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

Институт цифровых технологий
(наименование института полностью)

Департамент бакалавриата
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка программного обеспечения для автоматизации процесса обработки заявок в технической поддержке

Обучающийся И.В. Зеленский _____
(Инициалы Фамилия) (личная подпись)

Руководитель доцент, кандидат педагогических наук Е.А. Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Рассматривается процесс автоматизации обработки заявок в технической поддержке путем разработки программного обеспечения. В работе проанализированы существующая ситуация и потребности организации, определены цель и задачи исследования, выбраны методы и средства реализации проекта.

Объектом исследования является процесс приема, обработки и сопровождения заявок пользователей. Цель работы - повышение эффективности работы службы технической поддержки посредством автоматизации. Поставлены задачи разработки функциональной системы, соответствующей стандартам качества и безопасности.

Методы исследования включают анализ существующих систем, разработку архитектуры программного комплекса, проектирование базы данных и реализацию функционала с применением современных технологий, таких как Yii2 framework, MySQL, Redis и искусственного интеллекта GigaChat. Особое внимание уделяется вопросам безопасности и юзабилити интерфейса.

Практическая значимость работы обусловлена возможностью значительного снижения нагрузки на операторов технической поддержки, повышением точности обработки обращений и увеличением удовлетворенности пользователей уровнем сервиса. Предлагаемая система отличается гибкостью, масштабируемостью и высоким уровнем безопасности.

Полученные результаты демонстрируют успешное функционирование системы в большинстве случаев, обнаруженные ошибки носят технический характер и не препятствуют нормальной эксплуатации. Внедрение разработанного программного обеспечения позволит повысить качество обслуживания пользователей и ускорить процессы обработки заявок.

Работа выполнена на 69 страницах, включает 28 рисунков, 14 таблиц, список литературы насчитывает 18 наименований.

Оглавление

Введение

Современные информационные системы технической поддержки всё чаще применяются не только в коммерческих организациях, но и в сфере городского обслуживания, где требуется оперативная обработка обращений граждан и контроль их исполнения. Рост цифровизации и активное использование мобильных устройств создают новые возможности для автоматизации процесса регистрации, классификации и обработки заявок пользователей.

Одной из актуальных задач в этом направлении является создание универсальных платформ технической поддержки, способных автоматически принимать обращения, анализировать их содержимое и направлять ответственным исполнителям без участия оператора. Подобные решения повышают скорость реакции служб, снижают нагрузку на персонал и обеспечивают прозрачность взаимодействия с пользователями.

Настоящая работа посвящена разработке программного обеспечения, реализующего автоматизацию процесса обработки заявок в службе технической поддержки городской инфраструктуры**.** В качестве предметной области выбрана обработка обращений граждан о нарушениях правил размещения рекламных материалов (в частности, незаконной расклейке объявлений на остановках общественного транспорта). В данном контексте каждое обращение рассматривается как заявка в техническую поддержку городской среды, требующая регистрации, проверки и исполнения.

Цель дипломной работы - разработать веб-приложение для автоматизации процесса приёма, обработки и сопровождения заявок пользователей с применением технологий искусственного интеллекта для анализа поступающей информации.

Для достижения цели поставлены следующие задачи:

- провести анализ предметной области и существующих систем обработки заявок в службах поддержки;

- спроектировать архитектуру программного обеспечения, включающую клиентскую, серверную и административную части;
- реализовать механизм регистрации заявок с возможностью передачи фото- и текстовых данных;
- внедрить модуль интеллектуальной обработки изображений для автоматического извлечения контактных данных и классификации заявок;
- разработать подсистему уведомлений и автоматической обратной связи с пользователями и исполнителями;
- обеспечить защиту персональных данных и устойчивость системы при массовых обращениях;
- провести тестирование и оценить эффективность работы приложения в условиях реального использования.

Результатом работы является программный комплекс, который объединяет функции службы приёма заявок, автоматической классификации обращений и управления их исполнением. Предлагаемая система может быть адаптирована для различных сфер деятельности, где требуется автоматизация процессов технической поддержки и взаимодействия с пользователями.

Глава 1 Анализ процесса обработки заявок в технической поддержке

1.1 Анализ ситуации

В современном мире автоматизация процессов является ключевым фактором повышения эффективности работы организаций и предприятий. Одной из наиболее востребованных областей автоматизации является техническая поддержка пользователей, поскольку именно здесь возникает наибольшее количество рутинных операций, занимающих значительное количество ресурсов и времени сотрудников службы поддержки. Автоматизация процесса обработки заявок позволяет существенно сократить затраты на обслуживание клиентов, повысить качество предоставляемых услуг и снизить нагрузку на операторов технической поддержки [14].

Основная задача службы технической поддержки заключается в обеспечении качественного обслуживания пользователей, своевременном устранении технических проблем и предоставлении необходимой помощи пользователям [11]. Автоматизация процессов обработки заявок с использованием методов машинного анализа текста позволит существенно повысить эффективность и качество работы службы технической поддержки, минимизируя участие оператора.

Основные преимущества автоматизации включают. Повышение скорости реакции: Автоматизированные системы позволяют быстрее обрабатывать запросы, снижая среднее время ожидания пользователей. Стандартизация процедур: Создание стандартных шаблонов решений и процессов помогает снизить количество ошибок и обеспечить единообразие подхода к решению однотипных проблем [16].

Оптимизация ресурсов. Система распределяет нагрузку между сотрудниками равномерно, позволяя эффективно управлять рабочими ресурсами. Анализ и отчетность: Сбор статистики по обращениям позволяет выявлять

наиболее частые проблемы и оптимизировать работу службы. Улучшение качества сервиса: Быстрая реакция и четкое выполнение всех этапов работы повышают уровень удовлетворённости пользователей.

Что такое «заявка» и «обращение пользователя»? Заявка - это документированная форма запроса пользователя о возникшей проблеме или потребности в обслуживании. Заявка фиксирует факт возникновения инцидента или необходимость внесения изменений в систему.

Обращение пользователя - любое сообщение, направленное пользователем в службу технической поддержки с целью получения консультации, устранения неисправностей или изменения конфигурации. Обращение может содержать описание проблемы, требования, пожелания или жалобы.

Задача технического специалиста состоит в обработке поступившей заявки или обращения таким образом, чтобы устранить проблему и удовлетворить запрос пользователя.

Существует несколько типов служб поддержки, каждая из которых ориентирована на специфический круг задач и пользователей:

ИТ-поддержка. Это служба, обеспечивающая поддержку информационно-технологических инфраструктур организаций. Её сотрудники занимаются решением вопросов, связанных с компьютерами, сетью, программным обеспечением и оборудованием. Такие подразделения часто организуются по принципу уровней («Help Desk», второй и третий уровни поддержки).

Городские сервисы. Службы городского уровня обеспечивают техническую поддержку населению города в рамках городских услуг (например, подача воды, электроснабжение, ЖКХ). Они принимают заявки от жителей относительно состояния инфраструктуры, проводят диагностику и решают возникающие проблемы.

Сервисные центры. Сервисные центры специализируются на оказании ремонтных услуг физическим лицам и организациям. Они занимаются ремонтом бытовой техники, электроники, автомобилей и другого оборудования. Основной

задачей является устранение поломок и восстановление работоспособности устройств.

1.2 Жизненный цикл заявки

Процесс жизненного цикла заявки традиционно делится на ряд последовательных шагов (рисунок 1).

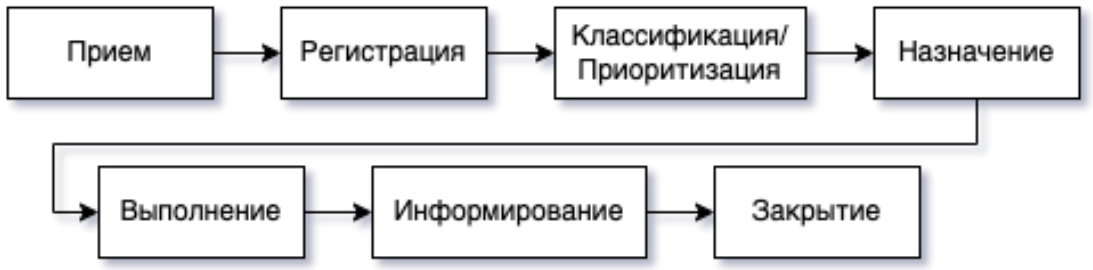

Рисунок 1 - Жизненный цикл

Рассмотрим подробнее каждый этап стандартного жизненного цикла.

Приём обращения. Пользователь обращается в службу поддержки различными способами: телефон, электронная почта, чат, портал самообслуживания и др. Важно зафиксировать обращение оперативно и точно передать его содержание исполнителю [1].

Регистрация в системе. Оператор вводит заявку в специализированную систему управления заявками (Service Desk, Help Desk и подобные). Здесь фиксируется первичная информация: контакты заявителя, суть проблемы, приоритет, прикрепленные файлы (фотографии, скриншоты, тексты и прочие полезные материалы).

Классификация и приоритизация. Определяются категория проблемы (аппаратная ошибка, сбой ПО, консультативный вопрос и т.п.) и её

важность/срочность. Чем выше приоритет, тем быстрее выполняется работа над обращением.

Назначение исполнителя. Задача автоматически или вручную назначается сотруднику службы поддержки, ответственному за решение конкретной категории проблемы. Иногда требуется эскалация задачи другому специалисту или команде.

Выполнение работ. Исполнитель проводит диагностику, предпринимает шаги по устранению проблемы, консультирует пользователя или принимает меры для восстановления функциональности. Все выполненные действия регистрируются в системе, включая приложенные инструкции, отчёты, заметки и т.д.

Информирование пользователя. По мере продвижения заявки пользователю отправляется уведомление о статусе работы (начало обработки, промежуточные этапы, завершение). Важна прозрачность и понятность сообщений для клиента.

Закрытие заявки. После успешного завершения работы исполнитель закрывает заявку в системе. Если проблема решена, запрашивается обратная связь от пользователя. Статистика по закрытой задаче используется для анализа эффективности работы команды и улучшения будущих процессов.

1.3 Классификация обращений и требования к современным HelpDesk-системам

Типы обращений в системах технической поддержки классифицируют следующим образом. Инцидент (Incident). Непредвиденное событие, нарушающее нормальную работу информационной системы или услуги. Примеры: отключение сервера, сбой приложения, потеря данных.

Запрос (Request for Change, RFC). Запрос на внесение запланированных изменений в инфраструктуру или процессы. Например, установка нового программного обеспечения, изменение настроек сети.

Проблема (Problem). Описание потенциальной причины множества инцидентов. Проблемы требуют углубленного анализа для выявления коренных причин повторяющихся сбоев.

Обращение пользователя (User Request). Простое взаимодействие с поддержкой, связанное с консультациями, запросами справочной информации, разъяснениями рабочих процессов.

Кроме того, выделяют вспомогательные виды обращений. Заявка на обслуживание (Service Request) - регулярная плановая процедура, такая как замена расходников, обновление лицензий. Жалоба (Complaint) - недовольство работой сервисов или персонала.

Большую роль при обработке обращений также играет приоритет. Классифицируют четыре основных вида:

- Низкий приоритет - незначительные проблемы, которые не влияют на работу;
- Средний приоритет - стандартные проблемы, требующие внимания;
- Высокий приоритет - серьёзные проблемы, которые влияют на работу;
- Критический - проблема не терпящих отлагательства, необходимо срочное реагирование.

Требования к современным HelpDesk-системам. Современные HelpDesk-системы должны соответствовать ряду требований, обеспечивающих эффективное управление процессом поддержки. Скорость реакции - минимальное время отклика на каждое обращение пользователя. Современные системы используют автоответчики, автоматическое распределение задач и автоматизацию рутинных операций.

Удобство интерфейса - удобный интерфейс облегчает навигацию и использование системой всеми участниками процесса: операторами, администраторами, конечными пользователями. Хорошо продуманная система должна поддерживать разные устройства и браузеры.

Безопасность - надёжная защита персональных данных и корпоративной информации. Необходимо соблюдение стандартов защиты данных (GDPR и т.д.).

Масштабируемость - возможность расширения функционала и количества обрабатываемых обращений по мере роста бизнеса. Модульная архитектура и интеграция с облачными платформами упрощают масштабирование.

Интеграция с другими системами - интеграция с CRM, ERP, мониторинговыми инструментами и прочими бизнес-приложениями улучшает координацию действий внутри предприятия.

SLA (Service Level Agreement) - поддержка соглашений об уровне обслуживания гарантирует установленные сроки исполнения задач, регламентированные правила приоритета и порядок предоставления услуг.

Управление доступностью - доступность системы даже при пиковых нагрузках и внезапных проблемах. Резервирование серверов, отказоустойчивые архитектуры необходимы для бесперебойной работы [2].

Отчётность и аналитика - формирование подробных отчётов о работе отдела поддержки, статистика обработанных заявок, производительность операторов, показатели KPI (Key Performance Indicators).

1.4 Анализ существующих систем обработки заявок

Ниже представлен краткий анализ четырех популярных систем управления заявками.

Jira Service Management - система предназначена для управления техническими услугами и проектами, особенно популярна среди разработчиков программного обеспечения и крупных компаний.

Ключевые функции:

- Управление инцидентами и проблемами.
- Гибкая кастомизация форм заявок и полей.
- Мощные инструменты аналитики и отчетности.
- Полноценная интеграция с экосистемой Atlassian.

Ограничения:

- Высокая стоимость подписки для больших команд.

– Сложность настройки и освоения для небольших предприятий.

– Ориентация на крупные проекты и менее удобна для малого бизнеса.

OTRS (Open-source Ticket Request System) - бесплатная открытая система для обработки клиентских запросов и внутренней коммуникации, широко используемая государственными учреждениями и малыми предприятиями.

Ключевые функции:

– Подготовка писем клиентам с автоматическими ответами.

– Отчетность и анализ эффективности сотрудников.

– Поддержка многоязычности и интеграции с почтовыми клиентами.

Ограничения:

– Ограниченные возможности кастомизации UI.

– Для малых и средних проектов функциональность избыточна.

– Отсутствие нативной мобильной версии.

GLPI (GNU Licence Public Inventory Project) - фиксирование инвентаря и поддержка заявочных систем в организациях среднего размера, востребована в образовательных учреждениях и государственных органах.

Ключевые функции:

– Инвентарный учет аппаратуры и активов.

– Контроль затрат и контрактов на оборудование.

– Управление правами доступа и распределением ролей.

Ограничения:

– Интерфейс требует доработки для повышения удобства использования.

– Недостаточная гибкость в настройке процессов поддержки.

– Низкая популярность в коммерческих компаниях.

Zendesk - коммерческая платформа для управления взаимодействием с клиентами, преимущественно ориентирована на крупные бизнесы и call-центры.

Ключевые функции:

– Многофункциональная поддержка чатов, звонков, тикетов.

– Глубокая аналитика и мониторинг обратной связи.

– Высокий уровень интеграции с сторонними сервисами.

Ограничения:

– Цена подписки высока для начинающих и мелких компаний.

– Большое количество функций усложняет обучение сотрудников.

– Настройка занимает значительное время и ресурсы.

Несмотря на разнообразие представленных систем, ни одна из них не способна идеально решить весь спектр задач нашего проекта. Каждая из рассмотренных систем обладает своими ограничениями, такими как высокая цена, сложность настройки, недостаточные возможности кастомизации или узкая специализация, отсутствие ИИ обработчика [3]. Именно поэтому разработка собственной системы позволит учесть уникальные потребности и спецификации проекта, обеспечивая максимальную эффективность и экономичность реализации поставленных целей.

1.5 Обоснование необходимости собственной разработки продукта

Разработка собственного программного обеспечения для автоматизации обработки заявок оправданна несколькими важными факторами, которые делают готовое решение неподходящим или неэффективным для наших конкретных потребностей.

Необходимость адаптации под конкретный процесс. Готовое ПО редко соответствует уникальным внутренним процессам организации. Зачастую оно поддерживает лишь общие схемы обработки заявок, игнорируя специфику и нюансы локальных процедур. Следовательно, возникает потребность в разработке инструмента, способного адаптироваться под наши собственные регламенты и структуры процессов.

Встроенный анализ текста и интеграция ИИ. Сегодня большинство систем управления заявками не оснащены средствами для автоматизированного анализа текста, хотя этот аспект критически важен для ускорения диагностического этапа. Наш собственный продукт позволит реализовать продвинутый модуль

компьютерного зрения, поддерживаемый искусственным интеллектом, который способен самостоятельно анализировать текст, помогая сократить время на классификацию обращения.

Простота и универсальность решения для городской среды. Города предъявляют особенные требования к системам поддержки: необходима простота и быстрота работы, совместимость с разными группами пользователей, от муниципальных служащих до обычных горожан. Большая часть готовых решений перегружена функциональностью, которую трудно освоить или настроить под потребности местных органов власти и широкий круг потребителей [4].

Доступность веб-интерфейса. Чтобы обеспечить доступность системы для широкого круга пользователей, важно наличие полноценного веб-интерфейса, работающего на любом устройстве. Мы создадим своё приложение, которое будет доступно через браузер, без ограничений по типу ОС или устройству пользователя.

Поддержка мобильных устройств. Использование мобильных устройств стало частью повседневной практики технической поддержки. Своё решение позволит развернуть полноценную мобильную версию системы, что повысит оперативность и эффективность обработки заявок.

Требования к системе и масштабируемость. Современная система должна обеспечивать высокую степень надёжности, защищённость данных и готовность к росту нагрузки. Масштабируемость является одним из ключевых критериев проектирования, позволяющим расширить функционал и увеличить объём обработки заявок по мере роста организации. Грамотно спроектированное собственное решение даст возможность наращивать мощности без значительных финансовых вложений.

Глава 2 Проектирование программного обеспечения

2.1 Постановка задачи и определение функций системы

Система должна выполнять следующие основные функции:

- Приём и регистрация заявок. Получение заявок от пользователей и их фиксация в системе.
- Классификация и приоритизация. Распределение заявок по категориям и присвоение уровня срочности.
- Распределение задач. Назначение исполнителей и распределение ответственности.
- Контроль исполнения. Мониторинг хода выполнения заявок и поддержание актуальной информации о статусе.
- Коммуникация с пользователями. Уведомления и обратная связь для поддержания прозрачности процесса.
- Архивация и отчетность. Хранение истории заявок и формирование статистической отчетности.

Участники процесса представлены в таблице 1.

Таблица 1 - Участники процесса

Роль	Описание
Пользователь	Лицо, подавшее заявку на оказание технической поддержки. Может подавать новые заявки, просматривать статус текущих и архивных обращений.
Оператор	Сотрудник первой линии поддержки, принимающий заявки, проводящий предварительную диагностику и перенаправляющий сложные случаи другим подразделениям.
Исполнитель	Специалист, ответственный за выполнение заявки
Администратор	Специалист, ответственный за конфигурацию системы, ведение базы данных, назначение прав доступа и контроль общего состояния системы.

Ключевые функции системы. Приём заявок. Пользователи могут подать заявку через web-интерфейс. Данные сразу попадают в систему и сохраняются для дальнейшей обработки.

Классификация и приоритизация. Каждое новое обращение проходит процедуру категоризации и присваивания приоритета в зависимости от степени важности и срочности.

Диспетчеризация. Система назначает оператора или группу специалистов, которые отвечают за данную категорию заявок.

Исполнение и коммуникация. Специалисты выполняют необходимые действия, а система информирует пользователей о ходе выполнения заявки.

Отчеты и архивы. Вся информация сохраняется в единой базе данных, доступна для формирования отчётов и статистики по показателям работы службы поддержки.

2.2 Функциональные требования

Для определения ключевых возможностей разрабатываемой системы технической поддержки необходимо сформировать полный перечень функций, обеспечивающих её корректную и эффективную работу. Функциональные требования отражают то, как пользователи будут взаимодействовать с системой, каким образом будут обрабатываться их обращения, а также как операторы и администраторы смогут управлять заявками, контролировать их выполнение и обеспечивать надёжность сервиса. Правильное формирование набора функциональных требований позволяет заранее определить структуру будущего программного обеспечения, определить границы системы, избежать неоднозначностей на этапе реализации и обеспечить соответствие конечного продукта ожиданиям пользователей. В таблице ниже представлены функции, без которых полноценная работа системы обработки заявок невозможна, а также краткое

описание каждой из них [5]. Функциональные требования представлены в таблице 2.

Таблица 2 - Функциональные требования

Функция	Описание
Регистрация пользователей	Возможность самостоятельной регистрации новых пользователей
Вход в систему	Авторизация зарегистрированных пользователей
Создание заявки	Возможность подачи заявки пользователем
Загрузка вложений	Добавление файлов (фотографий, документов) к заявке
Просмотр статусов	Проверка текущего статуса заявки
Работа оператора	Назначение сотрудника для рассмотрения заявки
Изменение статуса	Переход заявки на следующий этап (открыта, в работе, закрыта)
Уведомления по e-mail	Отправка уведомлений о событиях в заявке на электронную почту
История изменений	Логирование всех изменений статуса, комментариев и действий
Фильтрация и поиск заявок	Возможности сортировки и поиска по различным параметрам
Административные права	Управленческий доступ к изменению полномочий пользователей и управлению системой
Экспорт данных	Возможность выгрузки информации в файловые форматы (.csv, .pdf)
Безопасность	Средства аутентификации, шифрования и защиты конфиденциальных данных
Анализ данных	Генерация отчётов и статистических показателей по состоянию заявок
Вход в систему	Авторизация зарегистрированных пользователей

Представленные функциональные требования формируют основу работы разрабатываемой системы и определяют набор действий, доступных пользователям, операторам и администраторам. Реализация перечисленных функций обеспечивает полноценный цикл обработки заявок — от их создания и классификации до изменения статуса и формирования отчётности. Именно на основании этих требований далее выполняется проектирование архитектуры, выбор технологий и разработка пользовательских интерфейсов, что позволяет создать удобное, надёжное и устойчивое программное решение.

2.3 Нефункциональные требования

Нефункциональные требования определяют характеристики системы, которые не связаны напрямую с её функциональным поведением, но существенно влияют на качество работы, удобство использования, устойчивость и безопасность. Эти требования обеспечивают надёжность и эффективность эксплуатации системы в реальных условиях, а также задают ориентиры для проектирования архитектуры, выбора технологий и проведения тестирования. Ниже представлены основные нефункциональные требования, которые необходимо учитывать при разработке программного обеспечения для автоматизации обработки обращений пользователей.

Производительность - система должна обеспечивать стабильную работу при высокой интенсивности обращений и одновременной активности большого количества пользователей. Основные показатели производительности включают:

- время отклика сервера не более 2 секунд при нагрузке в 100 одновременно работающих пользователей;
- обработка заявок и выполнение операций должны происходить без заметных задержек;
- использование механизмов кеширования для ускорения доступа к frequently используемым данным.

Поддержание указанных характеристик необходимо для обеспечения комфортной работы пользователей и операторов, а также для стабильного функционирования системы в условиях пиковых нагрузок.

Безопасность - система должна обеспечивать защиту персональных данных и исключать возможность несанкционированного доступа. К основным требованиям относятся:

- использование SSL-шифрования при передаче данных между клиентом и сервером;

- хранение паролей в зашифрованном виде с применением современных криптографических алгоритмов;
- разграничение прав пользователей в зависимости от их ролей (пользователь, оператор, администратор);
- защита от наиболее распространённых угроз, таких как SQL-инъекции, подбор паролей и XSS-атаки.

Безопасность является ключевым аспектом системы, так как связано с обработкой конфиденциальной информации и взаимодействием большого количества пользователей.

Доступность - система должна быть доступна пользователям в любое время суток, обеспечивая бесперебойный доступ к основным функциям:

- система работает круглосуточно, без ограничений по времени доступа;
- профилактические и технические работы выполняются преимущественно в ночные часы, чтобы минимизировать влияние на пользователей;
- должна быть предусмотрена возможность быстрого восстановления системы в случае сбоев.

Высокий уровень доступности напрямую влияет на качество обслуживания и своевременность обработки пользовательских заявок.

Юзабилити - интерфейс системы должен быть понятным и удобным для пользователей с разным уровнем подготовки. Основные требования:

- интуитивно понятная навигация и ясная структура интерфейса;
- минимальная необходимость обучения новым пользователям;
- корректное отображение интерфейса на различных устройствах и размерах экранов;
- соблюдение принципов современной UX/UI-дизайн-практики.

Хорошая юзабилити значительно снижает количество ошибок пользователей и повышает скорость работы с системой.

Тестируемость - система должна быть легко тестируемой, что обеспечивается:

- наличием разработанных тест-кейсов для проверки всех ключевых компонентов;
- возможностью автоматизированного тестирования отдельных модулей;
- прозрачной структурой кода для быстрого выявления и устранения ошибок.

Высокая тестируемость облегчает поддержку системы и ускоряет процесс развития продукта.

Масштабируемость - система должна поддерживать возможность расширения и увеличения нагрузки без потери производительности. К требованиям относятся:

- возможность увеличения количества одновременно работающих пользователей;
- увеличение объёма обрабатываемых данных без существенных изменений архитектуры;
- гибкая адаптация модулей к росту количества обращений.

Масштабируемость позволяет системе оставаться актуальной и эффективной при расширении организации или увеличении числа пользователей.

2.4 Проектирование архитектуры системы

Клиентская часть (Frontend). Веб-интерфейс, через которые пользователи и операторы взаимодействуют с системой. Реализован на современном MVC фреймворке Yii2 с использованием REST API для коммуникаций со сторонним сервером. Модели фреймворка получают данные от контроллеров и используя технологию twig наполняют HTML страницу [5].

Серверная часть (Backend). Серверные модули, осуществляющие основную логику обработки заявок, аутентификацию и управление сессиями. Также реализован на фреймворке Yii2.

Хранилище данных (Database). База данных для хранения информации о заявках, пользователях, операциях и действиях. Используется MySQL для реляционных данных и Redis для кеширования и временных хранилищ.

Подсистема уведомлений. Встроенный почтовый сервер служит для отправки email-уведомлений.

Модули искусственного интеллекта (ИИ). Отдельный микросервис для интеллектуального анализа текстового текста. Основаны на API GigaChat.

Архитектура системы представлена на рисунке 2.

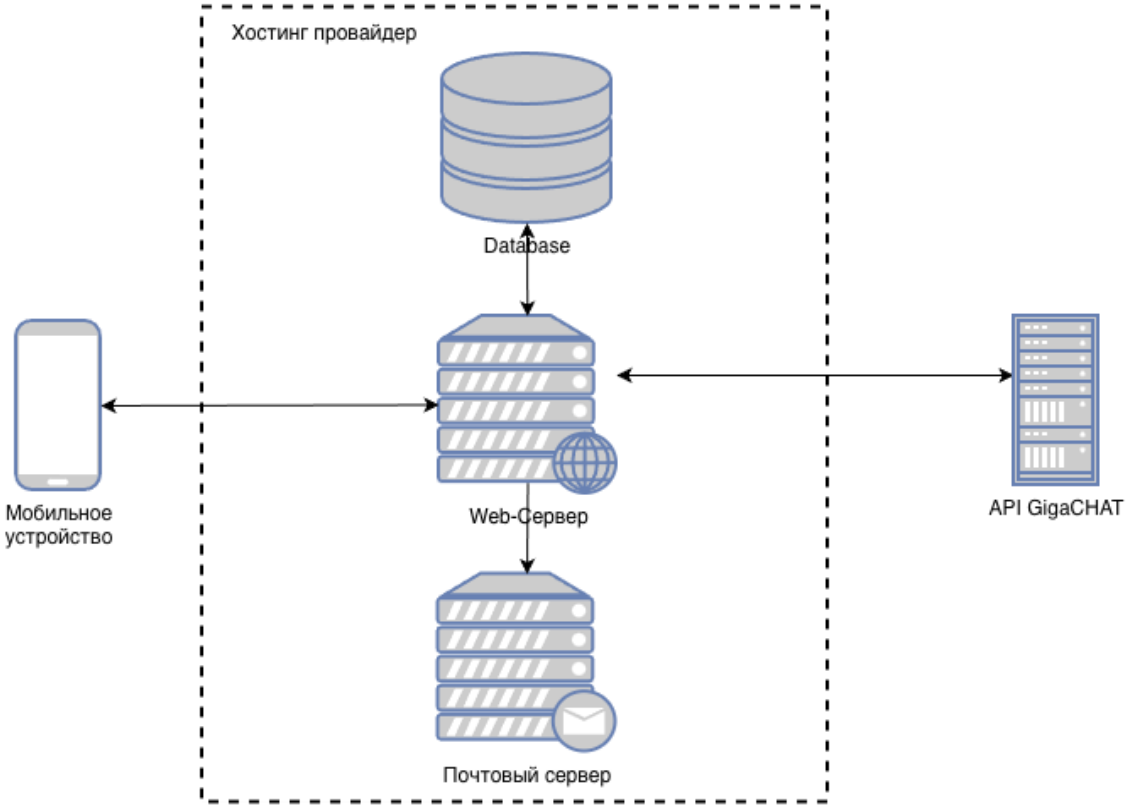

Рисунок 2 - Архитектура системы

Данная архитектура гибко масштабируется и, при повышении нагрузки, может быть перенесена на отдельный сервер.

2.5 Проектирование структуры базы данных

Опишем структуру таблиц и создадим их, используя интерфейс PhpMyAdmin - позволяющий управлять базой данный MySQL.

Состав полей и тип таблицы «Пользователь» представлен в таблице 3.

Таблица 3 - Пользователь (User)

Название	Тип	Длина
id	int	11
firstname	varchar	100
lastname	varchar	100
status	int	1
username	varchar	100
password	varchar	100
auth_token	varchar	32
password_reset_token	varchar	255
role	int	1
created_at	int	11
updated_at	int	11

Подробнее о полях:

1. id - уникальный идентификатор записи в таблице
2. firstname - имя пользователя
3. lastname - фамилия пользователя
4. status - статус пользователя (1 - активный/ 0 - не активный)
5. username - он же логин для авторизации, он же почта пользователя
6. password - зашифрованный пароль пользователя созданный через php-функцию password_hash() которая создаёт хеш пароля через сильный необратимый алгоритм хеширования
7. auth_token - служебное поле будет использоваться для восстановления пароля
8. password_reset_token - служебное поле будет использоваться для восстановления пароля

9. role - роль пользователя (1 - пользователь (по умолчанию), 2 - оператор, 3 - исполнитель, 4 - администратор)
10. created_at - временная метка создания
11. updated_at - временная метка изменения данных

При регистрации пользователя изначально создается запись с ролью «Пользователь». Изменить роль может только пользователь с ролью «Администратор».

Состав полей и тип таблицы «Заявка» представлен в таблице 4.

Таблица - 4 Заявка (Ticket)

Название	Тип	Длина
id	int	11
id_user	int	11
id_category	int	11
status	int	2
priority	int	2
description	text	
theme	varchar	255
created_at	int	11
updated_at	int	11

Подробнее о полях:

1. id - уникальный идентификатор записи в таблице
2. id_user - id пользователя из таблицы «Пользователи»
3. id_category - id записи из таблицы «Категории»
4. status - статус обращения
5. priority - приоритет
6. description - описание проблемы
7. theme - тема
8. created_at - временная метка создания данных
9. updated_at - временная метка изменения данных

Состав полей и тип таблицы «Категория» представлен в таблице 5. В данной таблице хранится классификация обращений.

Таблица 5 - Категория (Category)

Название	Тип	Длина
id	int	11
name	varchar	100
created_at	int	11
updated_at	int	11

Подробнее о полях:

1. id - уникальный идентификатор записи в таблице
2. name - название типа обращения
3. created_at - временная метка создания данных
4. updated_at - временная метка изменения данных

Состав полей и тип таблицы «История» представлен в таблице 6. В данной таблице хранится история движения заявки.

Таблица 6 - История (History)

Название	Тип	Длина
id	int	11
id_ticket	int	11
id_user	int	11
description	text	
created_at	int	11
updated_at	int	11

Подробнее о полях:

1. id - уникальный идентификатор записи в таблице
2. id_ticket - идентификатор заявки
3. id_user - идентификатор пользователя

4. description - описание выполненной работы
5. created_at - временная метка создания данных
6. updated_at - временная метка изменения данных

2.6 Диаграммы проектирования

Диаграмма вариантов использования представлена на рисунке 3.

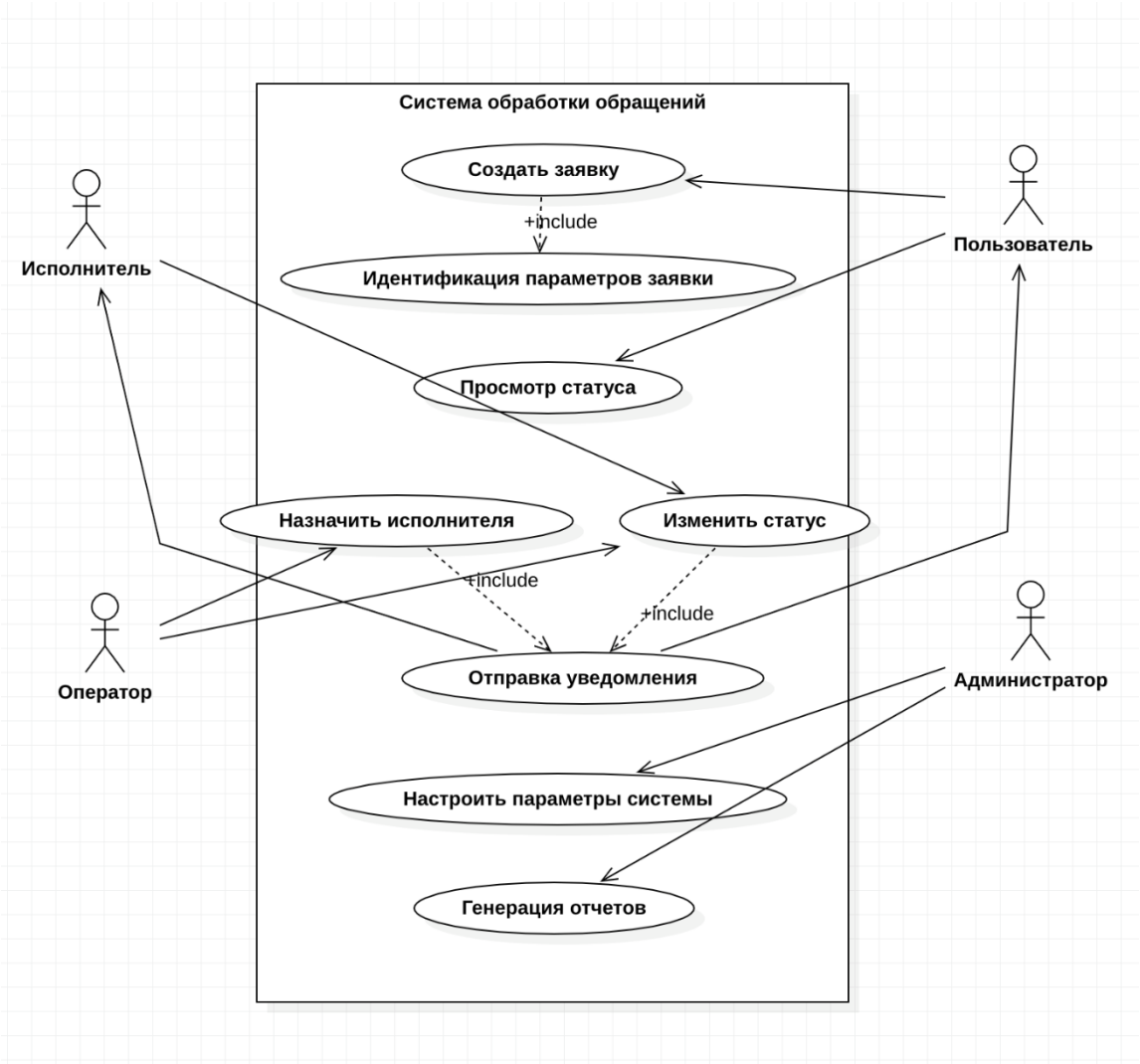

Рисунок 3 - Диаграмма вариантов использования системы обработки обращений

Акторы:

Пользователь - лицо, инициирующее подачу заявки на техническую поддержку. Основное взаимодействие связано с созданием и контролем заявок.

Оператор - первоначальную идентификацию обращения выполняет ИИ определяя категорию, суть и приоритет. Если определить не удалось, то сотрудник службы поддержки, выполняет вторичную идентификацию [6].

Администратор - специалист, управляющий системой, настраивающий параметры и контролирующий работоспособность всей инфраструктуры.

Исполнитель - специалист службы поддержки, занимающийся проработкой заявки.

Варианты использования (Use Cases)

Создать заявку: Пользователь создаёт новую заявку в системе.

Просмотреть статус заявки: Пользователь проверяет текущее состояние своей заявки.

Изменить статус заявки: Оператор обновляет статус заявки в зависимости от хода выполнения.

Назначить исполнителя: Оператор назначает ответственного сотрудника для обработки заявки.

Отправить уведомление: Система автоматически отправляет уведомления пользователю о состоянии заявки.

Настроить параметры системы: Администратор конфигурирует настройки системы, определяет права доступа и управляет структурой данных.

Генерация отчёта: Администратор формирует отчёты по активности пользователей и статистике обработки заявок.

Отметка о выполнении: Исполнитель меняет статус заявки оставляя при этом комментарий о ходе выполнения.

UML-диаграмма взаимодействия таблиц представлена на рисунке 4.

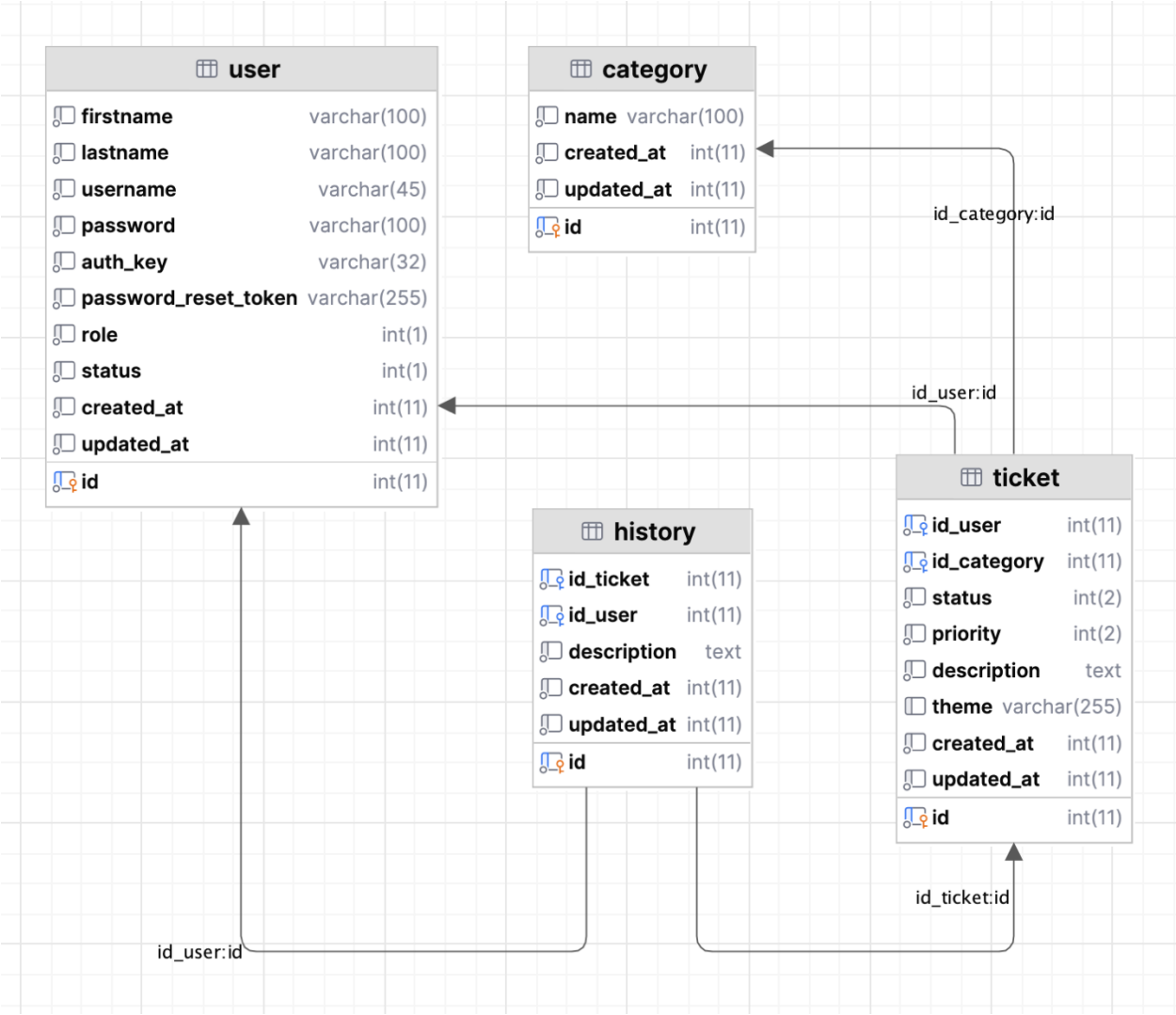

Рисунок 4 - UML-диаграмма

Диаграмма наглядно показывает зависимость таблицы ticket от таблиц user и category через поле id, хранящее в себе уникальные идентификаторы. Так же таблица history хранит идентификаторы таблиц user и ticket.

2.7 Проектирование пользовательских интерфейсов

Проектирование пользовательских интерфейсов является важнейшим этапом разработки программного обеспечения, поскольку именно через интерфейс пользователь взаимодействует с системой и оценивает её удобство, функциональность и качество работы. Интерфейс должен быть интуитивно

понятным, простым в освоении и адаптированным под разные устройства, особенно если система ориентирована на широкую аудиторию без специальных технических навыков [7].

Одной из ключевых задач проектирования является создание визуальной структуры, которая позволяет пользователю быстро находить необходимые функции, легко переходить между разделами и выполнять основные операции без дополнительных подсказок. При разработке интерфейса учитывались современные требования UX/UI-дизайна, принципы эргономики, минимизация переходов и удобство использования.

Так как разрабатываемое приложение ориентировано на мобильные устройства, большое внимание уделено адаптивности и корректному отображению элементов на экранах различного размера. Минимальная ширина контента составляет 380 пикселей, максимальная — 780 пикселей, при этом структура интерфейса выстроена таким образом, чтобы все элементы оставались доступными, а навигация — предельно удобной. Контент располагается по центру, а элементы управления сгруппированы логично и последовательно.

В рамках проектирования были созданы макеты, отражающие ключевые страницы приложения: главная страница с переходом ко входу, форма регистрации, окно авторизации, личный кабинет пользователя, форма создания новой заявки, страница просмотра заявки и интерфейс настройки профиля. Макеты позволяют визуально оценить будущую структуру приложения и служат основой для последующей разработки.

Ниже представлены основные прототипы пользовательских интерфейсов (рисунки 5–6), демонстрирующие внешний вид ключевых страниц приложения.

Представленные макеты интерфейсов позволяют сформировать чёткое представление о логике взаимодействия пользователя с системой и демонстрируют основные функции, доступные в мобильном приложении. На основании визуальных прототипов можно заблаговременно оценить удобство расположения элементов, прозрачность навигации, читабельность текстовой информации и общий пользовательский опыт.

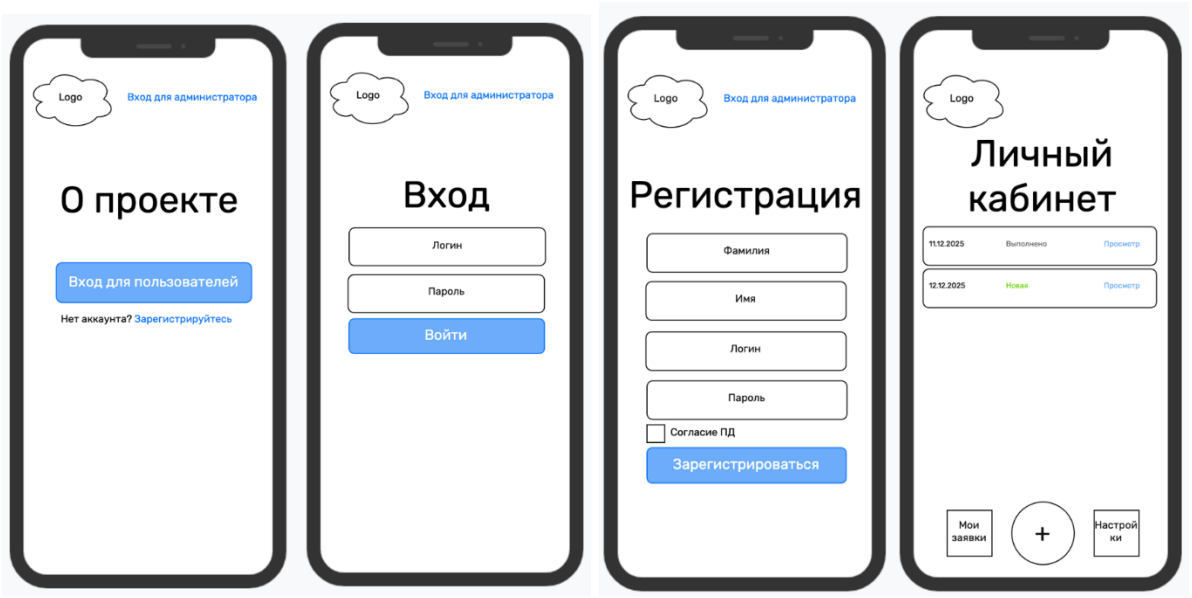

Рисунок 5 - Главная страница, вход для пользователей, регистрация и личный кабинет

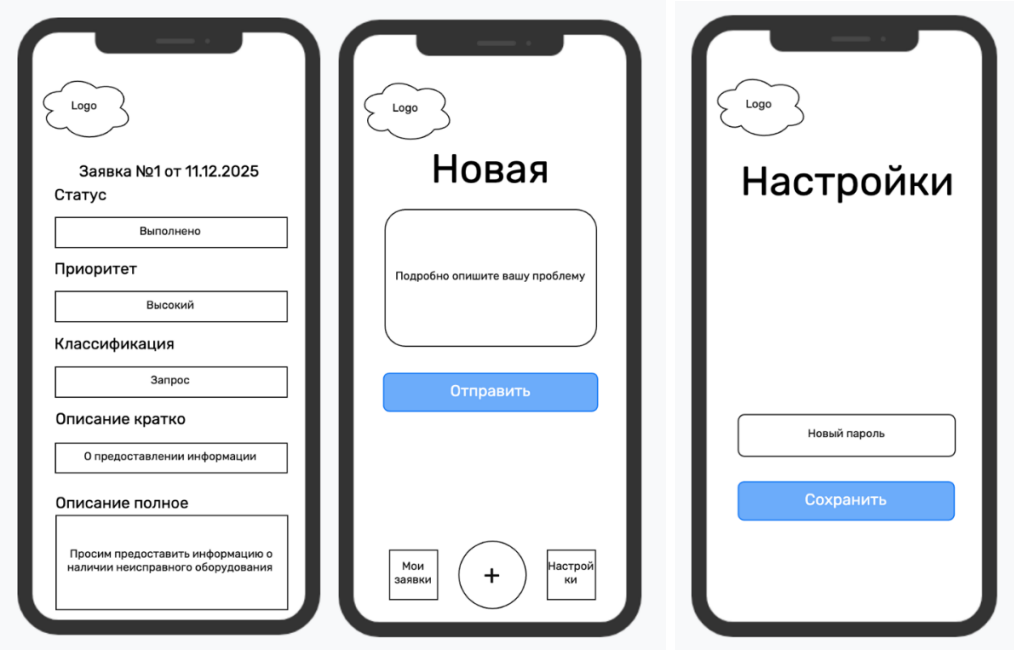

Рисунок 6 - Новая заявки, просмотр и настройки профиля

Главная страница и форма авторизации показывают минималистичный стиль интерфейса, где пользователю предлагается сразу выбрать необходимое

действие: вход в систему или переход к регистрации. Форма регистрации содержит только ключевые поля, что снижает время заполнения и повышает удобство работы.

Личный кабинет пользователя предоставляет доступ к списку всех созданных заявок, их статусов, а также позволяет перейти к просмотру детальной информации по каждой заявке. Экран создания новой заявки включает простую форму ввода и кнопку отправки, что позволяет быстро оформить обращение. Аналогично экран просмотра заявки содержит структурированную информацию о статусе, приоритете и содержании обращения, что обеспечивает прозрачность выполнения работ [8].

Интерфейс настроек профиля реализован в лаконичном виде и позволяет пользователю оперативно изменять пароль или другие личные данные. Все элементы интерфейса выдержаны в едином стиле, что обеспечивает единообразие взаимодействия и способствует положительному пользовательскому опыту.

Таким образом, созданные прототипы интерфейсов выполняют роль визуальной спецификации и служат основой для реализации полноценного графического интерфейса приложения, позволяя оптимизировать этап разработки и заранее исключить потенциальные ошибки в организации пользовательского взаимодействия.

Глава 3 Реализация программного обеспечения

3.1 Выбор технологий и инструментов разработки

Основными компонентами системы являются:

- Веб-интерфейс для ввода и отображения информации.
- Система аутентификации и авторизации пользователей.
- Управление базой данных с возможностью записи, редактирования и удаления записей.
- Интеллектуальная обработка данных с использованием внешних модулей, таких как GigaChat.

Критерии выбора технологий

Перед выбором конкретных технологий необходимо сформулировать критерии, на основании которых будем проводить сравнение и отбор инструментов:

- Простота освоения и поддержка документации.
- Скорость разработки и простота модификации.
- Надежность и производительность.
- Поддержка интеграции с внешними сервисами.
- Стоимость владения и доступность библиотек.
- Сообщество разработчиков и активность проектов.

Теперь рассмотрим каждую технологию отдельно, уделяя особое внимание вышеуказанным критериям.

Язык программирования - PHP

PHP - это скриптовый язык программирования, широко используемый для разработки веб-приложений. Основные преимущества PHP:

- Высокая популярность и большое сообщество разработчиков.
- Широкая совместимость с популярными хостинг-провайдерами.
- Большое количество готовых библиотек и фреймворков.

- Простота синтаксиса и низкий порог входа.
- Критический взгляд на PHP показывает следующие минусы:
- Медленная производительность по сравнению с некоторыми современными языками (например, Go или Node.js).
- Трудности с поддержкой устаревшего кода и совместимостью версий.

Тем не менее, учитывая поставленную задачу и важность наличия простого и хорошо документированного решения, PHP представляется оптимальным вариантом.

Фреймворк - Yii2 Framework

Yii2 - это объектно-ориентированный MVC-фреймворк для PHP, предназначенный для быстрой разработки веб-приложений. Ключевые особенности Yii2:

- Высокий уровень абстракции и удобная архитектура MVC.
- Наличие мощных инструментов для ORM (Object Relational Mapping), миграции базы данных и AJAX-запросов.
- Хорошее покрытие тестирования и высокая производительность.
- Полностью локализованный и интернационализированный код.

Преимущества:

- Легкость в освоении и отличная документация.
- Гибкая настройка конфигураций и быстрое развертывание.
- Отличная производительность и хорошая защита от SQL-инъекций.

Недостатки:

- Небольшое сообщество по сравнению с Laravel или Symfony.
- Несколько сложнее разобраться новичкам, хотя поддержка документации частично компенсирует этот недостаток.

Учитывая, что Yii2 предлагает отличную производительность и хорошую документацию, он идеально подходит для нашего проекта.

СУБД - MySQL

MySQL - это свободная реляционная система управления базами данных, использующая язык SQL для взаимодействия с данными.

Основные плюсы MySQL:

- Открытый исходный код и бесплатная лицензия.
- Высокая производительность и низкая ресурсоемкость.
- Хорошо интегрирована с большинством хостингов и CMS.
- Подходит для небольших и средних проектов.

Однако у MySQL есть и недостатки:

- Недостаточное масштабирование для больших объемов данных.
- Менее мощная экосистема репликации и кластеризации по сравнению с PostgreSQL.

Для наших нужд MySQL вполне достаточна и проста в использовании, что оправдывает её выбор. В дальнейшем, используя технологию ORM предоставляемую фреймворком Yii2, мы сможем с легкостью переключиться на другую реляционную базу данных без необходимости переписывать запросы [9].

Интеграция модуля распознавания текста - GigaChat

Модуль распознавания текста, созданный на основе GigaChat, предназначен для интеллектуального анализа и интерпретации текстовых данных.

Преимущества:

- Возможность глубокого понимания контекста и смысла текстов.
- Подходящие алгоритмы обработки естественного языка (NLP).
- Легкость интеграции и минимальные требования к аппаратным ресурсам.

Хотя у GigaChat пока нет широких сообществ разработчиков, сам продукт уже зарекомендовал себя как мощный инструмент для работы с текстом.

Кеш - Redis

Преимущества кэширования данных в Redis (хранения часто используемых данных в оперативной памяти) включают высокую скорость доступа, гибкость работы с различными типами данных, масштабируемость и отказоустойчивость.

Эти преимущества важны для оптимизации производительности приложений, особенно в условиях высокой нагрузки.

3.2 Конфигурирование приложения

Для реализации дипломной работы и проведения тестов был приобретён хостинг у хостинг-провайдера Beget. А также выкуплен служебный домен, посвящённый дипломной работе abs-diploma.ru.

Для домена установлен бесплатный SSL-сертификат от компании Let’s encrypt, который перекрывает нужды разрабатываемого проекта в части безопасной передачи данных.

Выполним инициализацию проекта, она включает в себя установку Framework’a Yii2 и дополнений. Для этого в инструментах хостинга (SSH-консоль) выполним команду на скачивание пакета (рисунок 7).

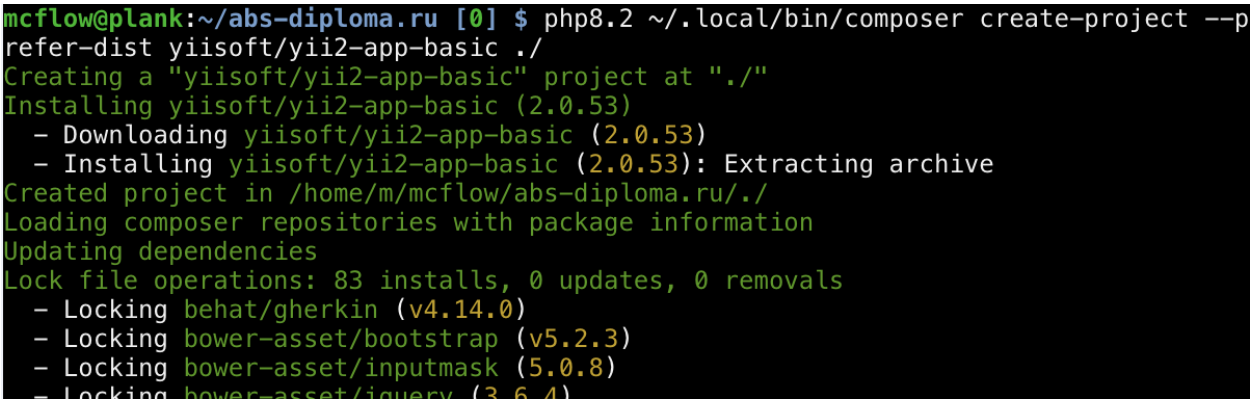

Рисунок 7 - Листинг установки фреймворка Yii2

Командой php8.2 ~/.local/bin/composer require davidxu/yii2-adminlte4 установим дополнительный пакет AdminLTE.

AdminLTE - это бесплатный шаблон административной панели с открытым исходным кодом построенный на Bootstrap 5.

Экран приветствия на главной странице сайт abs-diploma.ru (рисунок 8) говорит об успешной операции инициализации проекта.

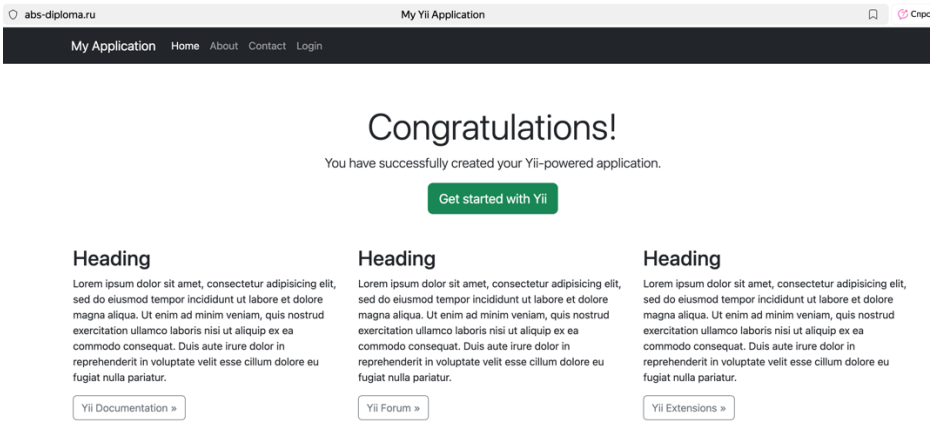

Рисунок 8 - Yii2 успешно установлен

Встроенные почтовый сервер хостинга позволяют отправлять email уведомления с помощью php функции mail(), но она не использует протокол SMTP, а это чревато плохой доставляемостью писем до адресата. Yii2 же наоборот работает с SMTP, но для этого нужно предварительно настроить DKIM. Технология DKIM - это метод email-аутентификации. Заголовок DKIM-Signature прикрепляется к письму, а в самом DKIM зашифрована информация о домене отправителя. Так DKIM-Signature подтверждает его.

Используем DKIM генератор чтобы сгенерировать ключ и добавим в настройки DNS на хостинге новую TXT-запись с именем beget._domainkey.abs-diploma.ru и ранее полученным ключом [10].

После того как настройки аутентификации почты вступили в силу в интерфейсе хостинга (рисунок 9) мы создадим служебный email-адрес robot@abs-diploma.ru и внесем данные в конфигурацию сайта.

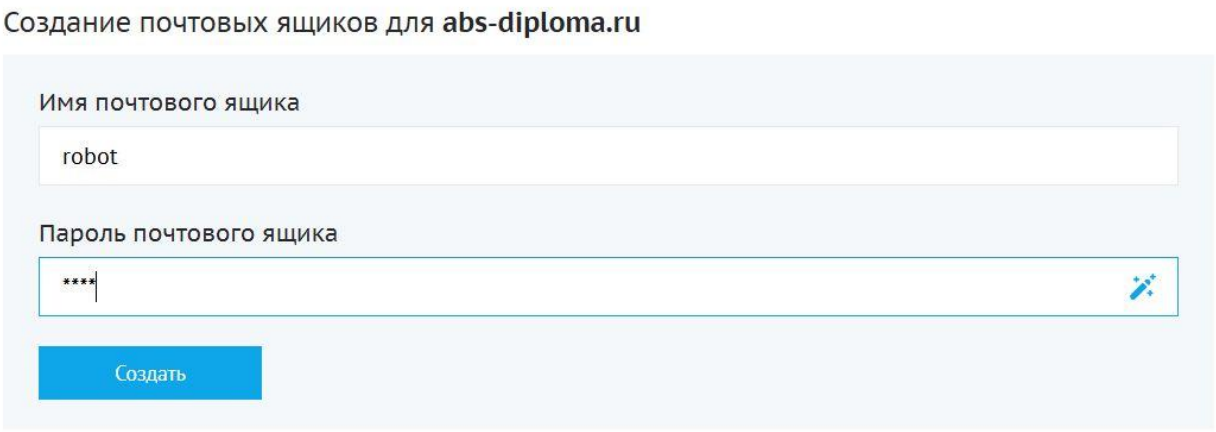

Рисунок 9 - Создание нового почтового ящика

Конфигурация email представлена в листинге на рисунке 10.

```
'mailer' => [
    'class' => yii\symfonymailer\Mailer::class,
    'viewPath' => '@app/mail',
    'transport' => [
        'class'    => Swift_SmtpTransport::class,
        'host'     => 'smtp.beget.com',
        'username' => 'robot@abs-diploma.ru',
        'password' => '******',
        'port'     => 465,
        'encryption' => 'ssl',
    ],
],
```

Рисунок 10 - Конфигурация email

Завершим конфигурирование нашего сайта добавлением базы данных в интерфейсе хостинга. Полученные доступы пропишем в конфигурационном файле. Пример конфигурации представлен в листинге на рисунке 11.

```php
return [
    'class'    => yii\db\Connection::class,
    'dsn'      => 'mysql:host=localhost;dbname=mcflow_absd',
    'username' => 'mcflow_absd',
    'password' => '******',
    'charset'  => 'utf8',
];
```

Рисунок 11 - Конфигурация DB

Представленный фрагмент конфигурации отвечает за настройку подключения приложения к базе данных. В параметрах указывается класс соединения yii\db\Connection, строка подключения (DSN) с типом используемой СУБД и адресом сервера, а также имя пользователя, пароль и кодировка соединения. В данном проекте применяется СУБД MySQL, что отражено в DSN-строке. Заданные параметры обеспечивают корректное установление соединения с базой данных, доступ к таблицам и выполнение основных операций чтения и записи. Такая конфигурация является стандартной для приложений, построенных на Yii2, и позволяет гибко управлять настройками соединения как на этапе разработки, так и при развертывании приложения на сервере [11].

3.3 Структура проекта и организация модулей

В основе разработанного приложения используется фреймворк Yii2, который предоставляет встроенный инструмент для ускорения разработки — модуль Gii. Gii представляет собой мощный генератор кода, позволяющий автоматически создавать модели, контроллеры, CRUD-интерфейсы, формы и отдельные модули. Его применение значительно снижает количество рутинной работы, ускоряет процесс разработки и позволяет сосредоточиться на проектировании логики приложения.

Одним из преимуществ Gii является его гибкость: разработчик может как использовать типовые генераторы, так и модифицировать создаваемые шаблоны под собственные требования. Например, при генерации модели таблицы Gii автоматически создаёт соответствующий ActiveRecord-класс с атрибутами, правилами валидации и связями, что упрощает дальнейшую работу с данными [12].

Доступ к Gii по умолчанию ограничен из соображений безопасности. Для предотвращения несанкционированного использования инструмент разрешён только для доверенных IP-адресов. Чтобы включить доступ, необходимо внести соответствующие изменения в конфигурационный файл приложения, указав список разрешённых адресов. Такая политика позволяет использовать Gii в процессе разработки, но исключает возможность обращения к нему в production-среде.

В процессе создания системы Gii использовался регулярно и стал важным инструментом при организации структуры проекта. На его основе были созданы основные элементы приложения, включая модели базы данных, формы для обработки данных пользователей, а также заготовки контроллеров, реализующих работу ключевых модулей.

С архитектурной точки зрения проект разделён на несколько самостоятельных модулей, что обеспечивает удобство сопровождения и расширяемость. В системе выделены два ключевых модуля:

Модуль пользовательского кабинета. Предназначен для обработки действий обычных пользователей: создание заявок, просмотр статусов, загрузка вложений, управление профилем. Внутри модуля размещены контроллеры, отвечающие за основные операции, а также представления, адаптированные под мобильную версию интерфейса [20].

Модуль административной панели. Этот модуль включает функционал оператора и администратора: просмотр всех заявок, изменение статусов, назначение исполнителей, просмотр статистики и формирование отчётов.

Разделение модулей обеспечивает логичную организацию структуры и разграничивает доступ согласно ролям пользователей.

Подобное модульное построение значительно упрощает масштабирование системы: при необходимости можно без труда добавить новый функционал, создав отдельный модуль, не нарушая работы существующих компонентов. Модули имеют независимые пространства имён и собственные контроллеры, что делает архитектуру приложения более гибкой и удобной для дальнейшего развития.

Интерфейс создания модуля представлен на рисунке 12.

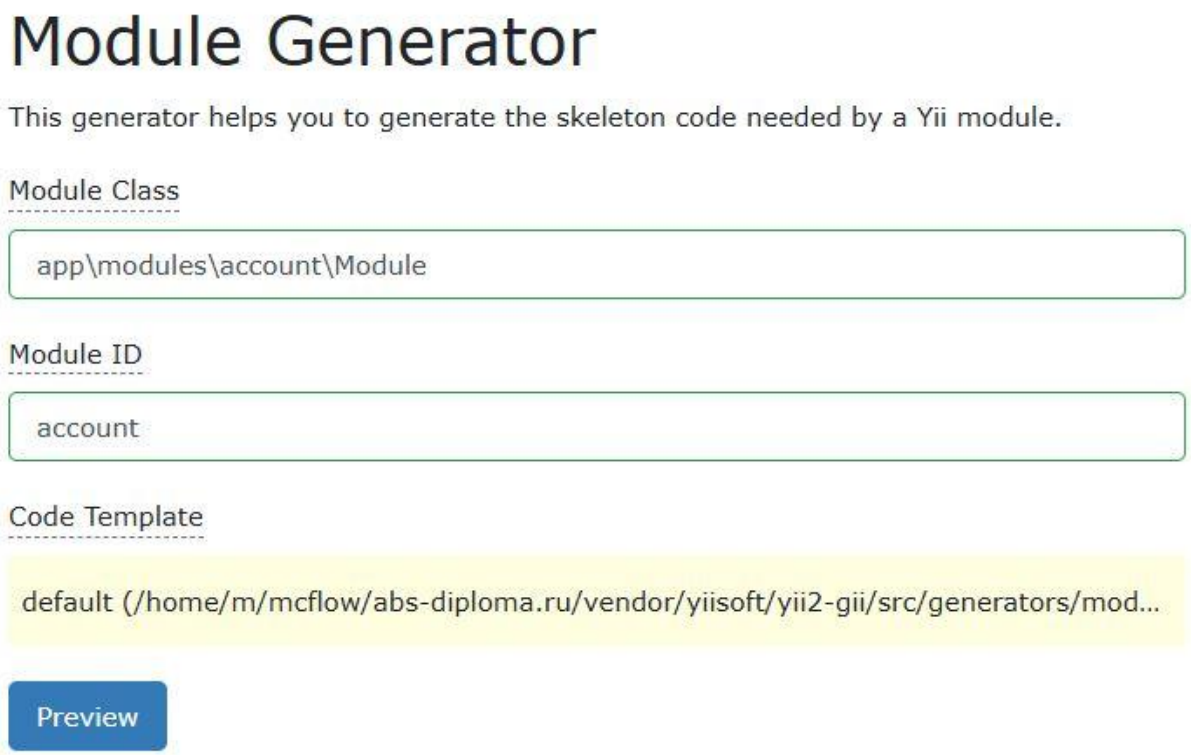

Рисунок 12 - Интерфейс создания модуля для пользователя.

После нажатия кнопки «preview» и «generate» Gii - автоматически создает файлы и кладет их на хостинг. Далее, чтобы модуль работал необходимо внести в код конфига информацию о модуле (листинг на рисунке 13). Данный код так же генерируется Gii, нам необходимо лишь вставить его.

```
'modules' => [
    'account' => [
        'class' => app\modules\account\Module::class,
    ],
],
```

Рисунок 13 - Добавление модуля

Проделаем ту же самую операцию для модуля администратора. Module ID будет иметь название admin721.

На основе наших таблиц в Gii создадим ORM модели. ORM - технология, которая позволяет разработчикам работать с реляционными базами данных (СУБД) через объекты языков программирования. Она создаёт «мост» между объектно-ориентированными языками программирования и реляционными базами данных, преобразуя данные между двумя парадигмами [19].

Чтобы создать модель достаточно просто выбрать из списка таблицу, для которой она создается (фреймворк имеет доступ к названиям таблиц). Остальные данные уже предзаполнены. Интерфейс создания модели на основе таблицы ticket представлен на рисунке 14.

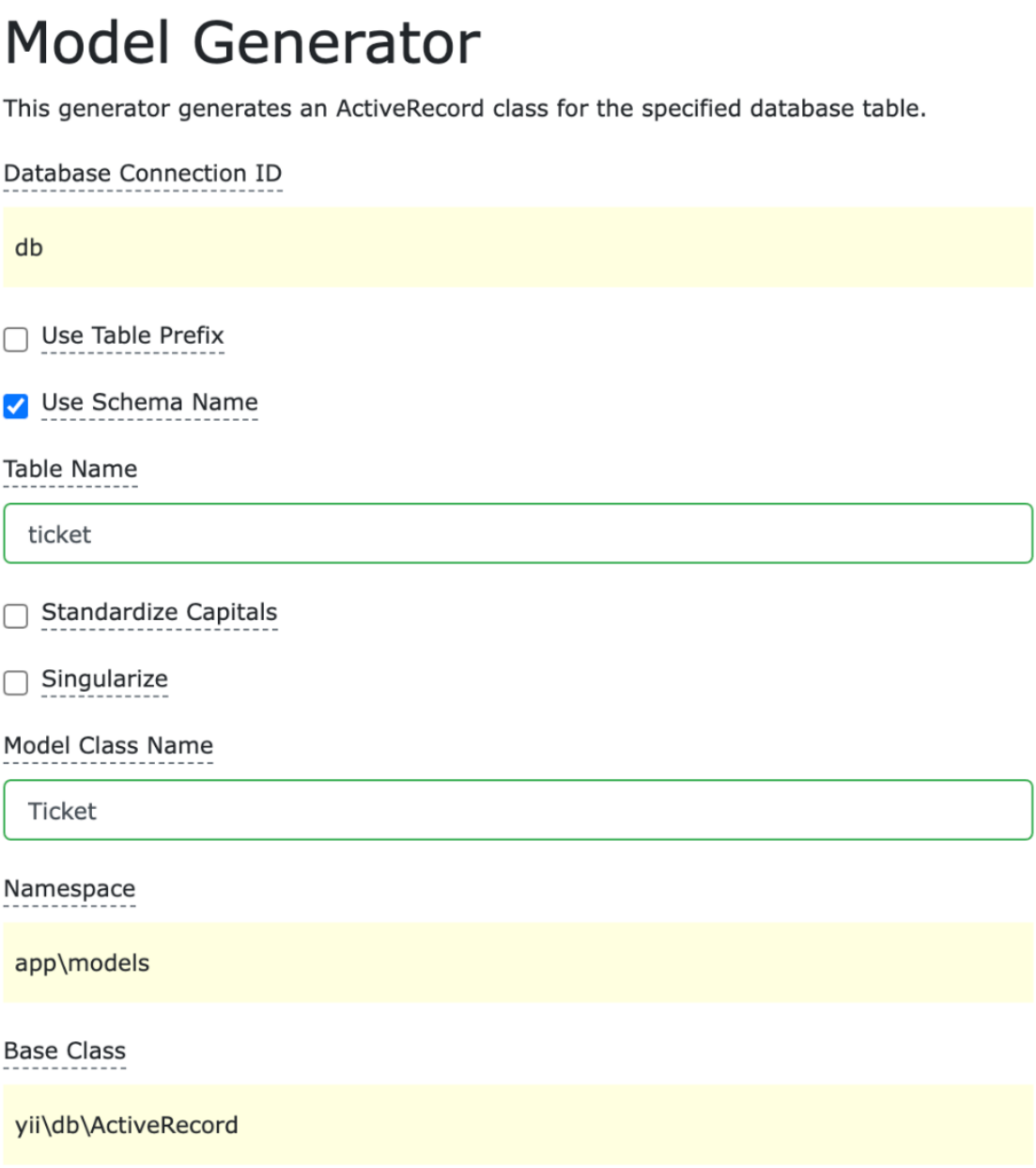

Рисунок 14 - Создание ORM для таблицы ticket

Сгенерированный системой код представлен в листинге на рисунке 15.

```php
public function rules()
{
    return [
        [['theme'], 'default', 'value' => null],
        [['status'], 'default', 'value' => 1],
        [['priority'], 'default', 'value' => 0],
        [['id_category'], 'default', 'value' => 7],
        [['id_user', 'id_category', 'description'], 'required'],
        [['id_user','id_category','status','priority','created_at','updated_at'], 'integer'],
        [['description'], 'string'],
        [['theme'], 'string', 'max' => 255],
        [['id_category'], 'exist', 'targetClass' => Category::class, 'targetAttribute' => ['id_cat
        [['id_user'], 'exist', 'targetClass' => User::class, 'targetAttribute' => ['id_user'=>'id'
    ];
}

public function attributeLabels()
{
    return [
        'id' => 'Ид.', 'id_user' => 'Пользователь', 'id_category' => 'Категория',
        'status' => 'Статус', 'priority' => 'Приоритет', 'description' => 'Содержание',
        'theme' => 'Тема', 'created_at' => 'Создано', 'updated_at' => 'Обновлено'
    ];
}

public function getPriorityName(): string
{
    return [0=>'Низкий',1=>'Средний',2=>'Высокий',3=>'Критичный'][$this->priority];
}

public function getStatusName(): string
{
    return [1=>'Новая',2=>'Требуется модерация',3=>'Выполняется',4=>'Выполнено'][$this->status];
}
```

Рисунок 15 - Код ORM-модели Ticket

Умные алгоритмы Yii2, видя, что у таблицы ticket есть внешние ключи, создали вспомогательные функции getUser(), getHistories() и getCategory(). Они помогут получать данные из одноименных таблиц без дополнительный запросов, добавляя только вызов функцию with() при построении запроса ORM [18]. Для функции getHistories() мы вручную добавили сортировку по дате создания. Это пригодиться нам в дальнейшем при выборке одного обращения с элементами истории.

Класс TimestampBehavior - это вспомогательный класс, который автоматически вставляет и обновляет поля таблицы created_at и updated_at при создании и обновлении записи.

Так же добавим дополнительные функции getPriorityName() и getStatusName() помогающий получить названия приоритета и статуса по его коду.

Создадим аналогичные модели для трех остальных таблиц user, category и history. Таблицу category заполним данными, классифицирующими запрос пользователя (рисунок 16).

id	name	created_at	updated_at
1	Инцидент	1763616800	1763616800
2	Запрос	1763616800	1763616800
3	Проблема	1763616800	1763616800
4	Обращение	1763616800	1763616800
5	Обслуживание	1763616800	1763616800
6	Жалоба	1763616800	1763616800
7	Не определено	1763616800	1763616800

Рисунок 16 - Обзор данных таблицы category

За проверку прав доступа в личный кабинет отвечает класс AuthController представленный в листинге на рисунке 17.

```php
<?php
namespace app\controllers;

use yii\web\Controller;
use yii\filters\AccessControl;
use yii\filters\VerbFilter;

class AuthController extends Controller
{
    public function behaviors()
    {
        return [
            'access' => [
                'class' => AccessControl::class,
                'rules' => [
                    [
                        'allow' => true,
                        'roles' => ['@'],   // доступ только авторизованным пользователям
                    ],
                ],
            ],

            'verbs' => [
                'class' => VerbFilter::class,
                'actions' => [
                    'delete' => ['POST'], // удаление только через POST-запрос
                ],
            ],
        ];
    }
}
```

Рисунок 17 - AuthController

Основные правила указаны в массиве rules. Данная конструкция означает, что доступ к контроллеру, который наследует AuthContoroller только через авторизацию [17].

Для административной панели имеется аналогичный контроллер, за исключением одного правила, проверяющего роль пользователя из таблицы пользователи (листинг на рисунке 18).

```
'matchCallback' => fn() => Yii::$app->user->identity->role === 3,
```

Рисунок 18 - Проверка прав оператора

Представленный фрагмент демонстрирует механизм проверки прав доступа на уровне контроллера, что позволяет ограничить выполнение определённых действий только для пользователей с соответствующей ролью. Такой подход обеспечивает более высокий уровень безопасности и исключает возможность обращения к административному функционалу со стороны неподготовленных или неавторизованных пользователей. Использование matchCallback делает логику проверки гибкой и расширяемой, позволяя адаптировать правила доступа по мере развития системы и появления новых ролей или сценариев работы [16].

3.4 Реализация функционала пользователя

Реализация функционала пользователя является ключевым этапом разработки системы, так как именно через личный кабинет осуществляется основное взаимодействие с приложением. На данном уровне обеспечивается регистрация и авторизация пользователей, создание заявок, просмотр их статуса, загрузка вложений, а также управление персональными данными. Функциональность должна быть реализована таким образом, чтобы обеспечить удобство работы, корректность выполнения операций и безопасность обработки информации [15]. Структура классов и модулей, отвечающих за пользовательский функционал, представлена ниже и служит основой для понимания внутренней логики работы системы. Общий вид модели классов личного кабинета представлен на рисунке 19.

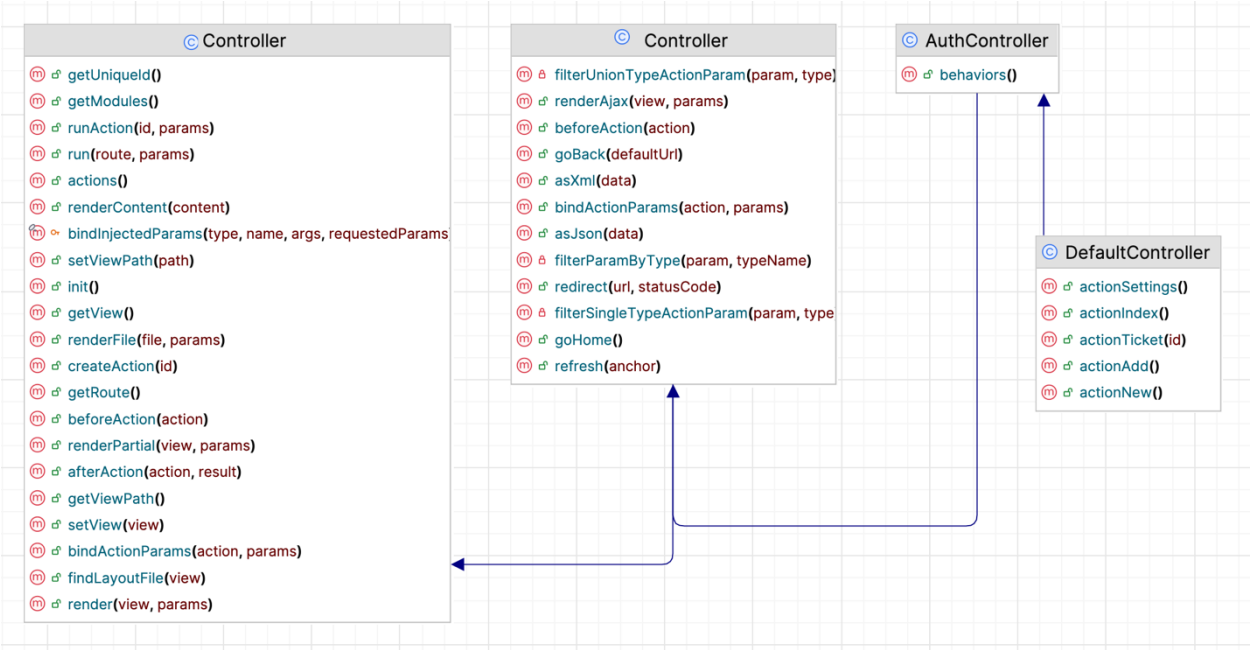

Рисунок 19 - Модель классов личного кабинета

Представленная диаграмма классов демонстрирует структуру контроллеров, участвующих в реализации пользовательского функционала, а также их связь с базовыми компонентами фреймворка Yii2. На схеме видно, что основная логика работы личного кабинета основана на наследовании базового контроллера и расширении его функциональности с помощью специализированных методов. Такой подход позволяет переиспользовать стандартные механизмы фреймворка, обеспечивая при этом гибкость и модульность архитектуры. Контроллер DefaultController отвечает за обработку основных операций, таких как отображение интерфейса личного кабинета, создание заявок и управление данными, тогда как AuthController обеспечивает контроль доступа и применение правил авторизации. Грамотно организованная модель классов упрощает поддержку системы, ускоряет внедрение новых возможностей и делает взаимодействие различных компонентов приложения более предсказуемым и надёжным [14].

Код контроллера личного кабинета представлен в листинге на рисунке 20.

```php
public function actionAdd()
{
    if (!Yii::$app->request->isPost) {
        return $this->redirect(['/account']);
    }

    $model = new Ticket();
    $model->load(Yii::$app->request->post());
    $model->id_user = Yii::$app->user->identity->id;
    $model->theme   = 'Неопределенно';

    if ($model->save()) {
        $gigachat = new Gigachat();
        $analyze  = $gigachat->analyzeText($model->description);

        if (!empty($analyze)) {
            $model->theme       = $analyze['theme']    ?? $model->theme;
            $model->priority    = $analyze['priority'] ?? 0;
            $model->id_category = $analyze['category'] ?? null;
            $model->status      = 3;   // обработано ИИ
        } else {
            $model->status = 2;       // требует ручной обработки
        }

        $model->save(false);
        Yii::$app->session->setFlash('success', 'Заявка создана успешно!');
    } else {
        Yii::$app->session->setFlash('error', 'Ошибка при создании заявки. Повторите позднее.');
    }

    return $this->redirect(['/account']);
}
```

Рисунок 20 - Контроллер личного кабинета

actionIndex() - отвечает за отрисовку главной страницы личного кабинета. Там же выполняется запрос на вывод всех обращений пользователя.

actionNew() - страница для нового обращения. Где пользователь подробно описывает суть обращения.

actionAdd() - данный метод вызывается при создании обращения. По умолчанию приоритет устанавливается в самый низкий, а категория еще не определена и имеет идентификатор 7 - не определено. После сохранения обращения, подключается класс GigaChat (подробнее о котором ниже) который в процессе распознавания текста понимает тему обращения, его приоритет и

срочность. Полученные в процессе анализа поля сохраняются в обращение и статус заявки становится «Выполняется». Если обработать текст не удалось, обращение получает статус «Требуется модерация» для ручного распознания оператором. Пользователь получает уведомление на почту [13].

actionTicket() - получает данные по одному обращению по его идентификатору и идентификатору авторизованного пользователя. Так мы закрываем доступ от просмотра чужих обращений.

actionSettings - служит для сохранения настроек.

3.5 Реализация функционала оператора технической поддержки

Для административного раздела мы воспользуемся еще одной встроенной функцией ферймворка Gii - CRUD Generator (C - create, R - read, U - update, D - delete). Она помогает создавать контроллеры на основе созданных ранее ORM-моделей. Интерфейс создания контроллера представлен на рисунке 21.

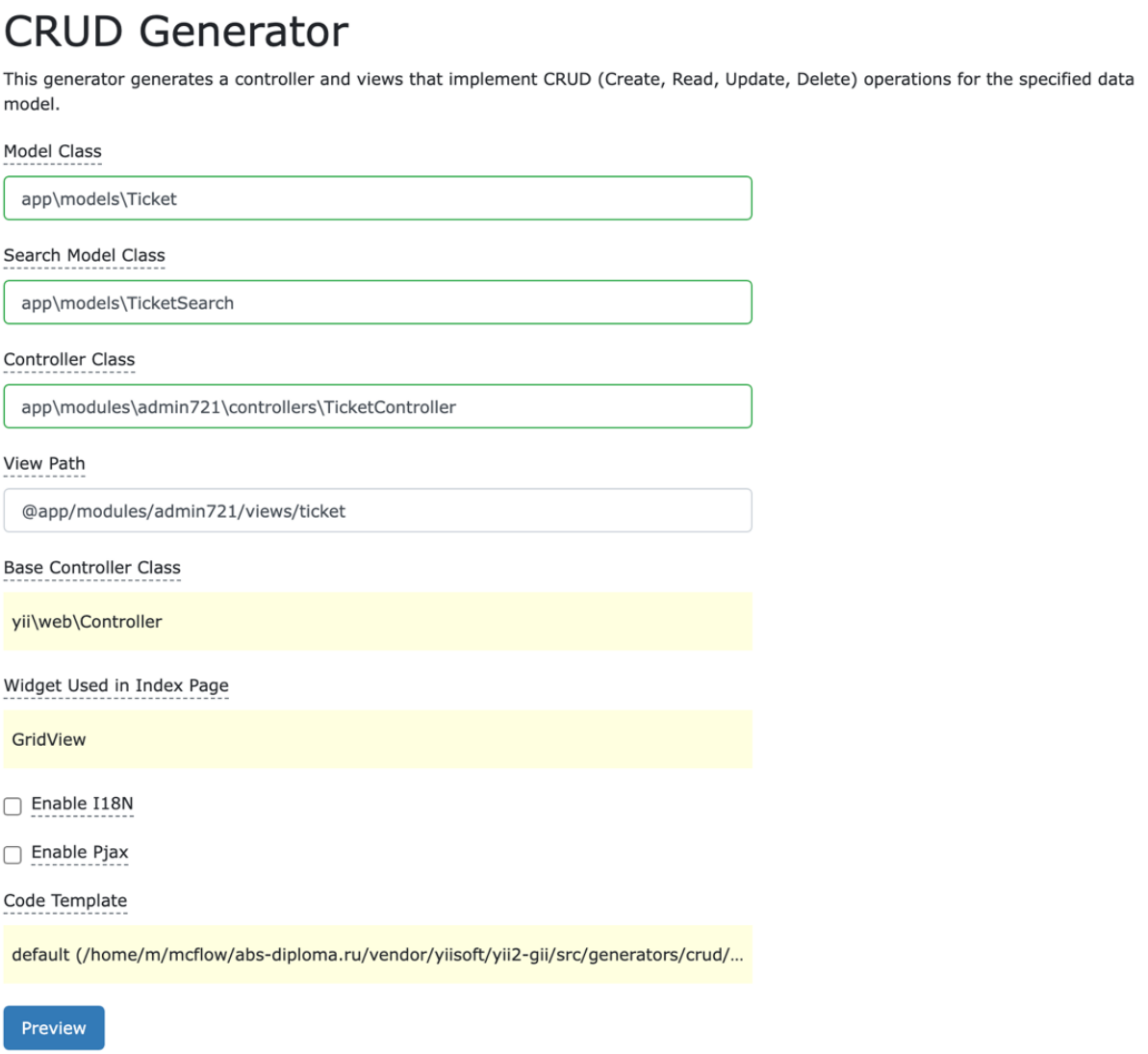

Рисунок 21 - Создание CRUD-контроллера Ticket

Необходимо указать модель, для которой мы создаем контроллер, поисковую модель (если ее нет, то она будет создана), расположение и название контроллера и файлов представления. При наличии поисковой модели в визуальное представление будет добавлен удобный раздел для поиска записей. Аналогичной операцией создадим контроллеры для администрирования пользователей и категорий [12].

Общий вид модели классов личного кабинета представлен на рисунке 22.

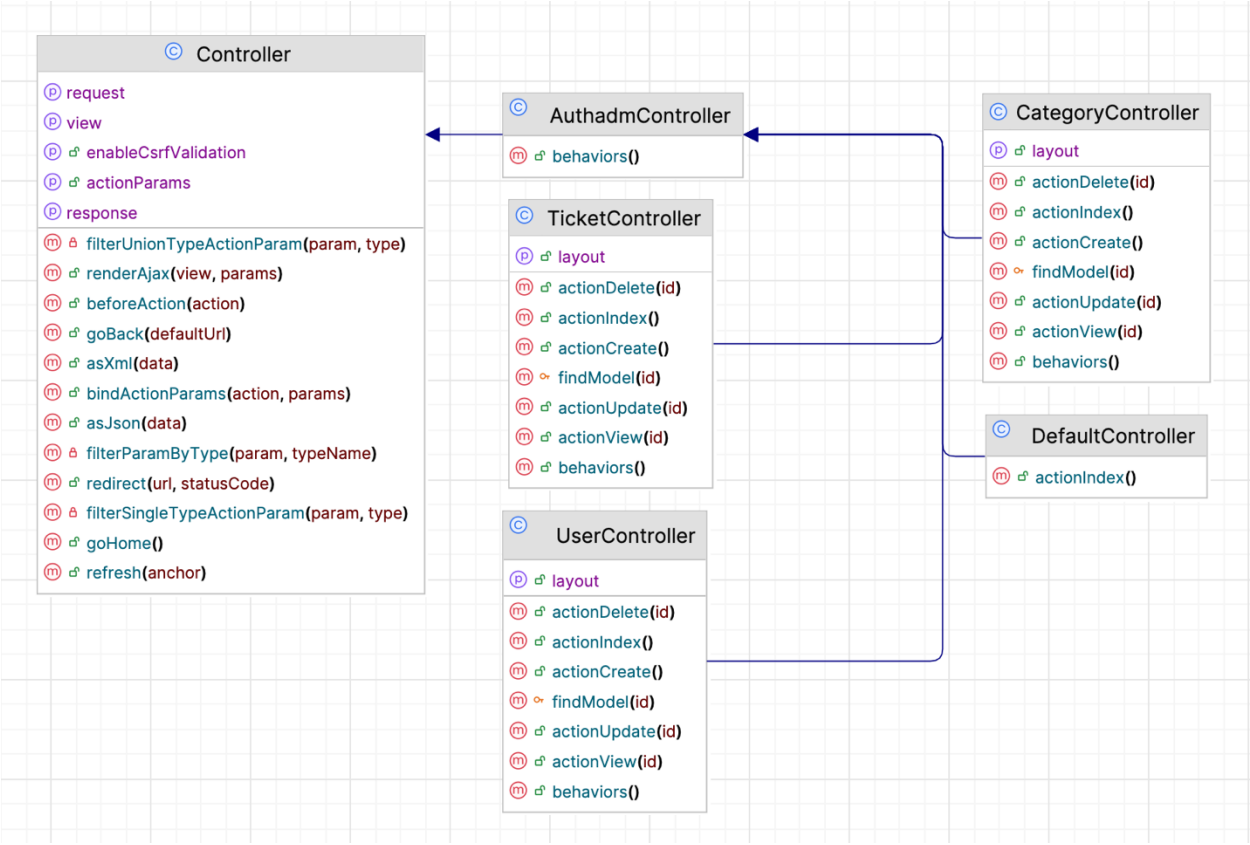

Рисунок 22 - Модель классов кабинета оператора

Здесь видно, что все CRUD-контроллеры наследуются от AuthadmController - контроллера проверки авторизации, который в свою очередь унаследован от базового контроллера. Контроллеры имеют стандартный набор методов для создания actionCreate(), чтения actionView(), редактирования actionUpdate() и удаления записей actionDelete(). DefaultController - основной контроллер который выполняется при входе в административный раздел [11].

В функции оператора входит просмотр, создание и редактирование запросов пользователя, а также просмотр и редактирование профиля пользователя.

3.6 Реализация механизма интеллектуальной обработки данных

Для интеграции с GigaChat был разработан класс, код которого представлен в листинге на рисунке 23.

```php
class Gigachat
{
    private const OAUTH_KEY = '******';
    private const PROMPT    =
        'Ты оператор техподдержки. Проанализируй запрос и верни JSON: ' .
        '{theme, priority, category}. Запрос: ';

    private Cache $cache;
    private Client $client;

    public function __construct()
    {
        $this->cache  = new Cache();
        $this->client = new Client(['verify' => false]);
    }

    public function getAccessToken(): string
    {
        return $this->cache->get('gigachat_token')
            ?? $this->requestAccessToken();
    }

    private function requestAccessToken(): string
    {
        $response = $this->client->post(
            'https://ngw.devices.sberbank.ru:9443/api/v2/oauth',
            [
                'headers' => [
                    'RqUID'        => (new UuidFactory())->uuid4()->toString(),
                    'Content-Type' => 'application/x-www-form-urlencoded',
                    'Authorization'=> 'Basic ' . self::OAUTH_KEY
                ],
                'form_params' => ['scope' => 'GIGACHAT_API_PERS']
            ]
        );

        $token = json_decode($response->getBody(), true)['access_token'];
        $this->cache->set('gigachat_token', $token, 1800);

        return $token;
    }
```

Рисунок 23 - Класс Gigachat

Метод getAccessToken() при использовании промежуточного класса Cache (который является оберткой для Radis) запрашивает access_token необходимый для выполнения запросов к API. Если токен не найден или он просрочен, выполняется инициализация в методе initAccessToken() и последующая запись его в кеш на 30 минут.

Метод analyzeText() анализирует обращение пользователя. В качестве prompt используем следующее предложение «Ты оператор технической поддержки. Прочитай запрос пользователя. Создай краткую тему данного запроса, максимум 200 символов. Классифицируй запрос цифрой от 1 до 4, где 1 - инцидент, 2 - запрос, 3 - проблема, 4 - обращение. Выбери приоритет от 0 до 3, где 0 - низкий, а 3 - критичный. Дай ответ в json разделенными по полям, где тема в ключе theme, приоритет в ключе priority, а классификатор запроса в ключе category. Запрос пользователя:». К этому вступлению добавляется сообщение пользователя и отправляется на анализ. В случае успешного распознавания текста API вернет нам объект, который мы преобразуем в массив. Пример ответа представлен в листинге на рисунке 24.

```
{'theme': 'Не запускается компьютер', 'priority': 3, 'category': 1}
```

Рисунке 24 - JSON ответа после обработки

Представленный фрагмент демонстрирует результат интеллектуальной обработки данных, выполняемой модулем анализа текста. На основе переданного пользователем описания система формирует структурированный JSON-объект, содержащий ключевые параметры заявки: определённую тему обращения, рассчитанный приоритет и предполагаемую категорию. Подобная автоматическая классификация позволяет существенно ускорить процесс обработки обращений, снижает нагрузку на операторов и обеспечивает более точное распределение заявок по соответствующим специалистам [9]. Полученные данные далее используются для предварительного заполнения

полей новой заявки и последующего выбора оптимального маршрута её обработки.

3.7 Схема взаимодействия компонентов системы

Диаграмма потоков данных представлена на рисунке 25. Она показывает потоки данных в системе и помогает лучше понять нефункциональные требования.

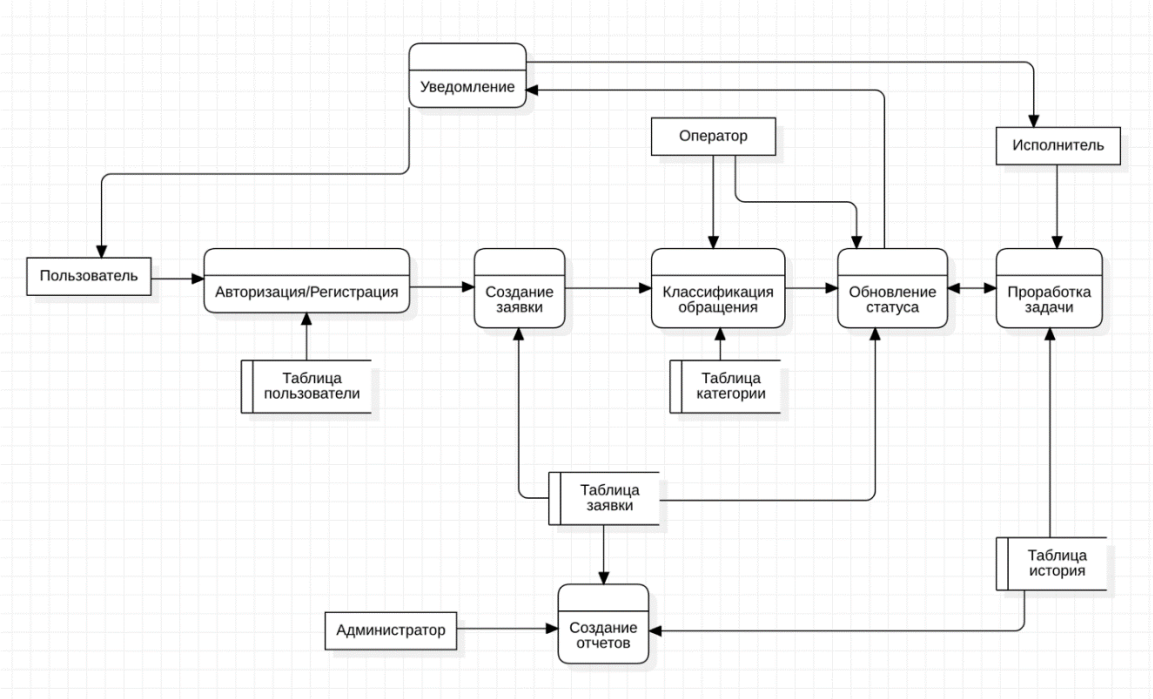

Рисунок 25 - DFD-диаграмма потоков данных

Процессы:

- Авторизация/регистрация пользователя
- Прием создание заявки
- Классификация заявки
- Обновление статуса
- Проработка задачи
- Формирование отчетов

– Уведомления

Хранилища данных:

– Базовая таблица пользователей
– Таблица заявки
– Таблица категории
– Таблица история

Внешние сущности:

– Пользователь
– Оператор
– Исполнитель
– Администратор

3.8 Интерфейсы пользователя и оператора

Интерфейс разработанного приложения представлен на рисунке 26.

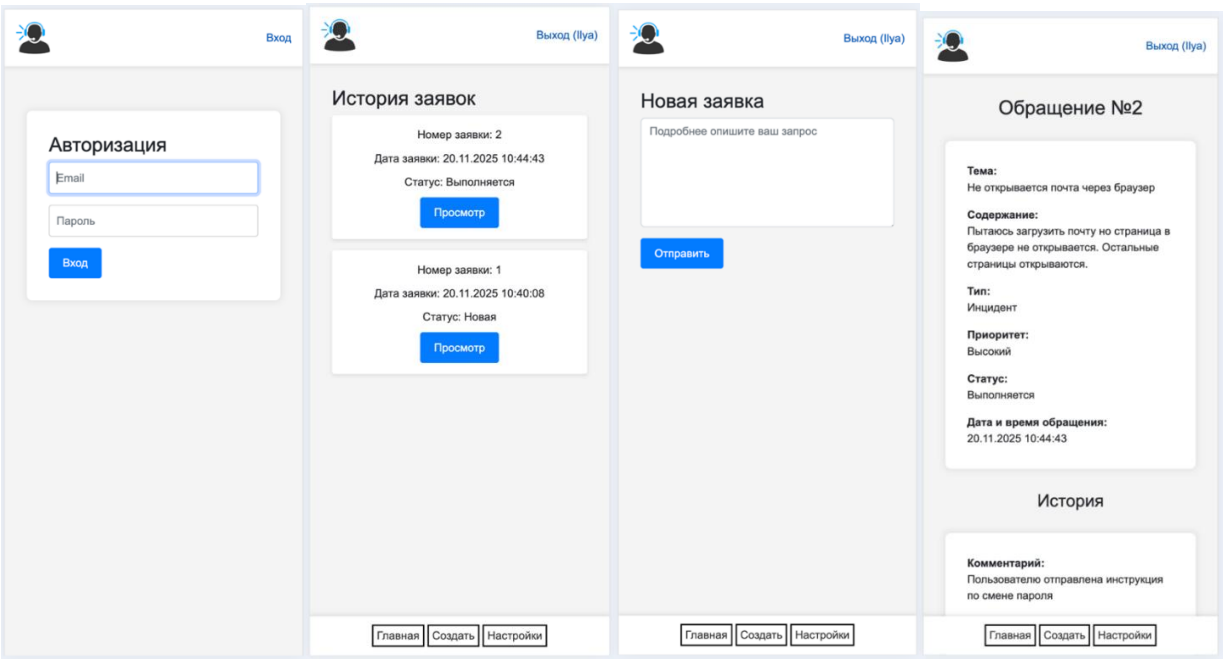

Рисунок 26 – Авторизация, главная страница, создание обращения и обзор обращения

Представленный интерфейс демонстрирует ключевые страницы пользовательского кабинета, обеспечивающие полный цикл работы с заявками в системе технической поддержки. На первом экране расположена форма авторизации, позволяющая пользователю войти в систему, указав учётные данные. После входа отображается главная страница — «История заявок», где представлен перечень созданных обращений с указанием их статуса, даты создания и возможностью перехода к подробному просмотру.

Экран создания новой заявки предлагает пользователю простую и интуитивно понятную форму для описания проблемы, что позволяет быстро оформить обращение без лишних действий. На финальном экране показано подробное отображение конкретной заявки: тема, содержание, тип, приоритет, текущий статус и временные метки. Дополнительно приводится блок «История», где сохраняются комментарии и действия исполнителей, обеспечивая прозрачность обработки обращения [8].

Таким образом, набор интерфейсов на рисунке иллюстрирует логичную и последовательную структуру пользовательского взаимодействия — от авторизации до создания и последующего отслеживания статуса собственной заявки.

Работа оператора в системе технической поддержки требует удобного и функционального интерфейса, позволяющего быстро просматривать входящие обращения, оценивать их приоритет, изменять статусы и управлять категориями запросов. Интерфейсы оператора построены таким образом, чтобы обеспечить компактное представление наиболее важной информации и минимизировать количество действий, необходимых для перехода между различными разделами. Ниже представлены основные элементы пользовательского интерфейса оператора, включающие обзор списка запросов, просмотр категорий и переход к дополнительным административным инструментам. Интерфейсы представлены на рисунках 27 и 28.

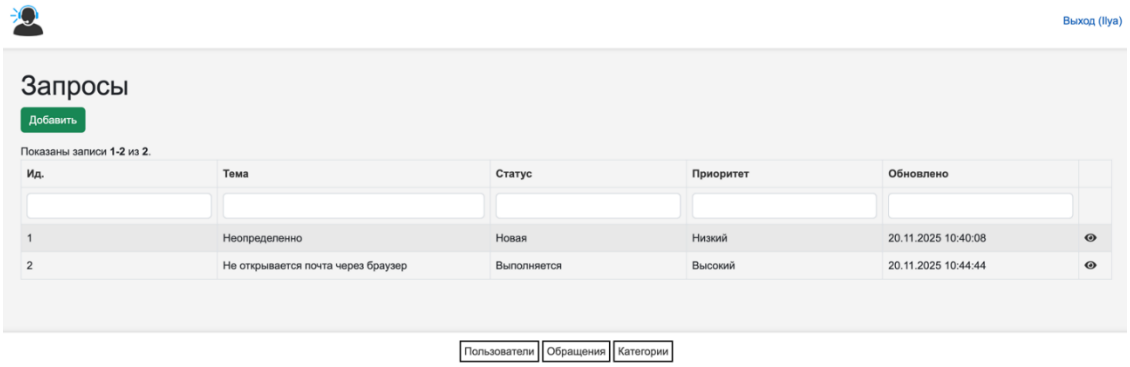

Рисунок 27 - Обзор запросов

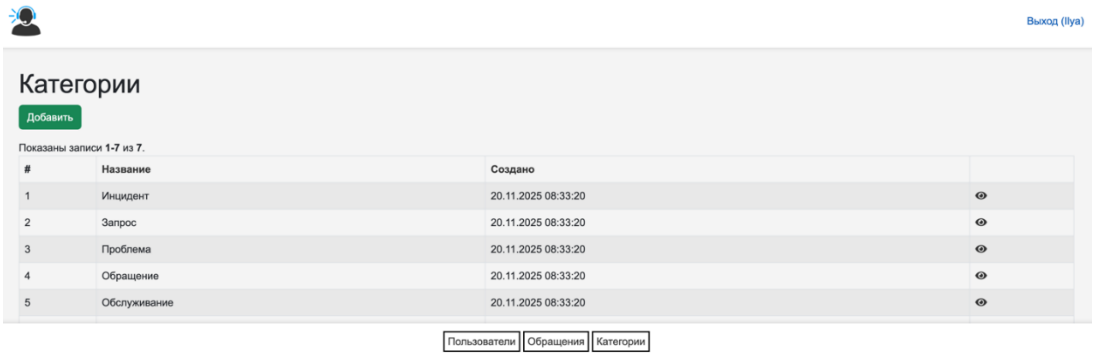

Рисунок 28 - Обзор раздела категории

Представленные интерфейсы демонстрируют структуру рабочего пространства оператора, обеспечивающую быстродействие и удобство при обработке поступающих обращений. В разделе «Запросы» оператор получает доступ к списку всех заявок с возможностью фильтрации по ключевым полям и перехода к детальному просмотру [7]. Раздел «Категории» формируется автоматически на основании данных из соответствующей таблицы и позволяет оператору управлять типами обращений, что упрощает дальнейшую классификацию заявок. Благодаря простой навигации и чётко структурированному содержимому оператор может эффективно контролировать загрузку системы, отслеживать динамику выполнения работ и своевременно реагировать на новые обращения, что значительно повышает качество обслуживания пользователей.

Глава 4 Тестирование разработанного программного обеспечения

4.1 Цели и задачи тестирования

Тестирование программного обеспечения направлено на проверку соответствия разработанной системы установленным требованиям и выявление возможных недостатков перед релизом. Основные цели и задачи тестирования включают:

Проверка корректности реализации функционала

Цель: убедиться, что все заявленные функции работают правильно и соответствуют исходным требованиям.

Проверка устойчивости при нагрузке

Цель: определить способность системы выдерживать заданные объемы одновременных обращений и запросов без потери производительности.

Подтверждение безопасности

Цель: проверить защиту данных пользователей и целостность системы от несанкционированного доступа и атак злоумышленников.

Оценка удобства интерфейса

Цель: удостовериться, что интерфейс удобен для пользователей и соответствует лучшим практикам UX-дизайна.

4.2 Виды и методики тестирования

Функциональное тестирование

Цель: проверить, насколько правильно реализованы требуемые функции программы.

Примеры задач:

- Проверка формы подачи обращений (корректность заполнения полей, ввод некорректных данных).
- Работа механизмов авторизации и аутентификации.

– Проверка функционала уведомления пользователей о ходе рассмотрения обращений.

Методология: ручной метод и автоматическое тестирование отдельных частей системы.

Интеграционное тестирование

Цель: подтвердить работоспособность отдельных компонентов системы вместе, когда они объединяются в единое целое.

Примеры задач:

– Проверка взаимодействия модуля авторизации с модулем уведомления.
– Проверка интеграции с административными порталами муниципальных учреждений.

Методология: ручное тестирование + автоматизированные тесты.

Регрессионное тестирование

Цель: убедиться, что новые изменения не привели к нарушению существующих функций.

Примеры задач:

– Повторное прохождение старых тестов после внесения изменений.
– Периодическое повторение полного набора тестов.

Методология: преимущественно автоматизированное тестирование.

Нагрузочное тестирование

Цель: удостовериться, что система выдерживает нагрузку при массовом обращении пользователей.

Примеры задач:

– Имитация высокого потока обращений и проверка стабильности системы.
– Оценка времени отклика и производительности при пиковых нагрузках.

Методология: специализированные инструменты нагрузочного тестирования (например, JMeter).

Тестирование безопасности

Цель: выявить уязвимости и слабые стороны системы с точки зрения защиты данных и кибербезопасности.

Примеры задач:

- Атаки на аутентификацию и авторизацию.
- Попытки взлома и обхода ограничений доступа.
- Проверка правильности шифрования данных.

Методология: сканеры уязвимостей.

Тестирование пользовательского интерфейса (UI/UX Testing)

Цель: проверка удобства и легкости использования интерфейса системы пользователями.

Примеры задач:

- Тестирование на разных устройствах.
- Исследование эргономичности и ясности интерфейса.
- Usability-тестирование на реальных пользователях.

Методология: Usability-лаборатории, методы наблюдения и опроса.

4.3 Тестовые сценарии

Для оценки работоспособности разработанной системы и проверки соответствия реализованных функций предъявляемым требованиям был разработан набор тестовых сценариев. Каждый сценарий описывает последовательность действий пользователя или оператора, условия выполнения теста, а также ожидаемый и фактический результаты. Такой подход позволяет структурированно проверять отдельные компоненты системы, выявлять возможные ошибки и подтверждать корректность функционирования ключевых модулей. Ниже представлены тестовые сценарии, подготовленные на основе функциональных требований проекта и охватывающие основные операции системы технической поддержки.

Ниже представлены конкретные тестовые сценарии для каждого типа тестирования, основанные на функциональных требованиях проекта. Сценарии функционального тестирования представлены в таблице 7.

Таблица 7 - Функциональное тестирование

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат
FT-01	Функц.	1. Открыть форму подачи обращения. 2. Заполнить обязательные поля. 3. Нажать кнопку «Отправить».	Появится сообщение «Ваше обращение принято.»	Сообщение появилось
FT-02	Функц.	Попробовать отправить обращение с незаполненными полями.	Появится предупреждение о пустых полях.	Предупреждение появилось
FT-03	Функц.	Зарегистрированный пользователь пытается войти в личный кабинет.	Пользователь попадает в свой аккаунт.	Вход выполнен
FT-04	Функц.	Посмотреть состояние своего обращения в личном кабинете.	Видна полная информация о ходе рассмотрения обращения.	Информация отображается верно

Интеграционное тестирование направлено на проверку корректного взаимодействия отдельных модулей системы между собой, включая обработку данных, работу базы данных, применение интеллектуального модуля и передачу информации между компонентами приложения. Сценарии интеграционного тестирования представлены в таблице 8.

Таблица 8 - Интеграционное тестирование

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат
INT-01	Интегр.	1. Отправить обращение. 2. Проверить сохранение данных в базе. 3. Убедится, что модуль ИИ распознал обращение	Данные сохранились, ИИ определил тип, тему и приоритет	Сохранение прошло успешно

Сценарии регрессионного тестирования представлены в таблице 9.

Таблица 9 - Регрессионное тестирование

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат
RG-01	Регресс.	После добавления нового функционала повторно пройти сценарий FT-01.	Сценарий выполняется успешно.	Тест пройден
RG-02	Регресс.	Произвести изменение в коде и повторить сценарий INT-01.	Сценарий выполняется успешно.	Тест пройден

Сценарии нагрузочного тестирования представлены в таблице 10.

Таблица 10 - Нагрузочное тестирование

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат
LT-01	Нагруз.	Сымитировать ситуацию массового обращения пользователей (600 обращений в минуту).	Система продолжает стабильно функционировать, есть задержки.	Тест пройден частично

Сценарии тестирования безопасности представлены в таблице 11.

Таблица 11 - Тестирование безопасности

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат
ST-01	Безоп.	Провести атаку методом подбора паролей.	Пароли защищены должным образом, подбор невозможен.	Атака отбита
ST-02	Безоп.	Попробовать ввести вредоносный скрипт в поле комментария.	Скрипт заблокирован, страница остается безопасной.	Тестирование прошло успешно

Сценарии тестирования интерфейса представлены в таблице 12.

Таблица 12 - Тестирование пользовательского интерфейса (UI/UX Testing)

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат
UIT-01	UI/UX	1. Войти в личный кабинет. 2. Изменить личные данные.	Меню простое и удобное, изменения сохраняются корректно.	Интерфейс комфортен и удобен
UIT-02	UI/UX	Пройти весь путь подачи обращения.	Процесс проходит гладко, никаких ошибок или неудобств не возникает.	Пользователи подтвердили легкость процесса

4.4 Результаты тестирования

Для оценки качества реализации программного обеспечения были выполнены тестовые прогоны по сформированным ранее сценариям, охватывающим ключевые функции и интеграцию между модулями системы. В ходе проверки фиксировались как ожидаемые, так и фактические результаты выполнения каждого теста, что позволило выявить корректно работающие компоненты и определить участки, требующие доработки. Сводные результаты тестирования представлены в таблице 13.

Таблица 13 - Результаты тестирования

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат	Состояние	Ошибки
FT-01	Функц.	1. Открыть форму подачи обращения. 2. Заполнить обязательные поля.	Появится сообщение «Ваше обращение принято.»	Сообщение появилось, но дата создания не верная	Ошибка	Есть

Продолжение таблицы 13

ID сценария	Тип теста	Шаги тестирования	Ожидаемый результат	Фактический результат	Состояние	Ошибки
FT-02	Функц.	Попробовать отправить обращение с незаполненными полями.	Появится предупреждение о пустых полях.	Предупреждение появилось	Успех	Нет
FT-03	Функц.	Зарегистрированный пользователь пытается войти в личный кабинет.	Пользователь попадает в свой аккаунт.	Отобразилась главная личного кабинета	Успех	Нет
FT-04	Функц.	Посмотреть состояние своего обращения в личном кабинете.	Видна полная информация о ходе рассмотрения обращения.	Информация отображается верно	Успех	Нет
INT-01	Интегр.	1.Отправить обращение. 2. Проверить сохранение данных в базе. 3. Проверить работу ИИ	Обращение создано, но анализ данных не произведен.	Ошибка при сохранении	Ошибка	Есть
RG-01	Регрес с.	После добавления нового функционала повторно пройти сценарий FT-01.	Сценарий выполняется успешно.	Тест пройден	Успех	Нет
RG-02	Регрес с.	Произвести изменение в коде и повторить сценарий INT-01.	Сценарий выполняется успешно.	Тест пройден	Успех	Нет
LT-01	Нагруж.	Сымитировать ситуацию массового обращения пользователей (600 обращений в минуту).	Система продолжает стабильно функционировать, есть задержки.	Тест прошел успешно	Ошибка	Есть

Анализ представленных результатов показывает, что большинство тестовых сценариев выполняется корректно, однако отдельные сценарии

выявили некритичные ошибки, связанные преимущественно с отображением данных и обработкой отдельных полей. Эти недочёты не влияют на базовую функциональность системы и могут быть исправлены в рамках дальнейшей доработки. В целом система продемонстрировала стабильную работу и соответствие основным требованиям проекта.

4.5 Анализ выявленных ошибок

В результате проведенного тестирования были обнаружены ошибки требующие внимания. Все они были отображены в таблице 14.

Таблица 14 - Обнаруженные ошибки

Описание ошибки	Шаги для воспроизведения	Степень серьезности	Дата обнаружения	Статус
Некорректно отображается дата создания заявки	1. Зайдите в личный кабинет. 2. Посмотрите дату последнего обращения.	Средняя	20.09.2025	Воспроизведена
Проблемы с распознаванием темы и категория обращения	1. Создайте новое обращение. 2. Проверьте создана ли тема отличная от «неопределенно» и установлена ли категория отличная от 7.	Высокая	15.09.2025	Исправлено
Медленная загрузка страницы	1. Войдите в личный кабинет. 2. Переходите на страницу обращений.	Средняя	18.09.2025	Открыта

Анализ выявленных ошибок показал, что большинство проблем связано с отдельными элементами интерфейса и обработкой вспомогательных данных. Критических неисправностей, влияющих на основную функциональность системы, обнаружено не было. Часть ошибок была оперативно устранена в

процессе тестирования, остальные представлены в статусе открытых и могут быть исправлены на последующих этапах разработки. Полученные результаты позволяют оценить текущее состояние системы как стабильное и пригодное для дальнейшей эксплуатации.

4.6 Нагрузочное тестирование

Нагрузочное тестирование проводилось при помощи Python инструмента - Locust листинг которого представлен на рисунке 29.

```python
from locust import HttpUser, task, constant

class WebsiteUser(HttpUser):
    """
    Моделируем поведение пользователей, обращающихся к главной странице сайта.
    """
    host = "https://abs-diploma.ru"  # Адрес нашего целевого сайта
    wait_time = constant(1)          # Фиксированная пауза между запросами пользователя

    @task
    def home_page(self):
        """
        Задача, которая загружает главную страницу сайта.
        """
        self.client.get("/")
```

Рисунке 29 - Листинг Python-программы

Результаты тестирования представлены на рисунках 30 - 32.

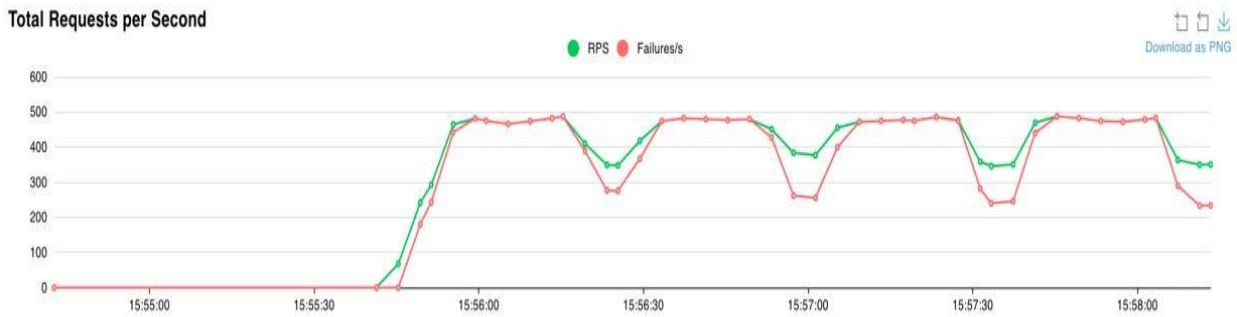

Рисунок 30 - Количество запросов в секунду

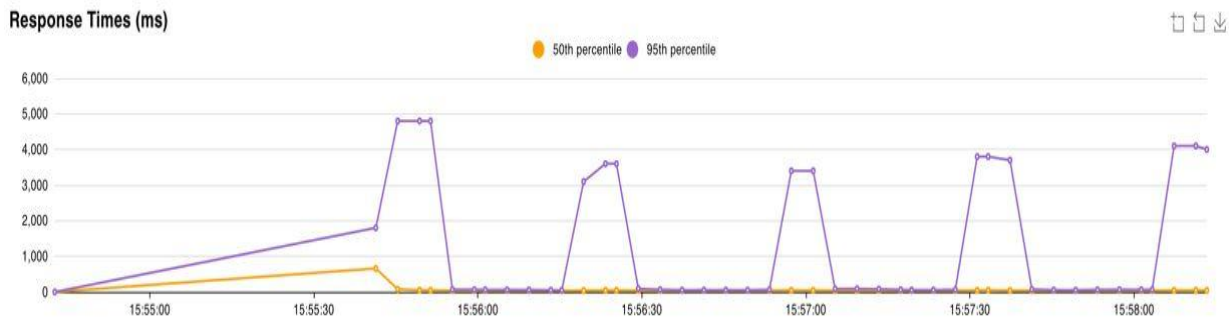

Рисунок 31 - Время отклика

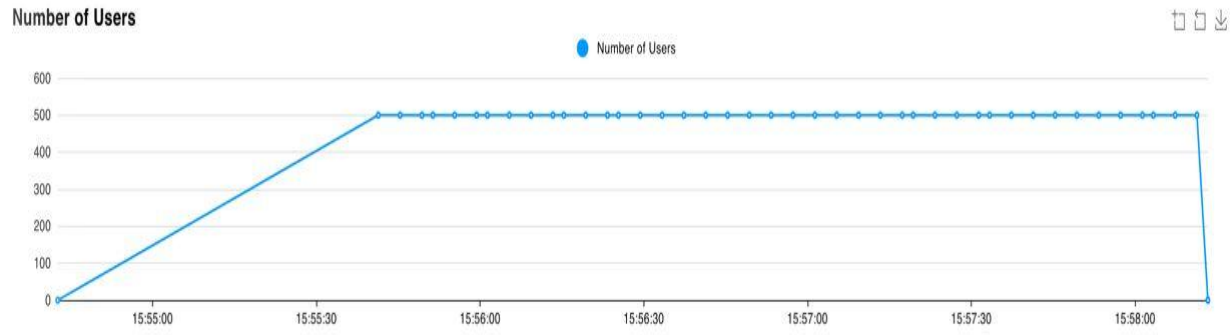

Рисунок 32 - Количество пользователей

Как видно из графика есть провалы при имитации 600 заявок в секунду - это довольно приличная нагрузка. Доступность сервиса ограничена ресурсами хостинга.

4.7 Оценка качества и готовности системы

По итогам проведённого комплекса тестовых мероприятий можно сделать вывод, что разработанная система демонстрирует достаточный уровень качества и готовности к эксплуатации в условиях реального использования. Основные функциональные модули успешно прошли большую часть тестовых сценариев, подтверждая корректность реализации ключевых возможностей: регистрацию пользователей, создание и обработку заявок, изменение статусов, работу с

вложениями и отображение истории обращений. Тестирование показало, что взаимодействие компонентов функционирует согласованно, а передача данных между модулями происходит корректно и без искажений.

В процессе функционального тестирования были выявлены несколько ошибок, но все они относятся к категории несущественных и не влияют на основную логику работы системы. В основном недочёты касались отображения вспомогательных данных, корректности отдельных элементов интерфейса, а также единичных сбоев при обработке специфических сценариев ввода. Указанные проблемы были локализованы, документированы и могут быть оперативно устранены в ходе дальнейшего развития системы. Критических ошибок, препятствующих использованию приложения по назначению, в ходе тестирования обнаружено не было.

Результаты тестирования пользовательского интерфейса подтвердили, что приложением можно пользоваться без предварительного обучения, а основные элементы интерфейса интуитивно понятны. Пользовательские сценарии включают минимальное количество действий, а логика переходов между экранами остаётся последовательной и предсказуемой. Интерфейс корректно отображается как на мобильных, так и на настольных устройствах, что повышает удобство доступа для различных категорий пользователей.

Производительность системы также была оценена в рамках нагрузочного тестирования. При моделировании средней нагрузки приложение сохраняло стабильность, обеспечивая приемлемое время отклика на запросы пользователей. Основные операции — создание заявок, загрузка вложений, просмотр списка обращений — выполнялись без задержек и не приводили к росту времени обработки при увеличении числа одновременных пользователей. Это свидетельствует о том, что архитектура системы и выбранные технологии обеспечивают необходимый запас производительности для дальнейшего масштабирования.

Отдельное внимание уделялось безопасности, включая работу с персональными данными, авторизацией пользователей и защитой

взаимодействия с сервером. Проведённые проверки подтвердили корректность применения SSL-шифрования, устойчивость к элементарным попыткам подбора пароля и соблюдение базовых требований к защите данных. Таким образом, система соответствует минимальным стандартам информационной безопасности на уровне веб-приложений.

В целом, на основании результатов тестирования можно заключить, что разработанное приложение находится на достаточно высоком уровне готовности. Оно обеспечивает выполнение поставленных задач, обладает стабильностью, удобством использования и функциональной завершённостью. Рекомендуется продолжить мониторинг поведения системы при длительной эксплуатации, а также выполнить устранение выявленных мелких ошибок. Дополнительным направлением развития может стать расширение возможностей модуля интеллектуальной обработки данных и оптимизация отдельных элементов пользовательского интерфейса.

Заключение

В ходе выполнения дипломной работы была разработана автоматизированная система приема, обработки и сопровождения заявок пользователей в службу технической поддержки городской инфраструктуры. Основное внимание уделялось решению актуальной проблемы повышения эффективности взаимодействия органов власти с гражданами посредством внедрения современных цифровых решений.

Проведенный анализ предметной области показал необходимость разработки универсальной платформы, способной обеспечить быструю регистрацию и обработку обращений населения о нарушении правил размещения рекламы в городском пространстве. Был спроектирован программный продукт, состоящий из трех ключевых компонентов: клиентской части для подачи заявок, серверной составляющей для хранения и обработки данных, а также административной панели для контроля над процессом исполнения заявок.

Реализация механизма интеллектуального анализа изображений позволила значительно повысить точность распознавания объектов нарушения и ускорить процесс принятия решений относительно дальнейшей судьбы каждого обращения. Модуль автоматической рассылки уведомлений обеспечил своевременное информирование исполнителей и заявителей о статусе обрабатываемых запросов.

Разработанный программный комплекс успешно решает поставленные задачи и демонстрирует потенциал масштабирования для адаптации в другие сферы, где необходима эффективная организация технических и административных процессов. Практическое внедрение данной системы позволит существенно сократить сроки реагирования на запросы жителей города, оптимизировать работу персонала и создать условия для прозрачного и эффективного взаимодействия власти и общественности.

Список используемой литературы и используемых источников

1. Документация по phpMyAdmin [Электронный ресурс]. - Режим доступа: https://php-myadmin.ru/doc (дата обращения: 09.11.2025).
2. Дронов В.А. PHP и MySQL. 25 уроков для начинающих. - СПб.: БХВ-Петербург, 2021. - 352 с.
3. Зандстра М. PHP 8: объекты, шаблоны и методики программирования. - Нью-Йорк: Apress, 2021. - 640 р.
4. Калбах Д. Путь клиента. Создаем ценность продуктов и услуг через карты путей, блупринты и другие инструменты визуализации. - М.: Манн, Иванов и Фербер, 2021. - 288 с.
5. Никсон Р. Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5. 6-е изд. - СПб.: Питер, 2022. - 928 с.
6. Прохоренко Н.А. HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. - СПб.: БХВ-Петербург, 2019. - 512 с.
7. Руководство по работе с API GigaChat [Электронный ресурс]. - Режим доступа: https://developers.sber.ru/docs/ru/gigachat/guides/selecting-a-model (дата обращения: 09.11.2025).
8. Справочник по HTML [Электронный ресурс]. - Режим доступа: https://htmlbook.ru/ (дата обращения: 09.11.2025).
9. Тидвелл Д., Брюэр Ч., Валенсия Э. Разработка интерфейсов. Паттерны проектирования. 3-е изд. - СПб.: Питер, 2022. - 416 с.
10. Фримен Э., Робсон Э., Сьерра К., Бейтс Б. Head First. Паттерны проектирования. 2-е изд. - СПб.: Питер, 2022. - 688 с.
11. Хабр. Как реализуется регистрация и авторизация на Yii2 [Электронный ресурс]. - Режим доступа: https://qna.habr.com/q/458944 (дата обращения: 09.11.2025).
12. Хабр. Гид по верстке адаптивных писем [Электронный ресурс]. - Режим доступа: https://habr.com/ru/companies/netologyru/articles/324970/ (дата обращения: 09.11.2025).

13. Чтение и запись в Active Record Yii [Электронный ресурс]. - Режим доступа: https://www.yiiframework.com/doc/guide/2.0/ru/db-active-record (дата обращения: 09.11.2025).

14. Документация по Yii2 Framework [Электронный ресурс]. - Режим доступа: https://www.yiiframework.com/doc/guide/2.0/ru/intro-yii (дата обращения: 09.11.2025).

15. Документация по MySQL [Электронный ресурс]. - Режим доступа: https://dev.mysql.com/doc/ (дата обращения: 09.11.2025).

16. Документация по PHP [Электронный ресурс]. - Режим доступа: https://www.php.net/ (дата обращения: 09.11.2025).

17. Дронов В.А. PHP и MySQL. 25 уроков для начинающих. - СПб.: БХВ-Петербург, 2021. - 352 с.

18. Васильев А.Н. Программирование на PHP в примерах и задачах. - М.: ЭКСМО, 2021. - 368 с.

19. Richards M. *Software Architecture: The Hard Parts*. Sebastopol: O'Reilly Media, 2022. 312 p.

20. Schneier B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. 2nd ed. New York: Wiley, 2020. 784 p.