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

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки, специальности)

Мобильные и сетевые технологии
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему: «Разработка и администрирование системы учета и управления заявками в службе технической поддержки организации»

Студент Е.С. Шеянов
(И.О. Фамилия) (личная подпись)

Руководитель к.э.н., доцент Т.А. Раченко
(ученая степень, звание, И.О. Фамилия)

Консультант к.п.н., доцент С.А. Гудкова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2024

Аннотация

Тема выпускной квалификационной работы – «Разработка и администрирование системы учета и управления заявками в службе технической поддержки организации».

Актуальность темы ВКР обусловлена развитием технологий и их интеграцией в управление бизнес-процессами в организациях различных сфер деятельности.

Объектом исследования является деятельность служба технической поддержки отдела информационных технологий организации ООО «НПО ДИЗ».

Предмет исследования – автоматизация процессов учёта и управления заявками в службе технической поддержки отдела информационных технологий организации.

Целью работы является разработка мобильного приложения, позволяющего автоматизировать процессы учёта и управления заявок в службе технической поддержки.

Структурно работа представлена введением, тремя главами, заключением, списком литературы и источников, а также четырьмя приложениями.

В первой главе работы описана организационная структура организации ООО «НПО ДИЗ», выполнен анализ деятельности службы технической поддержки отдела информационных технологий с использованием методологий IDEF0, описана сущность задачи на разработку приложения и произведен анализ уже существующих аналогов подобных систем.

Вторая глава посвящена логическому моделированию разрабатываемого мобильного приложения, проектированию базы данных и формулировке требований к аппаратному обеспечению для развертывания приложения на серверах и мобильных устройствах организации.

В третьей главе рассматриваются выбранные технологии разработки, описывается разработанное мобильное приложение, его функции, заполняемые формы, передаваемые и получаемые данные, а также процесс администрирования и авторизации пользователей.

Работа содержит: 72 страницы, 32 рисунка, 4 таблицы, 4 приложения и список из 25 используемых источников.

Abstract

The title of the graduation work is "Development and Administration of a System for Recording and Managing Requests in the Technical Support Service of the Organization."

The senior paper consists of introduction, three parts, a conclusion, tables, list of references including foreign sources.

The aim of this work is driven by the development of technologies and their integration into business process management in companies and enterprises across various fields of activity.

The object of the study is the activity of the technical support service within the information technology department of the organization "NPO DIZ" LLC.

The subject of the study is the automation of the process of recording and managing requests in the technical support service.

The purpose of this work is to develop a mobile application that automates the processes of recording and managing requests in the technical support service of the organization.

To develop a mobile application for automating technical support service processes, the organization’s business processes were analyzed, and the object for automation was identified and described. The development was conducted in the Java programming language using the Spring framework.

In the final part of the work, the developed mobile application was tested and demonstrated.

In conclusion, I would like to emphasize that the developed mobile application will improve the productivity of the technical support service in the information technology department of the organization "NPO DIZ" LLC. The application has allowed us to speed up the process of recording and managing requests.

Оглавление

Введение 6
Глава 1 Технико-экономическая характеристика предприятия 8
1.1 Описание деятельности компании ООО «НПО ДИЗ» 8
1.2 Концептуальное проектирование предметной области 10
1.3 Анализ существующих разработок для автоматизации комплекса задач 18
1.4 Разработка требований к приложению автоматизации учёта и управления заявками 20
1.5 Постановка задачи на разработку мобильного приложения 24
Глава 2 Проектирование мобильного приложения для службы технической поддержки 26
2.1 Выбор платформы реализации мобильного приложения 26
2.2 Выбор средств разработки 28
2.3 Проектирование концептуальной модели базы данных 31
2.4 Архитектура программного продукта 38
Глава 3 Реализация мобильного приложения 43
3.1 Описание разработанного программного решения 43
3.2 Реализация и обзор разработанного мобильного приложения 47
3.3 Реализация администрирования системы учета 57
3.4 Тестирование мобильного приложения 63
3.5 Развертывание и внедрение мобильного приложения 66
Заключение 68
Список используемой литературы и используемых источников 70
Приложение А Управление заявками 73
Приложение Б Сущность «пользователь» 76
Приложение В Конфигурация 78
Приложение Г Класс авторизации и регистрации 79

Введение

В современном мире трудно представить компанию или предприятие, не использующее программное обеспечение для оптимизации своих бизнес-процессов. Программное обеспечение способствует улучшению конкретных рабочих процессов внутри компании.

Тема данной выпускной квалификационной работы: «Разработка и администрирование системы учета и управления заявками в службе технической поддержки организации».

Объектом исследования является деятельность службы технической поддержки отдела информационных технологий организации ООО «НПО ДИЗ».

Предмет исследования – автоматизация процессов учёта и управления заявками в службе технической поддержки отдела информационных технологий организации.

Служба технической поддержки в отделе информационных технологий – важное структурное подразделение, от эффективности работы которого зависит функционирование всего предприятия. Таким образом разработка мобильного приложения для учёта и управления заявками является актуальным и привёт к росту производительности процессов внутри организации.

Интеграция мобильного приложения в процессы службы технической поддержки информационного отдела существенно ускорит процесс учета и управления заявками и уменьшит вероятность ошибок при работе.

Чтобы достичь цели, необходимо рассмотреть и выполнить следующие задачи:

- изучить деятельность организации ООО «НПО ДИЗ», его организационную структуру и объект автоматизации;
- проанализировать существующие программные решения, которые можно внедрить в систему предприятия;

– обосновать необходимость разработки и администрирования системы учёта и управления заявками в службе технической поддержки организации;
– смоделировать концептуальные модели бизнес-процессов «как есть» и «как будет» для организации;
– спроектировать разрабатываемое приложение;
– разработать модель данных;
– спроектировать базу данных приложения;
– провести демонстрационное тестирование разработанного мобильного приложения.

Первая глава описывает организацию ООО «НПО ДИЗ», характеризует объект автоматизации – службу технической поддержки отдела информационных технологий в соответствии с темой ВКР. В главе проводится концептуальное проектирование бизнес-процессов предприятия и сравниваются аналоги уже реализованных программных решений. Результаты сравнения обосновывают необходимость разработки собственного мобильного приложения для службы технической поддержки организации.

Во второй главе представлено логическое проектирование разрабатываемого мобильного приложения. Описывается архитектура приложения, модель данных, требования к целостности данных и серверному обеспечению для развертывания приложения на серверах предприятия и мобильных устройствах сотрудников.

Третья, заключительная глава, описывает выбранные технологии, использованные при разработке. Демонстрируется работа приложения, его функции, а также передаваемые и получаемые данные. Также описывается администрирование системы, тестирование и интеграция приложения.

Глава 1 Технико-экономическая характеристика предприятия

1.1 Описание деятельности компании ООО «НПО ДИЗ»

Полное название организации ООО «Научно-производственное объединение Димитровградский инструментальный завод». Предприятие находится на территории Ульяновской области, по адресу г. Димитровград, ул. Куйбышева, 34.

Организация специализируется на проектировании, изготовлении оснастки, штампов и нестандартного оборудования, а также оказывает услуги по механической обработке деталей и термообработке металла. С оборудованием с ЧПУ и сертифицированной системой менеджмента качества, предприятие гарантирует высокое качество продукции. Особенностью является возможность проведения испытаний штампов и изготовление мелких серий деталей.

Предприятие обладает широким спектром услуг, включая механическую и электроэрозионную обработку, термическую обработку, а также изготовление нестандартного оборудования по чертежам заказчика. С опытными сотрудниками и современным станочным парком, предприятие готово удовлетворить запросы клиентов в различных отраслях промышленности.

Рассмотрим организационную структуру организации, представленную на рисунке 1.

Предприятие разделено на функциональные отделы:

- отдел информационных технологий,
- финансовый отдел,
- отдел производства,
- отдел проектирования,
- юридический отдел.

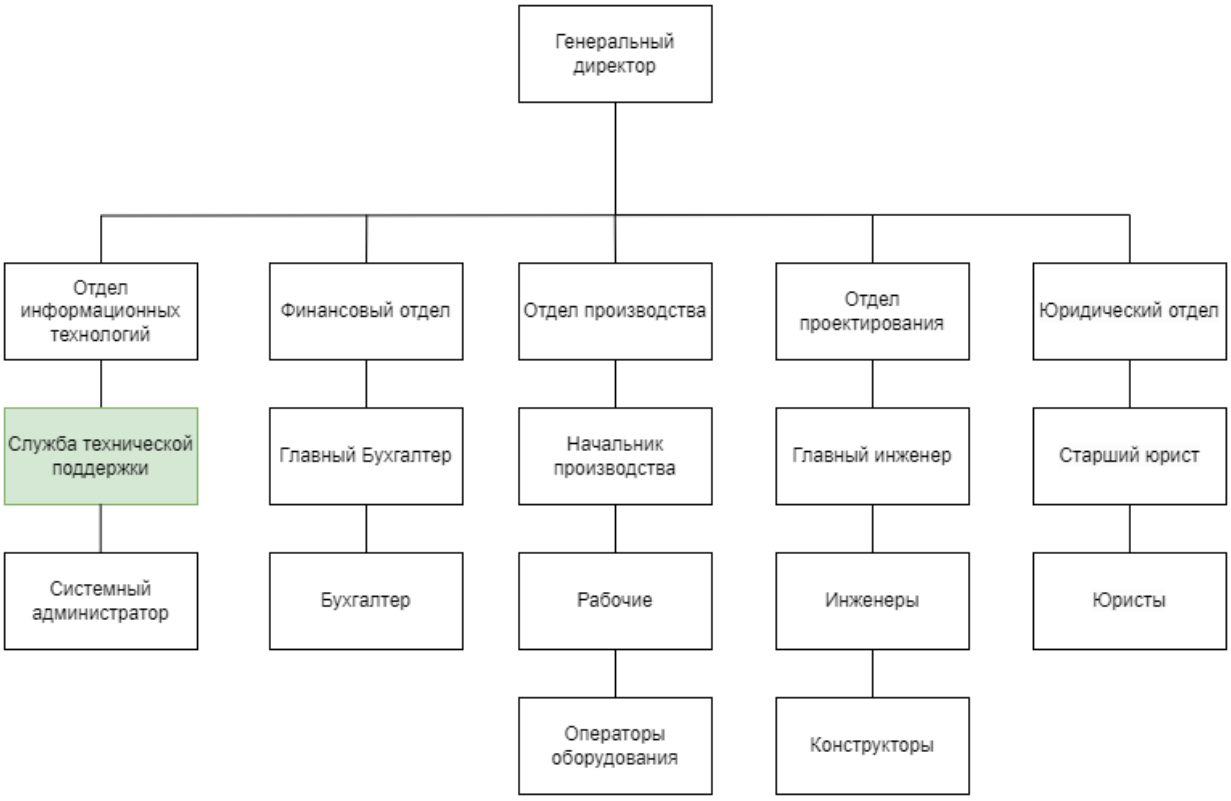

Рисунок 1 – Организационная структура предприятия ООО «НПО ДИЗ»

Отдел информационных технологий – этот отдел занимается установкой и обслуживанием корпоративного оборудования, а также технической поддержкой функциональных информационных систем.

Отдел кадров – отвечает за подбор персонала, разработку рабочих графиков и соблюдение трудового законодательства РФ.

Финансовый отдел – управляет финансами организации.

Юридический отдел – обеспечивает правовую защиту интересов компании, соблюдение законодательства и предоставляет юридические консультации руководству.

Отдел производства – организует все аспекты производственного процесса, от планирования до непосредственного изготовления продукции. Задачи включают контроль за качеством продукции, обеспечение технического обслуживания оборудования и оптимизацию производственных мощностей.

Отдел проектирования – специализируется на разработке новых продуктов и усовершенствовании существующих. Отдел занимается всем, начиная от концепции и заканчивая технической документацией, обеспечивая инновационность и соответствие продукции современным требованиям и стандартам.

Таким образом, структура управления компанией является функциональной, каждый отдел выполняет специфические задачи в рамках своих компетенций.

В настоящее время на предприятии акцент сделан на различных процессах обработки деталей, а также на изготовлении штампов и нестандартного оборудования. В случае возникновения проблем на производственной линии, единственным доступным способом обработки и оформления заявки является телефонный звонок сотрудником, заметившим неисправность, специалисту технической поддержки. Специалист принимает заявку, исправляет возникшую неисправность, а затем регистрирует её в журнале Excel. Однако из-за отсутствия единой системы управления процессами могут возникать проблемы с координацией и контролем, что, в свою очередь, может приводить к ошибкам в отчетности по выполненным задачам или к неполной информации о выполненной работе.

1.2 Концептуальное проектирование предметной области

Структурная и функциональная схема работы отдела информационных технологий «как сейчас» и «как будет в будущем» и описание этих схем.

В качестве объекта автоматизации выступает: деятельность службы технической поддержки в организации.

При построении моделей бизнес-процессов чаще всего используют одну из следующих нотаций:

- Event-driven Process Chain (EPC);
- Business Process Management Notation (BPMN);

– Integration Definition For Function Modeling (IDEF0);

«EPC (event-driven process chain, событийная цепочка процессов) – графический стандарт моделирования бизнес-процессов, нотация класса WorkFlow. EPC-метод был разработан Августом-Вильгельмом Шеером в начале 1990-х годов при создании ARIS. Отличительной особенностью является обязательное чередование событий и функций. На диаграммах EPC можно увидеть последовательность решений, функции, события, документы и статусы и другие элементы бизнес-процесса.» [8] Данная нотация также не является наилучшим выбором для моделирования бизнес-процессов организации ООО «НПО ДИЗ».

«Нотация BPMN (Business Process Management Notation, нотация моделирования бизнес-процессов) – нотация класса WorkFlow, один из наиболее распространенных методов описания бизнес-процессов на сегодня. В нотации BPMN используется базовый набор интуитивно понятных элементов, которые позволяют определять сложные семантические конструкции.» [8] Данная нотация не подходит для моделирования бизнес-процессов организации ООО «НПО ДИЗ».

IDEF0 – «это методология графического моделирования процессов, используемая для реализации систем и разработки программного обеспечения. Эти методы используются в функциональном моделировании данных, симуляции, объектно-ориентированном анализе и приобретении знаний.» [24].

Для реализации бизнес-процессов в отделе была выбрана нотация IDEF0, поскольку данная нотация позволяет продемонстрировать подробную реализацию моделируемого процесса.

При моделировании схемы бизнес-процессов для учёта и управления заявок службой технической поддержки в отделе информационных технологий, следует применить следующие модели:

– как есть – «это описание процессов в том состоянии, в котором они находятся в данный момент.» [5];

– как будет – «это представление целевого состояния.» [5].

С использованием программы Ramus Educational была создана концептуальная модель бизнес-процессов технической поддержки организации, основанная на методологии IDEF0, которая представлена на рисунке 2.

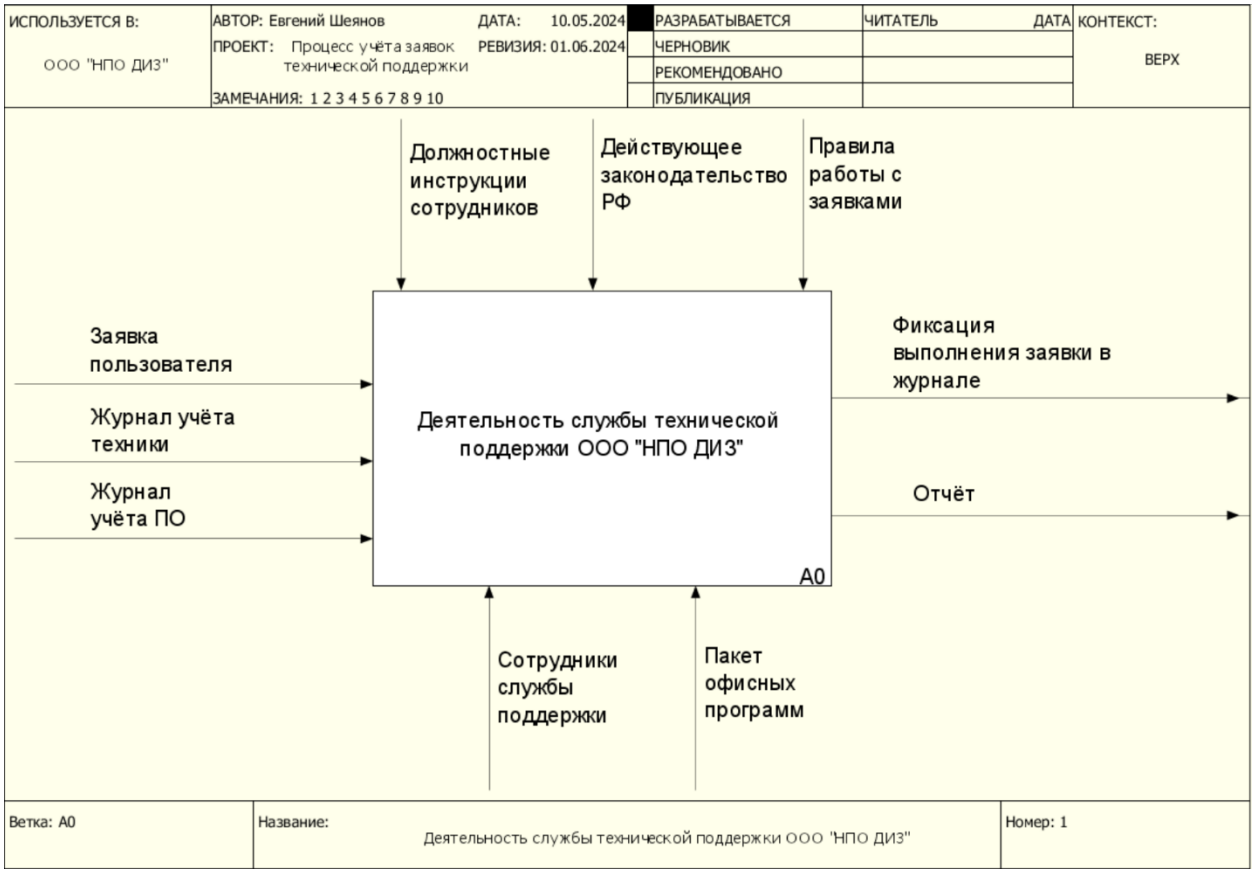

Рисунок 2 - Диаграмма (IDEF0) верхнего уровня «Как есть» процесса деятельности службы технической поддержки ООО «НПО ДИЗ»

Входными потоки являются:

– заявка пользователя;

– журнал учёта техники;

– журнал учёта ПО.

Потоками управления являются:

– должностные инструкции;

– действующее законодательство РФ;

– правила работы с заявками.

Потоками механизмов системы являются:

– сотрудники службы поддержки – персонал, занимающийся принятием, обработкой и завершением заявок;

– пакет офисных программ – набор программного обеспечения MS Excel, используемого сотрудниками службы технической поддержки для выполнения их задач.

Выходными данными являются:

– фиксация выполнения заявки в журнале – документирование выполнения и статуса заявки;

– отчёт – отчёт о проделанной работе.

На рисунке 3 представлена декомпозиция концептуальной модели 0-го уровня.

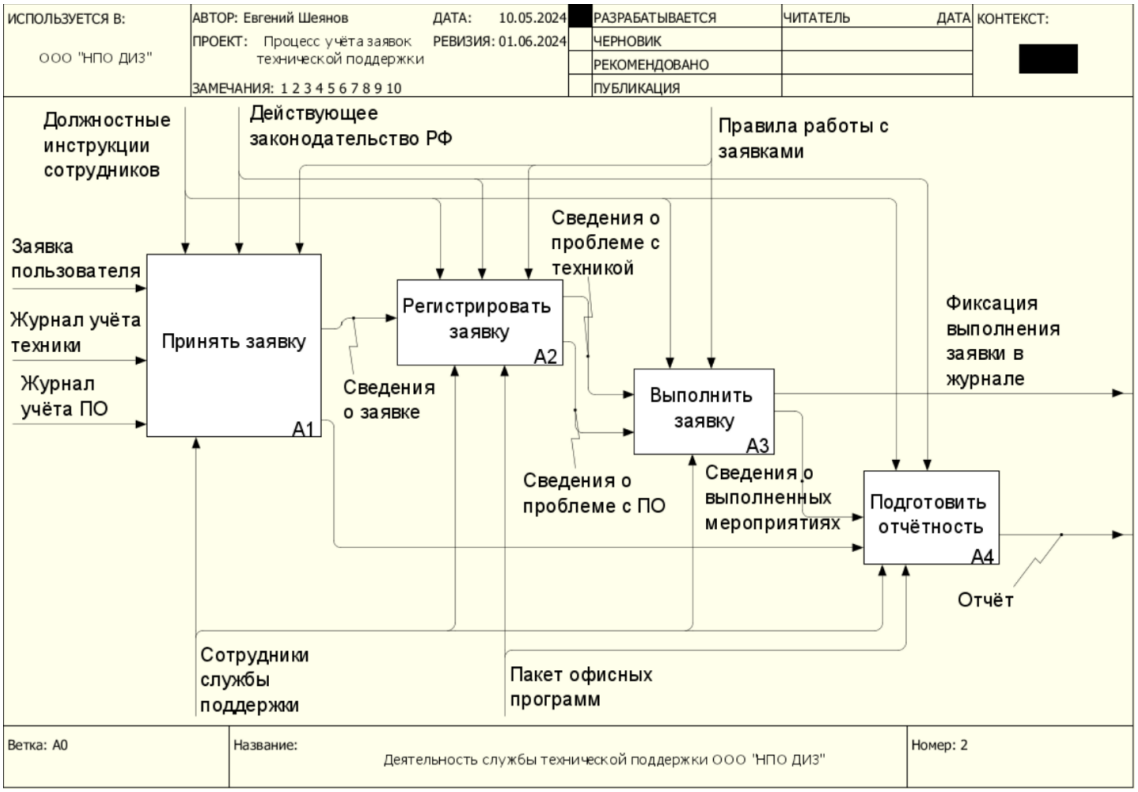

Рисунок 3 – Декомпозиция диаграммы A0 процесса деятельности службы технической поддержки ООО «НПО ДИЗ»

Декомпозиции состоит из четырёх блоков:

– принять заявку;

– регистрировать заявку;

– выполнить заявку;

– подготовить отчётность.

Для более глубокого анализа процесса рассмотрим декомпозиция блока «Выполнить заявку», изображенную на рисунке 4 более подробно.

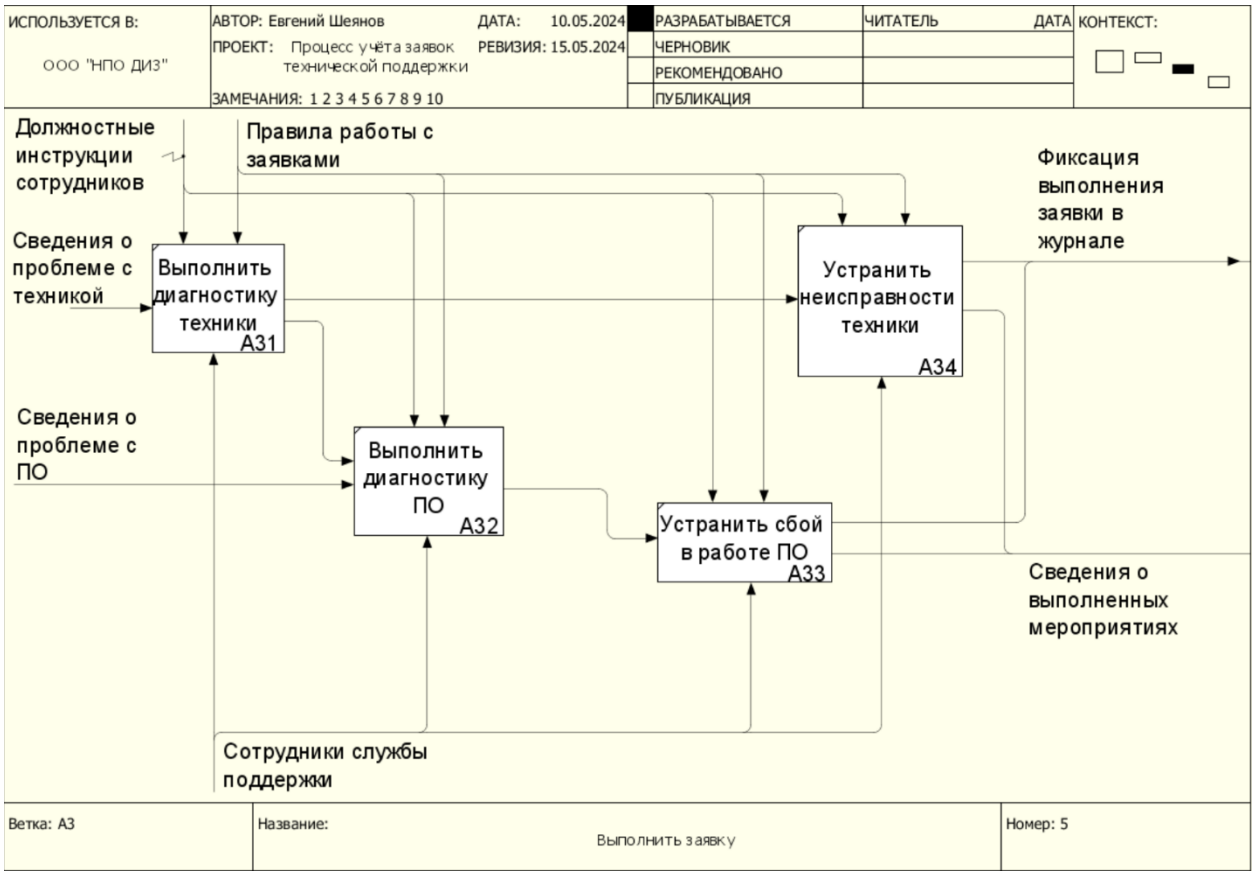

Рисунок 4 – Декомпозиция блока А3 «Выполнить заявку»

Декомпозиция бизнес-процесса «выполнить заявку» состоит из четырёх блоков:

– выполнить диагностику техники;

– выполнить диагностику программного обеспечения;

– устранить сбой в работе программного обеспечения;
– устранить неисправности техники.

Анализ модели «как есть» позволил выявить недостатки существующих бизнес-процессов. Использование телефонной связи пользователем для того, чтобы подать заявку в устной форме сотруднику службы поддержки. Также сотрудник заносит данную заявку в журнал в программе MS Excel, затем либо сам направляется к пользователю, либо сообщает компетентному лицу о новой заявке и происходит устранение неисправности. После решения проблемы сотрудник записывает информацию о проделанной работе в журнал. Недостатки включают высокую трудоемкость, вероятность ошибок, высокие затраты времени на регистрацию и использование Excel для ведения журнала.

Модель «как есть» используется в качестве основы для создания модели бизнес-процессов предприятия «как будет». Это способствует дальнейшему совершенствованию бизнес-процессов организации.

На рисунке 5 приведена контекстная диаграмма «как будет» деятельности предприятия.

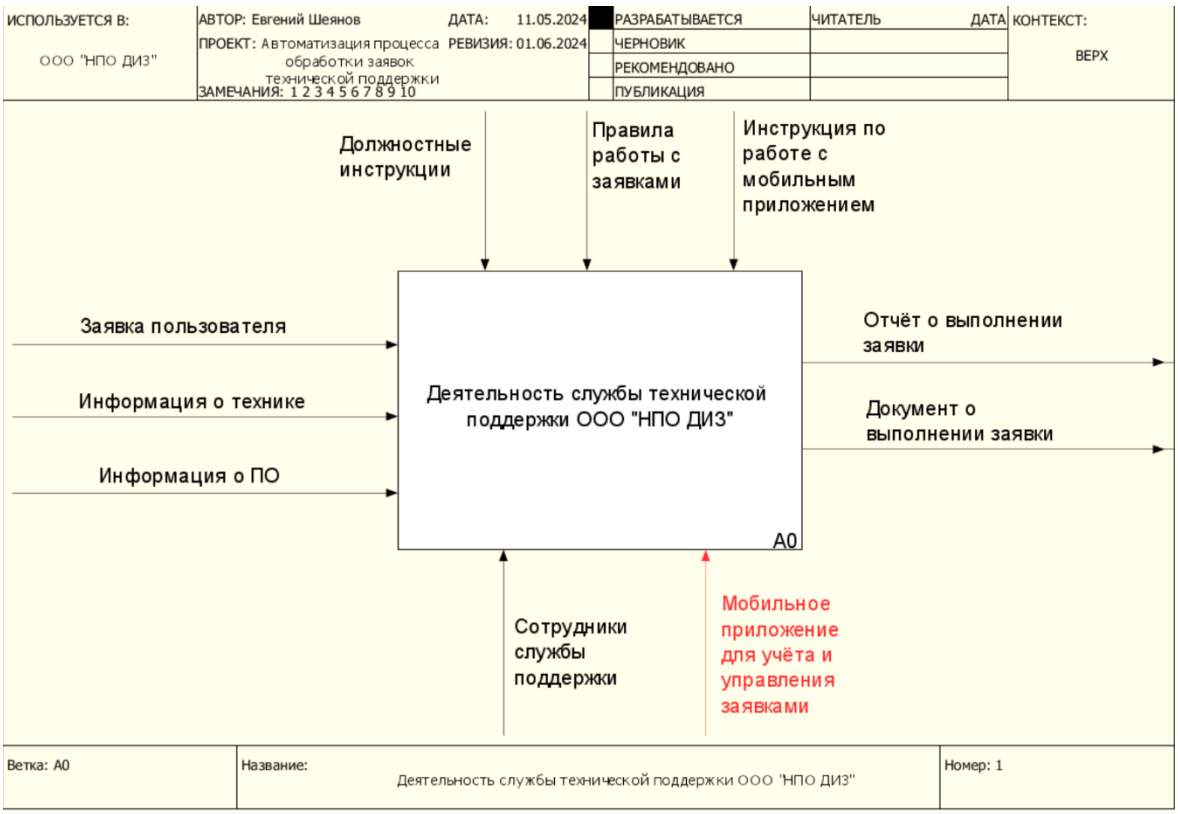

Рисунок 5 – Декомпозиция «как будет» контекстной диаграммы А-0

Автоматизация деятельности предприятия включает внедрение мобильного приложения для учета и управления заявками в работу службы поддержки.

На рисунке 6 представлена декомпозиция диаграммы А0 «как будет», иллюстрирующая деятельность организации.

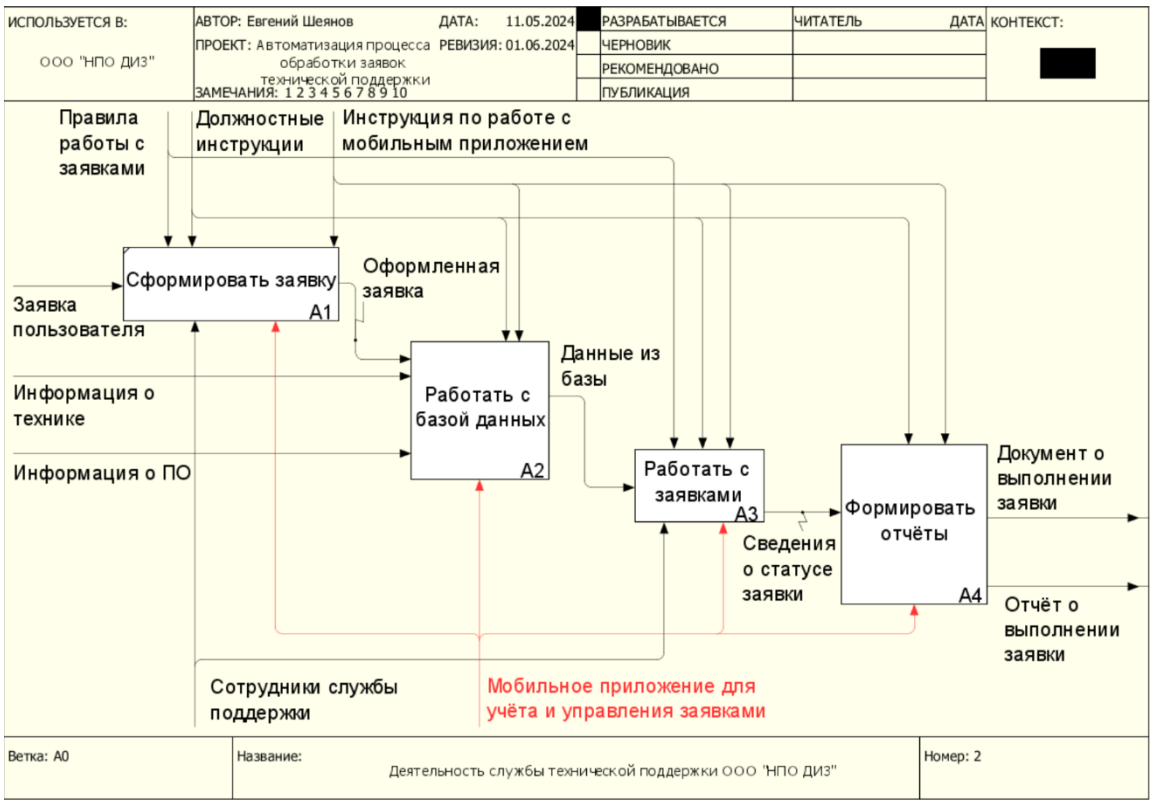

Рисунок 6 – Декомпозиция «как будет» контекстной диаграммы А-0

Декомпозиция «как будет» контекстной диаграммы А0 включает четыре процесса:

– сформировать заявку,

– работать с базой данных,

– работать с заявками,

– формировать отчёты.

На рисунке 6 можно увидеть, что все процессы работы с заявками, будут автоматизированы благодаря внедрению мобильного приложения для работы с заявками.

Разрабатываемое приложение позволит сократить время, затрачиваемое службой поддержки на формирование и обработку заявок и позволит быстро выполнять возникающие проблемы.

Построенная диаграмма бизнес-процессов деятельности организации «как будет», является обоснованием для технического задания на разработку

и администрирования системы учёта и управления для службы технической поддержки организации ООО «НПО ДИЗ».

1.3 Анализ существующих разработок для автоматизации комплекса задач

Существуют различные готовые решения, которые в настоящее время разработаны для автоматизации процесса учета и управления заявками, предлагая необходимый функционал. Среди них есть такие как ServiceNow Mobile, Jira Service Management и Freshservice. Рассмотрим каждый продукт отдельно.

ServiceNow является лидером в области управления рабочими процессами и обслуживания. Его мобильное приложение ServiceNow Mobile предоставляет широкий набор функций и интеграций, а также обладает высокой степенью масштабируемости и гибкости. Крупные предприятия часто предпочитают ServiceNow из-за его возможностей по автоматизации процессов и управлению сложными системами. Пользователи могут создавать, просматривать и обновлять заявки, отслеживать статус запросов, взаимодействовать с агентами поддержки и многое другое.

Jira Service Management также пользуется значительной популярностью, особенно в разработке ПО и IT-отраслях, благодаря своей интеграции с другими продуктами Atlassian, такими как Jira Software и Confluence. Их мобильное приложение позволяет агентам и клиентам создавать и отслеживать заявки технической поддержки, а также управлять процессами обработки запросов. Пользователи могут получать уведомления о важных событиях и оперативно реагировать на изменения в статусе заявок.

Freshservice пользуется популярностью благодаря своей простоте в использовании и доступности для малых и средних предприятий. Пользователи могут создавать, просматривать и обновлять заявки, управлять расписанием задач, общаться с агентами поддержки и выполнять множество

других действий. Приложение отличается интуитивно понятным интерфейсом и хорошо интегрируется с другими инструментами.

Конечно, существует множество систем учёта и управления заявками в виде мобильных приложений, однако, рассматривая эти приложения, можно проанализировать преимущества и недостатки каждого приложения. Для этого используем методику FURPS. Данная методика предусматривает пять категорий: Функциональность (Functionality), удобство использования (Usability), надёжность (Reliability), производительность (Perfomance), поддерживаемость (Supportabillity). Для большей точности шестым критерием оценим стоимость.

В таблице 1 отображён анализ существующих программных решений.

Таблица 1 - Сравнительная характеристика готовых решений

Категории FURPS	ServiceNow Mobile	Jira Service Management	Freshservice
Функциональность (Functionality)	Очень высокий уровень функциональности с возможностями интеграции и настройки.	Хорошая функциональность, в интеграции с другими продуктами компании.	Достаточный уровень функциональности для средних и малых предприятий.
Удобство использования (Usability)	Отличное мобильное приложение с высокой оценкой.	Хорошее удобство использования, интуитивно понятные интерфейсы.	Приемлемое удобство использования, но не слишком интуитивное.
Надежность (Reliability)	Высокая надежность с сильной безопасностью и стабильностью.	Надежное приложение с хорошими функциями безопасности.	Надежность хорошая, но могут быть вопросы с устойчивостью при больших загрузках.
Производительность (Performance)	Отличная производительность и оптимизация ресурсов.	Хорошая производительность, но могут возникнуть задержки при интенсивном использовании.	Достаточная производительность для стандартных операций.

Продолжение Таблицы 1

Категории FURPS	ServiceNow Mobile	Jira Service Management	Freshservice
Поддерживаемость (Supportability)	Превосходная поддерживаемость и легкость обновлений и настройки.	Хорошая поддерживаемость и документация, легкость в ведении изменений.	Поддерживаемость и обновления на уровне, но может потребоваться дополнительная настройка.
Стоимость	Высокая стоимость	Очень высокая стоимость	Средняя стоимость

Проанализировав существующие программные решения, было установлено, что ни одно из них не способно полностью удовлетворить потребности предприятия. К тому же, основным недостатком подобных систем является их высокая стоимость, которая может значительно варьироваться в зависимости от реализации.

Результаты анализа послужили основанием для начала разработки собственного приложения для учета и управления заявками, которое будет включать все необходимые для организации функции.

1.4 Разработка требований к приложению автоматизации учёта и управления заявками

На основе анализа бизнес-процессов организации и сопоставления различных решений для учета и управления заявками были определены требования, используя классификацию FURPS:

Функциональные требования включают:

– добавление новых заявок через мобильное приложение с указанием всех деталей. Это позволит пользователям быстро и удобно сообщать о проблемах, предоставляя полную информацию;

– выполнение заявок и изменение их статуса. Это позволит исполнителям эффективно управлять заявками и поддерживать пользователей в курсе процесса решения;
– разделение на типы пользователей: обычные пользователи, администраторы, исполнители. Это обеспечит различные уровни доступа и функциональность в соответствии с ролями пользователей;
– просмотр истории заявок в личном кабинете. Это предоставит пользователям возможность отслеживать историю своих заявок для лучшего понимания процесса обслуживания.

Удобство использования включает в себя: интуитивно понятный интерфейс для пользователей. Это обеспечит простоту в использовании системы для всех категорий пользователей, сокращая время обучения и повышая удовлетворенность.

Надёжность включает:

– стабильная работа приложения даже при большой нагрузке. Это обеспечит непрерывное обслуживание пользователей даже в условиях повышенной активности;
– высокий уровень безопасности и конфиденциальности данных. Это гарантирует защиту личной информации пользователей и предотвращает утечки данных;
– возможность сохранения данных в локальном хранилище при отсутствии интернет-соединения. Это обеспечит сохранность данных и возможность работы в автономном режиме.

Производительность включает в себя быструю загрузка данных и оперативное добавление заявок. Это обеспечит быстрый доступ к информации и повысит эффективность работы пользователей.

Поддерживаемость включает:

– регулярные обновления для исправления ошибок и добавления новых функций. Это обеспечит постоянное улучшение и совершенствование системы;
– обработка возможных ошибок связи с сервером. Это поможет минимизировать негативные последствия возможных сбоев в работе приложения;
– предоставление обратной связи пользователю о статусе отправки или получения данных. Это позволит пользователям быть в курсе текущего состояния своих запросов и оперативно реагировать на изменения.

Вышеописанные функции представим в виде диаграммы вариантов использования, представленной на рисунке 7.

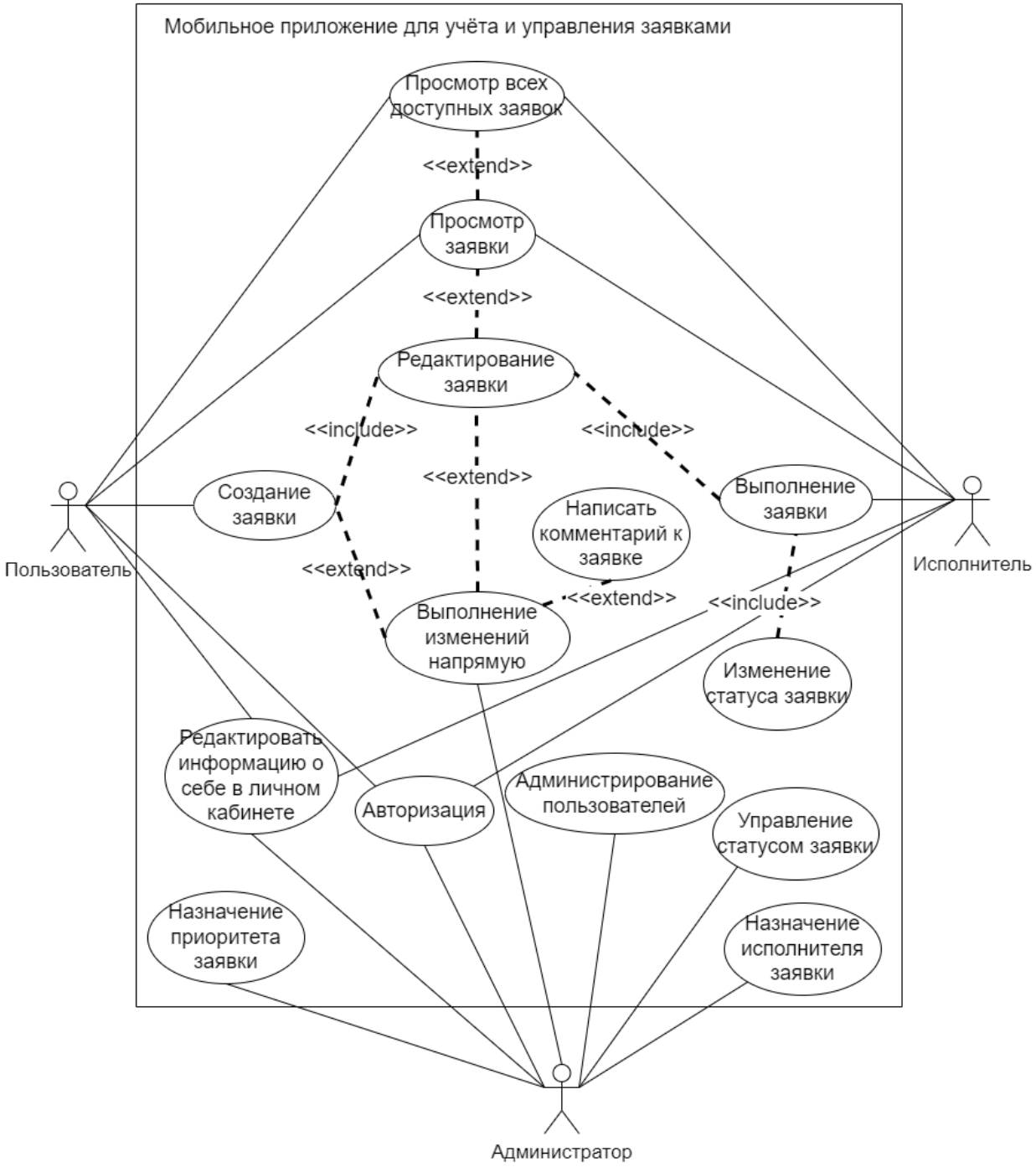

Рисунок 7 – Диаграмма прецедентов

Диаграмма прецедентов является одной из ключевых диаграмм UML. На диаграмме изображены три актёра: пользователь, администратор и исполнитель. Доступ имеют только лица с паролем и логином от проектируемой системы.

После авторизации пользователю доступны: просмотр всех доступных заявок, просмотр конкретной заявки, создание новой заявки, и возможность отредактировать информацию о себе в личном кабинете.

Исполнитель после процедуры авторизации имеет следующие возможности: выполнение назначенных ему заявок, изменение статуса заявки при выполнении, редактировать информацию о себе в профиле.

Администратор после авторизации получает допуск к следующим функциям: редактировать информацию о себе в личном кабинете, администрирование пользователей (регистрация новых пользователей), управление статусом, приоритетом заявки, назначение исполнителя заявки, а также выполнение изменений напрямую (в том числе просмотр и редактирование всех доступных заявок и каждой отдельной заявки).

1.5 Постановка задачи на разработку мобильного приложения

Выявлена потребность в разработке системы, которая бы позволила автоматизировать и упростить взаимодействие сотрудников смежных областей и технической поддержке организации.

Требуется разработать мобильное приложение для учёта и управления заявками в службе технической поддержки организации, которое обеспечит простоту и удобство в отправке и отслеживании заявок сотрудниками, а также упростит процесс управления заявками администратору и выполнение заявок исполнителям. Приложение должно предоставлять простой интерфейс для подачи заявок с мобильных устройств, позволяя пользователям указывать тему и детальное описание проблемы. Отправив заявку, пользователи смогут увидеть изменение статуса заявки с «новая» на статус «в работе». Необходима возможность просматривать свои поданные заявки и редактировать информацию о себе через личный профиль.

Для исполнителей необходимо разработать интерфейс, позволяющий просматривать все назначенные им заявки и обновлять их статусы, после их выполнения.

Администратор должен иметь возможность видеть интерфейс, позволяющий назначать заявки исполнителям, просматривать все поступающие заявки, управлять статусом и приоритетом заявок, редактировать заявку напрямую при необходимости, добавлять комментарии для исполнителей, а также администрировать пользователей.

Приложение должно ориентироваться на архитектурный стиль REST. «REST, или Representational State Transfer, – это архитектурный стиль, обеспечивающий стандарты между компьютерными системами в сети, упрощающий взаимодействие систем друг с другом. REST-совместимые системы, часто называемые системами RESTful, характеризуются тем, что они не сохраняют состояние и разделяют задачи клиента и сервера.» [25]

Все требования нацелены на то чтобы повысить эффективность сотрудников в организации в целом так и службу технической поддержки в частности.

Выводы по главе 1.

Описаны род занятия и организационная структура предприятия ООО «НПО ДИЗ». Проведён сравнительный анализ существующих коммерческих решений, была поставлена задача на разработку мобильного приложения для учёта и управления заявками в службе технической поддержки организации.

Глава 2 Проектирование мобильного приложения для службы технической поддержки

2.1 Выбор платформы реализации мобильного приложения

При создании мобильного приложения стоит уделить внимание выбору будущей платформы. Так как на рынке сейчас доступно множество вариантов мобильных ОС, стоит внимательно проанализировать и изучить каждый из них.

На рисунке 8 изображен сравнительный график популярности мобильных ОС.

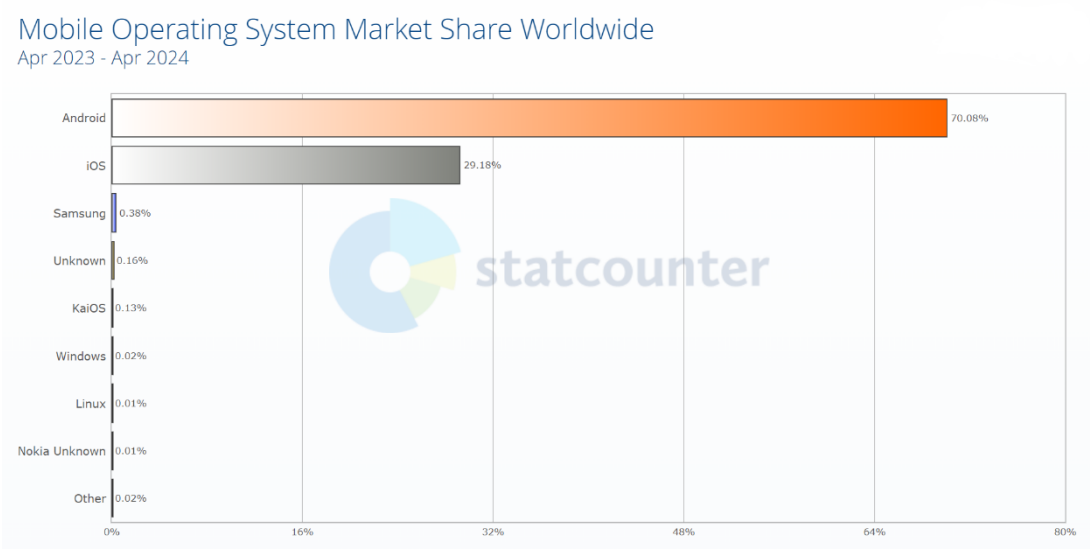

Рисунок 8 – Сравнительный график популярности мобильных ОС

Сравнительный график наглядно демонстрирует что лидирующую позицию занимает Android – мобильная платформа от Google, с рыночной долей в 70,08% по состоянию на апрель 2024 года. «Android - открытый исходный код мобильной операционной системы для широкого спектра устройств. Для возникновения Android многие компании сделали свои вложения. В настоящее время проект с открытым кодом Android во главе с

Google и большинство Android приложений лицензируются с Apache 2.02» [16]. Эта система привлекательна для разработчиков благодаря широкому ассортименту устройств и гибким возможностям для разработки.

Менее популярной является iOS - операционная система от Apple, используемая в таких устройствах, как iPhone и iPad. По состоянию на апрель 2023 года доля рынка составляет около 29,18%. «IOS – ОС, разработанная компанией Apple для своих мобильных устройств. Архитектура iOS основана на ядре Darwin, которое представляет собой свободное и открытое программное обеспечение. Это ядро обеспечивает низкоуровневые функции операционной системы, такие как управление памятью, управление процессами и обработка ввода-вывода. На верхнем уровне архитектура iOS включает слои, такие как Core OS, Core Services, Media и Cocoa Touch, которые предоставляют разработчикам множество инструментов и функций для создания приложений» [4]. iOS выделяется своей безопасностью, стабильностью и качеством пользовательского опыта.

Третье место по популярности занимает мобильная ОС от компании Samsung, Tizen – это мобильная операционная система, разработанная Samsung в сотрудничестве с Intel и Linux Foundation. Несмотря на свою универсальность, Tizen не смогла значительно распространиться на рынке мобильных операционных систем, имея всего около 0.38% рыночной доли.

В дальнейшем анализе мобильных операционных систем будут учитываться только Android, iOS и Tizen. Остальные операционные системы имеют слишком маленькую долю рынка, что делает их несущественными для сравнения в данном контексте. Это решение основано на том, что после Tizen следующие по популярности системы занимают крайне малый процент рыночной доли, и включение их в анализ не добавит значимости результатам исследования. Таким образом, основное внимание будет сосредоточено на трёх вышеупомянутых системах, которые являются ключевыми игроками на рынке мобильных ОС.

Отобразим результаты сравнения вышеперечисленных мобильных ОС в таблице 2.

Таблица 2 – Сравнение мобильных операционных систем

Критерий	Android	iOS	Tizen
Функциональность	+	+	-
Удобство использования	+	+	-
Безопасность	+	+	-
Стабильность	+	+	-
Популярность	+	+	-
Доступность	+	-	+

Сравнительный анализ текущих операционных систем, выявил что Android – это действительно универсальный выбор. Благодаря своей адаптивности, широкому спектру функций и поддержке со стороны разработчиков, Android отлично подходит так как, предлагает гибкую настройку мобильного опыта, имея при этом доступ к огромному выбору устройств для любого бюджета.

2.2 Выбор средств разработки

В разработке ПО для Android в основном используются три IDE: Eclipse, среда Intellij IDEA, разработанная JetBrains, а также совместное решение Google и JetBrains – Android Studio. Рассмотрим каждый продукт по отдельности:

«Eclipse – является бесплатной программной платформой с открытым исходным кодом, контролируется организацией Eclipse Foundation. Написана на языке программирования Java и основной целью её создания является

повышение продуктивности процесса разработки программного обеспечения» [7]. Из достоинств можно выделить простой и понятный интерфейс, возможность установки дополнительных плагинов, а также настраивание среды под нужды разработчика.

Intellij IDEA – это IDE, разработанная компанией JetBrains, которая поддерживает такие языки программирования, как Java, Kotlin, Groovy, Scala, а также имеет интеграцию с системами контроля версий.

«Android Studio основанная на программном обеспечении Intellij IDEA от компании JetBrains, официальное средство разработки Android приложений» [18]. «Данная среда разработки доступна для Windows, OS X и Linux» [17].

Язык программирования Java выбран, потому что обеспечивает отличную интеграцию с Android Studio, обладает обширной документацией, достаточным количеством обучающих материалов как онлайн так и офлайн, является стабильным и надежным языком программирования. Java – «это объектно-ориентированный язык, основанный на классах, который разработан для переносимости, что означает, что Java код может работать на различных аппаратных средствах и операционных системах. Java широко используется для разработки приложений корпоративного уровня, мобильных приложений, видеоигр и других типов программного обеспечения. Он известен своей философией "напиши один раз, запусти где угодно", поскольку код Java может быть скомпилирован для запуска на любой платформе, поддерживающей виртуальную машину Java (JVM).» [14]

Таким образом, выбор Java для разработки мобильного приложения является обоснованным решением и позволяет вести разработку более комфортно и эффективно.

Выбор Spring в качестве основного инструмента для разработки серверной логики для мобильного приложения службы технической поддержки был продиктован несколькими важными факторами:

- мощный инструмент для создания корпоративных приложений: Spring Framework – «фреймворк с открытым исходным кодом, написанный на Java. Его можно использовать для разработки на всех этих языках. Spring предоставляет огромный набор инструментов и библиотек, которые упрощают и ускоряют процесс разработки, позволяя сосредоточиться на бизнес-логике приложения.» [23];
- использование Spring Boot: значительно упрощает разработку, обеспечивая автоматическую настройку и интеграцию с популярными библиотеками и фреймворками.
- документация и поддержка сообщества: обширная документация и активное сообщество Spring облегчают обучение и решение возникающих проблем.

Также использование базы данных PostgresSQL обладает следующими преимуществами:

- поддержка сложных запросов: мощный механизм выполнения сложных SQL-запросов, включая подзапросы, оконные функции и CTE, является одним из преимуществ PostgreSQL;
- расширяемость и гибкость: PostgreSQL поддерживает множество типов данных, индексов и расширений, что позволяет легко адаптировать базу данных под конкретные требования приложения;
- высокая производительность и стабильность: известна своей устойчивостью, масштабируемостью. А также поддерживает ACID, что обеспечивает сохранность и целостность данных;
- интеграция с Spring: Spring Data JPA обеспечивает простую интеграцию с PostgreSQL, позволяя использовать ORM для работы с базой данных, что упрощает реализацию CRUD операций и других взаимодействий;

– сообщество и документация: широкое сообщество и обширная документация PostgreSQL способствуют поддержке и развитию приложений.

Решение в пользу использования Spring и PostgreSQL было принято из-за их способности обеспечить надежную и масштабируемую архитектуру для мобильного приложения технической поддержки. Spring предоставляет мощные инструменты для разработки бизнес-логики и обеспечения безопасности, а PostgreSQL обеспечивает высокую производительность и целостность данных. Вместе они образуют надежную платформу для создания функционального и эффективного мобильного приложения.

2.3 Проектирование концептуальной модели базы данных

В своей книге Дубейковский пишет: «проектирование базы данных включает в себя следующие этапы: построение концептуальной схемы по объектам предметной области, построение логической модели базы данных. Концептуальную схему целесообразно строить с использованием нотации Чена» [3].

Нотация Мартина, использованная в процессе проектирования логической модели данных содержит:

– сущность – «индивидуально идентифицируемый объект реальности, такой как человек, концепция или событие, о котором могут храниться данные.» [20];
– «атрибуты – это характеристики сущности, отношения многие ко многим или отношения один к одному» [19];
– отношение – указывает связи между сущностями.

Выделяет три типа отношений между сущностями:

– отношение – указывает связи между сущностями;

– один к одному – «один экземпляр сущности связан только с одним экземпляром другой сущности.» [11];
– один ко многим – «один экземпляр сущности связан со множеством экземпляров другой сущности.» [11];
– многие ко многим - «множество экземпляров одной сущности связаны со множеством экземпляров другой сущности.» [11].

Учитывая вышеперечисленное, нотация Мартина помогает изобразить предметную область на схеме включая её сущности, атрибуты и связи между ними.

Концептуальная схема предназначена для определения основных сущностей, необходимых для решения поставленных задач.

В соответствии с предметной областью в приложении были выделены следующие сущности:

– user (пользователь) – пользователь системы, который может создавать заявки и выполнять их. Каждый пользователь принадлежит к одному отделу и может иметь несколько ролей;
– ticket (заявка) – заявка, создаваемая пользователями для описания проблемы или запроса на обслуживание. Заявка имеет различные атрибуты, такие как название, описание, статус, время, приоритет и комментарии;
– role (роль) – роль пользователя в системе, определяющая его права и обязанности. Может иметь несколько ролей, таких как «Пользователь» «Администратор» или «Исполнитель»;
– department (отдел) – отдел, к которому принадлежит пользователь.

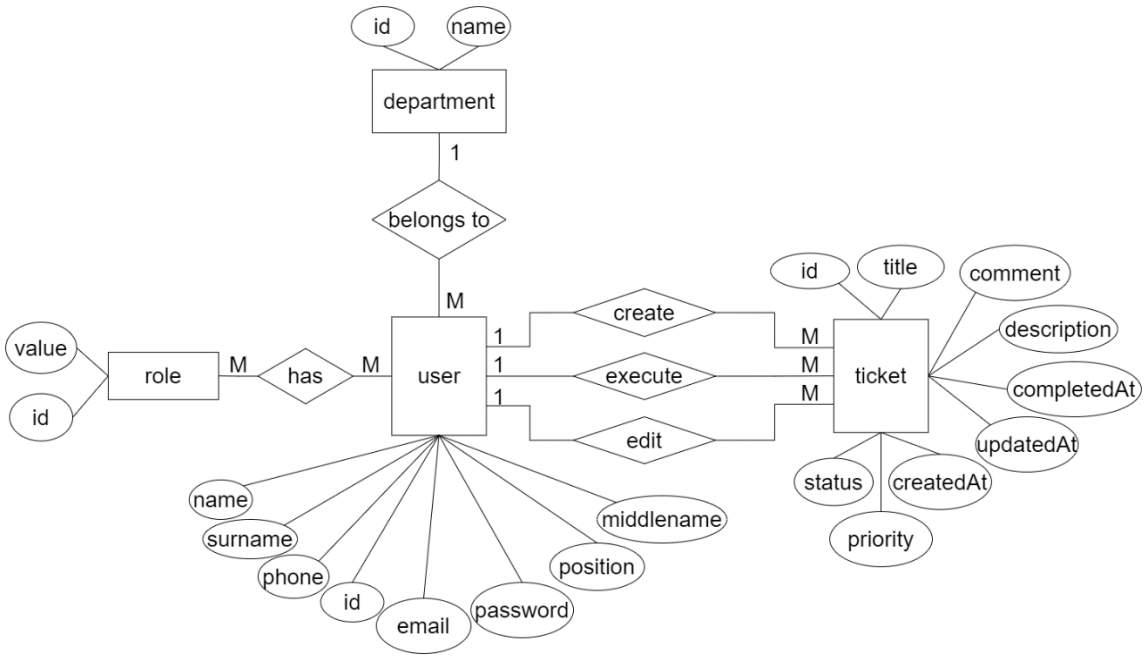

Рисунок 9 – Концептуальная модель базы данных

На рисунке 9 представлена концептуальная модель для базы данных проектируемой системы.

Концептуальная схема играет ключевую роль в создании логической модели данных информационной системы для службы технической поддержки организации.

Чтобы разрабатывать логическую модель данных, потребуется проделать анализ видов баз данных. На данный момент существует два вида баз данных: реляционные и нереляционные. Нереляционные базы данных имею два подвида: сетевые и иерархические. Рассмотрим особенности каждой из них:

«Реляционная модель данных была представлена доктором Эдгаром Ф. Коддом в 1970 году как способ упростить представление отношений данных. Реляционная модель представляет данные в виде отношений, которые по сути представляют собой таблицы со строками и столбцами. Каждая строка, также известная как кортеж, представляет одну запись данных, а каждый столбец соответствует атрибуту типа данных.» [6].

«Иерархическая модель данных – одна из первых моделей баз данных, разработанная в 1960-х годах. Он представляет данные с использованием древовидных структур, где каждый узел содержит один родительский и несколько дочерних узлов. Эта модель хорошо подходит для отношений «один ко многим» (1:N), когда родительский объект связан с несколькими дочерними объектами.» [6]

«Сетевая модель данных была разработана в конце 1960-х годов как развитие иерархической модели. Он расширяет иерархическую модель, позволяя узлу иметь несколько родительских и дочерних узлов. Эта гибкость позволяет сетевой модели данных представлять отношения «многие ко многим» (M:N), что делает ее подходящей для более сложных структур данных.» [6]

Проанализировав самые распространённые виды баз данных, было принято решение выбрать реляционную модели.

Так же, в процессе проектирования базы данных для приложения, необходимо выполнить нормализацию. «Нормализация базы данных (БД) - это метод проектирования реляционных БД, который помогает правильно структурировать таблицы данных. Процесс направлен на создание системы с четким представлением информации и взаимосвязей, без избыточности и потери данных.» [15].

Любая разрабатываемая база данных взаимодействует с системой управления базами данных (СУБД). На основании анализа для проектируемого приложения была выбрана реляционная модель данных. В качестве СУБД для этой базы данных было решено использовать PostgreSQL.

«PostgreSQL, или просто postgres, - объектно-реляционная система управления базами данных с открытым исходным кодом. На первом месте в ней стоит расширяемость, техническое совершенство и совместимость. Она конкурирует с основными базами данными: Oracle, MySQL, SQL Server и другими. Используется в самых разных секторах, включая государственные учреждения, открытые и коммерческие продукты. Это кросс-платформенная

СУБД, работающая в большинстве современных операционных систем, в т. ч. Windows, macOS и различных дистрибутивах Linux. Совместима со стандартами SQL и обладает всеми свойствами ACID.» [2]

Разработанная логическая модель для базы данных приложения приведена на рисунке 10.

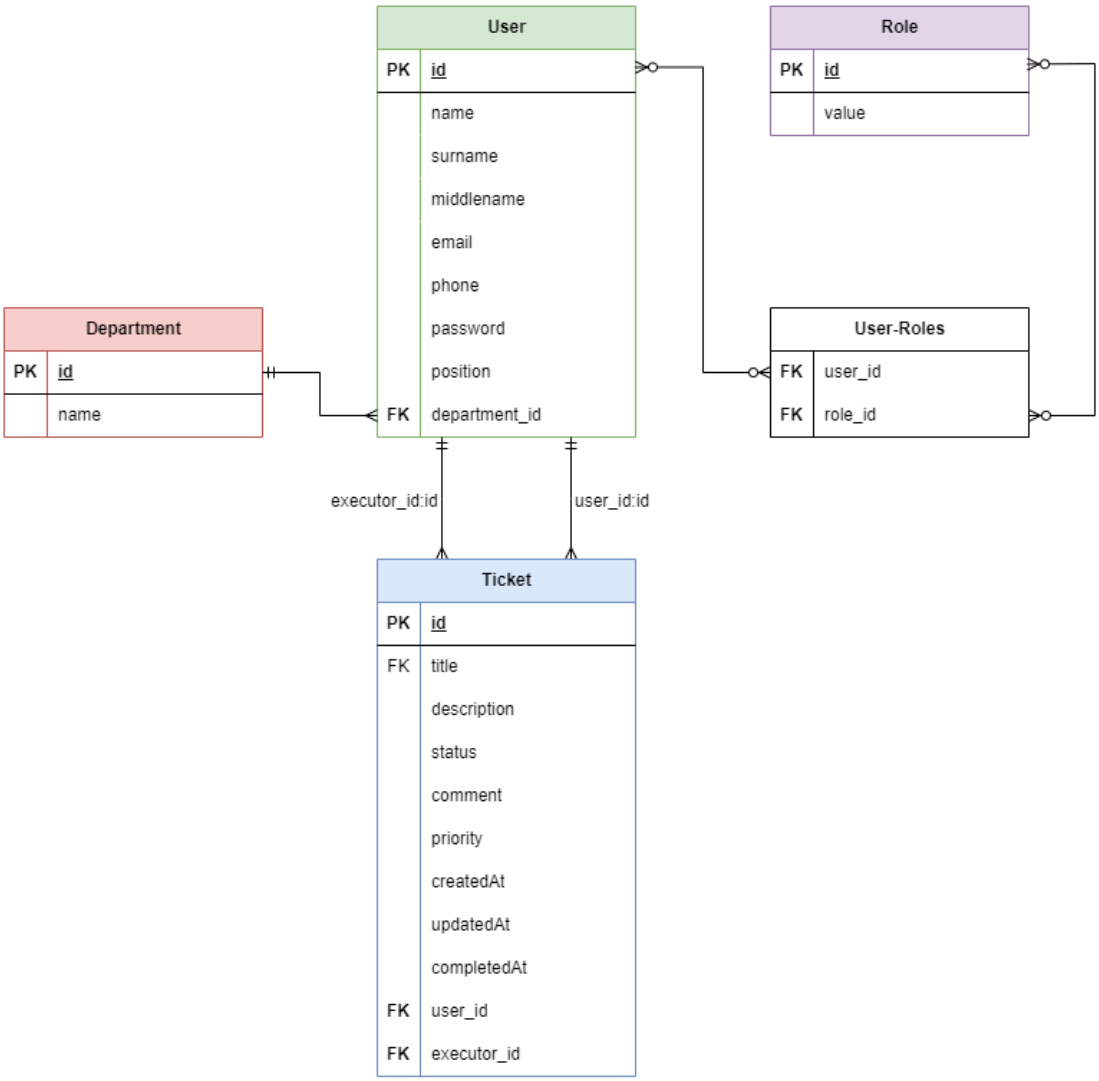

Рисунок 10 – Логическая модель базы данных

Представим связи между сущностями в таблице 3:

Таблица 3 – Структура связей данных

Foreign Key	Таблица	Связанная таблица	Связь
department_id	User	Department	1:M
user_id	UserRole	User	M:M
role_id	UserRole	Role	M:M
user_id	Ticket	User (создатель)	1:M
executor_id	Ticket	User (исполнитель)	1:M

Ориентируясь на разработанную логическую модель, была создана физическая модель. Поскольку в качестве СУБД выбрана PostgreSQL, структура физической модели была адаптирована для соответствия её требованиям. На рисунке 11 представлена физическая модель базы данных.

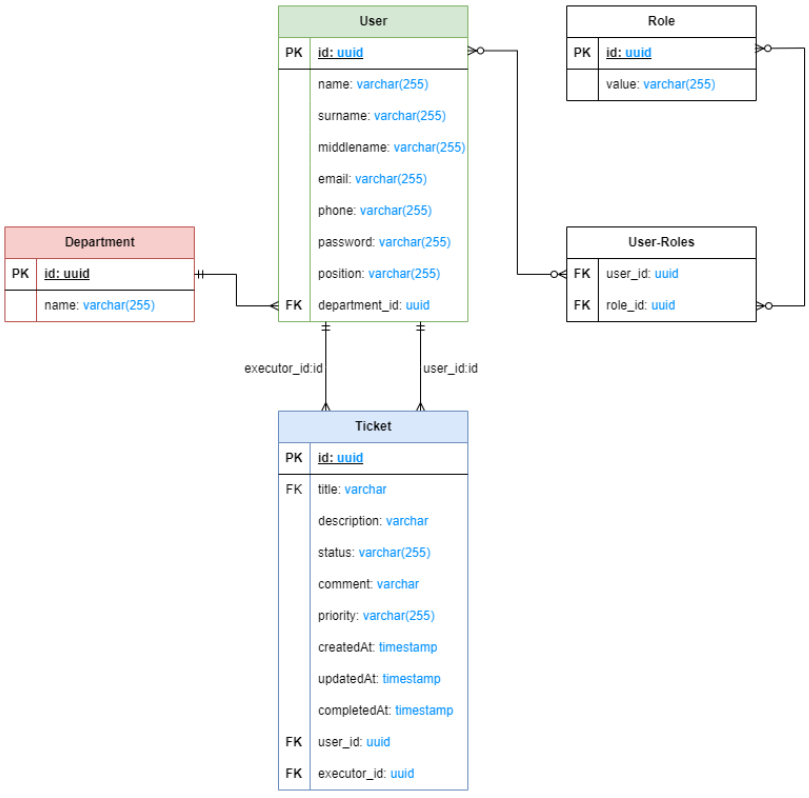

Рисунок 11 – Физическая модель базы данных

Для наглядности организации таблиц в базе данных, представим структуру БД в таблице 4.

Таблица 4 – Структура таблиц.

Номер	Название поля	Тип поля	Описание
User(Пользователь)			
1	id	uuid	Уникальный идентификатор пользователя (первичный ключ)
2	name	varchar(255)	Имя пользователя
3	surname	varchar(255)	Фамилия пользователя
4	middlename	varchar(255)	Отчество пользователя
5	email	varchar(255)	Электронная почта пользователя
6	phone	varchar(255)	Телефон пользователя
7	password	varchar(255)	Пароль пользователя
8	position	varchar(255)	Должность пользователя
9	department_id	uuid	Идентификатор отдела, к которому принадлежит пользователь (внешний ключ на department.id)
Ticket(Заявка)			
1	id	uuid	Уникальный идентификатор заявки (первичный ключ)
2	title	varchar	Название заявки
3	description	varchar	Описание заявки
4	status	varchar(255)	Статус заявки
5	comment	varchar	Комментарий к заявке
6	priority	varchar(255)	Приоритет заявки (например, "Высокий", "Средний", "Низкий")
7	createdAt	timestamp	Время создания заявки
8	updateAt	timestamp	Время последнего обновления заявки
9	completedAt	timestamp	Время завершения заявки
10	user_id	uuid	Идентификатор пользователя, создавшего заявку (внешний ключ на User.id)
11	executor_id	uuid	Идентификатор пользователя, исполняющего заявку (внешний ключ на User.id)
Department(Отдел)			
1	id	uuid	Уникальный идентификатор отдела (первичный ключ)
2	userId	uuid	Название отдела
Role(роль)			
1	id	uuid	Уникальный идентификатор роли (первичный ключ)
2	value	varchar(255)	Значение роли (пользователь, администратор, исполнитель)

Продолжение Таблицы 4

Номер	Название поля	Тип поля	Описание
User-Roles(пользователь - роль)			
1	user_id	uuid	Идентификатор пользователя (внешний ключ на user.id)
2	role_id	uuid	Идентификатор роли (внешний ключ на role.id)

Таблицы, описанные в спецификациях, будут использованы для создания структуры базы данных в рамках физической модели данных. Физическая модель определяет, как данные будут храниться, организованы и обрабатываться в реальной базе данных в специфичной системе управления базами данных (СУБД), такой как PostgresSQL. Эта модель данных адаптируется к особенностям и функциональным возможностям выбранной системы управления базами данных (СУБД). Это включает создание таблиц, определение полей и их типов данных, а также установку связей между таблицами при необходимости. Физическая модель данных становится основой для фактической реализации базы данных с применением выбранной СУБД.

2.4 Архитектура программного продукта

«Существует несколько видов архитектуры, которые могут быть использованы при разработке мобильных приложений» [1]. В мобильном приложении для учёта и управления заявками в службе технической поддержки будет использована архитектура MVVM (Model-View-ViewModel). «ViewModel в MVVM сохраняет свое состояние при изменении конфигурации экрана, например, при повороте устройства. Это уменьшает затраты на повторную загрузку данных и улучшает производительность» [10].

«В целом использование MVVM в Android разработке может значительно улучшить качество и производительность приложения, снизить затраты на разработку и обеспечить более легкое тестирование» [10]. В процессе разработки применялись активности, фрагменты, элементы пользовательского интерфейса и обработчики событий.

Для корректной работы приложения необходимы следующие компоненты:

- клиентское приложение на базе Android: в мобильном приложении предусмотрены функции создания, обновления и просмотра заявок, а также их назначения специалистам технической поддержки. Приложение должно быть настроено для работы через защищенный протокол HTTPS. Аутентификация пользователей осуществляется с помощью логин-форм и сессий;
- веб-сервер: серверное приложение на базе Spring Boot обрабатывает запросы от мобильного приложения. Реализованы контроллеры для управления заявками и пользователями, а для взаимодействия с базой данных используется Spring Data JPA. Сервер аутентификации интегрирован в серверное приложение и управляет доступом через логин-форму. Общение с приложением осуществляется по протоколу HTTPS;
- база данных PostgreSQL: для хранения информации о пользователях, ролях, заявках и отделах используется система управления базами данных PostgreSQL. База данных размещена на отдельном сервере, что обеспечивает более гибкое управление ресурсами и улучшенную масштабируемость. Взаимодействие с серверным приложением происходит через JPA.

Диаграмма развертывания мобильного приложения для учёта и управления заявками в службе технической поддержки организации изображена на рисунке 12.

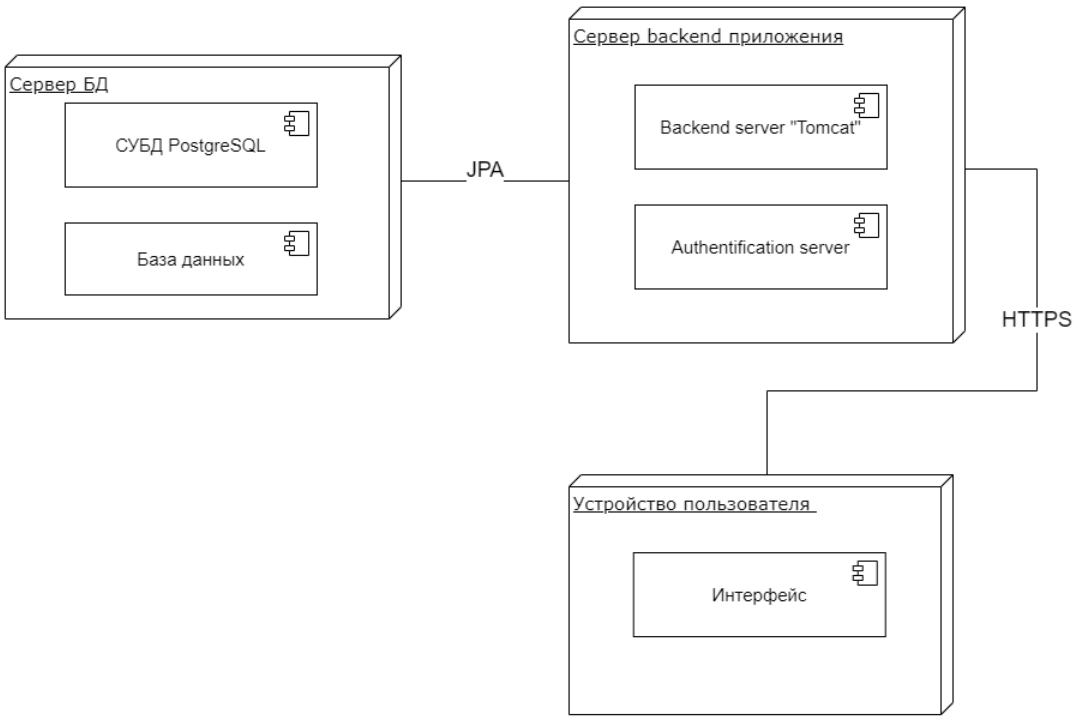

Рисунок 12 – Диаграмма развертывания

Чтобы «развернуть» разрабатываемое веб-приложение на серверах предприятия, сервер должен обладать достаточными техническими характеристиками. «Диаграммы развертывания UML представляют физическое размещение артефактов системы на ее узлах.» [7].

Как показано на рисунке 12, предприятие должно иметь как минимум два сервера: сервер для backend приложения, чтобы разместить само приложение, и сервер базы данных для хранения данных, необходимых для его правильного функционирования.

Для визуализации организации компонентов в системе на рисунке 13 представлена диаграмма компонентов.

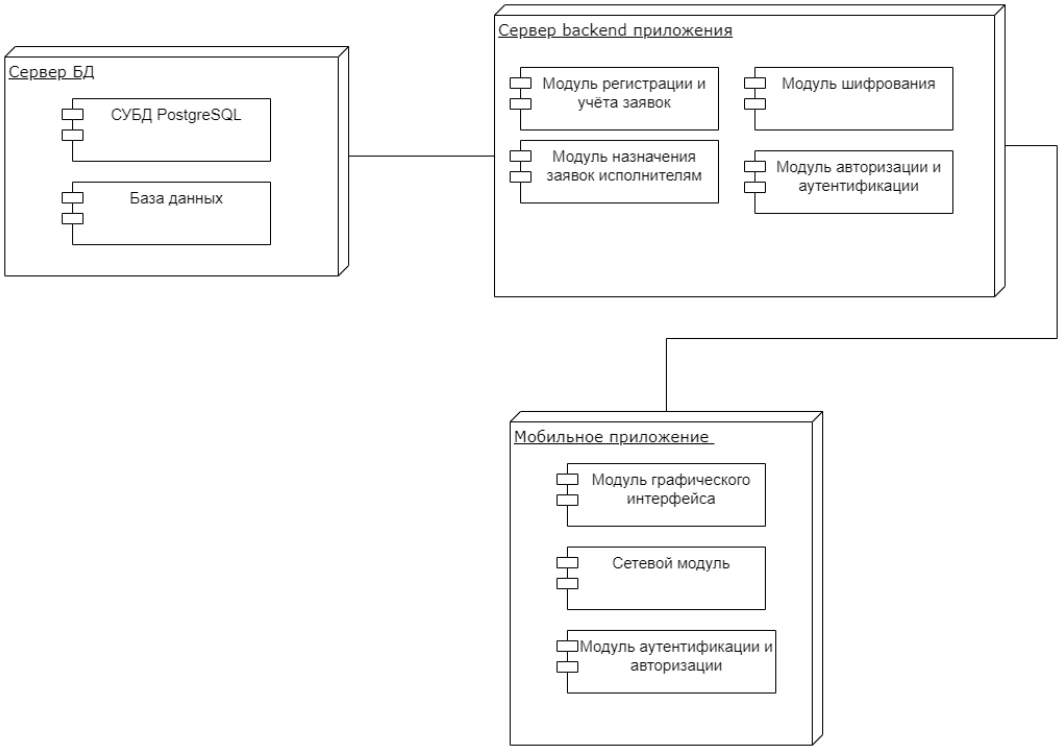

Рисунок 13 – Диаграмма компонентов

Сервер базы данных состоит из двух основных модулей: СУБД PostgreSQL, который управляет базой данных, хранящей данные пользователей, заявки, роли и отделы и базы данных, которая содержит физическую структуру данных и хранит всю необходимую информацию для работы приложения.

Сервер backend приложения представляет из себя 4 основных модуля: модуль регистрации и учета заявок, обрабатывающий создание, обновление и удаление заявок, включая приоритеты, статусы и категории. Модуль назначения заявок исполнителям, управляющий назначением заявок исполнителям и отслеживающий их выполнение. Модуль шифрования, обеспечивающий безопасность данных через шифрование. Также модуль авторизации и аутентификации, который обрабатывает аутентификацию и авторизацию пользователей, ограничивая доступ к системе.

Мобильное приложение имеет три основных модуля модуль графического интерфейса, обеспечивающий взаимодействие с

пользователем, отображая данные и принимая ввод. Сетевой модуль, выполняющий сетевые запросы к серверному приложению для взаимодействия с backend-сервером и модуль аутентификации и авторизации, управляющий входом в систему и проверкой прав доступа пользователей.

Разделение программы на взаимодействующие сервисы является эффективным подходом, обеспечивающим возможность модульного тестирования и повышающим отказоустойчивость, поскольку каждый модуль можно разрабатывать независимо от остальных.

Выводы по главе 2

В данной главе описан и обоснован выбор технологий разработки. Проведен анализ операционных систем для мобильной разработки, и выбрана платформа Android. Средой разработки выбрана Android Studio, а языком программирования – Java.

Для серверной логики выбран фреймворк Spring, а для управления базами данных – PostgreSQL.

Спроектированы концептуальная, логическая и физическая модели базы данных, определены основные сущности и их связи.

Для мобильного приложения выбрана архитектура MVVM, улучшая разделение логики и интерфейса. Рассмотрена диаграмма развертывания, и диаграмма компонентов, которые уточняют требования к аппаратному обеспечению.

Глава 3 Реализация мобильного приложения

3.1 Описание разработанного программного решения

Для учета и управления заявками в службе технической поддержки организации было разработано мобильное приложение. Приложение было написано на языке программирования Java и развернуто на платформе Android. Мобильное приложение взаимодействует с серверным приложением на базе Spring Boot и использует базу данных PostgreSQL для надежного хранения и управления данными.

Также используются следующие компоненты Spring Core, Spring Data JPA, Spring Security, Spring Web, Spring Boot версии 3.0 и выше, Spring Validator, а также сторонние библиотеки: PostgreSQL Driver, Hibernate, Lombok, JWT.

Созданное мобильное приложение позволяет пользователям организации создавать, обновлять, просматривать заявки при обнаружении проблем или необходимости в технической поддержке, а также отслеживать статус созданных ранее заявок. Приложение использует REST API, предоставляемый серверным приложением, и передает данные на веб-сервер организации, что позволяет службе технической поддержки оперативно работать с заявками. Также приложение имеет функционал администратора для назначения заявок компетентным исполнителям и различные интерфейсы для отдельных ролей таких как Пользователь, Администратор, Исполнитель.

Согласно принципу «code first», сначала были разработаны сущности и их зависимости в программном коде. Рисунок 14 демонстрирует пример класса-сущности.

```java
public class Ticket {
    @Column(name = "id", nullable = false)
    private UUID id;
    // Заголовок тикета
    @Column(name = "title", nullable = false)
    private String title;
    // Описание тикета
    @Column(name = "description")
    private String description;
    // Текущий статус тикета
    @Column(name = "status", nullable = false)
    private String status;
    // Дополнительный комментарий к тикету
    @Column(name = "comment")
    private String comment;
    // Уровень приоритета тикета
    @Column(name = "priority")
    private String priority;
    // Дата и время создания тикета
    @Column(name = "created_at", nullable = false)
    private LocalDateTime createdAt;
    // Дата и время последнего обновления тикета
    @Column(name = "updated_at")
    private LocalDateTime updatedAt;
    // Дата и время завершения тикета
    @Column(name = "completed_at")
    private LocalDateTime completedAt;
```

Рисунок 14 – Класс-сущность Ticket

Класс Ticket представляет собой сущность для хранения информации о тикетах в системе управления задачами. Каждый тикет имеет уникальный идентификатор (id), заголовок (title), описание (description), текущий статус (status), комментарий (comment), приоритет (priority), дата и время создания (createdAt), дата и время последнего обновления (updatedAt) и дата и время завершения (completedAt). Также каждый тикет связан с пользователем, который его создал (user) и пользователем, который назначен на его выполнение (executor).

Далее, на основе сущностей и их зависимостей, была спроектирована база данных, в качестве СУБД был выбран PostgreSQL. Физическая модель базы данных изображена на рисунке 15.

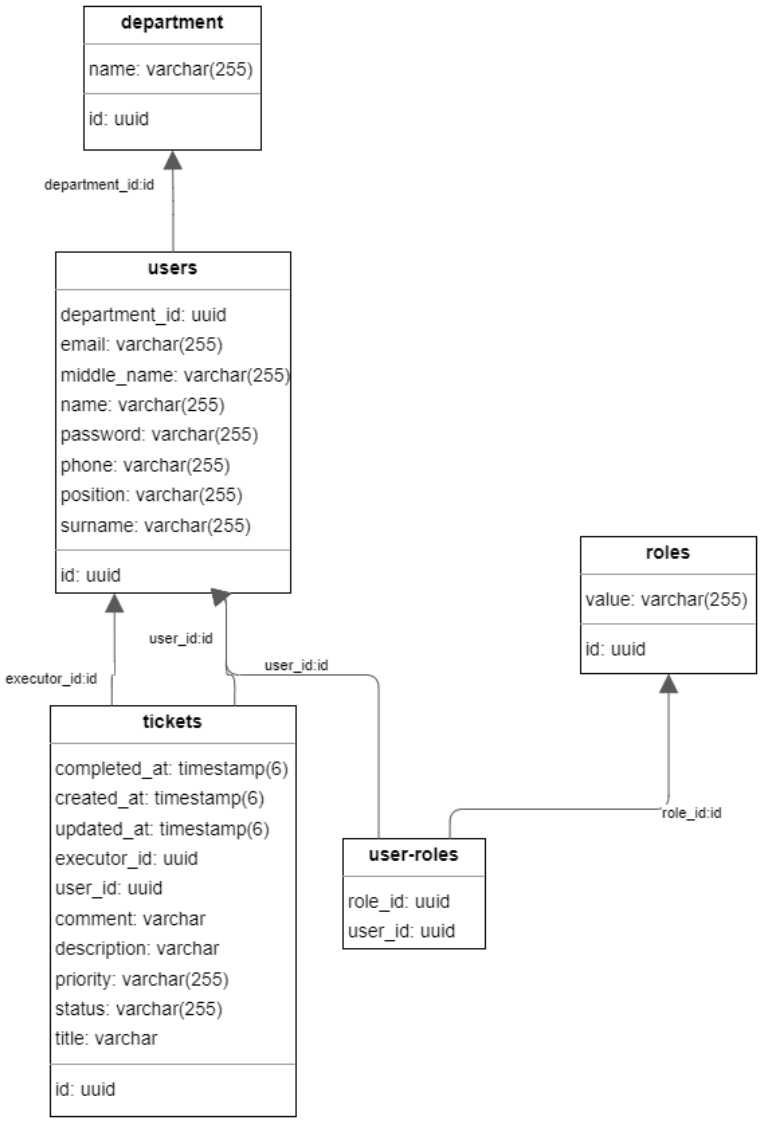

Рисунок 15 – Физическая модель БД

По спецификации JPA, «JPA означает Java Persistence API. Это платформа объектно-реляционного сопоставления (ORM), которая позволяет нам сопоставлять объекты Java с таблицами в реляционной базе данных. Другими словами, JPA предоставляет способ сохранения объектов Java в базе данных с помощью набора аннотаций, определяющих сопоставление между

классами Java и таблицами базы данных.» [22], для инкапсуляции слоя доступа к хранилищу данных рекомендуется использовать паттерн «Репозиторий».

Для четкого разделения зон ответственности, все классы были распределены по различным пакетам и выделены в отдельные «слои». Диаграмма пакетов представлена на рисунке 16.

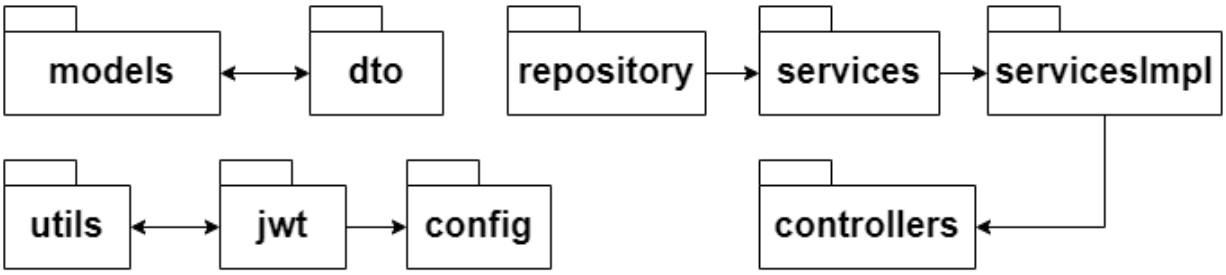

Рисунок 16 – Диаграмма пакетов приложения

- models – пакет, содержащий классы-сущности;
- dto – слой представления данных;
- repository – пакет с классами для доступа к хранилищу данных, реализующими паттерн «репозиторий»;
- services – содержит классы (интерфейсы) с сигнатурами методом для работы с данными;
- servicesImpl – содержит классы, реализующие классы пакета services, в класса реализуется логика работы с данными;
- controllers – в пакете содержатся классы, отвечающие за обработку HTTP запросов, приходящих от «клиента»;
- jwt – содержит классы конфигурации авторизации/регистрации;
- config – пакет с конфигурацией приложения.

Программный код некоторых классов приведен в приложении.

3.2 Реализация и обзор разработанного мобильного приложения

Первое что встречает пользователь, это экран авторизации. На рисунке 17 приведена экранная форма авторизации пользователя. Форма принимает следующие данные: почта, которая является логином и пароль. После успешной авторизации, пользователю присваивается JWT-токен, этот токен временный и является токеном доступа к функциям приложения. «JSON Web Token (JWT) – это открытый стандарт (RFC 7519), определяющий компактный и автономный способ безопасной передачи информации между сторонами в виде объекта JSON. Эту информацию можно проверить и ей можно доверять, поскольку она имеет цифровую подпись. JWT можно подписать с использованием секрета (с помощью алгоритма HMAC) или пары открытого/закрытого ключей с использованием RSA или ECDSA.» [21]

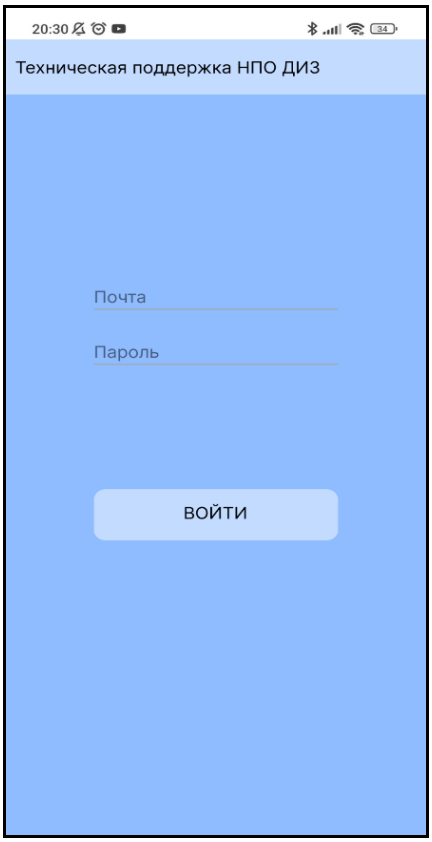

Рисунок 17 – Экранная форма авторизации пользователя

Процесс авторизации в приложении работает следующим образом: пользователь вводит свой адрес электронной почты и пароль в соответствующие поля и нажимает кнопку «Войти». После нажатия кнопки «Войти» введённые данные передаются на сервер для проверки. Затем сервер проверяет, существует ли пользователь с указанными данными в базе и соответствует ли пароль. В случае успешной проверки сервер возвращает токен аутентификации и данные пользователя. После этого мобильное приложение сохраняет полученный токен и данные пользователя для дальнейшей работы.

После успешной авторизации пользователю становятся доступны функции приложения в зависимости от его прав доступа. На рисунке 18 представлена экранная форма главного меню пользователя и доступных ему функций.

Рисунок 18 – Экранная форма главного меню приложения для пользователя

На главном экране пользователь может наблюдать основные функции приложения, которые включают создание новых заявок, просмотр созданных им заявок, а также управление личной информацией через личный кабинет. Каждое действие, связанное с взаимодействием с сервером, сопровождается необходимым обменом данными для выполнения соответствующих операций.

При нажатии на кнопку «НОВАЯ ЗАЯВКА» пользователь переходит на экран создания новой заявки. Заполненная форма заявки отправляется через POST-запрос к API endpoint, /api/tickets/create. Затем сервер обрабатывает запрос, создаёт новую заявку и сохраняет её в базе данных PostgreSQL через TicketRepository. В ответ сервер отправляет подтверждение успешного создания заявки.

Функция «МОИ ЗАЯВКИ» отправляет GET-запрос к API endpoint, /api/tickets/user/{userId}, где {userId} – идентификатор текущего пользователя. Сервер возвращает список заявок, созданных пользователем. Затем приложение отображает полученный список на экране.

Если пользователю необходимо зайти в личный кабинет, то для этого отправляется GET-запрос к API endpoint, например /api/users/{userId}. Сервер возвращает данные профиля пользователя. Приложение отображает полученные данные на экране.

На рисунке 19 изображена экранная форма составления новой заявки авторизованным пользователем.

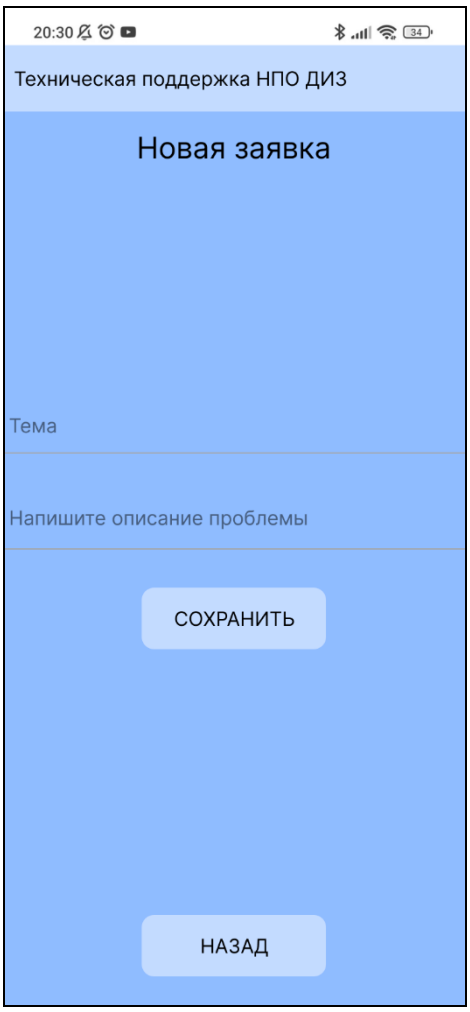

Рисунок 19 – Составление новой заявки

Экран создания новой заявки предназначен для удобного и быстрого оформления запросов на техническую поддержку. Пользователь вводит тему и описание проблемы, а затем отправляет заявку на сервер для обработки.

Когда авторизованный сотрудник сохраняет заполненную заявку, сервер принимает этот запрос и передает данные в соответствующий контроллер TicketController. Заявка сохраняется в репозитории TicketRepository. А в базе данных создается новая запись с указанными полями: title, description, userId, а также автоматически генерируемыми полями createdAt (время создания) и status (статус при создании новой заявки ставится по умолчанию «новая»).

После того как пользователь отправил заявку, теперь как администратор, так и сам пользователь могут просматривать её в списке заявок. На рисунках 20 и 21 представлены экранные формы для просмотра заявок пользователем и администратором соответственно.

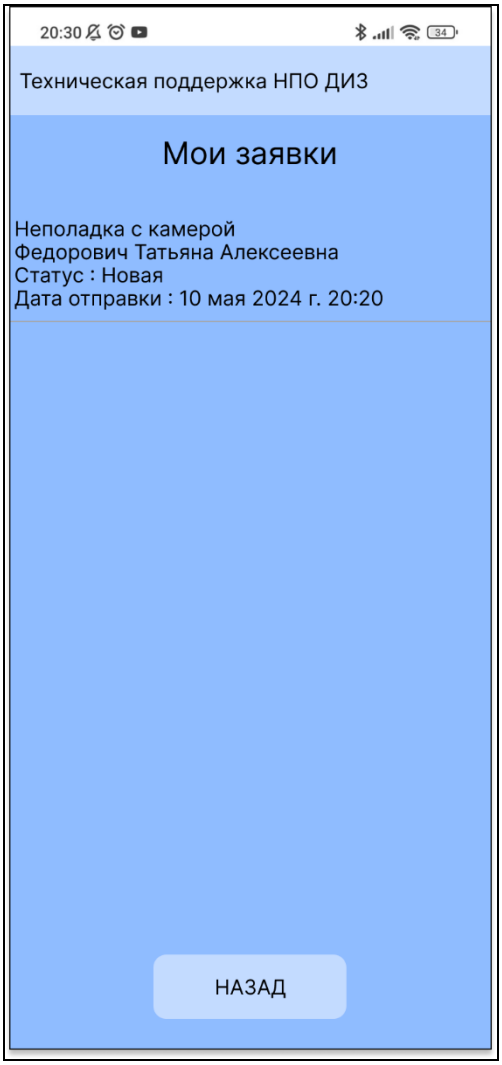

Рисунок 20 – Просмотр заявок пользователем

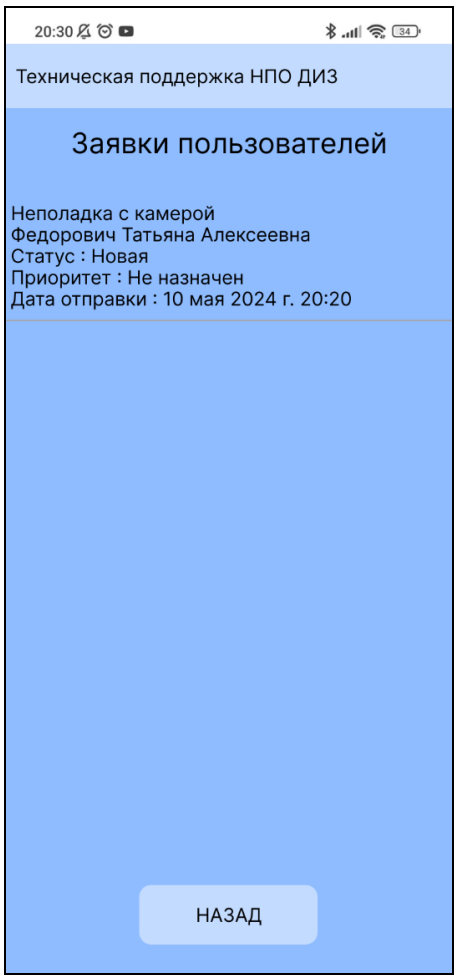

Рисунок 21 – Просмотр заявок администратором

На рисунке 22 демонстрируется отчёт сервера о созданной заявке.


```
{
  "id": "2c2df52f-b137-4659-9654-ff3593dbc950",
  "title": "Неполадка с камерой",
  "description": "Не работает камера слежения",
  "status": "Новая",
  "comment": null,
  "priority": null,
  "createdAt": "10.05.2024 20:20",
  "updatedAt": null,
  "completedAt": null,
  "user": {
    "id": "e5a7b8d9-6756-44ca-bcc4-b086ec5b0825",
    "name": "Татьяна",
    "surname": "Алексеевна",
    "middleName": "Федерович",
    "email": "tatianafeder@yandex.ru",
    "phone": "89325671212",
    "department": "Юридический отдел",
    "position": "Юрист"
  },
  "executor": null
}
```


Рисунок 22 – Серверное отображение новой заявки

В зависимости от роли сотрудника в системе, данные в списке заявок отличаются. Все роли видят поля с темой, ФИО пользователя заявки, статусом, и датой отправки. Исполнителю и администратору дополнительно представлено поле приоритета, для оказания более эффективной технической поддержки. А пользователь может быстро ориентироваться в своих запросах и отслеживать их статус.

В процессе обработки заявки администратор должен назначить статус заявки, её приоритет, а также исполнителя из выпадающих списков статуса, приоритета и исполнителя соответственно. А также при необходимости добавить комментарий для исполнителя заявки. На рисунке 23 демонстрируется экранная форма обработки заявки администратором.

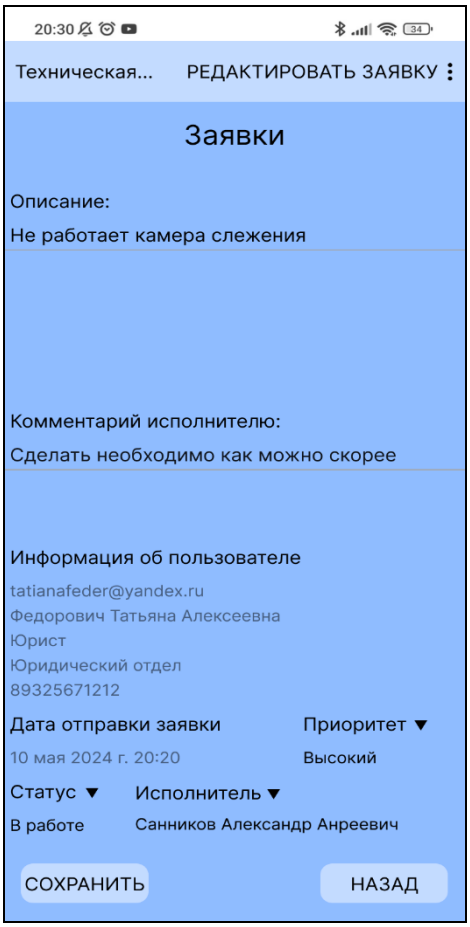

Рисунок 23 – Обработка заявок администратором

На рисунке 24 показано, как сервер возвращает детальную информацию об обработанной заявке, включая тему, описание, данные о назначенном исполнителе, комментарий, данные пользователя, статус время создания и обновления статуса заявки.

Рисунок 24 – Серверная информация об обработанной заявке

Сформированная и обработанная заявка отравляется в список заявок на выполнение соответствующим пользователем с ролью исполнителя. На рисунке 25 изображен отображаемый список заявок для выполнения исполнителем.

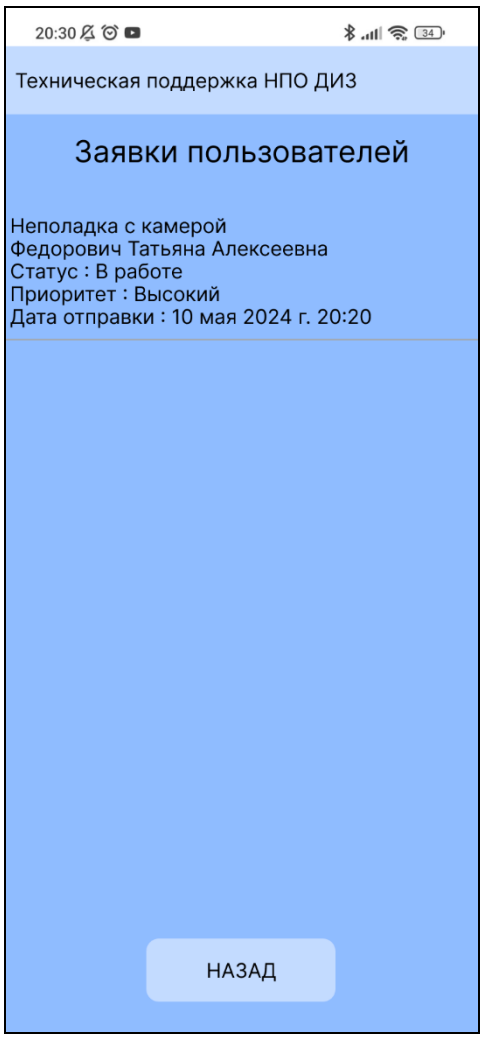

Рисунок 25 – Список назначенных заявок для исполнителя

После выполнения заявки, исполнитель также может изменить статус заявки на «завершен». Данные, которые содержит оформленная заявка в дальнейшем будет использоваться для составления отчётов службой поддержки. На рисунке 26 демонстрируется экранная форма выполняемой заявки от лица исполнителя.

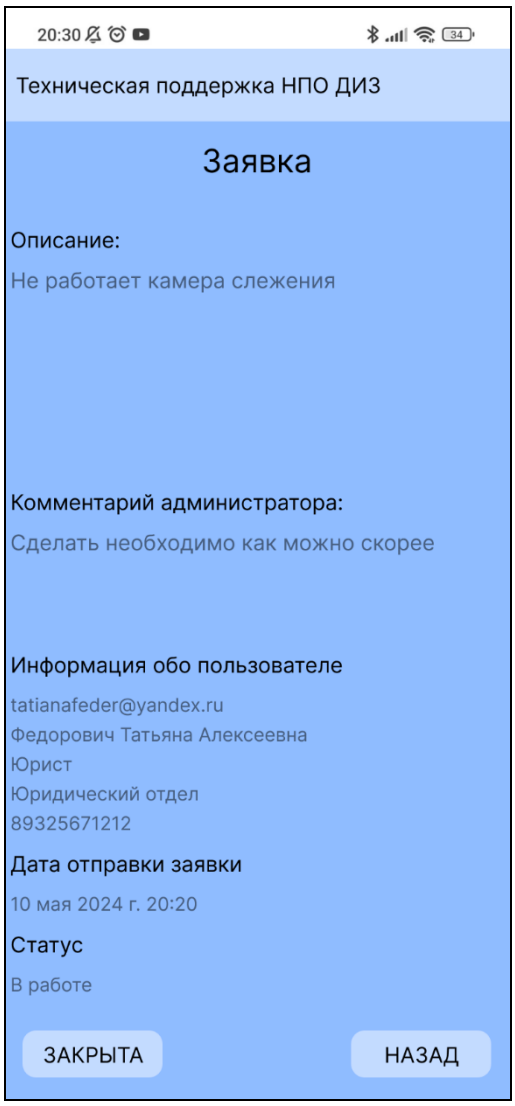

Рисунок 26 – Выполняемая заявка от лица исполнителя

После успешного выполнения сервер также обновляет информацию о текущем состоянии заявки. Рисунок 27 показывает серверное отображение завершенной заявки.

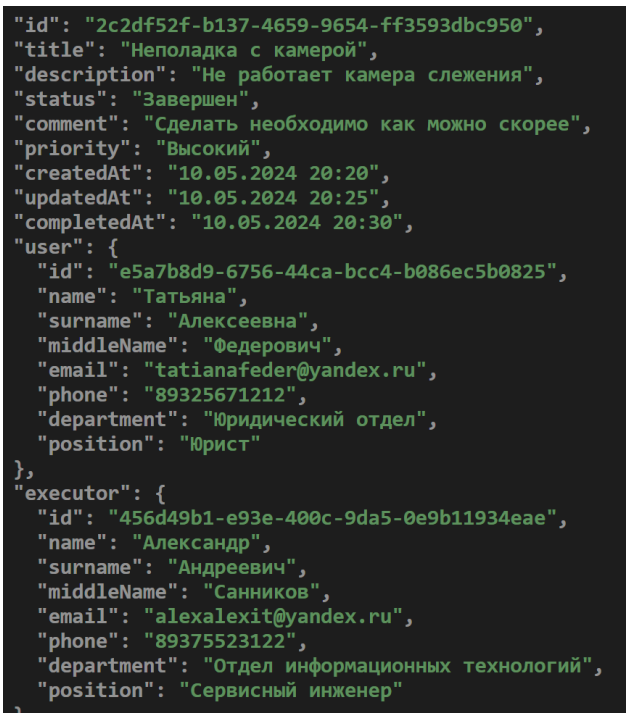

Рисунок 27 – Серверное отображение завершенной заявки

JSON-объект, который сервер отправляет в ответ на запрос завершенной заявки, содержит полную информацию о заявке и включает данные о пользователе, создавшем заявку, и исполнителе, завершившем работу.

3.3 Реализация администрирования системы учета

В данном разделе рассматривается функциональность, предоставляемая администратору для управления системой учета и управления заявками в службе технической поддержки организации. Администрирование системы осуществляется через интерфейс панели управления, который позволяет администраторам выполнять ключевые задачи управления пользователями и заявками.

Для входа в панель администрирования необходимо внести в поля ввода логин и пароль, которые соответствуют роли администратора. Рисунок

28 содержит демонстрацию ввода данных для авторизации в системе администрирования.

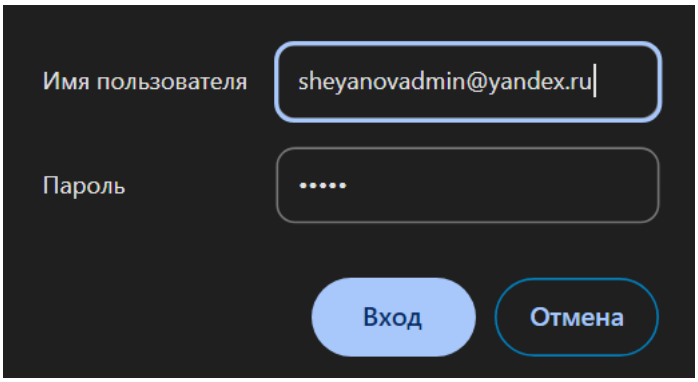

Рисунок 28 – Окно авторизации панели администрирования

На рисунке 29 изображена панель администрирования, которая предоставляет собой удобный и интуитивно понятный интерфейс для взаимодействия администратора с базой данных пользователей и заявок, а также внесения изменений.

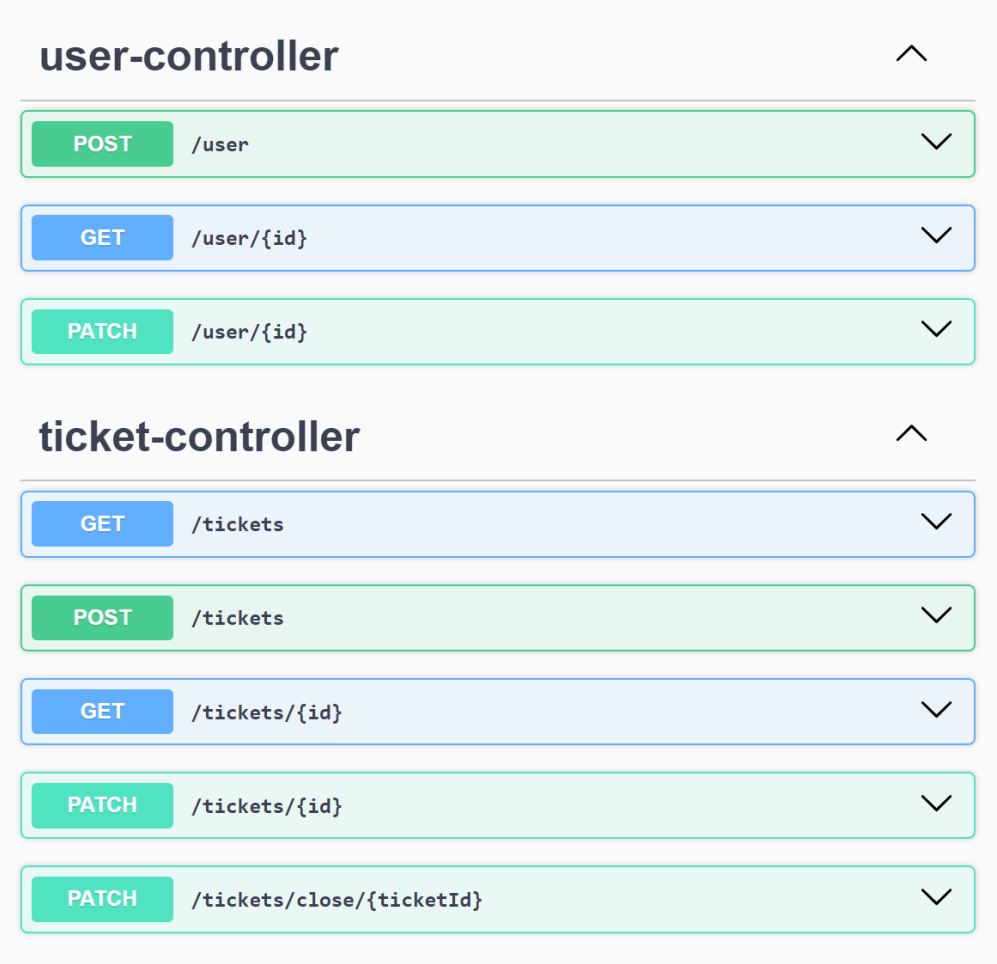

Рисунок 29 – Панель администрирования пользователями и заявками

С помощью этой панели администрирования, системный администратор имеет возможность управлять пользователями с помощью функций user-controller:

– POST /user – функция регистрации нового пользователя в системе. Позволяет администратору создавать новых пользователей. Для этого в тело запроса вводятся данные нового пользователя, включая логин и пароль и после выполнения новый пользователь сохраняется в базе данных.
– GET /user/{id} – функция получения информации о пользователе по ID. Благодаря этому администратор может получить подробную информацию о пользователе по его ID.

– PATCH /user/{id} – редактирование информации о пользователе по ID. Функция позволяет редактировать данные существующего пользователя в том числе назначить, изменить или убрать ему роль в системе (например «User», «Executor» или «Admin»).

С помощью ticket-controller администратор имеет следующие функции по администрированию заявок:

– GET /tickets – функция получения списка всех заявок находящихся в системе. Позволяет администратору осуществлять мониторинг поступающих заявок и их статусов.
– POST /tickets – функция создания новой заявки. В теле запроса передаются данные новой заявки, такие как заголовок, описание проблемы и комментарий, адресованный исполнителю.
– GET /tickets/{id} – получение информации о конкретной заявке по её ID. С помощью этой функции администратором осуществляется просмотр конкретной заявки пользователя для последующей обработки и анализа.
– PATCH /tickets/{id} – функция редактирования определенной заявки по ID. Эта функция является ключевой в обработке заявок, так как позволяет администратору изменять напрямую данные заявки при необходимости, а также назначать приоритет, обновлять статус и назначать исполнителя заявки.
– PATCH /tickets/close/{ticketid} – функция закрытия заявки. Администратор при необходимости имеет возможность закрыть заявку и в процессе обновить статус на «завершён».

Рассмотрим подробнее одну из ключевых функций панели администратора – регистрация нового пользователя в системе.

Администратор системы имеет возможность регистрировать в систему новых пользователей, что осуществляется через функцию POST /user изображенной на рисунке 30. Эта функция предназначена для добавления

новых пользователей в базу данных системы. Рассмотрим этот процесс более детально.

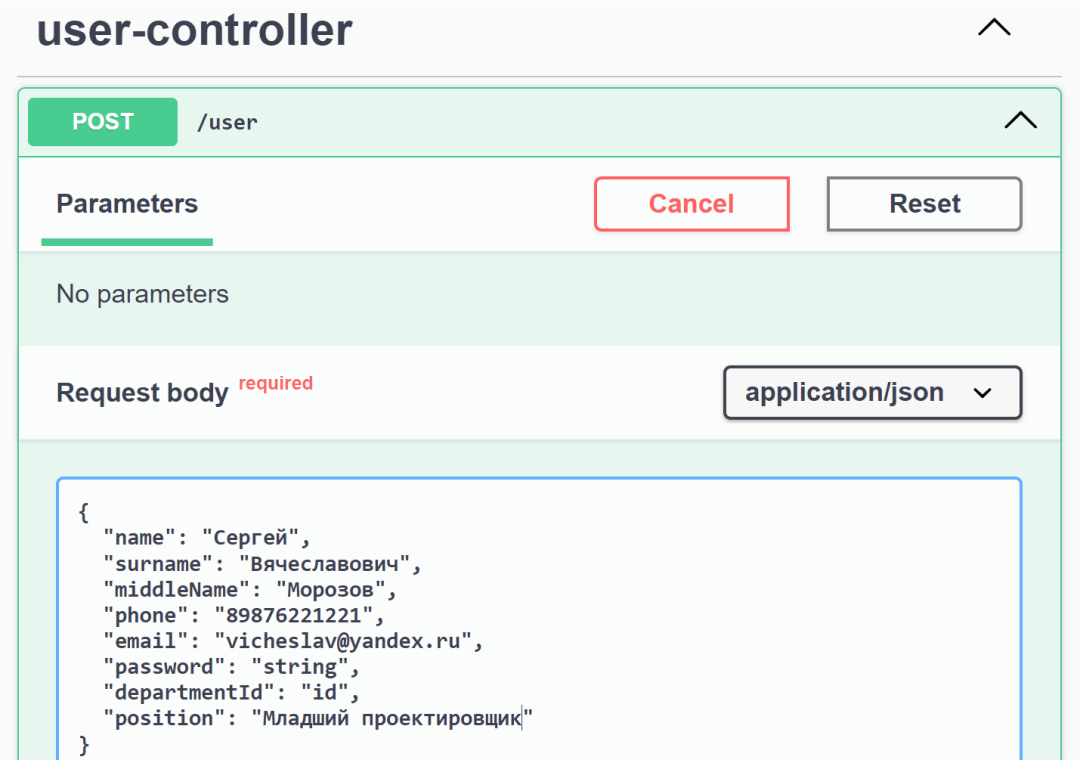

Рисунок 30 – Функция регистрации новых пользователей Post /user

Чтобы зарегистрировать нового пользователя в системе, администратор в теле запроса вводит информацию о новом пользователе. Поля для заполнения информации о новом пользователе включают в себя:

– id (UUID) – уникальный идентификатор пользователя;
– name (строка) – имя пользователя;
– surname (строка) – фамилия пользователя;
– middleName (строка) – отчество пользователя;
– phone (строка) – номер телефона пользователя;
– email (строка) – электронная почта пользователя;
– department (строка) – название отдела, к которому принадлежит пользователь;

– position (строка) – должность пользователя.

При успешной регистрации нового пользователя, сервер возвращает HTTP статус 200 и отчётный файл в формате JSON с данными нового пользователя. Результат успешной регистрации продемонстрирован на рисунке 31.

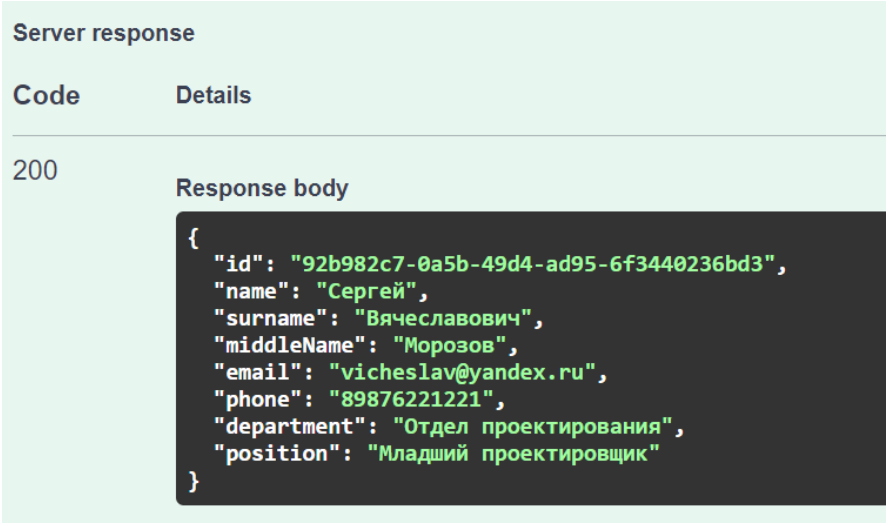

Рисунок 31 – Отчет успешной регистрации нового пользователя

Для этого контроллер валидирует полученные данные. Проверяет, что все обязательные поля заполнены, формат данных корректный, и что данные соответствуют бизнес-логике. Затем проверяет уникальность данных, чтобы убедиться, что такого пользователя еще в системе. Для этого контроллер отправляет запрос в базу данных. Поле нового пользователя в базе данных отображено на рисунке 32.

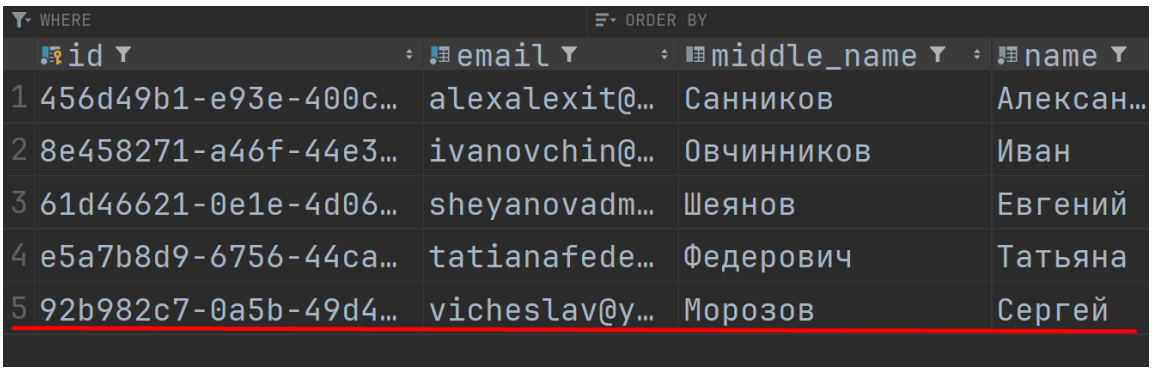

Рисунок 32 – Таблица базы данных пользователей

Введённый администратором пароль для нового пользователя перед сохранением в базе данных необходимо хешировать с помощью алгоритма bcrypt, для создания безопасного хеша. После успешной валидации и хеширования формируется объект пользователя с данными, полученными из запроса, и передаётся в базу данных.

3.4 Тестирование мобильного приложения

Тестирование является важной частью разработки любого приложения, включая наше приложение для планирования задач. Цель тестирования заключается в обнаружении и исправлении ошибок и проблем в приложении, а также проверке его функциональности, надежности и производительности.

Для тестирования нашего приложения были использовали различные методы тестирования, такие как модульное тестирование, функциональное тестирование и тестирование пользовательского интерфейса.

Модульное тестирование позволяет тестировать отдельные модули приложения независимо друг от друга. Был протестирован и отлажен каждый модуль приложения.

Функциональное тестирование позволяет тестировать приложение в целом, проверяя, как работают его различные функции и модули вместе. Было проведено функциональное тестирование нашего приложения,

используя различные тестовые сценарии, чтобы убедиться, что все функции работают должным образом и взаимодействуют друг с другом без ошибок.

Тестирование пользовательского интерфейса позволяет проверить, насколько удобным и интуитивным является интерфейс нашего приложения для пользователя. Было проведено тестирование пользовательского интерфейса, используя тестовых пользователей, чтобы оценить, насколько легко пользователи могут выполнить различные задачи в приложении и обнаружить возможные проблемы с пользовательским интерфейсом. Все выявленные ошибки и проблемы были исправлены, а приложение прошло все необходимые тесты перед его выпуском.

В рамках функционального тестирования мобильного приложения были проведены тщательные проверки основных функций, начиная с авторизации и аутентификации пользователей. Этот этап включал проверку возможности успешной авторизации с использованием зарегистрированных аккаунтов, а также обработку ошибок при вводе неверных учетных данных. Кроме того, была проведена проверка процесса создания нового аккаунта, включая корректную регистрацию нового пользователя, что включало в себя заполнение всех обязательных полей.

Одним из важных функциональных элементов приложения является возможность создания и отправки заявок. В рамках тестирования проводилась проверка возможности создания новых заявок пользователями с заполнением всех необходимых полей, таких как тема и описание проблемы. Затем проводилась проверка корректного сохранения созданных заявок в базе данных и успешной отправки их на сервер для последующей обработки.

Просмотр и изменение информации о пользователях также был предметом тестирования. Этот процесс включал проверку возможности просмотра информации о пользователях, включая их ФИО, отдел, занимаемую должность и контактные данные. Также проводилась проверка возможности изменения и сохранения измененных данных пользователей, а также корректного обновления этой информации в базе данных.

Кроме того, важным этапом была проверка процесса регистрации новых пользователей в приложении. Этот процесс включал в себя не только создание новой учетной записи, но и проверку всех этапов, необходимых для успешного завершения регистрации. В ходе тестирования проверялась возможность пользователя создать новый аккаунт, заполнив все обязательные поля формы регистрации, а также корректная обработка ошибок при неправильном вводе данных. Также была проведена проверка на уникальность введенных данных для избежания конфликтов в базе данных. Этот этап тестирования не только обеспечивает бесперебойную работу системы, но и гарантирует удобство пользовательского опыта при создании нового аккаунта в мобильном приложении.

Обновление статуса заявок также было подвергнуто тщательной проверке. Это включало изменение статуса заявок с "Новая" на "В работе" и "Завершен", а также проверку корректного обновления статусов заявок в базе данных. Кроме того, проводилась проверка отправки уведомлений пользователям о изменении статуса их заявок, чтобы обеспечить своевременное информирование пользователей о состоянии их запросов.

Также были проведены проверки обработки ошибок и исключительных ситуаций. Это включало проверку возможности использования функций, не соответствующих роли пользователя, а также проверку корректной обработки ошибок при вводе данных, таких как неправильный формат или отсутствие обязательных полей.

В результате функционального тестирования было выявлено, что все основные функции приложения работают корректно и соответствуют установленным требованиям. Однако, были выявлены некоторые ошибки и недочеты, которые были исправлены в процессе разработки, что способствовало улучшению стабильности и функциональности приложения.

3.5 Развертывание и внедрение мобильного приложения

Развертывание мобильного приложения – это процесс подготовки и публикации готового продукта для конечных пользователей. Этот этап включает несколько ключевых шагов.

Подготовка релиза:

- сборка финальной версии: компиляция и сборка финальной версии приложения, которая готова к распространению;
- тестирование релиза: проведение финального тестирования, чтобы убедиться в отсутствии критических ошибок и проблем с совместимостью.

Внутреннее развертывание:

- системы управления мобильными устройствами (MDM): в крупных организациях может потребоваться использование MDM для внутреннего развертывания корпоративных приложений;
- настройка политики безопасности: установка политик безопасности и контроля доступа, чтобы защитить данные и ограничить доступ к приложению только авторизованным пользователям.

Внедрение и поддержка мобильного приложения – это постоянный процесс, который начинается сразу после его развертывания. Он включает обучение пользователей, мониторинг работы приложения, сбор отзывов и выпуск обновлений.

Обучение пользователей:

- создание документации: разработка подробной пользовательской документации, включая руководства по установке и использованию, FAQ и решению распространенных проблем;
- проведение тренингов: организация тренингов и обучающих сессий для пользователей, особенно в случае корпоративных приложений;
- поддержка пользователей: создание службы поддержки для решения возникающих вопросов и проблем пользователей.

Мониторинг и аналитика:

- мониторинг производительности: непрерывное отслеживание производительности приложения, выявление и устранение проблем с производительностью, такими как медленная загрузка или сбои;
- сбор отзывов: организация сбора обратной связи от пользователей через встроенные функции приложения.

Обновления и улучшения:

- исправление ошибок: быстрое реагирование на обнаруженные ошибки и выпуск исправлений;
- обновления функциональности: регулярное добавление новых функций и улучшений в соответствии с отзывами пользователей и изменениями в бизнес-требованиях;
- обеспечение безопасности: регулярное обновление компонентов безопасности приложения для защиты от новых угроз и уязвимостей;
- тестирование обновлений: перед выпуском новых версий проведение тщательного тестирования всех изменений, чтобы убедиться в их стабильности и совместимости.

Эффективное развертывание и внедрение мобильного приложения требует скоординированного подхода и постоянного внимания к деталям.

Вывод по главе 3.

По итогам главы, было описано разработанное мобильное приложение службы технической поддержки, его структура, пакеты, используемые технологии и его база данных. Также было описана реализация администрирования системы учета включая все функции панели управления администратора. Были наглядно продемонстрированы функции приложения, заполняемые экранные формы, передаваемые данные и принцип администрирования приложения посредством JWT.

Заключение

Данный проект был разработан в связи с необходимостью исправления ситуации неэффективного использования рабочего времени сотрудниками отдела технической поддержки в компании. Отсутствие единой базы заявок клиентов приводило к увеличению времени на обработку данных и, как следствии, трудовых и стоимостных затрат. Решением данной проблемы являлась создание автоматизированной системы для обработки и учета заявок в службу технической поддержки.

Итогом выполненной работы является информационная система специалиста отдела технической поддержки компании, отвечающая требованиям, сформулированным при постановке задачи.

В ходе выполненной работы были достигнуты следующие результаты:

- изучена деятельность ООО «НПО ДИЗ» и его организационная структура;
- была проведена глубокая аналитика деятельности ООО "НПО ДИЗ" и изучена его организационная структура;
- описаны и проанализированы бизнес-процессы компании, после чего было выполнено концептуальное проектирование их улучшенных версий;
- проведен обзор существующих решений, подходящих для интеграции в систему предприятия;
- определены и обоснованы задачи разработки и администрирования системы учета и управления заявками в службе технической поддержки;
- разработаны модели данных, лежащие в основе предметной области, а также спроектирована база данных для мобильного приложения.
- проведено описание используемых технологий и создано мобильное приложение, полностью соответствующее требованиям и ожиданиям пользователей;

– выполнено тестирование разработанной системы, включая ее функциональность и производительность;
– произведено развертывание и внедрение мобильного приложения в систему организации.

Мобильное приложение для службы технической поддержки автоматизирует процессы учёта и управления заявками в службе технической поддержки отдела информационных технологий ООО «НПО ДИЗ». Спроектированная база данных упростила подготовку отчетов и снизила вероятность ошибок.

После внедрения мобильного приложения служба технической поддержки организации сможет обрабатывать большее количество заявок за меньшее время, что повысит общую производительность организации. Кроме того, внедрение автоматизированной системы учета и управления заявками также способствует улучшению коммуникации между системными администраторами, сотрудниками отдела технической поддержки и простыми сотрудниками организации, смежных отделов. Благодаря удобному интерфейсу и быстрому доступу к информации, исполнители и администраторы смогут оперативно решать возникающие проблемы в организации и предоставлять необходимую помощь, что повысит уровень удовлетворенности клиентов и укрепит их доверие к компании.

Список используемой литературы и используемых источников

1. Бежик А. А., Мажей Я. В. Архитектура приложений. Чем является? Для чего используется? Основные виды и критерии хорошей архитектуры // Столыпинский вестник. – 2022. – Т. 4. – №. 9. – С. 4998-5008
2. Джуба С., Волков А.Д40 Изучаем PostgreSQL 10 / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2019. – 400 с.: ил.
3. Дубейковский В. И. Практика функционального моделирования с AllFusion Process Modeler. М.: ДИАЛОГ-МИФИ, 2021. 464 с., с. 382
4. Куликов С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. - Минск: Четыре четверти. 2020. 312 с.
5. Модели As is & To be [Электронный ресурс]. URL: https://blog.bitobe.ru/article/modeli-as-is-to-be/
6. Модели данных в СУБД [Электронный ресурс]. URL: https://appmaster.io/ru/blog/modeli-dannykh-v-subd (дата обращения: 27.05.2024)
7. Моисеев А.Н., Литовченко М.И. М74 Основы языка UML: учеб. пособие. – Томск : Издательство Томского государственного университета, 2023. – 96 с
8. Нотации моделирования бизнес-процессов [Электронный ресурс]. URL: https://leanvector.ru/blog-eksperta/notatsii-modelirovaniya-biznes-protsessov/ (дата обращения: 27.05.2024)
9. Невзорова О.А., Миннегалиева Ч.Б. Основы UML / О.А. Невзорова, Ч.Б. Миннегалиева. – Казань: Казан.ун-т, 2021. – 43 с.
10. Старушенкова Екатерина Евгеньевна, Палютин Руслан Раисович Паттерн проектирования MVVM, как один из способов написания «чистого» кода в Android-приложении на Jetpack Compose // E-Scio. 2023. №4 (79). URL: https://cyberleninka.ru/article/n/pattern-proektirovaniya-mvvm-kak-odin-iz-sposobov-napisaniya-chistogo-koda-v-android-prilozhenii-na-jetpack-compose (дата обращения: 09.05.2024)

11. Сущности и связи: как и для чего системные аналитики создают ER-диаграммы [Электронный ресурс]. URL: https://practicum.yandex.ru/blog/chto-takoe-er-diagramma/ (дата обращения: 30.05.2024)

12. Тарасов С. В. СУБД для программиста. Базы данных изнутри. – 2015

13. 11. Троцкий Д. В., Городецкий В. И. Сценарная модель знаний и язык описания процессов для оценки и прогнозирования ситуаций // Тр. СПИИРАН. 2009. Вып. 8. С. 94—127

14. Что такое Java? Определение, значение и особенности [Электронный ресурс]. URL: https://appmaster.io/ru/blog/chto-takoe-java-opredelenie-znachenie-osobennosti (дата обращения: 28.05.2024)

15. Что такое нормализация базы данных [Электронный ресурс]. URL: https://wiki.merionet.ru/articles/chto-takoe-normalizaciya-bazy-dannyh (дата обращения: 20.04.2024)

16. Android 2. Программирование приложений для планшетных компьютеров и смартфонов Рето Майер, М. Эксмо, 2011, 4-8 с.

17. Android Open Source Project – Issue Tracker // Android Open Source Project – Issue Tracker URL: https://code.google.com/p/android/ (Дата обращения 10.05.2024).

18. Android Studio The Official IDE for Android // Android Studio URL: https://developer.android.com/studio/index.html (Дата обращения 10.05.2024).

19. Entity-Relationship Diagram Symbols and Notation [Электронный ресурс]. URL: https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning (дата обращения: 31.05.2024)

20. Entity-Relationship-Diagram [Электронный ресурс]. URL: https://michael-fuchs-sql.netlify.app/2021/03/03/entity-relationship-diagram-erd/#entity (дата обращения: 31.05.2024)

21. Introduction to JSON Web Tokens [Электронный ресурс]. URL: https://jwt.io/introduction (дата обращения: 31.05.2024)

22. Java Persistence API (JPA) For Database Access [Электронный ресурс]. URL: https://www.turing.com/kb/jpa-for-database-access (дата обращения: 31.05.2024)

23. Spring – что это за фреймворк и как он устроен [Электронный ресурс]. URL: https://ru.hexlet.io/blog/posts/spring-framework (дата обращения: 26.04.2024)

24. What is IDEF? [Электронный ресурс]. URL: https://www.edrawsoft.com/what-is-idef.html (дата обращения: 31.05.2024)

25. What is REST? [Электронный ресурс]. URL: https://www.codecademy.com/article/what-is-rest (дата обращения: 31.05.2024)

Приложение А

Управление заявками

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/tickets")
public class TicketController {

    private final TicketRepository ticketRepository;
    private final TicketMapper ticketMapper;
    private final UserRepository userRepository;

    @GetMapping("/{id}")
    @PreAuthorize("hasAnyAuthority({'ADMIN', 'USER'})")
    public TicketResponse getTicketById(@PathVariable UUID id) {
        Ticket ticket =
ticketRepository.findById(id).orElseThrow(EntityNotFoundException::new);
        return ticketMapper.toTicketResponse(ticket);
    }

    @GetMapping
    @PreAuthorize("hasAnyAuthority({'ADMIN', 'USER'})")
    public List<TicketResponse> getAllTickets() {
        List<Ticket> ticketsFromDb = ticketRepository.findAll();
        return ticketMapper.toListTicketResponse(ticketsFromDb);
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    @PreAuthorize("hasAnyAuthority({'ADMIN', 'USER'})")
```

```
public TicketResponse createTicket(@RequestBody TicketCreateRequest createRequest) {
        Ticket ticketForCreate = ticketMapper.toEntity(createRequest);
        ticketForCreate.setCreatedAt(LocalDateTime.now());

        String email = SecurityContextHolder.getContext().getAuthentication().getName();
        User user = userRepository.findByEmail(email).orElseThrow(EntityNotFoundException::new);
        ticketForCreate.setUser(user);

        ticketForCreate.setStatus(ACCEPTED);
        Ticket savedTicket = ticketRepository.save(ticketForCreate);
        return ticketMapper.toTicketResponse(savedTicket);
    }

    @PatchMapping("/{id}")
    @PreAuthorize("hasAnyAuthority({'ADMIN'})")
    public TicketResponse updateTicket(@PathVariable UUID id,
                                       @RequestBody TicketUpdateRequest updateRequest) {
        Ticket tickerFromRepository = ticketRepository.findById(id).orElseThrow(EntityNotFoundException::new);
        Ticket tickerUpdateEntity = ticketMapper.toEntity(id, updateRequest);

        ticketMapper.softUpdate(tickerFromRepository, tickerUpdateEntity);

        tickerFromRepository.setUpdatedAt(LocalDateTime.now());
```

```
 if (updateRequest.getExecutorId() != null) {
            User executor = userRepository.findById(updateRequest.getExecutorId()).orElseThrow(EntityNotFoundException::new);
            tickerFromRepository.setExecutor(executor);
        }

        if (tickerFromRepository.getStatus().equals(COMPLETED)) {
            tickerFromRepository.setCompletedAt(LocalDateTime.now());
        }

        Ticket updatedTicket = ticketRepository.save(tickerFromRepository);
        return ticketMapper.toTicketResponse(updatedTicket);
    }

    @PatchMapping("/close/{ticketId}")
    @PreAuthorize("hasAnyAuthority({'ADMIN', 'EXECUTOR'})")
    public TicketResponse closeTicket(@PathVariable UUID ticketId) {
        Ticket ticket = ticketRepository.findById(ticketId).orElseThrow(EntityNotFoundException::new);
        ticket.setStatus(COMPLETED);
        ticket.setCompletedAt(LocalDateTime.now());

        Ticket closedTicket = ticketRepository.save(ticket);
        return ticketMapper.toTicketResponse(closedTicket);
    }
}
```

Приложение Б

Сущность «пользователь»

```
@Getter
@Setter
@Entity
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Table(name = "users")
public class User {

    @Id
    @GeneratedValue
    @Column(name = "id", nullable = false)
    private UUID id;

    @Column(name = "name", nullable = false)
    private String name;

    @Column(name = "surname", nullable = false)
    private String surname;

    @Column(name = "middle_name")
    private String middleName;

    @Column(name = "email", nullable = false, unique = true)
    private String email;

    @Column(name = "phone", nullable = false, unique = true)
```

```
    private String phone;

    @Column(name = "password", nullable = false)
    private String password;

    @ManyToOne(fetch = FetchType.EAGER, cascade = CascadeType.DETACH)
    @JoinColumn(name = "department_id", referencedColumnName = "id")
    private Department department;

    @Column(name = "position", nullable = false)
    private String position;

    @ManyToMany(fetch = FetchType.EAGER, cascade =
CascadeType.DETACH)
    @JoinTable(name = "\"user-roles\"",
            joinColumns = @JoinColumn(name = "user_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles = new HashSet<>();

    @ToString.Exclude
    @OneToMany(mappedBy = "user", fetch = FetchType.EAGER, cascade =
CascadeType.DETACH)
    private List<Ticket> tickets;
}
```

Приложение В

Конфигурация

```
spring.datasource.driverClassName=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://localhost:5432/request-db
spring.datasource.username=postgres
spring.datasource.password=postgres

#JPA
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=true
spring.jpa.open-in-view=false

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.show_sql=true

logging.level.org.springframework.transaction.interceptor = TRACE
logging.level.com.zaxxer.hikari.pool.HikariPool=DEBUG

security.jwt.secret-
key=3cfa76ef14937c1c0ea519f8fc057a80fcd04a7420f8e8bcd0a7567c272e007b
# 1h in millisecond
security.jwt.expiration-time=3600000
```

Приложение Г

Класс авторизации и регистрации

```
@Service
public class AuthServiceImpl implements AuthService {
   private final UserServiceImpl userService;
   private final JwtTokenUtils jwtTokenUtils;
   private final AuthenticationManager authenticationManager;

   @Autowired
   public AuthServiceImpl(UserServiceImpl userService, JwtTokenUtils jwtTokenUtils, AuthenticationManager authenticationManager) {
      this.userService = userService;
      this.jwtTokenUtils = jwtTokenUtils;
      this.authenticationManager = authenticationManager;
   }


   @Override
   public ResponseEntity<?> createAuthToken(@RequestBody JwtRequest authRequest) {
      try {
         authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(authRequest.getUsername(), authRequest.getPassword()));
      } catch (BadCredentialsException e) {
         return new ResponseEntity<>(new JwtException("Невверные логин или пароль"), HttpStatus.UNAUTHORIZED);
      }
```

```
        UserDetails userDetails = userService.loadUserByUsername(authRequest.getUsername());
        String token = jwtTokenUtils.generateToken(userDetails);
        return ResponseEntity.ok(new JwtResponse(token));
    }

    @Override
    public ResponseEntity<?> createNewUser(@RequestBody RegistrationUserDto registrationUserDto) {
        if (!registrationUserDto.getPassword().equals(registrationUserDto.getConfirmPassword())) {
            return new ResponseEntity<>(new IncorrectDataException("Пароли не совпадают"), HttpStatus.BAD_REQUEST);
        }
        if (userService.findByUsername(registrationUserDto.getUsername()).isPresent()) {
            return new ResponseEntity<>(new IncorrectDataException("Пользователь с таким логинов уже зарегистрирован"), HttpStatus.BAD_REQUEST);
        }
        User user = userService.createNewUser(registrationUserDto);
        return ResponseEntity.ok(new UserDto(user.getUsername(), user.getNameOfThePosition(), user.getWorkplace()));
    }
}
```