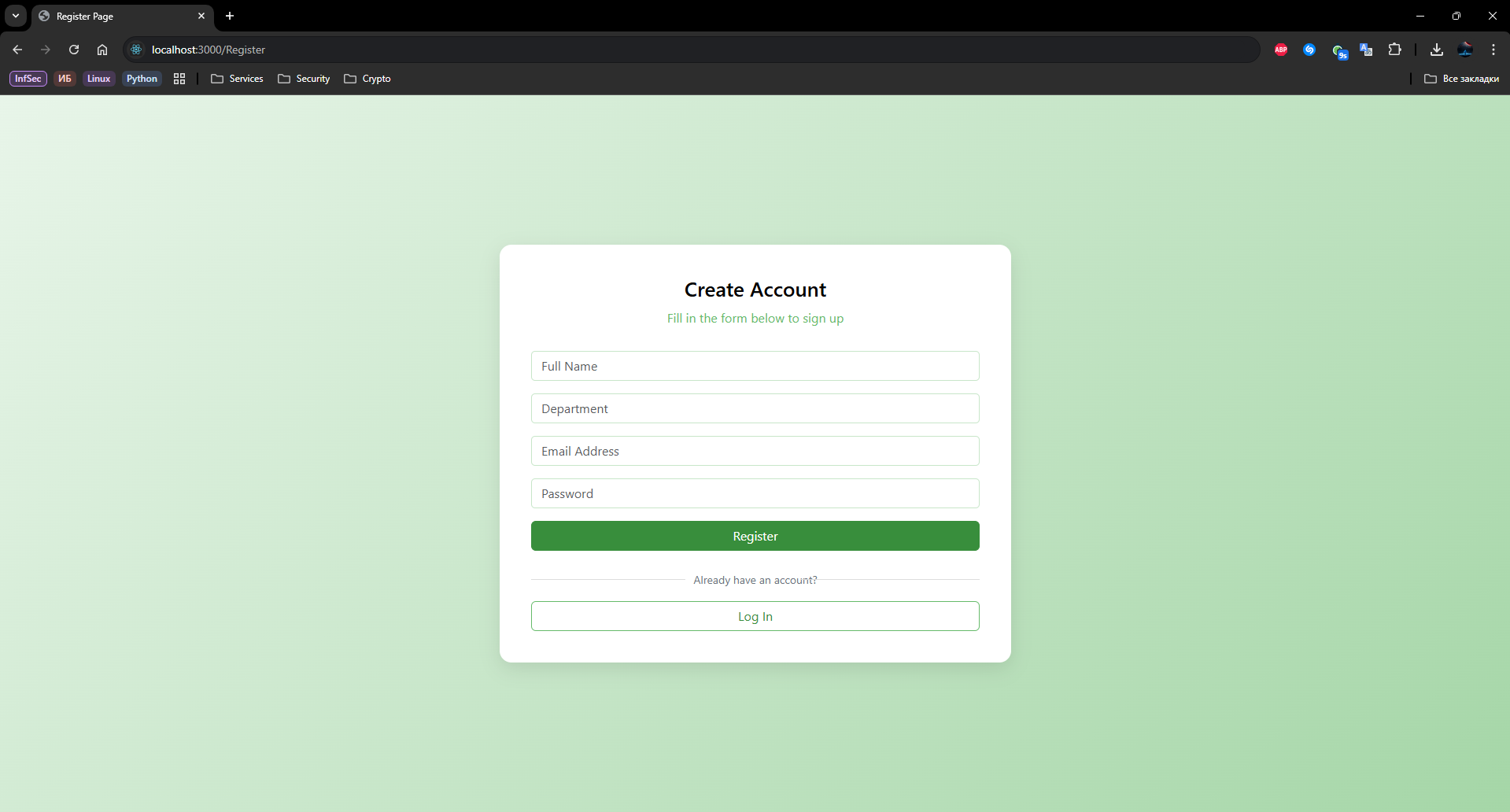


Рисунок 5.8 – Страница регистрации

Информация о всех сервисах на панели администратора показана на рисунке 5.9.

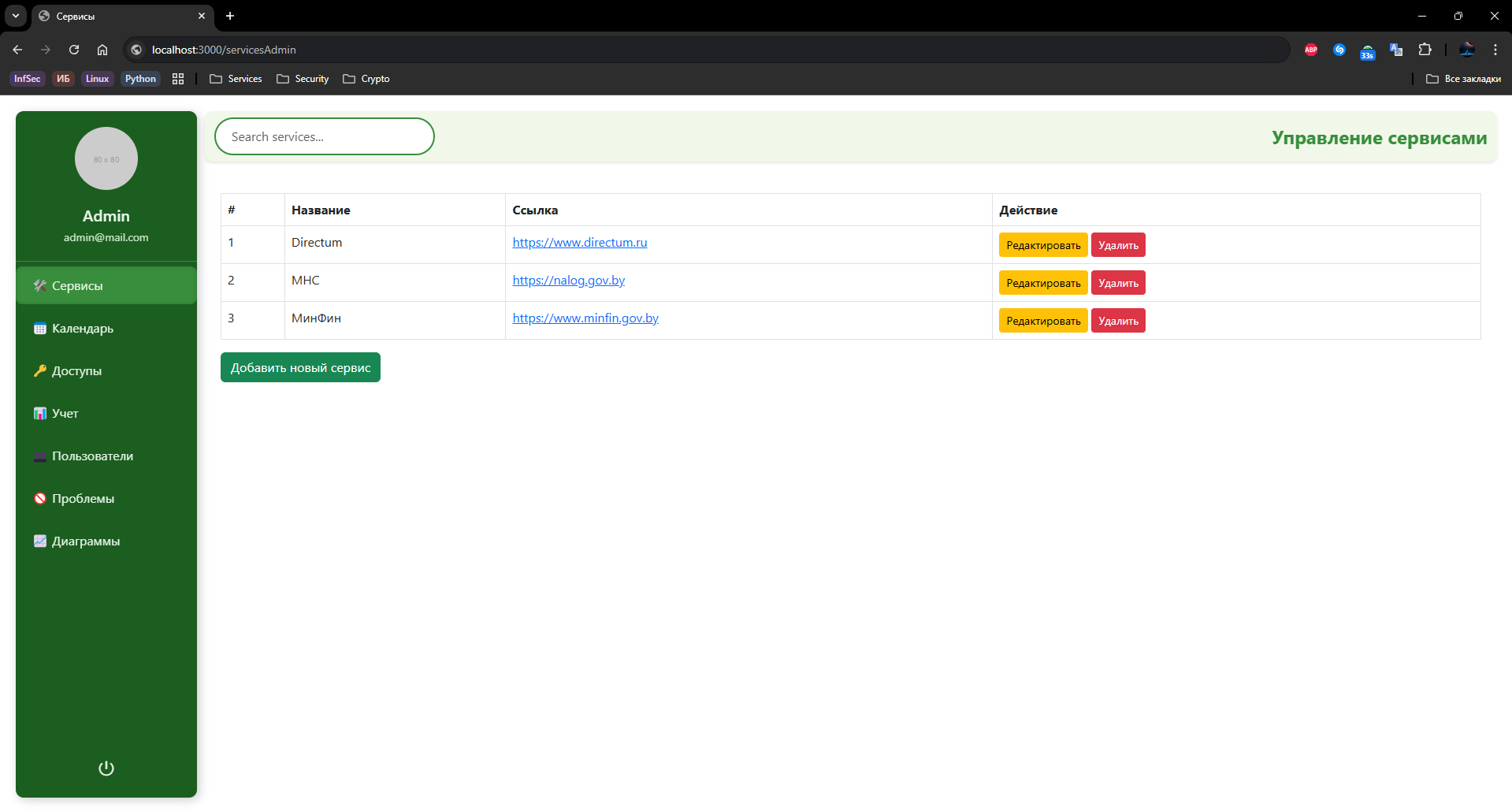


Рисунок 5.9 – Страница с сервисами в панели администратора

В панели администратор может видеть все видеоконференции, которые запланировали пользователи и их редактировать. Календарь на панели администратора показана на рисунке 5.10.

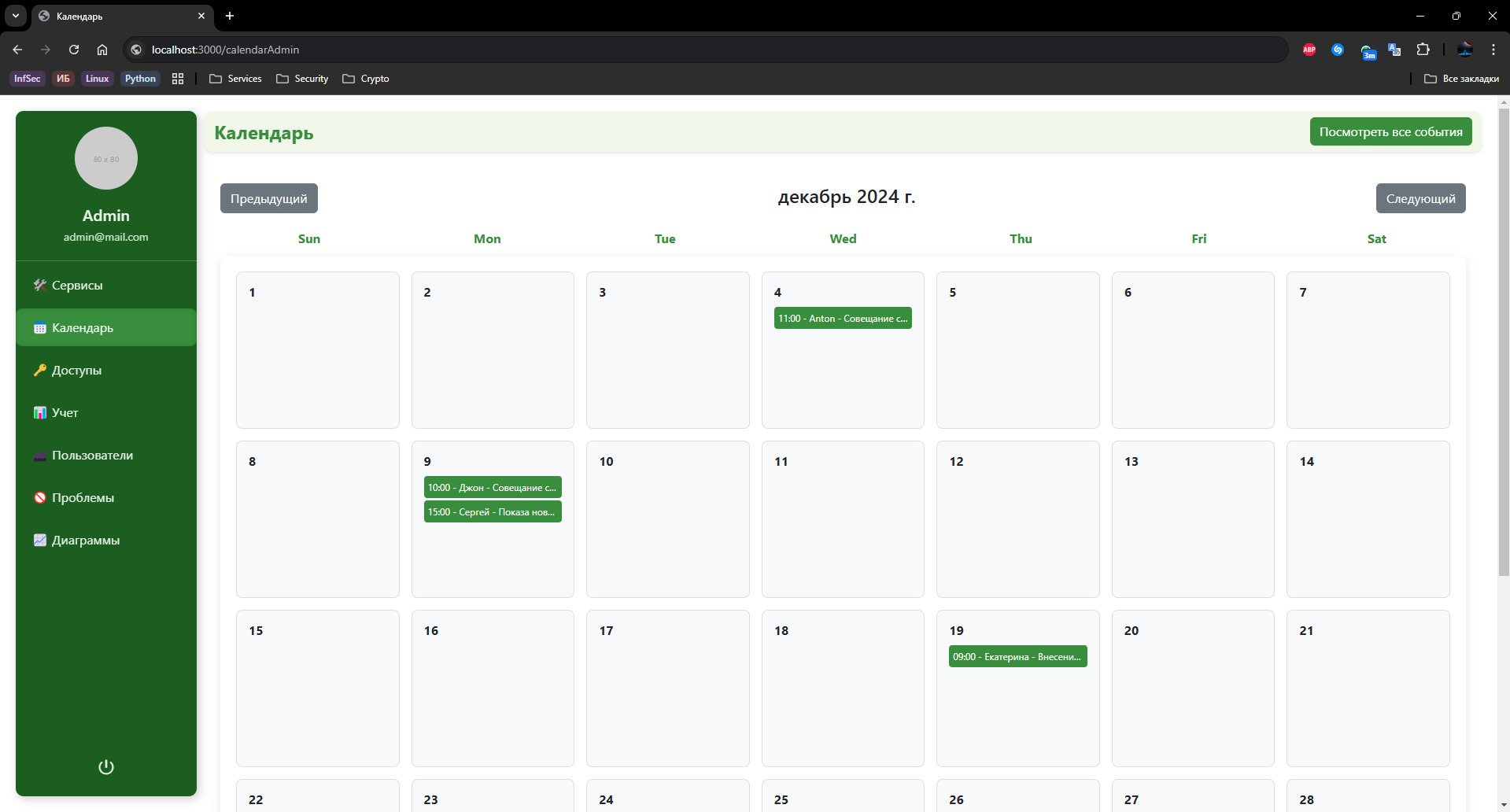


Рисунок 5.10 – Календарь на панели администрирования

Нажав на кнопку «просмотр всех событий», администратор может удобно, в виде списка просматривать все забронированные события. Диалоговое окно просмотра всех событий показано на рисунке 5.11.

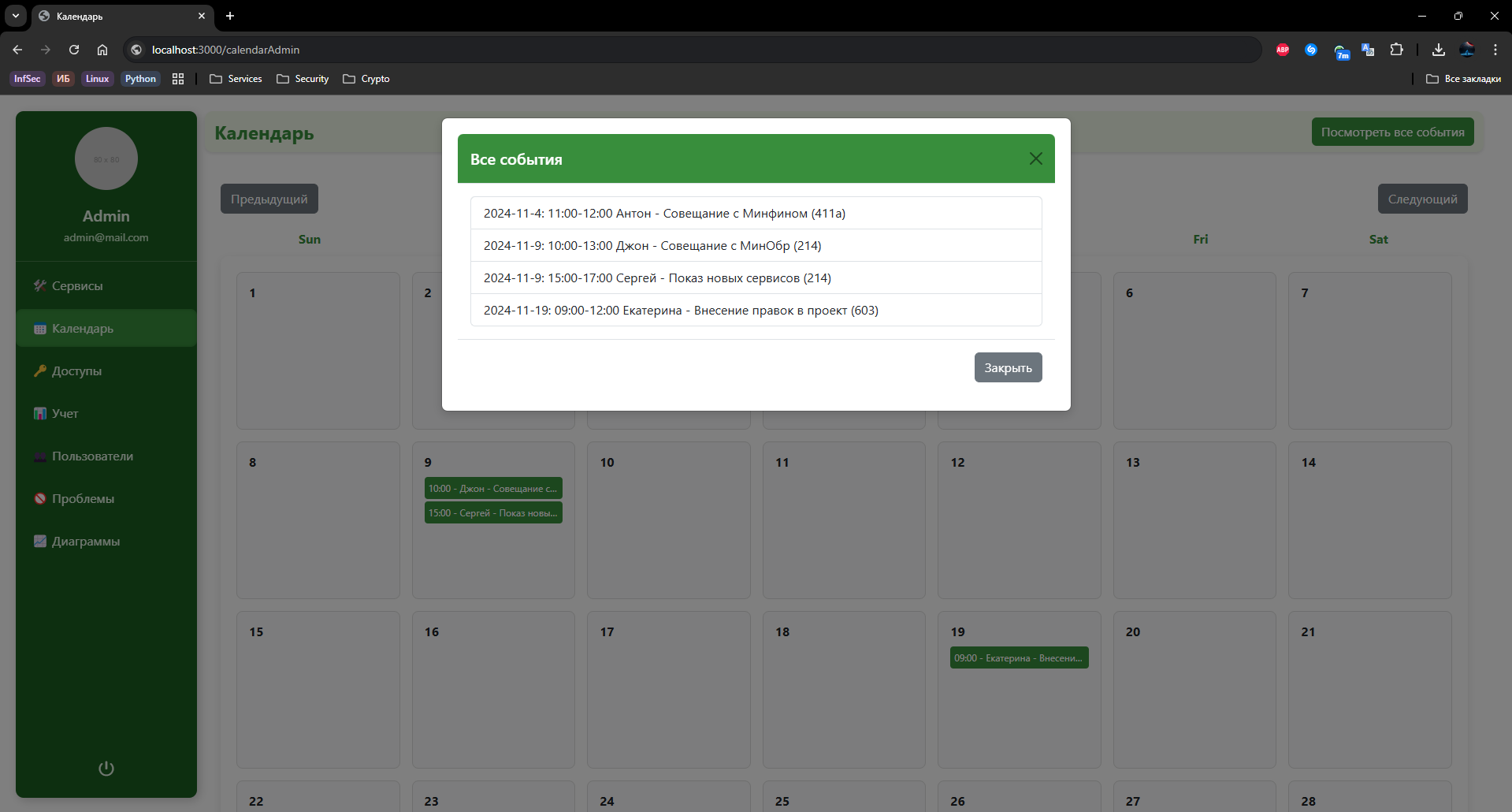


Рисунок 5.11 – Диалоговое окно просмотра всех событий в календаре

Информация о всех заявках на доступы на панели администратора показана на рисунке 5.12.

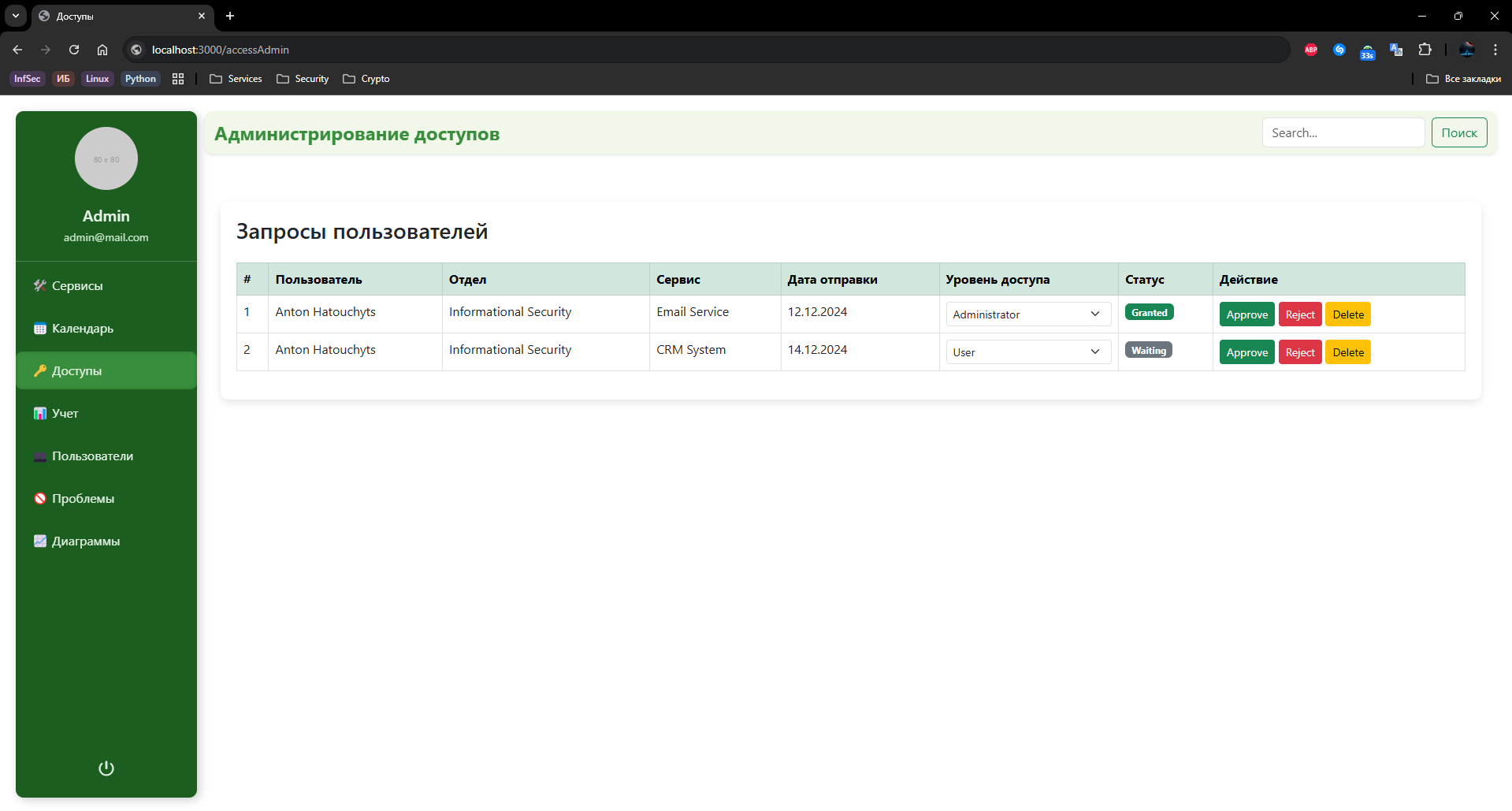


Рисунок 5.12 – Страница с заявками на доступы

Для ведения учета компьютерной техники, в панели администрирования предусмотрена страница с компьютерами пользователей. Страница с компьютерами пользователей представлена на рисунке 5.13.

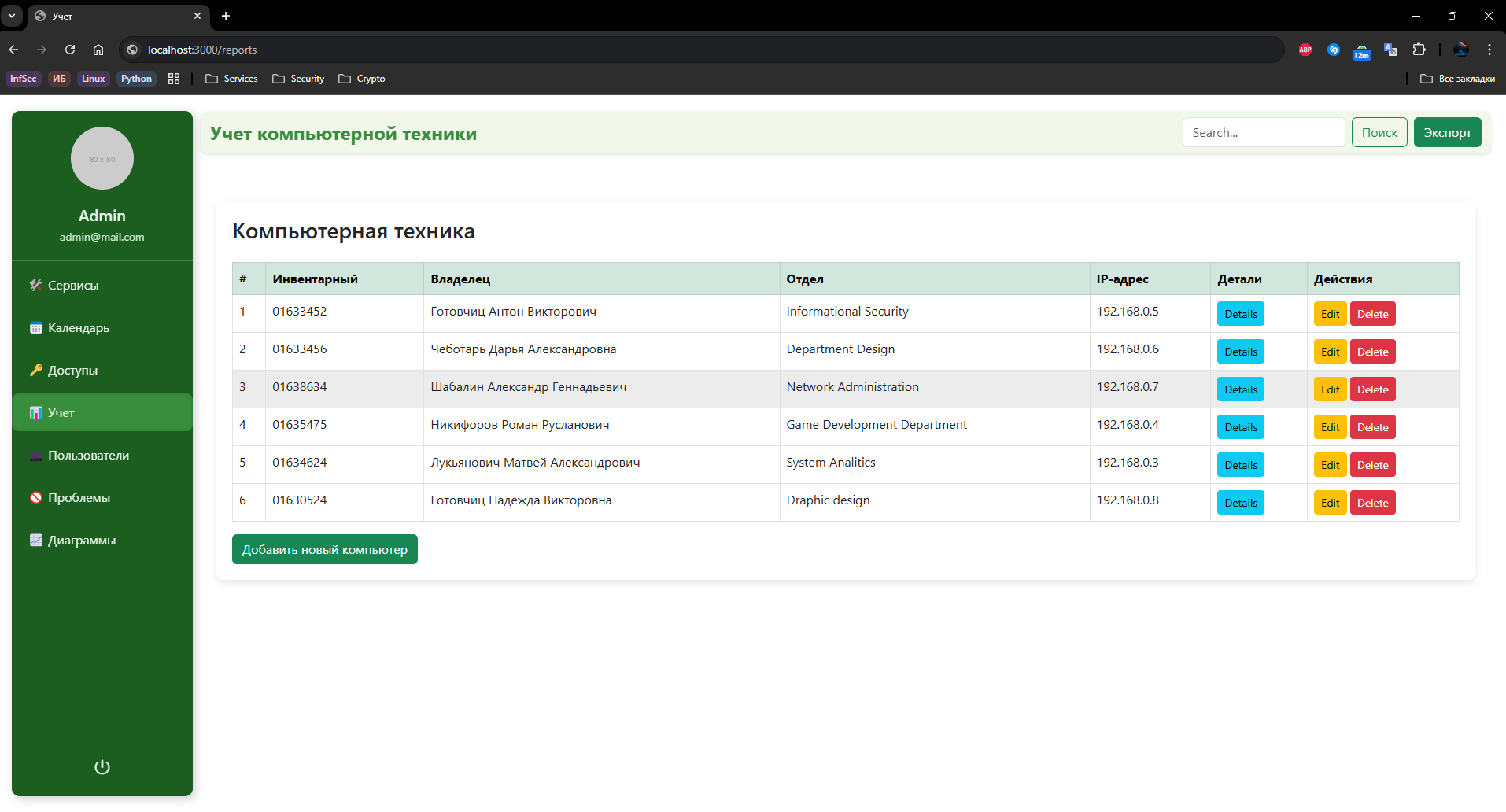


Рисунок 5.13 – Страница с компьютерами пользователей

Диалоговое окно для просмотра более подробной информации о характеристиках компьютера пользователя представлено на рисунке 5.14.

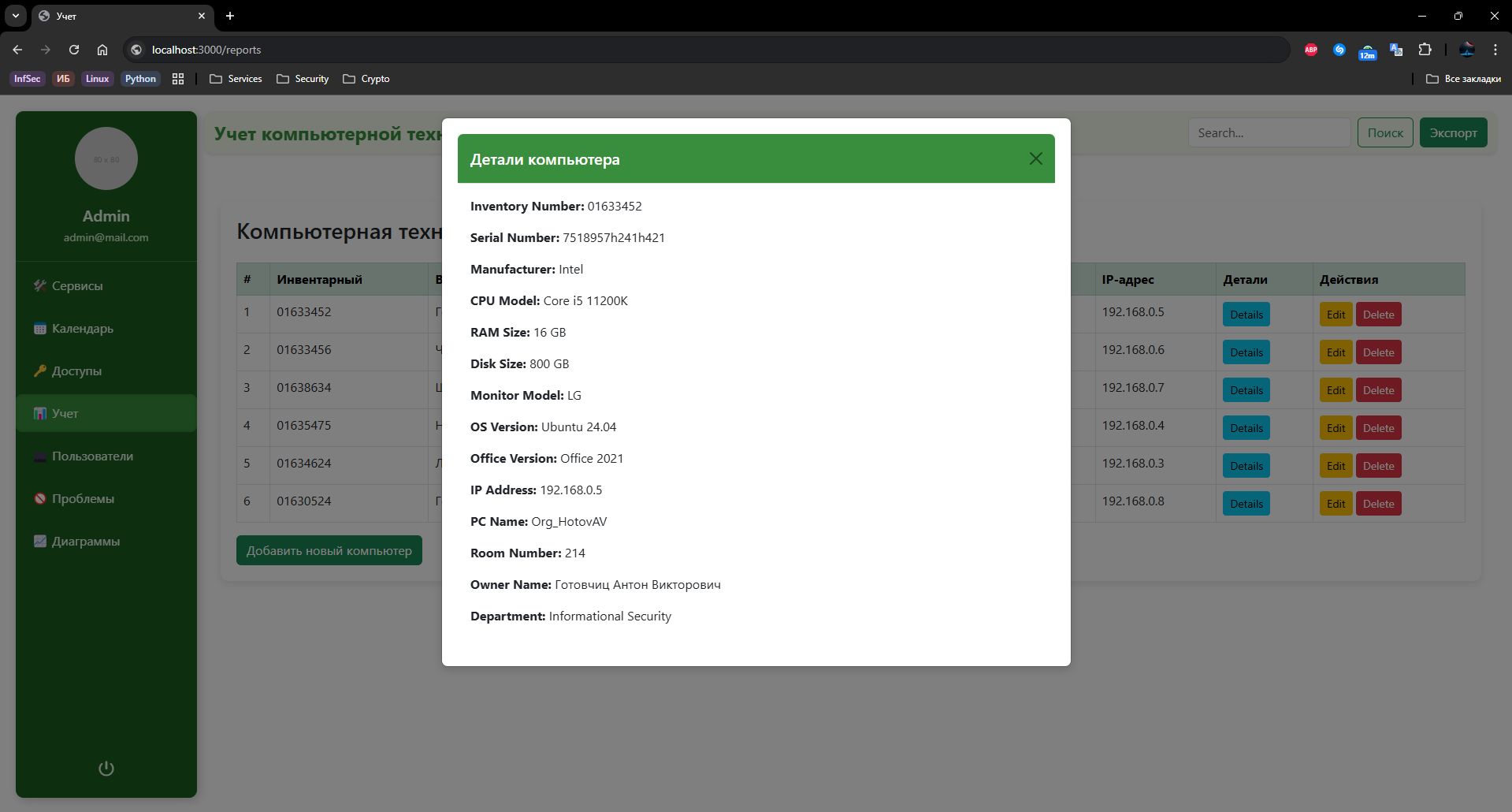


Рисунок 5.14 – Подробная информация о компьютере

Страница с информацией о пользователях представлен на рисунке 5.15.

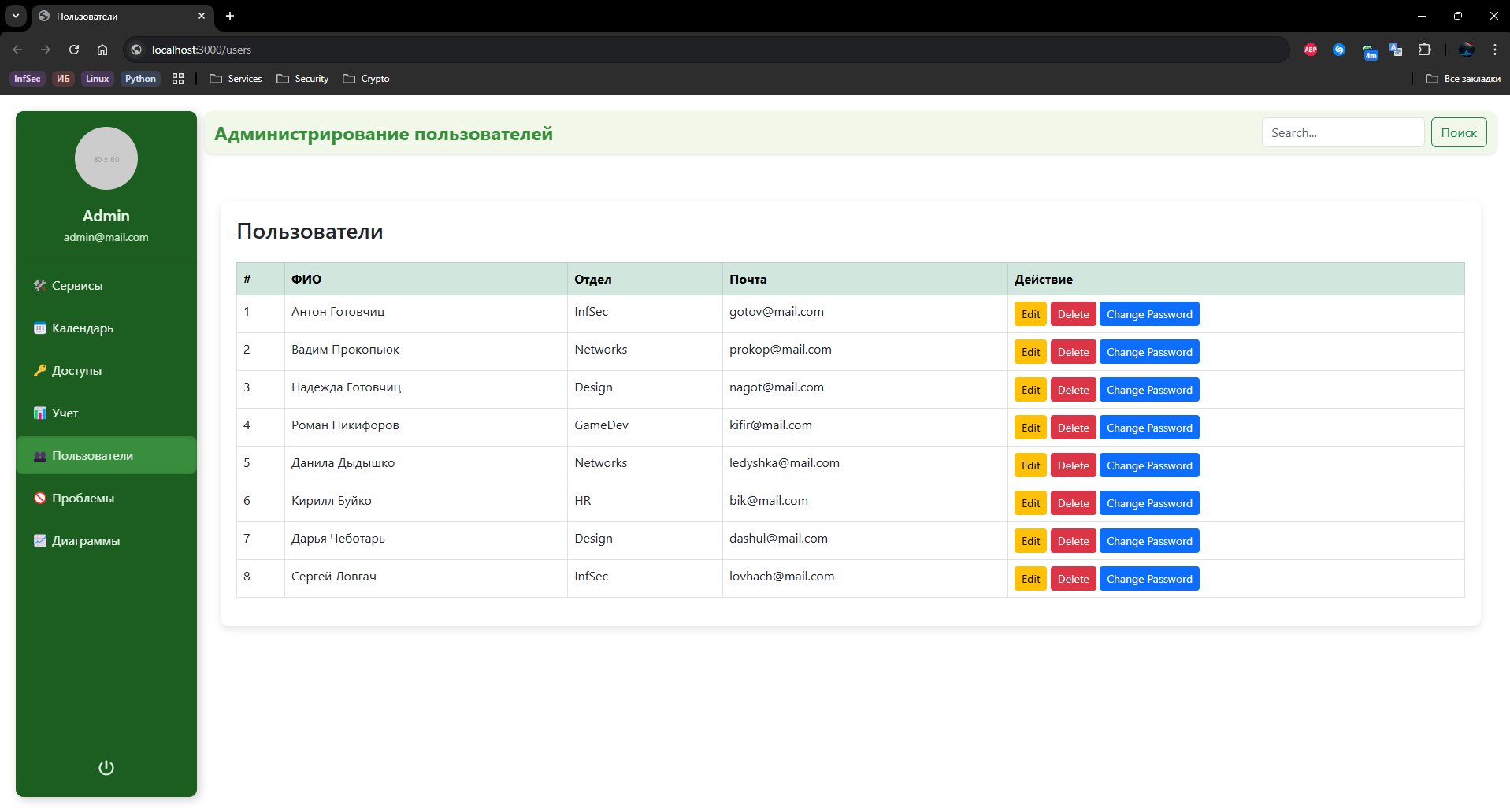


Рисунок 5.15 – Страница с пользователями

Страница с заявками пользователей на решение проблемы представлена на рисунке 5.16

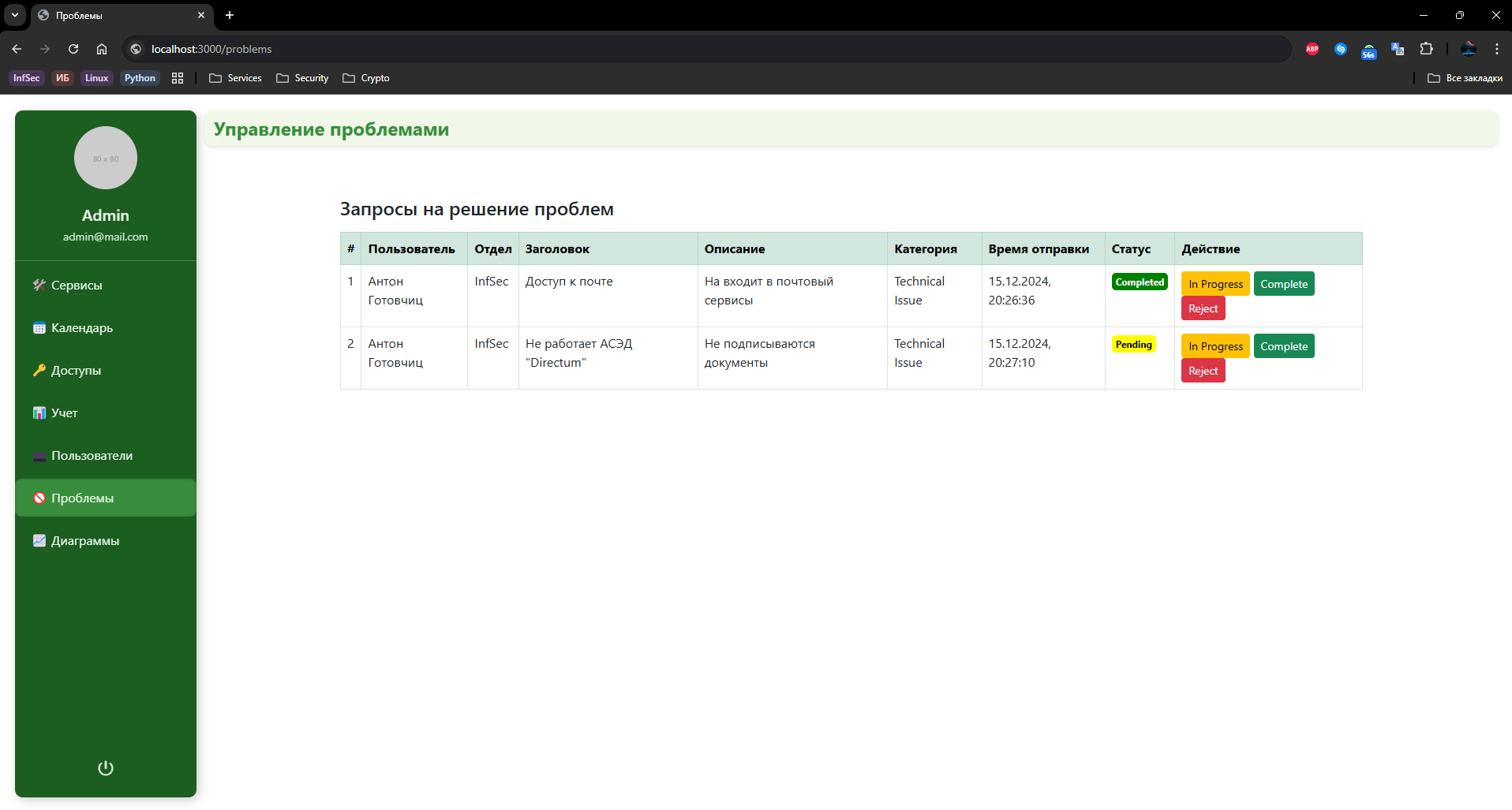


Рисунок 5.16 – Страница с проблемами пользователей

Для просмотра статистики по количеству решенных проблем, заявок на предоставление доступов, версий офиса на компьютерах пользователей, в программном средстве присутствует страница с диаграммами которая представлена на рисунке 5.17.

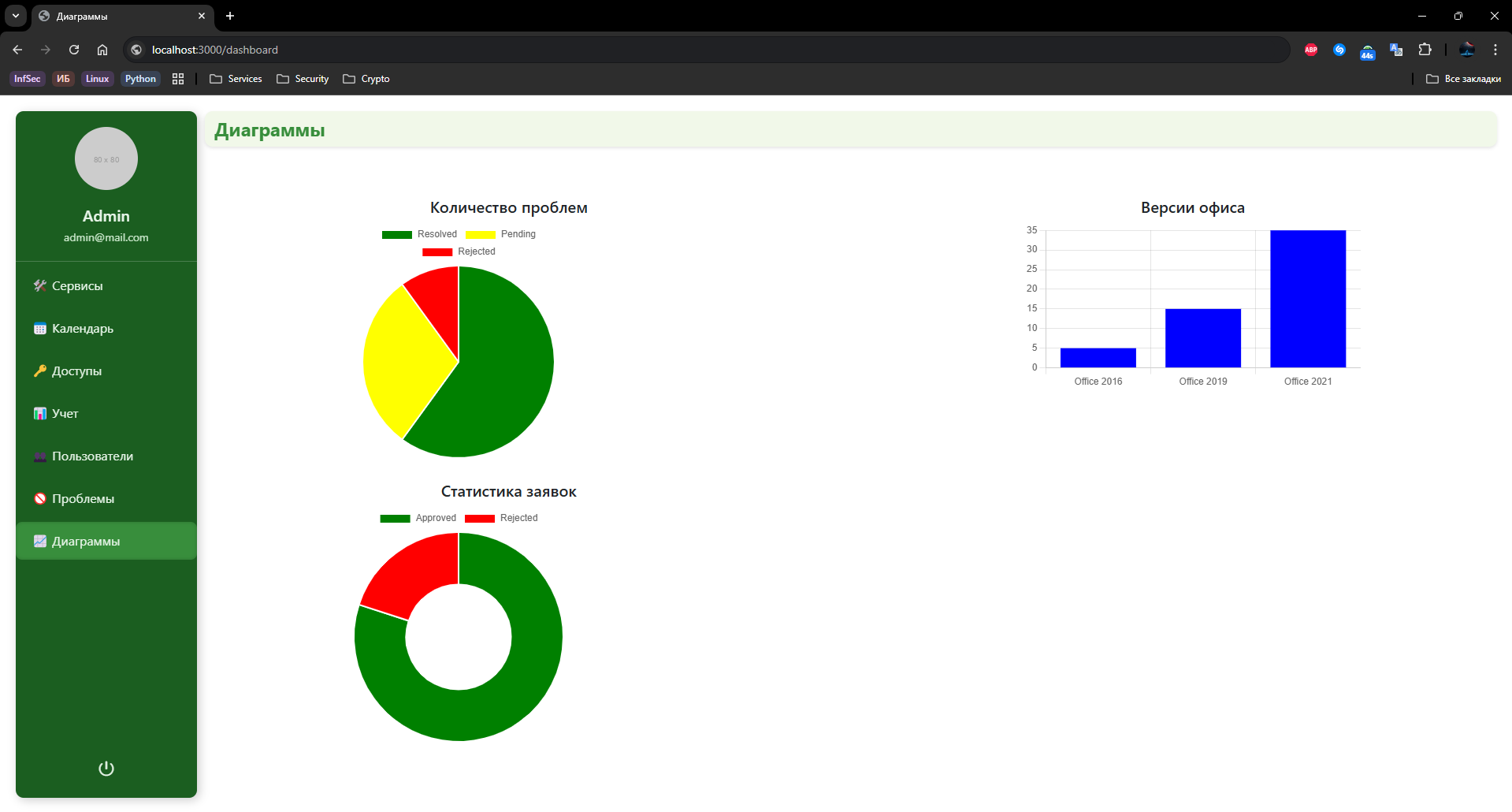


Рисунок 5.17 – Страница с диаграммами

1. **ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО СРЕДСТВА АВТОМАТИЗАЦИИ БИЗНЕС-ПРОЦЕССОВ ИНФОРМАЦИОННОГО ОТДЕЛА ОРГАНИЗАЦИИ**

## **Характеристика программного средства**

Целью создания программного средства является автоматизация бизнес-процессов информационного отдела организации, включая управление задачами, обработку заявок, планирование видеоконференций и работу с базой данных. Система направлена на повышение эффективности работы сотрудников, снижение человеческого фактора и ускорение выполнения административных и технических процессов.

Программное средство используется в информационном отделе для выполнения административных и технических задач, таких как:

* постановка и контроль выполнения задач;
* обработка заявок сотрудников;
* генерация отчетов.

Задачи:

* автоматизация рутинных операций;
* оптимизация документооборота;
* уменьшение времени обработки заявок.

Ожидаемый эффект:

* снижение трудозатрат на выполнение рутинных операций;
* увеличение скорости выполнения задач;
* сокращение ошибок при обработке документации.

## **Расчет инвестиций в разработку программного средства**

Для расчета затрат на основную заработную плату (Зо) необходимо определить состав команды, месячные оклады членов команды, их часовые оклады и трудоемкость работы.

Состав команды разработчиков:

* системный архитектор – разрабатывает архитектуру системы;
* программист – реализует функционал программного средства;
* тестировщик – проводит тестирование продукта;
* дизайнер – разрабатывает пользовательский интерфейс.

Исходные данные:

* количество рабочих часов в месяце 168;
* коэффициент премий и стимулирующих выплат (Кпр) 80%.

Месячные оклады специалистов:

* DevOps-инженер (2500 р.);
* программист (3000 р.);
* тестировщик (1800 р.);
* дизайнер (1500 р).

Трудоемкость разработки (часы):

* DevOps-инженер (80 ч.);
* программист (120 ч.);
* тестировщик (100 ч.);
* дизайнер (60 ч.).

Формула для расчета основной заработной платы:

|  |  |
| --- | --- |
| , | (6.1) |

где Кпр – коэффициент премий и стимулирующих выплат;

Зч𝑖 – часовой оклад исполнителя i-й категории (р.);

ti – трудоемкость работ исполнителя i-й категории (ч).

Формула для расчета часового оклада:

|  |  |
| --- | --- |
| , | (6.2) |

Таблица 6.1 – Расчет основной заработной платы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория исполнителя | Месячный оклад, р. | Часовой оклад, р. | Трудоемкость, ч. | Итого, р. |
| DevOps-инженер | 2500 | 14,88 | 80 | 1190,4 |
| Программист | 3000 | 17,85 | 120 | 2142 |
| Тестировщик | 1800 | 10,71 | 100 | 1071 |
| Дизайнер | 1500 | 8,92 | 60 | 535,2 |
| Итого: | | | | 4938,6 |
| Премия и иные стимулирующие выплаты (80%) | | | | 3950,88 |
| Всего затрат на основную заработную плату разработчиков | | | | 8889,48 |

Расчет затрат на дополнительную заработную плату разработчиков. Дополнительная заработная плата (Зд) рассчитывается на основе основной заработной платы (Зо) с учетом норматива дополнительной заработной платы (Нд).

|  |  |
| --- | --- |
| , | (6.3) |

где Нд – норматив дополнительной заработной платы (примем 15%). Рассчитаем дополнительную заработную плату:

|  |  |
| --- | --- |
| Зд = 8889,48 **⸱** 15 / 100 = 1 333,42 р. |  |

Отчисления на социальные нужды (Рсоц) рассчитываются на основе суммы основной (Зо) и дополнительной (Зд) заработной платы с учетом норматива отчислений на социальные нужды.

|  |  |
| --- | --- |
| , | (6.4) |

где Нсоц – норматив отчислений (по состоянию на июль 2021 года – 34,6%).

|  |  |
| --- | --- |
| Рсоц = (8889,48 + 1 333,42) × 34,6 / 100 = 3537,12 р. |  |

Прочие расходы (Рпр) включают затраты на оборудование, электроэнергию, аренду рабочих мест и другие дополнительные затраты. Они рассчитываются как доля от основной заработной платы (Зо) с учетом норматива прочих расходов (Нпр).

|  |  |
| --- | --- |
| , | (6.5) |

где Нпр – норматив прочих расходов (примем 30%).

|  |  |
| --- | --- |
| Рпр = 8889,48 × 30 / 100 = 2666,84 р. |  |

Общая сумма инвестиций (Зр) – это сумма всех затрат: основной заработной платы, дополнительной заработной платы, отчислений на социальные нужды и прочих расходов.

|  |  |
| --- | --- |
| , | (6.6) |

Рассчитаем общую сумму инвестиций:

|  |  |
| --- | --- |
| Зр = 8889,48 + 1333,42 + 3537,12 + 2666,84 = 16426,86 р. |  |

Таблица 6.2 – расчет инвестиций (затрат) на разработку программного средства

|  |  |  |
| --- | --- | --- |
| Наименование статьи затрат | Формула/таблица для расчета | Сумма, р |
| 1. Основная заработная плата разработчиков | Таблица (6.1) | 8889,48 |
| 1. Дополнительная заработная плата разработчиков | Формула (6.3) | 1333,42 |
| 1. Отчисления на социальные нужды | Формула (6.4) | 3537,12 |
| 1. Прочие расходы | Формула (6.5) | 2666,84 |
| 1. Общая сумма инвестиций (затрат) на разработку | Формула (6.6) | 16426,86 |

## **Расчет экономического эффекта от использования программного средства**

Экономический эффект от внедрения программного средства определяется за счет снижения текущих затрат, таких как трудозатраты, материальные ресурсы, затраты на заработную плату, а также за счет увеличения производительности труда и качества выполнения задач. Рассмотрим расчет экономического эффекта подробнее.

Цели внедрения программного средства:

* снижение трудоемкости выполнения «ручных» операций;
* сокращение времени, затрачиваемого сотрудниками на выполнение задач;
* снижение численности сотрудников, занятых рутинными процессами;
* сокращение времени на подготовку отчетности;
* увеличение числа обрабатываемых документов;
* уменьшение времени на создание отчетов;
* повышение качества сервиса;
* сокращение времени на устранение ошибок и инцидентов;
* рост объемов работы;
* увеличение заказов и скорости их выполнения.

Программное средство позволяет сократить трудозатраты сотрудников на выполнение рутинных операций. Рассмотрим расчет экономии на заработной плате и начислениях на заработную плату, вызванной снижением трудоемкости.

Исходные данные:

* до внедрения программного средства сотрудник выполнял 100 ч работы в месяц;
* после внедрения – 50 ч работы в месяц;
* часовой оклад сотрудника: 15 р;
* количество сотрудников: 4 человек;
* норматив дополнительной заработной платы (Нд): 15%;
* норматив отчислений на социальные нужды (Нсоц): 34,6%.

Формула для расчета экономии на заработной плате:

|  |  |
| --- | --- |
| , | (6.7) |

где Кпр – коэффициент премий (примем равным 1);

𝑡р без п.с – трудоемкость до внедрения программного средства (100 ч.);

𝑡р с п.с – трудоемкость после внедрения программного средства (50 ч.);

Тч – часовой оклад сотрудника (15 р.);

𝑁п – количество сотрудников (4 человека);

Нд – норматив дополнительной заработной платы (15%);

Нсоц – норматив отчислений на социальные нужды (34,6%).

|  |  |
| --- | --- |
| Эз.п = 50 × 15 × 4 × 1,15 × 1,346 = 4643,7 р. |  |

|  |  |
| --- | --- |
| Годовая экономия = 4643,7 × 12 = 55724,4 р. |  |

Расчет экономии за счет сокращения численности сотрудников. Предположим, что внедрение программного средства позволяет сократить численность сотрудников на 1 человека (выполнявшего рутинные операции).

Исходные данные:

* годовая заработная плата высвобожденного сотрудника: 36000 р. (3000 р. в месяц);
* норматив дополнительной заработной платы (Нд): 15%;
* норматив отчислений на социальные нужды (Нсоц): 34,6%.

|  |  |
| --- | --- |
| , | (6.8) |

где ∆Ч – численность сотрудников, сокращаемых за счет автоматизации (1 человек);

З – годовая заработная плата одного сотрудника (36000 р.).

|  |  |
| --- | --- |
| Эз.п.п = 36000 × 1,15 × 1,346 = 55724,4 р. |  |

Расчет экономии на материальных ресурсах. Внедрение программного средства позволяет снизить расходы на бумагу, печать отчетов и обработку документов.

Исходные данные:

* экономия бумаги: 500 листов в месяц;
* цена 1 листа: 0,20 р.;
* коэффициент транспортных расходов (Ктр): 1,1;
* плановый объем работы (𝑁п): 12 месяцев.

Для расчета экономии используется формула 1.9:

|  |  |
| --- | --- |
| , | (6.9) |

где Нр без п.с – расход бумаги до внедрения программного средства (500 листов);

Нр с п.с – расход бумаги после внедрения программного средства (0 листов);

Цм – цена за единицу материала (0,20 р.);

Ктр – коэффициент транспортных расходов.

|  |  |
| --- | --- |
| Эм = 1,1 × 500 × 0,2 × 12 = 1 320 р. |  |

Расчет годовой экономии на текущих затратах. Суммируем все виды экономии:

* экономия на трудозатратах сотрудников: 55724 р;
* экономия за счет сокращения численности сотрудников: 55724,4 р;
* экономия на материальных ресурсах: 1320 р.

Итого годовая экономия:

|  |  |
| --- | --- |
| Этек = 55724 + 55724,4 + 1320 = 112768,4 р. |  |

Расчет прироста чистой прибыли (∆Пч). Для расчета прироста чистой прибыли учитывается налог на прибыль (Нп).

Исходные данные:

* ставка налога на прибыль (Нп): 20%;
* текущие затраты на сопровождение программного средства (∆Зтек п.с): 10 000 р.

Формула для расчета:

|  |  |
| --- | --- |
| ), | 6.10) |

Итого прирост чисто прибыли:

|  |  |
| --- | --- |
| ∆Пч = 102 768,4 × 0,8 = 82214,72 р. |  |

## **Расчет показателей экономической эффективности разработки и использования программного средства в организации**

После расчета затрат на разработку программного средства (пункт 1.2) и экономического эффекта от его использования (пункт 1.3), необходимо определить, насколько эффективно вложение средств в проект. Для этого рассчитываются следующие показатели:

* срок окупаемости инвестиций (Ток);
* рентабельность инвестиций (ROI).

Расчет срока окупаемости инвестиций (Ток). Срок окупаемости инвестиций (Ток) – это время, за которое вложенные средства (затраты на разработку) будут возвращены за счет полученного экономического эффекта. Рассчитывается по формуле:

|  |  |
| --- | --- |
| , | (6.11) |

где Зр – затраты на разработку программного средства (из пункта 1.2): 16426,86 р;

Эгод – годовой экономический эффект (из пункта 1.3): 112768,4 р.

|  |  |
| --- | --- |
| Ток = 16426,86 / 112768,4 = 1,68 месяца |  |

Рентабельность инвестиций (ROI) показывает, насколько прибыльными являются вложения. Она рассчитывается как отношение прибыли, полученной от проекта, к затратам на его разработку. Рассчитывается по формуле:

|  |  |
| --- | --- |
| , | (6.12) |

где ∆Пч – прирост чистой прибыли (из пункта 1.3): 84270,08 р.

Зр – затраты на разработку программного средства: 16426,86 р.

|  |  |
| --- | --- |
| ROI = 65787,86 / 16426,86 × 100% = 400,48% |  |

Рентабельность вложений в проект составляет 400,48%.

Расчет коэффициента экономической эффективности (Е). Коэффициент экономической эффективности (Е) показывает, сколько прибыли приносит каждый вложенный рубль. Рассчитывается по формуле:

|  |  |
| --- | --- |
| , | (6.13) |

где Эгод – годовой экономический эффект: 112768,4 р;

Зр – затраты на разработку: 16426,86 р.

|  |  |
| --- | --- |
| Е = 112768,4 / 16426,86 = 6,86 |  |

Итого коэффициент экономической эффективности показывает, что каждый рубль, вложенный в разработку программного средства, приносит 6,86 рубля прибыли.

В заключении моно сказать, что проект является высокоэффективным и быстро окупаемым. Вложения окупаются за один месяц, а рентабельность превышает 1000%. Это свидетельствует о целесообразности разработки и внедрения программного средства для автоматизации бизнес-процессов.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломного проекта была разработана система для автоматизации бизнес-процессов информационного отдела организации, направленная на улучшение эффективности работы с задачами, видеоконференциями, запросами на доступ и управлением сервисами. В процессе работы над проектом были исследованы существующие подходы к автоматизации бизнес-процессов в информационных отделах и выбраны оптимальные технологии для разработки программного средства.

Сформулированы требования к функциональности и безопасности системы, разработаны архитектура и интерфейс приложения. Для реализации были использованы современные инструменты, такие как Python, JavaScript, Bootstrap и PostgreSQL, что обеспечило гибкость и масштабируемость системы.

Созданы базы данных и соответствующие схемы, спроектированы и реализованы основные функциональные модули, включая календарь для планирования задач, систему уведомлений, управление доступом и интеграцию с видеоконференцсвязью. Программное средство было протестировано на различных этапах разработки, включая тестирование функциональности, производительности и безопасности.

В результате тестирования были выявлены и устранены ошибки, а также проведены оптимизации для улучшения производительности при высоких нагрузках. Внесенные исправления позволили системе работать стабильно и эффективно даже при большом объеме данных и высоком числе пользователей.

Подготовлена подробная документация, включающая руководство по эксплуатации программного средства, а также технико-экономическое обоснование разработки.

Таким образом, все поставленные цели дипломного проекта были достигнуты. Программное средство для автоматизации бизнес-процессов информационного отдела разработано, протестировано и готово к внедрению в реальную эксплуатацию.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Система «Antiplagius» [Электронный ресурс]. – Режим доступа: *https://antiplagius.ru.*
2. Левин, В. М. Основы проектирования информационных систем: Учебное пособие / В. М. Левин. – М.: Юрайт, 2018. – 345 с.
3. Парахин, В. Н. Системы управления базами данных: Учебник для вузов / В. Н. Парахин. – М.: ИНФРА-М, 2019. – 462 с.
4. Кириченко, И. С. Основы UML и проектирования информационных систем: учебное пособие / И. С. Кириченко. – СПб.: Питер, 2020. – 320 с.
5. Макаров, П. В. Автоматизация бизнес-процессов: Учебное пособие / П. В. Макаров. – М.: Юрайт, 2017. – 273 с.
6. Таненбаум, Э. Архитектура компьютера. Структурный подход / Э. Таненбаум. – СПб.: Питер, 2014. – 752 с.
7. Панкратов, А. А. Организация компьютерных систем и сетей / А. А. Панкратов, О. С. Соловьёв. – СПб.: БХВ-Петербург, 2019. – 496 с.
8. Кантер, А. Б. Современные средства разработки веб-приложений / А. Б. Кантер. – М.: Финансы и статистика, 2016. – 224 с.
9. Бэйти, Б. Современные базы данных. Проектирование и разработка / Б. Бэйти, Э. Сингх. – М.: Вильямс, 2021. – 480 с.
10. Эрикссон, М., Пенкер, М. Бизнес-моделирование с помощью UML: учебник для вузов / М. Эрикссон, М. Пенкер. – М.: Диалектика, 2019. – 496 с.
11. Бёрд, Л. Введение в базы данных с PostgreSQL / Л. Бёрд, М. Иордан. – М.: Вильямс, 2020. – 384 с.
12. Дейт, К. Дж. Основы систем баз данных / К. Дж. Дейт, Э. Фишер. – М.: Вильямс, 2018. – 1072 с.
13. Флэнаган, Д. JavaScript. Подробное руководство / Д. Флэнаган. – М.: Питер, 2021. – 1024 с.
14. Стюарт, М. Web-разработка с использованием HTML, CSS и JavaScript / М. Стюарт. – М.: Лори, 2017. – 576 с.
15. Зандстра, М. Python для веб-разработки. Создание приложений с Django / М. Зандстра. – СПб.: Питер, 2019. – 608 с.

# ПРИЛОЖЕНИЕ А (обязательное) Исходный код программного средства

*from django.shortcuts import render, redirect*

*from django.contrib.auth import authenticate, login*

*from django.contrib.auth.decorators import login\_required*

*from django.contrib import messages*

*from .models import User # Используем кастомную модель пользователя*

*def login\_view(request):*

*if request.method == 'POST':*

*email = request.POST.get('email')*

*password = request.POST.get('password')*

*user = authenticate(request, username=email, password=password)*

*if user:*

*login(request, user)*

*return redirect('services') # Перенаправление на страницу с сервисами*

*else:*

*return render(request, 'accounts/auth/login.html', {*

*'error\_message': 'Invalid email or password',*

*})*

*return render(request, 'accounts/auth/login.html')*

*def register\_view(request):*

*if request.method == 'POST':*

*full\_name = request.POST.get('full\_name')*

*email = request.POST.get('email')*

*password = request.POST.get('password')*

*department = request.POST.get('department') # Новое поле*

*if User.objects.filter(email=email).exists():*

*messages.error(request, 'Email already registered.')*

*return render(request, 'accounts/auth/register.html')*

*# Создание пользователя*

*user = User.objects.create\_user(*

*email=email,*

*password=password,*

*full\_name=full\_name,*

*department=department # Сохраняем department*

*)*

*user.save()*

*login(request, user) # Автоматический вход после регистрации*

*return redirect('services') # Перенаправление на страницу сервисов*

*return render(request, 'accounts/auth/register.html')*

*@login\_required*

*def services\_view(request):*

*return render(request, 'accounts/user/services.html')*

*from django.db import models*

*from django.contrib.auth.models import AbstractBaseUser, BaseUserManager, PermissionsMixin*

*class UserManager(BaseUserManager):*

*def create\_user(self, email, password=None, \*\*extra\_fields):*

*if not email:*

*raise ValueError("The Email field must be set")*

*email = self.normalize\_email(email)*

*user = self.model(email=email, \*\*extra\_fields)*

*user.set\_password(password)*

*user.save(using=self.\_db)*

*return user*

*def create\_superuser(self, email, password=None, \*\*extra\_fields):*

*extra\_fields.setdefault('is\_staff', True)*

*extra\_fields.setdefault('is\_superuser', True)*

*return self.create\_user(email, password, \*\*extra\_fields)*

*class User(AbstractBaseUser, PermissionsMixin):*

*full\_name = models.CharField(max\_length=255)*

*department = models.CharField(max\_length=255, blank=True, null=True)*

*email = models.EmailField(unique=True)*

*password = models.CharField(max\_length=255)*

*role = models.CharField(max\_length=10, choices=[('user', 'User'), ('admin', 'Admin')], default='user')*

*is\_active = models.BooleanField(default=True)*

*is\_staff = models.BooleanField(default=False)*

*objects = UserManager()*

*USERNAME\_FIELD = 'email'*

*REQUIRED\_FIELDS = ['full\_name']*

*def \_\_str\_\_(self):*

*return self.email*

*class Meta:*

*db\_table = 'users' # Имя таблицы в базе данных*

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Доступы</title>*

*<link href="../../css/services.css" rel="stylesheet">*

*<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">*

*<style>*

*.status-pending {*

*background-color: yellow;*

*color: black;*

*padding: 5px;*

*border-radius: 5px;*

*}*

*.status-granted {*

*background-color: green;*

*color: white;*

*padding: 5px;*

*border-radius: 5px;*

*}*

*.status-rejected {*

*background-color: red;*

*color: white;*

*padding: 5px;*

*border-radius: 5px;*

*}*

*.status-waiting {*

*background-color: orange;*

*color: white;*

*padding: 5px;*

*border-radius: 5px;*

*}*

*</style>*

*</head>*

*<body>*

*<div class="sidebar">*

*<div class="profile">*

*<img src="https://via.placeholder.com/80" alt="Profile">*

*<h5>Anton Hatouchyts</h5>*

*<p>gotov@gmail.com</p>*

*</div>*

*<a href="./services.html" class="nav-link">🛠 Сервисы</a>*

*<a href="./calendar.html" class="nav-link">📅 Календарь</a>*

*<a href="./access.html" class="nav-link active">🔑 Доступы</a>*

*<a href="./problems.html" class="nav-link">🚫 Проблемы</a>*

*<div class="logout">*

*<button onclick="location.href='../Auth/login.html'" class="logout-btn" title="Logout">⏻</button>*

*</div>*

*</div>*

*<nav class="navbar">*

*<div class="container-fluid">*

*<h3 class="page-title">Доступы</h3>*

*</div>*

*<hr class="section-divider">*

*</nav>*

*<div class="content">*

*<div class="table-container">*

*<h3 class="mb-4">Запросы на доступ</h3>*

*<table class="table table-bordered">*

*<thead class="table-success">*

*<tr>*

*<th>Сервис</th>*

*<th>Дата создания</th>*

*<th>Статус</th>*

*<th>Назначеные права</th>*

*<th>Действие</th>*

*<th>Печать</th>*

*</tr>*

*</thead>*

*<tbody id="requestTable">*

*<!-- Requests will populate here dynamically -->*

*</tbody>*

*</table>*

*<button class="btn btn-success" data-bs-toggle="modal" data-bs-target="#requestModal">Отправить запрос на доступ</button>*

*</div>*

*<!-- Modal for submitting a new request -->*

*<div class="modal fade" id="requestModal" tabindex="-1" aria-labelledby="requestModalLabel" aria-hidden="true">*

*<div class="modal-dialog">*

*<div class="modal-content">*

*<div class="modal-header">*

*<h5 class="modal-title" id="requestModalLabel">Отправить запрос</h5>*

*<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>*

*</div>*

*<div class="modal-body">*

*<form id="requestForm">*

*<div class="mb-3">*

*<label for="serviceSelect" class="form-label">Выбрать сервис</label>*

*<select id="serviceSelect" class="form-select">*

*<option value="Email Service">Почта</option>*

*<option value="File Storage">FTP с БГУИР</option>*

*<option value="CRM System">Directum</option>*

*</select>*

*</div>*

*<button type="submit" class="btn btn-success">Отправить</button>*

*</form>*

*</div>*

*</div>*

*</div>*

*</div>*

*</div>*

*<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>*

*<script>*

*const requestTable = document.getElementById('requestTable');*

*const requestForm = document.getElementById('requestForm');*

*// Load requests from LocalStorage*

*const loadRequests = () => {*

*const requests = JSON.parse(localStorage.getItem('accessRequests')) || [];*

*requestTable.innerHTML = requests.map((req, index) => `*

*<tr>*

*<td>${req.service}</td>*

*<td>${req.date}</td>*

*<td><span class="${getStatusClass(req.status)}">${req.status}</span></td>*

*<td>${req.rights || 'User'}</td>*

*<td><span class="button-action button-cancel" onclick="cancelRequest(${index})">Отмена</span></td>*

*<td>*

*<button class="btn btn-primary btn-sm" onclick="printRequest(${index})">Печать</button>*

*</td>*

*</tr>*

*`).join('');*

*};*

*// Get status class*

*const getStatusClass = (status) => {*

*switch (status.toLowerCase()) {*

*case 'pending':*

*return 'status-pending';*

*case 'granted':*

*return 'status-granted';*

*case 'rejected':*

*return 'status-rejected';*

*case 'waiting':*

*return 'status-waiting';*

*default:*

*return '';*

*}*

*};*

*// Save request to LocalStorage*

*const saveRequest = (service) => {*

*const requests = JSON.parse(localStorage.getItem('accessRequests')) || [];*

*const newRequest = {*

*service,*

*date: new Date().toLocaleDateString(),*

*status: 'Waiting',*

*rights: null,*

*};*

*requests.push(newRequest);*

*localStorage.setItem('accessRequests', JSON.stringify(requests));*

*loadRequests();*

*};*

*// Cancel a request*

*const cancelRequest = (index) => {*

*const requests = JSON.parse(localStorage.getItem('accessRequests')) || [];*

*requests.splice(index, 1);*

*localStorage.setItem('accessRequests', JSON.stringify(requests));*

*loadRequests();*

*};*

*// Handle form submission*

*requestForm.addEventListener('submit', (e) => {*

*e.preventDefault();*

*const service = document.getElementById('serviceSelect').value;*

*saveRequest(service);*

*requestForm.reset();*

*const modal = bootstrap.Modal.getInstance(document.getElementById('requestModal'));*

*modal.hide();*

*});*

*const printRequest = (index) => {*

*const requests = JSON.parse(localStorage.getItem('accessRequests')) || [];*

*const request = requests[index];*

*const printWindow = window.open('', '\_blank');*

*printWindow.document.write(`*

*<html>*

*<head>*

*<title>Print Request</title>*

*<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">*

*</head>*

*<body>*

*<div class="container mt-5">*

*<h3>Access Request</h3>*

*<p><strong>Service:</strong> ${request.service}</p>*

*<p><strong>Date:</strong> ${request.date}</p>*

*<p><strong>Status:</strong> ${request.status}</p>*

*<p><strong>Granted Rights:</strong> ${request.rights || 'User'}</p>*

*</div>*

*</body>*

*</html>*

*`);*

*printWindow.document.close();*

*printWindow.print();*

*};*

*// Initial load*

*loadRequests();*

*</script>*

*</body>*

*</html>*

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Сервисы</title>*

*<link href="../../../static/css/services.css" rel="stylesheet">*

*<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">*

*<style>*

*.tile {*

*background-color: #e8f5e9;*

*color: #388e3c;*

*border-radius: 8px;*

*text-align: center;*

*padding: 20px;*

*margin: 10px;*

*font-size: 18px;*

*font-weight: bold;*

*cursor: pointer;*

*transition: transform 0.3s, box-shadow 0.3s;*

*}*

*.tile:hover {*

*transform: scale(1.05);*

*box-shadow: 0 6px 12px rgba(0, 0, 0, 0.15);*

*}*

*.service-tile {*

*cursor: pointer;*

*}*

*.add-tile {*

*background-color: #e0f7fa;*

*color: #00796b;*

*border: 2px dashed #00796b;*

*display: flex;*

*align-items: center;*

*justify-content: center;*

*font-size: 36px;*

*}*

*.modal-header {*

*background-color: #388e3c;*

*color: white;*

*}*

*.modal-footer button {*

*margin-left: 10px;*

*}*

*.selected-service {*

*background-color: #c8e6c9;*

*border: 2px solid #388e3c;*

*}*

*</style>*

*</head>*

*<body>*

*<div class="sidebar">*

*<div class="profile">*

*<img src="https://via.placeholder.com/80" alt="Profile">*

*<h5>Anton Hatouchyts</h5>*

*<p>gotov@gmail.com</p>*

*</div>*

*<a href="./services.html" class="nav-link active">🛠 Сервисы</a>*

*<a href="./calendar.html" class="nav-link">📅 Календарь</a>*

*<a href="./access.html" class="nav-link">🔑 Доступы</a>*

*<a href="./problems.html" class="nav-link">🚫 Проблемы</a>*

*<div class="logout">*

*<button onclick="location.href='./login.html'" class="logout-btn" title="Logout">⏻</button>*

*</div>*

*</div>*

*<nav class="navbar">*

*<div class="container-fluid">*

*<div class="d-flex justify-content-between align-items-center w-100">*

*<h3 class="page-title">Сервисы</h3>*

*<button id="editLinksButton" class="btn btn-success">Изменить ссылки</button>*

*</div>*

*</div>*

*<hr class="section-divider">*

*</nav>*

*<div class="content">*

*<div class="tiles" id="serviceTiles">*

*<div class="tile add-tile" id="addServiceTile" data-bs-toggle="modal" data-bs-target="#addServiceModal">*

*<span>+</span>*

*</div>*

*</div>*

*</div>*

*<!-- Modal for selecting multiple services -->*

*<div class="modal fade" id="addServiceModal" tabindex="-1" aria-labelledby="addServiceModalLabel" aria-hidden="true">*

*<div class="modal-dialog modal-lg">*

*<div class="modal-content">*

*<div class="modal-header">*

*<h5 class="modal-title" id="addServiceModalLabel">Выберите сервисы</h5>*

*<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>*

*</div>*

*<div class="modal-body">*

*<div class="row" id="availableServicesContainer">*

*<!-- Services will be dynamically added here -->*

*</div>*

*</div>*

*<div class="modal-footer">*

*<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Отменить</button>*

*<button id="addSelectedServicesButton" type="button" class="btn btn-success">Добавить</button>*

*</div>*

*</div>*

*</div>*

*</div>*

*<!-- Modal for editing links -->*

*<div class="modal fade" id="linksModal" tabindex="-1" aria-labelledby="linksModalLabel" aria-hidden="true">*

*<div class="modal-dialog">*

*<div class="modal-content">*

*<div class="modal-header">*

*<h5 class="modal-title" id="linksModalLabel">Редактировать ссылки</h5>*

*<button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>*

*</div>*

*<div class="modal-body">*

*<ul id="linksList" class="list-group">*

*<!-- List of links will be dynamically added here -->*

*</ul>*

*</div>*

*<div class="modal-footer">*

*<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Закрыть</button>*

*</div>*

*</div>*

*</div>*

*</div>*

*<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>*

*<script>*

*const serviceTiles = document.getElementById('serviceTiles');*

*const addServiceTile = document.getElementById('addServiceTile');*

*const editLinksButton = document.getElementById('editLinksButton');*

*const linksModal = new bootstrap.Modal(document.getElementById('linksModal'));*

*const linksList = document.getElementById('linksList');*

*const availableServicesContainer = document.getElementById('availableServicesContainer');*

*const addSelectedServicesButton = document.getElementById('addSelectedServicesButton');*

*const availableServices = [*

*"Directum",*

*"ЖД",*

*"Ilex",*

*"НРПА",*

*"МНС",*

*"МинФин"*

*];*

*let userServices = JSON.parse(localStorage.getItem('userServices')) || [];*

*let selectedServices = [];*

*function updateDisplay() {*

*serviceTiles.innerHTML = '';*

*userServices.forEach((serviceName) => {*

*const newTile = document.createElement('div');*

*newTile.classList.add('tile');*

*newTile.innerHTML = `<a href="#">${serviceName}</a>`;*

*newTile.onclick = () => alert(`Navigating to ${serviceName}`); // Replace with actual navigation logic*

*serviceTiles.appendChild(newTile);*

*});*

*serviceTiles.appendChild(addServiceTile);*

*}*

*function renderAvailableServices() {*

*availableServicesContainer.innerHTML = '';*

*availableServices.forEach((serviceName) => {*

*const serviceCard = document.createElement('div');*

*serviceCard.classList.add('col-md-4', 'mb-3');*

*serviceCard.innerHTML = `*

*<div class="card service-tile" onclick="toggleServiceSelection('${serviceName}')">*

*<div class="card-body text-center">*

*<h5 class="card-title">${serviceName}</h5>*

*</div>*

*</div>*

*`;*

*availableServicesContainer.appendChild(serviceCard);*

*});*

*}*

*function toggleServiceSelection(serviceName) {*

*if (selectedServices.includes(serviceName)) {*

*selectedServices = selectedServices.filter(service => service !== serviceName);*

*} else {*

*selectedServices.push(serviceName);*

*}*

*updateServiceCardStyles();*

*}*

*function updateServiceCardStyles() {*

*const cards = availableServicesContainer.querySelectorAll('.card');*

*cards.forEach(card => {*

*const title = card.querySelector('.card-title').textContent;*

*if (selectedServices.includes(title)) {*

*card.classList.add('selected-service');*

*} else {*

*card.classList.remove('selected-service');*

*}*

*});*

*}*

*addSelectedServicesButton.addEventListener('click', () => {*

*selectedServices.forEach(service => {*

*if (!userServices.includes(service)) {*

*userServices.push(service);*

*}*

*});*

*localStorage.setItem('userServices', JSON.stringify(userServices));*

*selectedServices = [];*

*updateDisplay();*

*const modal = bootstrap.Modal.getInstance(document.getElementById('addServiceModal'));*

*modal.hide();*

*});*

*editLinksButton.addEventListener('click', () => {*

*renderLinksList();*

*linksModal.show();*

*});*

*function renderLinksList() {*

*linksList.innerHTML = '';*

*userServices.forEach((serviceName) => {*

*const listItem = document.createElement('li');*

*listItem.classList.add('list-group-item', 'd-flex', 'justify-content-between', 'align-items-center');*

*listItem.innerHTML = `*

*<span>${serviceName}</span>*

*<button class="btn btn-sm btn-danger" onclick="deleteService('${serviceName}')">Удалить</button>*

*`;*

*linksList.appendChild(listItem);*

*});*

*}*

*function deleteService(serviceName) {*

*userServices = userServices.filter(service => service !== serviceName);*

*localStorage.setItem('userServices', JSON.stringify(userServices));*

*updateDisplay();*

*renderLinksList();*

*}*

*// Initialize the application*

*updateDisplay();*

*renderAvailableServices();*

*</script>*

*</body>*

*</html>*

*<!DOCTYPE html>*

*<html lang="en">*

*<head>*

*<meta charset="UTF-8">*

*<meta name="viewport" content="width=device-width, initial-scale=1.0">*

*<title>Login</title>*

*<link href="/static/css/styles.css" rel="stylesheet">*

*<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">*

*<link href="https://cdn.jsdelivr.net/npm/bootstrap-icons/font/bootstrap-icons.css" rel="stylesheet">*

*</head>*

*<body>*

*<div class="login-wrapper d-flex justify-content-center align-items-center vh-100">*

*<div class="login-container shadow-lg p-4 rounded" style="max-width: 400px; width: 100%; background: #fff;">*

*<img src="/static/images/tea.png" alt="Logo" class="logo mb-3 d-block mx-auto" style="width: 100px;">*

*<h2 class="text-center">Authorization</h2>*

*<p class="subtitle text-center text-muted">Log in to access your account</p>*

*<form method="POST" action="/accounts/login/" class="mt-4">*

*{% csrf\_token %}*

*<div class="input-group mb-3">*

*<span class="input-group-text"><i class="bi bi-envelope"></i></span>*

*<input*

*type="email"*

*name="email"*

*class="form-control"*

*placeholder="Email Address"*

*required*

*value="{{ email|default:'' }}"*

*>*

*</div>*

*<div class="input-group mb-3">*

*<span class="input-group-text"><i class="bi bi-lock"></i></span>*

*<input*

*type="password"*

*name="password"*

*class="form-control"*

*placeholder="Password"*

*required*

*>*

*</div>*

*{% if error\_message %}*

*<div class="alert alert-danger" role="alert">*

*{{ error\_message }}*

*</div>*

*{% endif %}*

*<button type="submit" class="btn btn-success w-100">Log In</button>*

*</form>*

*<div class="divider mt-4 mb-3 text-center">*

*<span>Don't have an account?</span>*

*</div>*

*<a href="/accounts/register/" class="btn btn-outline-success w-100">Create Account</a>*

*</div>*

*</div>*

*<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>*

*</body>*

*</html>*

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | *Дополнительные сведения* | | | | | |
|  | | | | | *Текстовые документы* |  | | | | | |
|  | | | | |  |  | | | | | |
| *БГУИР ДП 1-40 01 01 01 040 ПЗ* | | | | | *Пояснительная записка* | *89 с.* | | | | | |
|  | | | | | *Отзыв руководителя* |  | | | | | |
|  | | | | | *Рецензия* |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | | *Графические документы* |  | | | | | |
|  | | | | |  |  | | | | | |
| *ГУИР.181073.001 СА* | | | | | *Программного средства* | *Формат А1* | | | | | |
|  | | | | | *Схема алгоритма* |  | | | | | |
| *ГУИР.181073.002 СА* | | | | | *Авторизация пользователя* | *Формат А1* | | | | | |
|  | | | | | *Схема алгоритма* |  | | | | | |
| *ГУИР.181073.003 ПЛ* | | | | | *Диаграмма вариантов* | *Формат А1* | | | | | |
|  | | | | | *использования* |  | | | | | |
|  | | | | | *Плакат* |  | | | | | |
| *ГУИР.181073.004 ПЛ* | | | | | *Диаграмма IDEF0* | *Формат А1* | | | | | |
|  | | | | | *Плакат* |  | | | | | |
| *ГУИР.181073.005 ПЛ* | | | | | *Диаграмма развертывания* | *Формат А1* | | | | | |
|  | | | | | *Плакат* |  | | | | | |
| *ГУИР.181073.006 ПЛ* | | | | | *Диаграмма классов* | *Формат А1* | | | | | |
|  | | | | | *Плакат* |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  | | | | |  |  | | | | | |
|  |  |  |  |  | *БГУИР ДП 1-40 01 01 01 040 Д1* | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| *Изм.* | *Л.* | *№ докум.* | *Подп.* | *Дата* | *Программное средство автоматизации бизнес-процессов информационного отдела организации*  *Ведомость дипломного проекта* | | *Лит* | | | *Лист* | *Листов* |
| *Разраб.* | | *Готовчиц А.В.* |  |  |  | *Т* |  | *89* | *89* |
| *Провер.* | | *Корзан А.С.* |  |  | *Кафедра ИСиТ*  *гр. 181073* | | | | |
| *Т.контр.* | | *Савенко А.Г.* |  |  |
| *Н.контр.* | | *Шелягович А.С.* |  |  |
| *Утв.* | |  |  |  |

