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

Кафедра «Прикладная математика и информатика»
(наименование)

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

Разработка программного обеспечения
(направленность (профиль) / специализация)

# ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка программного обеспечения информационной системы управления заявками аварийно–диспетчерской службы ЖКХ

Обучающийся Р.Е. Игнатьев
(Инициалы Фамилия) (личная подпись)

Руководитель д.т.н., доцент, С.В. Мкртычев
(ученая степень (при наличии), ученое звание, Инициалы Фамилия)

Тольятти 2024

## Аннотация

Тема бакалаврской работы – «Разработка программного обеспечения информационной системы управления заявками аварийно–диспетчерской службы ЖКХ».

Работа включает введение, три главы, заключение, список используемой литературы и используемых источников.

В введении обсуждаются актуальность исследования, цели и задачи работы, а также практическая значимость разработки системы для улучшения управления заявками в сфере жилищно–коммунального хозяйства (ЖКХ) и повышения эффективности функционирования аварийно–диспетчерской службы.

Первая глава ориентирована на определение задач разработки программного обеспечения и обзор существующих решений в области ЖКХ.

Вторая глава включает проектирование системы, выбор методологии, создание модели данных и логическое проектирование.

Третья глава охватывает процесс разработки программного обеспечения (ПО), а также выбор средств разработки, создание базы данных, проектирование пользовательских интерфейсов, настройку маршрутизации веб–страниц, а также этапы тестирования и усовершенствования системы.

Заключение содержит выводы о результатах работы и рекомендации по ее совершенствованию.

Ключевые слова: ЖКХ, информационная система, управление заявками, аварийно–диспетчерская служба, программное обеспечение, проектирование, тестирование.

Работа на 51 странице, включает 19 рисунков, 4 таблиц.

Список используемой литературы и используемых источников содержит 25 источников.

**Оглавление**

# Введение

В условиях современных требований к качеству обслуживания и оперативности работы служб жилищно–коммунального хозяйства (ЖКХ), эффективное управление заявками от жителей становится неотъемлемой частью функционирования этих систем. Важнейшей задачей является автоматизация процессов, которые позволят повысить скорость и точность обработки заявок, улучшить взаимодействие между диспетчерами, исполнителями и заявителями, а также обеспечить прозрачность и контроль на всех этапах выполнения работ.

Актуальность темы исследования обусловлена необходимостью внедрения информационных систем для управления заявками в сфере ЖКХ, что позволит значительно повысить эффективность работы аварийно–диспетчерских служб. Автоматизация обработки заявок даст возможность оперативно реагировать на аварийные ситуации, минимизировать временные затраты на выполнение задач, улучшить качество обслуживания и повысить удовлетворенность населения.

Объектом исследования является процесс управления заявками в аварийно–диспетчерской службе ЖКХ.

Предметом исследование – программное обеспечение информационной системы для обработки этих заявок.

Цель работы заключается в создании программного обеспечения (ПО) для информационной системы управления заявками (ИСУЗ) в аварийно–диспетчерской службе ЖКХ, которое обеспечит эффективное решение задач по регистрации, обработке и отслеживанию заявок на всех стадиях их выполнения.

Для достижения этой цели были сформулированы следующие задачи:

- исследование существующих решений в сфере управления заявками в ЖКХ и их функциональных особенностей;
- формирование требований к программному обеспечению с учетом

специфики работы аварийно–диспетчерских служб;

- проектирование архитектуры системы и моделей данных;
- реализация функционала системы для различных пользователей;
- проведение тестирования системы и подготовка к внедрению.

Практическая значимость работы заключается в разработке программного обеспечения, которое поможет существенно повысить эффективность работы аварийно–диспетчерской службы, улучшить качество обслуживания и ускорить процесс решения аварийных ситуаций.

В первой главе работы рассматриваются функциональные и архитектурные особенности систем управления заявками в сфере ЖКХ, проводится анализ существующих решений и формулируются требования к проектируемому программному обеспечению.

Во второй главе рассматриваются подходы к проектированию системы, включая выбор методологии и разработку модели данных.

Третья глава охватывает этапы реализации системы, а также тестирование и оптимизацию программного обеспечения.

Заключение содержит итоги выполнения выпускной квалификационной работы.

# Глава 1 Постановка задачи на разработку программного обеспечения информационной системы управления заявками аварийно–диспетчерской службы ЖКХ

## 1.1 Функциональные и архитектурные особенности информационных систем управления заявками в сфере ЖКХ

Программное обеспечение (ПО) информационной системы управления заявками (ИСУЗ) аварийно–диспетчерской службы ЖКХ должно обеспечивать высокую оперативность, удобство и прозрачность управления заявками жителей.

Функциональность системы охватывает весь цикл обработки заявки – от ее создания до завершения задачи. ИСУЗ должна обеспечить прием и регистрацию заявок, их назначение и отслеживание, а также формирование отчетов по выполненным задачам.

Система должна иметь интерфейсы для разных групп пользователей: диспетчеров, исполнителей и заявителей, каждый из которых будет иметь доступ к функционалу, соответствующему его роли. Для диспетчеров требуется возможность назначения и переназначения задач, для исполнителей – контроль статуса заявок и обратная связь с диспетчерами, а для жителей – информирование о статусе их заявок. Важным требованием является поддержка мобильных устройств для работы исполнителей на месте [4].

Типовая архитектура ИСУЗ включает следующие компоненты:

- пользовательский интерфейс;
- базу данных для хранения заявок, пользователей и историй выполнения задач;
- функционал управления заявками, в том числе их создание, изменение и назначение исполнителей;
- модуль интеграции с внешними системами (ГИС ЖКХ, ЕГРЮЛ) для взаимодействия с государственными органами;

– систему отчетности и аналитики для мониторинга выполненных задач и генерации отчетов.

Ключевыми аспектами являются масштабируемость системы, обеспечение безопасности данных, управление доступом и защита конфиденциальной информации о жителях и выполненных задачах [16].

## 1.2 Разработка требований к программному обеспечению информационной системы управления заявками ЖКХ

Формулировка требований к ПО ИСУЗ ЖКХ базируется на изучении основных процессов управления заявками и применении методологии FURPS+ (Functionality, Usability, Reliability, Performance, Supportability) [5]. Этот подход позволяет систематизировать требования, выделив функциональные и нефункциональные аспекты системы, которые необходимо учесть при проектировании и разработке.

Требования к программному обеспечению информационной системы управления заявками, сформулированные в соответствии с технологией FURPS+, приведены в таблице 1.

Таблица 1 – Требования к ПО ИСУЗ ЖКХ

| Требования | Описание |
|---|---|
| Функциональные Functionality | Прием, обработка и распределение заявок, интеграция с внешними системами. |
| Удобство использования Usability | Интуитивно понятный интерфейс для различных категорий пользователей (диспетчеры, исполнители, заявители). |
| Надежность Reliability | Гарантированная сохранность и целостность данных заявок. |
| Производительность Performance | Обеспечение оперативного отклика на запросы от пользователей и обработка заявок. |

Продолжение таблицы 1

| Требования | Описание |
|---|---|
| Поддержка Supportability | Возможность регулярных обновлений и дальнейшего сопровождения системы. |
| Дополнительные (+) | Интеграция с мобильными устройствами для удобства работы исполнителей на местах. |

Проектирование системы предусматривает использование микросервисной архитектуры, что позволит модульно развивать систему и масштабировать ее под растущие потребности. В качестве базы данных предлагается использовать реляционную СУБД для хранения данных о пользователях и заявках.

## 1.3 Обзор и анализ аналогов программного обеспечения управления заявками в сфере ЖКХ

Изучение и анализ доступных аналогичных программных решений для управления заявками в ЖКХ дает возможность выделить основные функции и оценить их сильные и слабые стороны.

Система Filta является примером ПО, которое направлено на автоматизацию бизнес–процессов и упрощение управления заявками в ЖКХ.

Основные функции системы включают:

- эффективно регистрировать и обрабатывать заявки от жителей, следить за их статусом и назначать ответственных исполнителей;
- автоматически распределять задачи между сотрудниками в зависимости от типа заявки и требуемых навыков;
- формирование отчетов о выполненных работах, что облегчает контроль за качеством обслуживания;
- доступно приложение, которое позволяет сотрудникам на местах

получать информацию о задачах, обновлять статус заявок и фиксировать выполненные работы.

Преимущества: Filta отличается удобным интерфейсом и высоким уровнем автоматизации бизнес–процессов, что упрощает выполнение задач на местах и повышает оперативность работы.

Недостатки: ограниченная возможность интеграции с государственными информационными системами и узкая специализация на мобильных решениях, что может ограничить гибкость использования системы в ЖКХ.

Окно интерфейса ИСУ Filta представлено на рисунке 1.

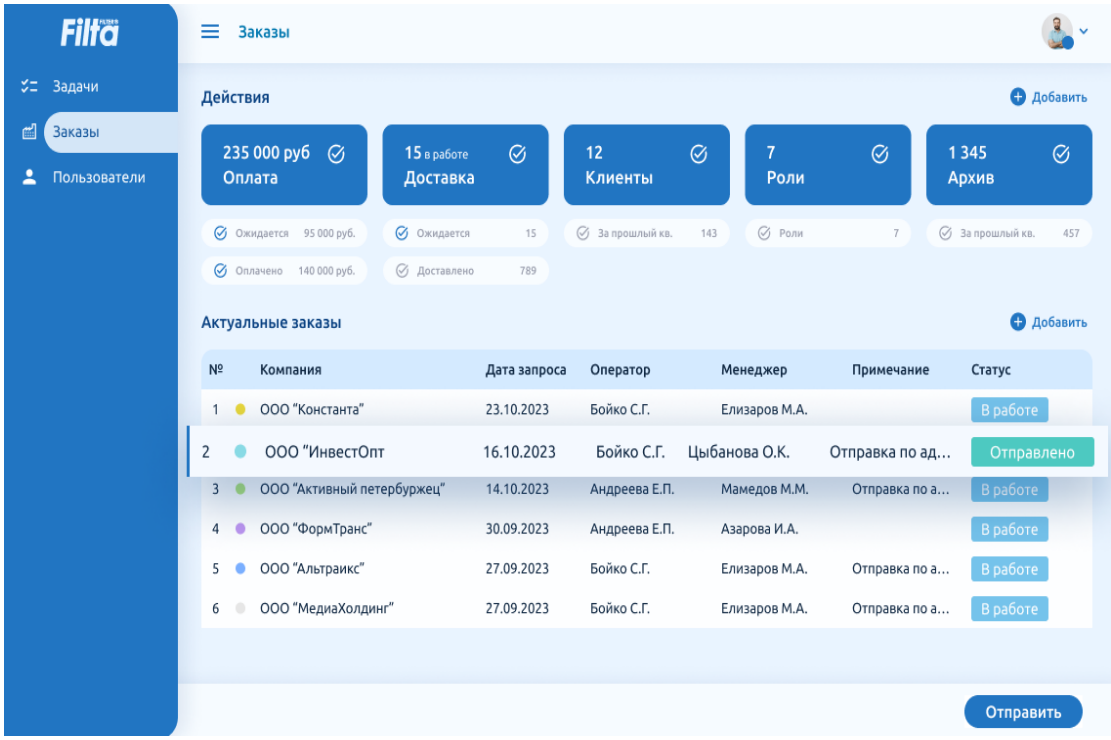



Рисунок 1 – Интерфейс ИСУ Filta

Система ITSM 365 представляет собой комплексное решение, которое охватывает управление заявками и инцидентами, а также поддерживает интеграцию с другими системами и модулями [17].

Основные функции:

– позволяет регистрировать и отслеживать заявки от клиентов, а также назначать задачи и контролировать их выполнение;
– поддерживает интеграцию с бухгалтерскими и финансовыми системами, что облегчает документооборот и финансовый учет;
– включает инструменты для создания отчетности и анализа данных, возможность оценить качество обслуживания эффективности процессов.

Преимущества: обширный функционал и возможность интеграции с различными модулями и системами, что делает систему гибким инструментом для использования в сфере ЖКХ.

Недостатки: высокая стоимость внедрения и обслуживания, а также сложность настройки под специфические требования ЖКХ. Окно интерфейса ИСУ ITSM 365 представлено на рисунке 2.

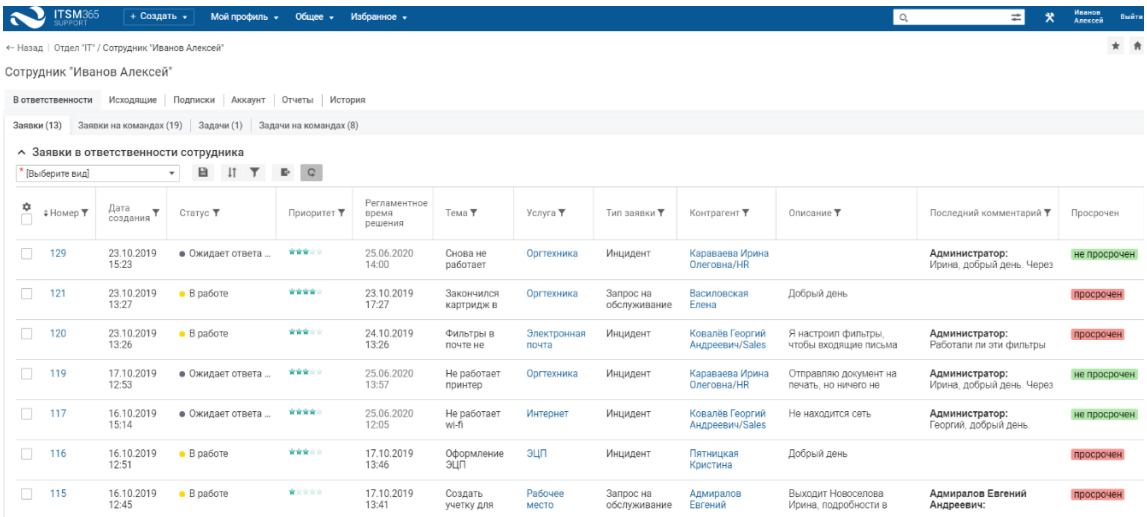



Рисунок 2 – Интерфейс ИСУ ITSM 365

Система 1С: Управление ЖКХ специально разработана для нужд жилищно–коммунального хозяйства и включает в себя широкий функционал для автоматизации различных процессов [12]. Основные возможности:

– позволяет регистрировать и отслеживать заявки, а также вести учет инцидентов;

– включает функции для расчета квартплаты, учета начислений и долгов, а также работы с субсидиями;
– обеспечивается взаимодействие с государственными системами, что облегчает подачу отчетности и обмен данными;
– мобильное приложение позволяет жильцам подавать заявки и отслеживать их выполнение, а сотрудникам получать задания и обновлять их статус.

Преимущества: глубокая интеграция с государственными системами и обширный функционал для управления процессами в ЖКХ.

Недостатки: высокая стоимость внедрения и сложность обучения пользователей, в части настройки и работы с финансовыми модулями [1].

Окно интерфейса ИСУ Система 1С: Управление ЖКХ представлено на рисунке 3.

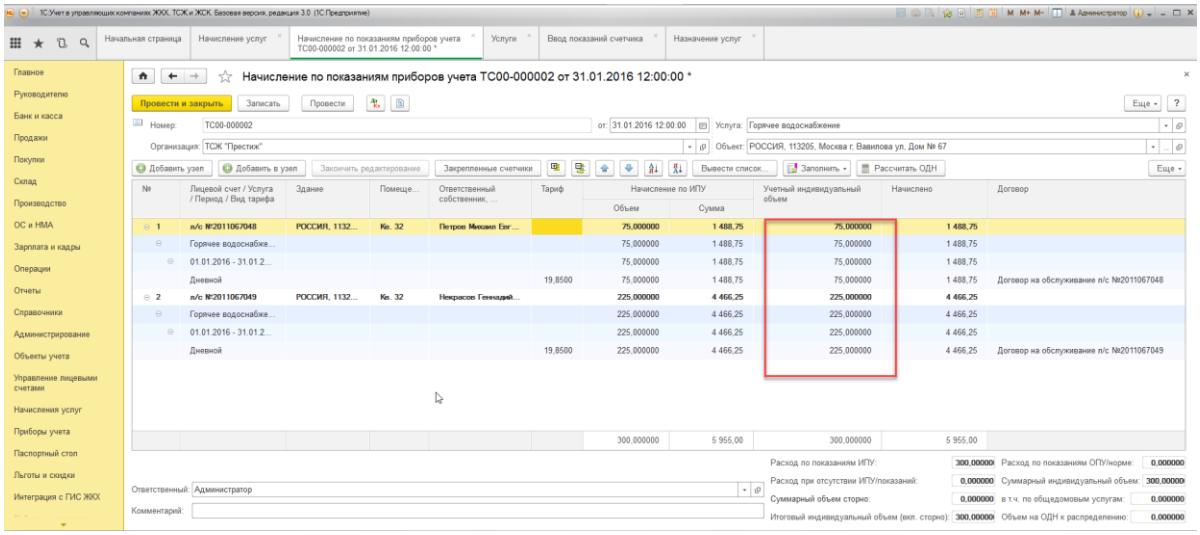



Рисунок 3 – Интерфейс ИСУ Система 1С: Управление ЖКХ

В таблице 2 приведен сравнительный анализ рассмотренных систем управления, в котором их характеристики оценены по шкале от 0 до 3. Оценка 0 означает полное несоответствие предъявляемым требованиям, в то время как оценка 3 указывает на полное соответствие этим требованиям.

Таблица 2 – Сравнительный анализа рассмотренных аналогов

| Функционал | Filta | ITSM 365 | 1С: Управление ЖКХ |
|---|---|---|---|
| Автоматизация заявок ЖКХ | 3 | 3 | 3 |
| Мобильные решения | 3 | 2 | 3 |
| Интеграция с гос. системами | 1 | 2 | 3 |
| Управление логистикой | 0 | 2 | 1 |
| Удобство интерфейса | 3 | 2 | 2 |
| Отчетность и аналитика | 2 | 3 | 3 |
| Интеграция с фин. и бухг. системами | 1 | 3 | 3 |
| Гибкость настроек | 2 | 3 | 2 |
| Стоимость внедрения | 2 | 1 | 2 |
| Итог (сумма баллов) | 17 | 21 | 22 |

По результатам анализа существующих аналогов было установлено, что для сферы ЖКХ необходимо решение с расширенными возможностями интеграции с государственными системами и мобильными устройствами, что обеспечит повышение оперативности работы и удобства для пользователей.

Выполненный обзор и анализ существующих на данный момент программных решений для управления заявками в сфере жилищно–коммунального хозяйства (ЖКХ) позволяет выделить ключевые функциональные особенности, преимущества и недостатки различных систем.

Что позволяет более точно определить потребности отрасли и сформировать требования к разработке или адаптации программного обеспечения, которые будут служить основой для разработки новых программных решений или адаптации уже существующих.

## 1.4 Проектные ограничения поддерживаемого ПО

Ограничения по функциональности:

– охват полного цикла обработки заявок – от их поступления до завершения задачи;
– предоставление отдельных интерфейсов для трех групп пользователей: диспетчеров, исполнителей и заявителей, каждая должна иметь доступ только к соответствующему функционалу;
– поддержка интеграции с государственными системами (ГИС ЖКХ, ЕГРЮЛ) для обеспечения взаимодействия с органами власти.

Ограничения по архитектуре:

– реализация с использованием микросервисной архитектуры, что позволит ее модульное развитие и масштабирование;
– хранение данных заявок и пользователей должна использоваться реляционная СУБД;
– поддержка работы на мобильных устройствах, что необходимо для оперативной работы исполнителей на местах.
– Ограничения по производительности:
– обеспечение оперативного создания и изменений заявок, а также быстрый отклик на запросы пользователей;
– спроектировать с возможностью увеличения числа пользователей и объемов данных без снижения производительности.

Ограничения по безопасности:

– обеспечивать высокий уровень защиты персональных данных жителей и информации о заявках, соблюдая требования законодательства о защите данных;
– поддерживать разделение прав доступа для различных ролей (диспетчер, исполнитель, заявитель), чтобы минимизировать риски несанкционированного доступа.

Ограничения по интеграции:

– возможность интеграции с внешними сервисами (государственные системы, бухгалтерские программы) для автоматизации документооборота и обмена данными;
– доступ на мобильных платформах для удобства работы исполнителей на местах.

Ограничения по поддержке и сопровождению:

– поддерживать возможность регулярных обновлений и сопровождения без нарушения текущей работы системы;
– обеспечивать высокую надежность, предотвращая потерю данных в случае сбоев и аварий.

Ограничения по удобству использования:

– интерфейсы системы должны быть интуитивно понятными для каждой группы пользователей. Необходимо минимизировать количество действий, необходимых для выполнения основных операций;
– интерфейсы мобильных устройств должны быть адаптированы под работу в полевых условиях с ограниченным доступом к сети и небольшими экранами.

Ограничения по стоимости и времени разработки: создание и внедрение системы должно быть осуществлено в пределах утвержденного бюджета

Все этапы разработки – проектирование, разработка, тестирование, внедрение должны быть завершены в установленные сроки для соответствия графику выполнения преддипломной практики.

Выводы по главе 1

Для разработки программного обеспечения для информационной системы управления заявками аварийно–диспетчерской службы ЖКХ были определены ключевые требования, которые должны быть учтены при проектировании системы. Прежде всего, это функциональность системы, включающая полный цикл обработки заявок, от их создания до завершения

задачи, а также интеграцию с внешними государственными системами для повышения эффективности работы. Требования по удобству использования системы должны обеспечить интуитивно понятный интерфейс для различных пользователей: диспетчеров, исполнителей и заявителей.

Анализ существующих аналогов программного обеспечения выявил, что для успешной реализации системы необходимо учесть важные аспекты, такие как поддержка мобильных устройств для исполнителей на местах и возможность интеграции с государственными и финансовыми системами. Сравнительный анализ показал, что наиболее подходящими для разработки в сфере ЖКХ являются решения с высокой гибкостью и возможностью масштабирования.

Проектирование информационной системы управления заявками должно учитывать современные архитектурные подходы, такие как микросервисная архитектура и использование реляционных СУБД для хранения данных [18]. Также необходимо обеспечить безопасность данных и защиту конфиденциальной информации, что требует реализации эффективных механизмов контроля доступа.

Таким образом, на основе проведенного анализа и оценки потребностей отрасли можно заключить, что создание специализированного программного обеспечения для управления заявками в ЖКХ позволит значительно повысить оперативность, прозрачность и эффективность работы аварийно–диспетчерской службы, что в свою очередь будет способствовать улучшению качества обслуживания населения.

# Глава 2 Проектирование программного обеспечения информационной системы управления заявками аварийно–диспетчерской службы ЖКХ

## 2.1 Разработка диаграммы вариантов использования программного обеспечения

При проектировании ИСУЗ аварийно–диспетчерской службы ЖКХ необходимо учитывать специфику ее работы и требуемый функционал. Аварийно–диспетчерская служба занимается обработкой заявок от жителей, управлением техническими бригадами и их выездом на устранение неисправностей. Основной целью системы является оперативное и прозрачное управление заявками, а также организация работы персонала с минимальными затратами времени и ресурсов.

Концепция разработки ПО ИСУЗ должна базироваться на следующих принципах:

- прозрачность и оперативность обработки заявок;
- удобство для сотрудников службы, принимающих заявки и контролирующих их выполнение;
- интеграция с существующими базами данных и системами мониторинга состояния объектов ЖКХ;
- обеспечение автоматизации процессов распределения заявок и контроля их исполнения.

Диаграмма вариантов использования (Use Case Diagram) представляет собой начальный этап в описании функционала системы и иллюстрации взаимодействия между пользователями и информационной системы управления заявками. Она позволяет создать структуру системы, выделить ключевые сценарии использования и установить связи между пользователями и системой [9]. Основные участники, взаимодействующие с системой:

- диспетчер принимает заявки от жителей, регистрирует их в системе,

контролирует процесс выполнения работ;

– технический специалист (бригада) выполняет работы по заявкам, фиксирует статус выполнения в системе;
– житель подает заявку на устранение неисправности (например, через телефон или интернет–портал), может следить за статусом своей заявки.

Диаграмма вариантов использования на рисунке 4 представлены основные процессы работы ИСУЗ, такие как прием и регистрация заявок, распределение задач между техническими специалистами, отслеживание статуса заявок, закрытие заявки после выполнения работы и формирование отчетов о выполненных заданиях.

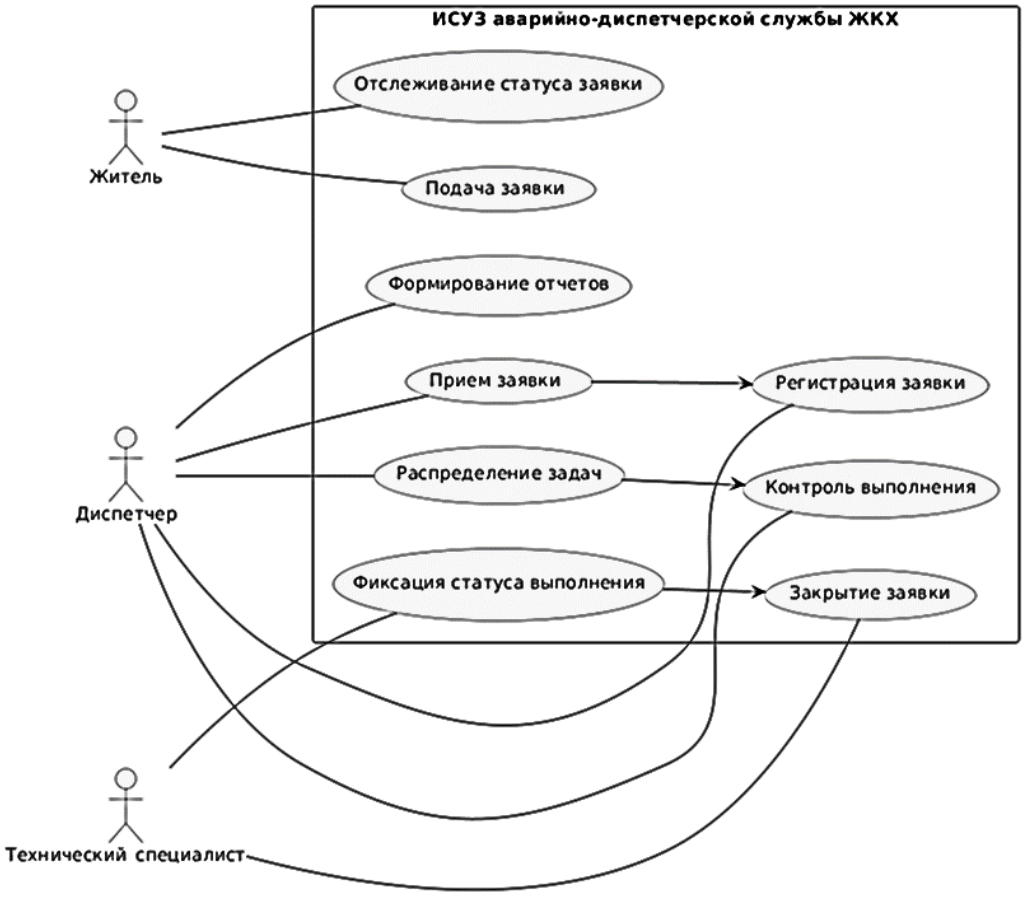



Рисунок 4 – Диаграмма вариантов использования ИСУЗ

Для успешного функционирования ИСУЗ необходимо также учитывать возможные исключительные ситуации, такие как сбои в работе системы или некорректная информация, переданная пользователем, где необходима важность дифференциации функционала и участников в работе по UML [9].

Важно предусмотреть механизмы обратной связи для быстрого реагирования на возникающие проблемы, а также внедрить систему уведомлений, чтобы все участники процесса были проинформированы о статусе заявки и выполнении работ.

Это поможет повысить эффективность работы службы, улучшить взаимодействие с жителями и минимизировать время, затрачиваемое на обработку заявок и выполнение заданий.

## 2.2 Описание вариантов использования

Для более детального понимания работы системы и взаимодействия пользователей с ИСУЗ важно описать основные варианты использования, что позволит точно определить, каким образом пользователи будут взаимодействовать с системой для достижения своих целей. Каждый вариант использования (use case) включает в себя:

- краткое описание процесса;
- предусловия, при которых данный сценарий может быть выполнен;
- основной поток событий, которые выполняются для достижения цели;
- альтернативные потоки, выполняемые в случае отклонений или ошибок [7];
- постусловия, конечные результаты выполнения сценария.

В таблице 3 представлен вариант использования процесса приема и регистрации заявки.

Таблица 3 – Вариант использования процесса приема и регистрации заявки

| Краткое описание | Сценарий использования процесса приема и регистрации заявки |
|---|---|
| Предусловия | Система включена, пользователь авторизован. |
| Основные потоки сценария приема заявки | Житель обращается в службу по телефону или через интернет–портал.<br>Диспетчер вводит данные заявки (адрес, суть проблемы, контактная информация).<br>Система присваивает заявке уникальный идентификатор и сохраняет ее. |
| Основные потоки сценария проверки статуса заявки | Диспетчер проверяет статус заявки в системе.<br>Система отображает актуальную информацию о заявке (статус выполнения, закрепленная бригада).<br>Если заявка решена, диспетчер уведомляет жителя о завершении работ. |
| Основные потоки сценария распределения заявки | Диспетчер проверяет информацию о доступности бригад и распределяет заявку на соответствующую бригаду для выполнения работ.<br>Система автоматически обновляет статус заявки. |
| Альтернативный поток | Если не удается связаться с жителем, система отмечает заявку как "ожидает уточнения". |
| Альтернативный поток при отсутствии свободных бригад | Если на момент регистрации заявки нет свободных бригад, заявка помещается в очередь, а жителю отправляется уведомление о задержке выполнения. |
| Постусловия | Заявка зарегистрирована и ожидает обработки.<br>Заявка может находиться в очереди до момента назначения бригады или разрешения проблемы. |

На рисунке 5 проиллюстрирован процесс управления заявками аварийно–диспетчерской службы ЖКХ на UML–диаграмме активностей, на которой показан как поток управления переходит от одной деятельности к другой [19].

Пример варианта использования процесса приема и регистрации заявки разработан с учетом современных методов, по методологиям разработки ПО [6].

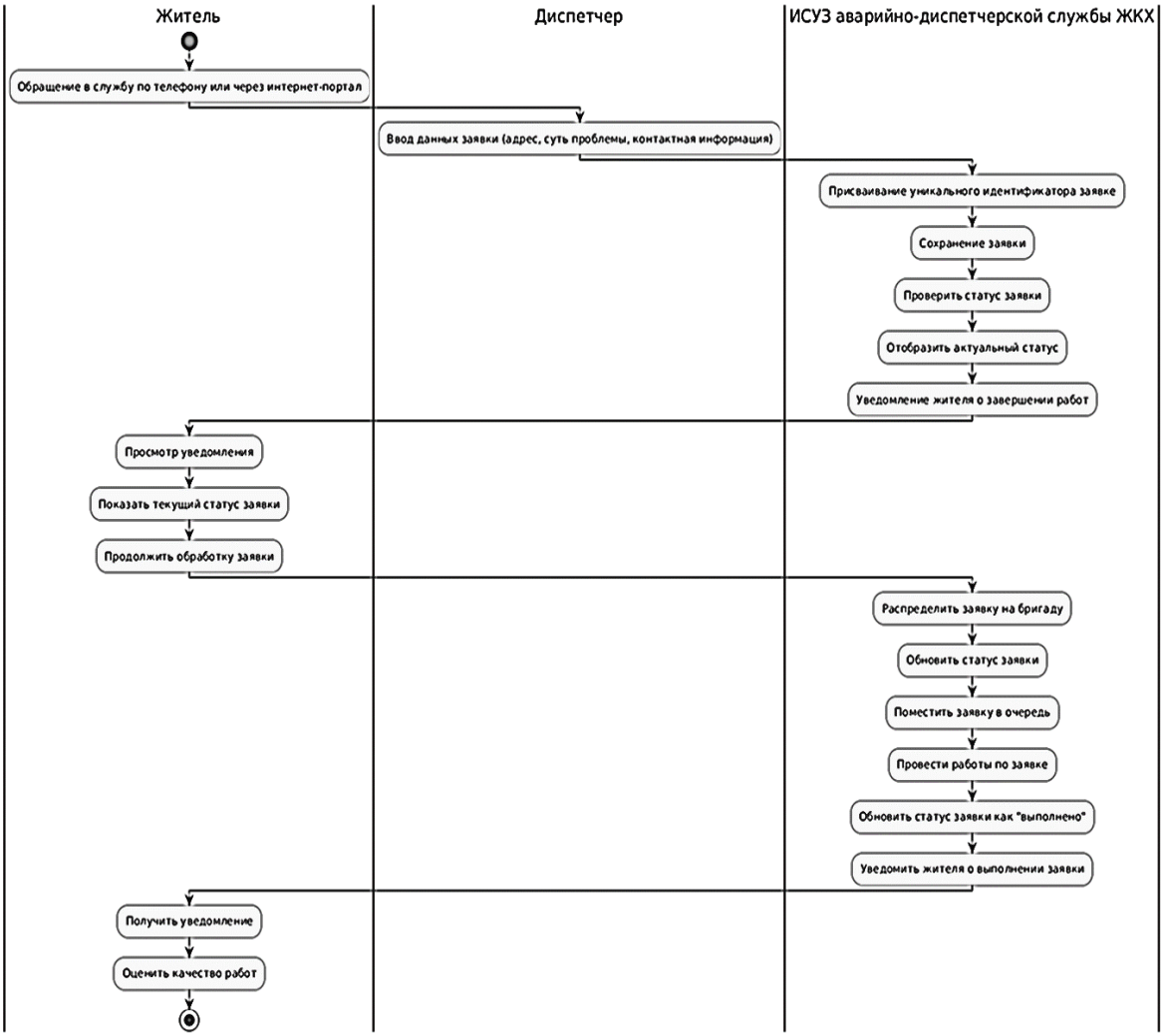



Рисунок 5 – Диаграмма деятельности для управления заявками

Процесс начинается с обращения жителя в службу аварийно–диспетчерской службы ЖКХ через телефон или интернет–портал. Диспетчер вводит данные заявки, включая адрес, суть проблемы и контактную информацию. Система присваивает заявке уникальный идентификатор и сохраняет ее. Житель может проверять статус заявки, получая уведомления о ее текущем состоянии. Система распределяет заявку на бригаду, обновляет статус и помещает ее в очередь на выполнение. После выполнения работ статус обновляется на "выполнено", и житель уведомляется об этом. В конце процесса житель получает уведомление о выполнении и оценивает качество работы.

## 2.3 Выбор методологии проектирования программного обеспечения

Для разработки ИСУЗ аварийно–диспетчерской службы ЖКХ целесообразно рассмотреть несколько популярных методологий: Waterfall, Agile, Scrum, и выбрать наиболее подходящую для выполнения поставленных задач.

Waterfall, или каскадная модель, представляет собой линейный метод разработки программного обеспечения, при котором каждая фаза должна быть завершена прежде, чем начнется следующая [15].

Процесс начинается с тщательного анализа требований, после чего, следовательно, переходят к проектированию, разработке, тестированию и внедрению.

Такой подход идеально подходит для проектов с ясно определенными требованиями, которые не предполагают значительных изменений в ходе разработки.

Agile – гибкая методология разработки, ориентированная на быстрое реагирование на изменения и постоянное взаимодействие с заказчиком и командой [3].

Основные принципы Agile заключаются в инкрементальной разработке с регулярной обратной связью, что позволяет оперативно вносить коррективы в функционал системы.

Scrum, как один из наиболее распространенных фреймворков Agile, включает в себя итерационную модель разработки с краткосрочными циклами, называемыми спринтами, продолжительностью от одной до четырех недель [15].

Каждый спринт завершает разработку работоспособного фрагмента системы, который может быть оценен и протестирован.

В таблице 4 наглядно представлены достоинства и недостатки методологий, перечисленных ранее.

Таблица 4 – Сравнение методологий

| Методология | Описание | Достоинства | Недостатки |
|---|---|---|---|
| Waterfall | Линейный процесс разработки с последовательным переходом от одной фазы к другой. Подходит для проектов с четко сформулированными требованиями. | Точная структура и документация проекта. Подходит для проектов с неизменными требованиями. | Низкая гибкость, сложность внесения изменений, минимальное взаимодействие с заказчиком. |
| Agile | Гибкая методология, ориентированная на взаимодействие с заказчиком и быстрое реагирование на изменения требований. | Гибкость, активное взаимодействие с заказчиком, быстрое реагирование на изменения. | Требует высококвалифицированных разработчиков, минимум документации. |
| Scrum | Методология Agile, разделяющая проект на итерации с выпуском готового функционала по завершении каждого спринта. | Предполагает прозрачность процессов, оперативное внедрение промежуточных версий, а также постоянный анализ и корректировку работы | Метод требует наличия опытных разработчиков и активного взаимодействия между членами команды. |

Для разработки ИСУЗ аварийно–диспетчерской службы ЖКХ наиболее подходящей является методология Agile, так как она позволит гибко реагировать на изменения, что особенно важно для внедрения в условиях ЖКХ.

Применение Agile обеспечит оперативную обратную связь и возможность вносить корректировки на всех этапах проекта, что позволит учесть специфику конкретных аварийных ситуаций и требований пользователей.

## 2.4 Логическое проектирование программного обеспечения

Логическое проектирование системы включает в себя определение ее архитектуры, структуры данных, а также взаимодействие компонентов и модулей. В случае ИСУЗ аварийно–диспетчерской службы ЖКХ была выбрана многослойная архитектура, которая разделяет систему на несколько функциональных уровней, таких как пользовательский интерфейс, бизнес–логика и управление данными [10]. Данный подход обеспечивает четкую организацию и взаимодействие между различными слоями системы.

Многослойная архитектура позволяет разделить функциональные обязанности, обеспечивая модульность и упрощая процесс сопровождения и обновления системы. Основные компоненты архитектуры включают:

- пользовательский интерфейс (UI), который организует взаимодействие с пользователем;
- бизнес–логику, ответственную за управление основными процессами и операциями системы;
- управление данными, которые отвечают за взаимодействие с базой данных и другими источниками информации.

Для реализации бизнес–логики и управления данными в системе была выбрана архитектурная модель MVC (Model–View–Controller).

Эта модель представляет собой паттерн проектирования, который делит приложение на три взаимосвязанных компонент (рисунок 6).

Model (Модель) – управляет данными системы, их обработкой и хранением.

View (Представление) – приводит данные пользователю посредством предоставления интерфейса для взаимодействия.

Controller (Контроллер) – управляет связью между Model и View, координируя действия и обеспечивая их согласованную работу.


Рисунок 6 – Основные взаимодействия в шаблоне MVC пользователя с системой

Такая структура обеспечивает гибкость системы, позволяя легко обновлять или заменять отдельные компоненты без влияния на другие части системы. В результате обеспечивается модульность и масштабируемость, что особенно важно для больших и сложных систем. Например, если в системе изменяются бизнес–процессы, связанные с аварийными заявками или расчетами стоимости работ, эти изменения могут быть локализованы в компоненте бизнес–логики без необходимости модификации пользовательского интерфейса или системы управления данными [11].

Для проектирования архитектуры ПО ИСУЗ также необходимо определить ключевые модули системы, их взаимосвязи и взаимодействие. Тщательная проработка архитектуры на этапе логического проектирования также снижает вероятность возникновения ошибок в процессе разработки и последующей эксплуатации системы.

На рисунке 7 представлена диаграмма пакетов архитектуры MVC.

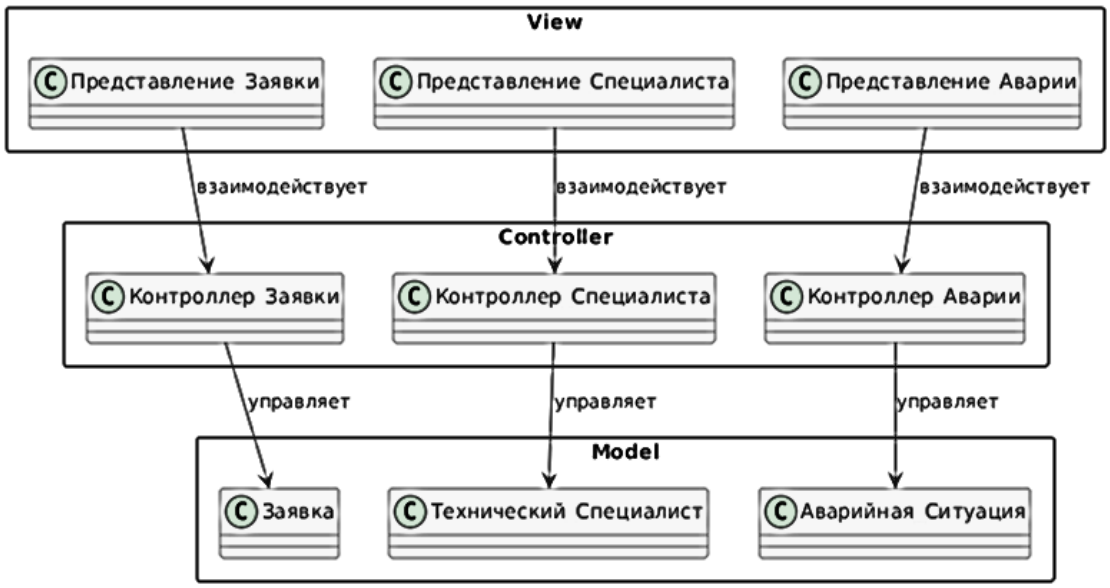



Рисунок 7 – Диаграмма пакетов архитектуры MVC

На рисунке 8 показано, что входит в каждый из пакетов.

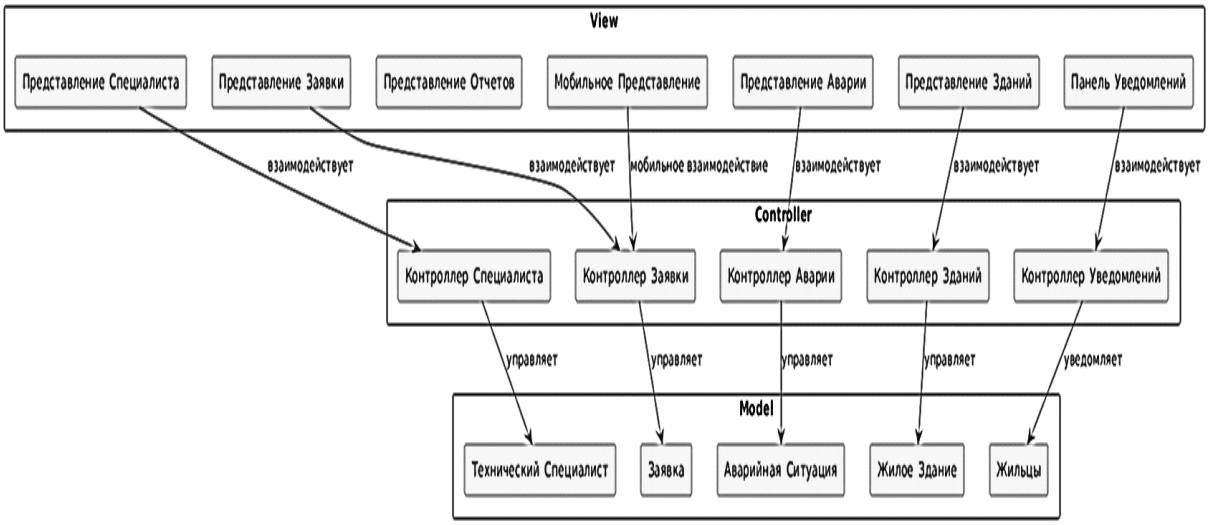



Рисунок 8 – Диаграмма компонентов архитектуры MVC

Шаблон MVC гарантирует ясное разделение функций между представлением данных, их обработкой и взаимодействием с пользователем, что делает его идеальным для разработки сложных систем с потенциалом для будущего расширения.

## 2.5 Проектирование модели данных

Для надежного хранения и оперативного доступа к данным рекомендуется использовать базы данных (БД). Базы данных предлагают эффективный способ организации, хранения и управления информацией, что способствует поддержанию целостности и актуальности данных о заявках и клиентах.

В контексте информационной системы управления заявками наиболее подходящей является реляционная база данных, которая идеально подходит для хранения информации о заявках и сотрудниках, а также для управления этими данными в процессе обработки заявок. В реляционных базах данных информация организована в таблицах, каждая из которых состоит из записей (строк) и столбцов. Записи содержат данные о конкретных объектах системы, таких как заявки, бригады или клиенты, в то время как столбцы определяют тип хранимых данных, например, статус или контактную информацию [8].

Для обеспечения функциональности разработана логическая модель данных, которая включает три основные таблицы:

- таблица «Заявки»: содержит уникальный идентификатор заявки, дату подачи, адрес, статус выполнения и дополнительную информацию, связанную с текущим состоянием заявки. Эта таблица позволяет отслеживать все запросы на обслуживание;
- таблица «Бригады»: хранит информацию о бригадах и технических специалистах, которые занимаются обработкой и выполнением заявок. Включает данные о сотрудниках, такие как идентификатор специалиста, имя, контактная информация и должность. Это позволяет обеспечить удобное распределение задач и отслеживать занятость специалистов;
- таблица «Жители»: содержит данные о клиентах, подавших заявки, включая их уникальный идентификатор, ФИО, адрес проживания и контактные данные. Это дает возможность поддерживать актуальную

информацию о жителях, а также связывать конкретные заявки с соответствующими клиентами.

Такая структура базы данных позволяет эффективно управлять данными, обеспечивает целостность и безопасность информации, а также ускоряет доступ к данным для выполнения задач аварийно–диспетчерской службы. Для визуализации взаимосвязей между сущностями системы разработана UML–диаграмма (рисунок 9), которая иллюстрирует связи типа "один ко многим" между таблицами.

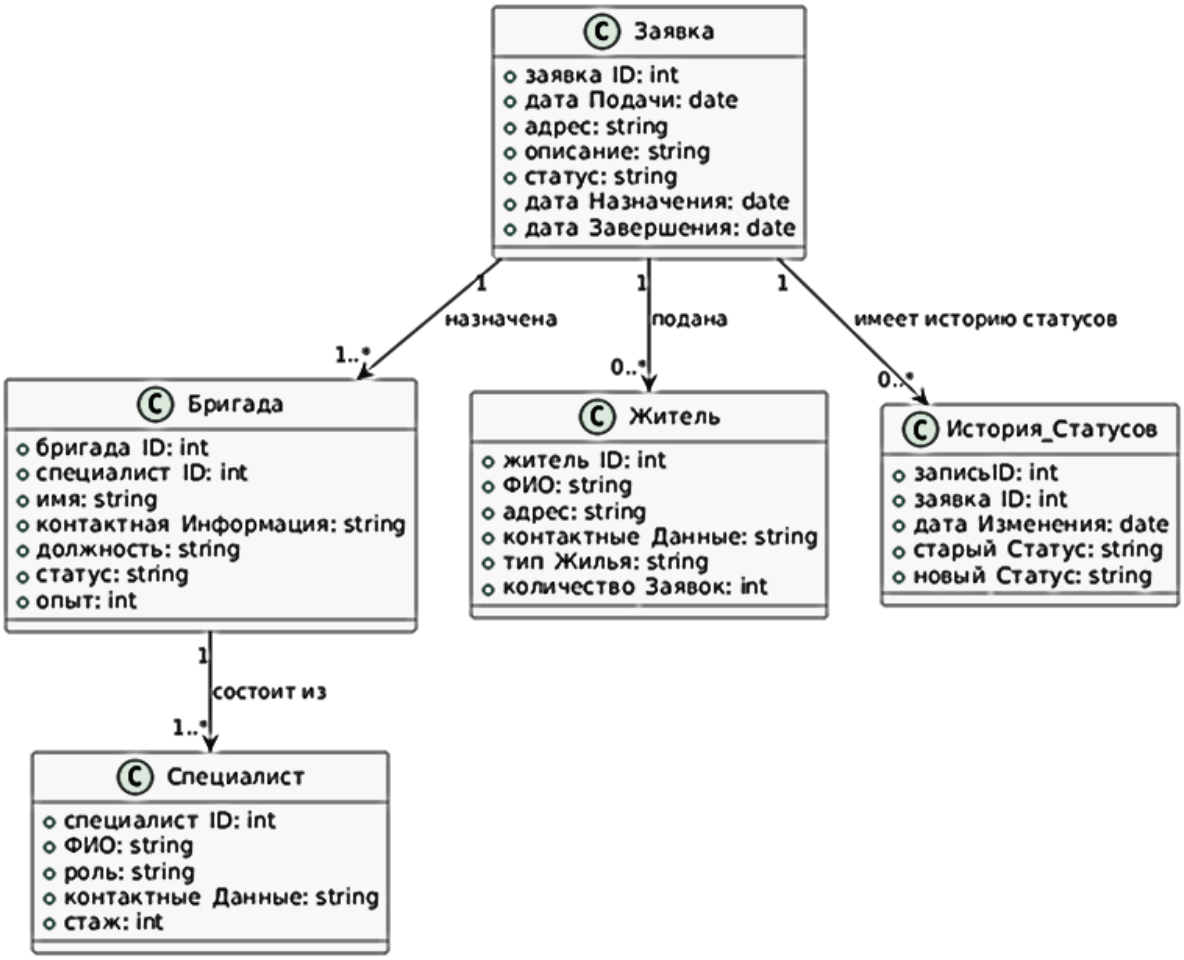



Рисунок 9 – Спроектированная логическая модель данных

Например, один житель может подать несколько заявок, а одна бригада может обслуживать несколько заявок, что отражает потребности в гибкости и масштабируемости системы.

Выводы по главе 2

В процессе проектирования информационной системы управления заявками аварийно–диспетчерской службы ЖКХ был разработан четкий и структурированный подход, обеспечивающий эффективное функционирование системы. Важным шагом стал выбор гибкой методологии Agile, которая позволит оперативно реагировать на изменения и вносить коррективы на всех этапах разработки. Это обеспечит соответствие системы актуальным требованиям и специфике работы службы, что особенно важно в условиях ЖКХ, где часто происходят непредсказуемые изменения.

Также была спроектирована многослойная архитектура системы, что позволит разделить функциональные обязанности, повысив гибкость и модульность. Это важно для обеспечения долгосрочной поддержки системы и возможности ее расширения. В рамках логического проектирования была выбрана модель MVC, которая эффективно разделяет взаимодействие с пользователем, обработку данных и бизнес–логику, что повышает масштабируемость и уменьшает сложность последующих изменений.

Дополнительно, была спроектирована логическая модель данных, включающая реляционную базу данных для управления заявками, специалистами и жителями. Такая структура данных позволяет эффективно отслеживать статус заявок и обеспечивать связь между участниками процесса, что в свою очередь минимизирует вероятность ошибок и ускоряет процесс работы с заявками.

В результате, проектирование ИСУЗ аварийно–диспетчерской службы ЖКХ обеспечит создание системы, способной не только эффективно управлять заявками, но и обеспечивать гибкость для дальнейших изменений и оптимизаций, что важно для постоянного улучшения качества обслуживания населения.

# Глава 3 Реализация и тестирование программного обеспечения ИС

## 3.1 Выбор средств разработки ПО

Разработка программного обеспечения требует тщательного выбора технологий, которые будут обеспечивать как краткосрочную, так и долгосрочную стабильность системы. При выборе инструментов для создания ИСУЗ были учтены такие факторы, как масштабируемость, поддержка сообщества, производительность и простота интеграции.

Из вышесказанного следует, что для реализации проекта требуется выбрать язык программирования и фреймворк:

- python был выбран в качестве основного языка программирования, так как он обеспечивает быструю разработку, высокую читаемость кода и удобную экосистему библиотек, широко используется в веб–разработке, и его удобство в работе с разными типами данных, делает его идеальным для построения сложных информационных систем;
- django – это высокоуровневый веб–фреймворк на Python, который предлагает наличие встроенных средств для управления аутентификацией, маршрутизацией, ORM для работы с базой данных, и системы шаблонов для фронтенда. Одним из ключевых преимуществ Django является его система администрирования, которая позволяет быстро разрабатывать и тестировать интерфейсы управления данными, а также предлагает REST API через библиотеки, такие как Django REST Framework, что делает возможным создание API для интеграции с мобильными приложениями или сторонними системами [2].

Для хранения данных выбрана PostgreSQL, одна из самых мощных реляционных баз данных с открытым исходным кодом. Так как PostgreSQL обеспечивает:

- поддержку сложных транзакций и индексов, что необходимо для

работы с большим количеством заявок;

- расширенные возможности обработки геоданных (PostGIS), что может быть полезно для привязки заявок к географическим координатам (например, для точного указания места аварий);
- надежность и высокая производительность при масштабировании системы [23].

Для разработки фронтенда была выбрана библиотека React, предназначенная для создания пользовательских интерфейсов, которая обеспечивает отличную производительность за счет использования "виртуального DOM". React позволяет создавать интерактивные и динамичные интерфейсы, что важно для системы, где пользователи часто работают с таблицами данных, фильтрами, а также обновляют информацию в режиме реального времени [24].

В паре с React использовался Redux для управления состоянием приложения, что облегчает обработку данных на клиентской стороне. Redux особенно эффективно применяется при работе с асинхронными запросами к серверу, позволяя централизованно контролировать состояние и обеспечивать стабильную синхронизацию данных [25].

Для управления исходным кодом использовалась система Git, а для работы и контроля версий – платформа GitHub. Git позволяет легко отслеживать изменения в коде, откатываться к предыдущим версиям, а также организовать параллельную работу нескольких разработчиков через ветвление [22].

На стороне сервера используется Nginx для обработки запросов и балансировки нагрузки. В связке с ним работает Gunicorn, WSGI–сервер, который обеспечивает быстрый запуск Python–приложений [20]. Nginx позволяет эффективно обслуживать статические файлы (CSS, JavaScript) и обрабатывать SSL–сертификаты, а Gunicorn отвечает за выполнение Python–кода, обеспечивая стабильную и быструю работу серверной логики приложения.

### 3.2 Разработка логической структуры базы данных

Разработка базы данных играет критическую роль в архитектуре ИСУЗ, так как она определяет, как будут храниться и обрабатываться данные заявок, пользователей и служебной информации. В системе есть два основных типа пользователей: клиенты, создающие заявки, и операторы службы ЖКХ, обрабатывающие эти заявки, а также, есть администрация, которая управляет настройками систем. Важно правильно спроектировать роли и права доступа, чтобы разграничить, какие действия могут выполнять пользователи разных типов.

Заявки (Request) – основная сущность системы, содержит такие данные:

- идентификатор заявки;
- тип заявки;
- описание проблемы;
- статус;
- дата поступления и выполнения;
- управляющая компания;
- бланк заявки;
- город.

Справочник, который содержит перечень различных видов аварийных ситуаций, таких как утечка газа, повреждение электросетей и так далее. Это позволяет структурировать информацию о заявках и легко находить нужную информацию в типы заявок (RequestType).

Этот справочник определяет возможные состояния заявки (например, "выполнена", "ожидает", "отложена", "в работе"). Каждый статус заявок (RequestStatus) должен быть связан с определенными правами изменения, чтобы предотвратить неправильное использование. На рисунке 10 представлен фрагмент кода модели данных.

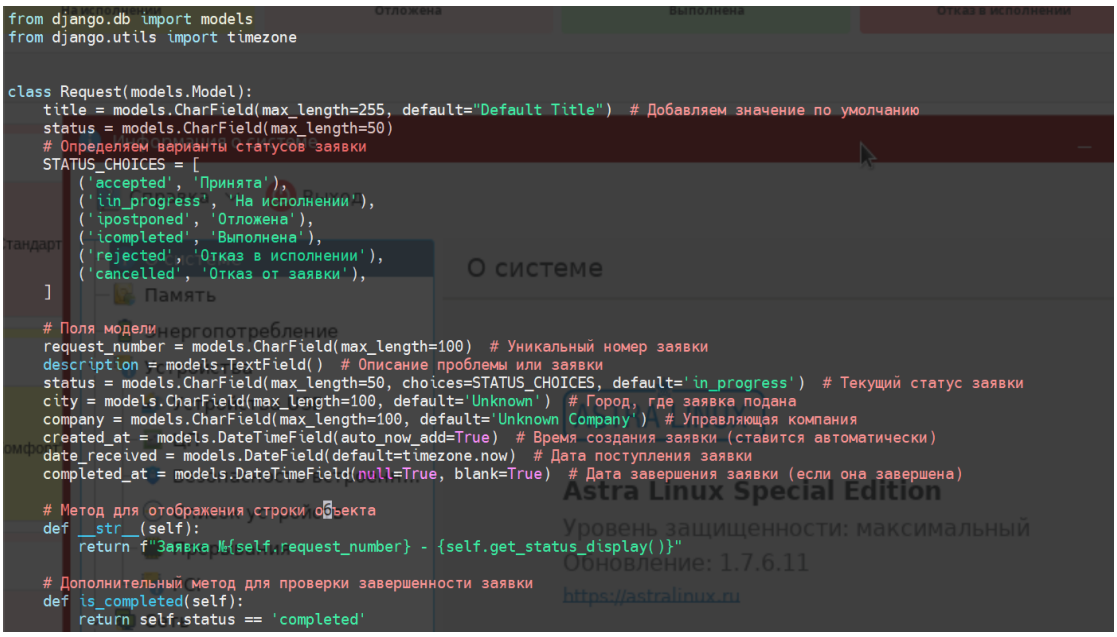



Рисунок 10 – Фрагмент кода модели данных

Для повышения производительности системы используются индексы для наиболее часто запрашиваемых данных (например, статусы заявок и идентификаторы пользователей), что позволяет ускорить поиск и фильтрацию данных в больших таблицах. На рисунке 11 представлен фрагмент кода обработки запросов от пользователей.

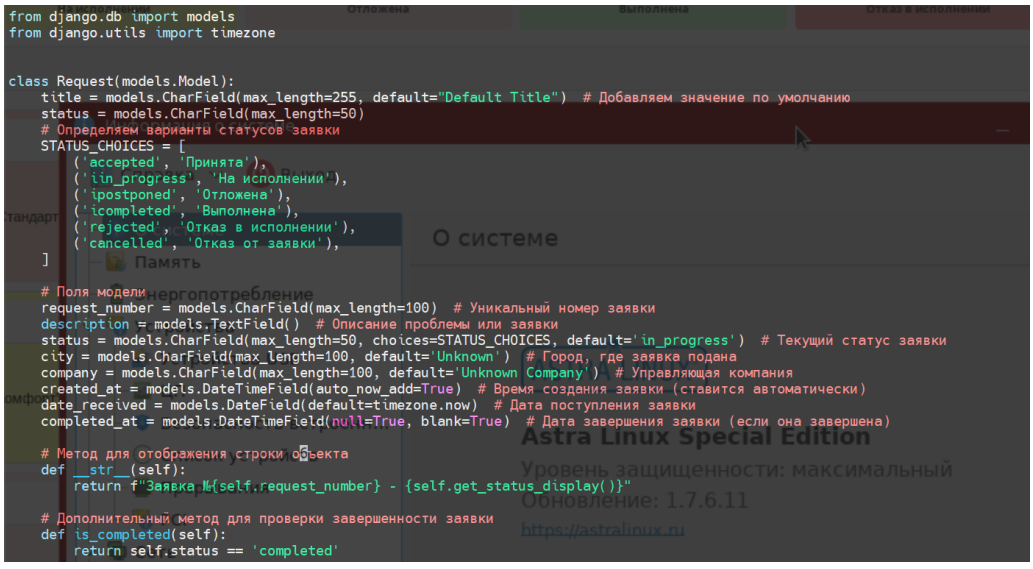



Рисунок 11 – Фрагмент кода обработки запросов от пользователей

Для обеспечения гибкости и масштабируемости базы данных, в дальнейшем предусмотрено расширение функционала системы, что позволит добавлять новые типы заявок, статусы и справочники без значительных изменений в основной структуре. Важно также внедрить систему резервного копирования данных, чтобы минимизировать риски потери информации и обеспечить устойчивость системы к сбоям. Дополнительная оптимизация запросов и регулярные обновления индексов будут способствовать улучшению производительности базы данных по мере роста объема информации.

### 3.3 Проектирование и разработка пользовательского интерфейса

Пользовательский интерфейс должен быть интуитивно понятным и адаптивным для использования как операторами, так и клиентами. Основные страницы:

- форма создания заявки – ключевая точка взаимодействия клиента с системой, она должна быть простой и понятной, с минимальными полями для заполнения, а также возможность прикрепить фотографии, или дополнительные документы;
- операторы аварийно–диспетчерской службы должны иметь возможность быстро просматривать новые и текущие заявки, реализована система фильтров по статусам заявок и дате создания для упрощения поиска нужной информации;
- страница, на которой оператор может видеть все детали заявки, изменять ее статус, а также добавлять комментарии или изменять ответственного за выполнение задачи.

Реализация интерфейса заключается в следующем:

- интерфейс реализован на основе компонентов React, что делает его модульным и легко расширяемым, каждый компонент отвечает за

свою часть логики (форма заявки, таблица заявок, фильтры), что упрощает как процесс разработки и тестирование;

– адаптивный дизайн спроектирован для корректного отображения как на настольных компьютерах, так и на мобильных устройствах. Применяются медиазапросы CSS, а библиотека Bootstrap помогает создать гибкую сетку интерфейса.

Для разработки визуальной части интерфейса, включая графические элементы, цветовую палитру и макеты, используется онлайн–графический редактор Figma.

На рисунке 12 представлен макет всех заявок от жильцов.

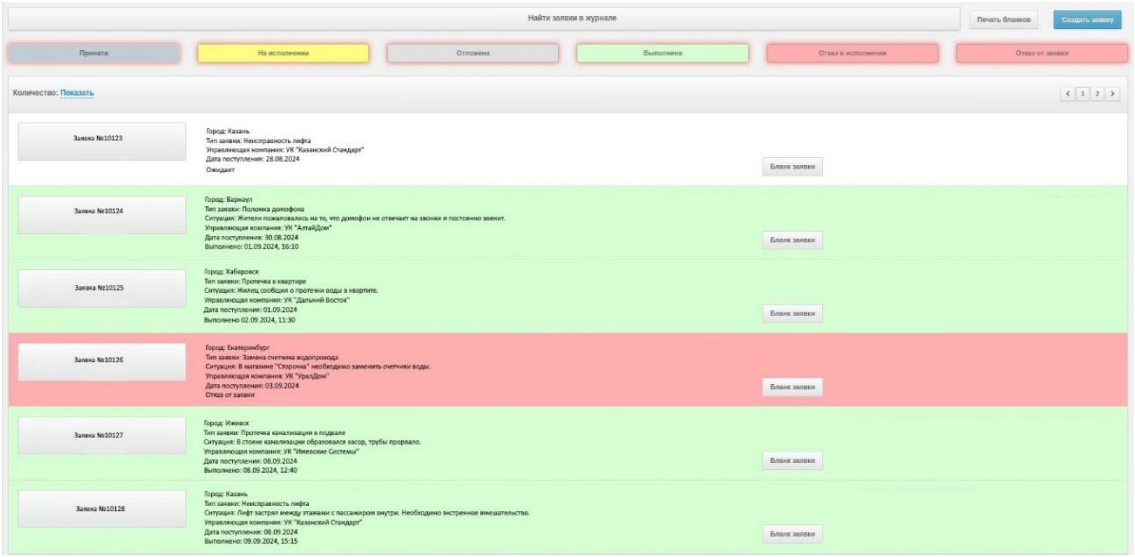



Рисунок 12 – Макет формы списка заявок от жильцов

Эффективность работы системы напрямую зависит от интуитивности интерфейса и скорости обработки заявок. Для обеспечения полноценного взаимодействия с системой, интерфейс включает механизмы визуальной индикации и информативных уведомлений. Операторы могут получать мгновенные обновления статусов заявок и отслеживать их в режиме реального времени, что способствует оперативному реагированию на аварийные ситуации. Клиенты же получают уведомления о принятых мерах, изменениях

статусов и сроках выполнения заявок, что повышает уровень их удовлетворенности.

Внедрение таких технологий как React и Bootstrap позволит не только создать адаптивный и масштабируемый интерфейс, но и заложить основу для дальнейшего расширения системы, например, добавления новых функций или интеграции с другими системами ЖКХ [14]. Гибкость структуры системы позволяет легко обновлять и совершенствовать ее в соответствии с потребностями пользователей и новыми требованиями.

### 3.4 Маршрутизация веб–страниц приложения

Маршрутизация – важная часть любой веб–системы, так как она отвечает за организацию переходов между страницами и передачу данных через URL.

Django предлагает мощную систему маршрутизации через файл urls.py. Для каждого маршрута указывается соответствующий ему обработчик (view–функция или класс) [21]. Примеры маршрутов:

- /login/ – страница для авторизации пользователей, форма входа с проверкой введенных данных;
- /dashboard/ – панель оператора с доступом к заявкам, как основной интерфейс оператора с таблицей заявок и возможностью их сортировки и фильтрации;
- /requests/<id>/ – детальная страница заявки, оператор может изменять статус заявки, просматривать ее историю, добавлять комментарии и загружать документы.

Некоторые маршруты требуют обработки динамических данных, таких как идентификаторы заявок. Django позволяет легко создавать маршруты через передачу переменных в URL.

Это дает возможность строить маршруты, которые динамически

отображают нужную информацию в зависимости от запроса. На рисунке 13 представлен интерфейс формы списка заявок от жильцов.

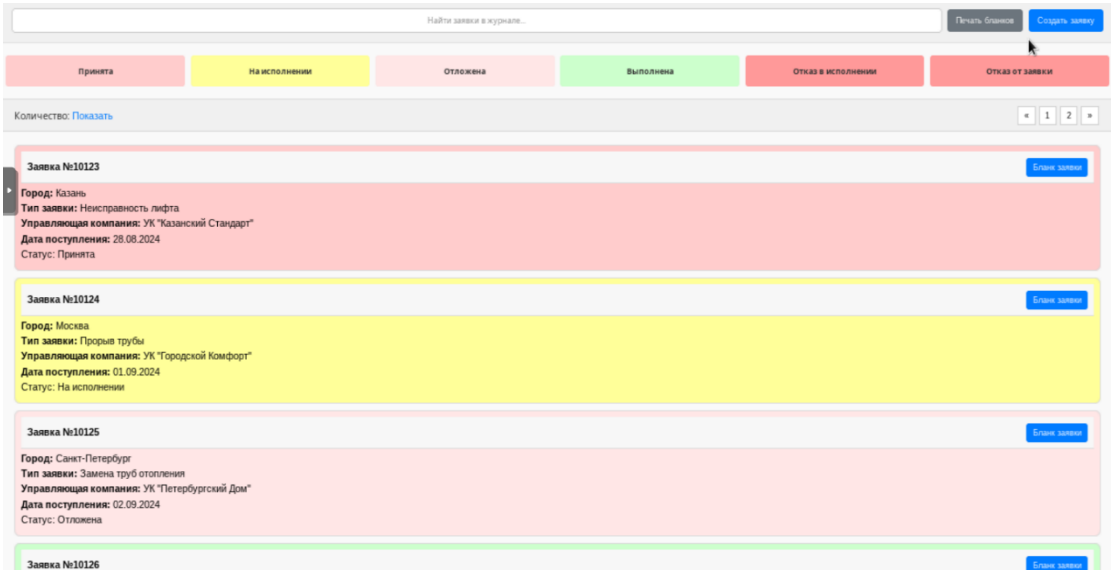



Рисунок 13 – Интерфейс формы списка заявок от жильцов

При нажатии на создание заявки можно выбрать тип заявки так как форма позволяет выбрать, кто создает заявку: житель квартиры или сотрудник (общедомовая заявка), что может повлиять на дальнейшую маршрутизацию или приоритет выполнения заявки. Для заполнения адреса можно ввести вручную или выбрать из выпадающего списка при вводе. Необходимо указывать поля Фамилии, Имени и Отчества для идентификации заявителя. А также Контактные данные номера телефона и адреса электронной почты для связи с заявителем. В выпадающем списке можно выбрать один из готовых шаблонов, таких как:

- отключение ГВС (горячее водоснабжение);
- отключение ХВС (холодное водоснабжение);
- слабый напор воды;
- проблемы с лифтом;
- отопление;
- работа плотника и другие.

Подготовлены типы заявок, которые можно выбрать, такие как:

– ремонт дома;
– территория;
– коммерческая;
– аварийная;
– вандализм;
– общее имущество;
– текущий ремонт.

Есть возможность указать организацию, которая будет выполнять заявку, а также добавить конкретного специалиста, который будет решать проблему. Кнопки управления:

– кнопка "Сохранить" сохраняет заявку в системе для дальнейшей обработки;
– кнопка "Отмена" отменяет процесс создания заявки.

Можно указать, для кого предназначена заявка: для всех в доме или только для заявителя, что особенно актуально для общедомовых проблем.

### 3.5 Архитектура программного обеспечения

Для понимания процесса тестирования важно рассмотреть архитектуру программного обеспечения ИСУЗ. Система разрабатывалась на основе многослойной архитектуры, которая обеспечивала модульность, масштабируемость и легкость в поддержке и обновлении системы. Основные слои системы представлены ниже.

Клиентский слой (Frontend):

– разработан на основе React с использованием Redux для управления состоянием приложения;
– включает в себя формы для создания и редактирования заявок,

управление пользователями, отображение таблиц заявок и их статусов;

– поддерживает адаптивный интерфейс для корректного отображения на различных устройствах (смартфонах, планшетах, настольных компьютерах).

Серверный слой (Backend):

– построен на Django с использованием Django REST Framework для создания API;
– обрабатывает все запросы от клиентской части и взаимодействует с базой данных;
– реализует бизнес–логику обработки заявок, управление пользователями, а также обработку прав доступа на уровне API.

Слой данных:

– система использует PostgreSQL как основную СУБД для хранения данных, включая данные о заявках, пользователях, ролях и статусах;
– для хранения и обработки геоданных, таких как координаты местоположения заявки, используется расширение PostGIS.

В системе также предусмотрены интеграции с внешними сервисами, такими как SMS–уведомления и почтовые системы для оповещения пользователей о статусе заявок.

На рисунке 14 представлена архитектурная часть ПО ЖКХ, где UI (User Interface) – пользовательский интерфейс, который отображает форму, принимает заявку и показывает статус заявки. Controllers – контроллеры, которые обрабатывают запросы и обновляют статусы. Business Logic – бизнес–логика, отвечающая за обработку заявок и их статусов. Database (DB) – СУБД, хранящая данные о заявках и их статусах External API – взаимодействие с внешними системами по требованию.

Например, интеграция с внешней системой мониторинга или управления.

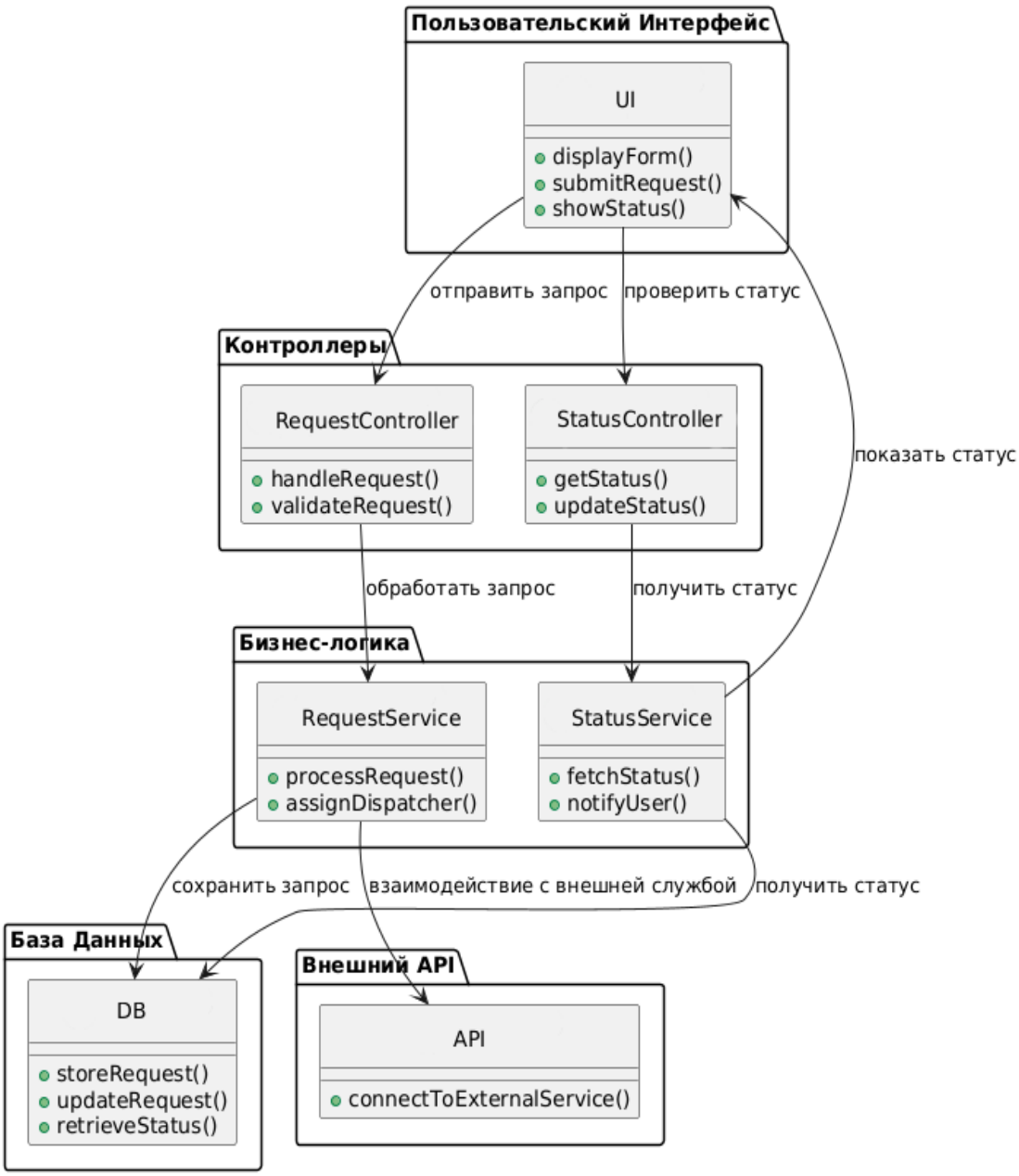



Рисунок 14 – Архитектура ПО ЖКХ

Для обеспечения устойчивости и масштабируемости системы в процессе эксплуатации, архитектура ПО должна предусматривать возможность гибкой настройки и интеграции с новыми сервисами и компонентами. Это включает в себя поддержку облачных решений для распределения нагрузки, использование микросервисной архитектуры, которая позволит изолировать ключевые функциональные блоки и упростить их обновление и масштабирование. Также важно предусмотреть мониторинг

состояния системы и сбор статистики для своевременного выявления и устранения потенциальных проблем в работе программы, что поможет поддерживать высокое качество обслуживания пользователей.

## 3.6 Тестирование программного обеспечения информационной системы управления заявками

Тестирование программного обеспечения является важным этапом в процессе разработки информационной системы управления заявками, поскольку оно обеспечивает выявление потенциальных ошибок и проверку соответствия системы предъявляемым требованиям [13].

В ходе разработки ИСУЗ было применено несколько видов тестирования, которые позволили удостовериться в функциональности и стабильности всех компонентов системы, а также проверить взаимодействие между ними.

Функциональное тестирование охватывает проверку всех функций системы:

- создание новой заявки;
- изменение статусов;
- управление пользователями.

Важно отметить, что функциональное тестирование было направлено на обеспечение того, чтобы каждая из этих функций выполнялась корректно, без ошибок и в соответствии с установленными требованиями. Кроме того, оно гарантировало, что системы обработки заявок не имеют уязвимостей, которые могут привести к сбоям или неправильной обработке данных.

На рисунке 15 представлены виды тестирования и их различие в проведении.


Рисунок 15 – Пирамида проведения тестирования

Интеграционное тестирование представляло собой проверку взаимодействия различных модулей системы. Было важным этапом для проверки взаимодействия различных компонентов системы и обеспечения их совместной работы. На этом этапе акцент был сделан на том, чтобы убедиться, что все модули системы правильно взаимодействуют друг с другом. Это тестирование охватывало взаимодействие базы данных, серверной логики и пользовательского интерфейса. В частности, важно было проверить, как серверная часть обрабатывает запросы к базе данных и правильно ли отображаются результаты на интерфейсе пользователя.

Задача заключалась в том, чтобы убедиться, что данные, поступающие от серверной части, обрабатываются базой данных без потерь и искажений, а затем правильно передаются в интерфейс для отображения. Например, при запросе списка заявок от жильцов должно было быть гарантировано, что все данные, связанные с заявками, корректно передаются от сервера через базу

данных и отображаются пользователю в нужном формате. Такой подход гарантирует надежность и стабильность работы системы, особенно в условиях интенсивной эксплуатации, когда одновременно обрабатывается множество запросов.

Кроме того, интеграционное тестирование позволило проверить взаимодействие с внешними сервисами, такими как системы уведомлений, почтовые серверы, а также платежные системы. Это особенно важно для ИСУЗ, поскольку успешная интеграция с такими сервисами гарантирует, что система будет правильно взаимодействовать с внешними платформами и обеспечивать пользователю все необходимые функции в процессе эксплуатации.

Рисунок 16 демонстрирует результаты API системы, отображающие список заявок от жильцов.

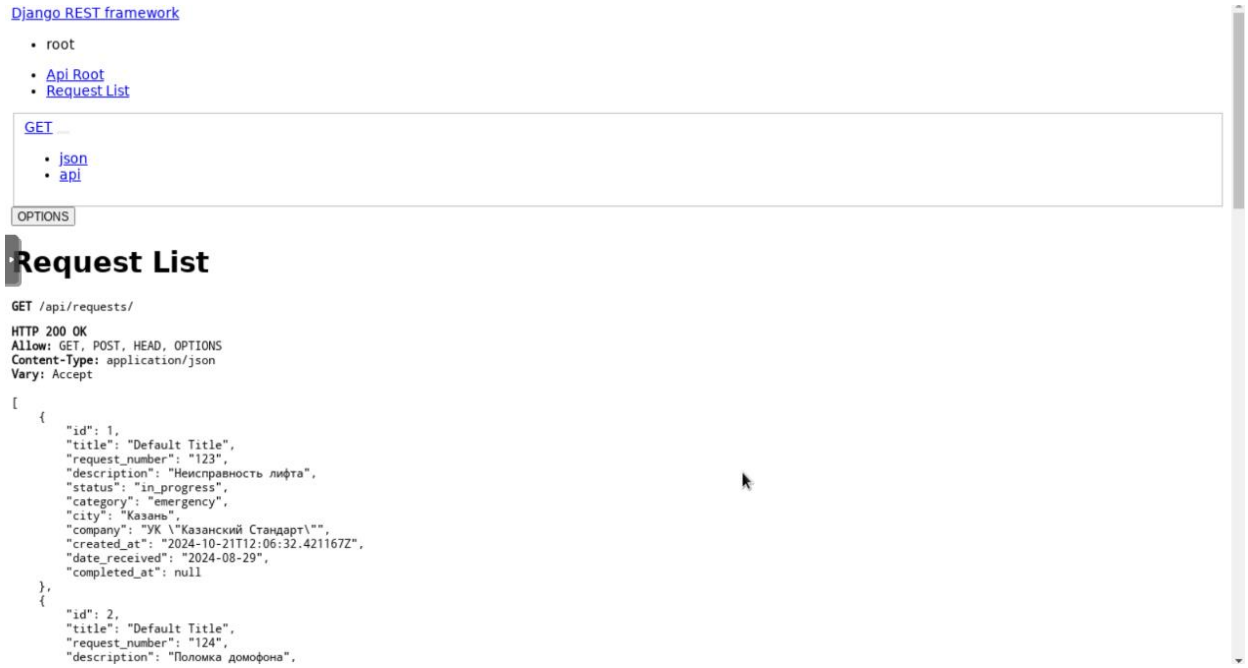



Рисунок 16 – Результаты API системы списка заявок от жильцов

Проводилось тестирование адаптивности интерфейса на различных устройствах и в различных браузерах. Были проведены тесты с целью

проверки корректности отображения элементов интерфейса и правильной работы всех функций, таких как формы, кнопки, фильтры.

На основе полученных результатов интеграционного тестирования были внесены необходимые корректировки и оптимизации для повышения производительности системы, улучшения скорости обработки данных и устранения возможных ошибок, которые могли возникнуть при взаимодействии модулей. Эти корректировки сделали систему более устойчивой и надежной в реальных условиях эксплуатации, минимизировав риски возникновения сбоев или потери данных при высокой нагрузке.

На рисунке 17 отображены результаты модульных тестов системы. На рисунках 18 и 19 представлены фрагменты кода автотестов модульного тестирования.

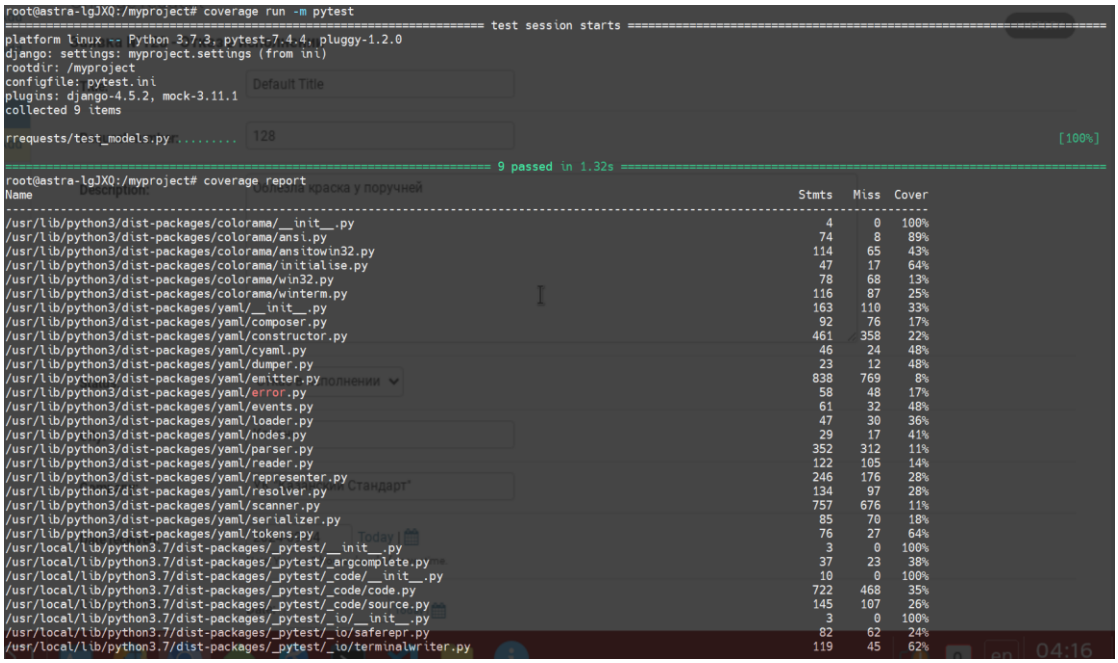
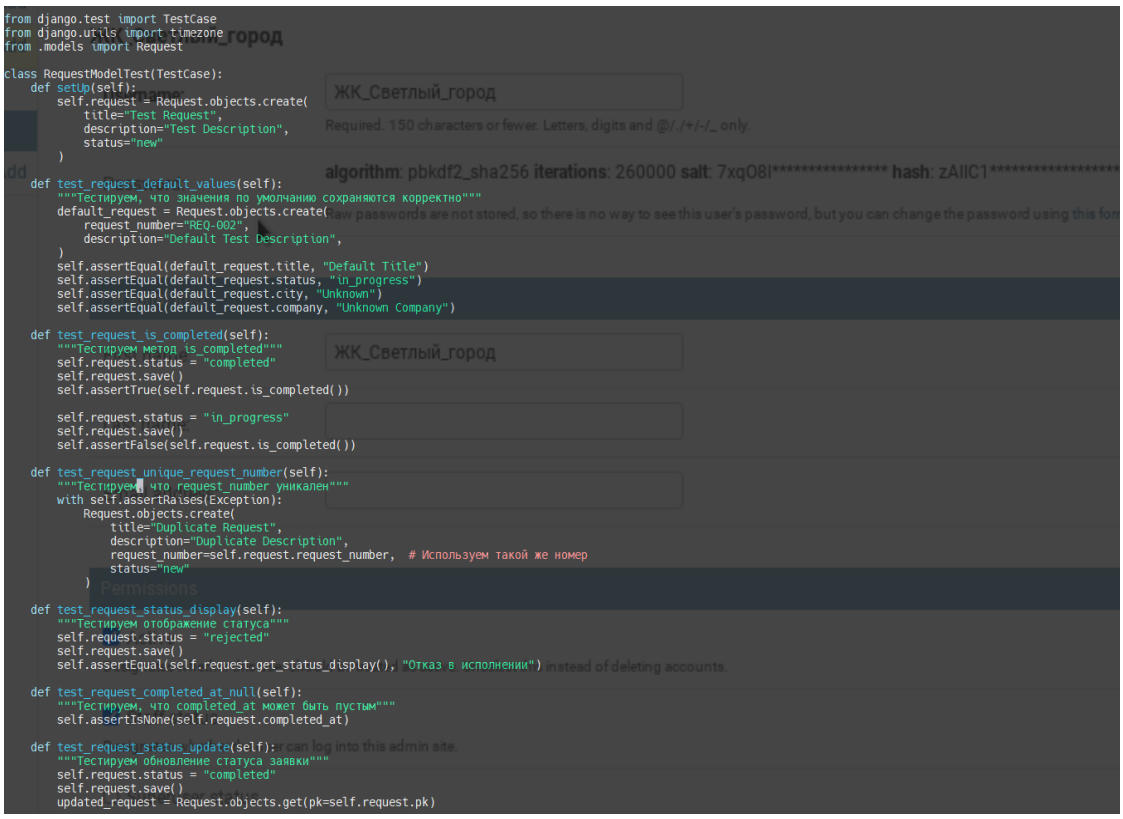
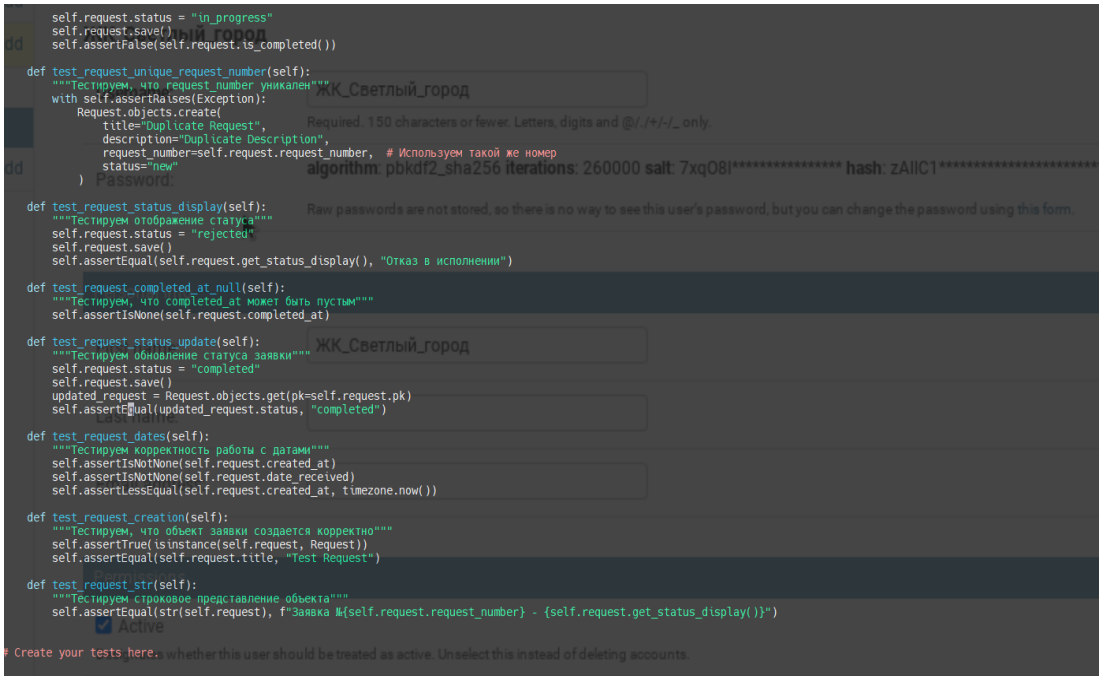



Рисунок 17 – Результаты модульных тестов системы



```python
from django.test import TestCase
from django.utils import timezone
from .models import Request

class RequestModelTest(TestCase):
    def setUp(self):
        self.request = Request.objects.create(
            title="Test Request",
            description="Test Description",
            status="new"
        )

    def test_request_default_values(self):
        """Тестируем, что значения по умолчанию сохраняются корректно"""
        default_request = Request.objects.create(
            request_number="REQ-002",
            description="Default Test Description",
        )
        self.assertEqual(default_request.title, "Default Title")
        self.assertEqual(default_request.status, "in_progress")
        self.assertEqual(default_request.city, "Unknown")
        self.assertEqual(default_request.company, "Unknown Company")

    def test_request_is_completed(self):
        """Тестируем метод is_completed"""
        self.request.status = "completed"
        self.request.save()
        self.assertTrue(self.request.is_completed())

        self.request.status = "in_progress"
        self.request.save()
        self.assertFalse(self.request.is_completed())

    def test_request_unique_request_number(self):
        """Тестируем, что request_number уникален"""
        with self.assertRaises(Exception):
            Request.objects.create(
                title="Duplicate Request",
                description="Duplicate Description",
                request_number=self.request.request_number,  # Используем такой же номер
                status="new"
            )

    def test_request_status_display(self):
        """Тестируем отображение статуса"""
        self.request.status = "rejected"
        self.request.save()
        self.assertEqual(self.request.get_status_display(), "Отказ в исполнении")

    def test_request_completed_at_null(self):
        """Тестируем, что completed_at может быть пустым"""
        self.assertIsNone(self.request.completed_at)

    def test_request_status_update(self):
        """Тестируем обновление статуса заявки"""
        self.request.status = "completed"
        self.request.save()
        updated_request = Request.objects.get(pk=self.request.pk)
```

Рисунок 18 – Первая часть фрагмента кода автотестов модульного тестирования



```python
        self.request.status = "in_progress"
        self.request.save()
        self.assertFalse(self.request.is_completed())

    def test_request_unique_request_number(self):
        """Тестируем, что request_number уникален"""
        with self.assertRaises(Exception):
            Request.objects.create(
                title="Duplicate Request",
                description="Duplicate Description",
                request_number=self.request.request_number,  # Используем такой же номер
                status="new"
            )

    def test_request_status_display(self):
        """Тестируем отображение статуса"""
        self.request.status = "rejected"
        self.request.save()
        self.assertEqual(self.request.get_status_display(), "Отказ в исполнении")

    def test_request_completed_at_null(self):
        """Тестируем, что completed_at может быть пустым"""
        self.assertIsNone(self.request.completed_at)

    def test_request_status_update(self):
        """Тестируем обновление статуса заявки"""
        self.request.status = "completed"
        self.request.save()
        updated_request = Request.objects.get(pk=self.request.pk)
        self.assertEqual(updated_request.status, "completed")

    def test_request_dates(self):
        """Тестируем корректность работы с датами"""
        self.assertIsNotNone(self.request.created_at)
        self.assertIsNotNone(self.request.date_received)
        self.assertLessEqual(self.request.created_at, timezone.now())

    def test_request_creation(self):
        """Тестируем, что объект заявки создается корректно"""
        self.assertTrue(isinstance(self.request, Request))
        self.assertEqual(self.request.title, "Test Request")

    def test_request_str(self):
        """Тестируем строковое представление объекта"""
        self.assertEqual(str(self.request), f"Заявка №{self.request.request_number} - {self.request.get_status_display()}")

# Create your tests here.
```

Рисунок 19 – Вторая часть фрагмента кода автотестов модульного тестирования

Система была протестирована на работу под высокой нагрузкой, когда одновременно в системе работает большое количество пользователей. Это помогло выявить узкие места в производительности, которые потребовали дальнейшей оптимизации.

### 3.7 Доработка программного обеспечения

После тестирования были внесены изменения для устранения выявленных проблем и улучшения работы системы.

Для ускорения работы системы были добавлены индексы на поля, которые часто используются в запросах (например, статусы заявок и даты). Это улучшило производительность при обработке большого количества данных.

Интерфейс был доработан для более удобной работы пользователей:

- добавлены интерактивные подсказки при вводе данных;
- улучшена система валидации, чтобы минимизировать ошибки при заполнении форм.

Некоторые маршруты были скорректированы для улучшения логики переходов. В частности, были добавлены редиректы после выполнения важных операций (например, после изменения статуса заявки). Кроме того, оптимизированы механизмы маршрутизации для ускорения обработки запросов и повышения общей производительности.

На этапе тестирования безопасности были найдены уязвимости, такие как возможность межсайтового скриптинга (XSS) и атаки с подделкой межсайтовых запросов (CSRF). Были внедрены защитные механизмы, такие как токены CSRF и экранирование входных данных, чтобы предотвратить такие атаки.

После всех доработок система прошла повторное тестирование и была готова к развертыванию в эксплуатацию.

Выводы по главе 3

Был описан процесс выбора и разработки компонентов для информационной системы управления заявками (ИСУЗ). Программное обеспечение разрабатывалось с учетом масштабируемости, производительности и удобства интеграции.

Основной язык программирования – Python, а Django использован как веб–фреймворк, благодаря встроенным инструментам и удобной системе администрирования [23]. Для хранения данных была выбрана PostgreSQL благодаря ее высокой надежности и поддержке работы с географическими данными. Для фронтенд–части использовались React и Redux, что позволило эффективно управлять состоянием приложения и создавать интерактивные интерфейсы.

Разработка базы данных сосредоточена на хранении информации о заявках и пользователях. Для улучшения производительности используются индексы и внешние ключи. Интерфейс был спроектирован с учетом адаптивности, простоты и удобства для пользователей.

Маршрутизация в Django управляет переходами между страницами и обработкой динамических данных, таких как идентификаторы заявок.

# Заключение

В ходе исследования в области разработки программного обеспечения был проведен полный цикл создания информационной системы управления заявками (ИСУЗ) аварийно–диспетчерской службы жилищно–коммунального хозяйства (ЖКХ).

Анализ требований. На первом этапе была выполнена глубокая проработка требований к системе с использованием модели FURPS+. Это позволило не только выявить функциональные потребности, но и учесть важные нефункциональные аспекты, такие как производительность, безопасность и масштабируемость системы.

Также было проведено детальное исследование аналогичных решений, что подтвердило необходимость разработки уникальной системы для ЖКХ, которая обеспечивала бы мобильную поддержку и интеграцию с государственными сервисами.

На стадии проектирования системы был выбран подход с применением гибкой методологии разработки Agile, в сочетании с фреймворком Scrum, что позволило обеспечить гибкость управления проектом и оперативную адаптацию системы к изменяющимся требованиям.

Была создана многослойная архитектура с применением паттерна MVC (Model–View–Controller), что обеспечило четкое разделение функциональности системы на отдельные слои и упростило ее поддержку и масштабирование. Разработанная модель данных, построенная на реляционной базе данных, гарантировала надежное хранение информации и эффективное управление заявками, специалистами и клиентами.

Разработка системы осуществлялась с использованием итерационной методологии Scrum, что позволяло на каждом спринте реализовывать и тестировать функциональные модули системы.

## Список используемой литературы и используемых источников

1. Акатова, Н. А. Автоматизация бизнес–процессов предприятия средствами типовых программных решений. Модуль 2 «Управление производством в 1С: ERP»: учебно–методическое пособие / Н. А. Акатова. – Москва: МИСИС, 2020. – 262 с. – ISBN 978–5–907227–15–6. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/178084 (дата обращения: 14.09.2024). – Режим доступа: для авториз. пользователей.

2. Алпатов, А. Н. Архитектура, проектирование и разработка программных средств: учебное пособие / А. Н. Алпатов, И. Е. Рогов. – Москва: РТУ МИРЭА, 2023. – 120 с. – ISBN 978–5–7339–1972–0. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/386189 (дата обращения: 09.09.2024). – Режим доступа: для авториз. пользователей.

3. Баланов, А. Н. Внедрение методологий в IT: Agile, Scrum и другие: учебное пособие для вузов / А. Н. Баланов. – Санкт–Петербург: Лань, 2024. – 188 с. – ISBN 978–5–507–48919–0. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/401123 (дата обращения: 11.09.2024). – Режим доступа: для авториз. пользователей.

4. Бабушкин, В. М. Разработка защищенных программных средств информатизации производственных процессов предприятия: учебное пособие / В. М. Бабушкин. – Казань: КНИТУ–КАИ, 2020. – 256 с. – ISBN 978–5–7579–2463–2. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/193486 (дата обращения: 11.09.2024). – Режим доступа: для авториз. пользователей.

5. Богомолов, А. В. Проектирование информационных систем для бакалавров, обучающихся по направлению подготовки 09.03.03 «Прикладная информатика»: методические рекомендации / А. В. Богомолов, Э. А. Игнатьева, К. Н. Фадеева. – Чебоксары: ЧГПУ им. И. Я. Яковлева, 2022. – 48

с. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/354065 (дата обращения: 08.09.2024). – Режим доступа: для авториз. пользователей.

6. Брежнев, Р. В. Методы и средства проектирования информационных систем и технологий: учебное пособие / Р. В. Брежнев. – Красноярск: СФУ, 2021. – 216 с. – ISBN 978–5–7638–4416–0. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/181656 (дата обращения: 15.09.2024). – Режим доступа: для авториз. пользователей.

7. Визуальное программирование: учебное пособие / А. А. Тюгашев. – Самара: СамГУПС, 2020. – 147 с. – ISBN 978–5–98941–325–6. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/161313 (дата обращения: 10.09.2024). – Режим доступа: для авториз. пользователей.

8. Котлинский, С. В. Разработка моделей предметной области автоматизации: учебник для вузов / С. В. Котлинский. – Санкт–Петербург: Лань, 2021. – 412 с. – ISBN 978–5–8114–8035–7. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/183204 (дата обращения: 06.09.2024). – Режим доступа: для авториз. пользователей.

9. Нафикова, А. Р. Объектно–ориентированный анализ и проектирование программного обеспечения на языке UML: учебное пособие / А. Р. Нафикова. – Уфа: БГПУ имени М. Акмуллы, 2022. – 118 с. – ISBN 978–5–907475–48–9. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/219221 (дата обращения: 14.09.2024). – Режим доступа: для авториз. пользователей.

10. Остроух, А. В. Проектирование информационных систем: монография / А. В. Остроух, Н. Е. Суркова. – 2–е изд., стер. – Санкт–Петербург: Лань, 2021. – 164 с. – ISBN 978–5–8114–8377–8. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/175513 (дата обращения: 03.09.2024). – Режим доступа: для авториз. пользователей.

11. Петрова, А. Н. Технологии WEB: учебное пособие / А. Н. Петрова. –

Комсомольск–на–Амуре: КНАГУ, 2018. – 176 с. – ISBN 978–5–7765–1360–2. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/151717 (дата обращения: 03.09.2024). – Режим доступа: для авториз. пользователей.

12. Плотникова, С. Н. 1С: Управление производственным предприятием: учебное пособие / С. Н. Плотникова, Л. А. Козлова. – Киров: Вятская ГСХА, 2012. – 91 с. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/129653 (дата обращения: 12.09.2024). – Режим доступа: для авториз. пользователей.

13. Попова, Ю. Б. Тестирование и отладка программного обеспечения: учебное пособие / Ю. Б. Попова. – Минск: БНТУ, 2020. – 66 с. – ISBN 978–985–583–056–7. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/248642 (дата обращения: 17.09.2024). – Режим доступа: для авториз. пользователей.

14. Перепелица, Ф. А. Эффективная разработка веб–сайтов. Bootstrap: учебное пособие / Ф. А. Перепелица. – Санкт–Петербург: НИУ ИТМО, 2015. – 71 с. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/91557 (дата обращения: 09.09.2024). – Режим доступа: для авториз. пользователей.

15. Проектные методологии управления. Agile и Scrum: учебное пособие / Ю. Д. Агеев, Ю. А. Кавин, И. С. Павловский. – Москва: Аспект Пресс, 2020. – 160 с. – ISBN 978–5–7567–0982–7. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/169666 (дата обращения: 10.09.2024). – Режим доступа: для авториз. пользователей.

16. Сборник информационных систем и компьютерных технологий: учебное пособие. – Таганрог: ТГТУ, 2021. – 176 с. – ISBN 978–5–903717–88–3. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/249264 (дата обращения: 04.09.2024). – Режим доступа: для авториз. пользователей.

17. Тебайкина, Н. И. Применение концепции ITSM при вводе в действие

информационных систем: учебное пособие / Н. И. Тебайкина. – Екатеринбург: УрФУ, 2014, 72 с. – ISBN 978–5–7996–1249–8. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/98268 (дата обращения: 31.08.2024). – Режим доступа: для авториз. пользователей.

18. Тихонова, Н. А. Проектирование информационной системы: учебное пособие / Н. А. Тихонова. – Омск: ОмГУПС, 2021. – 37 с. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/190259 (дата обращения: 13.09.2024). – Режим доступа: для авториз. пользователей.

19. Цифровизация производства: учебно–методическое пособие / И. Н. Хаймович, Е. Г. Демьяненко, С. Г. Симагина, Е. А. Мешкова. – Самара: Самарский университет, 2023, 168 с. – ISBN 978–5–7883–1892–9. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/406676 (дата обращения: 09.09.2024). – Режим доступа: для авториз. пользователей.

20. Янцев, В. В. Web–программирование на Python : учебное пособие/ В. В. Янцев. – 3–е изд., перераб. – Санкт–Петербург: Лань, 2024. – 180 с. – ISBN 978–5–507–48364–8. – Текст: электронный // Лань: электронно–библиотечная система. – URL: https://e.lanbook.com/book/392993 (дата обращения: 08.09.2024). – Режим доступа: для авториз. пользователей.

21. Django documentation. URL: https://docs.djangoproject.com (дата обращения: 14.09.2024).

22. GitHub Docs. URL: https://docs.github.com/en (дата обращения: 14.09.2024).

23. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/current/ (дата обращения: 14.09.2024).

24. React documentation. URL: https://react.dev/learn (дата обращения: 21.09.2024).

25. Redux introduction. URL: https://react–redux.js.org/introduction/getting–started (дата обращения: 21.09.2024).