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

Институт математики, физики и информационных технологий
(наименование института полностью)

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
(наименование профиля / специализации)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка мобильного приложения учёта заявок технической поддержки компании»

Обучающийся В.В. Рыгалов
(Инициалы Фамилия) (личная подпись)

Руководитель канд. пед. наук, доцент, О.А. Крайнова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2023

Аннотация

Название бакалаврской работы «Разработка мобильного приложения учёта заявок технической поддержки компании»:

Работа выполнена в объеме 78 страниц, включая 47 иллюстраций и 5 таблиц. Выпускная работа состоит из введения, трёх глав, заключения и списка литературы.

Тема работы связана с разработкой мобильного приложения, которое обеспечивает учет и отслеживание заявок технической поддержки компании. Работа обосновывает актуальность данной темы в контексте улучшения качества технического обслуживания сотрудников. Целью работы является разработка функционального мобильного приложения, способного оптимизировать процесс учета заявок и повысить эффективность работы службы поддержки. Задачи работы включают анализ текущего бизнес-процесса технической поддержки, разработку требований к приложению, анализ аналоговых решений, выбор платформы и средств разработки, проектирование модели данных и выбор архитектуры программного продукта.

В результате работы было разработано мобильное приложение, которое позволяет пользователям удобно создавать и отслеживать заявки на техническую поддержку. Разработанное мобильное приложение полностью соответствует поставленным требованиям и способствует оптимизации процесса работы службы поддержки, сокращению времени ответа на запросы сотрудников и повышению качества обслуживания.

Общая структура работы включает анализ бизнес-процесса, разработку требований, анализ аналогов, выбор платформы и средств разработки, проектирование модели данных.

В конце работы представлено разработанное мобильное приложение и показаны результаты функционального тестирования, которые показали, что все основные функции приложения работают корректно и соответствуют требованиям.

Abstract

The bachelor's work titled "Development of a Mobile Application for Technical Support Ticket Management" presents the following information. The thesis comprises 78 pages, including 47 illustrations and 5 tables. It consists of an introduction, three chapters, a conclusion, and a list of references. The author has no previous publications on the topic of this thesis.

The research focuses on the development of a mobile application that facilitates the management and tracking of technical support tickets within a company. The study justifies the relevance of this topic in the context of enhancing the quality of technical support provided to employees. The objective of the thesis is to develop a functional mobile application capable of optimizing the ticket management process and improving the efficiency of the support department. The tasks involved in this project include analyzing the current technical support business process, formulating application requirements, conducting a comparative analysis of existing solutions, selecting the platform and development tools, designing a data model, and choosing the software architecture.

As a result, a mobile application has been developed that enables users to create and track technical support tickets conveniently. The developed application fully meets the specified requirements and contributes to streamlining the support department's workflow, reducing response time to employee queries, and enhancing the quality of service provided.

The overall structure of the work encompasses an analysis of the business process, requirement development, comparative analysis of existing solutions, platform and development tool selection, and data model design. The final sections of the thesis present the developed mobile application and demonstrate the results of functional testing, which confirm that all core functions of the application operate correctly and meet the requirements.

Оглавление

Введение

В современных условиях эффективная организация и управление бизнес-процессами является ключевым фактором успешной работы предприятий. Одним из важных аспектов в этом процессе является учет и обработка заявок технической поддержки компании. Быстрое и качественное решение проблем сотрудников становится важным фактором в обеспечении высокого уровня стабильности работы предприятия. Однако существующие подходы к учету заявок могут быть неэффективными и неудобными в использовании. В связи с этим, разработка мобильного приложения, способного упростить и улучшить процесс учета и обработки заявок технической поддержки, является актуальной задачей.

Целью данной дипломной работы является разработка мобильного приложения учета заявок технической поддержки компании, которое позволит упростить и улучшить процесс учета и обработки заявок.

В ходе работы произведём анализ бизнес-процессов предприятия, включающий краткую характеристику самого предприятия и выбор CASE-средств для описания бизнес-процессов. Также будет проведен анализ модели текущих бизнес-процессов, включая представление их в виде диаграмм.

На основе проведенного анализа разработаем требования к мобильному приложению учета заявок технической поддержки компании. Будет проведен обзор и анализ аналогов мобильных приложений, уже существующих для учета заявок технической поддержки, для выявления преимуществ и недостатков существующих решений.

На основе сформулированных требований и проведенного анализа поставим задачу на разработку мобильного приложения учета заявок технической поддержки компании. В рамках задачи проведём проектирование приложения, включая выбор платформы реализации, средств разработки и проектирование модели данных. Также разработаем архитектуру программного продукта.

Реализация мобильного приложения будет выполнена с использованием выбранной платформы и средств разработки. После реализации приложения проведём тестирование для проверки его функциональности, стабильности и соответствия поставленным требованиям. Также напишем описание разработанного решения, включающее краткое описание функционала приложения и особенностей его реализации.

Практическая значимость работы заключается в возможности улучшения процесса учёта и обработки заявок технической поддержки, повышении эффективности работы и уровня удовлетворенности специалистов компании.

Работа была апробирована на научно-практической конференции «Студенческие Дни науки в ТГУ».

Глава 1 Анализ бизнес-процессов предприятия

1.1 Краткая характеристика предприятия

ГАУ СО «Арена» (Государственное автономное учреждение Самарской области «Арена) – это многофункциональные спортивные комплексы, которые находятся в разных городах Самарской области: спортивный комплекс «Труд», Легкоатлетический манеж в городе Тольятти, Ледовый дворец спорта «Сызрань-Арена», Спортивный комплекс «Победа» и Ледовый дворец спорта «Лада-Арена». На рисунке 1 показаны подразделения ГАУ СО «Арена».

Рисунок 1 – Структурные подразделения ГАУ СО «Арена»

Мобильное приложение требуется разработать для отдела информационных технологий, связи и защиты информации Ледового Дворца спорта «Лада-Арена», который находится по адресу: Самарская область, город Тольятти, улица Ботаническая 5.

Спортивный объект «Лада-Арена» был построен специально для хоккейной команды «Лада». Из-за технического устаревания и недостаточной вместимости (2900 мест) Дворца спорта «Волгарь», где ранее выступала команда «Лада», хоккейную команду хотели исключить из лиги КХЛ, и в 2007 году было принято решение о строительстве нового Ледового Дворца спорта «Лада-Арена». В августе 2013 года «Лада-Арена» была достроена и введена в эксплуатацию. На данный момент, вместимость Ледового Дворца составляет

6200 мест [16]. Так же «Лада-Арена» является спортивным и культурным центром города Тольятти. Ледовый дворец «Лада-Арена» - домашняя ледовая площадка хоккейной команды «Лада», он является центром подготовки молодёжной команды «Ладья» и базой ДЮСШ «Лада». Главная арена Ледового Дворца является многофункциональной и позволяет проводить не только хоккейные матчи, но и массовые катания, ледовые шоу, концерты, фестивали, чемпионаты и т.д. Организационная структура ГАУ СО «Арена» показана на рисунке 2.

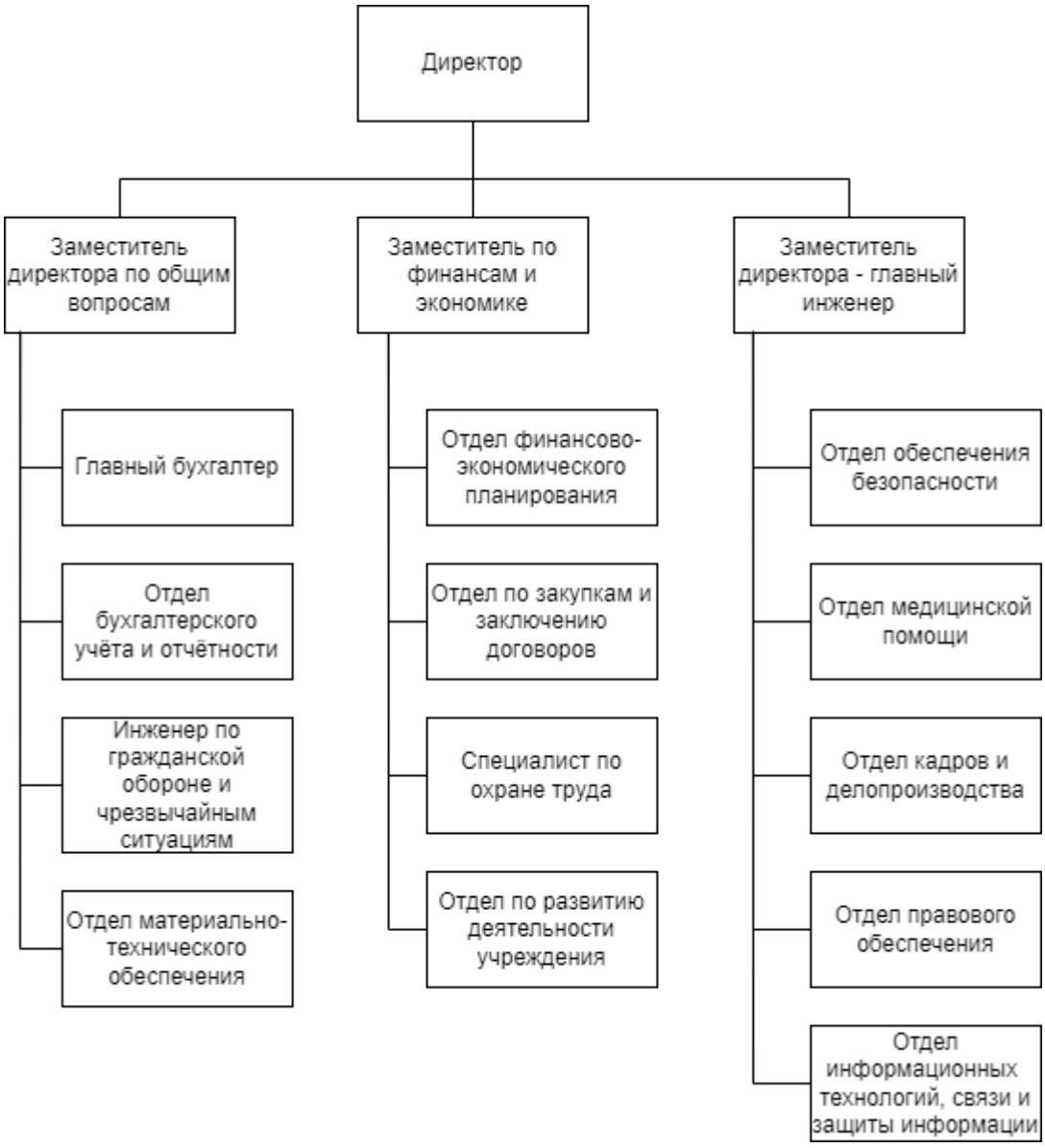


Рисунок 2 – Организационная структура ГАУ СО «Арена»

Отдел информационных технологий, связи и защиты информации Ледового Дворца спорта «Лада-Арена» отвечает за поддержку и развитие информационной инфраструктуры комплекса. В частности, отдел занимается следующими задачами:

1. Поддержка и обновление компьютерного оборудования, программного обеспечения и сетевой инфраструктуры, обеспечивающих работу всех подразделений комплекса.
2. Разработка и внедрение новых информационных систем и технологий, улучшающих работу комплекса и повышающих качество предоставляемых услуг.
3. Обеспечение безопасности информационных ресурсов комплекса, включая защиту от внешних угроз, вирусов, хакерских атак и несанкционированного доступа.
4. Организация и поддержка работы локальной сети, Wi-Fi и Интернет-соединения, необходимых для работы персонала комплекса и гостей.
5. Обеспечение бесперебойной работы информационных систем и технологий, предотвращение сбоев и аварийных ситуаций.
6. Организация и поддержка работы системы видеонаблюдения и контроля доступа на территории комплекса.
7. Проведение профилактических работ, технического обслуживания и ремонта оборудования и сетевой инфраструктуры.
8. Обучение персонала комплекса работе с информационными системами и технологиями.
9. Поддержка пользователей в вопросах работы с программным обеспечением и аппаратным обеспечением.
10. Приём заявок в техническую поддержу от сотрудников «Лада Арена».

В целом, отдел информационных технологий, связи и защиты информации Ледового Дворца спорта «Лада-Арена» играет важную роль в

обеспечении эффективной работы комплекса и обеспечении высокого уровня сервиса.

1.2 Выбор CASE-средств для описания бизнес-процессов

Выбор CASE-средства для описания бизнес-процессов является важным этапом при разработке мобильных приложений. CASE-средства предоставляют возможность описать бизнес-процессы, построить их модели и произвести анализ на основе полученных данных.

Существует множество CASE-средств, которые могут быть использованы для описания бизнес-процессов, таких как Bizagi, Lucidchart, Visual Paradigm, OpenText AppWorks, Diagrams.net и другие. Рассмотрим данные CASE-средства подробнее:

1. Bizagi Studio - CASE-средство, которое позволяет создавать модели бизнес-процессов и автоматизировать их выполнение. Bizagi Studio поддерживает стандарт BPMN 2.0, что позволяет использовать ее для моделирования сложных процессов и анализа их эффективности.

2. Lucidchart - онлайн-инструмент для создания диаграмм, который также поддерживает стандарт BPMN 2.0. Lucidchart предоставляет пользователю широкие возможности для создания рисунков, блок-схем и других диаграмм, что позволяет создавать наглядные модели бизнес-процессов.

3. Visual Paradigm - CASE-средство, которое позволяет разрабатывать модели бизнес-процессов, анализировать их истребования и на основе этого создавать программное обеспечение. Visual Paradigm обладает простым интерфейсом и поддерживает различные стандарты для моделирования бизнес-процессов.

4. OpenText AppWorks - это интегрированная платформа для создания, управления и автоматизации бизнес-процессов. Она использует стандарт

BPMN 2.0 и включает возможности для автоматизации процессов и управления ими.

5. Diagrams.net – это онлайн-инструмент для создания диаграмм и блок-схем, который позволяет легко создавать наглядные модели бизнес-процессов [2]. Редактор diagrams.net предоставляет более 50 типов диаграмм, включая BPMN 2.0 - стандарт для моделирования бизнес-процессов. Она имеет широкий набор инструментов для редактирования диаграмм, а также может быть интегрирована с различными облачными сервисами, в том числе с Firebase, используемым в разрабатываемом приложении.

Сравним данные CASE-средства по следующим критериям:

1. Функциональность: Все перечисленные сервисы предоставляют широкий набор инструментов для создания различных типов диаграмм. Lucidchart и Visual Paradigm, например, предоставляют более широкий спектр инструментов, чем Bizagi, но менее, чем Diagrams.net. OpenText AppWorks имеет узкую специализацию на BPMN диаграммах.

2. Интерфейс и удобство использования: все сервисы обладают простым и интуитивно понятным интерфейсом, который позволяет быстро создавать и редактировать диаграммы. Diagrams.net, благодаря своей простоте, позволяет более быстро начать работу с сервисом.

3. Интеграция с другими приложениями: Lucidchart и Visual Paradigm предоставляют более широкий спектр интеграций с другими приложениями, чем Bizagi и Diagrams.net. OpenText AppWorks также интегрируется с другими приложениями, но в основном с фокусом на BPMN.

4. Стоимость: Diagrams.net является бесплатным сервисом с открытым исходным кодом, в то время как Lucidchart и Visual Paradigm предлагают платные подписки, предоставляющие расширенные функциональные возможности. Bizagi и OpenText AppWorks доступны только по запросу, что может усложнить процесс получения доступа.

5. Возможности совместной работы: все перечисленные сервисы обеспечивают возможность совместной работы над диаграммами, однако

Lucidchart, Diagrams.net и Visual Paradigm предоставляют более удобные и гибкие инструменты для совместной работы.

Все вышеперечисленные свойства Программного обеспечения для наглядности были представлены в виде таблицы 1.

Таблица 1 – Свойства Программного обеспечения

	Bizagi	Lucidchart	Visual Paradigm	OpenText AppWorks	Diagrams.net
Функциональность	-	+	+	-	+
Интерфейс и удобство использования	+	+	+	+	+
Интеграция с другими приложениями	+	+	+	+	+
Стоимость	-	-	-	-	+
Возможности совместной работы	+	+	+	+	+

Для разработки мобильного приложения учёта заявок технической поддержки компании использование diagrams.net будет наиболее эффективным и удобным решением.

1.3 Анализ модели бизнес-процесса

В данном разделе проводится анализ текущей модели бизнес-процесса по учёту заявок технической поддержки в компании. На данный момент, сотрудники компании обращаются в службу поддержки по телефону или по электронной почте. Работник технического отдела принимает заявку, приступает к выполнению, а после регистрирует её в журнале.

Для более подробного анализа были построены диаграммы бизнес-процессов, описывающие текущий процесс учёта заявок технической поддержки.

Контекстная диаграмма данного процесса представлена на рисунке 3.

Рисунок 3 – Контекстная диаграмма процесса учета заявок

Для наглядности и простоты понимания декомпозиция блока A0 представлена на рисунке 4.

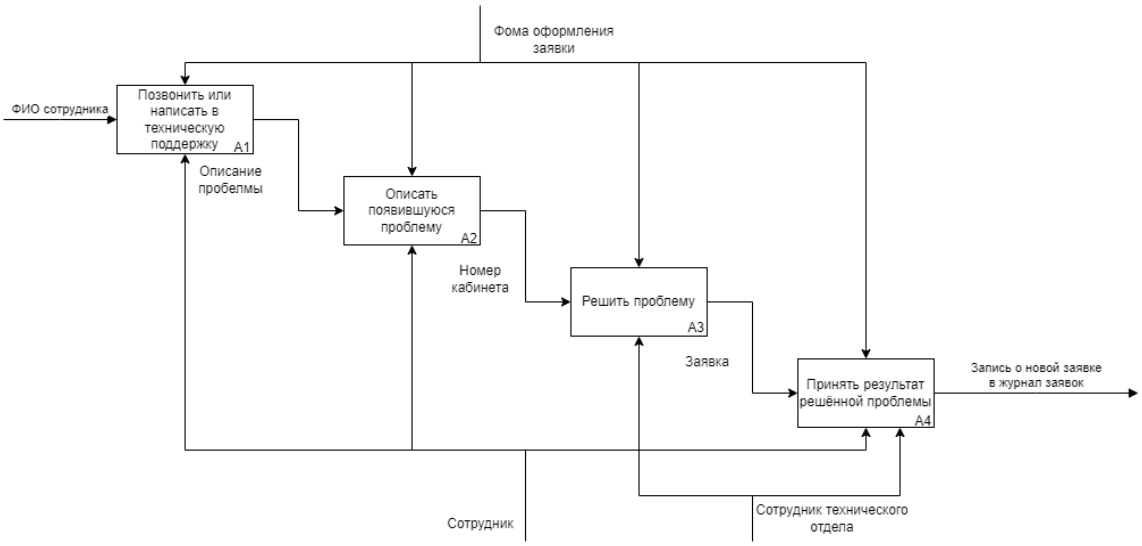


Рисунок 4 – Декомпозиция блока A0

Чтобы ещё лучше понять процесс, выполним декомпозицию блоков A1, A2 и A4, которые представлены на рисунках 5, 6 и 7 соответственно.

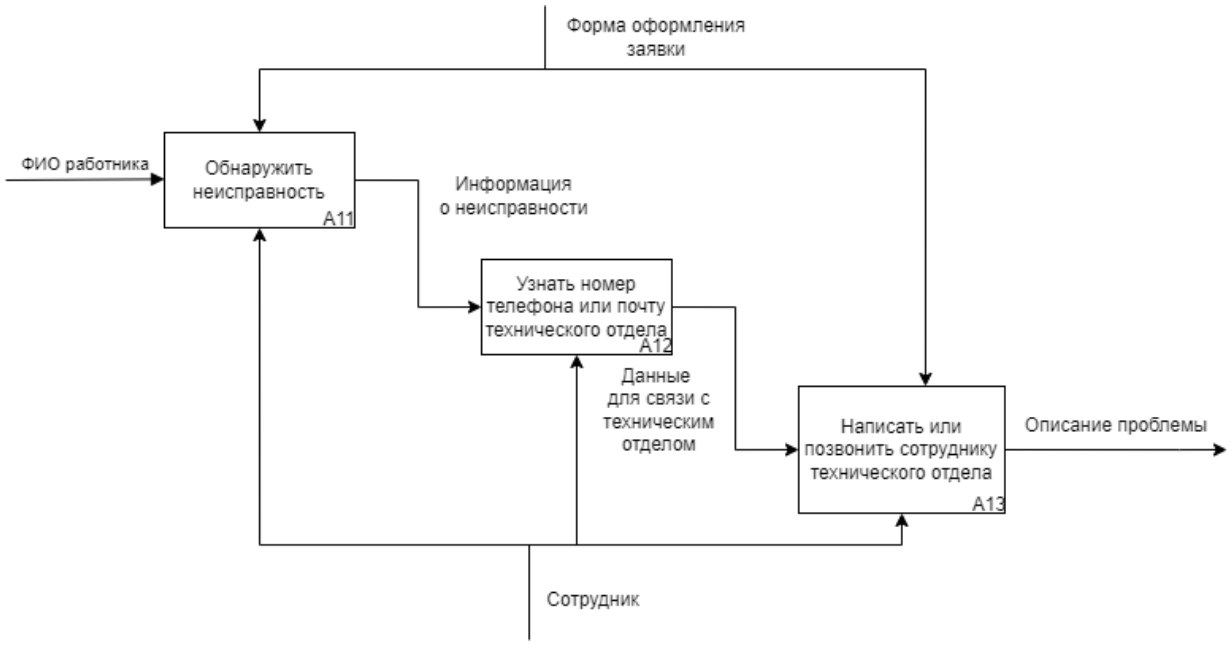


Рисунок 5 – Декомпозиция блока A1

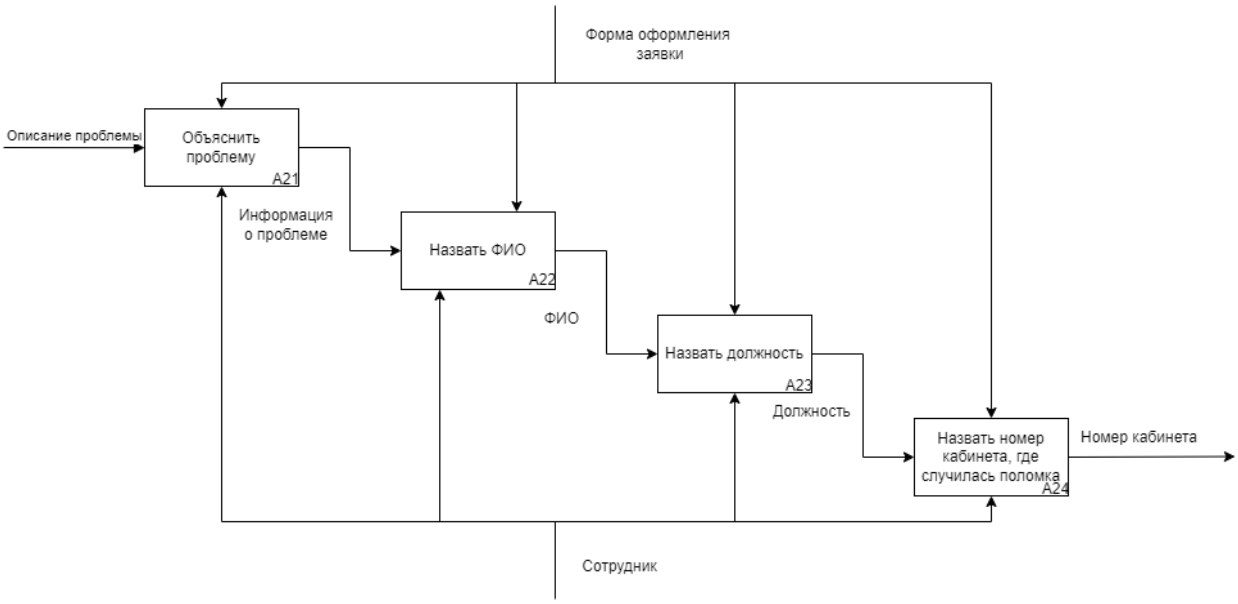


Рисунок 6 – Декомпозиция блока A2

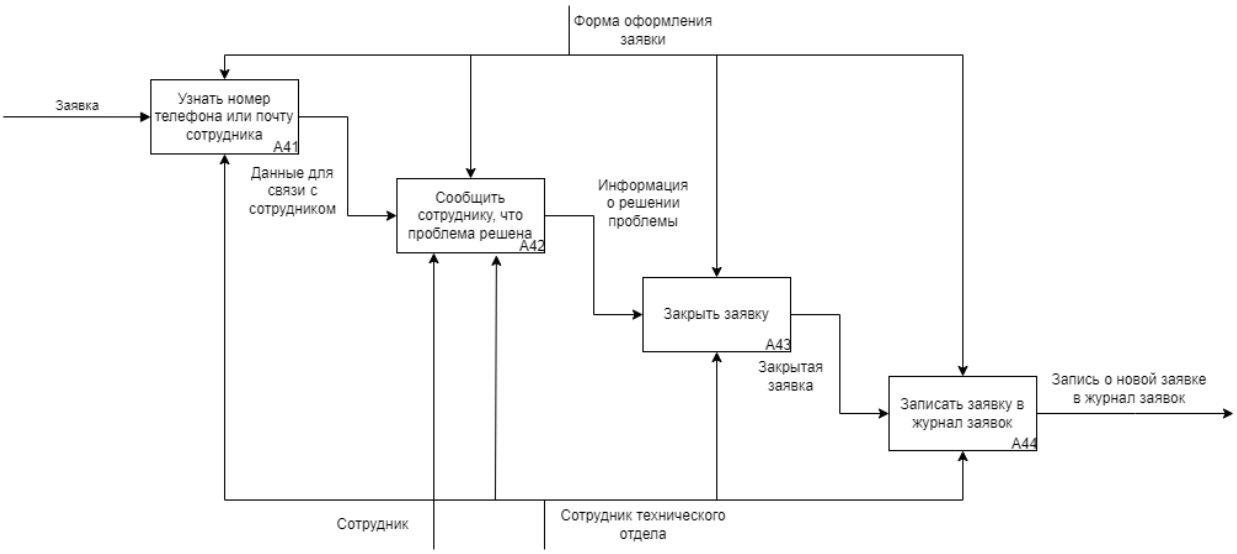


Рисунок 7 – Декомпозиция блока A4

С помощью DFD диаграммы визуально покажем процесс с точки зрения данных на рисунке 8.

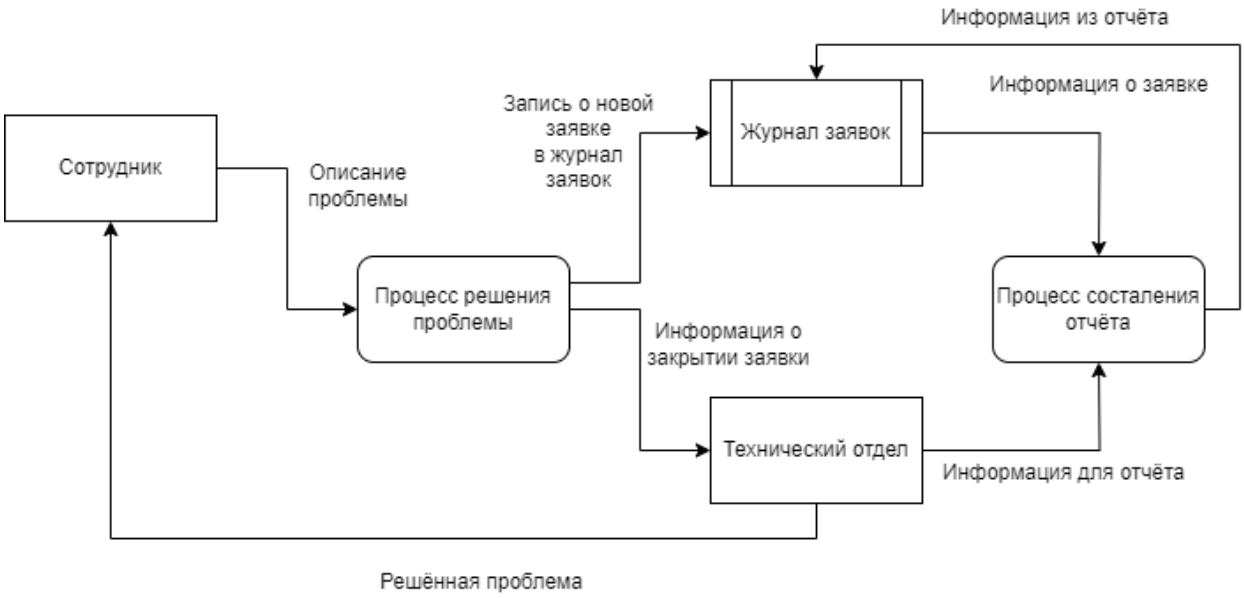


Рисунок 8 – DFD диаграммы процесса выполнения заявки

Анализ модели бизнес-процесса позволил выявить недостатки текущей системы учёта заявок технической поддержки, такие как отсутствие возможности отслеживания статуса заявки сотрудником и отсутствие уведомлений о состоянии заявки, также сам процесс оформления заявки занимает много времени. Для решения этих проблем предлагается внедрить мобильное приложение учёта заявок технической поддержки компании.

1.4 Обзор и анализ аналогов мобильных приложений для учёта заявок технической поддержки компании

Для решения задачи учета заявок технической поддержки в компаниях уже существуют различные мобильные приложения, которые могут предоставлять подобный функционал. Рассмотрим несколько аналогов и проанализируем их основные возможности и преимущества.

Freshdesk - это мобильное приложение для управления заявками в области поддержки клиентов. Оно позволяет создавать, просматривать и обновлять заявки на любом этапе их жизненного цикла. Также приложение имеет функционал мониторинга и анализа производительности технической

поддержки, что помогает компаниям оптимизировать работу своих специалистов.

Zendesk - это мобильное приложение для управления заявками технической поддержки и управления проектами. Приложение позволяет создавать и просматривать заявки, назначать задачи и отслеживать прогресс выполнения работ. Оно также имеет интеграцию с социальными сетями, что позволяет управлять запросами клиентов из разных источников в одном месте.

Jira Service Desk - это мобильное приложение для управления заявками технической поддержки и управления проектами. Оно позволяет создавать и просматривать заявки, отслеживать их выполнение и обновление, а также анализировать производительность специалистов технической поддержки. Одной из ключевых особенностей Jira Service Desk является возможность интеграции с другими приложениями и сервисами, что позволяет настраивать процессы управления заявками под нужды компании.

Сравним данные приложения по соответствию требованиям FURPS+.

FURPS+ - это метод классификации требований к программному обеспечению, который включает в себя следующие категории [3]:

- Functionality (Функциональность) - возможности программного обеспечения;
- Usability (Удобство использования) - удобство использования программного обеспечения пользователем;
- Reliability (Надежность) - стабильность работы программного обеспечения;
- Performance (Производительность) - скорость и эффективность работы программного обеспечения;
- Supportability (Поддерживаемость) - возможность обслуживания и развития программного обеспечения;
- + (Дополнительные характеристики) - другие требования, которые могут быть важны для конкретного проекта.

Ниже представлен анализ аналогов мобильных приложений учёта заявок по каждой из категорий FURPS+.

Functionality:

– Freshdesk: позволяет создавать, назначать и отслеживать заявки на поддержку, управлять базой знаний, чатом с клиентами и др. функциональностью;

– Zendesk: предоставляет возможность создания заявок, управления командой технической поддержки, мониторинга производительности и др. функциональности;

– Jira Service Desk: позволяет создавать и управлять заявками, устанавливать приоритеты, отслеживать процесс решения проблемы и др. функциональность.

Usability:

– Freshdesk: позволяет пользователям легко создавать и отслеживать заявки, используя мобильное приложение;

– Zendesk: обладает удобным интерфейсом и простым в использовании мобильным приложением;

– Jira Service Desk: простой интерфейс, удобство использования, наличие мобильного приложения.

Reliability:

– Freshdesk: обеспечивает стабильную работу приложения и минимальное время простоя;

– Zendesk: предоставляет стабильную работу и регулярные обновления для улучшения производительности;

– Jira Service Desk: обладает высокой степенью надежности и стабильности работы.

Performance:

– Freshdesk: имеет высокую скорость и эффективность работы, быстрое обновление статуса заявок и быстрый доступ к информации;

– Zendesk: позволяет быстро создавать и управлять заявками, быстро обрабатывать запросы и др. действия;

– Jira Service Desk: обеспечивает высокую скорость работы и эффективность при обработке большого количества заявок.

Supportability:

– Freshdesk: широкий функционал поддержки, обучение пользователей, документация, техническая поддержка;

– Zendesk: хороший функционал поддержки, обучение пользователей, документация, техническая поддержка;

– Jira Service Desk: широкий функционал поддержки, обучение пользователей, документация, техническая поддержка.

+ (Дополнительные характерисики): Стоимость и возможность модернизировать приложения и добавления новых функций по нуждам компании. К сожалению, все аналоги являются платными и не предусматривают возможность модернизации и добавления новых функций по нуждам компании.

Для наглядности, проанализированные данные были представлены в таблице 2.

Таблица 2 – сравнительный анализ аналогов с требованиями FURPS+

	Freshdesk	Zendesk	Jira Service Desk
Functionality	+	+	+
Usability	+	+	+
Reliability	+	+	+
Performance	+	+	+
Supportability	+	+	+
Дополнительные характеристики	-	-	-

Исходя из классификации FURPS+, можно сделать вывод, что при разработке мобильного приложения учёта заявок необходимо уделить внимание всем аспектам, описанным выше, чтобы создать надежное, быстрое и удобное для пользователя приложение, которое будет способствовать эффективной работе технической поддержки компании. Все рассмотренные приложения предоставляют основной функционал по учету заявок технической поддержки, но различаются в дополнительных возможностях и функционалах. При разработке мобильного приложения учета заявок технической поддержки компании следует учитывать преимущества и недостатки каждого из аналогов и на их основе формировать требования к приложению.

1.5 Разработка требований к мобильному приложению учёта заявок технической поддержки компании

На основе проведенного анализа бизнес-процессов предприятия, а также анализа аналогов мобильных приложений для учёта заявок технической поддержки компании с учётом классификации требований FURPS, были сформулированы следующие требования к разрабатываемому мобильному приложению:

1. Functionality:

– возможность регистрации новых пользователей;

– возможность регистрации новых заявок в системе технической поддержки через мобильное приложение, включая возможность ввода необходимой информации для создания заявки (тему проблемы и описание);

– возможность просмотра статуса заявки пользователями, в том числе изменения статуса заявки на стороне администратора;

– возможность просмотра списка всех пользователей администратором;

– возможность просмотра истории заявок пользователя в личном кабинете;

– возможность редактирование информации о себе в личном кабинете для пользователя и для администратора.

2. Usability:

– интуитивно понятный и простой интерфейс для максимального удобства использования.

3. Reliability:

– быстрое и безотказное функционирование мобильного приложения для учета заявок технической поддержки компании;

– высокий уровень безопасности и защиты конфиденциальности пользовательских данных.

4. Performance:

– высокая скорость работы приложения, включая быструю загрузку списков заявок и быстрое добавление новых заявок в систему;

– поддержка работы с большим количеством заявок и пользователей, обеспечивающая высокую производительность приложения даже при большой нагрузке.

5. Supportability:

– поддержка работы в режиме офлайн для сохранения и обработки заявок в отсутствие Интернет-соединения.

Вышеперечисленный функционал лёг в основу диаграммы вариантов использования, которая изображена на рисунке 9.

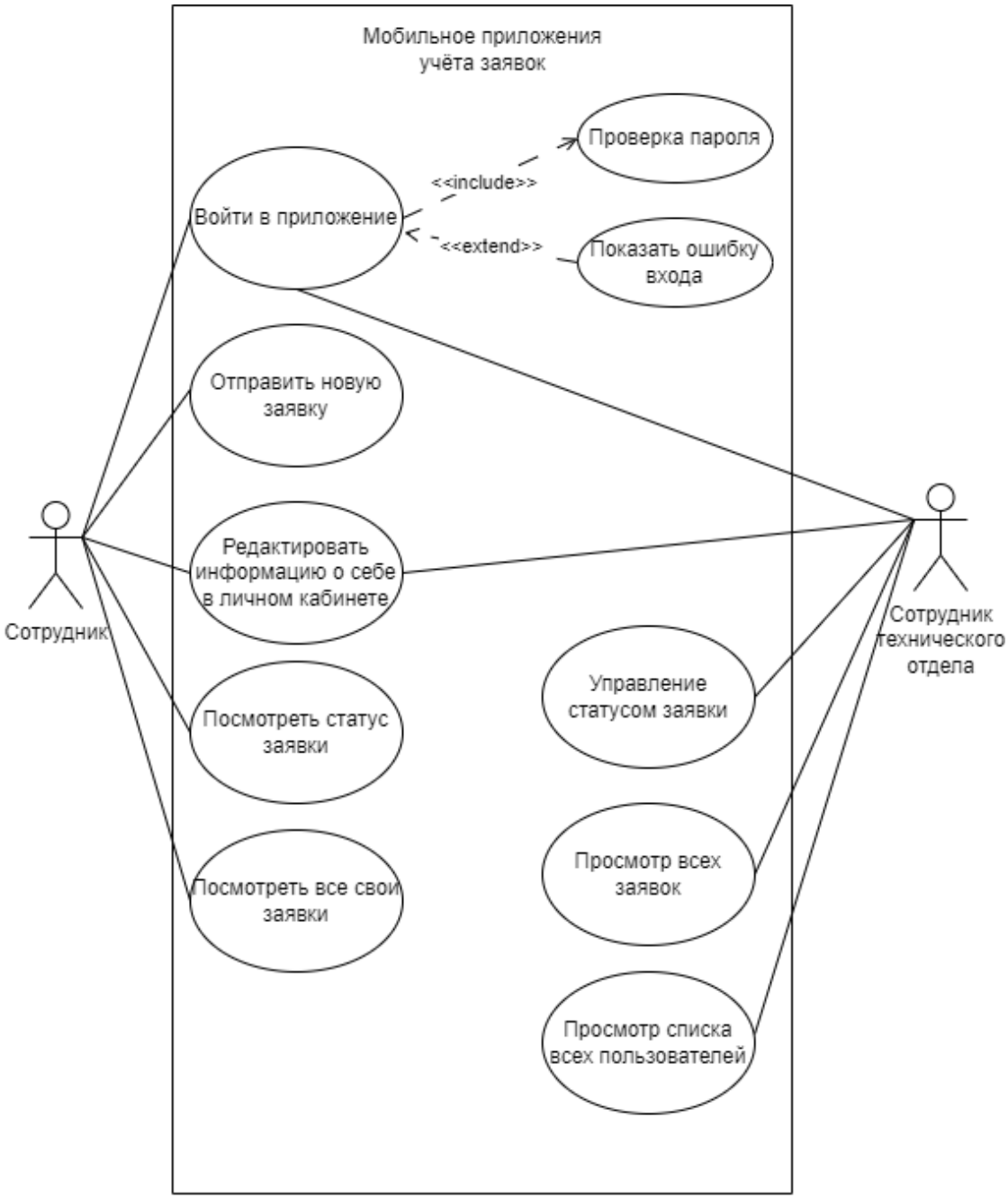


Рисунок 9 – Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой графическую модель, которая описывает функциональность мобильного приложения для учета заявок технической поддержки.

Диаграмма включает двух актеров: сотрудника и сотрудника технического отдела.

Функции для сотрудника: вход в приложение, отправка новой заявки, редактирование информации о себе в личном кабинете, просмотр статуса заявки и просмотр всех своих заявок.

Функции для сотрудника технического отдела: вход в приложение, управление статусом заявки, просмотр всех заявок, просмотр списка всех пользователей и редактирование информации о себе в личном кабинете.

Первоначальный вход в систему возможен только для зарегистрированных пользователей, то есть для сотрудников и сотрудников технического отдела, и обеспечивает доступ к функционалу, необходимому для каждой из ролей.

1.6 Постановка задачи на разработку мобильного приложения учёта заявок технической поддержки компании

Чтобы сделать процесс написания заявки более быстрым и удобным, необходимо модифицировать бизнес-процесс [17]. Для модернизации данного процесса можно использовать веб-приложение или мобильное приложение. Но приложение должно иметь важную функцию – уведомления о статусе заявки, которую нельзя реализовать в веб-приложении. Также мобильное приложение может быть удобнее в использовании, так как сотрудники могут запускать его непосредственно с мобильных устройств, а не заходить в браузер и вводить адрес сайта. Таким образом, мобильное приложение учёта заявок может предоставить ряд преимуществ перед веб-приложением, которые могут улучшить пользовательский опыт и общую эффективность использования системы учёта заявок.

Внедрение мобильного приложения оптимизирует работу сотрудника технического отдела и упростит обращение других сотрудников в поддержку.

Контекстная диаграмма процесса с мобильным приложением показана на рисунке 10.

Рисунок 10 – Контекстная диаграмма процесса с мобильным приложением

Для лучшего понимания и восприятия декомпозиция блока A0 представлена на рисунке 11.

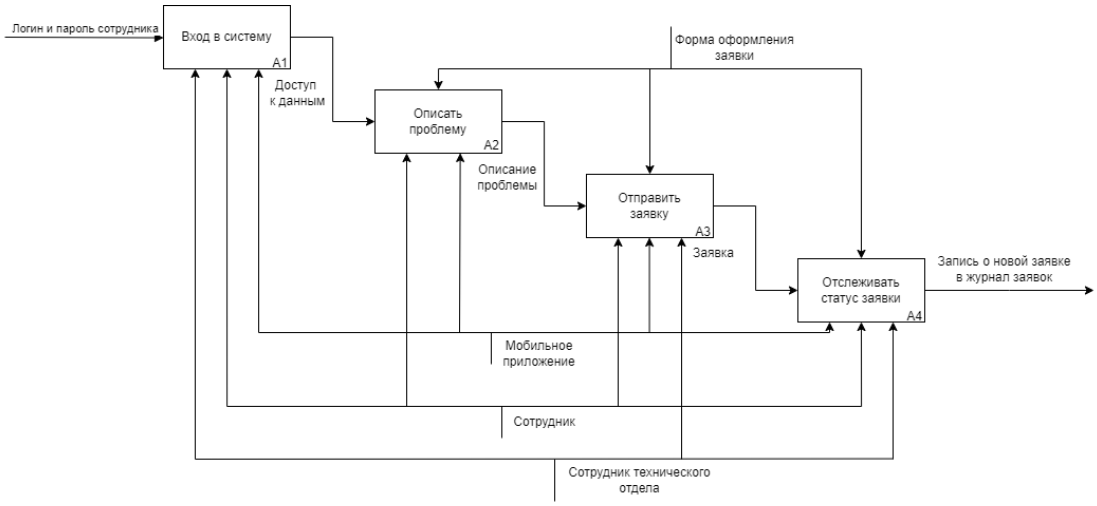


Рисунок 11 – Декомпозиция блока A0

Для наглядности и простоты понимания, выполним декомпозицию блоков А2, А3 и А4, которые представлены на рисунках 12, 13 и 14 соответственно.

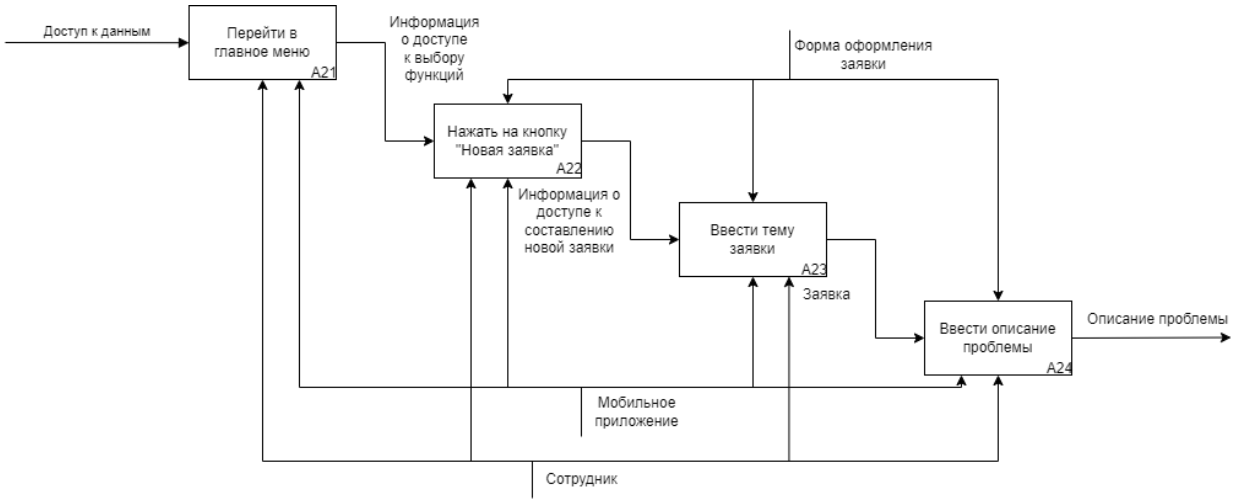


Рисунок 12 – Декомпозиция блока А2

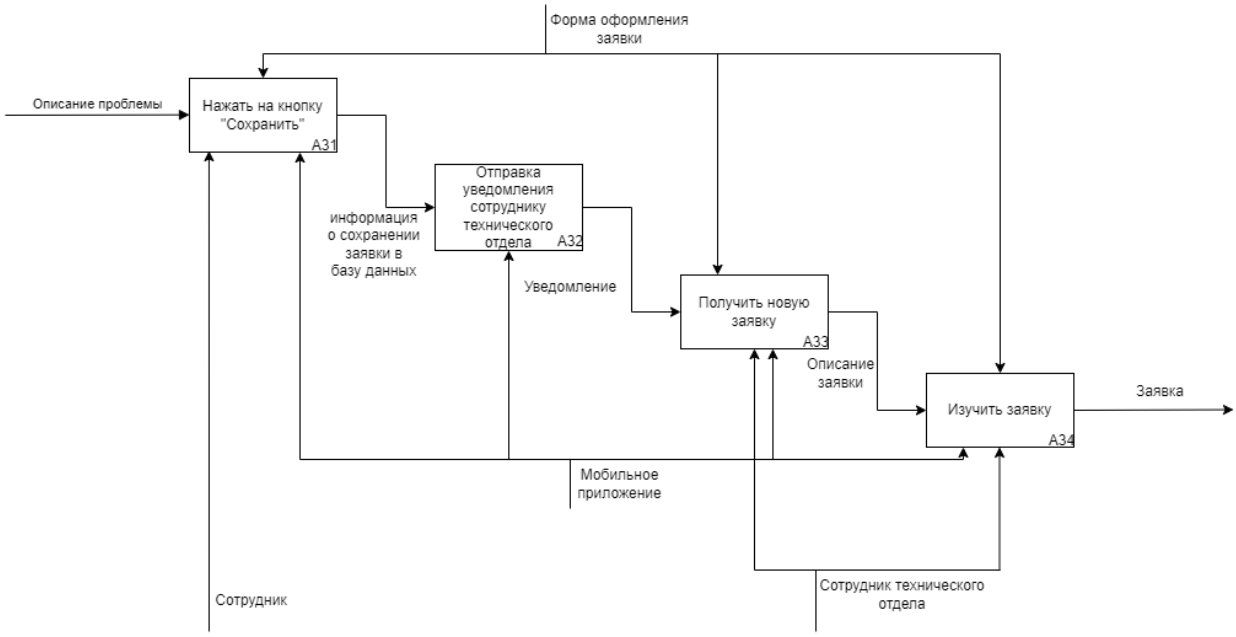


Рисунок 13 – Декомпозиция блока А3

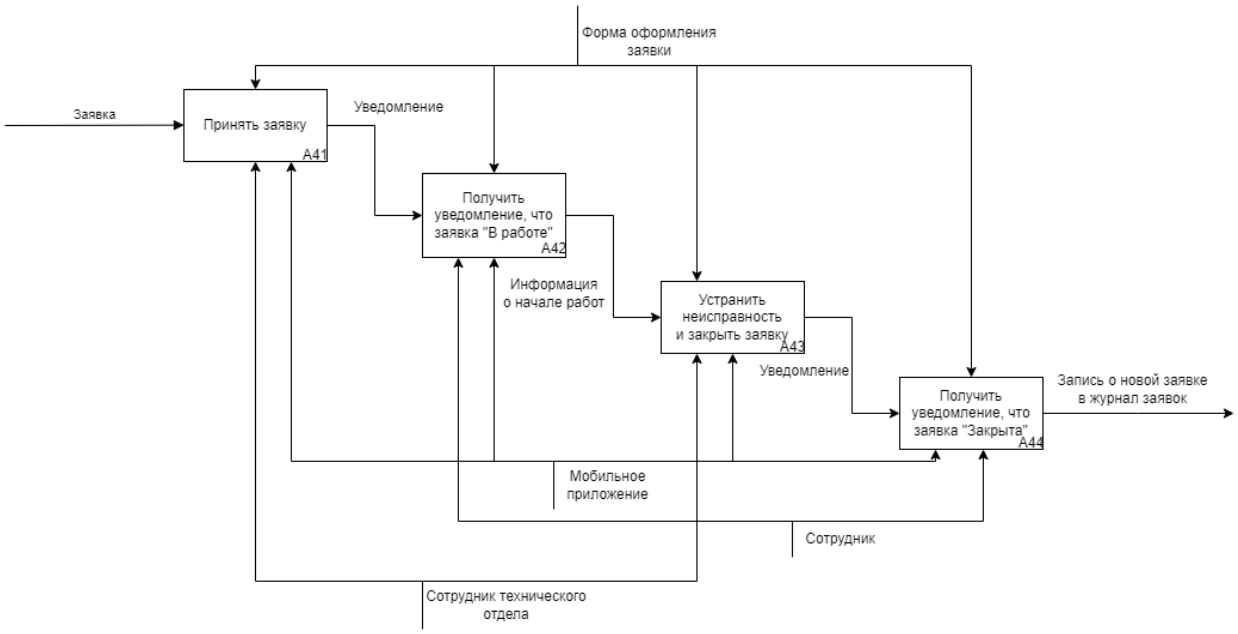


Рисунок 14 – Декомпозиция блока A4

С помощью DFD диаграммы покажем визуальное взаимодействие данных в процессе на рисунке 15.

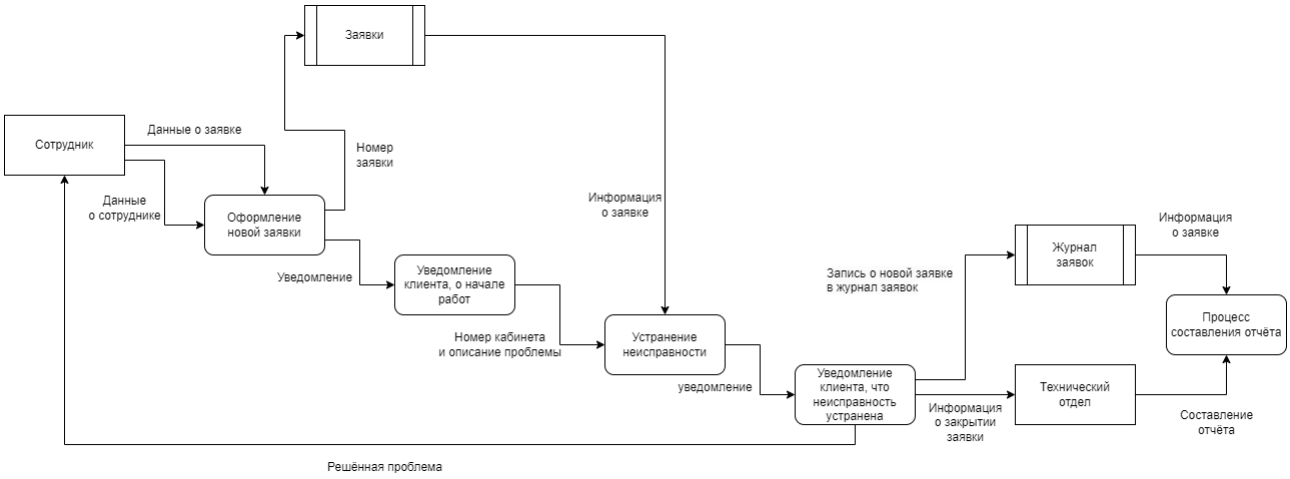


Рисунок 15 – DFD диаграмма визуального взаимодействия

Необходимо создать мобильное приложение учёта заявок технической поддержки компании, которое позволит сотрудникам компании легко и

удобно отправлять заявки на техническую поддержку, а также отслеживать их статус.

Приложение должно позволять сотрудникам оставлять заявки через удобный интерфейс с помощью мобильного устройства. Сотрудники должны иметь возможность написать тему проблемы и описать ее в свободной форме.

После отправки заявки сотрудники должны получать уведомления о статусе ее выполнения и об изменениях в статусе заявки.

Также сотрудники должны иметь возможность просматривать все свои отправленные заявки и изменять информацию о себе в личном кабинете.

Для сотрудников технической поддержки должен быть создан удобный интерфейс для работы с заявками. Сотрудники технической поддержки должны иметь возможность просматривать все заявки, которые поступили, и отмечать их статусы. Кроме того, сотрудники технической поддержки должны иметь возможность отправлять уведомления сотрудникам о статусе выполнения заявки.

Важно также обеспечить защиту персональных данных сотрудников и сохранность данных о заявках. Для этого необходимо использовать современные методы шифрования данных и механизмы аутентификации и авторизации пользователей.

Интерфейс приложения должен быть удобным и интуитивно понятным, чтобы сотрудники могли быстро и без проблем отправлять заявки на техническую поддержку. Приложение должно работать на мобильных устройствах с операционной системой Android.

Данное приложение сделает пользование услугами технической поддержки более продуктивным, так как сотрудники смогут меньше тратить времени на оформление заявки и объяснение произошедшей проблемы.

Выводы по главе 1

В рамках первой главы была проведена краткая характеристика предприятия и определены задачи отдела информационных технологий, связи и защиты информации. Для концептуального моделирования было выбрано наиболее подходящее CASE-средство, после чего была показана текущая модель бизнес-процесса. Далее был произведен сравнительный анализ аналогов мобильного приложения учета заявок технической поддержки, разработаны требования к мобильному приложению, а также представлена оптимизированная модель бизнес-процесса. В заключении главы были сформулированы задачи на разработку мобильного приложения.

Глава 2 Проектирование мобильного приложения учёта заявок технической поддержки компании

2.1 Выбор платформы реализации мобильного приложения

При выборе мобильной операционной системы для приложения технической поддержки, необходимо учитывать ряд факторов, таких как функциональность, удобство использования, безопасность, стабильность, популярность и доступность.

Пользователи в настоящее время имеют широкий выбор мобильных операционных систем, предоставляющих доступ к различным сервисам и функциональности. Ниже приведены наиболее популярные мобильные операционные системы:

1. Android - это операционная система для мобильных устройств, разработанная Google. Android является самой популярной мобильной платформой в мире, с долей рынка 68,61% на конец апреля 2023 года. Она предлагает широкий выбор устройств и возможностей для разработчиков [7];

2. iOS - это мобильная операционная система, разработанная Apple для своих устройств, таких как iPhone и iPad [5]. Она имеет долю рынка около 30,61% на конец апреля 2023 года. iOS обеспечивает безопасность и надежность, а также предлагает богатый пользовательский опыт;

3. Windows Phone - это операционная система для мобильных устройств, разработанная Microsoft. Она имеет долю рынка около 0,2% на конец апреля 2023 года. Windows Phone предлагает удобный интерфейс и интеграцию с другими продуктами Microsoft;

4. Blackberry OS - это операционная система для мобильных устройств, разработанная Blackberry. Она имеет долю рынка около 0,1% на конец апреля 2023 года. Blackberry OS предлагает высокий уровень безопасности и надежности.

Для большей наглядности на рисунке 16 изображён график популярности мобильных операционных систем [6].

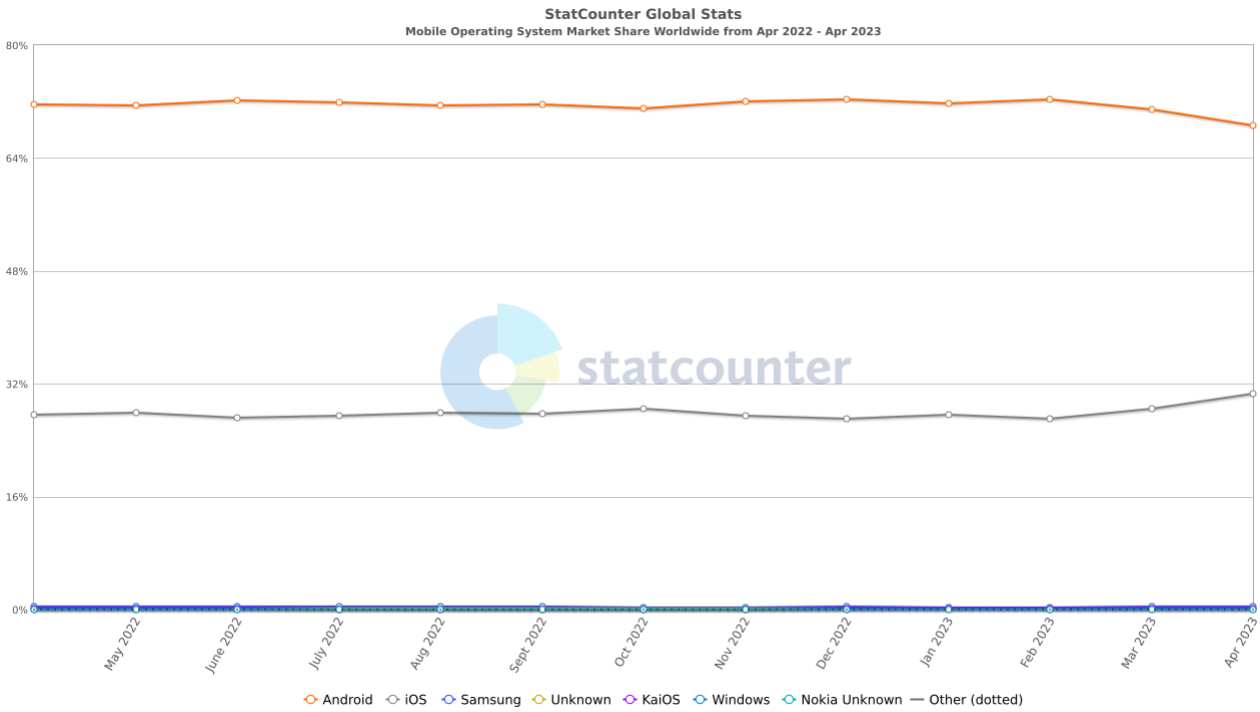


Рисунок 16 – График популярности мобильных операционных систем

Сравним данные мобильные операционные системы по следующим критериям: функциональность, удобство использования, безопасность, стабильность, популярность и доступность [13].

1. Функциональность:

– Android: предлагает обширный набор функций и возможностей, позволяющих разработчикам создавать разнообразные приложения и настраивать интерфейс под свои нужды;

– iOS: также предлагает широкий набор функциональных возможностей, но более строго контролирует доступ разработчиков к некоторым системным функциям и инструментам;

– Windows Phone: обладает функциональностью, сравнимой с Android и iOS, но может быть немного ограничен в некоторых аспектах, таких как доступность приложений;

– BlackBerry OS: имеет свои уникальные функции, такие как высокая безопасность и интеграция с корпоративными сервисами, но может быть менее разнообразным в функциональном плане.

2. Удобство использования:

– Android: Интерфейс интуитивно понятен и привычен для пользователей, особенно для тех, кто привык к использованию продуктов Google;

– iOS: имеет простой и интуитивно понятный пользовательский интерфейс, что делает его легким в использовании для новых пользователей. Он предлагает стандартизированное взаимодействие и дизайн приложений;

– Windows Phone: Windows Phone также имеет простой и стильный пользовательский интерфейс, но может отличаться от привычного пользовательского опыта в Android и iOS;

– BlackBerry OS: имеет свою собственную систему навигации и управления, которая может быть несколько отличной от других операционных систем.

3. Безопасность:

– Android: имеет мощные механизмы безопасности, включая систему разрешений, обновления безопасности и встроенную защиту от вредоносного ПО;

– iOS: известен своей высокой безопасностью, благодаря строгим правилам и контролю со стороны Apple над приложениями и доступом к системным ресурсам;

– Windows Phone: также предлагает некоторые механизмы безопасности, но из-за низкой популярности и ограниченной доступности сторонних приложений, экосистема Windows Phone менее подвержена атакам и вирусам;

– BlackBerry OS: славится своей высокой безопасностью, которая делает его популярным среди корпоративных пользователей.

4. Стабильность:

– Android: Стабильность Android-устройств может варьироваться в зависимости от производителя и модели. В некоторых случаях на различных устройствах могут возникать проблемы с производительностью или несовместимостью. Однако, с постоянным развитием платформы и улучшением аппаратного и программного обеспечения, стабильность Android-устройств постоянно улучшается;

– iOS: известен своей стабильностью и плавной работой на всех устройствах Apple. Благодаря контролируемой экосистеме и оптимизации операционной системы для конкретных устройств, iOS обычно обеспечивает высокую стабильность и производительность;

– Windows Phone: Windows Phone также предлагает стабильную работу, особенно на устройствах, разработанных Microsoft. Однако, с учетом того, что Windows Phone больше не активно поддерживается, новые обновления и исправления могут быть ограничены;

– BlackBerry OS: BlackBerry OS изначально разрабатывалась с упором на стабильность и надежность. Эта операционная система обладает репутацией высокой стабильности, особенно в корпоративной среде.

5. Популярность и доступность:

– Android: Android является наиболее популярной мобильной операционной системой в мире (68,61%). Его широкая распространенность обеспечивает доступность множества устройств различных производителей и различных ценовых категорий;

– iOS: iOS также имеет значительную популярность (30,61%), особенно среди пользователей Apple, которые предпочитают экосистему этой компании. Однако, доступность iOS ограничена только на устройствах Apple;

– Windows Phone: Windows Phone имеет гораздо меньшую популярность по сравнению с Android и iOS (0,1%). Это может отразиться на доступности устройств и приложений;

– BlackBerry OS: BlackBerry OS также имеет ограниченную популярность, особенно в широких массах пользователей (0,2%). Это может повлиять на доступность и поддержку приложений.

Ниже приведена таблица 3 для сравнения вышеперечисленных мобильных операционных систем.

Таблица 3 – Сравнение мобильных операционных систем

	Android	IOS	Windows Phone	BlackBerry OS
Функциональность	+	+	-	-
Удобство использования	+	+	+	+
Безопасность	+	+	+	+
Стабильность	+	+	+	+
Популярность и доступность	+	-	-	-

Сравнение различных операционных систем показывает, что Android является более гибкой и открытой платформой, что позволяет разработчикам более свободно работать с аппаратным обеспечением устройств и более быстро выпускать обновления и исправления ошибок. Кроме того, Android имеет большое количество устройств разных производителей, вследствие привлекает большее количество пользователей.

2.2 Выбор средств разработки

Android Studio был выбран в качестве среды разработки для создания мобильного приложения технической поддержки по ряду причин:

1. Официальная среда разработки: Android Studio является официальной интегрированной средой разработки (IDE) для платформы Android, разработанной и поддерживаемой компанией Google. Благодаря этому, она обладает широким набором инструментов, ресурсов и обновлений, специально созданных для разработки приложений под Android [1];

2. Интуитивный интерфейс: Android Studio предоставляет простой и интуитивно понятный пользовательский интерфейс, что упрощает работу. Интуитивная навигация, автозаполнение кода и встроенные инструменты позволяют эффективно разрабатывать приложения;

3. Мощные инструменты и функциональность: Android Studio предлагает широкий спектр инструментов и функций, которые значительно облегчают процесс разработки. Это включает в себя эмуляторы устройств для тестирования приложений, визуальные редакторы пользовательского интерфейса, отладочные инструменты;;

4. Поддержка языка программирования Java: Android Studio предоставляет полную поддержку языка программирования Java, который является официальным языком разработки приложений под Android. Java является широко используемым языком с огромным сообществом разработчиков, множеством ресурсов и библиотек, что облегчает процесс разработки;

5. Интеграция с другими инструментами: Android Studio интегрируется с другими инструментами и службами разработки, такими как Firebase, которая предоставляет широкий набор облачных услуг для разработки мобильных приложений. Это позволяет использовать Firebase для упрощения работы с базой данных, аутентификацией пользователей, обработкой уведомлений и другими функциями, что существенно ускоряет процесс разработки.

Исходя из этих преимуществ, выбор Android Studio является логичным и обоснованным для разработки мобильного приложения технической поддержки.

Выбор языка программирования Java для разработки мобильного приложения технической поддержки осуществлен по следующим причинам [9]:

1. Поддержка от Android Studio: Android Studio, основная интегрированная среда разработки для платформы Android, обладает прекрасной поддержкой Java. Инструменты, редакторы и отладчики Android Studio предоставляют широкие возможности для разработки приложений на языке Java и облегчают процесс разработки;
2. Опыт: уже имеется опыт работы с языком программирования Java по учебному курсу университета. Это упрощает процесс разработки, позволяет использовать уже накопленный опыт и ускоряет процесс освоения различных Android-специфичных инструментов и фреймворков;
3. Доступность ресурсов и документации: Java имеет обширную базу ресурсов и документации, включая официальную документацию от Oracle, множество книг, учебных материалов и онлайн-ресурсов. Это обеспечивает легкий доступ к информации и помощи при разработке приложения;
4. Стабильность и надежность: Java является одним из самых надежных и стабильных языков программирования, с многолетней историей применения в широком спектре проектов. Это обеспечивает стабильность и надежность разработанного приложения[12].

В целом, выбор Java в данном случае обоснован опытом, доступностью ресурсов и поддержкой Android-платформы, что обеспечит более комфортное и эффективное разработку мобильного приложения технической поддержки компании.

Firebase был выбран в качестве инструмента разработки для мобильного приложения технической поддержки по ряду причин [4]:

1. Облачные услуги: Firebase предоставляет широкий набор облачных услуг, которые значительно упрощают разработку мобильных приложений. Он предлагает готовые решения для различных функций, таких как аутентификация пользователей, база данных в реальном времени, хостинг,

облачные функции, хранение файлов, уведомления и многое другое. Благодаря этому, Firebase позволяет сосредоточиться на основной функциональности приложения, минимизируя необходимость в разработке и настройке собственных серверных компонентов;

2. Простота интеграции: Firebase легко интегрируется с платформой Android и предоставляет удобные инструменты для быстрой и безболезненной интеграции с мобильными приложениями. Firebase SDK (Software Development Kit) предоставляет готовые API и компоненты, которые позволяют взаимодействовать с облачными услугами Firebase;

3. Реальное время и синхронизация данных: Firebase Realtime Database предоставляет мощную базу данных в реальном времени, которая автоматически синхронизируется с клиентскими приложениями. Это позволяет создавать приложения, которые мгновенно реагируют на изменения данных и обеспечивают синхронизацию между разными устройствами и платформами;

4. Масштабируемость и надежность: Firebase обеспечивает высокую масштабируемость и надежность своих облачных услуг. Использование Firebase позволяет избежать заботы о серверной инфраструктуре, масштабировании и обслуживании, поскольку все это обеспечивается Firebase;

5. Документация: Firebase имеет большое количество документации, учебных ресурсов, сообществ и форумов, что позволяет достаточно легко научиться работать в данной системе.

В целом, выбор Firebase для разработки мобильного приложения технической поддержки обусловлен его широкими возможностями, простотой интеграции, надежностью.

OneSignal был выбран для отправки уведомлений в приложении учёта заявок технической поддержки по нескольким причинам [8]:

1. Простота интеграции: OneSignal предоставляет простой и удобный способ интеграции сервиса уведомлений в мобильное приложение. Он

предоставляет набор готовых библиотек и SDK для различных платформ, включая Android, что значительно упрощает процесс интеграции в приложение;

2. Мощные функциональные возможности: OneSignal предлагает широкий набор функций для отправки уведомлений. Это включает гибкую настройку внешнего вида уведомлений, возможность отправки уведомлений по геолокации или группам пользователей, поддержку пуш-уведомлений с различными типами контента (текст, изображения, звуки и т. д.), а также возможность отправки уведомлений на различные платформы (Android, iOS, веб);

3. Масштабируемость и надежность: OneSignal обеспечивает высокую масштабируемость и надежность в доставке уведомлений. Сервис имеет распределенную инфраструктуру и использует различные каналы доставки, чтобы гарантировать, что уведомления будут доставлены на целевые устройства надежно и вовремя.

Исходя из этих причин, OneSignal был выбран в качестве решения для отправки уведомлений в приложении учёта заявок технической поддержки, так как он сочетает в себе простоту интеграции, мощные функциональные возможности, надежность и масштабируемость.

2.3 Проектирование модели данных

Инфологическое проектирование - это процесс разработки внешней (инфологической) модели информации о предметной области, которая не зависит от физических аспектов ее реализации. Цель инфологического моделирования состоит в предоставлении понятных и удобных для человека способов сбора и представления информации, которая будет храниться в базе данных [18].

Существуют различные методы разработки внешних моделей, такие как смысловые сети, язык инфологического моделирования и ER-схемы. Однако, самым популярным подходом является моделирование сущности-связи (Entity-Relationship), который включает следующие основные компоненты:

1. Сущность: это различимый и заметный объект, информация о котором должна быть хранится в базе данных;

2. Атрибут: представляет характеристики именованного объекта. Каждый атрибут должен быть уникальным для определенного вида объекта, но может быть общим для разных видов объектов;

3. Связь: представляет собой связь или отношение между двумя или более объектами. Она указывает на взаимосвязь и зависимость между объектами.

Концептуальная модель, в свою очередь, демонстрирует информационные сущности, их особенности и взаимосвязи без учета физической реализации данных. Она является моделью предметной области и служит основой для дальнейшего проектирования и реализации базы данных. На рисунке 17 представлена концептуальная модель.

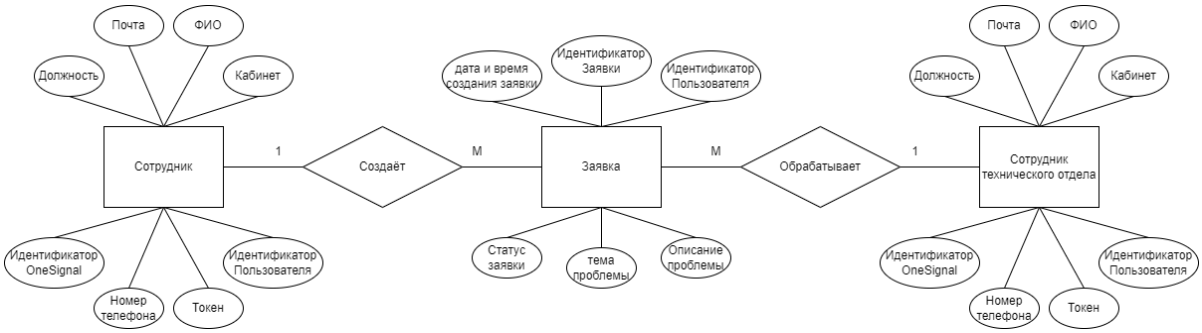


Рисунок 17 – Концептуальная модель данных

Объекты и атрибуты, предложенные в концептуальном модели, были определены после проведения исследования предметной области.

Данные о сотруднике и сотруднике технического отдела будут находиться в одной таблице, но в концептуально модели разделены, т.к. выполняют разные действия с заявкой.

Логическая модель данных отражает логические взаимосвязи между информационными компонентами, независимо от их содержимого и условий сохранения. Она представляет собой шаблон будущей информационной системы, разработанный на основе информационных концепций и не привязанный к конкретному хранилищу данных. В логической модели данных основными структурными компонентами являются объекты, их взаимосвязи и свойства (атрибуты) [15].

Логическая модель данных представлена на рисунке 18. При создании будет использоваться методология IDEF1X.

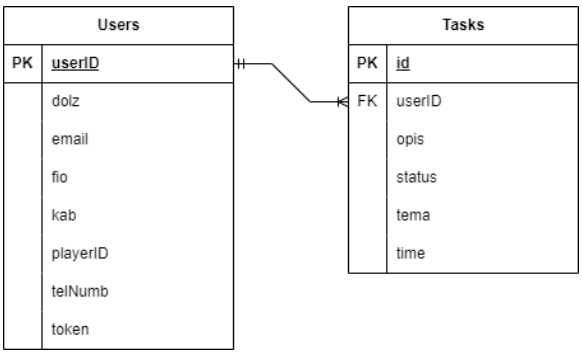


Рисунок 18 - Логическая модель данных

В таблице 4 описаны связи, представленные на рисунке 18.

Таблица 4 – Связи данных

Ключ связи	Таблица	Подчинённая таблица	Связь
userID (уникальный идентификатор пользователя)	Users (пользователи)	Tasks (Заявки)	1:M

На основе логической модели данных была простроена физическая модель данных.

Физическая модель данных построена по принципу логической модели и представляет информацию с использованием Firebase Realtime Database в качестве базы данных. Связи, созданные на логическом уровне, могут быть отражены в структуре пути и ключей в Firebase Realtime Database, а атрибуты могут быть представлены в виде значений данных, хранящихся по определенным ключам. Физическая модель данных, специфическая для Firebase Realtime Database, обеспечивает синхронизацию и обновление данных в режиме реального времени. Физическая модель данных представлена на рисунке 19.

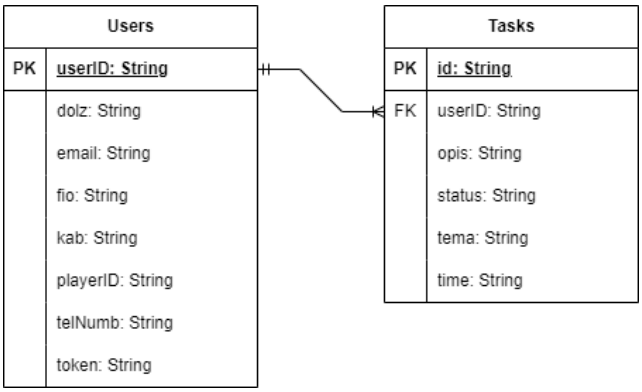


Рисунок 19 – Физическая модель данных

Ниже в таблице 5 представлена структура таблиц.

Таблица 5 – Структура таблиц

Номер	Название поля	Тип поля	Описание
	Users (Пользователи)		
1	userID	String	ID пользователя (уникальный идентификатор пользователя)
2	dolz	String	Должность (строка, указывающая на должность пользователя)
3	email	String	Почта (строка, содержащая адрес электронной почты пользователя)
4	fio	String	ФИО (строка, содержащая полное имя пользователя)
5	kab	String	Номер кабинета (строка, указывающая на номер кабинета пользователя)
6	playerID	String	ID OneSignal (уникальный идентификатор пользователя в системе OneSignal)
7	telNumb	String	Номер телефона (строка, содержащая номер телефона пользователя)
8	token	String	Токен устройства (строка, содержащая уникальный идентификатор устройства пользователя)
	Tasks (Заявки)		
1	id	String	ID заявки (уникальный идентификатор заявки)
2	userID	String	ID пользователя (идентификатор пользователя, связанный с данной заявкой)
7	opis	String	Описание проблемы (строка, содержащая описание проблемы, связанной с данной заявкой)
9	status	String	Статус заявки (строка, указывающая на текущий статус заявки)
10	tema	String	Тема проблемы (строка, указывающая на тему проблемы, связанной с данной заявкой)
11	time	String	Время отправления заявки (дата и время, когда заявка была отправлена)

Описанные спецификации таблиц будут использоваться при создании структуры базы данных в рамках физической модели данных. Физическая модель данных определяет, как информация будет храниться, организована и обрабатываться в конкретной СУБД, такой как Firebase Realtime Database. Она учитывает особенности и возможности выбранной СУБД и предоставляет

инструкции по созданию таблиц, определению полей и их типов данных, а также устанавливает связи между таблицами, если это применимо. Таким образом, физическая модель данных служит основой для фактической реализации базы данных с использованием конкретной СУБД [19].

2.4 Архитектура программного продукта

Существует несколько видов архитектуры, которые могут быть использованы при разработке мобильных приложений[11]. Наиболее распространённые виды архитектуры мобильных приложений представлены ниже:

1. MVC (Model-View-Controller): Это классическая архитектура, которая разделяет приложение на три основных компонента: модель (хранение данных и бизнес-логику), представление (отображение пользовательского интерфейса) и контроллер (управление взаимодействием между моделью и представлением). MVC обеспечивает разделение ответственностей и улучшает модульность и переиспользуемость кода;

2. MVP (Model-View-Presenter): Эта архитектура основана на концепции MVC, но с некоторыми изменениями. Основное отличие заключается в том, что Presenter берет на себя большую часть ответственности за управление взаимодействием между моделью и представлением. MVP облегчает тестирование и обеспечивает лучшую разделение логики и представления;

3. MVVM (Model-View-ViewModel): Эта архитектура также основана на концепции MVC, но включает в себя дополнительный компонент - ViewModel. ViewModel служит связующим звеном между моделью и представлением, предоставляя данные и методы для их обработки. MVVM улучшает разделение интерфейса пользователя и бизнес-логики, а также облегчает тестирование и поддержку кода;

4. Clean Architecture: Это архитектурный подход, который стремится создать приложение, которое легко поддерживать, тестируемое и независимое от конкретных фреймворков или библиотек. Он использует принципы разделения интерфейсов и зависимостей, разделяя приложение на слои с четкими границами. Clean Architecture облегчает масштабирование приложения и его развитие в долгосрочной перспективе;

5. Reactive Architecture: Это архитектурный подход, который строится на идее реактивного программирования. Он основан на потоковой обработке данных и реакции на события. Реактивная архитектура позволяет создавать отзывчивые и масштабируемые приложения, которые легко адаптируются к изменяющимся условия;

6. «Основанное на стандартных компонентах Android»: Это означает, что код и компоненты приложения могут быть организованы в соответствии с принятными практиками разработки для платформы Android, но без строгого применения конкретного архитектурного паттерна.

Архитектура разработанного мобильного приложения основана на стандартных компонентах Android. Приложение организовано в соответствии с принятыми практиками разработки для платформы Android, используя предоставляемые Android SDK компоненты и функциональность. В рамках разработки были использованы активности, элементы интерфейса, обработчики событий, а также взаимодействие с Firebase и OneSignal для обеспечения функциональности учета заявок технической поддержки компании. Организация кода и компонентов приложения позволяет эффективно управлять его функциональностью, взаимодействием с пользователями и внешними сервисами.

В диаграмме развёртывания представлены компоненты, необходимые для функционирования мобильного приложения учёта заявок технической поддержки компании. В развёртывании приложения участвуют следующие элементы:

1. Клиентские устройства: Пользователи устанавливают мобильное приложение на свои устройства, такие как смартфоны или планшеты, работающие на операционной системе Android. Клиентское приложение предоставляет пользователю возможность создавать и управлять заявками технической поддержки;

2. Firebase: Firebase используется в качестве облачной платформы для хранения данных и обмена информацией между клиентскими устройствами и сервером. Firebase обеспечивает хранение и синхронизацию данных заявок, а также авторизацию пользователей. Он также предоставляет API для взаимодействия с базой данных и облачными функциями;

3. OneSignal: Для отправки уведомлений в приложении используется интеграция с OneSignal. OneSignal используется для отправки уведомлений о новых заявках или изменении статуса существующих заявок.

Диаграмма развёртывания изображена на рисунке 20.

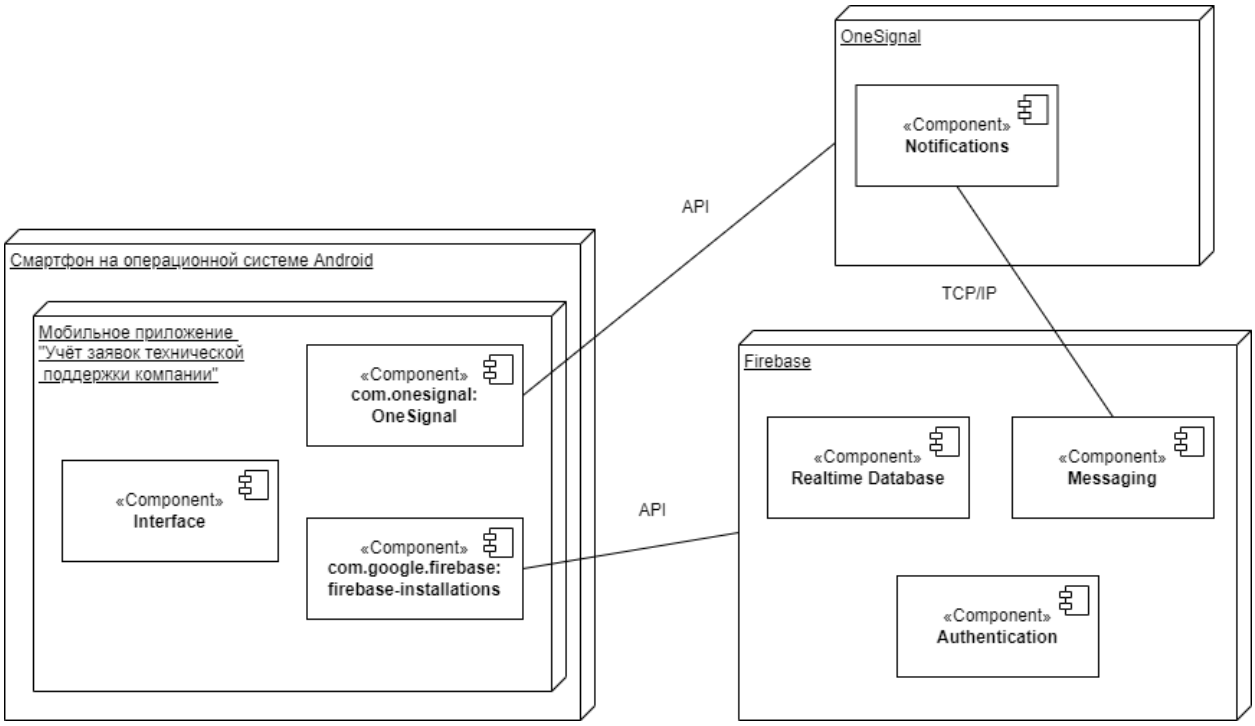


Рисунок 20 – Диаграмма развёртывания

Взаимодействие между этими компонентами позволяет приложению эффективно функционировать. Клиентские устройства обмениваются данными с Firebase, осуществляют создание и обновление статуса заявок. Firebase сохраняет и синхронизирует данные заявок между клиентскими устройствами и сервером. При необходимости OneSignal отправляет уведомления о новой заявке или об изменении статуса заявки.

Таким образом, диаграмма развёртывания показывает, как клиентские устройства взаимодействуют с Firebase и OneSignal, образуя основу для функционирования мобильного приложения учёта заявок технической поддержки.

Диаграммы компонентов используются для визуализации организации компонентов системы и зависимостей между ними. Они позволяют получить высокоуровневое представление о компонентах системы [10].

Диаграмма компонентов представлена на рисунке 21.

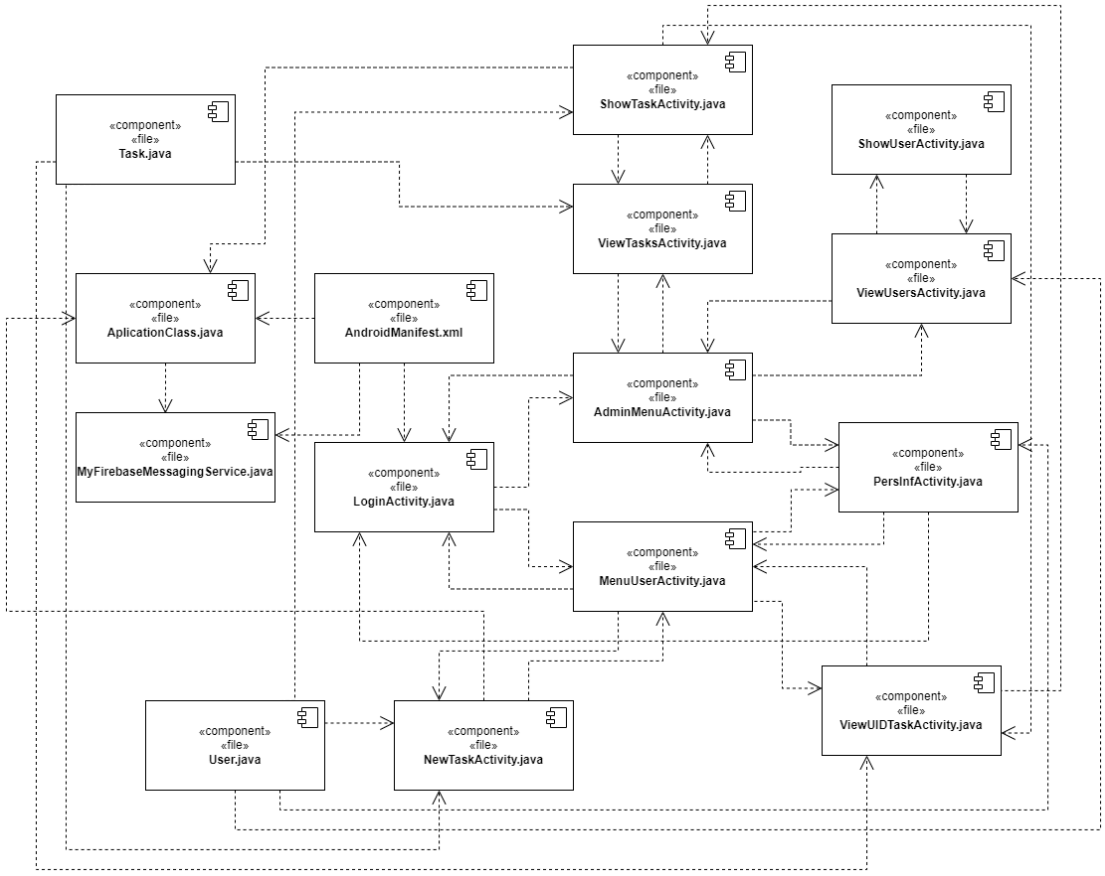


Рисунок 21 – Диаграмма компонентов

Взаимодействие между этими компонентами обеспечивает полноценное функционирование приложения. Диаграммы компонентов и развёртывания позволяют наглядно представить архитектурную структуру приложения учёта заявок технической поддержки, выделяя основные компоненты и их взаимосвязи.

Выводы по главе 2

В рамках второй главы был проведён сравнительный анализ мобильных операционных систем, в ходе которого мобильные операционные системы сравнивались по нескольким критериям. В результате данного анализа был выбран Android в качестве платформы для разработки мобильного приложения учёта заявок технической поддержки компании. Выбор данной платформы был обоснован. Далее были приведены средства разработки мобильного приложения: Android Studio, Firebase, OneSignal и язык программирования Java. Выбор средств разработки также был обоснован. После была спроектирована модель данных: концептуальная, логическая и физическая. Также были приведены виды архитектуры программного продукта. В итоге была выбрана архитектура разрабатываемого мобильного приложения, основанная на стандартных компонентах Android. В конце второй главы были представлены диаграмма развёртывания и диаграмма компонентов.

Глава 3 Реализация мобильного приложения учёта заявок технической поддержки компании

3.1 Краткое описание разработанного решения

Разработанное решение представляет собой мобильное приложение для учёта заявок технической поддержки компании. Оно разработано на языке программирования Java для платформы Android с использованием Android Studio, Firebase и OneSignal.

Приложение позволяет сотрудникам создавать новые заявки на техническую поддержку, отслеживать статус уже созданных заявок. Сотрудникам технического отдела позволяет обрабатывать полученные заявки и изменять их статус. Оно обеспечивает удобный и простой интерфейс для сотрудников, что позволяет им легко заполнять информацию о проблеме, заполнять тему заявки и отслеживать её статус.

Разработанное решение полностью интегрировано с Firebase, платформой облачных сервисов от Google. Firebase используется для хранения данных пользователей, включая информацию о заявках, аутентификации пользователей и отправки уведомлений. Благодаря этому, приложение обеспечивает надёжное и безопасное хранение данных пользователей, а также возможность аутентификации через их собственную почту.

Также в приложении используется сервис OneSignal для отправки уведомлений сотрудникам о статусе и обновлениях их заявок, а сотрудникам технического отдела о появлении новой заявки. OneSignal предоставляет простой способ интеграции и управления уведомлениями, а также гибкие настройки для внешнего вида и содержания уведомлений.

Разработанное решение предоставляет компании эффективный инструмент для учёта заявок технической поддержки, улучшает взаимодействие с сотрудниками компании и обеспечивает более быстрое и эффективное реагирование на их проблемы и запросы.

3.2 Реализация аутентификации и авторизации

Для обеспечения безопасности и контроля доступа к функциональности приложения была реализована система аутентификации и авторизации [14]. В качестве основного механизма аутентификации была выбрана авторизация через почту, что позволяет сотрудникам компании использовать свои рабочие электронные адреса для входа в приложение.

При реализации аутентификации и авторизации использовалась интеграция с Firebase Authentication. Firebase Authentication предоставляет готовые инструменты и API для аутентификации пользователей, обеспечивая безопасное хранение учётных данных и процесс входа.

На рисунке 22 представлена панель управления зарегистрированными пользователями.

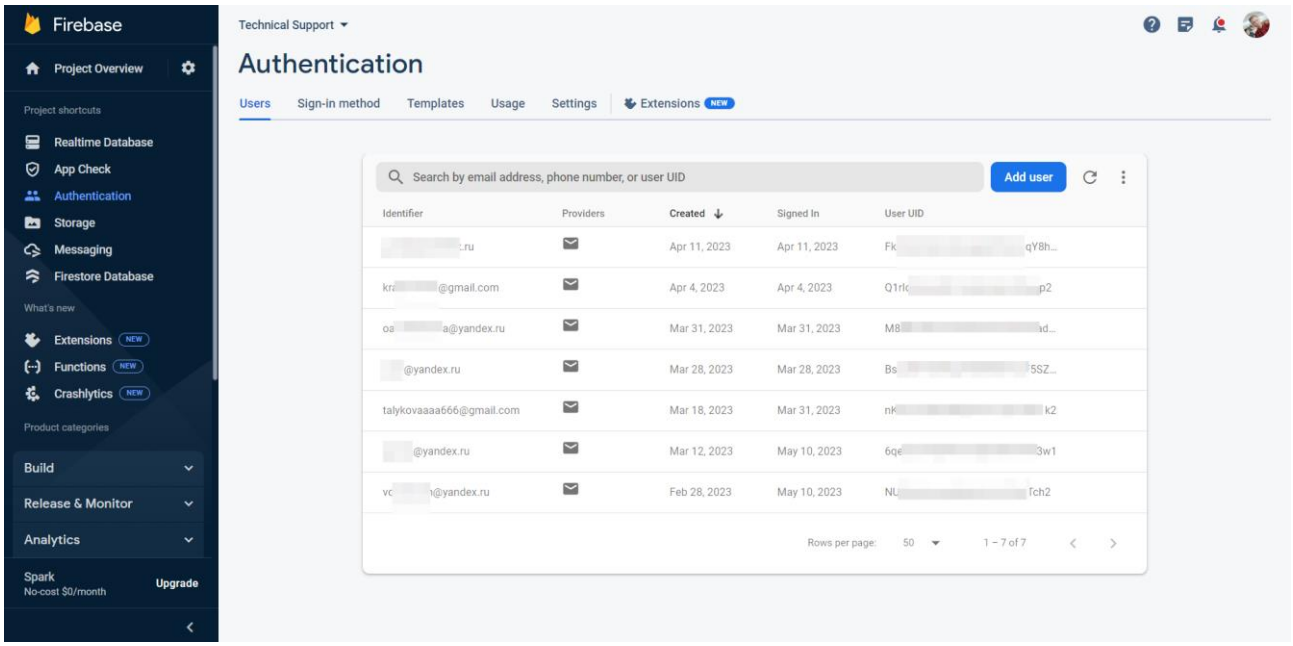


Рисунок 22 – Панель управления зарегистрированными пользователями

При запуске приложения пользователю предлагается ввести свой электронный адрес и пароль. Затем эти данные проверяются с использованием Firebase Authentication. В случае успешной аутентификации пользователю

предоставляется доступ к основным функциям приложения, включая создание и отслеживание заявок.

На рисунке 23 представлен экран входа и регистрации пользователя, а на рисунке 24 стартовый экран приложения, на котором также отображён логин пользователя и 2 кнопки. Кнопка «Начать», которая позволяет перейти к основному функционалу приложения, и кнопка «выйти из аккаунта», которая позволяет выйти из текущего аккаунта.

Рисунок 23 – Экран входа и регистрации

Рисунок 24 – Стартовый экран приложения

Важно отметить, что доступ к определенным функциям приложения может быть ограничен в зависимости от доступа сотрудника, что обеспечивает принципы авторизации. Например, администратор может иметь расширенные права доступа, такие как просмотр и редактирование статуса всех заявок, просмотр списка всех пользователей и информации о них, в то время как обычные сотрудники могут иметь доступ только к своим собственным заявкам.

На рисунках 25 и 26 представлены меню администратора и пользователя соответственно.

Рисунок 25 – Меню администратора

Рисунок 26 – Меню пользователя

Реализация аутентификации и авторизации в приложении обеспечивает безопасность данных пользователей, предотвращает несанкционированный доступ и обеспечивает конфиденциальность информации.

На рисунке 27 представлен фрагмент кода регистрации нового пользователя.

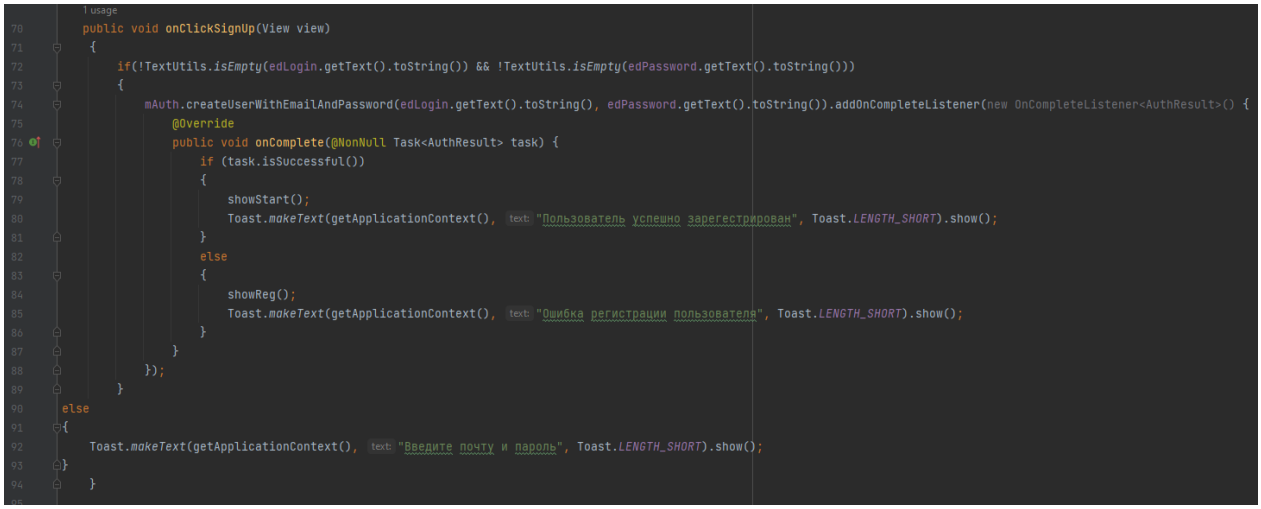


Рисунок 27 – Фрагмент кода регистрации нового пользователя

Этот код представляет обработчик события нажатия на кнопку регистрации (onClickSignUp). Внутри метода onClickSignUp происходит проверка наличия введенного логина и пароля. Если оба поля не пусты, вызывается метод createUserWithEmailAndPassword объекта mAuth (который является экземпляром класса FirebaseAuth) для создания нового пользователя с указанным логином и паролем.

Метод createUserWithEmailAndPassword возвращает объект Task<AuthResult>, и на него устанавливается слушатель OnCompleteListener, который будет выполнен после завершения операции создания пользователя. Внутри слушателя проверяется успешность операции: если регистрация прошла успешно, вызывается метод showStart() (для переключения на стартовый экран после успешной регистрации) и выводится краткое

сообщение с помощью Toast о успешной регистрации. В противном случае вызывается метод showReg() (для отображения текущего экрана регистрации) и выводится сообщение об ошибке регистрации.

Если одно или оба поля (логин и пароль) пусты, выводится соответствующее сообщение с текстом «Введите почту и пароль» с помощью Toast.

На рисунке 28 изображён фрагмент кода входа в систему существующего пользователя.

Рисунок 28 – Фрагмент кода входа в систему существующего пользователя

Этот код представляет обработчик события нажатия на кнопку входа (onClickSignIn). Внутри метода onClickSignIn происходит проверка наличия введенного логина и пароля. Если оба поля не пусты, вызывается метод signInWithEmailAndPassword объекта mAuth (экземпляр класса FirebaseAuth) для аутентификации пользователя с указанным логином и паролем.

Метод signInWithEmailAndPassword также возвращает объект Task<AuthResult>, и на него устанавливается слушатель OnCompleteListener, который будет выполнен после завершения операции входа. Внутри слушателя проверяется успешность операции: если вход выполнен успешно, вызывается метод showStart() (для переключения на стартовый экран после успешного входа) и выводится краткое сообщение с помощью Toast о успешном входе. В противном случае вызывается метод showReg() (для

отображения текущего экрана регистрации) и выводится сообщение об ошибке входа.

На рисунке 29 изображён фрагмент кода перехода в меню пользователя или администратора, в зависимости от доступа сотрудника.

```
1 usage
public void onClickStart(View view)
{
    FirebaseUser currentUser = mAuth.getCurrentUser();
    String userName = currentUser.getEmail();



    String admin = "a          ex.ru";
    if (userName.equals(admin))
    {
        Intent i = new Intent( packageContext: LoginActivity.this, AdminMenuActivity.class);
        startActivity(i);
    }
    else
    {
        Intent i = new Intent( packageContext: LoginActivity.this, MenuUserActivity.class);
        startActivity(i);
    }

}
```

Рисунок 29 – Фрагмент кода перехода в меню пользователя или администратора

Этот код представляет обработчик события нажатия на кнопку старта (onClickStart). Внутри метода onClickStart происходит получение текущего пользователя с помощью метода getCurrentUser() из объекта mAuth (экземпляр класса FirebaseAuth). Затем из объекта currentUser получается адрес электронной почты пользователя с помощью метода getEmail().

Далее в коде определяется строка admin, которая представляет адрес электронной почты администратора. Затем происходит сравнение userName (адрес электронной почты текущего пользователя) с admin.

Если userName соответствует admin, то создается новый Intent для перехода на AdminMenuActivity (это активити, отображающее меню администратора) и вызывается метод startActivity() для запуска этого Intent.

В противном случае создается новый Intent для перехода на MenuUserActivity (это активити, отображающее меню для обычного пользователя) и вызывается метод startActivity() для запуска этого Intent.

3.3 Реализация личного кабинета пользователя

В разработанном приложении был реализован личный кабинет пользователя, предоставляющий возможность просмотра и изменения персональной информации.

После успешной аутентификации пользователь получает доступ к меню, которое соответствует доступу сотрудника, в котором ему будут доступны функции, в том числе доступ к своему личному кабинету, где он может просмотреть информацию, связанную с его профилем. Это включает в себя ФИО, должность, номер кабинета, номер телефона и адрес электронной почты. Эти данные извлекаются из базы данных Firebase и автоматически отображаются в соответствующих полях.

На рисунке 30 изображён экран личного кабинета, а на рисунке 31 изображён экран редактирования информации в личном кабинете.

Рисунок 30 – Экран личного кабинета

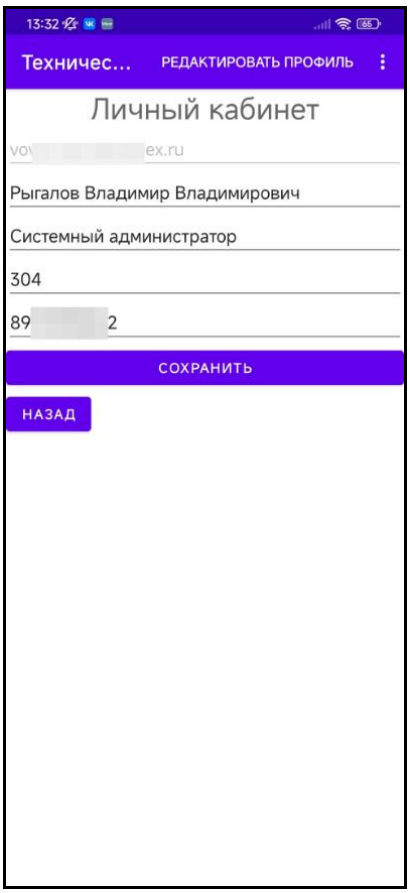


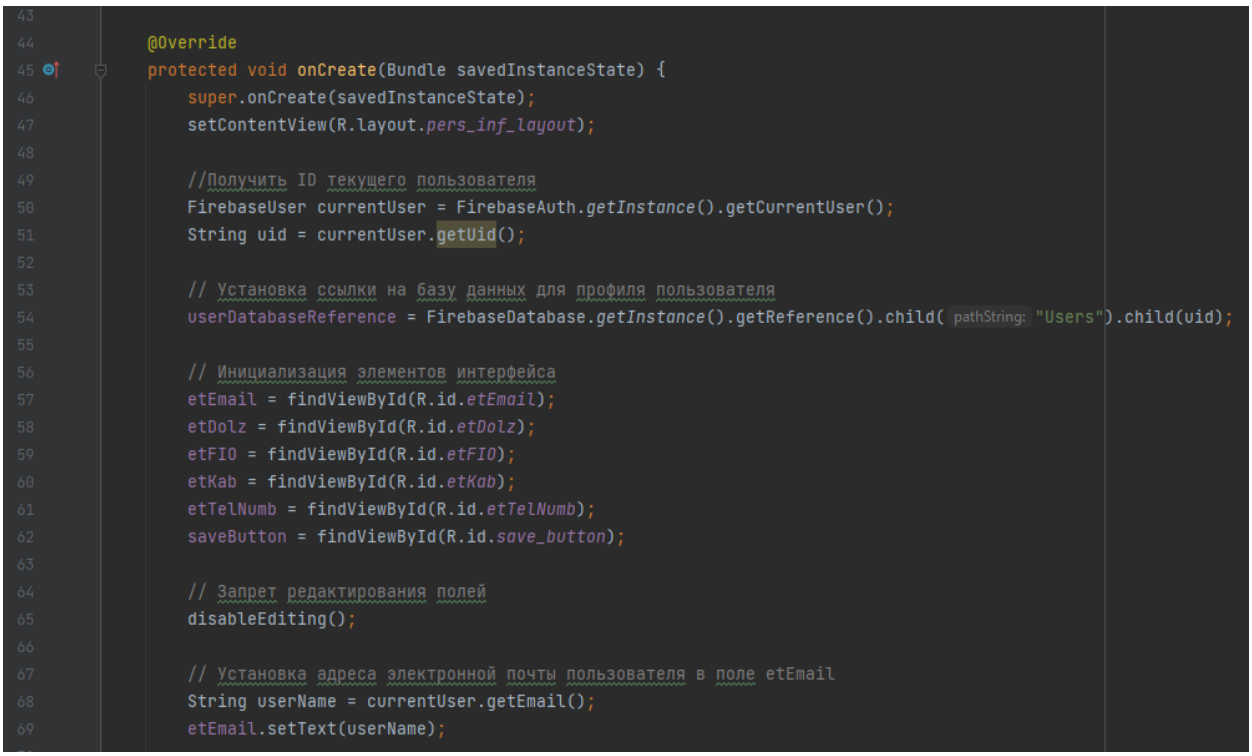
Рисунок 31 – Экран редактирования информации в личном кабинете

Пользователь имеет возможность изменить свои персональные данные, если они изменились (но не почту, т.к. она является логином для входа). На странице личного кабинета предоставляются соответствующие поля для каждого атрибута информации. Пользователь может внести изменения в одно или несколько полей и сохранить их, чтобы обновить информацию в базе данных.

Кроме того, в личном кабинете пользователь может выйти из своего аккаунта и перейти на экран входа и регистрации.

Реализация личного кабинета пользователя обеспечивает удобный и интуитивно понятный интерфейс для просмотра и изменения персональной информации.

На рисунке 32 представлен фрагмент кода получения ссылки на базу данных для профиля текущего пользователя.



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.pers_inf_layout);

    //Получить ID текущего пользователя
    FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
    String uid = currentUser.getUid();

    // Установка ссылки на базу данных для профиля пользователя
    userDatabaseReference = FirebaseDatabase.getInstance().getReference().child( pathString: "Users").child(uid);

    // Инициализация элементов интерфейса
    etEmail = findViewById(R.id.etEmail);
    etDolz = findViewById(R.id.etDolz);
    etFIO = findViewById(R.id.etFIO);
    etKab = findViewById(R.id.etKab);
    etTelNumb = findViewById(R.id.etTelNumb);
    saveButton = findViewById(R.id.save_button);

    // Запрет редактирования полей
    disableEditing();

    // Установка адреса электронной почты пользователя в поле etEmail
    String userName = currentUser.getEmail();
    etEmail.setText(userName);
```

Рисунок 32 – Фрагмент кода получения ссылки на базу данных

В данном коде происходит инициализация и настройка личного кабинета пользователя. Рассмотрим, что происходит в методе onCreate():

1. Получение текущего пользователя:

– Получаем экземпляр класса FirebaseAuth с помощью метода FirebaseAuth.getInstance(),

– Получаем текущего пользователя с помощью метода getCurrentUser(),

– Получаем идентификатор пользователя (UID) с помощью метода getUid();

2. Установка ссылки на базу данных для профиля пользователя:

– Получаем экземпляр класса FirebaseDatabase с помощью метода FirebaseDatabase.getInstance(),

– Получаем ссылку на базу данных, используя метод getReference(),

– Устанавливаем ссылку на узел "Users" в базе данных, соответствующий текущему пользователю, с помощью метода child(uid);

3. Инициализация элементов интерфейса:

– Находим элементы интерфейса по их идентификаторам с помощью метода findViewById(),

– Привязываем найденные элементы к соответствующим переменным (etEmail, etDolz, etFIO, etKab, etTelNumb, saveButton);

4. Запрет редактирования полей:

– Вызываем метод disableEditing(), который отключает возможность редактирования полей ввода;

5. Установка адреса электронной почты пользователя в поле etEmail:

– Получаем адрес электронной почты текущего пользователя с помощью метода getEmail(),

– Устанавливаем полученный адрес в поле ввода etEmail с помощью метода setText().

На рисунке 33 представлен фрагмент кода сохранения информации о пользователе в базу данных.

```
// Обработчик клика на кнопку сохранения
saveButton.setOnClickListener(new View.OnClickListener() {
                                @Override
                                public void onClick(View view) {
                                    //Сохранение данных в базу данных
                                    FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
                                    String uid = currentUser.getUid();
                                    String Email = etEmail.getText().toString().trim();
                                    String Dolz = etDolz.getText().toString().trim();
                                    String FIO = etFIO.getText().toString().trim();
                                    String Kab = etKab.getText().toString().trim();
                                    String TelNumb = etTelNumb.getText().toString().trim();
                                    String UserId = uid;
                                    String PlayerID = OneSignal.getDeviceState().getUserId();


                                    // получение токена устройства
                                    FirebaseMessaging.getInstance().getToken().addOnCompleteListener(task -> {
                                        if (!task.isSuccessful()) {
                                            return;
                                        }


                                        // получение токена
                                        String Token = task.getResult();

                                        // Создание объекта User с полученными данными
                                        User user = new User(Email, Dolz, FIO, Kab, TelNumb, UserId, Token, PlayerID);
                                        // Сохранение объекта User в базе данных
                                        userDatabaseReference.setValue(user);
                                        Toast.makeText( context: PersInfActivity.this,  text: "Информация сохранена", Toast.LENGTH_SHORT).show();
                                        disableEditing();


                                    });
                                }
                            });
```

Рисунок 33 – Фрагмент кода сохранения информации о пользователе в базу данных

Обработчик клика на кнопку сохранения выполняет следующие действия:

1. Получение текущего пользователя с помощью FirebaseAuth.getInstance().getCurrentUser();

2. Извлечение идентификатора пользователя (uid) и текстовых значений из полей ввода (email, должность, ФИО, кабинет, номер телефона);

3. Получение идентификатора устройства из OneSignal (PlayerID) с помощью OneSignal.getDeviceState().getUserId();

4. Получение токена устройства с использованием FirebaseMessaging.getInstance().getToken();

5. Создание объекта User с полученными данными (email, должность, ФИО, кабинет, номер телефона, идентификатор пользователя из OneSignal, токен устройства, идентификатор пользователя);

6. Сохранение объекта User в базе данных, используя userDatabaseReference.setValue(user);

7. Отображение всплывающего сообщения "Информация сохранена" с помощью Toast;

8. Вызов метода disableEditing() для запрета редактирования полей.

Таким образом, при клике на кнопку сохранения, данные пользователя из полей ввода сохраняются в базе данных Firebase Realtime Database, а затем происходит отображение сообщения об успешном сохранении и запрет редактирования полей.

На рисунке 34 представлено, как информация о пользователе выглядит в базе данных.

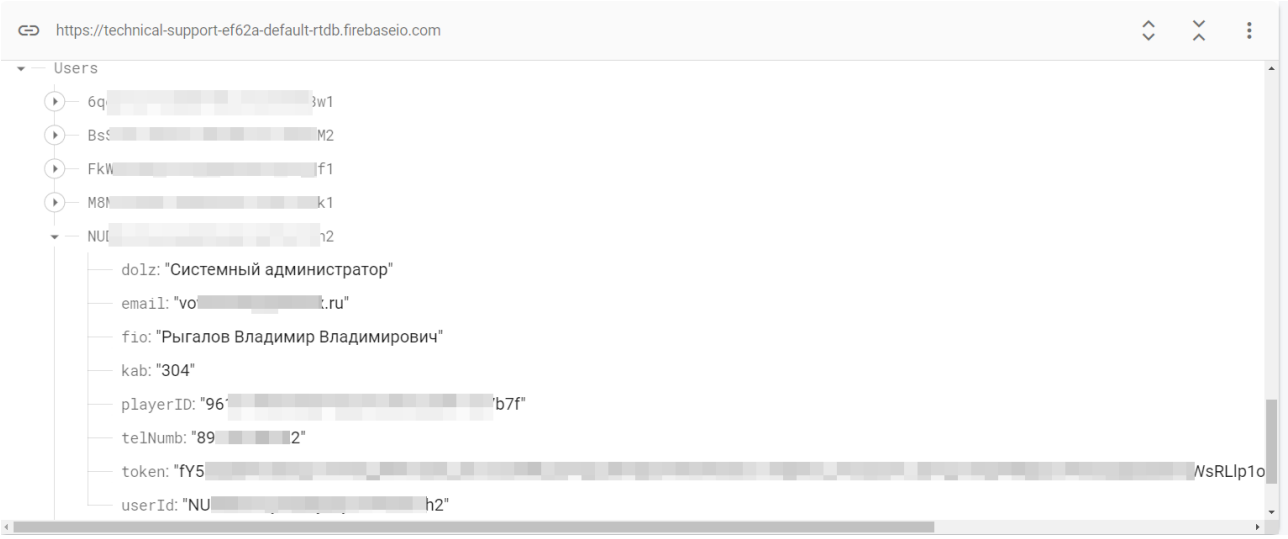


Рисунок 34 – Вид информации о пользователе в базе данных

На рисунке 35 представлен фрагмент кода отображения информации о пользователе.

```
    userDatabaseReference.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot dataSnapshot) {
            //Populate the edit texts with the user's profile data
            User user = dataSnapshot.getValue(User.class);
            if (user != null) {
                etEmail.setText(user.getEmail());
                etDolz.setText(user.getDolz());
                etFIO.setText(user.getFIO());
                etKab.setText(user.getKab());
                etTelNumb.setText(user.getTelNumb());
            }
        }

        @Override
        public void onCancelled(DatabaseError databaseError) {

        }
    });
}
```

Рисунок 35 – Фрагмент кода отображения информации о пользователе

UserDatabaseReference.addValueEventListener(...) устанавливает слушатель событий для базы данных, который отслеживает изменения данных в узле userDatabaseReference.

Когда данные в узле изменяются, метод onDataChange(...) вызывается с объектом DataSnapshot, содержащим актуальные данные из базы данных. В данном случае, данные преобразуются в объект класса User с помощью метода dataSnapshot.getValue(User.class). Затем значения полей объекта User используются для заполнения соответствующих полей ввода (EditText) интерфейса.

Если объект user не равен null, то значения его полей (email, должность, ФИО, кабинет, номер телефона) устанавливаются в соответствующие поля ввода (etEmail, etDolz, etFIO, etKab, etTelNumb).

Таким образом, данный код обновляет значения полей ввода на основе актуальных данных из базы данных и обеспечивает синхронизацию между интерфейсом и базой данных.

3.4 Реализация отправки и просмотра заявок

В рамках реализации функциональности отправки и просмотра заявок были выполнены следующие действия:

Отправка заявки:

– Разработана форма, где пользователь может ввести информацию о заявке, такую как описание проблемы и тема. На рисунке 36 представлен экран создания новой заявки;

– При сохранении заявки, введённые данные сохраняются в базе данных Firebase. На рисунке 37 представлено, как информация о заявке выглядит в базе данных;

– Для улучшения опыта пользователя реализован механизм автоматической генерации времени отправки заявки и установки начального статуса заявки «Открыта».

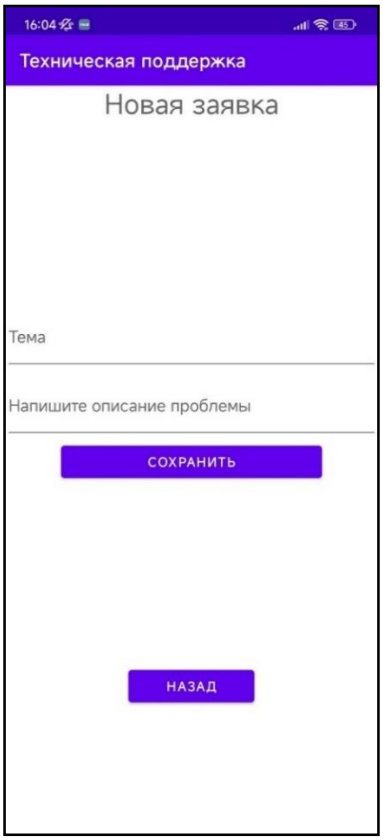


Рисунок 36 – Экран создания новой заявки

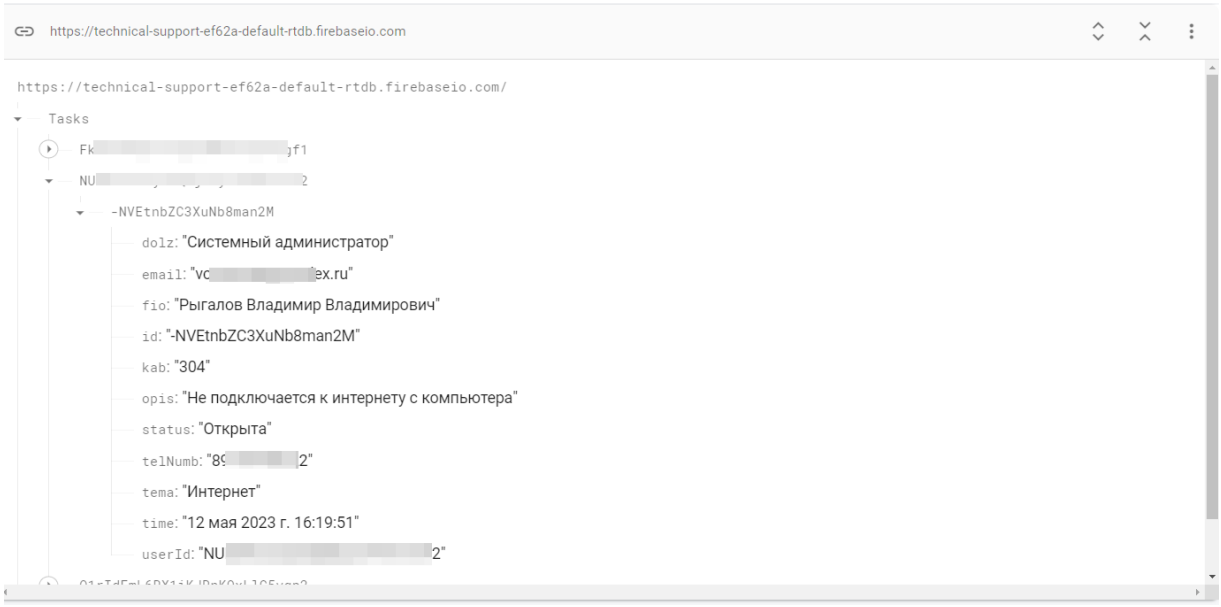


Рисунок 37 – Вид информации о заявке в базе данных

Просмотр и изменение статуса заявок:

– Создана страница, где пользователь может просмотреть информацию о своих заявках, а администратор может просмотреть все заявки. На рисунках 38 и 39 изображены экраны заявок для администратора и пользователя соответственно;

– Заявки извлекаются из базы данных Firebase и отображаются в списке с соответствующими деталями, включая тему проблемы, ФИО отправителя и текущий статус. Администратор имеет возможность изменить статус заявки. Например, он может обновить статус на «В работе» или «Закрыта», указывая текущее состояние заявки. На рисунке 40 представлен экран просмотра информации о заявке, в котором можно просмотреть подробную информацию о заявке и изменить её статус (для обычного пользователя возможность изменения статуса заявки отсутствует).

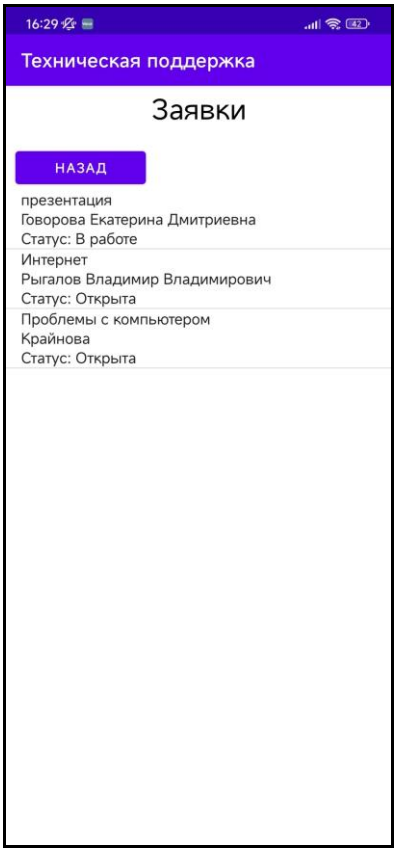


Рисунок 38 – Экран заявок для администратора

Рисунок 39 – Экран заявок для пользователя

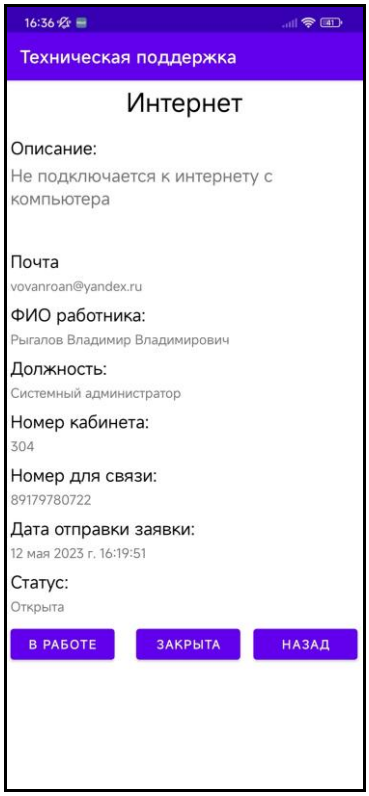


Рисунок 40 – Экран просмотра информации о заявке

Уведомления об изменении статуса заявки:

– При изменении статуса заявки, пользователю отправляется уведомление о новом статусе;

– Для отправки уведомлений используется OneSignal API, которое позволяет доставлять уведомления на устройства пользователей;

– Пользователь получает уведомление с информацией о новом статусе своей заявки, чтобы быть в курсе изменений и актуального состояния заявки. Администратор получает уведомление, когда пользователь создаёт новую заявку. На рисунках 41 и 42 показаны уведомления для пользователя о смене статуса заявки на «В работе» и «Закрыта» соответственно, а на рисунке 43 показано уведомление для администратора о получении новой заявки.

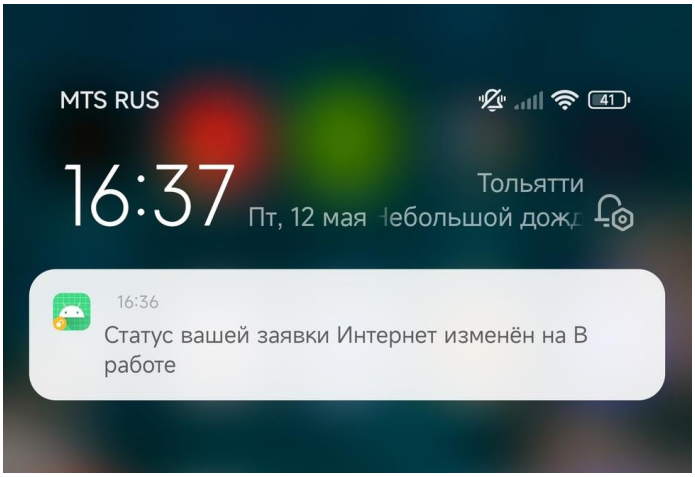


Рисунок 41 – Уведомление для пользователя о смене статуса заявки на «В работе»

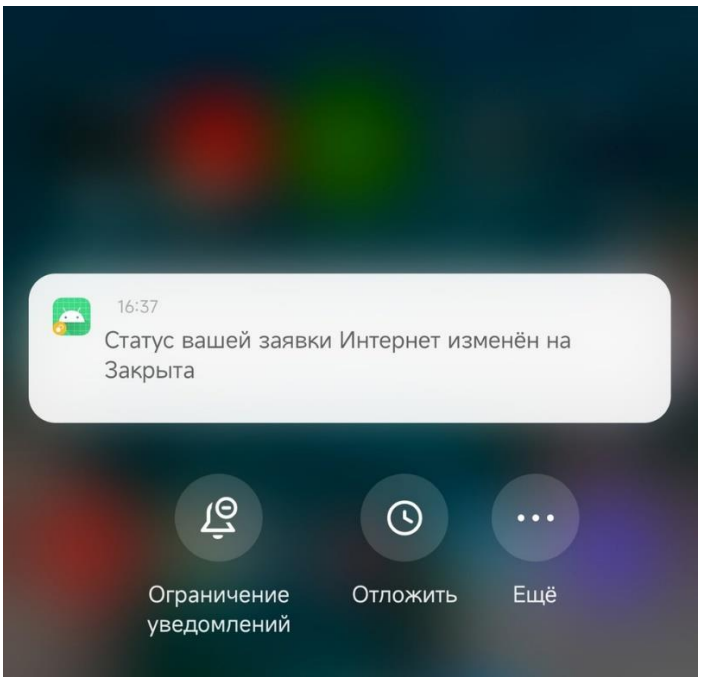


Рисунок 42 – Уведомление для пользователя о смене статуса заявки на «Закрыта»

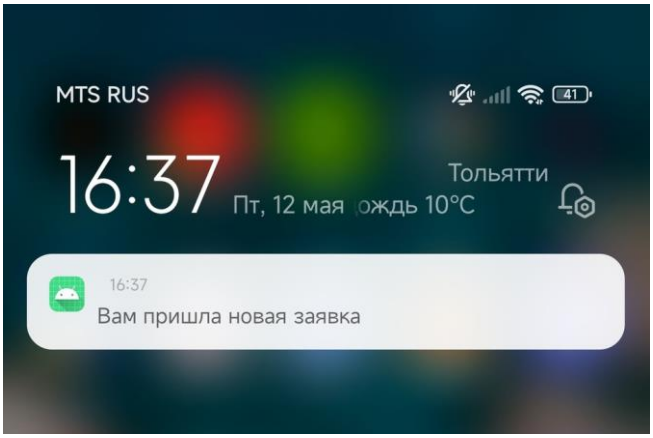
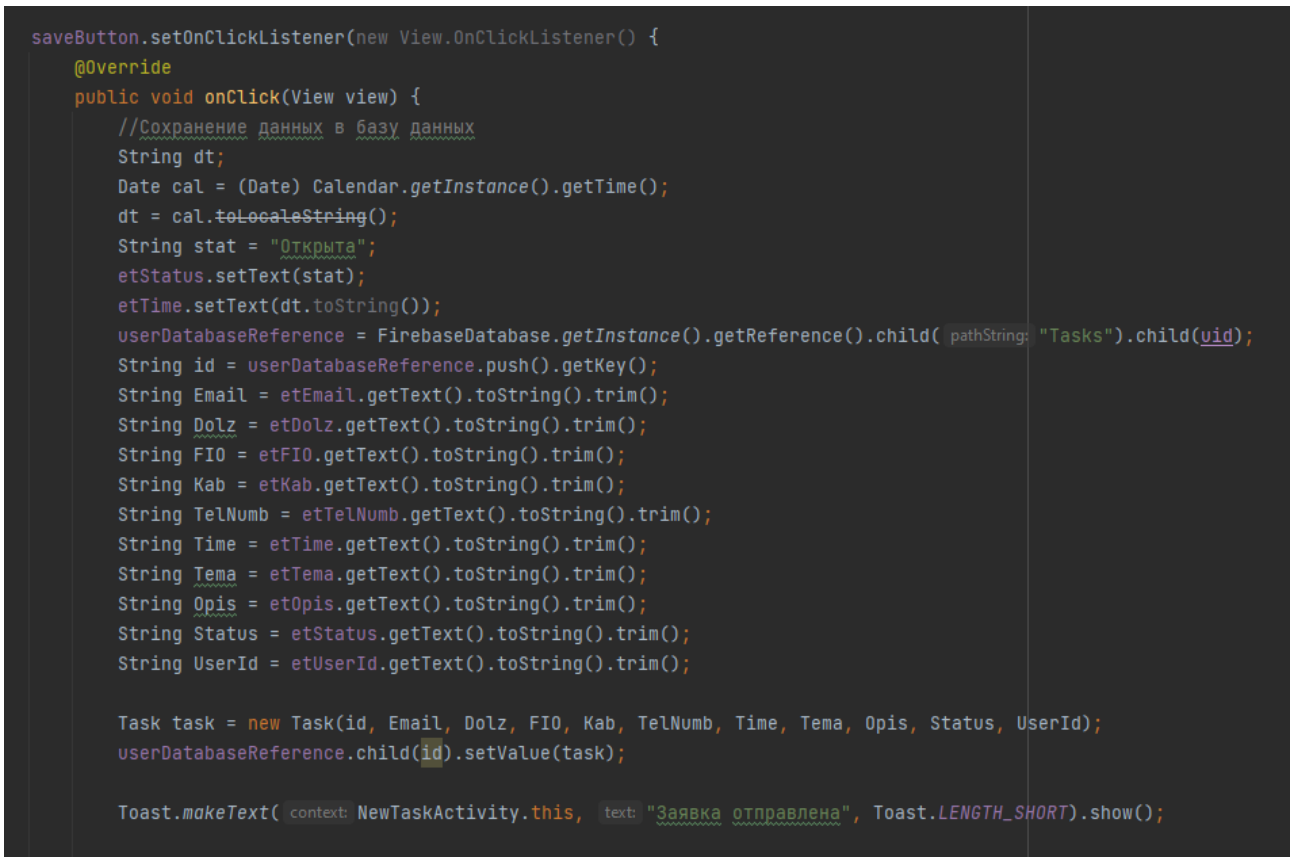


Рисунок 43 – Уведомление для администратора о получении новой заявки

Таким образом, реализация функциональности отправки и просмотра заявок включает возможность изменения статуса заявки и отправки уведомлений пользователю о любых изменениях статуса, а администратору о создании новой заявки, чтобы обеспечить более эффективное взаимодействие с заявками технической поддержки.

На рисунке 44 представлен фрагмент кода создания новой заявки.



```
saveButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        //Сохранение данных в базу данных
        String dt;
        Date cal = (Date) Calendar.getInstance().getTime();
        dt = cal.toLocaleString();
        String stat = "Открыта";
        etStatus.setText(stat);
        etTime.setText(dt.toString());
        userDatabaseReference = FirebaseDatabase.getInstance().getReference().child( pathString: "Tasks").child(uid);
        String id = userDatabaseReference.push().getKey();
        String Email = etEmail.getText().toString().trim();
        String Dolz = etDolz.getText().toString().trim();
        String FIO = etFIO.getText().toString().trim();
        String Kab = etKab.getText().toString().trim();
        String TelNumb = etTelNumb.getText().toString().trim();
        String Time = etTime.getText().toString().trim();
        String Tema = etTema.getText().toString().trim();
        String Opis = etOpis.getText().toString().trim();
        String Status = etStatus.getText().toString().trim();
        String UserId = etUserId.getText().toString().trim();

        Task task = new Task(id, Email, Dolz, FIO, Kab, TelNumb, Time, Tema, Opis, Status, UserId);
        userDatabaseReference.child(id).setValue(task);

        Toast.makeText( context: NewTaskActivity.this,  text: "Заявка отправлена", Toast.LENGTH_SHORT).show();
```

Рисунок 44 – Фрагмент кода создания новой заявки

В данном фрагменте кода реализовано действие, которое выполняется при нажатии на кнопку сохранения (saveButton). При нажатии на кнопку происходит следующее:

1. Получение текущего времени и даты:

– Создается объект Calendar, который представляет текущую дату и время,

– Получаем текущее время и дату с помощью метода getTime() и преобразуем его в строку с помощью toLocaleString(),

– Полученная строка даты и времени сохраняется в переменную dt;

2. Установка статуса заявки и времени:

– Устанавливается значение «Открыта» в поле etStatus для отображения статуса заявки,

– Устанавливается значение времени (dt) в поле etTime для отображения времени создания заявки;

3. Создание ссылки на базу данных и определение уникального идентификатора для новой заявки:

– Получаем ссылку на базу данных Firebase, на путь "Tasks" для сохранения заявок,

– Генерируется уникальный идентификатор (id) для новой заявки с помощью метода push().getKey();

4. Получение значений полей ввода:

– Получаем значения из соответствующих полей ввода, таких как электронная почта (etEmail), должность (etDolz), ФИО (etFIO), кабинет (etKab), номер телефона (etTelNumb), время (etTime), тема (etTema), описание (etOpis), статус (etStatus) и идентификатор пользователя (etUserId),

– Значения полей сохраняются в соответствующие переменные;

5. Создание объекта Task и сохранение в базе данных:

– Создается новый объект Task с помощью полученных значений,

– Затем, используя ссылку на базу данных и уникальный идентификатор, созданная задача сохраняется в базе данных;

6. Отображение уведомления:

– При помощи Toast отображается сообщение "Заявка отправлена" для информирования пользователя об успешной отправке заявки.

Таким образом, данный код реализует сохранение новой заявки в базе данных и отображение уведомления о успешной отправке.

На рисунке 45 представлен фрагмент кода отображения списка заявок.

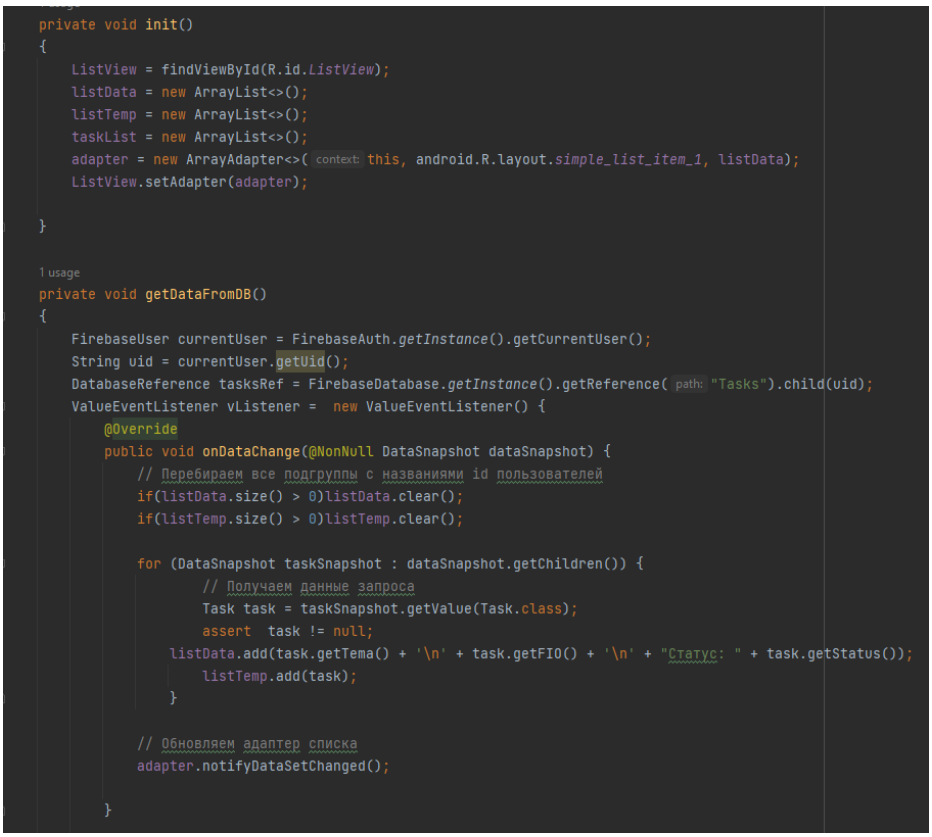


Рисунок 45 – Фрагмент кода отображения списка заявок

Данный фрагмент кода реализует два метода: init() и getDataFromDB(). Рассмотрим каждый из них:

1. Метод init():

– Инициализирует элементы интерфейса, такие как ListView,

– Создает новые экземпляры списков listData, listTemp и taskList,

– Создает новый адаптер ArrayAdapter и связывает его с ListView;

2. Метод getDataFromDB():

– Получает текущего пользователя (FirebaseUser) через FirebaseAuth,

– Получает ссылку на базу данных Firebase для задач текущего пользователя (tasksRef),

– Создает слушатель значений (ValueEventListener), который будет отслеживать изменения данных в базе данных;

3. В методе onDataChange() обрабатываются полученные данные из базы данных:

– Очищаются списки listData и listTemp, чтобы обновить их,

– Перебираются все подгруппы (taskSnapshot) с данными задач;

4. Для каждой заявки (Task) из базы данных:

– Получаются данные заявки (task) из taskSnapshot,

– Добавляется информация о заявке в список listData в формате: тема, ФИО и статус заявки,

– Заявка (task) добавляется в список listTemp,

– После обработки всех заявок обновляется адаптер списка с помощью adapter.notifyDataSetChanged().

Таким образом, методы init() и getDataFromDB() используются для инициализации и получения данных из базы данных, а также обновления списка заявок (ListView) на основе полученных данных.

На основе данного кода также была реализована функция просмотра списка пользователей, которая доступна только администратору (Приложение А).

На рисунке 46 представлен экран просмотра списка пользователей.

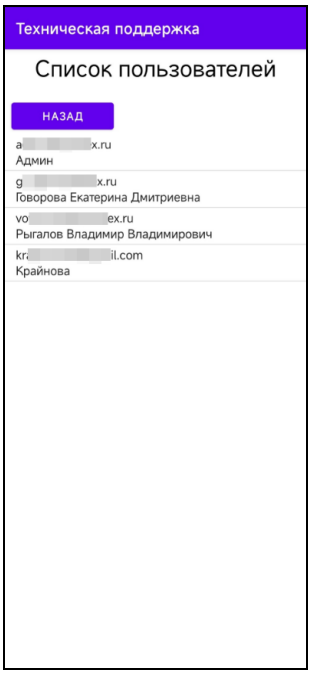
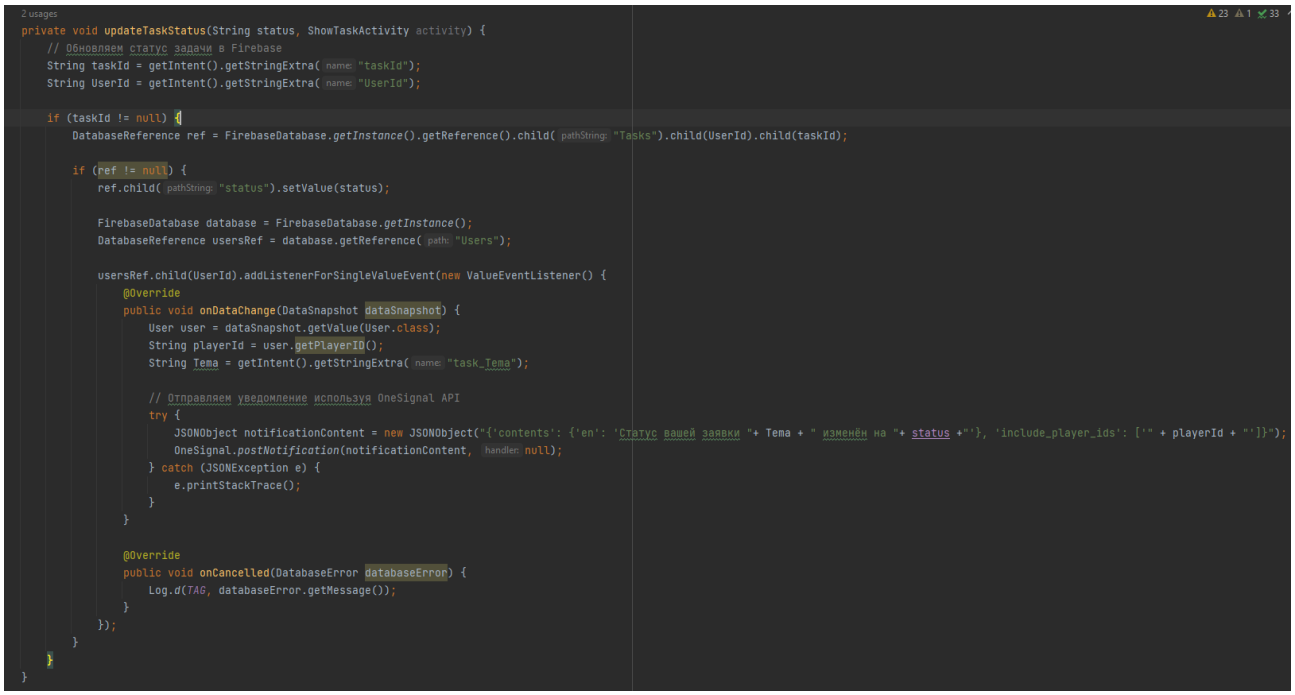


Рисунок 46 – Экран просмотра списка пользователей

На рисунке 47 представлен фрагмент кода, отвечающий за изменение статуса заявки и отправку уведомления о смене статуса заявки пользователю.



```
2 usages
private void updateTaskStatus(String status, ShowTaskActivity activity) {
    // Обновляем статус задачи в Firebase
    String taskId = getIntent().getStringExtra( name: "taskId");
    String UserId = getIntent().getStringExtra( name: "UserId");

    if (taskId != null) {
        DatabaseReference ref = FirebaseDatabase.getInstance().getReference().child( pathString: "Tasks").child(UserId).child(taskId);

        if (ref != null) {
            ref.child( pathString: "status").setValue(status);

            FirebaseDatabase database = FirebaseDatabase.getInstance();
            DatabaseReference usersRef = database.getReference( path: "Users");

            usersRef.child(UserId).addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(DataSnapshot dataSnapshot) {
                    User user = dataSnapshot.getValue(User.class);
                    String playerId = user.getPlayerID();
                    String Tema = getIntent().getStringExtra( name: "task_Tema");

                    // Отправляем уведомление используя OneSignal API
                    try {
                        JSONObject notificationContent = new JSONObject("{'contents': {'en': 'Статус вашей заявки "+ Tema + " изменён на "+ status +"'}, 'include_player_ids': ['" + playerId + "']}");
                        OneSignal.postNotification(notificationContent, handler: null);
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }

                @Override
                public void onCancelled(DatabaseError databaseError) {
                    Log.d(TAG, databaseError.getMessage());
                }
            });
        }
    }
}
```

Рисунок 47 – Фрагмент кода, отвечающий за изменение статуса заявки и отправку уведомления о смене статуса заявки пользователю

Данный код реализует метод updateTaskStatus, который выполняет следующие действия:

1. Получает информацию о заявке из предыдущего активити (используется getIntent().getStringExtra() для получения значений taskId и UserId);
2. Получает ссылку на заявку (ref) в базе данных Firebase на основе UserId и taskId;
3. Если ссылка (ref) не является пустой, то:

– Обновляет поле "status" заявки в базе данных Firebase с помощью ref.child("status").setValue(status), где status - новое значение статуса заявки,

– Получает ссылку на базу данных Firebase с помощью FirebaseDatabase.getInstance().getReference("Users") для работы с данными пользователей,

– С помощью метода addListenerForSingleValueEvent добавляет слушатель значений, который будет выполняться один раз и получать данные о пользователе;

4. В методе onDataChange() обрабатываются полученные данные о пользователе:

– Получает информацию о пользователе (User) из dataSnapshot,

– Получает playerId пользователя (для отправки уведомления) с помощью user.getPlayerID(),

– Получает тему задачи (Tema) из предыдущего активити с помощью getIntent().getStringExtra("task_Tema"),

– Создает JSON-объект (notificationContent), содержащий текст уведомления и идентификатор получателя,

– Отправляет уведомление с помощью OneSignal API, используя OneSignal.postNotification(notificationContent, null);

5. В случае ошибки (onCancelled()) выводит сообщение об ошибке в журнал с помощью Log.d().

Таким образом, данный метод обновляет статус задачи в базе данных Firebase, отправляет уведомление пользователю об изменении статуса с использованием OneSignal API и выводит сообщение об ошибке, если что-то пошло не так.

Далее метод updateTaskStatus используется в обработчике нажатий в кнопках изменения статуса «В работе» и «Закрыта».

3.5 Тестирование мобильного приложения

В рамках функционального тестирования мобильного приложения были проверены следующие основные функции [20]:

1. Авторизация и аутентификация пользователей:

– Проверка возможности успешной авторизации с использованием зарегистрированного аккаунта,

– Проверка обработки ошибок при вводе неверных учетных данных,

– Проверка создания нового аккаунта и корректной регистрации нового пользователя,

– Проверка на получение доступа к определённому функционалу, в зависимости от прав пользователя;

2. Создание и отправка заявок:

– Проверка возможности создания новой заявки с заполнением всех необходимых полей (тема, описание),

– Проверка корректного сохранения созданной заявки в базе данных,

– Проверка отправки заявки на сервер и успешной обработки данных,

– Проверка отправки уведомления администратору о получении новой заявки;

3. Просмотр и изменение информации о пользователе:

– Проверка возможности просмотра информации о пользователе, такой как имя, должность, контактные данные и т.д.,

– Проверка возможности изменения и сохранения измененных данных пользователя,

– Проверка корректного обновления информации о пользователе в базе данных;

4. Обновление статуса заявок:

– Проверка возможности изменения статуса заявки с «Открыта» на «В работе» и «Закрыта»,

– Проверка корректного обновления статуса заявки в базе данных,

– Проверка отправки уведомления пользователю о изменении статуса заявки;

5. Обработка ошибок и исключительных ситуаций:

– Проверка обработки ошибок при отсутствии Интернет-соединения и предоставлении соответствующих сообщений пользователю,

– Проверка корректной обработки ошибок ввода данных, таких как неправильный формат электронной почты или незаполненные обязательные поля.

Результаты функционального тестирования показали, что все основные функции приложения работают корректно и соответствуют требованиям. Были выявлены и исправлены некоторые ошибки и недочеты, что позволило улучшить стабильность и функциональность приложения.

Выводы по главе 3

В рамках 3 главы было кратко описано разработанное решение. Была подробно описана реализация основного функционала приложения. Также было проведено функциональное тестирование, результаты которого показывают, что основной функционал приложения работает правильно и соответствует требованиям. Благодаря тестированию были обнаружены и исправлены некоторые недочёты, которые удалось исправить, что помогло сделать функционирование приложения стабильным.

Заключение

В рамках данной дипломной работы была поставлена цель разработки мобильного приложения учета заявок технической поддержки компании с целью упрощения и улучшения процесса учета и обработки заявок. Исходя из актуальности данной задачи и необходимости повышения эффективности работы и уровня удовлетворенности специалистов компании, разработка такого приложения имеет высокую практическую значимость.

В рамках работы был проведен анализ бизнес-процессов предприятия, включающий описание самого предприятия и выбор CASE-средств для описания бизнес-процессов. Также был проведен анализ модели текущих бизнес-процессов с использованием диаграмм, что позволило более детально изучить их особенности и выявить потенциальные улучшения.

На основе проведенного анализа были сформулированы требования к мобильному приложению учета заявок технической поддержки компании. Кроме того, был проведен обзор и анализ аналогов существующих мобильных приложений для учета заявок, что позволило выявить их преимущества и недостатки.

На основе сформулированных требований и проведенного анализа была поставлена задача на разработку мобильного приложения, произведено проектирование приложения, выбрана платформы реализации и средства разработки, а также было произведено проектирование модели данных. Архитектура разработанного мобильного приложения основана на стандартных компонентах Android.

Реализация мобильного приложения выполнена с использованием выбранной платформы и средств разработки. Проведено тестирование, в результате которого была проверена его функциональность, стабильность и соответствие поставленным требованиям.

В результате работы была достигнута поставленная цель – разработано мобильное приложение для учета заявок технической поддержки компании.

Разработанное приложение позволяет упростить и улучшить процесс учета и обработки заявок, повышая эффективность работы и уровень удовлетворенности сотрудников компании.

Работа была апробирована на научно-практической конференции «Студенческие Дни науки в ТГУ».

Список используемой литературы

1. Android Studio. [Электронный ресурс] / Режим доступа: URL: https://developer.android.com/studio/intro (дата обращения 04.03.2023).
2. Diagrams.net. [Электронный ресурс] / Режим доступа: URL: https://en.wikipedia.org/wiki/Diagrams.net (дата обращения 02.04.2023).
3. Eeles P. Capturing architectural requirements //IBM Rational developer works. – 2005.
4. Firebase for Android. [Электронный ресурс] / Режим доступа: URL: https://firebase.google.com/docs/android/setup?authuser=0&hl=en (дата обращения 27.02.2023).
5. Goadrich M. H., Rogers M. P. Smart smartphone development: iOS versus Android //Proceedings of the 42nd ACM technical symposium on Computer science education. – 2011. – С. 607-612.
6. Mobile Operating System Market Share. [Электронный ресурс] / Режим доступа: URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (дата обращения 15.04.2023).
7. Nimodia C., Deshmukh H. R. Android operating system //Software Engineering. – 2012. – Т. 3. – №. 1. – С. 10.
8. OneSignal SDK. [Электронный ресурс] / Режим доступа: URL: https://documentation.onesignal.com/docs/android-sdk-setup (дата обращения 17.03.2023).
9. Афанасьев А. В. Актуальность языка программирования java в разработке мобильных приложений в 2020 году //Информационные технологии в процессе подготовки современного специалиста. – 2020. – С. 25-29.
10. Афанасьев А. Н., Войт Н. Н. Разработка компонентно-сервисной платформы обучения: диаграммы классов программного компонента сценария на UML-языке //Вестник Ульяновского государственного технического университета. – 2012. – №. 2 (58). – С. 32-36.
11. 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.
12. Вязовик Н., Жилин Е. Программирование на Java //М.: Интуит. ру. – 2003.
13. Зиятдинова А., Староверова Н. А. Аналитический обзор и сравнение возможностей операционных систем для мобильных устройств //Фундаментальные исследования. – 2015. – №. 9-2. – С. 227-231.

14. Иванов В. В., Лубова Е. С., Черкасов Д. Ю. Аутентификация и авторизация //Проблемы современной науки и образования. – 2017. – №. 2 (84). – С. 31-33.

15. Микляев И. А., Ундозерова А. Н., Кудаева М. В. Универсальная логическая модель базы данных //Arctic Environmental Research. – 2010. – №. 1. – С. 93-98.

16. Об учреждении «Арена». [Электронный ресурс] / Режим доступа: URL: https://www.tlt-arena.ru/about/ (дата обращения 24.02.2023).

17. Оптимизация бизнес-процессов предприятия. [Электронный ресурс] / Режим доступа: URL: https://piter-consult.ru/home/Articles/Simply-about-the-difficult/Let-entrust-business-processes.html (дата обращения 24.02.2023).

18. Подольская О. В., Коржов С. А. ИНФОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ //INTERNATIONAL INNOVATION RESEARCH. – 2017. – С. 11-13.

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

20. Юрченко А. Н. Функциональное тестирование как одна из методологий тестирования программного обеспечения //Вестник современных исследований. – 2018. – №. 1.1. – С. 155-156.

Приложение А

Фрагмент кода отображения списка пользователей

```
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;
import java.util.List;

public class ViewUsersActivity extends AppCompatActivity {

    private ListView ListView;
    private ArrayAdapter<String> adapter;
    private List<String> listData;
    private List<User> listTemp;
```

```
    private DatabaseReference userDatabaseReference;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.view_users_layout);
        init();
        getDataFromDB();
        setOnClickItem();

    }

    private void init()
    {
        ListView = findViewById(R.id.ListView);
        listData = new ArrayList<>();
        listTemp = new ArrayList<>();
        adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,
listData);
        ListView.setAdapter(adapter);
        userDatabaseReference =
FirebaseDatabase.getInstance().getReference().child("Users");
    }

    private void getDataFromDB()
    {
        ValueEventListener vListener =  new ValueEventListener() {
            @Override
```

```
        public void onDataChange(@NonNull DataSnapshot dataSnapshot)
        {
            if(listData.size() > 0)listData.clear();
            if(listTemp.size() > 0)listTemp.clear();
        for(DataSnapshot ds : dataSnapshot.getChildren())
        {
            User user = ds.getValue(User.class);
            assert  user != null;
            listData.add(user.getEmail() + '\n' + user.getFIO());
            listTemp.add(user);

        }
        adapter.notifyDataSetChanged();
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {

        }
    };
userDatabaseReference.addValueEventListener(vListener);
}

private void setOnClickItem()
{
    ListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> parent, View view, int position,
```

```
long id) {
            User user  = listTemp.get(position);
            Intent i = new Intent(ViewUsersActivity.this, ShowUserActivity.class);
            i.putExtra("user_Email", user.getEmail());
            i.putExtra("user_FIO", user.getFIO());
            i.putExtra("user_Dolz", user.getDolz());
            i.putExtra("user_Kab", user.getKab());
            i.putExtra("user_TelNum", user.getTelNumb());
            i.putExtra("user_Token", user.getToken());
            i.putExtra("user_PlayerID", user.getPlayerID());
            startActivity(i);
        }
    });

}

public void onClickNazad(View view)
{
    FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
    String userName = currentUser.getEmail();

    String admin = "admin@yandex.ru";
    if (userName.equals(admin))
    {
        Intent i = new Intent(ViewUsersActivity.this, AdminMenuActivity.class);
        startActivity(i);
    }
    else
```

```
        {
            Intent i = new Intent(ViewUsersActivity.this, MenuUserActivity.class);
            startActivity(i);
        }
    }
}
```