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

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки)

Разработка программного обеспечения
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка веб-приложения для автоматизации учёта заявок в социально ориентированной организации

Обучающийся Д.В. Вострова
(Инициалы Фамилия) (личная подпись)

Руководитель к.п.н., доцент Е.А. Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Темой выпускной квалификационной работы является разработка веб-приложения для автоматизации учёта заявок в социально ориентированной организации. Объектом исследования выступает деятельность организации, связанная с приёмом, обработкой и хранением заявок от граждан.

Актуальность выбранной темы обусловлена низким уровнем цифровизации процессов в некоммерческом секторе, значительным объёмом ручного труда, риском дублирования данных и неэффективным взаимодействием с заявителями.

Цель работы – разработка веб-приложения для автоматизации обработки заявок.

В ходе выполнения работы были поставлены и решены следующие задачи: анализ предметной области и выявление функциональных требований; обоснование архитектуры системы и выбор технологий разработки; проектирование и реализация базы данных; разработка пользовательского интерфейса; обеспечение безопасности хранения данных и тестирование системы.

В работе применялись современные методы объектно-ориентированного проектирования, принципы построения клиент-серверных информационных систем, фреймворк Flask и СУБД SQLite.

Практическая значимость заключается в возможности использования разработанного приложения в деятельности социально ориентированных организаций для оптимизации процессов учёта и обработки заявок, а также при подготовке специалистов в области информационных технологий.

Выпускная квалификационная работа состоит из трёх глав, включает 5 таблиц, 26 рисунков.

Оглавление

Введение

Организации социальной направленности осуществляют деятельность, связанную с оказанием поддержки населению, реализацией программ помощи, проведением благотворительных мероприятий и координацией волонтерских проектов. Эффективность их работы определяется своевременной обработкой обращений и заявок граждан. В большинстве случаев учет заявок осуществляется вручную либо с применением разрозненных программных средств, что приводит к увеличению времени обработки информации, росту вероятности ошибок и затрудняет осуществление анализа и планирования.

Автоматизация процесса учета заявок является необходимым условием повышения производительности и прозрачности деятельности социально ориентированных организаций. Использование специализированных информационных систем способствует сокращению временных затрат на обработку обращений, снижению числа ошибок, оптимизации распределения ресурсов и повышению качества предоставляемых услуг.

Целью выпускной квалификационной работы является разработка веб-приложения, предназначенного для автоматизации учета заявок в социально ориентированной организации. Для достижения поставленной цели в работе последовательно решаются следующие задачи:

– проведение анализа предметной области и определение требований к функционалу системы;

– выбор программно-технических средств и проектирование архитектуры программного обеспечения;

– разработка структуры базы данных и пользовательского интерфейса;

– реализация веб-приложения с учетом требований к обеспечению информационной безопасности [7];

– проведение тестирования разработанной системы и оценка ее эффективности.

Объектом исследования выступает деятельность социально ориентированных организаций в части обработки входящих заявок. Предметом исследования являются методы и средства автоматизации указанного процесса с использованием современных информационных технологий [2], [15].

В работе применяются методы объектно-ориентированного проектирования, моделирования бизнес-процессов и построения клиент-серверных информационных систем.

Структура выпускной квалификационной работы включает три главы. В первой главе рассматриваются теоретические основы предметной области и анализируются существующие программные решения. Во второй главе осуществляется проектирование программного обеспечения. В третьей главе приведены результаты реализации, тестирования и анализа эффективности функционирования разработанной информационной системы.

Глава 1 Анализ предметной области и теоретические основы разработки веб-приложений

1.1 Особенности деятельности социально ориентированных организаций

Социально ориентированные организации (СОО) играют ключевую роль в современном гражданском обществе, предоставляя необходимую поддержку уязвимым слоям населения и активно работая над разрешением острых социальных проблем. В соответствии с российским законодательством, к социально ориентированным некоммерческим организациям относятся структуры, чья главная задача заключается в решении социальных задач, содействии развитию гражданского общества и осуществлении благотворительной деятельности.

Спектр деятельности этих организаций включает: оказание материальной и социальной помощи нуждающимся, многодетным семьям, людям с ограниченными возможностями и пожилым гражданам; предоставление юридических консультаций и психологической поддержки; стимулирование детского и юношеского творчества, проведение образовательных мероприятий; продвижение волонтерского движения и гражданской активности; содействие в трудоустройстве, адаптации к новым профессиям и переквалификации. Также, они реализуют благотворительные и социальные проекты в сферах образования, здравоохранения и культуры.

В быстро меняющихся социально-экономических условиях, потребность в услугах таких организаций неуклонно растет. Увеличивается число людей, обращающихся за срочной помощью и поддержкой. При этом, многие СОО испытывают нехватку ресурсов, как кадровых, так и финансовых. В сложившейся ситуации особенно важна рациональная организация работы, позволяющая сократить административные расходы и оперативно реагировать на поступающие запросы.

Несмотря на повсеместное распространение цифровых технологий, социальная сфера заметно отстает в их освоении. Социально ориентированные организации (СОО) зачастую продолжают использовать устаревшие методы работы с обращениями граждан:

– Ведение документации в бумажном виде и регистрация заявок в журналах.

– Применение электронных таблиц, например, Excel, не предназначенных для комплексной автоматизации.

– Обработка запросов через электронную почту, социальные сети и мессенджеры без централизованной системы.

Такие подходы замедляют рассмотрение обращений, повышают риск потери данных и ошибок, а также затрудняют анализ и отчетность. Отсутствие единой системы учета снижает прозрачность процессов и, как следствие, доверие граждан.

Автоматизация учета и обработки заявок способна решить большинство этих проблем. Внедрение информационных технологий в СОО позволяет:

– Ускорить обработку обращений за счет автоматизации рутинных операций и снижения объема ручного труда.

– Повысить точность обработки заявок, минимизируя человеческий фактор и вероятность ошибок.

– Обеспечить прозрачность работы организации для сотрудников и внешних заинтересованных лиц.

– Создать единую базу данных для анализа информации, что улучшает планирование и прогнозирование деятельности.

– Оптимизировать использование ресурсов организации, направляя их на решение социальных проблем.

Внедрение цифровых технологий позволяет СОО расширить охват аудитории и повысить доступность услуг, особенно для граждан, находящихся в удаленных регионах. Особенности деятельности СОО, связанные с высокой социальной ответственностью, требуют оперативной обработки информации.

Решением является создание специализированного веб-приложения для автоматизации учета заявок, которое эффективно удовлетворит потребности СОО и повысит качество услуг.

Теоретические аспекты разработки веб-приложений

На сегодняшний день для автоматизации учёта и обработки заявок применяются различные информационные системы и программные комплексы. Среди них можно выделить несколько основных категорий решений:

– системы управления задачами и проектами;

– универсальные системы электронного документооборота;

– специализированные платформы, ориентированные на нужды некоммерческих и социальных организаций.

1.2 Теоретические аспекты разработки веб-приложений

1.2.1 Системы управления задачами и проектами

Среди наиболее востребованных представителей этой категории можно выделить Trello, Jira и Asana. Их основное предназначение – облегчение командного взаимодействия, управление проектами и задачами в рамках небольших и средних рабочих групп [16]. Эти платформы характеризуются удобным интерфейсом, оперативной установкой и небольшими требованиями к обучению персонала.

Тем не менее, подобные инструменты обладают рядом существенных недостатков:

– Ограниченная гибкость в настройке структуры данных под различные виды запросов.

– Затруднения при интеграции с внешними платформами, такими как бухгалтерские программы, социальные регистры и базы данных.

– Отсутствие встроенных средств для автоматизированного создания отчетов и аналитических обзоров без дополнительной конфигурации или подключения сторонних сервисов.

Следовательно, несмотря на легкость внедрения и эксплуатации, функционал этих решений часто оказывается недостаточным для полноценного решения специфических задач, стоящих перед социально ориентированными организациями.

1.2.2 Системы автоматизации документооборота

Другим направлением, которое широко используется в различных организациях, являются универсальные платформы электронного документооборота (ЭДО), такие как Bitrix24 и 1С Документооборот. Эти системы предназначены для комплексной автоматизации внутренних процессов организации, включая обработку входящей корреспонденции, распределение задач, контроль исполнения и архивирование документов.

Достоинствами таких решений являются:

– развитый функционал управления документами и задачами;

– возможности интеграции с бухгалтерскими и другими специализированными системами (например, продуктами на платформе 1С);

– высокая степень настраиваемости под внутренние процессы организации.

Тем не менее, внедрение систем такого типа требует значительных ресурсов — как финансовых, так и временных. Организации сталкиваются со следующими сложностями:

– необходимость глубокого изменения внутренних процессов для соответствия логике работы системы;

– высокие затраты на лицензирование и поддержку системы;

– необходимость профессиональной технической поддержки и обучения персонала работе с достаточно сложным программным обеспечением.

Кроме того, избыточный функционал ЭДО-систем часто оказывается неоправданным для небольших социально ориентированных организаций, которые нуждаются в простом, удобном и доступном инструменте.

1.2.3 Специализированные платформы для некоммерческих организаций

Существуют и специализированные решения, которые ориентированы именно на потребности социальных и благотворительных организаций. К таким платформам относятся Salesforce for Nonprofits, Microsoft Dynamics 365 for Nonprofits и российская система «МойСоццентр». Эти платформы предлагают мощный функционал для работы с обращениями граждан, управления донорами, ведения учёта полученных средств и отчетности.

Основные преимущества специализированных решений:

– глубокая адаптация к специфике работы некоммерческих организаций;

– интеграция с внешними государственными сервисами и реестрами;

– возможность автоматической генерации отчетов и аналитики.

Однако такие платформы имеют следующие недостатки:

– высокая стоимость владения и значительные затраты на внедрение;

– необходимость привлечения квалифицированного IT-персонала для постоянного обслуживания и поддержки;

– сложность настройки и адаптации под конкретные процессы небольшой организации, если требуется нестандартная логика работы.

Кроме того, многие зарубежные решения плохо адаптированы под российскую специфику, включая нормативные требования и интеграцию с государственными системами.

1.2.4 Сравнительный анализ существующих решений

В таблице 1 представлен подробный сравнительный анализ рассмотренных классов программных решений по ключевым критериям оценки применительно к задачам социально ориентированных организаций.

Таблица 1 – Сравнение существующих решений по автоматизации обработки заявок

Критерий оценки	Системы управления задачами (Trello, Jira)	Системы ЭДО (Bitrix24, 1С)	Специализированные платформы (Salesforce, Dynamics 365)
Простота внедрения	Высокая	Средняя	Низкая
Стоимость владения	Низкая	Средняя-Высокая	Высокая
Гибкость настройки	Средняя	Средняя-Высокая	Высокая
Функционал учета заявок	Средний	Высокий	Очень высокий
Простота использования для сотрудников	Высокая	Средняя	Средняя-Низкая
Возможности интеграции с гос. системами	Низкая	Средняя	Средняя-Высокая
Соответствие российской специфике	Среднее	Высокое	Среднее-Низкое

Из проведённого анализа следует, что существующие решения, несмотря на свои достоинства, имеют ряд ограничений, затрудняющих их использование в небольших социально ориентированных организациях. Эти ограничения включают высокие затраты, сложность настройки и внедрения, а также отсутствие необходимой степени адаптации к российской специфике, в связи с чем актуальным решением становится создание собственного веб-приложения, полностью соответствующего специфическим задачам и процессам социально ориентированной организации, с удобным интерфейсом и доступной стоимостью внедрения и обслуживания [2], [15].

1.3 Теоретические аспекты разработки веб-приложений

Создание веб-приложений представляет собой один из самых распространённых и востребованных подходов к автоматизации различных бизнес-процессов, в том числе и в социально ориентированных организациях. Под веб-приложением понимается программное обеспечение,

функционирующее в веб-среде и доступное пользователям через браузер по сети Интернет.

1.3.1 Общие понятия и принципы веб-приложений

Веб-приложения являются одним из наиболее эффективных и широко используемых инструментов автоматизации задач и процессов в современных организациях. Под веб-приложением подразумевается программный продукт, доступ к которому пользователи получают с помощью веб-браузера по протоколам сети Интернет или локальной сети организации. Такой подход обеспечивает не только удобство работы, но и независимость от конкретного устройства и его технических характеристик [6].

Веб-приложения характеризуются следующими ключевыми признаками:

– Кроссплатформенность – способность приложения работать на любых операционных системах и устройствах, которые поддерживают работу с веб-браузерами (Windows, macOS, Linux, Android, iOS и др.);

– Отсутствие необходимости установки специализированного программного обеспечения – для использования веб-приложения пользователю достаточно иметь стандартный браузер, что существенно облегчает распространение и эксплуатацию программного обеспечения;

– Централизованность данных – все пользовательские данные и настройки хранятся и обрабатываются на сервере, что обеспечивает их доступность с разных устройств и в любое время;

– Возможность совместного использования данных и одновременной работы нескольких пользователей с одними и теми же ресурсами.

Преимущества веб-приложений

Использование веб-приложений имеет ряд значительных преимуществ по сравнению с традиционными десктопными решениями:

– Удобство распространения – приложение не нужно устанавливать отдельно на каждое устройство пользователя, что экономит время и ресурсы организации;

– Масштабируемость и гибкость – при росте количества пользователей или расширении функционала системы приложение легко масштабируется благодаря централизованному серверу и гибкости архитектуры;

– Единая версия приложения – все пользователи получают доступ к единой версии системы, а любые изменения или исправления мгновенно становятся доступны всем пользователям без необходимости обновления на стороне клиента;

– Экономическая эффективность – веб-приложения не требуют лицензирования на каждое рабочее место и снижают затраты на техническую поддержку;

– Удобство администрирования и поддержки – за счёт централизованного хранения данных и приложений администратор легко может контролировать доступы, безопасность и обновления.

Требования к современным веб-приложениям

Для обеспечения высокого качества работы и удовлетворения современных пользовательских ожиданий веб-приложения должны соответствовать следующим требованиям:

– Производительность и скорость работы – приложение должно быстро реагировать на запросы пользователя, минимизировать время ожидания ответа и обеспечивать высокую отзывчивость интерфейса;

– Надёжность и отказоустойчивость – система должна стабильно функционировать в условиях высокой нагрузки и быстро восстанавливаться после возможных сбоев;

– Информационная безопасность – необходимо гарантировать защиту конфиденциальности пользовательских данных и устойчивость системы к внешним угрозам;

– Адаптивность и удобство пользовательского интерфейса – интерфейс должен корректно отображаться на различных устройствах и экранах

(десктопных, планшетах, смартфонах) [8], обеспечивая удобство работы для широкого круга пользователей;

– Лёгкость обучения и использования – пользователи должны легко осваивать работу с системой без необходимости специальной подготовки.

Таким образом, использование веб-приложений для автоматизации деятельности социально ориентированных организаций обеспечивает не только удобство эксплуатации и доступность, но и экономическую эффективность, надёжность и безопасность обработки данных. Данные преимущества делают выбор именно этого формата реализации наиболее целесообразным для поставленной в дипломной работе задачи.

1.3.2 Клиент-серверная архитектура веб-приложений

Клиент-серверная архитектура веб-приложений включает в себя два основных компонента:

– Клиентская часть (Frontend) – интерфейс пользователя, реализуемый с помощью HTML, CSS и JavaScript-фреймворков (React, Angular, Vue.js). Клиентская часть обеспечивает взаимодействие пользователя с приложением, получение и отправку данных на сервер и отображение информации.

– Серверная часть (Backend) – логика обработки данных, авторизация и аутентификация пользователей, взаимодействие с базами данных, генерация отчётов и другие процессы. Серверная часть реализуется на различных языках программирования и фреймворках: Python (Django, Flask), JavaScript (Node.js, Express.js), PHP (Laravel, Symfony), Ruby (Ruby on Rails).

1.3.3 Базы данных для веб-приложений

Эффективное функционирование веб-приложения напрямую зависит от выбранной базы данных и организации её работы. База данных представляет собой структурированное хранилище информации, предназначенное для надежного хранения, поиска и обработки данных в рамках работы информационных систем. В контексте веб-приложений выбор базы данных является критически важным этапом, так как именно от нее зависит производительность системы, целостность информации, безопасность и

возможность её дальнейшего масштабирования.

Современные базы данных для веб-приложений разделяются на две основные группы:

– Реляционные базы данных (SQL)

– Нереляционные базы данных (NoSQL)

Реляционные базы данных являются наиболее распространённым типом баз данных, в которых данные хранятся в виде взаимосвязанных таблиц. Связь между таблицами осуществляется с помощью уникальных идентификаторов (первичных и внешних ключей) [4]. Для взаимодействия с такими базами используется язык структурированных запросов SQL (Structured Query Language).

Примеры популярных реляционных баз данных:

– PostgreSQL — мощная и производительная открытая СУБД, поддерживающая сложные запросы и транзакции, имеет широкие возможности по оптимизации и масштабированию [13].

– MySQL — широко распространенная и простая в использовании СУБД, подходит для небольших и средних проектов.

– Oracle Database — коммерческая СУБД с расширенными возможностями масштабирования и безопасности, применяется преимущественно в крупных корпоративных системах.

– Microsoft SQL Server — СУБД от компании Microsoft, интегрированная с другими продуктами компании, используется в корпоративных приложениях на платформе Windows.

Преимущества реляционных баз данных:

– строгая структура данных, обеспечивающая целостность и надежность информации;

– поддержка транзакционной модели (ACID-принципы: атомарность, согласованность, изоляция и долговечность);

– высокая производительность при работе с четко структурированными данными и сложными запросами;

– развитые средства аналитики и формирования отчетности;

– высокая степень стандартизации и совместимости с большинством инструментов и платформ разработки.

Недостатки реляционных баз данных:

– сложность изменения структуры данных после начала эксплуатации;

–необходимость в сложных процедурах миграции данных при изменении схемы базы данных;

– ограниченные возможности горизонтального масштабирования.

Нереляционные базы данных (NoSQL) хранят информацию в нереляционной форме и не используют SQL-запросы как основной инструмент взаимодействия с данными. Они появились как ответ на возросшие потребности веб-приложений в гибкости, скорости и обработке больших объемов плохо структурированной информации.

Наиболее распространённые нереляционные базы данных:

– MongoDB — документоориентированная база данных, в которой данные хранятся в виде документов формата JSON. MongoDB обеспечивает высокую производительность и легкость горизонтального масштабирования [3].

– Redis — база данных в памяти, работающая по принципу ключ-значение. Используется для кеширования данных и повышения производительности приложений.

– Cassandra — высокопроизводительная распределённая база данных, способная выдерживать огромные нагрузки и большое количество параллельных запросов.

Преимущества нереляционных баз данных:

– гибкость структуры данных, возможность оперативно менять схемы хранения информации;

– высокая скорость работы с большими объемами данных;

– простота горизонтального масштабирования, когда добавление новых серверов существенно повышает производительность;

– высокая производительность при простых типах запросов и использовании

баз данных в реальном времени.

Недостатки нереляционных баз данных:

- отсутствие строгой схемы данных может привести к избыточности и дублированию информации;
- ограниченные возможности для выполнения сложных аналитических запросов;
- меньшие гарантии целостности данных по сравнению с реляционными системами

Реляционные базы данных являются наиболее распространённым типом баз данных, в которых данные хранятся в виде взаимосвязанных таблиц. Связь между таблицами осуществляется с помощью уникальных идентификаторов (первичных и внешних ключей). Для взаимодействия с такими базами используется язык структурированных запросов SQL (Structured Query Language).

Для данного проекта по автоматизации учёта и обработки заявок в социально ориентированной организации наибольшее значение имеют надёжность, простота развертывания и эксплуатации, а также совместимость с используемым стеком технологий [10], [11], [18]. На основании этих требований, а также с учетом анализа особенностей разработки на Python с использованием микрофреймворка Flask, было принято решение использовать в качестве базы данных SQLite.

SQLite представляет собой компактную встраиваемую базу данных с открытым исходным кодом [1], [9], [21]. Особенностью SQLite является то, что она не требует отдельного процесса сервера и сохраняет данные в виде файла непосредственно в файловой системе операционной системы.

Данный выбор обусловлен следующими преимуществами. Простота развертывания и использования - SQLite не требует отдельного процесса базы данных, дополнительного администрирования или сложной настройки, что значительно облегчает развёртывание и поддержку приложения. Легковесность и компактность - данная СУБД отличается минимальными

требованиями к ресурсам системы, что позволяет эффективно использовать её даже на маломощных серверах или устройствах с ограниченными аппаратными ресурсами. Высокая совместимость с фреймворком Flask - SQLite широко используется в проектах, реализованных на Flask, благодаря простоте интеграции и поддержке из коробки. Flask, совместно с библиотекой SQLAlchemy, позволяет эффективно управлять данными и взаимодействовать с SQLite, обеспечивая удобный инструмент для работы с базой данных. Отсутствие необходимости настройки и администрирования - SQLite полностью избавляет разработчиков и пользователей от необходимости конфигурирования или администрирования базы данных. Это является важным преимуществом для небольших социальных организаций, не имеющих выделенного IT-персонала. Поддержка ACID-транзакций и стандарта SQL - несмотря на простоту и компактность, SQLite обеспечивает соблюдение ACID-принципов (атомарность, согласованность, изоляция и долговечность транзакций), что гарантирует надежность и целостность данных [10], [9], [2].

В то же время SQLite обладает рядом ограничений, о которых следует помнить:

– ограниченные возможности горизонтального масштабирования и одновременного использования большим количеством пользователей;
– не подходит для высоконагруженных проектов с большим числом параллельных соединений и огромными объемами данных.

Тем не менее, для целей разрабатываемого веб-приложения, предназначенного для автоматизации процессов учёта и обработки заявок в социально ориентированной организации, SQLite является оптимальным выбором. Она полностью удовлетворяет требованиям по простоте использования, стабильности, безопасности и интеграции с фреймворком Flask.

Таким образом, выбор SQLite в качестве базы данных обеспечивает необходимый уровень надежности, простоту внедрения и эффективную

поддержку системы автоматизированного учёта заявок, что позволит значительно упростить эксплуатацию и поддержку веб-приложения на стороне конечных пользователей.

1.3.4 Стек технологий для веб-приложений

Стек технологий — это совокупность программных средств, используемых для разработки и функционирования веб-приложений. Наиболее распространённые стеки включают LAMP, MEAN/MERN и Python-ориентированные решения.

Стек LAMP (Linux, Apache, MySQL, PHP) традиционно применяется для разработки классических серверных веб-приложений. Его отличают простота установки, надёжность и невысокие требования к инфраструктуре. Вместе с тем, для современных динамичных и масштабируемых систем LAMP может быть менее гибким.

Современные проекты часто используют JavaScript-стек MEAN или MERN, объединяющий MongoDB, Express, Angular или React, а также Node.js. Применение единого языка программирования на стороне клиента и сервера позволяет ускорить разработку и упростить сопровождение кода. Данный стек оптимален для создания высокоинтерактивных и быстро развиваемых веб-систем, однако требует постоянного обновления компетенций в быстро меняющейся экосистеме.

Отдельное место занимает Python-стек, включающий язык программирования Python, один из популярных фреймворков (Flask или Django), а также базы данных PostgreSQL или SQLite. Такой подход отличается простотой освоения, возможностью быстрой реализации типовых функций и гибкостью архитектурных решений. Flask выбирают для небольших и средних проектов, Django — для крупных и комплексных систем. В качестве СУБД SQLite применяется для локальных или тестовых сред, PostgreSQL — для более сложных задач, требующих высокой надёжности и масштабируемости.

Выбор технологического стека зависит от специфики проекта: предполагаемой нагрузки, требований к масштабированию, доступных ресурсов и опыта команды. Каждый из перечисленных стеков имеет свои преимущества и ограничения, поэтому для эффективной реализации веб-приложения необходимо учитывать цели и задачи разрабатываемой информационной системы.

1.3.5 Обоснование выбора технологий для разработки приложения

Выбор стека технологий для реализации веб-приложения по автоматизации учёта заявок в социально ориентированной организации осуществлялся с учётом нескольких ключевых факторов: простоты внедрения, минимальных затрат на обслуживание, лёгкости поддержки, а также соответствия требованиям надёжности, безопасности и доступности. Основной акцент был сделан на применение технологий с открытым исходным кодом и низким порогом входа, что особенно важно в условиях ограниченных ресурсов, характерных для небольших некоммерческих организаций.

На основе анализа возможных решений был выбран следующий стек:

– язык программирования Python;

– микрофреймворк Flask;

– встроенная реляционная база данных SQLite;

–стандартные средства для клиентской части: HTML, CSS, JavaScript.

В качестве языка программирования был выбран Python, потому что:

- он обладает лаконичным синтаксисом, что ускоряет разработку и повышает читаемость кода;
- является универсальным и широко применимым в различных сферах, от веб-разработки до анализа данных и автоматизации;
- имеет обширную экосистему готовых библиотек и фреймворков, что позволяет быстро реализовывать даже сложные функции без необходимости писать всё с нуля;
- активно поддерживается сообществом, что обеспечивает наличие

документации, обучающих материалов и готовых решений для типовых задач;

- полностью интегрируется с необходимыми инструментами для работы с базами данных, веб-интерфейсами и безопасностью.

В качестве веб-фреймворка был выбран Flask, потому что:

- он предоставляет минимальный, но гибкий набор инструментов, необходимый для разработки веб-приложений, не навязывая строгую архитектуру;
- позволяет самостоятельно формировать структуру проекта, адаптируя её под конкретные требования предметной области;
- отличается лёгкостью освоения и высокой скоростью начального развертывания проекта;
- хорошо совместим с расширениями, такими как SQLAlchemy (ORM), Jinja2 (шаблонизатор), WTForms (обработка форм), что упрощает реализацию стандартных веб-функций;
- –предоставляет средства для построения REST API, что делает приложение масштабируемым и готовым к последующей интеграции с мобильными или внешними системами.

В качестве базы данных была выбрана SQLite, потому что:

- она идеально подходит для малонагруженных приложений и проектов с локальной или ограниченной многопользовательской эксплуатацией;
- не требует отдельного сервера и сложной настройки;
- легко интегрируется с Flask через SQLAlchemy и поддерживается «из коробки»;
- обеспечивает полную поддержку транзакций и SQL-совместимость;
- –позволяет быстро развернуть систему даже в условиях отсутствия специализированного IT-персонала;
- предоставляет достаточно функциональности для обработки заявок, учёта их статусов и формирования отчётности.

В случае масштабирования проекта возможен переход на PostgreSQL без кардинальных изменений в архитектуре системы.

Для клиентской части были выбраны HTML, CSS, JavaScript, потому что:

- они являются базовыми и кроссплатформенными технологиями, поддерживаемыми всеми современными браузерами;
- позволяют создавать адаптивный, понятный и доступный интерфейс для различных категорий пользователей;
- обеспечивают полную совместимость с Flask и возможностью внедрения Jinja2-шаблонов.

Таким образом, стек Python + Flask + SQLite + HTML/CSS/JavaScript был выбран как оптимальное решение, сочетающее в себе:

- простоту разработки и сопровождения;
- низкие требования к инфраструктуре;
- гибкость архитектурных решений;
- достаточную функциональность для автоматизации задач учёта и обработки заявок;
- –открытость и отсутствие лицензионных затрат.

Данный набор технологий идеально отвечает потребностям социально ориентированных организаций и позволяет реализовать устойчивую, удобную и расширяемую информационную систему в кратчайшие сроки и с минимальными затратами.

1.3.6. Обеспечение информационной безопасности в веб-приложениях

Информационная безопасность (ИБ) является неотъемлемым элементом проектирования и реализации любых веб-приложений, особенно в случае, если они работают с персональными или конфиденциальными данными. В контексте социально ориентированных организаций это может включать информацию о заявителях, сотрудниках, типах оказываемой помощи и другой чувствительной информации, утечка которой способна нанести ущерб

репутации и привести к нарушению законодательства.

Веб-приложения, как правило, являются уязвимыми для целого класса угроз, включая сетевые атаки, манипуляции с формами, несанкционированный доступ и вмешательство в структуру базы данных. Поэтому обеспечение защиты данных — не факультативная задача, а обязательное требование к качественной информационной системе.

Основные аспекты обеспечения ИБ в веб-приложении. Безопасность веб-приложений обеспечивается комплексом мер, охватывающих как архитектурные решения, так и конкретные реализации на уровне кода. В рамках реализации проекта на основе Flask и SQLite предусмотрен комплекс мер по обеспечению безопасности системы. Аутентификация и авторизация пользователей осуществляются с использованием библиотек Flask-Login и Flask-Security, что позволяет надёжно идентифицировать пользователей и разграничивать их права доступа [7], [10]. Для повышения безопасности учетных данных пароли сохраняются в зашифрованном виде, что исключает возможность их восстановления в случае компрометации базы данных.

Передача данных между клиентской и серверной частями приложения осуществляется по защищённому протоколу HTTPS с применением SSL-сертификатов, что обеспечивает конфиденциальность информации и предотвращает её перехват в процессе обмена [7], [19].

Для защиты от распространённых видов атак реализован ряд технических решений. Защита от CSRF-атак обеспечивается путём валидации специальных CSRF-токенов, встраиваемых в каждую форму. От атак XSS система защищена посредством экранирования всех пользовательских данных средствами шаблонизатора Jinja2. Для предотвращения SQL-инъекций в проекте применяется объектно-реляционное отображение SQLAlchemy, исключающее прямую подстановку пользовательских данных в SQL-запросы.

Организовано безопасное управление сессиями и cookie-файлами. Все пользовательские сессии шифруются, а для cookie-файлов заданы флаги безопасности HttpOnly и Secure. Кроме того, установлено ограничение

времени жизни сессий, что минимизирует риски при несанкционированном доступе к устройствам пользователей.

Для предотвращения перебора паролей предусмотрены механизмы блокировки или задержки после нескольких неудачных попыток входа в систему [12]. При необходимости может быть интегрирована дополнительная защита с помощью CAPTCHA. Также реализован контроль входа пользователей в систему и ведение журналов событий, что позволяет фиксировать основные действия и своевременно выявлять подозрительную активность.

Особое внимание уделено ограничению прав доступа к файловой системе сервера: приложение развёртывается с минимально необходимыми привилегиями, а все загружаемые файлы проходят обязательную валидацию путей и форматов. Для обеспечения сохранности данных регулярно выполняется резервное копирование базы данных с последующим хранением копий на внешних носителях или в облачных хранилищах. Кроме того, в целях защиты от известных уязвимостей поддерживается актуальность используемых зависимостей: все программные библиотеки и компоненты системы регулярно обновляются до последних стабильных версий.

Выводы по главе 1.

В первой главе была проведена всесторонняя работа по анализу предметной области, в рамках которой подробно рассмотрены особенности функционирования социально ориентированных организаций (СОО), а также существующие подходы и программные решения, применяемые для автоматизации процессов учета заявок в подобных структурах. Были проанализированы актуальные программные комплексы, включающие в себя системы управления задачами, электронный документооборот, корпоративные информационные системы и специализированные программные продукты, представленные как отечественными, так и зарубежными разработчиками.

Результаты проведённого анализа показали, что большинство

существующих решений имеют ряд ограничений при применении в условиях небольших некоммерческих организаций. Кроме того, часть существующих решений ориентирована на крупные корпоративные структуры и не учитывает специфики деятельности организаций, работающих в социальной сфере и обладающих ограниченными финансовыми и техническими ресурсами. Данная ситуация обусловливает актуальность разработки специализированного решения, ориентированного на небольшие организации, которому характерны простота эксплуатации, доступность внедрения, минимальные затраты на сопровождение и соответствие ключевым требованиям СОО.

В теоретической части главы было обосновано применение современных технологий, обеспечивающих эффективную разработку целевого программного продукта. В качестве языка программирования выбран Python, обладающий высокой гибкостью, богатым набором библиотек и активным сообществом разработчиков. Для реализации серверной части выбран легковесный и производительный веб-фреймворк Flask, обеспечивающий быструю разработку, простоту настройки и масштабируемость проекта. В качестве системы управления базами данных используется SQLite, обеспечивающая простоту развёртывания, отсутствие необходимости в отдельном сервере базы данных и достаточную производительность при работе с небольшими объёмами данных, что оптимально соответствует требованиям разрабатываемого проекта.

Таким образом, проведённый анализ предметной области, существующих решений и доступных технологий позволил сформулировать требования к разрабатываемой системе, определить её архитектуру, функциональные и технические характеристики, а также заложил основу для проектирования и последующей реализации веб-приложения, максимально соответствующего задачам социально ориентированных организаций.

Глава 2 Проектирование веб-приложения для учета заявок

2.1 Постановка задачи и функциональные требования

Целью разрабатываемого проекта является создание специализированного веб-приложения, предназначенного для автоматизации процесса учёта заявок в социально ориентированной организации. При разработке системы особое внимание уделяется удобству эксплуатации, интуитивно понятному интерфейсу, простоте взаимодействия с системой со стороны конечных пользователей, а также доступности функционала для сотрудников, не обладающих специальной технической подготовкой.

Разработка предполагает внедрение целого ряда функциональных возможностей, обеспечивающих полную автоматизацию процесса обработки заявок. Каждая заявка, регистрируемая в системе, должна автоматически получать уникальный идентификатор. Формат нумерации заявок представляет собой последовательность вида SOO001, SOO002 и так далее. Присвоенный номер должен быть уникальным и неизменяемым на всём протяжении жизненного цикла соответствующей записи, что обеспечивает надёжную идентификацию и последующий учёт обращений. Данная особенность позволит организовать точный контроль количества и порядка обработки заявок, упростить отслеживание и систематизацию данных в дальнейшем.

Система должна обеспечивать авторизованным пользователям, а также администраторам доступ к просмотру, обработке и фильтрации заявок. Пользователи, имеющие доступ к системе, должны иметь возможность просматривать все зарегистрированные обращения, выполнять их обработку в зависимости от предоставленных прав доступа, а также осуществлять фильтрацию данных по различным критериям, что существенно повышает удобство работы с системой в случае большого объёма информации. При отображении информации о каждой заявке в системе должны отображаться её уникальный номер, краткое и полное описание, а также текущий статус.

Особое внимание при проектировании функциональной части системы уделяется отображению актуального статуса каждой заявки. В рамках функционирования приложения предусматривается несколько фиксированных статусов заявок, таких как «новая», «в работе», «выполнена» и «отклонена». Это позволяет участникам процесса оперативно ориентироваться в состоянии обработки каждого отдельного обращения, а администраторам — эффективно организовывать работу с входящим потоком заявок.

Требования к разрабатываемой системе включают также реализацию стабильной работы с базой данных, в качестве которой используется SQLite. Такой выбор обусловлен простотой развертывания, отсутствием необходимости в сложной настройке и внешних зависимостях, а также достаточной производительностью при обработке заявок в небольших организациях. Данные, сохраняемые в базе, должны надёжно сохраняться на сервере и быть доступны для поиска и дальнейшей обработки.

Интерфейс приложения разрабатывается таким образом, чтобы обеспечить его доступность для пользователей, не обладающих специализированными знаниями в области информационных технологий. Взаимодействие с системой должно быть максимально упрощено, с понятной навигацией и минимальным числом действий, необходимых для выполнения операций.

Существенной частью проектирования является обеспечение устойчивости работы приложения при возникновении ошибок ввода и предотвращении некорректных действий со стороны пользователя. В процессе реализации разрабатываются механизмы обработки исключительных ситуаций и валидации вводимых данных, которые позволяют минимизировать вероятность сбоев. Кроме того, важным требованием к системе является защита от наиболее распространённых видов атак, в частности, от SQL-инъекций. Для этого в проекте применяются современные методы защиты и использования безопасных средств взаимодействия с базой данных,

позволяющих исключить непосредственную подстановку пользовательских данных в SQL-запросы.

Таким образом, сформулированные требования обеспечивают создание надёжного, безопасного и функционально полного программного продукта, ориентированного на стабильную работу в условиях ограниченных ресурсов социально ориентированных организаций, а также соответствующего современным стандартам разработки информационных систем.

2.2 Проектирование структуры базы данных

В разрабатываемом веб-приложении для хранения информации о пользователях и заявках используется встроенная реляционная система управления базами данных SQLite. Данный выбор обоснован особенностями проекта: система не требует установки отдельного серверного программного обеспечения, обладает высокой производительностью при работе с ограниченными объемами данных и обеспечивает простоту развёртывания, что особенно важно для небольших социально ориентированных организаций [14], [13]. Немаловажным преимуществом SQLite является его полная интеграция с используемым фреймворком Flask, что упрощает разработку серверной части приложения и обеспечивает стабильное взаимодействие между приложением и базой данных.

Проектируемая структура базы данных включает две основные таблицы: User и Application. Таблица User предназначена для хранения данных о пользователях системы. Каждая запись в данной таблице содержит уникальный идентификатор пользователя, логин, зашифрованный пароль и специальное поле, определяющее уровень доступа пользователя. Для реализации разграничения прав доступа в системе предусмотрено использование логического поля is_admin, на основании которого определяется роль пользователя. Администраторы обладают расширенными полномочиями по управлению системой, в том числе правами на управление

учетными записями пользователей и выполнение операций с заявками, которые недоступны рядовым пользователям.

Таблица Application предназначена для хранения сведений о заявках, регистрируемых в системе. Каждая заявка создаётся только авторизованным пользователем, при этом для фиксации автора заявки используется обязательное поле user_id, представляющее собой внешний ключ, ссылающийся на таблицу User. При создании каждой новой заявки системе автоматически присваивается уникальный номер, формируемый в поле app_number. После создания заявки присвоенный номер не подлежит изменению, что обеспечивает однозначную идентификацию каждой записи в системе.

Отдельное внимание уделяется хранению статуса каждой заявки. Для отражения текущего состояния обработки обращения в системе предусмотрено поле status, в котором фиксируется этап работы с заявкой: например, «новая», «в работе», «выполнена» или «отклонена» [3]. Характерной особенностью реализации является то, что список допустимых статусов определён логикой работы приложения и реализован в программной части системы, без выделения отдельной таблицы справочника статусов в базе данных. Такой подход позволяет обеспечить необходимую гибкость при сохранении простой структуры базы данных.

Таким образом, разработанная структура базы данных соответствует функциональным требованиям системы, обеспечивает стабильное и надёжное хранение данных, а также поддерживает механизмы авторизации, идентификации пользователей и разграничения их прав доступа в процессе работы с заявками.

2.3 Проектирование пользовательского интерфейса

Одной из ключевых задач, решаемых в процессе разработки веб-приложения для учёта заявок, является создание интуитивно понятного,

доступного и удобного пользовательского интерфейса. Основной акцент при проектировании интерфейса сделан на простоту освоения и минимальные требования к техническим навыкам конечных пользователей, что особенно актуально для сотрудников социально ориентированных организаций, где уровень подготовки персонала может значительно варьироваться. Интерфейс системы должен обеспечивать лёгкий и быстрый доступ ко всем основным функциям приложения, сводя к минимуму время, необходимое для обучения работе с системой.

В процессе проектирования пользовательского интерфейса были применены современные принципы разработки, обеспечивающие высокую степень удобства и функциональности. Основное внимание уделялось обеспечению простоты и ясности взаимодействия: все элементы управления расположены логично и интуитивно понятны, ключевые функции доступны пользователю буквально в один-два клика. Навигация организована таким образом, чтобы пользователь мог легко перемещаться между основными разделами системы, независимо от уровня своей подготовки. Интерфейс адаптирован к ролям пользователей: для операторов доступен базовый набор функций, включающий работу с заявками, тогда как для администраторов предусмотрен расширенный функционал, включающий управление учетными записями пользователей и настройку системы.

Особая роль в проектировании интерфейса отводится организации рабочих экранов приложения. При входе в систему пользователю предоставляется страница авторизации, на которой необходимо ввести логин и пароль для доступа к основным функциям. В случае ошибки входа система предоставляет пользователю понятные сообщения о причине неудачной попытки, что позволяет оперативно скорректировать вводимые данные. После успешной авторизации пользователь попадает на главную панель управления системой, где представлена краткая сводная информация о текущем состоянии заявок. Здесь отображаются данные о количестве заявок в различных статусах: новых, находящихся в работе, завершённых или отклонённых, что позволяет

получить общее представление о текущей загрузке системы и оперативно перейти к нужному разделу.

Основная часть работы с заявками осуществляется через интерфейс списка заявок, представленного в табличной форме. Здесь пользователь может просматривать полную информацию по зарегистрированным обращениям, осуществлять фильтрацию записей по различным статусам, а также выполнять сортировку данных. Для администраторов в данной таблице дополнительно отображаются элементы управления, позволяющие выполнять редактирование и удаление заявок, что обеспечивает гибкость управления данными.

Создание новых заявок и их редактирование осуществляется посредством специальной формы, включающей необходимые для заполнения поля. Пользователь указывает фамилию, имя и отчество заявителя, описание обращения и текущий статус заявки. Для обязательных к заполнению полей предусмотрены визуальные индикаторы, помогающие избежать пропуска важных данных. После успешного сохранения данных система выводит соответствующее уведомление, подтверждающее корректное выполнение операции.

Для администраторов системы дополнительно реализован раздел управления пользователями. В данном разделе отображается список зарегистрированных пользователей системы, предоставляется возможность добавления новых учетных записей с назначением соответствующей роли, а также осуществляется редактирование или удаление учетных записей при необходимости.

Разработка пользовательского интерфейса велась с использованием современных веб-технологий. В качестве основного инструментария для разметки страниц применялись технологии HTML5 и CSS3, обеспечивающие корректное отображение интерфейса в различных браузерах и на различных устройствах [17]. Для стилизации и построения адаптивной сетки был использован фреймворк Bootstrap, предоставляющий обширный набор

готовых компонентов и шаблонов оформления. Динамические элементы интерфейса, а также валидация форм на стороне клиента реализованы с помощью языка JavaScript. Для организации шаблонов страниц при работе с серверной частью Flask применялась система шаблонизации Jinja2, позволяющая эффективно комбинировать программную логику и разметку страниц.

Таким образом, спроектированный пользовательский интерфейс обеспечивает лёгкость и удобство взаимодействия пользователей с системой, минимизируя время на обучение и обеспечивая эффективную работу даже при минимальном уровне технической подготовки персонала.

Выводы по главе 2

В ходе проектирования системы были определены основные требования к функционалу и безопасности веб-приложения, а также спроектирована его структура. Разработана логическая и физическая модели данных, отражающие ключевые сущности предметной области и их взаимосвязи. Особое внимание уделено обеспечению уникальной идентификации заявок, организации структуры пользовательских ролей и разграничению прав доступа. Проектирование пользовательского интерфейса выполнено с учетом специфики социально ориентированных организаций и направлено на создание интуитивно понятной, доступной и функциональной среды для конечных пользователей. В качестве программно-технической основы системы выбраны проверенные технологии Python, Flask и SQLite, что обеспечивает стабильную работу приложения, простоту сопровождения и возможность дальнейшего масштабирования. Сформированные проектные решения стали основой для последующей реализации программного продукта.

Глава 3 Реализация и тестирование веб-приложения

3.1 Выбор технологий и архитектуры

Для реализации веб-приложения по автоматизации учёта заявок была выбрана современная и надёжная технология — стек на основе Python и Flask. Такой выбор обоснован следующими причинами:

– Python — популярный язык программирования, хорошо подходящий для быстрой разработки и поддерживаемый большим количеством библиотек.

– Flask — легковесный веб-фреймворк для Python, который позволяет быстро создавать масштабируемые и поддерживаемые веб-приложения, а также легко интегрировать дополнительные модули (например, авторизацию, работу с базой данных).

– SQLAlchemy — ORM-библиотека для работы с базой данных, даёт удобный объектно-ориентированный интерфейс для операций с БД и обеспечивает независимость от СУБД.

– SQLite — простая и встроенная база данных, не требующая отдельного сервера, отлично подходит для данного проекта, полностью покрывает задачи хранения заявок и пользователей.

– Flask-Login — библиотека для простой и безопасной реализации авторизации и управления сессиями пользователей.

– Bootstrap 5 — популярный CSS-фреймворк, используемый для быстрой верстки адаптивного и современного интерфейса.

Структура приложения соответствует классической архитектуре MVC (Model–View–Controller), что позволяет разделить логику работы с данными, обработку маршрутов и отображение пользовательского интерфейса.

Для эффективной разработки проекта использовались следующие инструменты:

– PyCharm — интегрированная среда разработки для Python, предоставляющая все необходимые средства для работы с кодом, запуска и отладки веб-приложений, управления зависимостями и контроля версий.

– DBeaver — современный инструмент для визуального управления базами данных, который применялся для просмотра структуры и содержимого таблиц SQLite, выполнения SQL-запросов и резервного копирования данных.

Использование данных инструментов значительно ускорило процесс реализации приложения, упростило отладку, визуализацию архитектуры и контроль работы базы данных.

Структура каталогов и файлов проекта представлена в таблице 2 и на рисунке 1.

Таблица 2 – Структура каталогов и файлов проекта

Файл/папка	Назначение
run.py	Точка входа, запуск Flask-приложения
config.py	Конфигурация (секретный ключ, строка БД и т.д.)
app/__init__.py	Создание экземпляра приложения, инициализация БД
app/models.py	Описание моделей (User, Application)
app/routes.py	Реализация маршрутов (URL, обработчики)
app/templates/	HTML-шаблоны (интерфейс)
app/static/	CSS-стили, JS и изображения

Таблица описывает структуру проекта, включая ключевые файлы и папки, их расположение и функциональное назначение. Это позволяет разработчикам быстро ориентироваться в проекте и понимать логику его организации.

На рисунке 1 представлена структура каталогов и файлов проекта веб-приложения для автоматизации учёта заявок.

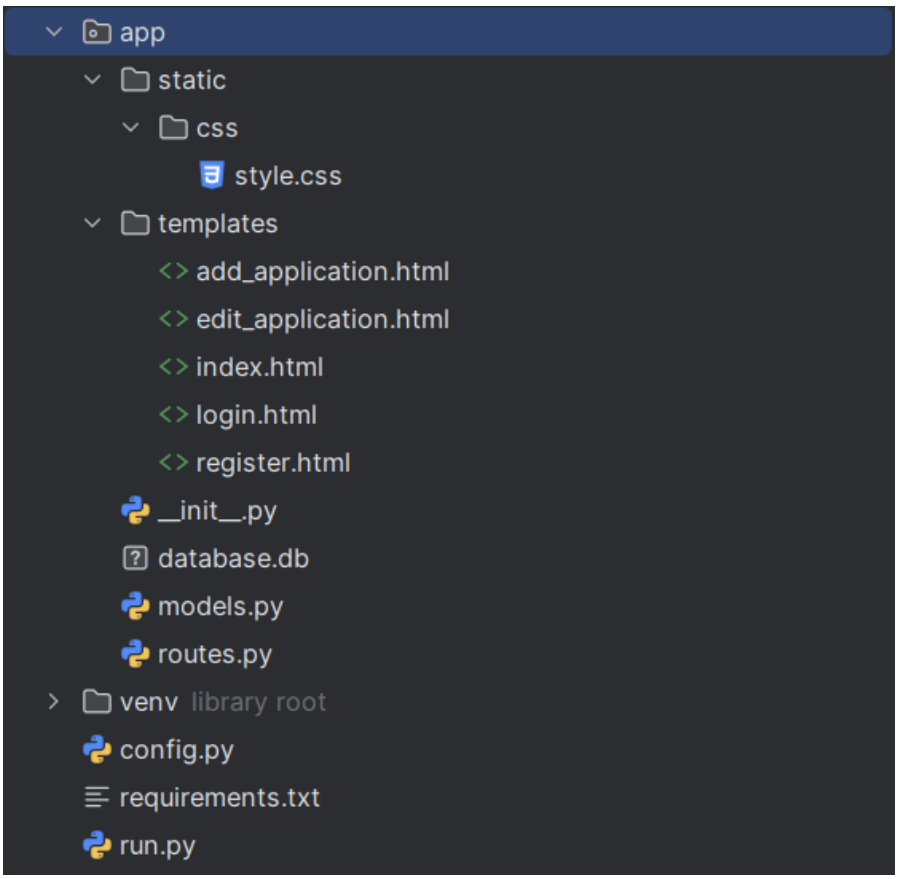

Рисунок 1 – Структура каталогов и файлов проекта

Структура проекта организована в соответствии с архитектурой MVC (Model-View-Controller), что обеспечивает чёткое разделение логики данных, представления и управления. Это упрощает поддержку и масштабируемость приложения.

3.2 Структура и проектирование базы данных

Для хранения информации о пользователях и заявках в веб-приложении используется реляционная база данных SQLite.

Данная СУБД выбрана за простоту внедрения, отсутствие необходимости в отдельном сервере и широкую поддержку в экосистеме Python.

В проекте реализованы две основные сущности:

– пользователь (User), описание представлено в таблице 3;

– заявка (Application), описание представлено в таблице 4.

Визуальный пример таблицы представлен на рисунке 5.

Связь между таблицами реализована по принципу “один-ко-многим”: один пользователь может создать много заявок, каждая заявка принадлежит только одному пользователю.

ER-диаграмма базы данных приложения представлена на рисунке 2.

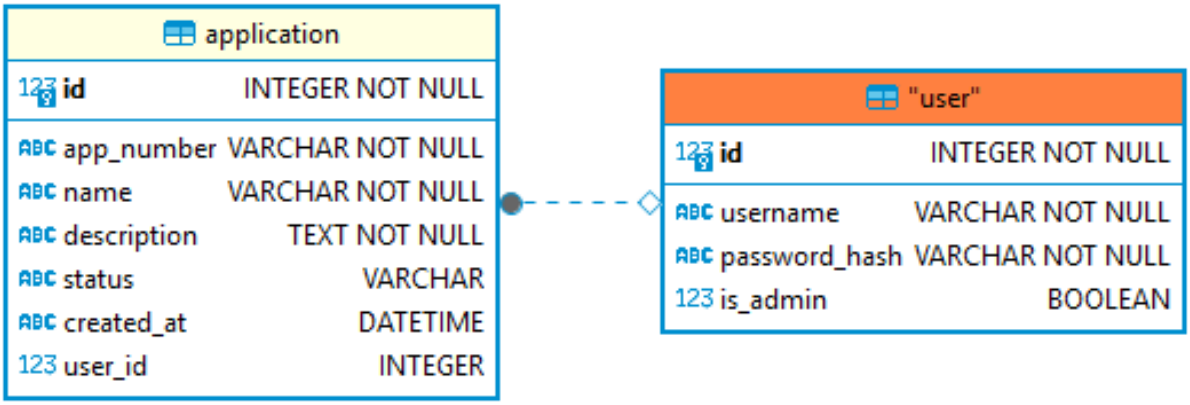

Рисунок 2 – ER-диаграмма базы данных приложения

На диаграмме представлены основные сущности базы данных веб-приложения для учёта заявок: User (пользователь) и Application (заявка). Отображены их атрибуты, включая уникальные идентификаторы, имена, хэшированные пароли, статусы заявок и даты создания. Между сущностями установлена связь «один ко многим» по внешнему ключу user_id, обеспечивающая привязку каждой заявки к конкретному пользователю.

Таблица 3 – Описание таблицы User

Поле	Тип	Описание
id	Integer	Уникальный идентификатор (PK)
username	String	Имя пользователя
password_hash	String	Хеш пароля
is_admin	Boolean	Признак администратора

В таблице 3 приведена структура сущности User, описывающей учетные записи пользователей веб-приложения. Каждый пользователь имеет уникальный идентификатор id, имя пользователя username, зашифрованный пароль password_hash и признак административных прав is_admin. Поле is_admin определяет доступ к расширенным функциям управления заявками.

В таблице 4 представлена структура сущности Application, которая хранит данные о заявках. Каждая заявка имеет уникальный идентификатор id, уникальный номер app_number, краткое название name, подробное описание description, текущий статус status, дату создания created_at, а также внешний ключ user_id, указывающий на пользователя, создавшего заявку.

Таблица 4 – Описание таблицы Application

Поле	**Тип**	**Описание**
id	Integer	Уникальный идентификатор (PK)
app_number	String	Уникальный номер заявки (например, SOO1)
name	String	Краткое наименование заявки
description	Text	Описание заявки
status	String	Статус (“В работе”, “Выполнено”, “Отклонено”)
created_at	DateTime	Дата и время создания заявки (видно в бд)
user_id	Integer	Внешний ключ на пользователя (FK)

На рисунке 5 представлена визуализация структуры таблиц базы данных User и Application. Отображены их атрибуты, типы данных и связи. Таблица Application содержит внешний ключ user_id, обеспечивающий связь с таблицей User. Диаграмма иллюстрирует взаимосвязь данных, реализующую учет заявок в системе.

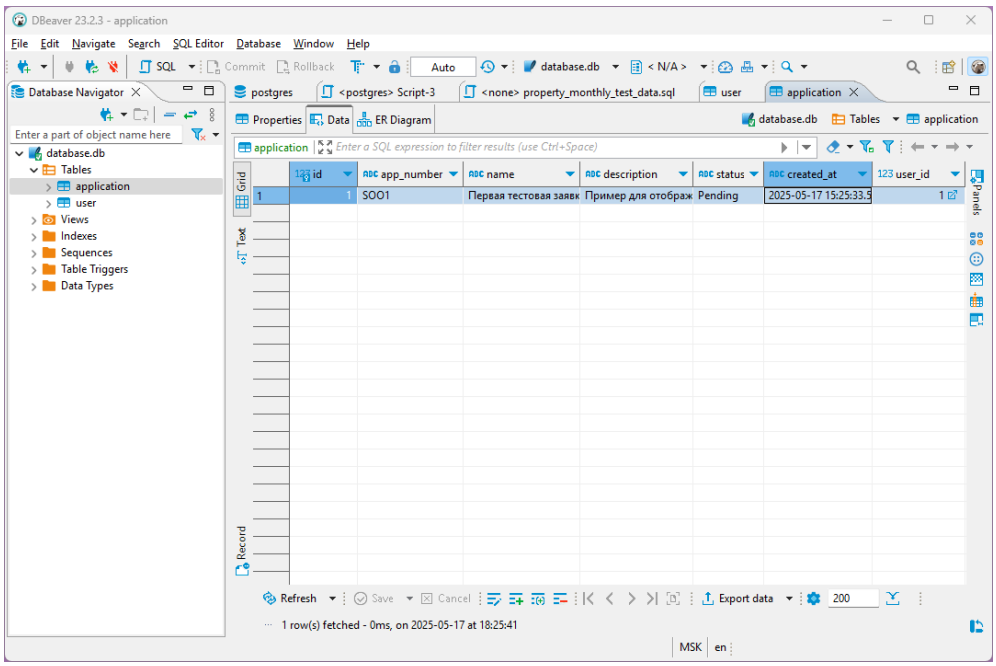

Рисунок 5 – Поля и структура базы данных

Фрагмент кода модели (models.py) представлен на рисунке 6.

```python
from app import db
from datetime import datetime
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash


class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)

    def set_password(self, password):  2 usages (1 dynamic)
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):  1 usage (1 dynamic)
        return check_password_hash(self.password_hash, password)


class Application(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    app_number = db.Column(db.String(20), unique=True, nullable=False)
    name = db.Column(db.String(100), nullable=False)
    description = db.Column(db.Text, nullable=False)
    status = db.Column(db.String(20), default='Pending')
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=True)
    user = db.relationship('User', backref='applications')
```

Рисунок 6 – Описание моделей данных (User и Application) на языке Python (SQLAlchemy)

На рисунке приведён фрагмент кода, реализующий модели таблиц базы данных с использованием ORM SQLAlchemy [5]. Описаны классы User и Application, определяющие поля, типы данных, связи между сущностями, а также ограничения целостности. Модель отражает структуру базы данных, используемой в веб-приложении.

Логика генерации номера заявки.

Номер заявки генерируется автоматически по принципу SOO + [номер, на 1 больше, чем у последней заявки], листинг кода представлен в рисунке 7.

Это обеспечивает уникальность каждой заявки и удобство поиска/сопоставления.

```
# Получаем максимальный существующий номер заявки
last_app = Application.query.order_by(Application.id.desc()).first()
if last_app and last_app.app_number.startswith('SOO'):
    try:
        last_number = int(last_app.app_number.replace('SOO', ''))
    except ValueError:
        last_number = last_app.id
else:
    last_number = 0
new_number = f"SOO{last_number + 1}"
```

Рисунок 7 – Генерация уникального номера заявки

На рисунке представлен фрагмент программного кода, реализующий алгоритм автоматической генерации уникального номера заявки в формате SOO1, SOO2 и т.д. Логика основана на определении максимального текущего значения и увеличении его на единицу для формирования следующего номера заявки.

3.3 Реализация основных функций

В данном разделе приводится описание реализации основных функциональных возможностей разработанной системы. Рассматриваются ключевые процессы взаимодействия пользователя с приложением, включая добавление, редактирование и просмотр данных, организацию работы с заявками, а также обеспечиваются механизмы авторизации, разграничения прав доступа и визуализации информации.

3.3.1. Регистрация и авторизация пользователей

– Пользователь может зарегистрироваться в системе, указав уникальное имя и пароль, рисунок 8, 9.

– Для аутентификации реализована безопасная авторизация, где пароли хранятся в виде хэшей.

– После входа пользователь получает доступ к личному кабинету и может работать с заявками.

– Реализовано разделение ролей: администратор и обычный пользователь.

– Обычный пользователь может только добавлять заявки.

– Администратор может редактировать и удалять любые заявки.

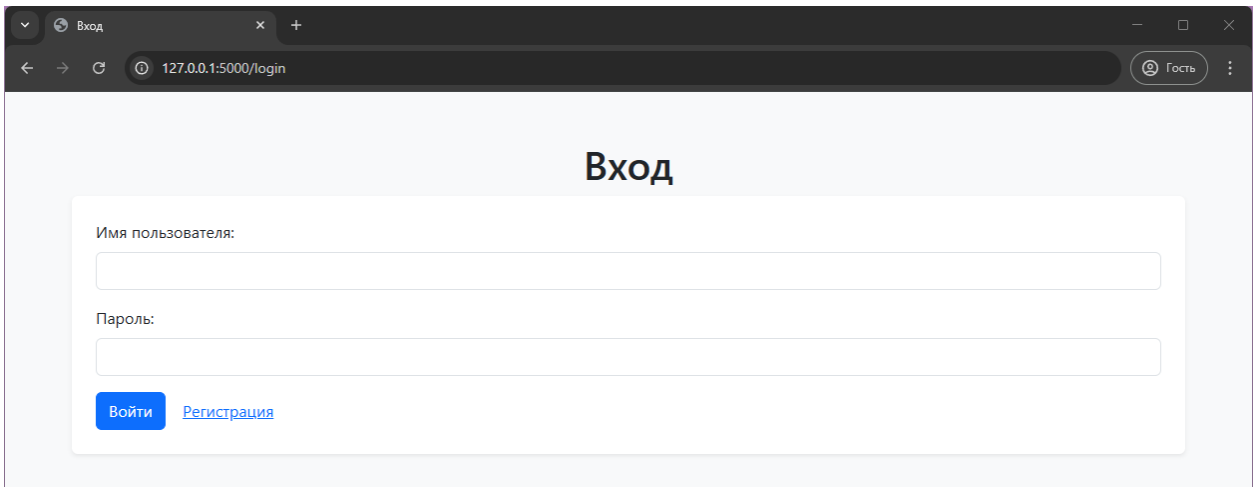

Рисунок 8 – Форма входа пользователя

На рисунке 8 представлена форма аутентификации веб-приложения. Пользователь вводит логин (имя пользователя) и пароль для доступа к системе. При вводе некорректных данных отображаются информативные сообщения об ошибке. Реализация формы обеспечивает защиту данных пользователя и является частью механизма авторизации системы.

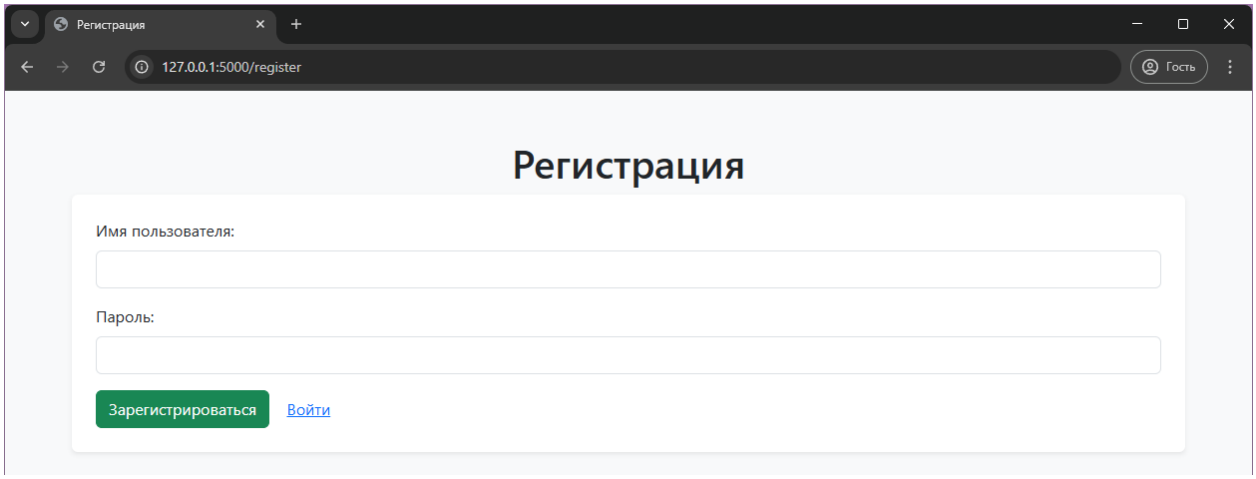

Рисунок 9 – Форма регистрации пользователей

На рисунке изображена форма регистрации нового пользователя в системе. Пользователь вводит уникальное имя (логин) и пароль для создания учетной записи. При регистрации предусмотрена валидация данных: проверяется уникальность имени пользователя и корректность введённого пароля. После успешной регистрации пользователь получает доступ к функционалу приложения в соответствии с назначенной ролью.

3.3.2. Добавление заявок

Каждый авторизованный пользователь может добавить новую заявку (рисунок 11). При создании заявки пользователю нужно указать только имя и описание, статус устанавливается по умолчанию как “В работе”. Каждая заявка получает уникальный неизменяемый номер (SOO1, SOO2, и т.д.), который формируется автоматически.

Фрагмент кода добавления заявки (routes.py) представлен на рисунке 10.

```python
@app.route('/add', methods=['GET', 'POST'])
@login_required
def add_application():
    if request.method == 'POST':
        name = request.form['name']
        description = request.form['description']

        # Получаем максимальный существующий номер заявки
        last_app = Application.query.order_by(Application.id.desc()).first()
        if last_app and last_app.app_number.startswith('SOO'):
            try:
                last_number = int(last_app.app_number.replace('SOO', ''))
            except ValueError:
                last_number = last_app.id
        else:
            last_number = 0
        new_number = f"SOO{last_number + 1}"

        new_app = Application(
            app_number=new_number,
            name=name,
            description=description,
            status='Pending',
            user_id=current_user.id
        )
        db.session.add(new_app)
        db.session.commit()
        flash( message: f'Заявка успешно добавлена! Номер заявки: {new_number}', category: 'success')
        return redirect(url_for('index'))
    return render_template('add_application.html')
```

Рисунок 10 – Пример кода добавления новой заявки и генерации уникального номера

На рисунке 10 показан фрагмент кода серверной части приложения (файл routes.py), реализующий логику добавления новой заявки пользователем. При сохранении заявки автоматически генерируется уникальный номер в формате SOO1, SOO2 и т.д. Код обрабатывает получение данных из формы, формирует номер заявки, сохраняет запись в базе данных и отображает уведомление о результате операции.

На рисунке 11 представлена форма создания новой заявки в веб-приложении. Пользователь вводит название заявки и её описание. После отправки формы заявке автоматически присваивается уникальный номер, а статус по умолчанию устанавливается как «В работе». Интерфейс формы обеспечивает обязательность заполнения основных полей и информирует пользователя о результате добавления.

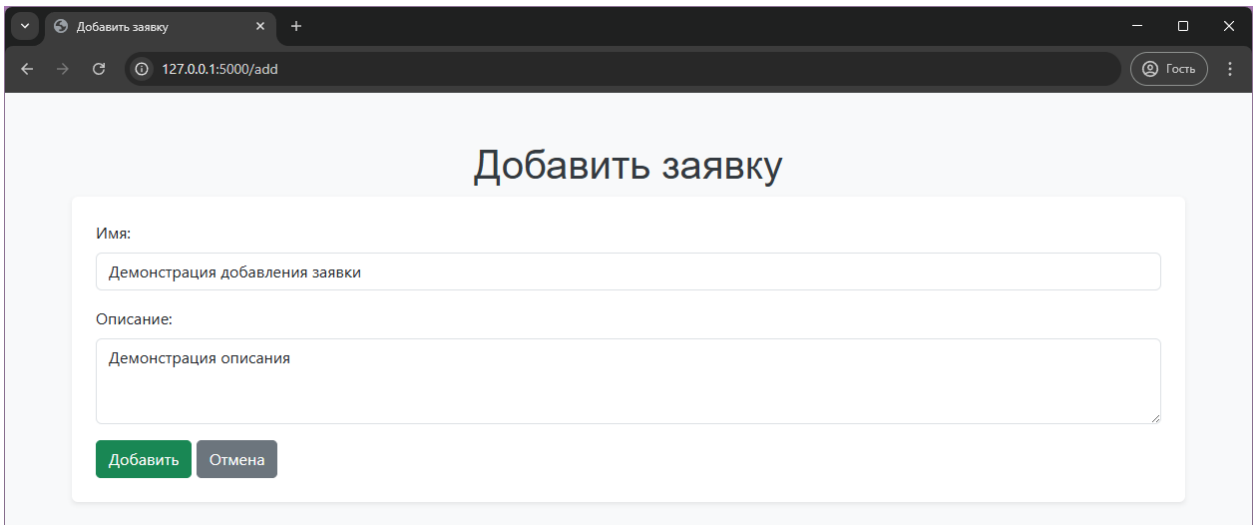

Рисунок 11 – Форма добавления заявки

Таким образом, в рамках реализации функционала добавления заявок обеспечена простота и удобство интерфейса для пользователей, автоматическое формирование уникального номера каждой заявки, а также контроль корректности введённых данных. Реализация данного модуля позволяет эффективно организовать регистрацию заявок в системе и создать основу для дальнейшей их обработки и учёта.

3.3.3. Просмотр и фильтрация заявок

После добавления заявок пользователями система обеспечивает удобный механизм просмотра и фильтрации данных. Данный функционал позволяет оперативно отслеживать статус заявок, осуществлять поиск по критериям и обеспечивать прозрачность обработки обращений. Реализация интерфейса просмотра заявок ориентирована на упрощение работы пользователей и минимизацию времени на поиск необходимой информации.

– На главной странице (рис. 12) отображается список всех заявок с их статусом, номером, именем и описанием.

– Реализована фильтрация заявок по статусу (“Все”, “В работе”, “Выполнено”, “Отклонено”).

– Для каждого пользователя видны все заявки.

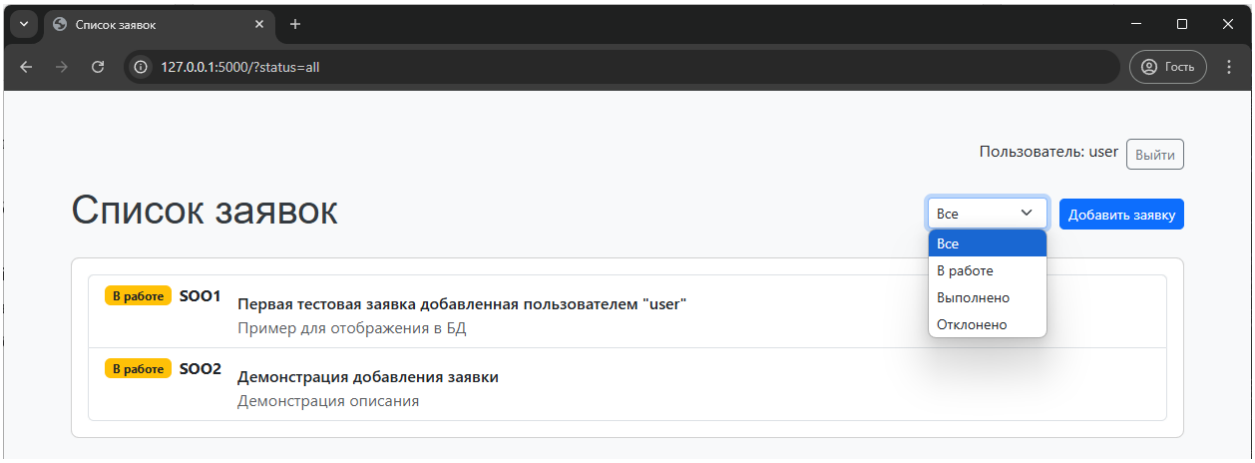

Рисунок 12 – Главная страница со списком и фильтром заявок

На рисунке показан интерфейс главной страницы веб-приложения, отображающей список всех заявок. Для каждой заявки указаны её уникальный номер, имя, описание и текущий статус. В верхней части реализована панель фильтрации, позволяющая пользователю выбирать отображение заявок по различным статусам («Все», «В работе», «Выполнено», «Отклонено»), что упрощает навигацию и контроль обработки заявок.

3.3.4. Редактирование и удаление заявок

Только пользователь с правами администратора может редактировать или удалять любую заявку. На рисунке 13 предоставлен листинг кода редактирования заявки с проверкой прав администратора. Обычный пользователь не видит кнопки «Редактировать» (рис. 15) и «Удалить». При входе под администратором появляются кнопки «Редактировать» и «Удалить», которые представлены на рисунке 14.

```python
@app.route('/edit/<int:id>', methods=['GET', 'POST'])
@login_required
def edit_application(id):
    if not current_user.is_admin:
        flash( message: 'Доступ запрещен! Только администратор может редактировать заявки.', category: 'danger')
        return redirect(url_for('index'))
    app_to_edit = Application.query.get_or_404(id)
    if request.method == 'POST':
        app_to_edit.name = request.form['name']
        app_to_edit.description = request.form['description']
        app_to_edit.status = request.form['status']
        db.session.commit()
        flash( message: 'Заявка успешно обновлена!', category: 'success')
        return redirect(url_for('index'))
    return render_template( template_name_or_list: 'edit_application.html', application=app_to_edit)

@app.route('/delete/<int:id>', methods=['POST'])
@login_required
def delete_application(id):
    if not current_user.is_admin:
        flash( message: 'Доступ запрещен! Только администратор может удалять заявки.', category: 'danger')
        return redirect(url_for('index'))
    app_to_delete = Application.query.get_or_404(id)
    db.session.delete(app_to_delete)
    db.session.commit()
    flash( message: 'Заявка успешно удалена!', category: 'success')
    return redirect(url_for('index'))
```

Рисунок 13 – Пример кода редактирования заявки с проверкой прав администратора

На рисунке 13 представлен фрагмент серверного кода, реализующий функционал редактирования заявок. В коде предусмотрена проверка прав доступа: редактирование разрешено только для пользователей с административными правами. При попытке выполнения операции пользователями без соответствующих прав система блокирует доступ, обеспечивая безопасность и корректность обработки данных.

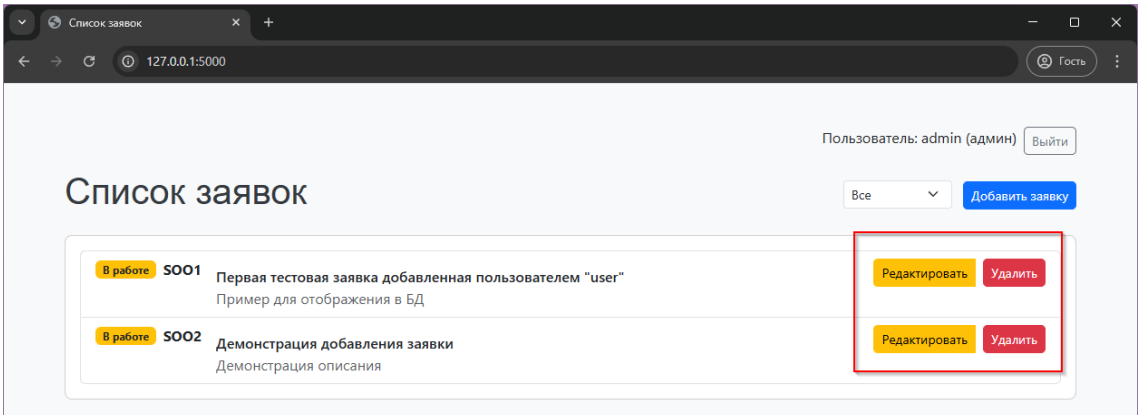

Рисунок 14 – Кнопки «Редактировать» и «Удалить»

На рисунке 14 показан фрагмент пользовательского интерфейса, отображающий административные элементы управления заявками. Для пользователей с правами администратора доступны кнопки «Редактировать» и «Удалить», позволяющие изменять содержимое заявки или полностью удалять её из системы. Для обычных пользователей данные элементы интерфейса недоступны, что обеспечивает разграничение прав доступа.

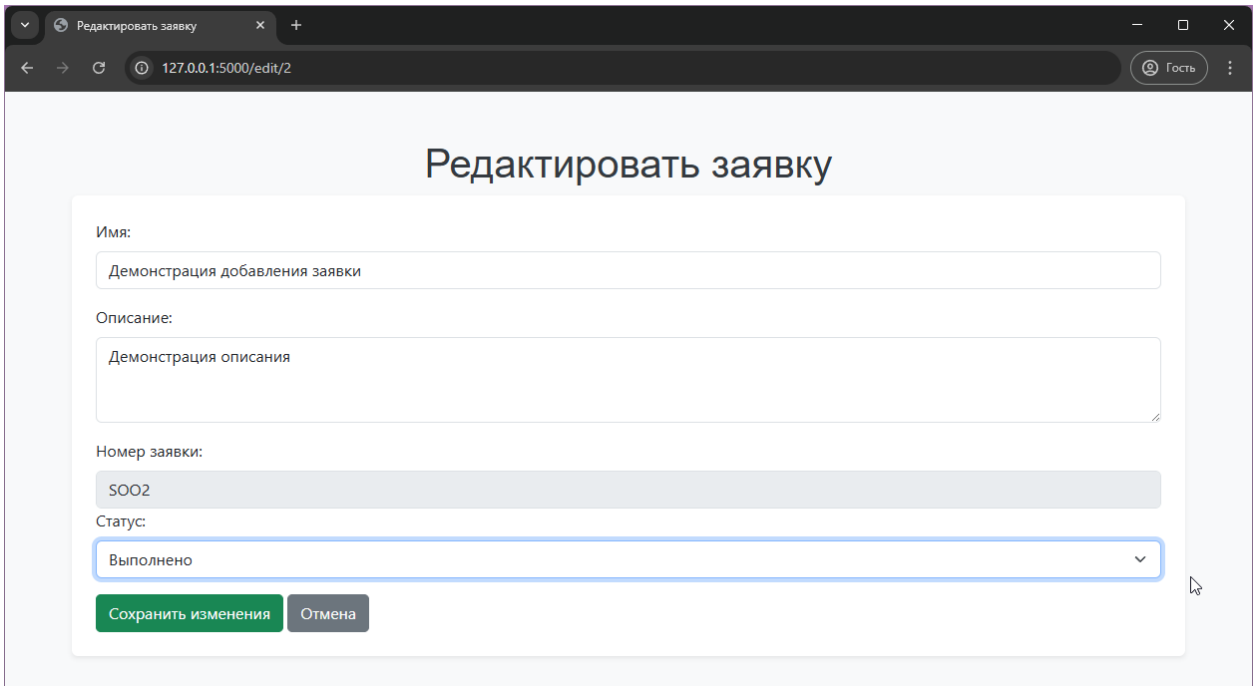

Рисунок 15 – Форма редактирования заявки

На рисунке 15 представлена форма редактирования заявки, доступная администраторам системы. Интерфейс позволяет изменять название заявки, её описание и текущий статус. Обязательные поля визуально выделены, а успешное сохранение изменений сопровождается уведомлением пользователя. Реализация данной формы обеспечивает удобство администрирования и поддерживает актуальность данных в базе заявок.

Таким образом, функционал редактирования и удаления заявок реализован с учётом разграничения прав пользователей и требований информационной безопасности. Администраторы имеют возможность

оперативно корректировать сведения и управлять заявками, тогда как обычные пользователи ограничены в доступе к данным операциям. Такая модель обеспечивает стабильную работу системы и предотвращает несанкционированное изменение информации.

3.3.5. Система уведомлений и обработка ошибок

Для всех ключевых операций используются flash-сообщения, информирующие пользователя о результате (успех/ошибка) (рис. 16). Все маршруты, связанные с заявками, защищены авторизацией (декоратор @login_required).

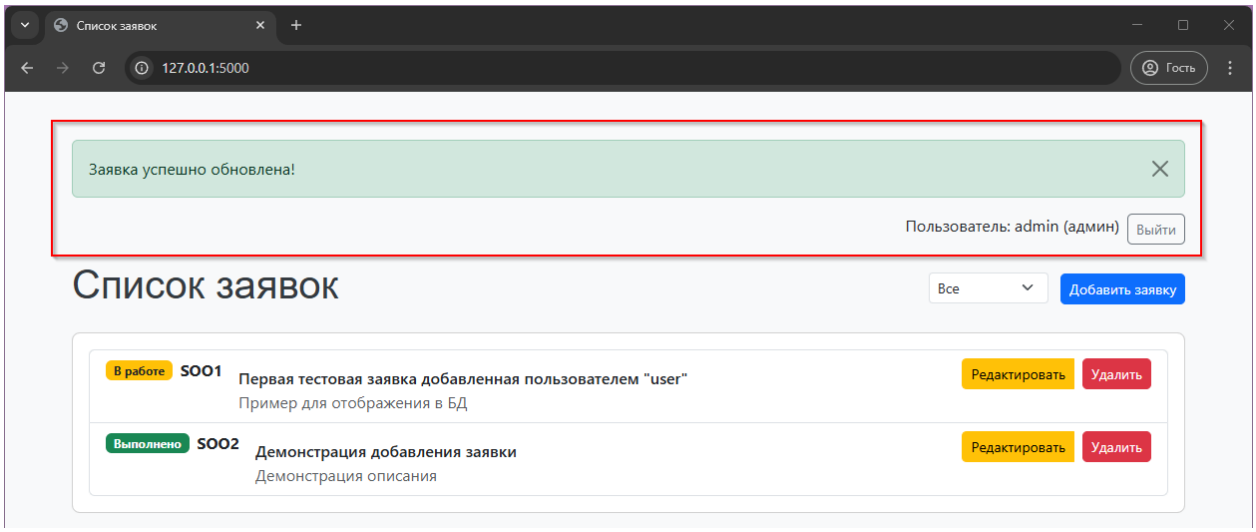

Рисунок 16 – Использование flash-сообщений для уведомления пользователя

На рисунке 16 показан пример использования flash-сообщений в веб-приложении. После выполнения действий — таких как добавление, редактирование или удаление заявки — пользователю выводятся уведомления о результате операции. Это обеспечивает обратную связь, информирует о успешных действиях или возникших ошибках, повышая удобство и прозрачность взаимодействия с системой.

Таким образом, использование системы уведомлений и обработка ошибок позволяют информировать пользователей о результатах их действий

и предотвращать возникновение критических сбоев. Реализация flash-сообщений и защита маршрутов с помощью авторизации повышают удобство работы, безопасность приложения и стабильность его функционирования при различных сценариях использования.

3.3.6. Интерфейс пользователя

– Используется фреймворк Bootstrap 5 для стилизации и адаптивности всех страниц.

– Интерфейс интуитивно понятен (рис. 17): статус заявки и номер выделены, описание — ниже.

– Обычный пользователь не видит недоступных для него функций (редактирование/удаление).

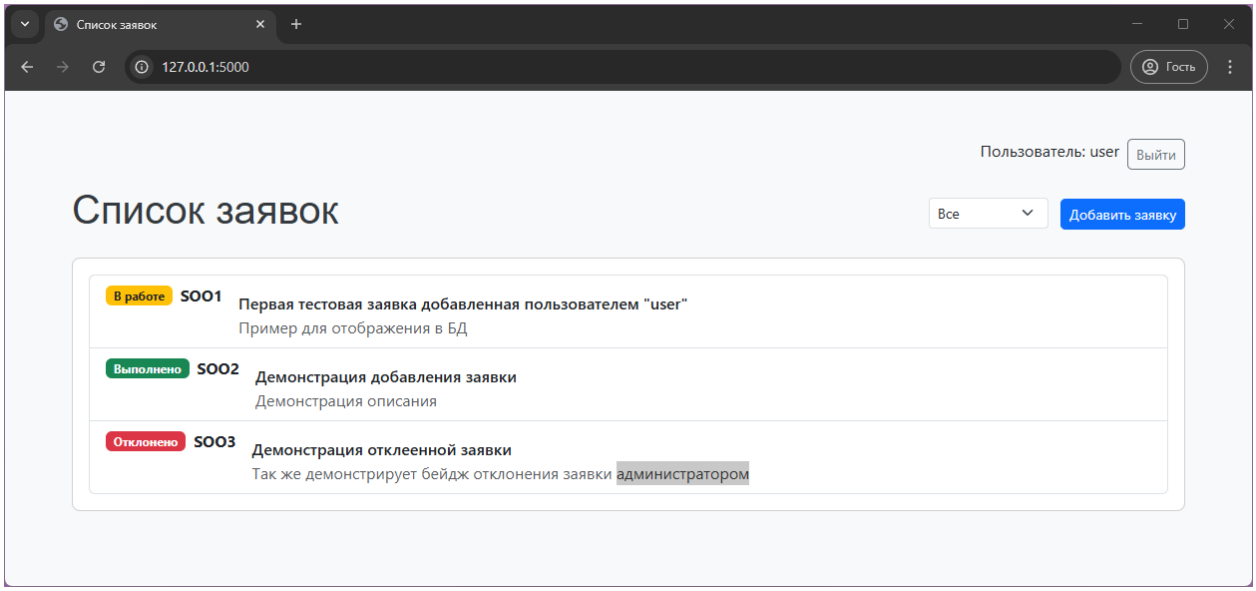

Рисунок 17 – Пример интерфейса списка заявок с бейджами и номерами

На рисунке 17 представлен элемент пользовательского интерфейса веб-приложения со списком заявок. Для каждой заявки отображаются уникальный номер, имя, краткое описание и текущий статус, выделенный с помощью цветных бейджей. Такой визуальный формат обеспечивает наглядность

информации, позволяет быстро оценить текущее состояние обработки заявок и упрощает навигацию для пользователей различных категорий.

Таким образом, реализованный пользовательский интерфейс обеспечивает интуитивную навигацию, визуальную понятность и адаптивность отображения данных. Использование современных средств верстки и стилизации позволило создать удобную, функциональную и доступную для пользователей систему, что повышает эффективность работы и снижает время на обучение персонала.

3.4 Механизмы безопасности

В процессе разработки веб-приложения для учёта заявок особое внимание уделялось вопросам безопасности, так как система содержит персональные данные пользователей и потенциально важную информацию для организации. В приложении реализованы следующие механизмы защиты.

Безопасная авторизация и хранение паролей. Для хранения учетных данных пользователей используется защищённое хеширование паролей (алгоритмы из библиотеки Werkzeug). Ни один пароль не хранится в базе данных в открытом виде, пример отображен на рисунке 18. При входе пользователя сравнение введённого пароля происходит только через сравнение хэшей.

```
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)

    def set_password(self, password):  2 usages (1 dynamic)
        self.password_hash = generate_password_hash(password)

    def check_password(self, password):  1 usage (1 dynamic)
        return check_password_hash(self.password_hash, password)
```

Рисунок 18 – Пример безопасного хранения и проверки пароля пользователя.

На рисунке 18 представлен фрагмент программного кода, демонстрирующий механизм безопасного хранения и проверки паролей с использованием библиотеки Werkzeug. При регистрации пароли преобразуются в хэш-значения перед сохранением в базу данных. При аутентификации производится сравнение хэшей, что исключает возможность хранения паролей в открытом виде и обеспечивает защиту учетных данных пользователей.

Ограничение доступа к функциям по ролям. Для всех маршрутов, связанных с заявками, применяется декоратор @login_required, который не позволяет обращаться к этим страницам неавторизованным пользователям. Разделение прав между пользователями реализовано с помощью флага is_admin в таблице пользователей:

– Обычный пользователь может только создавать заявки.

– Администратор может редактировать и удалять заявки.

Листинг кода примера проверки прав пользователя при попытке редактирования заявки представлен на рисунке 19.

```
@login_required
def edit_application(id):
    if not current_user.is_admin:
        flash( message: 'Доступ запрещен! Только администратор может редактировать заявки.', category: 'danger')
        return redirect(url_for('index'))
    app_to_edit = Application.query.get_or_404(id)
    if request.method == 'POST':
        app_to_edit.name = request.form['name']
        app_to_edit.description = request.form['description']
        app_to_edit.status = request.form['status']
        db.session.commit()
        flash( message: 'Заявка успешно обновлена!', category: 'success')
        return redirect(url_for('index'))
    return render_template( template_name_or_list: 'edit_application.html', application=app_to_edit)
```

Рисунок 19 – Пример проверки прав пользователя при попытке редактирования заявки

На рисунке представлен фрагмент серверного кода, реализующего проверку прав доступа при редактировании заявки. В коде осуществляется проверка роли пользователя: только при наличии административных прав разрешается редактирование данных. В случае отсутствия соответствующих

полномочий генерируется сообщение об ошибке и блокируется выполнение операции. Такой подход обеспечивает защиту данных и предотвращает несанкционированные изменения.

Ограничение отображения кнопок в интерфейсе. Пользователи, не имеющие прав администратора, не видят в интерфейсе кнопок «Редактировать» и «Удалить», что дополнительно защищает от случайных попыток получить доступ к функциям (рис. 20).

```
{% if current_user.is_admin %}
<div class="btn-group mt-2 mt-md-0">
    <a href="{{ url_for('edit_application', id=app.id) }}" class="btn btn-warning btn-sm">Редактировать</a>
    <form action="{{ url_for('delete_application', id=app.id) }}" method="POST" class="ms-2">
        <button type="submit" class="btn btn-danger btn-sm">Удалить</button>
    </form>
</div>
{% endif %}
```

Рисунок 20 – Отображение административных функций только для админов

На рисунке 20 показан пользовательский интерфейс веб-приложения с реализованным разграничением прав доступа. Для пользователей без административных полномочий элементы управления — кнопки «Редактировать» и «Удалить» — не отображаются в интерфейсе. Такая реализация снижает вероятность ошибочных действий, повышает безопасность и дополнительно ограничивает доступ к функционалу, предназначенному только для администраторов.

Так же использование Flask-Login обеспечивает надёжное управление сессией, предотвращает несанкционированный доступ при выходе пользователя. В случае попытки выполнения запрещённого действия пользователю выводится информативное флеш-сообщение («Доступ запрещен!» и др.), что дополнительно снижает вероятность социальной инженерии.

3.5 Тестирование и примеры работы

Для подтверждения корректности работы веб-приложения был проведён комплекс ручных тестов, охватывающих как обычные сценарии, так и возможные ошибочные ситуации.

Тестирование проводилось с использованием различных ролей пользователей: «администратор» и «обычный пользователь». Результаты теста представлены в таблице 5.

Таблица 5 – Основные тест-кейсы

№	Сценарий	Ожидаемый результат	Статус
1	Регистрация нового пользователя	Пользователь успешно зарегистрирован	Пройден
2	Авторизация с корректными данными	Вход в систему, доступ к функциям	Пройден
3	Авторизация с неверным паролем	Отображение сообщения об ошибке	Пройден
4	Добавление заявки обычным пользователем	Заявка появляется в списке, номер SOO...	Пройден
5	Проверка, что статус заявки “В работе”	Статус по умолчанию “В работе”	Пройден
6	Попытка редактировать/удалить заявку	Кнопки отсутствуют, доступ запрещён	Пройден
7	Редактирование/удаление заявки админом	Операция выполняется, статус меняется	Пройден
8	Фильтрация заявок по статусу	Отображаются только заявки с выбранным статусом	Пройден
9	Выход из системы	Перенаправление на страницу авторизации	Пройден
10	Попытка доступа к приложениям без входа	Перенаправление на страницу логина	Пройден

В таблице 5 приведены основные сценарии тестирования веб-приложения, охватывающие ключевые функции системы: регистрацию и авторизацию пользователей, добавление, просмотр, редактирование и удаление заявок, разграничение прав доступа, фильтрацию заявок, а также поведение системы при ошибках и выходе из учетной записи. Каждый тест-кейс включает описание сценария, ожидаемый результат и статус успешного прохождения теста, что подтверждает стабильную работу системы при различных условиях эксплуатации.

После успешного проведения тестирования были выполнены демонстрационные испытания, иллюстрирующие работу основных функций веб-приложения. В данном разделе приведены скриншоты пользовательского интерфейса, отражающие ключевые сценарии взаимодействия с системой при добавлении, просмотре, редактировании и управлении заявками.

Форма авторизации — рисунок 21 демонстрирует вход в систему.

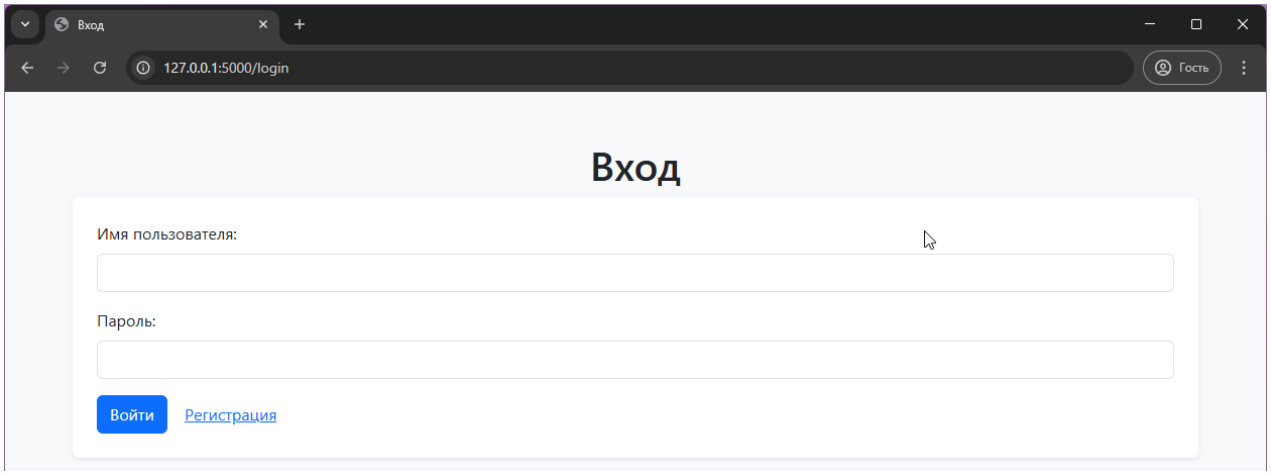

Рисунок 21 – Страница входа пользователя

На рисунке 21 показан экран авторизации веб-приложения. Пользователь вводит зарегистрированные логин и пароль для получения доступа к системе. Реализована проверка введённых данных, отображение сообщений об ошибках при некорректном вводе, а также защита данных в

процессе аутентификации. Данный интерфейс является обязательным этапом при работе с системой и обеспечивает ограничение доступа неавторизованных пользователей.

Главная страница (список заявок) с фильтрацией — видно статусы, номера, описание (рис. 22).

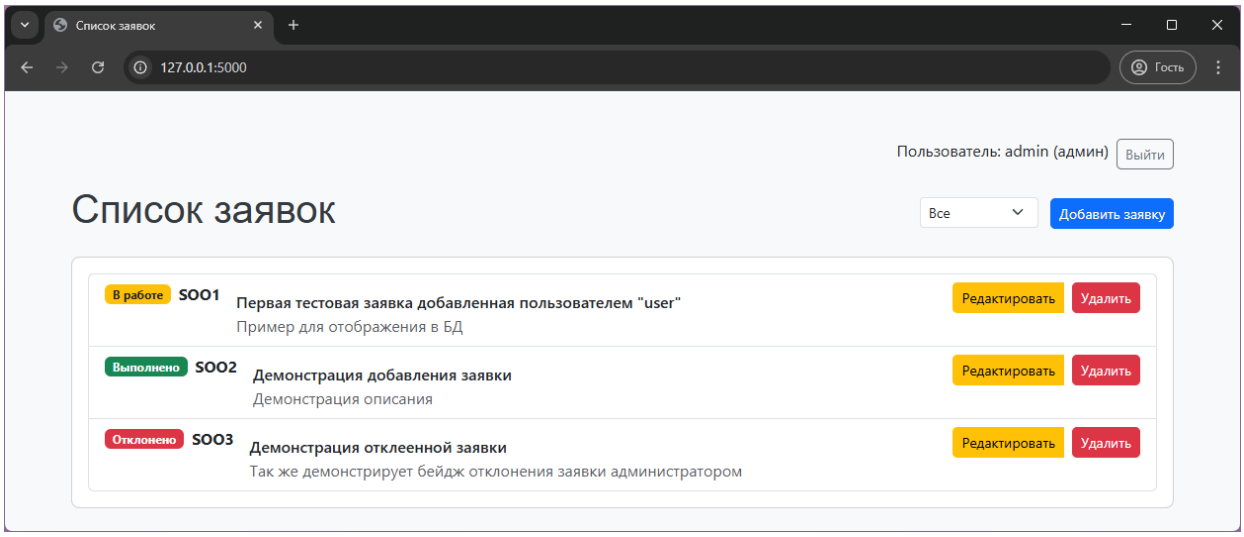

Рисунок 22 – Список заявок с фильтром по статусу

На рисунке 22 представлена главная страница веб-приложения, содержащая список всех зарегистрированных заявок. Для каждой заявки отображаются её уникальный номер, название, краткое описание и текущий статус. В верхней части страницы размещён элемент фильтрации, позволяющий пользователю выбирать отображение заявок по различным статусам, что облегчает поиск и контроль обработки заявок.

Форма добавления заявки представлена на рисунке 23.

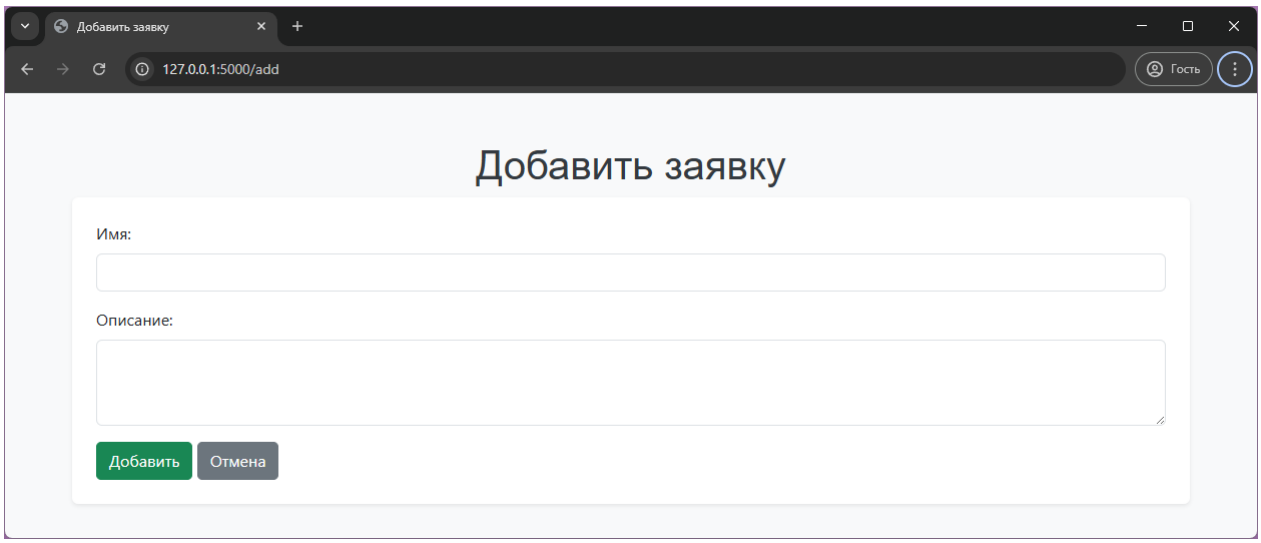

Рисунок 23 – Форма добавления заявки

Попытка доступа к функциям админа обычным пользователем — отсутствуют кнопки, либо сообщение «Доступ запрещен!» при ручном вводе URL. Сообщение о запрете доступа к функциям редактирования показано на рисунке 24.

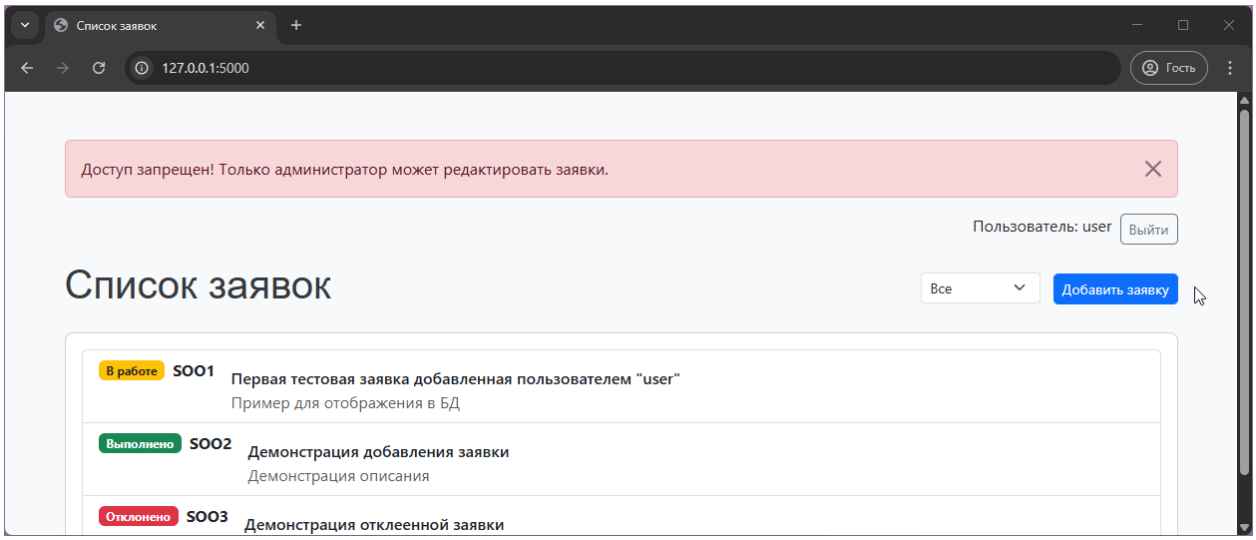

Рисунок 24 – Сообщение о запрете доступа к функциям редактирования

Форма редактирования заявки (админ) — поля для изменения статуса и содержимого (рис. 25).

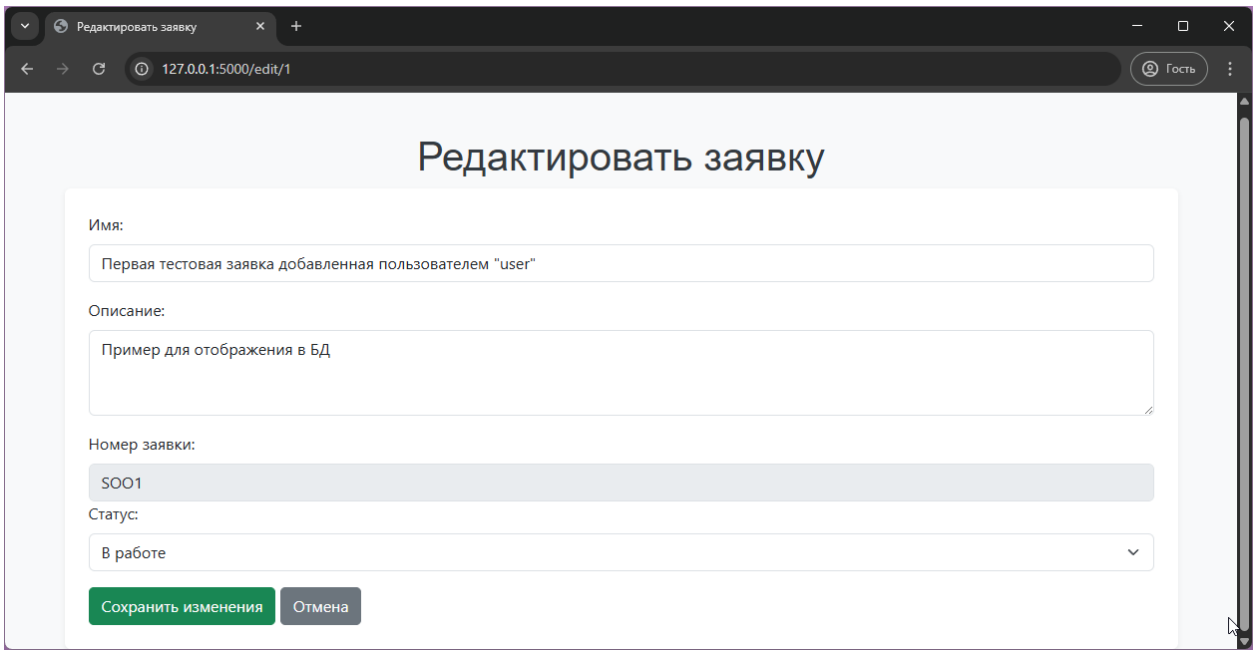

Рисунок 25 – Редактирование заявки администратором

На рисунке 25 представлена форма редактирования заявки, доступная только для пользователей с правами администратора. Интерфейс позволяет изменять содержимое заявки: её название, описание и статус. Реализация формы предусматривает проверку корректности введённых данных и вывод уведомлений о результате редактирования. Такая организация интерфейса обеспечивает удобное управление заявками и поддержание актуальности данных в системе.

Таким образом, приведённые скриншоты демонстрируют корректную работу основных функций веб-приложения: авторизации пользователей, создания, отображения, фильтрации и редактирования заявок с учётом разграничения прав доступа. Интерфейс системы обеспечивает удобство взаимодействия для конечных пользователей и подтверждает соответствие реализованного функционала поставленным требованиям.

3.5.1 Проверка генерации номера заявки

Для каждой новой заявки автоматически создаётся уникальный номер вида SOO1, SOO2 и т.д. Номер отображается в списке и не может быть изменён пользователем. Листинг представлен на рисунке 26.

```
last_app = Application.query.order_by(Application.id.desc()).first()
if last_app and last_app.app_number.startswith('SOO'):
    try:
        last_number = int(last_app.app_number.replace('SOO', ''))
    except ValueError:
        last_number = last_app.id
else:
    last_number = 0
new_number = f"SOO{last_number + 1}"
```

Рисунок 26 – Генерация уникального номера заявки при добавлении

На рисунке 26 представлен фрагмент программного кода, реализующего автоматическое формирование уникального номера заявки при её добавлении. Номер создаётся в формате SOO1, SOO2 и далее, на основе определения текущего максимального значения в базе и его увеличения на единицу. Данный механизм гарантирует уникальность каждого номера заявки и упрощает их идентификацию и учёт в системе.

Таким образом, реализованный алгоритм генерации номеров заявок обеспечивает автоматическое формирование уникальных идентификаторов без участия пользователя. Это исключает возможность дублирования номеров, упрощает ведение учёта и повышает надёжность хранения информации в базе данных.

3.5.2 Пример типовой пользовательской сессии

Для демонстрации работы системы был проведён пример типовой пользовательской сессии, отражающий основные сценарии взаимодействия пользователя и администратора с веб-приложением [20]. В ходе сессии проверялись корректность выполнения операций по авторизации, добавлению

и обработке заявок, а также функционирование механизмов разграничения прав доступа и защиты системы.

– Пользователь заходит на страницу авторизации и успешно входит в систему.

– Добавляет новую заявку — система присваивает номер SOO...

– Проверяет, что кнопки “Редактировать” и “Удалить” недоступны.

– Администратор заходит, видит все заявки, может редактировать и удалять.

– После выхода из системы попытки открыть /add или /edit/<id> ведут на страницу авторизации.

Таким образом, результаты моделирования типовой пользовательской сессии подтвердили корректную работу всех основных функций приложения. Система стабильно обрабатывает действия пользователей, обеспечивает защиту от несанкционированного доступа и корректно обрабатывает различные сценарии использования в соответствии с заданной логикой ролей и прав.

Выводы по главе 3.

В рамках третьей главы была проведена полная разработка и внедрение веб-приложения для автоматизации учёта заявок в социально ориентированной организации. В результате реализации поставленных задач были достигнуты следующие результаты:

– Разработано современное и удобное веб-приложение с использованием стека Python, Flask, SQLAlchemy, SQLite и Bootstrap, что обеспечило надёжную и масштабируемую архитектуру [22].

– Реализована система регистрации и авторизации пользователей с разграничением прав доступа (администратор и обычный пользователь), что повысило уровень безопасности и гибкости системы.

– Внедрён уникальный неизменяемый номер для каждой заявки, что обеспечивает удобную идентификацию и контроль учёта.

– Пользовательский интерфейс выполнен адаптивным и интуитивно понятным, что облегчает работу конечных пользователей.
– Реализованы все основные функции: добавление, просмотр, фильтрация, редактирование и удаление заявок с учётом ролей пользователей.
– Обеспечена защита данных и управление доступом с помощью безопасного хранения паролей, авторизации, разграничения ролей и обработки несанкционированных действий.
– Проведённое тестирование показало, что система стабильно функционирует, корректно реагирует на ошибки и предотвращает несанкционированный доступ к данным.

В целом, веб-приложение полностью соответствует требованиям, поставленным во введении и аналитической части работы. Решённые задачи демонстрируют эффективность предложенной архитектуры для автоматизации учёта заявок в организациях подобного профиля.

Таким образом, поставленные цели главы полностью достигнуты, что подтверждается результатами тестирования и демонстрацией работы приложения.

Заключение

В ходе выполнения выпускной квалификационной работы разработано веб-приложение для автоматизации учета заявок в социально ориентированной организации. Основной задачей являлось создание удобного сервиса, позволяющего эффективно регистрировать, обрабатывать и отслеживать заявки пользователей. Для достижения этой цели реализован набор функций: добавление, просмотр, изменение и удаление заявок, а также уникальная система нумерации и механизм отображения статусов для контроля исполнения.

В процессе работы приобретены практические навыки в области программирования и проектирования современных информационных систем, освоены инструменты Python, Flask, SQLite, методы построения web-интерфейсов, обработки данных и работы с базами данных. Особое внимание уделялось вопросам безопасности и надежности хранения пользовательской информации.

Выполнение проекта включало несколько этапов:

– Изучение предметной области. Проведен анализ специфики работы социально ориентированных организаций, определены основные потребности сотрудников и выявлены проблемы существующего документооборота. На основании анализа сформулирована основная задача — создание максимально простой, но функциональной системы.

– Изучение существующих решений. Рассмотрены и проанализированы доступные программные продукты для автоматизации учета заявок. Недостатки таких решений — избыточная сложность и высокая стоимость — подтвердили необходимость разработки собственного приложения.

– Формирование функциональных и нефункциональных требований. Для разрабатываемого приложения определены ключевые требования к функционалу, безопасности, удобству использования.

– Проектирование архитектуры и интерфейса. Разработана структура базы данных, спроектирован простой и понятный пользовательский интерфейс, составлены диаграммы сущностей и вариантов использования, продуманы сценарии работы для разных ролей.

– Реализация программного кода. Приложение создано с использованием современных технологий: Flask (Python) для серверной части, SQLite для хранения данных. Проект реализован по принципу модульности, что обеспечивает возможности для дальнейшего развития.

– Тестирование и отладка. На этапе тестирования проведена проверка устойчивости приложения к ошибкам, корректности обработки различных сценариев работы и удобства использования.

– Оценка эффективности. Разработка показала, что внедрение подобной системы позволяет ускорить обработку заявок, сократить количество ошибок и повысить прозрачность работы организации.

В результате выполнения всех поставленных задач закреплены навыки проектирования и реализации web-приложений, работы с базами данных, проектирования архитектуры программных систем, а также анализа бизнес-процессов и внедрения их автоматизации на практике.

Все задачи, обозначенные в начале работы, реализованы в полном объеме, а поставленная цель — разработка веб-приложения для автоматизации учета заявок в социально ориентированной организации — достигнута.

Список используемой литературы и используемых источников

1. Беннетт А., Ларкин К. Python. Разработка приложений. — СПб.: Питер, 2020. — 480 с.
2. Вахтеров В. В., Зиннуров Р. А. Проектирование информационных систем. — Казань: КГТУ, 2017. — 212 с.
3. Гринев В. А. Проектирование баз данных: учебник. — М.: КНОРУС, 2022. — 432 с.
4. Дейт К. Дж. Введение в системы баз данных. — М.: Вильямс, 2018. — 1024 с.
5. Дронов В. А., Чернышов С. С. Проектирование интерфейсов. — СПб.: Питер, 2020. — 336 с.
6. Евсеев Г. П., Жуков А. А. Основы построения клиент-серверных приложений. — М.: Академия, 2019. — 304 с.
7. Зайцев Д. В. Безопасность веб-приложений: учебное пособие. — М.: КНОРУС, 2019. — 220 с.
8. Кузнецов А. А., Федорова С. Г. Разработка интерфейсов пользователя. — М.: Интернет-Университет Информационных Технологий, 2021. — 340 с.
9. Макарычев С. В. Основы программирования на Python. — М.: ДМК Пресс, 2021. — 368 с.
10. Малков И. А. Flask. Разработка веб-приложений на Python. — М.: ДМК Пресс, 2020. — 256 с.
11. Марченко А. В. Разработка веб-приложений на Python и Flask. — СПб.: БХВ-Петербург, 2020. — 256 с.
12. Мартин Р. Ч. Чистая архитектура. Искусство разработки программного обеспечения. — М.: ДМК Пресс, 2021. — 352 с.
13. Молоков Д. В. SQLite. Практическое руководство. — СПб.: БХВ-Петербург, 2017. — 320 с.

14. Павлов А. В. SQLite: руководство разработчика. — М.: ДМК Пресс, 2020. — 288 с.

15. Поляков А. В. Системный анализ и проектирование информационных систем. — М.: Директ-Медиа, 2021. — 328 с.

16. Шумаков А. С., Горюнов А. С. Технологии программирования и проектирования web-приложений. — М.: Директ-Медиа, 2018. — 295 с.

17. Мэтиз Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения. — СПб.: Питер, 2021. — 640 с.

18. Grinberg M. Flask Web Development: Developing Web Applications with Python. — 2nd ed. — O’Reilly Media, 2018. — 350 p.

19. Stuttard D., Pinto M. The Web Application Hacker’s Handbook: Finding and Exploiting Security Flaws. — 2nd ed. — Wiley, 2011. — 912 p.

20. Matplotlib Development Team. Matplotlib Documentation [Electronic resource]. URL: https://matplotlib.org/stable/contents.html (accessed: 10.06.2025).

21. Lutz M. Learning Python. — 5th ed. — O’Reilly Media, 2013. — 1648 p.

22. Beazley D. M., Jones B. K. Python Cookbook. — 3rd ed. — O’Reilly Media, 2013. — 706 p.