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)

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

Бизнес информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка мобильного приложения «Планировщик задач» для организации

Обучающийся В.Н. Кириллов 
(Инициалы Фамилия) (личная подпись)

Руководитель К.э.н., доцент Т.А. Раченко
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2023

Аннотация

Дипломная работа состоит из 3 глав и включает 81 страницу, 20 рисунков, 4 таблицы, 6 приложений и основывается на 24 источниках литературы. Объектом исследования процесс планирования задач в организации, а предметом исследования – проектирование, разработка и тестирование мобильного приложения для автоматизации процесса планирования задач в организации.

Целью данного исследования было разработать мобильное приложение «Планировщик задач» для организации ООО «Департамент финансовых услуг». В современных условиях конкуренции и быстро меняющейся рыночной среды автоматизация планирования задач играет важную роль, обеспечивая упрощение и ускорение работы, а также повышение точности прогнозирования сроков выполнения задач.

В результате выполнения дипломной работы было разработано мобильное приложение «Планировщик задач», которое может быть использовано в ООО «Департамент финансовых услуг» для повышения эффективности управления задачами и временем. Пользователи приложения смогут эффективно управлять своими задачами и проектами, оптимизировать распределение ресурсов и времени, что в итоге приведет к повышению производительности и прибыльности компании.

Оглавление

Введение

В настоящее время, когда технологии становятся все более доступными и удобными, организация личного и рабочего времени становится все сложнее.

Многие люди имеют нагрузку на работе, учебе и личной жизни, и им необходимо управлять своим временем, чтобы достигать поставленных целей и выполнять задачи в срок.

Разработка мобильного приложения Планировщик задач для организации может помочь пользователям оптимизировать их рабочее и личное время, обеспечивая удобный и простой способ управления задачами.

Актуальность темы связана с растущей потребностью людей и организаций, в особенности ООО «Департамент финансовых услуг», в инструментах эффективного управления временем и задачами.

Развитие мобильных приложений для планирования задач может удовлетворить эту потребность и повысить экономическую эффективность работы компаний.

Объектом исследования является процесс планирования задач в организации.

Предмет исследования: проектирование, разработка и тестирование мобильного приложения для автоматизации процесса планирования задач в организации.

Цель исследования: разработать мобильное приложение «Планировщик задач» ООО «Департамент финансовых услуг».

Задачи исследования:

1. описать организацию ООО «Департамент финансовых услуг»;
2. произвести анализ бизнес-процессов организации;
3. дать обзор существующих решений в области программного обеспечения «планировщик задач»;
4. определить требования к будущей системе;

5. разработать постановку задачи на разработку приложения;

6. обосновать выбор средств разработки;

7. спроектировать элементы модели;

8. произвести логическое и физическое моделирование приложения;

9. разработать интерфейс и архитектурную модель приложения;

10. оценить эффективность проекта;

11. реализовать дизайн приложения;

12. реализовать модули приложения;

13. произвести тестирование приложения.

Научная новизна исследования заключается в проектировании программного обеспечения, основанного на алгоритмах оптимизации планирования задач, которые позволяют находить наилучший план выполнения задач с учетом времени и приоритетов.

Использование подобных технологий позволит автоматизировать бизнес-процессы организаций.

Проектирование приложения планировщика задач для организации имеет высокую теоретическую значимость, поскольку это позволяет решать проблемы, связанные с неэффективным использованием времени и недостаточной организованностью.

Создание приложения планировщика задач позволяет пользователю организовать свои задачи, распределить их по приоритетам и срокам, и следить за их выполнением.

Это упрощает процесс управления задачами, позволяя пользователям сосредоточиться на более важных задачах и улучшить свою продуктивность.

Кроме того, проектирование приложения планировщика задач включает в себя разработку интерфейса, который позволяет пользователям легко и быстро находить нужную информацию и выполнять необходимые действия.

Необходимо отметить, что такой дизайн может применяться в других областях, где требуется удобный и интуитивно понятный интерфейс для выполнения задач.

Практическая значимость работы заключается в том, что разработанное приложение может быть использовано в бизнес-среде для повышения эффективности управления задачами и временем, что в свою очередь может привести к повышению производительности и прибыльности компании. Пользователи приложения смогут управлять своими задачами и проектами, оптимизировать распределение времени и ресурсов, что позволит им улучшить свою работу и достичь более высоких результатов.

Методы исследования: литературный обзор, анализ конкурентов, классификация, моделирование, тестирование.

Работа состоит из введения, трех глав, разделенных на параграфы, заключения и списка использованной литературы.

Глава 1 Описание предметной области

1.1 Описание организации ООО «Департамент финансовых услуг»

Общество с ограниченной ответственностью «Департамент финансовых услуг» зарегистрировано 20.09.2012 года по следующему юридическому адресу: Самарская область, город Тольятти, улица Гидростроевская, дом 21, помещение 1003. Индекс организации: 445020. Регистратором ООО «Департамент финансовых услуг» выступает Инспекция Федеральной налоговой службы по Красноглинскому району г. Самары, №6313. Форма собственности: частная. Уставный капитал: 10 тысяч рублей.

Организационно-правовой формой организации является общество с ограниченной ответственностью, следовательно, ООО «Департамент финансовых услуг». Предприятие оказывает финансовые и правовые услуги для сторонних ИП, ООО и физическим лицам. Реализация услуг организации разбита на 2 вида: разовые обращения; полное бухгалтерское и налоговое сопровождение.

Основными реквизитами организации выступают: ИНН 6324032081; ОГРН 112634010229; КПП 632401001.

Цель организации: получение прибыли от реализации финансовых и правовых услуг.

Миссия организации: оказание комплексных услуг в области учета, налогов и права, способствующих совершенствованию системы управления предприятием заказчика.

Основные задачи организации:

1. обеспечение точности финансовой отчетности. Главная задача ООО «Департамента финансовых услуг» состоит в том, чтобы вести учет операций и подготавливать точные финансовые отчеты, которые могут быть использованы для принятия управленческих решений;

2. соблюдение правовых и налоговых требований. Организация должна соблюдать все законодательные и налоговые требования, связанные с ведением учета и отчетности;

3. управление денежными потоками. ООО «Департамент финансовых услуг» должен помогать своим клиентам управлять денежными потоками, планировать бюджет и прогнозировать доходы/ расходы;

4. консультирование клиентов. Организация реализует своим клиентам консультации по вопросам, связанным с налогами, правом, финансами и бухгалтерским учетом;

5. повышение эффективности бизнеса. ООО «Департамент финансовых услуг» должен реализовать своим клиентам повышение эффективности своего бизнеса: анализировать финансовые показатели и идентифицировать возможности для улучшения;

6. сохранение конфиденциальности. Организация должна соблюдать высокие стандарты конфиденциальности и защиты данных своих клиентов.

Директор ООО «Департамент финансовых услуг» несет ответственность за руководство и управление ее деятельностью. Его основные обязанности включают в себя:

1. разработка стратегии и планирование деятельности организации;

2. организация бухгалтерского учета и отчетности в соответствии с законодательством и требованиями клиентов;

3. обеспечение соблюдения законодательства о бухгалтерском учете и налогообложении;

4. управление финансовыми ресурсами организации и контроль за их использованием;

5. разработка и внедрение системы контроля качества бухгалтерских услуг;

6. обеспечение своевременного и точного выполнения бухгалтерских работ;

7. руководство и координация работы сотрудников;

8. взаимодействие с клиентами и установление долгосрочных отношений;

9. разработка и внедрение новых услуг и технологий в области бухгалтерского учета;

10. решение возникающих проблем и конфликтов в работе организации.

Главный бухгалтер отвечает за финансовые операции компании и подчиненных ей бухгалтеров. К обязанностям относятся:

1. ведение бухгалтерского учета. Главный бухгалтер отвечает за правильное ведение бухгалтерского учета организации, включая составление бухгалтерской отчетности в соответствии с законодательством РФ;

2. контроль за финансовыми операциями. Главный бухгалтер контролирует выполнение финансовых операций компании, включая оплату счетов, платежи по налогам и зарплаты сотрудников;

3. подготовка отчетности. Главный бухгалтер подготавливает финансовую отчетность компании, включая баланс, отчет о прибылях и убытках, отчет о движении денежных средств и др.;

4. налоговый учет. Главный бухгалтер отвечает за подготовку и подачу налоговой отчетности ООО «Департамента финансовых услуг», включая расчет налогов и уплату налогов;

5. разработка бухгалтерской политики. Главный бухгалтер отвечает за разработку бухгалтерской политики компании, включая определение правил учета, расходов/доходов и т.д.

Бухгалтер-юрист, выполняет ряд обязанностей, связанных с оказанием качественных услуг своим клиентам. Ниже приведены основные обязанности бухгалтера-юриста в ООО «Департаменте финансовых услуг»:

1. оказание консультаций по вопросам бухгалтерского и налогового учета, а также правовым вопросам, связанным с деятельностью клиентов;

2. разработка и согласование договоров с клиентами на оказание юридических и бухгалтерских услуг;

3. проведение анализа юридических рисков клиентов, связанных с бухгалтерскими операциями;

4. подготовка и представление отчетности перед налоговыми органами и другими контролирующими органами;

5. осуществление налогового планирования и оптимизации налоговых платежей клиентов;

6. составление и согласование документов по регистрации, ликвидации и реорганизации организаций-клиентов;

7. участие в юридических консультациях по вопросам, связанным с деятельностью клиентов;

8. проведение анализа бухгалтерской отчетности клиентов и предоставление рекомендаций по ее улучшению.

Обязанности бухгалтера-инспектора по кадрам:

1. подготовка документов и формирование отчетности по кадровым вопросам, таким как оформление документов на прием, увольнение, перевод сотрудников, расчет и начисление заработной платы, налогов и страховых взносов и т.д.;

2. соблюдение законодательства в области трудовых отношений, социального страхования и налогообложения, контроль за исполнением трудовых договоров, регулярное обновление знаний в этой области;

3. обеспечение актуальности и достоверности кадровой документации, контроль за соблюдением порядка ее хранения и архивации;

4. поддержание связи с сотрудниками и директором организации по вопросам трудовых отношений, консультации по кадровым вопросам, разрешение конфликтов, проведение переговоров;

5. участие в организации и проведении процессов подбора, отбора и адаптации новых сотрудников, организации тренингов и семинаров для повышения квалификации персонала;

6. работа с системами учета и анализа кадровых данных, анализ и оценка эффективности работы персонала, подготовка отчетов и рекомендаций для руководства компании;

7. соблюдение конфиденциальности и защита персональных данных сотрудников компании в соответствии с законодательством и политикой организации.

Таким образом, древовидная организационная структура ООО «Департамент финансовых услуг» представлена на рисунке 1.

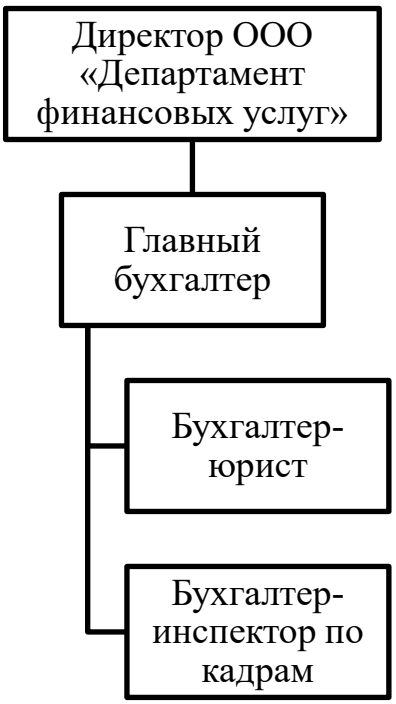

Рисунок 1 – Организационная структура ООО «Департамент финансовых услуг»

Как видно из рисунка 1, в ООО «Департаменте финансовых услуг» 3 наемных работника и само руководство в лице директора организации.

ООО «Департамент финансовых услуг» оказывает широкий спектр юридических и бухгалтерских услуг, включая консультации по вопросам налогообложения, правового регулирования деятельности компаний,

составление и анализ договоров, юридическое сопровождение сделок и транзакций, а также ведение бухгалтерского учета и отчетности.

Основной вид деятельности организации: деятельность в области права, бухгалтерского учета и аудита, консультирование по вопросам коммерческой деятельности и управления предприятием.

Дополнительные виды деятельности:

– деятельность в области права;

– деятельность в области бухгалтерского учета и аудита;

– деятельность в области бухгалтерского учета;

– аудиторская деятельность;

– предоставление услуг по трудоустройству;

– предоставление услуг по подбору персонала;

– предоставление прочих услуг.

Таким образом, можно резюмировать, что основным видом деятельности ООО «Департамент финансовых услуг» является «Деятельность в области права, бухгалтерского учета и аудита, консультирование по вопросам коммерческой деятельности и управления предприятием».

Также организация предоставляет дополнительные виды услуг, общее количество которых равняется 7-ю позициями.

ООО «Департамент финансовых услуг» в г. Тольятти оказывает свои услуги свыше 10 лет, что говорит о большом опыте и надежности компании. За это время ООО «Департамент финансовых услуг» завоевало доверие клиентов и зарекомендовало себя как надежный и профессиональный партнер в области финансовых услуг.

Компания постоянно совершенствует свои сервисы и следит за изменениями на рынке, чтобы предложить своим клиентам наиболее актуальные и выгодные условия.

1.2 Анализ бизнес-процессов (IDEF0)

Бизнес-процессы – это последовательность действий, которые выполняются в организации для достижения ее целей. Они являются основой функционирования любой компании и могут включать в себя такие процессы, как производство товаров, обработку заказов, управление финансами, маркетинг и т.д [1, с. 54].

Анализ бизнес-процессов организации позволяет выявить сильные и слабые стороны ее деятельности, а также найти пути для повышения эффективности и оптимизации затрат.

IDEF0 (Integration Definition for Function Modeling) — это метод моделирования бизнес-процессов, разработанный ВВС США. Он используется для описания функций, действий и отношений внутри организации, а также входов и выходов каждого процесса. Модели IDEF0 можно использовать для анализа и улучшения существующих процессов, а также для разработки новых [3, с. 98]. Обозначения, используемые в диаграммах IDEF0, основаны на прямоугольниках, стрелках и других символах, которые представляют различные элементы процесса, такие как входы, выходы, элементы управления и механизмы.

Контекстная диаграмма процесса обработки заявки отражена на рисунке 2.

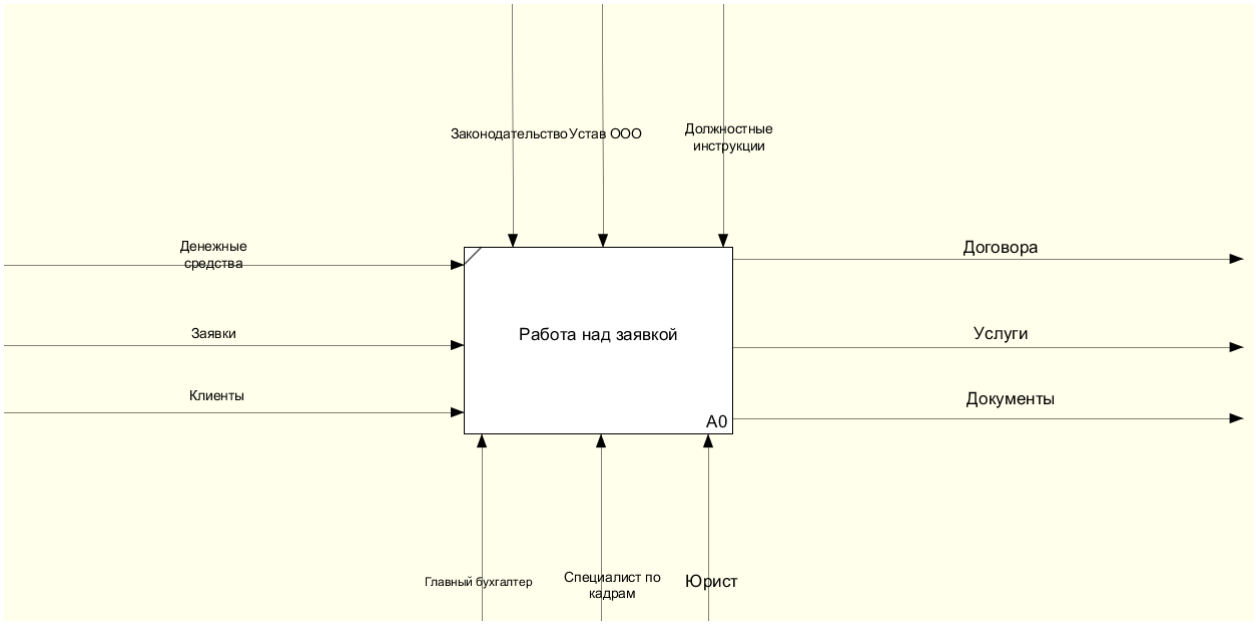

Рисунок 2 – Контекстная диаграмма процесса работы над заявкой

На рисунке 3 отражена IDEF0-диаграмма бизнес-процессов «как есть».

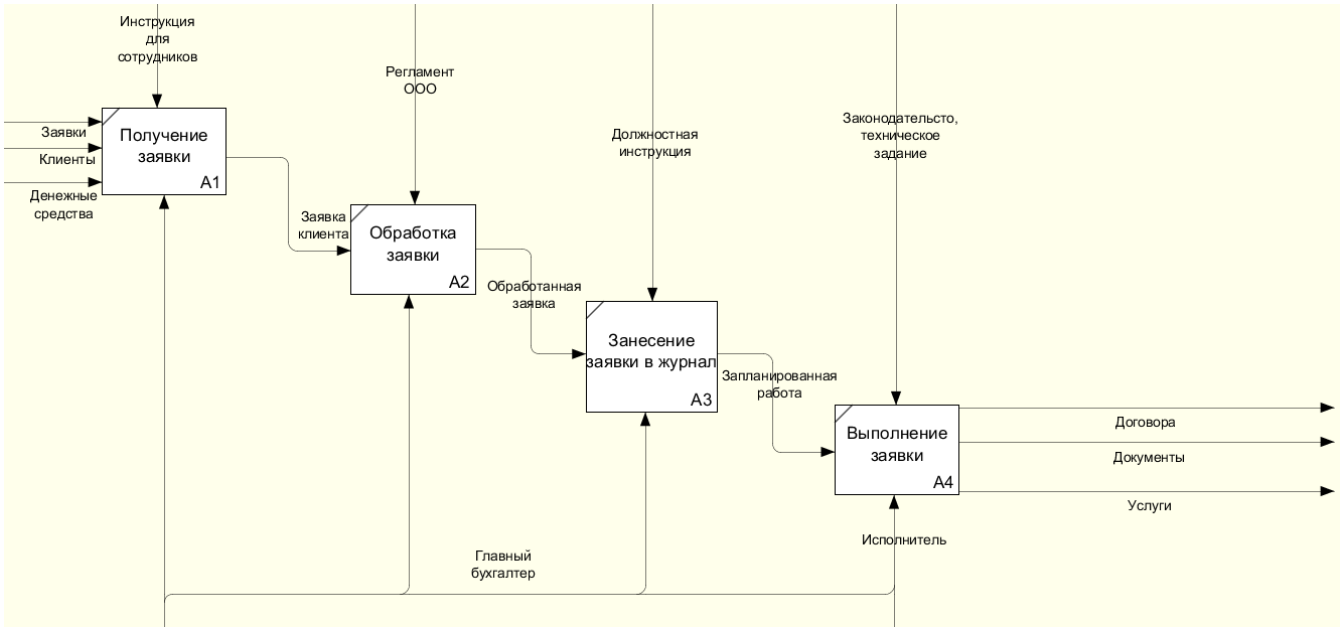

Рисунок 3 – IDEF0-диаграмма работы над заявкой ООО «Департамент финансовых услуг», as-is

Каждый сотрудник организации имеет свою спецификацию: бухгалтерскую или юридическую. При поступлении заказа, в зависимости от его специфики, выбирается исполнитель. Некоторые, наиболее масштабные

проекты, делаются сотрудниками сообща. При этом в компании отсутствует какая-либо методология управления проектами.

Одной из наиболее важных проблем является неосведомленность о статусе работы между сотрудниками, то есть, наблюдается низкая кросс-функциональность сотрудников, что приводит к торможению процессов работы, снижает качество конечного продукта, а зачастую и срыв сроков исполнения.

Следующей вытекающей проблемой является низкое качество управления ресурсами – слишком частые переключения сотрудников с одного вида работ на другие, дублирование усилий, избыточный уровень психологических нагрузок, недостаточная мотивация и неблагоприятный психологический климат в организации.

Исправить данную проблему возможно внедрением четкого планирования задач в стиле гибкой методологии управления проектами «Канбан» [6, с. 400]. Методология основана на том, что каждый из сотрудников получает так называемые «карточки», на которых указаны запланированные задачи, срок их исполнения. То есть, задачи распределяются между сотрудниками и «упустить» из виду ту или иную задачу становится невозможно [2, с. 100]. В современном мире информационных технологий процесс «выдачи карточек» может быть значительно упрощен, для этого достаточно использовать программное обеспечение «Планировщик задач», которое будет присылать уведомление о предстоящей задаче [7]. Благодаря автоматизированному планированию в ООО «Департамент финансовых услуг» значительно повысится трудовая эффективность. На рисунке 4 отражена диаграмма IDEF0 «как должно быть».

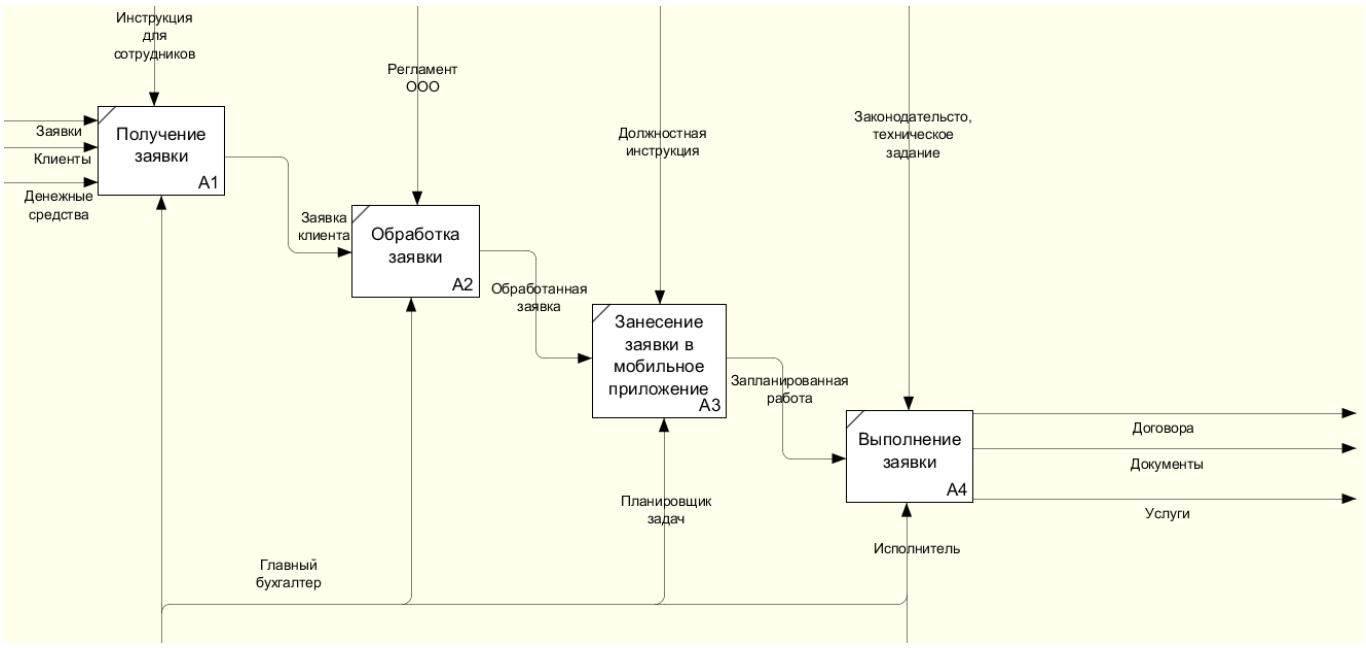

Рисунок 4 – IDEF0-диаграмма работы над заявкой в ООО «Департамент финансовых услуг», to-be

Далее необходимо описать процесс работы над заявкой при помощи мобильного приложения (рис.5).

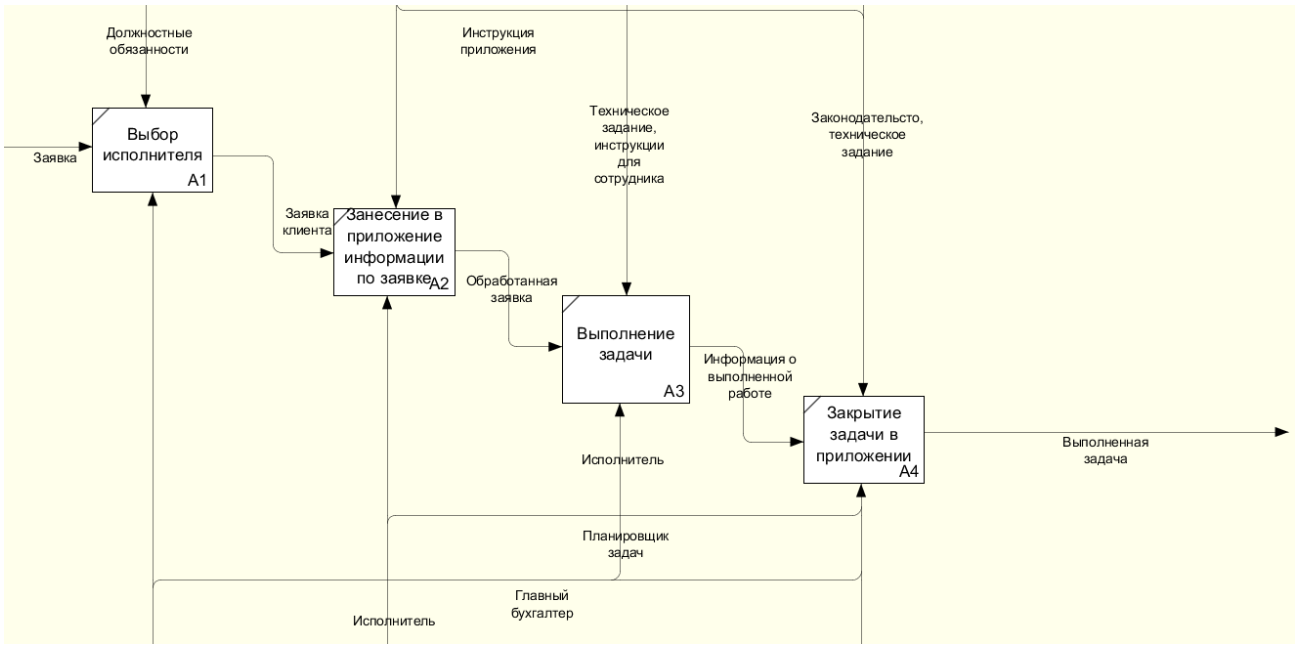

Рисунок 5 – Работа над заявкой в мобильном приложении

Таким образом, планирование позволит сократить издержки, повысить качество продукции или услуг, а также увеличить удовлетворенность клиентов.

1.3 Обзор существующих решений в области программного обеспечения «планировщик задач»

На данный момент существует множество мобильных приложений для планирования задач, которые могут помочь управлять временем и улучшить продуктивность.

Необходимо рассмотреть наиболее популярные приложения.

Todoist – это многофункциональный планировщик задач, который позволяет управлять своим временем и улучшить продуктивность. Это одно из самых популярных приложений для планирования задач, используемых миллионами пользователей по всему миру [12, с. 736].

Основная цель Todoist – помочь пользователям управлять своими задачами, устанавливать приоритеты, устанавливать сроки и создавать списки задач [17, с. 765]. Приложение имеет интуитивно понятный интерфейс и множество функций, таких как функция повторения задач, напоминания и уведомления о сроках [5, с. 16].

Todoist имеет возможность интеграции с другими приложениями, что позволяет синхронизировать задачи между устройствами. Это значит, что пользователь может работать с одним и тем же списком задач на своем компьютере, смартфоне и планшете. Todoist также интегрируется с другими популярными сервисами, такими как Google Календарь, Dropbox, Zapier и Slack [10].

Основные возможности Todoist:

– создание задач – пользователь может создавать задачи, указывать их описание и устанавливать приоритеты;

– списки задач – Todoist позволяет создавать списки задач, что делает процесс организации задач более структурированным и удобным;

– функция повторения задач – Todoist позволяет настроить повторение задач, например, ежедневно, еженедельно или каждый год. Это может быть особенно полезно для задач, которые необходимо выполнять регулярно;

– установка сроков – пользователь может устанавливать сроки выполнения задач, что помогает организовать свой график и не пропустить важные дела;

– прикрепление файлов – Todoist позволяет прикреплять файлы к задачам, например, фотографии, документы или ссылки на веб-страницы. Это может быть полезно для более подробного описания задачи или для хранения дополнительной информации.

– уведомления о сроках – Todoist отправляет уведомления пользователю о приближающихся сроках выполнения задач;

– интеграция с другими приложениями – Todoist интегрируется с другими популярными сервисами, такими как Google Календарь, Dropbox, Zapier и Slack. Это позволяет пользователю синхронизировать задачи между различными приложениями и работать более эффективно.

– избранные задачи – пользователь может отмечать задачи как избранные, чтобы быстрее находить их в общем списке задач;

– цветовая маркировка – Todoist позволяет пользователю цветовым кодом помечать задачи для лучшей организации и быстрого поиска;

– делегирование задач – пользователь может делегировать задачи другим пользователям Todoist, например, коллегам или членам семьи, чтобы лучше распределить ответственность.

Todoist доступен на многих платформах, включая Windows, Mac, iOS, Android, а также в браузерах.

Таким образом, такой планировщик задач можно использовать как в корпоративных, так и в индивидуальных целях [8, с. 1040].

Дизайн приложения отражен на рисунке 6.

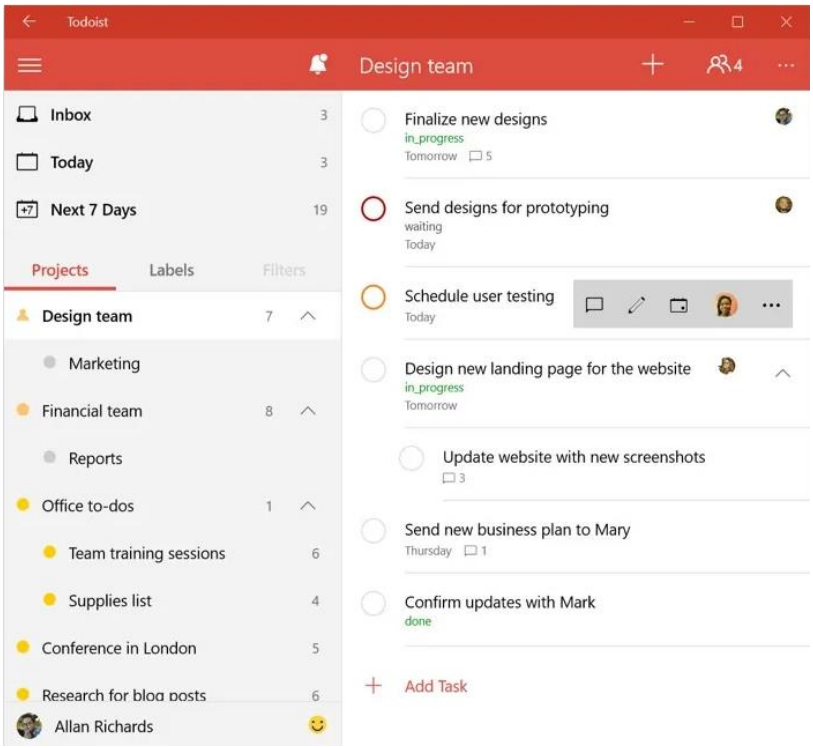

Рисунок 6 – Приложение Todoist

Asana – это приложение для управления задачами и проектами, которое позволяет пользователям создавать, отслеживать и управлять задачами на любых устройствах, включая мобильные телефоны и планшеты.

Ключевые функции Asana мобильного приложения включают в себя [10]:

– создание задач – пользователи могут легко создавать новые задачи, присваивать заголовки, описания, сроки выполнения, приоритеты и теги;

– управление задачами – Asana позволяет пользователям легко управлять задачами, отслеживать их прогресс, добавлять комментарии, вложения и даже перетаскивать их в различные проекты и категории;

– уведомления – Asana уведомляет пользователей о новых задачах, изменениях в задачах, приближающихся сроках и других событиях в режиме реального времени. Пользователи могут настроить уведомления, чтобы получать оповещения только о тех событиях, которые им интересны;

– просмотр календаря – приложение позволяет пользователям просматривать календарь задач, чтобы легко увидеть все задачи, которые назначены на определенные даты. Они могут также просматривать календарь по категориям, проектам и командам;

– совместная работа – пользователям предоставляется возможность работать вместе с другими участниками команды над задачами и проектами. Они могут приглашать коллег в Asana, делиться задачами и проектами, а также отслеживать прогресс других участников команды;

– интеграции – имеется возможность интегрироваться с другими приложениями и инструментами, такими как Google Drive, Slack, Zoom и многие другие. Это позволяет пользователям легко связывать Asana с другими инструментами, которые они уже используют в своей работе.

Данное мобильное приложение предоставляет пользователям интуитивно понятный интерфейс и удобный способ управления задачами и проектами из любой точки мира. Интерфейс отражен на рисунке 7 [9, с. 81].

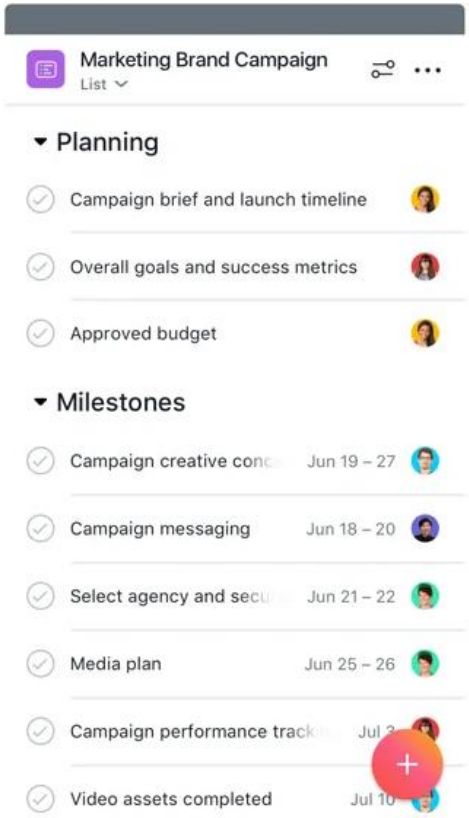

Рисунок 7– Приложение Asana

Basecamp – это универсальный инструмент для управления проектами и коммуникаций, доступный в веб-версии и мобильных приложениях для iOS и Android устройств. Вот несколько ключевых функций мобильного приложения Basecamp[10]:

– управление проектами – пользователи могут создавать проекты, добавлять задачи, файлы, комментарии и другую информацию, необходимую для организации и управления проектом [13, с. 256].;

– задачи – задачи можно легко создавать, удалять, редактировать и присваивать ответственных за их выполнение. Пользователи могут устанавливать сроки выполнения задач, прикреплять файлы и документы, а также отслеживать прогресс их выполнения;

– сообщения – Basecamp позволяет пользователям отправлять сообщения внутри проектов, создавать чаты и дискуссии, делиться файлами и документами, а также присваивать задачи и комментарии;

– календарь – можно просматривать календарь задач и событий, чтобы легко планировать свое время и контролировать сроки выполнения задач.

– совместная работа – Basecamp позволяет работать над проектами в команде, добавлять пользователей в проекты, делиться информацией и задачами, обсуждать детали проекта и отслеживать прогресс работ;

– интеграции – приложение интегрируется с другими инструментами и сервисами, такими как Dropbox, Google Drive, Zapier, Trello и многие другие. Это позволяет пользователям связывать Basecamp с другими инструментами, которые они уже используют в своей работе. Интерфейс отражен на рисунке 8.

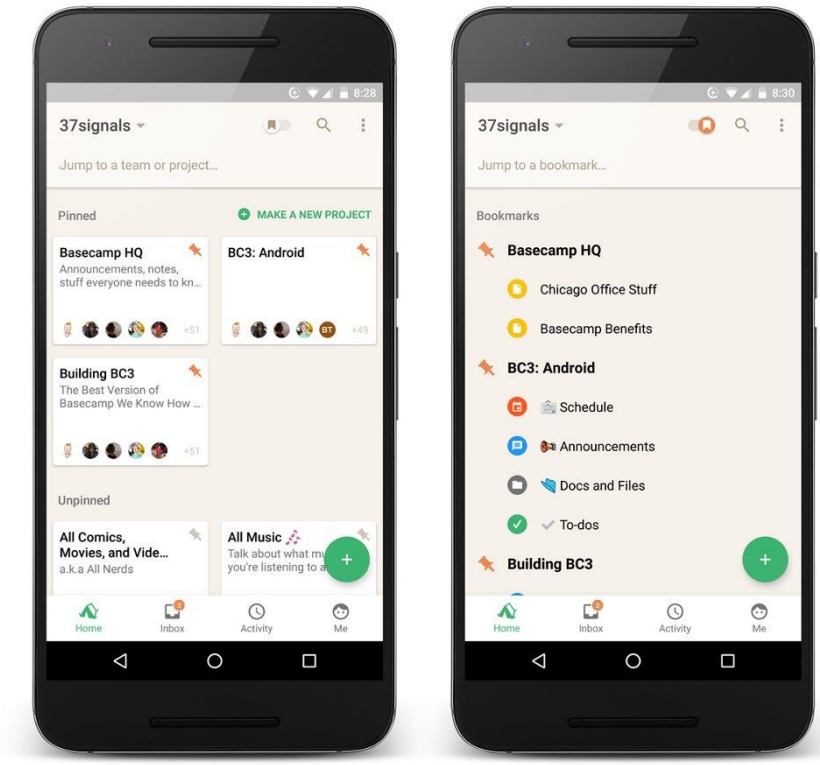

Рисунок 8 – Интерфейс Basecamp

Таким образом, рассмотрев несколько наиболее популярных решений, , было выявлено, что необходима разработка собственного приложения для организации. К основным причинам можно отнести следующие причины [10]:

1. Адаптация к специфическим потребностям компании - разработка собственного приложения позволяет создать инструмент, который полностью соответствует потребностям рассматриваемой компании [14, с. 128]. Практика использования приложения покажет – какие функции стоит добавить, доработать, а какие, наоборот, не нужны. Собственное приложение позволит также настраивать интерфейс и функциональность приложения под свои нужды.

2. Улучшение эффективности и продуктивности – с помощью собственного приложения появится возможность создавать персонализированные списки задач и управлять ими так, как удобно именно вам. Это может увеличить эффективность и продуктивность работы.

3. Повышение безопасности данных – при разработке собственного приложения для планирования задач можно убедиться в том, что данные компании защищены от несанкционированного доступа и хранятся на серверах, которые соответствуют требованиям по безопасности.

4. Экономия времени и ресурсов – собственное приложение может значительно упростить и ускорить процессы планирования задач, что позволит сотрудникам компании сосредоточиться на более важных задачах и экономить время и ресурсы. [16, с. 176].

Дополнительная возможность для монетизации – приложение можно использоваться для широкого круга пользователей, поэтому это может стать дополнительной возможностью для монетизации [19, с. 256].

Сравнительный анализ приложений отражен в таблице 1.

Таблица 1 – Сравнительный анализ рассмотренных приложений

Функционал	Todoist	Asana	Basecamp
Задачи и проекты	Да	Да	Да
Управление временем	Встроенные инструменты управления временем	Встроенные инструменты управления временем	Не встроено, но поддерживает интеграции
Коммуникация	Комментарии к задачам	Комментарии и общение в команде	Встроенный чат и форум
Календарь	Да	Да	Да
Интеграции	Интеграция со многими популярными приложениями	Интеграция со многими популярными приложениями	Ограниченная интеграция
Мобильное приложение	Да	Да	Да
Цена	Бесплатная версия, платные начинаются от $3/месяц	Бесплатная версия, платные начинаются от $10.99/пользователь/месяц	$99/месяц, не зависит от количества пользователей

В целом, разработка собственного приложения для планирования задач станет отличным инструментом для оптимизации рабочих процессов, увеличения эффективности и улучшения безопасности данных в компании.

1.4 Определение требований к системе

Требования к системе отражены в таблице 2.

Таблица 2 – Основные требования к системе

Требования	Статус	Полезность	Риск	Стабильность
Функциональные требования				
Приложение должно быть совместимо с современными серверами (Apache Web Server 2.0 и выше)	Одобренная	Высокая	Средний	Средняя
Непрерывное подключение к базе данных MySQL	Одобренная	Критичная	Средний	Средняя
Совместимость с Android версии 6.0 и выше, совместимость с IOS 8 и выше;	Одобренная	Важная	Низкий	Высокая
Возможность хранения выполненных задач в архиве	Предложенные	Средняя	Низкий	Средняя
Поддержка разрешений qHD (540×960) HD (1280×720) FHD (1920×1080) QHD (2560×1440) WXGA (1280×800) WQXGA (2560×1600);	Одобренная	Высокая	Низкий	Высокая
Отчетность и аналитика выполненных задач	Предложенные	Высокая	Средний	Средняя
Требования к надежности				
Аутентификация пользователя	Одобренные	Очень высокая	Низкий	Средняя
Автономное создание резервной копии базы данных	Предложенные	Критичная	Средний	Средняя
Уведомления	Одобренные	Высокая	Низкий	Очень высокая
Требования к удобству использования				
Современный интуитивный Пользовательский интерфейс	Одобренные	Очень высокая	Средний	Очень высокая

Продолжение таблицы 2

Возможность совместной работы над проектом	Одобренные	Очень высокая	Высокий	Средняя
Требования к производительности				
Автономная функциональность	Одобренные	Высокая	Средний	Средняя
Время отклика приложения не должно превышать 5 секунд	Предложенные	Высокая	Средний	Средняя
Требования к поддержке				
Интеграция с другими инструментами	Предложенные	Средняя	Средний	Средняя
Максимально время полного восстановления работы приложения –24 часа	Предложенные	Высокая	Высокий	Низкая
Ограничения проектирования				
Язык программирования Kotlin	Одобренные	Высокая	Низкий	Средняя

Ниже описаны общие требования для приложения «Планировщик задач»:

1. Аутентификация пользователя. Мобильное приложение должно позволять пользователям безопасно аутентифицироваться в приложении, используя свой корпоративный адрес электронной почты и пароль.

2. Управление задачами. Приложение должно позволять пользователям создавать, назначать задачи и управлять ими, а также устанавливать сроки и приоритеты для каждой задачи [20, с. 413].

3. Совместная работа. Приложение должно поддерживать совместную работу членов команды, позволяя им обмениваться задачами, файлами и комментариями.

4. Уведомления. Приложение должно уведомлять пользователей, когда им назначаются задачи, приближаются крайние сроки или когда задачи выполнены.

5. Интеграция с другими инструментами [21, с. 362]. Приложение должно интегрироваться с другими инструментами, обычно используемыми в организации, такими как программное обеспечение для управления проектами, системы управления документами и инструменты обмена сообщениями.

6. Безопасность: приложение должно обеспечивать шифрование всех данных во время передачи и хранения и должно соответствовать соответствующим законам о защите данных и конфиденциальности.

7. Пользовательский интерфейс. Приложение должно иметь понятный и интуитивно понятный пользовательский интерфейс, который позволяет пользователям легко перемещаться и выполнять задачи.

8. Кроссплатформенная совместимость: приложение должно быть совместимо с мобильными устройствами iOS и Android.

9. Автономная функциональность: приложение должно иметь автономную функциональность, позволяющую пользователям получать доступ к своим задачам и обновлять их, даже если они не подключены к Интернету.

10. Отчетность и аналитика. Приложение должно предоставлять функции отчетности и аналитики, чтобы позволить руководству отслеживать выполнение задач, производительность пользователей и другие соответствующие показатели.

Технические требования:

– совместимость с Android версии 6.0 и выше;

– совместимость с IOS 8 и выше;

– возможность работы через Wi-fi, а также через мобильные данные;

– поддержка разрешений qHD (540×960) HD (1280×720) FHD (1920×1080) QHD (2560×1440) WXGA (1280×800) WQXGA (2560×1600);

– приложение должно быть оптимизировано для быстрой загрузки и бесперебойной работы;

– приложение должно быть спроектировано таким образом, чтобы предотвращать несанкционированный доступ, защищать пользовательские данные и обеспечивать безопасную передачу данных [22, с. 160].;

– приложение должно иметь эффективную систему управления данными, которая может обрабатывать большие объемы данных без замедления работы приложения.

1.5 Постановка задачи на разработку приложения

Как было сказано выше – разработка программного обеспечения «Планировщик задач» является актуальным решением, так как благодаря этому появится возможность наиболее эффективного управления рабочим процессом в организации. Автором работы были определены задачи для разработки приложения. Необходимо рассмотреть основные модули, которые должны быть в приложении:

1. Аутентификация и авторизация: модуль, который позволяет пользователям входить в систему и получать доступ к своим задачам. Этот модуль должен использовать метод аутентификации «Логин-пароль».

2. Управление задачами: модуль, который позволяет пользователям создавать, редактировать и удалять задачи. Этот модуль также должен включать такие функции, как назначение задач членам команды, установка сроков выполнения и установка напоминаний. Задача должна добавляться следующим образом: «Пользователь заполняет название задачи в специально отведенном поле. В следующем поле пользователь должен ввести описание задачи. Далее необходимо выбрать точные сроки выполнения задачи, а именно дату и время. Также стоит внедрить функцию напоминанию: пользователь должен указать за какое время до «дедлайна» необходимо напомнить о данной задаче. Данная функция должна быть подключена к модулю «Уведомления и напоминания».

3. Корпоративный модуль. Модуль, который позволяет членам команды совместно работать над задачами. Этот модуль может включать такие функции, как комментарии к задачам, совместное использование файлов и уведомления о назначении задач.

4. Отчетность и аналитика: модуль, который предоставляет пользователям информацию о скорости выполнения задач, производительности команды и других важных показателях. Этот модуль может включать информационные панели, диаграммы и отчеты.

5. Уведомления и напоминания: модуль, который рассылает пользователям уведомления и напоминания о предстоящих задачах, обновлениях задач и других важных событиях.

6. Автономная функциональность: модуль, который позволяет пользователям получать доступ к своим задачам и обновлять их, даже когда они не в сети. Этот модуль может включать автономную синхронизацию и хранение данных.

Постановка задач в отношении интерфейса приложения:

1. Каждый элемент интерфейса должен быть интуитивно понятен, шрифты, подписи должны быть легко читаемыми и понятными для всех членов команды, которые работают над проектом;

2. Необходимо разработать интуитивный дизайн: приложении должно быть ярким и «понятной», с использованием цветовых кодов, меток и других визуальных элементов, чтобы было легко определить состояние каждой задачи;

3. Необходимо определить макет доски, столбцы, дизайн карточек, представляющих задачи, а также шрифты. Основная страница приложения должна быть выполнена в неярких тонах (белый, серый, светло-голубой). Карточки с задачами должны выделяться по цвету. В описании задач должен использоваться шрифт размером 12 px - 14 px. Заголовки задач должны иметь шрифт 16-18 px. Связь между пользователями должна осуществляться через сервер, на котором будет храниться back-end приложения. Развертывание приложения не сервере должно иметь следующие этапы [18, с. 90]:

1. выбор конфигурации сервера, обновление ОС;

2. установка глобальных пакетов и модулей;

3. создание виртуального окружения;

4. создание базы данных и администратора;

5. получение всех необходимых сертификатов;

6. создание связей back-end и front-end.

После разработки приложения, оно должно быть протестировано. Данный пункт включает тестирование на наличие ошибок, определение проблем с пользовательским интерфейсом и производительность. После локального тестирования, необходимо открыть доступ к приложению для пользователей. Пользователи должны получить инструкции в отношения использования приложения. Данный этап сопряжен с этапом группового тестирования, так как пользователи могут обнаружить дополнительные ошибки и несоответствия [23, с. 376]. Заключительный этап: необходимо отслеживать проблемы с приложением на всех этапах его использования. При необходимости нужно производить обновление приложения и добавлять новые функции.

Выводы по главе: В данной главе была рассмотрена необходимость разработки планировщика задач для автоматизации процессов планирования и управления задачами в рамках организации ООО «Департамент финансовых услуг». Были рассмотрены основные задачи, которые может решать планировщик, такие как оптимизация использования ресурсов, управление сроками выполнения задач, распределение задач между исполнителями и т.д. [25, с. 352]. Автоматизация планирования задач не только упрощает и ускоряет работу, но и позволяет снизить вероятность ошибок и повысить точность прогнозирования сроков выполнения задач. Это особенно важно в современных условиях, когда компании сталкиваются с высокой степенью конкуренции и необходимостью быстрого реагирования на изменения в рыночной среде. Таким образом, разработка планировщика задач является необходимым условием для эффективного управления бизнес-процессами организации и повышения ее конкурентоспособности. Таким образом, нами были определены требования будущей системы, а также произведена постановка задач на разработку.

Глава 2 Проектирование и разработка мобильного приложения «Планировщик задач»

2.1 Обоснование выбора средств разработки

При разработке мобильных приложений существует несколько интегрированных сред разработки, которые позволяют разработчикам создавать приложения для разных платформ. Необходимо рассмотреть три наиболее популярные среды разработки для мобильных приложений: Microsoft Xamarin, Eclipse и Android Studio, а также обосновать выбор Android Studio для разработки приложения «Планировщик задач» [27, с. 416].

Microsoft Xamarin – это платформа для разработки кроссплатформенных мобильных приложений на основе языка C#. Она позволяет создавать приложения для Android, iOS и Windows, используя общий код на C#. Xamarin также предоставляет интеграцию с средами разработки Visual Studio и Visual Studio for Mac [11, с. 50].

Eclipse – это свободно распространяемая интегрированная среда разработки, которая используется для разработки мобильных приложений на платформе Android. Она предоставляет набор инструментов для создания, отладки и тестирования приложений на Java.

Android Studio – это интегрированная среда разработки, которая является официальной средой разработки для мобильных приложений на платформе Android. Она предоставляет мощный набор инструментов и функций для разработки приложений, включая редактор макетов, поддержку Kotlin, встроенный генератор кода, поддержку Git и многие другие.

Необходимо рассмотреть более подробно каждую из этих сред разработки [4, с. 16]:

1. Microsoft Xamarin:

– платформа для кроссплатформенной разработки мобильных приложений на C#.

– позволяет создавать приложения для Android, iOS и Windows;

– интеграция с средами разработки Visual Studio и Visual Studio for Mac;

– предоставляет возможность переиспользовать код для разных платформ;

– поддержка языка программирования C#.

Eclipse:

– бесплатная интегрированная среда разработки для мобильных приложений на платформе Android.

– предоставляет набор инструментов для создания, отладки и тестирования приложений на Java.

– имеет большое сообщество разработчиков и обширную документацию.

– позволяет использовать множество сторонних плагинов и расширений.

Android Studio:

– официальная интегрированная среда разработки для мобильных приложений на платформе Android.

– редактор макетов, поддержка Kotlin, встроенный генератор кода, поддержка Git и многие другие функции.

– интеграция с Android SDK и Android Emulator [28, с. 240].

– обширная поддержка и документация со стороны Google [5, с. 32].

Таким образом, была выбрана среда разработки Android Studio для разработки приложения «Планировщик задач», так как она обеспечивает максимальную поддержку для разработки мобильных приложений на платформе Android, имеет мощный набор инструментов и функций, а также обширную поддержку со стороны Google. Кроме того, мы выбрали язык программирования Kotlin для разработки приложения, так как он

обеспечивает высокую скорость разработки, улучшенную безопасность и короткий и понятный синтаксис.

Что касается выбора языка программирования, то для разработки приложений на платформе Android можно использовать несколько языков, например, Java и Kotlin.

Наглядное сравнение этих языков программирования представлено в таблице 3.

Таблица 3 – Сравнение языков Kotlin и Java

Параметры	Kotlin	Java
Скорость разработки	Высокая, благодаря удобному синтаксису и меньшему количеству кода	Ниже, из-за более многословного синтаксиса
Безопасность	Улучшенная, благодаря строгой типизации и отсутствию null-значений по умолчанию	Ниже, из-за отсутствия строгой типизации и наличия null-значений
Синтаксис	Короткий и понятный, с удобным использованием расширений	Более многословный, требует больше кода для реализации
Взаимодействие с Java	Полностью совместим с Java, возможно использование существующего Java-кода	Можно использовать существующий Java-код, но не полностью совместим с Kotlin
Поддержка Android	Полная	Полная

Как видно из таблицы, Kotlin предоставляет ряд преимуществ перед Java при разработке мобильных приложений на платформе Android. Kotlin обеспечивает более короткий и понятный синтаксис, что позволяет ускорить процесс разработки и упростить поддержку кода в дальнейшем. Благодаря строгой типизации и отсутствию null-значений по умолчанию, Kotlin также предоставляет улучшенную безопасность по сравнению с Java.

Кроме того, Kotlin является полностью совместимым с Java, что позволяет использовать существующий Java-код в приложениях,

разработанных на Kotlin. На практике это означает, что разработчики, уже знакомые с Java, могут легко начать использовать Kotlin и продолжать использовать существующий Java-код, а также наоборот.

Исходя из этого, мы выбрали Kotlin для разработки приложения «Планировщик задач».

В роли архитектуры будет выступать клиент-серверная архитектура – это подход к построению программного приложения, при котором функциональность приложения разделена между клиентской и серверной частями. Клиент – это программа или устройство, которое запрашивает услуги или ресурсы от сервера. Сервер – это программа или устройство, которое предоставляет услуги или ресурсы клиенту.

Мной был выбран сервер на базе Intel Xeon E5-2690v4, 32 гб оперативной памяти. Вычислительной мощности хватит на будущее расширение аудитории приложения [29, с. 256].

Операционная система: Ubuntu Server 20.04 LTS.

Сервер приложений: для запуска приложения был выбран сервер приложений Apache Tomcat 9.

База данных: для хранения данных о задачах мы выбрали реляционную базу данных MySQL 8.0.

2.2 Проектирование элементов модели

2.2.1 Концептуальная модель

Необходимо разработать концептуальную схему приложения (см. рисунок 9).

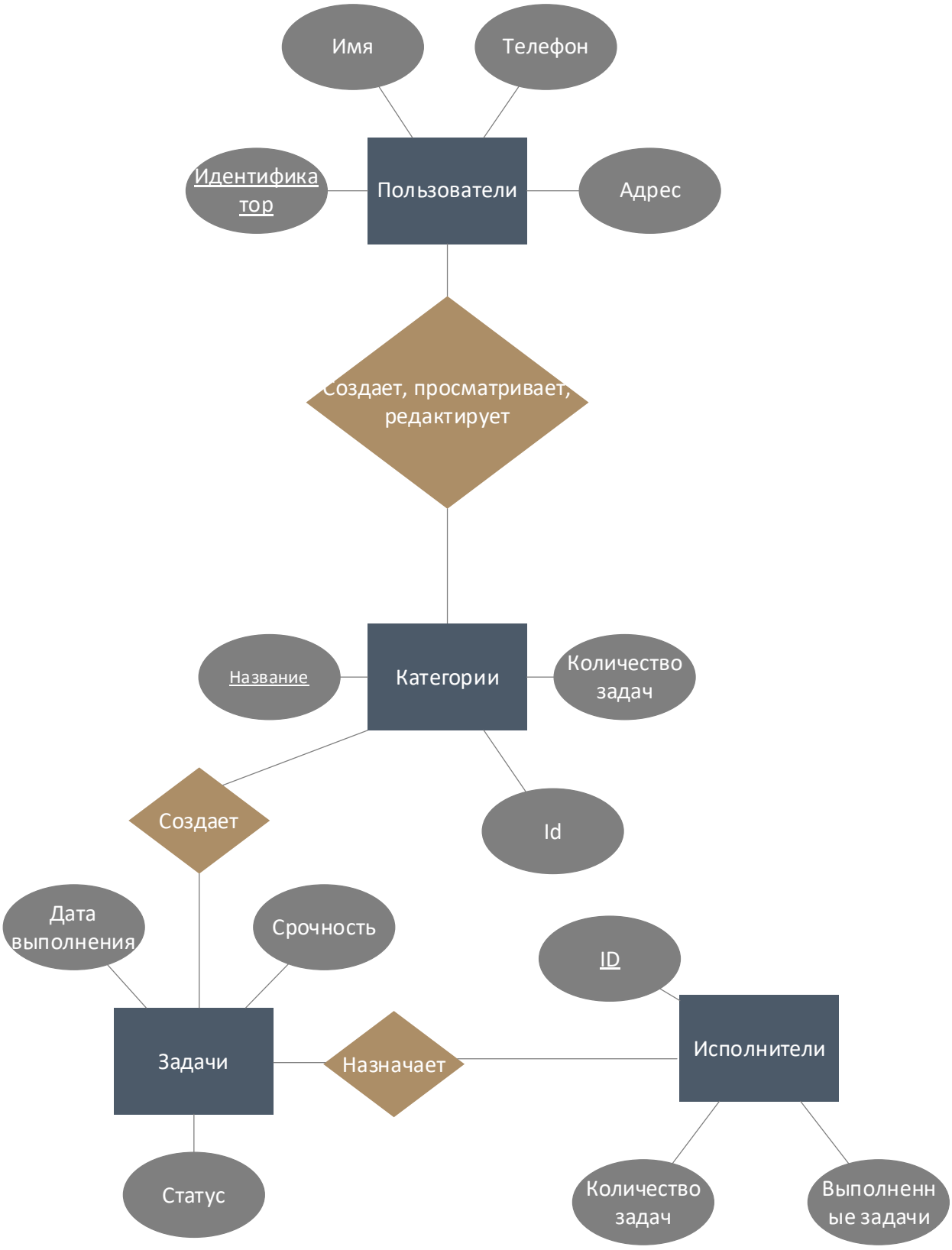

Рисунок 9 – Концептуальная модель

Концептуальная модель приложения «Планировщик задач» отображает основные сущности и связи между ними. Она включает в себя следующие сущности:

1. Пользователи – сотрудники и руководство, которое использует приложение для планирования задач.

2. Задачи – основная сущность приложения, которую создают пользователи для управления своими делами. Каждая задача содержит заголовок, описание, сроки выполнения и статус.

3. Категории – категории задач, используемые для организации и фильтрации задач. Каждая категория содержит название и описание.

4. Исполнители – пользователи, которые могут быть назначены на выполнение задачи.

Связи между сущностями:

1. Пользователи могут создавать, просматривать и редактировать задачи.

2. Задачи могут быть назначены на выполнение определенным исполнителям.

3. Задачи могут быть отнесены к определенным категориям.

4. Категории могут содержать несколько задач.

5. Каждая задача может иметь только одного исполнителя, но исполнитель может быть назначен на несколько задач.

6. Задачи могут находиться в различных состояниях: «в ожидании», «в работе», «завершена».

Концептуальная модель является основой для дальнейшего проектирования и разработки приложения. Она позволяет определить необходимые сущности и связи между ними, а также основные функциональные требования к приложению.

2.2.2 Диаграммы прецедентов и вариантов использования

Диаграммы вариантов использования (Use Case Diagrams) являются одним из ключевых инструментов анализа требований и проектирования систем. Они помогают описать функциональные возможности системы, а

также взаимодействие ее компонентов с акторами (пользователями или другими системами).

Для приложения «Планировщик задач» можно выделить несколько ключевых вариантов использования:

– регистрация нового пользователя;

– авторизация пользователя;

– создание новой задачи;

– просмотр списка задач;

– редактирование задачи;

– удаление задачи.

Каждый из этих вариантов использования может быть описан более подробно с помощью вариантов использования второго уровня. Например, для создания новой задачи можно описать следующие шаги:

– пользователь нажимает на кнопку «Создать задачу»;

– пользователь вводит название и описание задачи;

– пользователь выбирает дату и время начала и окончания задачи;

– пользователь выбирает приоритет задачи;

– пользователь сохраняет задачу.

Пример диаграммы использования отражен на рисунке 10.

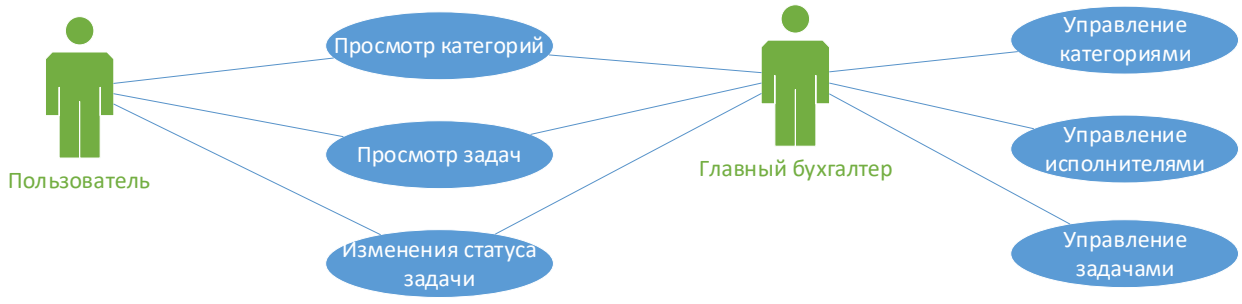

Рисунок 10 – Диаграмма вариантов использования

Диаграммы прецедентов (Use case diagram) позволяют описать функциональность системы с точки зрения пользователей и других внешних

агентов. Они помогают идентифицировать актеров системы (то есть внешние сущности, взаимодействующие с системой) и определить основные задачи, которые они выполняют.

Для приложения «Планировщик задач» можно определить следующие диаграммы прецедентов:

1. Диаграмма «Авторизация» - описывает процесс авторизации пользователя в системе. Актеры: Пользователь. Прецеденты: Вход в систему (см. рисунок 11).

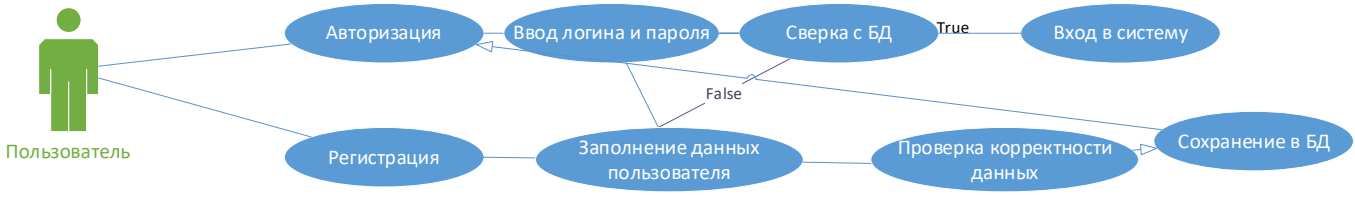

Рисунок 11– Диаграмма прецедента для пользователя приложения «Авторизация» и регистрация

2. Диаграмма «Управление задачами» - описывает основные действия, которые пользователи могут выполнить с задачами в системе. Актеры: Главный бухгалтер (администратор). Прецеденты: Создание задачи, Редактирование задачи, Удаление задачи, Назначение задачи на исполнителя, Просмотр списка задач (см. рисунок 12).

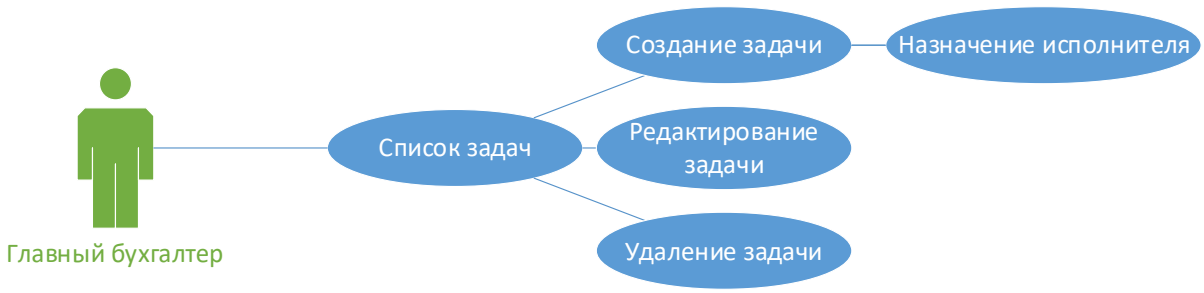

Рисунок 12 – Диаграмма прецедента для главного бухгалтера «Управление задачами»

При этом, пользователи не могут создавать задачи.

3. Диаграмма «Управление категориями» - описывает действия, которые пользователи могут выполнить с категориями задач. Актеры: Главный бухгалтер (администратор). Прецеденты: Создание категории, Редактирование категории, Удаление категории, Просмотр списка категорий (см. рисунок 13).

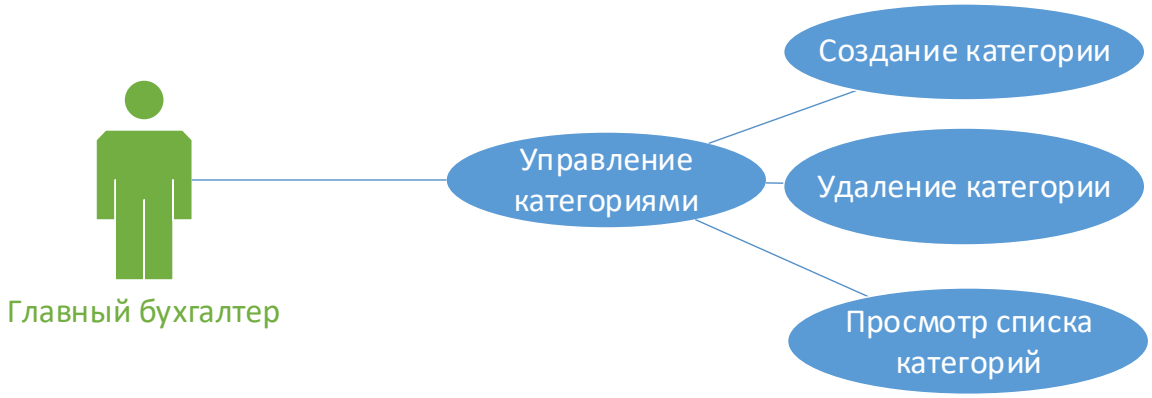

Рисунок 13 – Диаграмма прецедента «Управление категориями»

При этом пользователи могут лишь просматривать категории.

4. Диаграмма "Управление исполнителями" - описывает действия, которые пользователи могут выполнить с исполнителями задач. Актеры: Главный бухгалтер. Прецеденты: Добавление исполнителя, Удаление исполнителя, Просмотр списка исполнителей (см. рисунок 14).

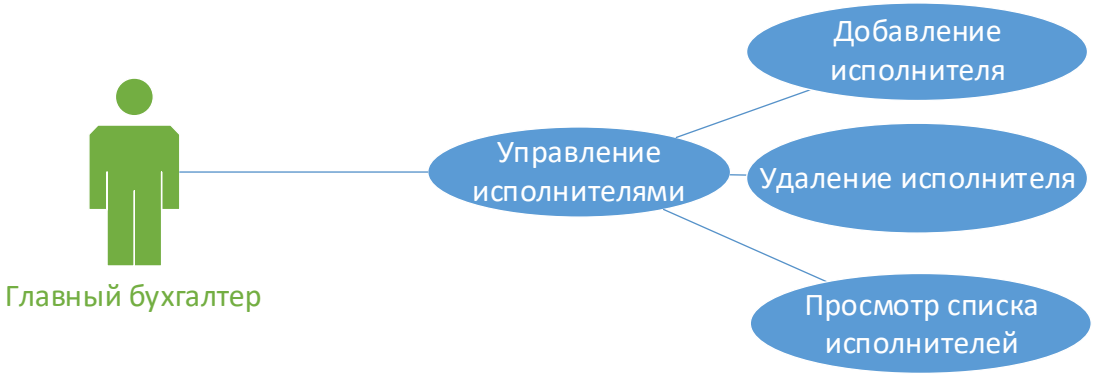

Рисунок 14 – Диаграмма прецедента «Управление исполнителями»

Для пользователей этот функционал отсутствует.

Диаграммы прецедентов помогают определить основные функциональные требования к системе и упрощают ее понимание для всех участников проекта.

2.3 Логическое и физическое моделирование

Ниже рассмотрено логическое моделирование приложения.

Класс «Task» представляет собой конкретную задачу в контексте планировщика задач. У каждой задачи есть уникальный идентификатор (id), название (title), подробное описание (description), категория (category), указатель на пользователя, который назначен для выполнения этой задачи (assignee), срок выполнения (due_date), приоритет (priority) и текущий статус (status), который может быть «В ожидании», «В процессе» или «Завершено».

Класс «Category» используется для группировки задач по категориям. У каждой категории есть уникальный идентификатор (id) и название (name).

Класс «User» представляет пользователя, который может быть назначен на выполнение задач. Каждый пользователь имеет уникальный идентификатор (id), имя (name) и роль (role). Роль может быть любой, например, «Главный бухгалтер» или «Менеджер проекта» [30, с. 144].

Класс «TaskManager» предоставляет методы для управления задачами. Это включает создание новой задачи (create_task), получение задачи по идентификатору (get_task), обновление задачи по идентификатору (update_task), удаление задачи по идентификатору (delete_task), получение списка задач по идентификатору категории (get_tasks_by_category) и назначение исполнителя для задачи (assign_task).

Наконец, класс «UserManager» предоставляет методы для управления пользователями. Это включает создание нового пользователя (create_user), получение пользователя по идентификатору (get_user), обновление пользователя по идентификатору (update_user), удаление пользователя по

идентификатору (delete_user) и получение списка пользователей по роли (get_users_by_role).

Логическая модель данных отражена на рисунке 15.

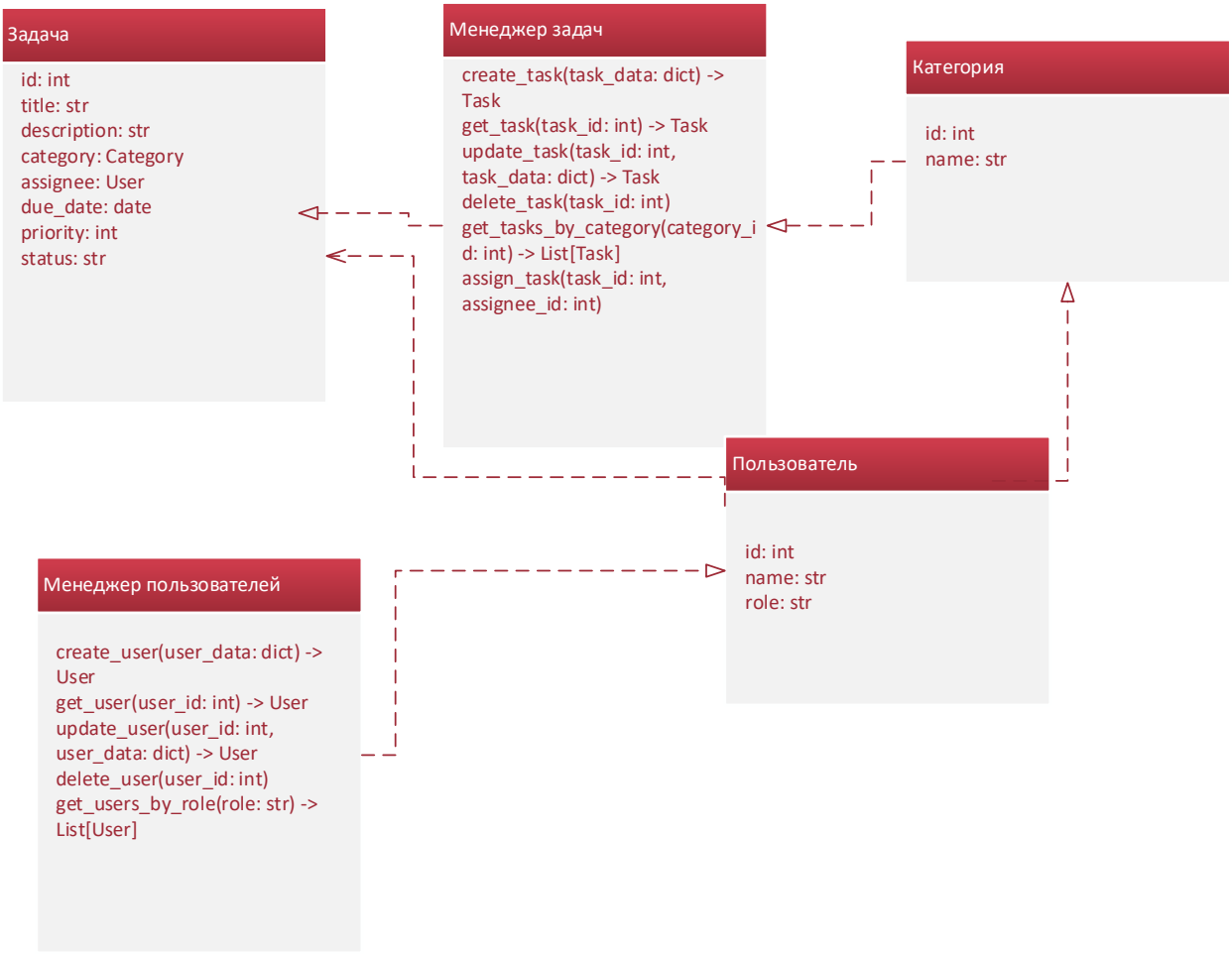

Рисунок 15 – Логическая модель данных приложения

Физическое моделирование приложения описывает архитектуру системы, включая компоненты, используемые технологии и сетевые протоколы.

Для приложения будет использоваться клиент-серверная архитектура, где клиентом является мобильное приложение, а сервером - база данных и бэкэнд-сервер.

При проектировании взаимодействия между серверной и клиентской частями приложения, необходимо рассмотреть общую архитектуру. Так, в нашем случае будет использоваться архитектуру REST, которая представляет собой стандартизированный подход к организации взаимодействия между клиентом и сервером.

На серверной стороне будет использоваться Django, который обеспечивает обработку серверной логики и предоставляет возможность создания API. В то же время API, или программный интерфейс приложения, представляет собой набор правил и протоколов для обмена данными между сервером и клиентом [26, с 49].

На клиентской стороне будет использоваться Axios для отправки HTTP-запросов к API. Данные запросы могут варьироваться по типу: можно отправлять запросы GET для получения данных, запросы POST для отправки данных, запросы PUT для обновления данных и запросы DELETE для удаления данных.

Данные между сервером и клиентом будут передоваться в формате JSON, так как он легко обрабатывается как на сервере, так и на клиенте, что обеспечивает гладкое взаимодействие между ними.

Важным аспектом при проектировании взаимодействия между сервером и клиентом является обработка ошибок. Необходимо предусмотреть различные коды статуса HTTP и сообщения об ошибках, чтобы корректно обрабатывать различные ситуации, которые могут возникнуть.

Не менее важным является вопрос безопасности. Для этого в Django будет использоваться система аутентификации, которая предлагает функции для регистрации, входа и выхода пользователей, а также проверки прав доступа.

На клиентской стороне необходимо использоваться Alpine.js для обработки и отображения данных, полученных от сервера. Он также будет за отправку данных на сервер с использованием Axios

Взаимодействие клиента с сервером отражена на рисунке 16.

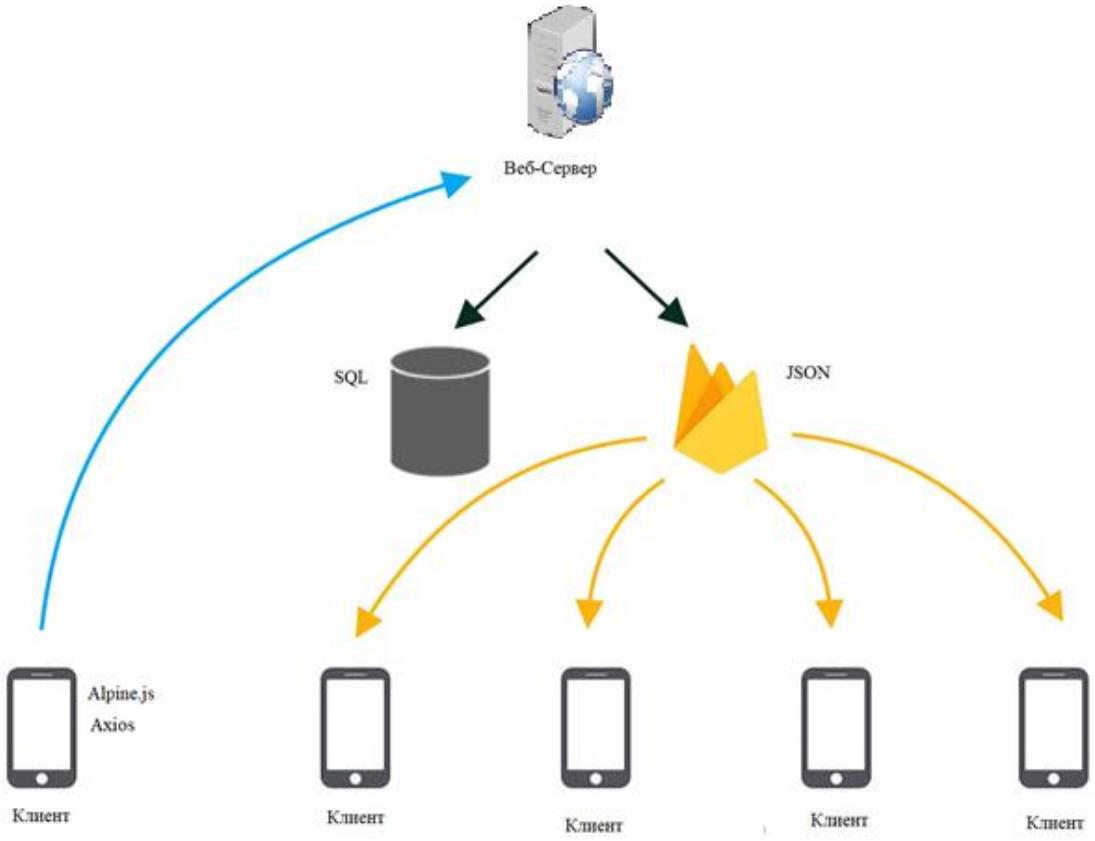

Рисунок 16– Взаимодействие клиента с сервером

Физическая модель базы данных отражена на рисунке (см. рисунок 17).

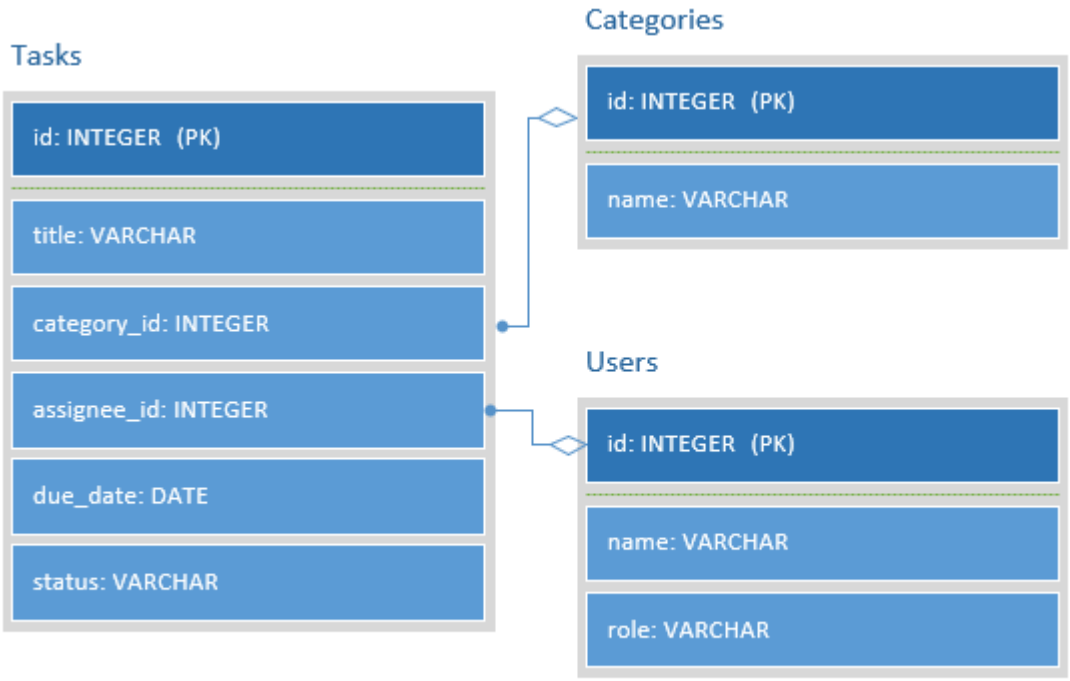

Рисунок 17 – Физическая модель базы данных

Таблица Tasks является местом хранения данных о задачах. В ней есть уникальный идентификатор каждой задачи, представленный как INT, который служит главным ключом. Название каждой задачи хранится в поле title типа VARCHAR, что позволяет использовать строки длиной.Каждая задача имеет поле description типа TEXT для хранения описания задачи. Это поле может принимать строковые значения произвольной длины. Задача также связана с определенной категорией через поле category_id типа INT, которое ссылается на соответствующую запись в таблице Categories. Аналогичным образом, через поле assignee_id типа INT задача связывается с пользователем, которому она назначена, что обеспечивается ссылкой на запись в таблице Users. Каждая задача имеет дату выполнения, хранящуюся в поле due_date типа DATE, приоритет, хранящийся в поле priority типа INT, и статус, хранящийся в поле status типа VARCHAR.

Таблица Categories служит для хранения информации о различных категориях задач. У каждой категории есть уникальный идентификатор, хранящийся в поле id типа INT, и название, хранящееся в поле name типа VARCHAR.

Таблица Users содержит данные о пользователях. У каждого пользователя есть уникальный идентификатор, хранящийся в поле id типа INT, и имя, хранящееся в поле name типа VARCHAR(255). Также в поле role типа VARCHAR указывается роль пользователя.

2.4 Разработка интерфейсов и архитектурной модели приложения

В приложении будут следующие модели:

– Экран пользователя (UserModel);

– Экран задачи (TaskModel);

– Экран категории задач (CategoryModel);

– Экран исполнителя задач (ExecutorModel).

Каждая из этих моделей будет иметь свои свойства и методы. Например, для модели задачи определены следующие свойства:

– Название задачи;

– Описание задачи;

– Дата выполнения;

– Категория задачи;

– Исполнитель задачи.

Для отображения информации пользователю будут реализованы следующие представления:

– Экран авторизации/регистрации пользователя (Login/Register Activity);

– Главный экран приложения (Main Activity);

– Экран создания/редактирования задачи (Create/Edit Task Activity);

– Экран просмотра списка задач (Task List Activity);

– Экран просмотра списка категорий задач (Category List Activity);

– Экран просмотра списка исполнителей задач (Executor List Activity).

В каждом из этих представлений будут определены элементы пользовательского интерфейса (например, кнопки, текстовые поля и т.д.), которые позволят пользователю взаимодействовать с приложением.

ViewModel будет посредником между моделью и представлением. Данный модуль отвечает за обновление данных модели и передачу их в представление.

Контроллеры будут использованы для реализации логики приложения, такой как проверка прав доступа пользователя и обработка событий. Контроллер будет обрабатывать нажатия на кнопки и выполнять соответствующие действия (например, создание новой задачи или редактирование существующей).

Для ускорения разработки и повышения производительности приложения мы можем использовать сторонние библиотеки, такие как:

– Retrofit – для работы с REST API;

– Gson – для сериализации/десериализации данных;

– Glide – для загрузки и кэширования изображений;

– Room – для работы с базой данных SQLite;

– Kotlin Coroutines – для асинхронной обработки и выполнения задач;

– Dagger Hilt – для управления зависимостями в приложении.

Для создания главного экрана планировщика задач мы используем фрагменты (Fragments) в Android Studio. Фрагменты – это модульные элементы пользовательского интерфейса, которые использованы в различных макетах и включены в активности. В разрабатываемом приложении главный экран будет состоять из следующих фрагментов:

– фрагмент для отображения списка задач пользователя;

– фрагмент для добавления новой задачи;

– фрагмент для редактирования задачи;

– фрагмент для отображения категорий задач;

– фрагмент для добавления новой категории;

– фрагмент для редактирования категории;

– фрагмент для отображения списка исполнителей задач;

– фрагмент для добавления нового исполнителя;

– фрагмент для редактирования исполнителя.

Также будет использоваться Navigation Component в Android Studio для навигации между фрагментами и создания единой структуры приложения.

Также при разработке будет использоваться Model-View-ViewModel (MVVM) архитектурный паттерн, чтобы обеспечить четкое разделение бизнес-логики и пользовательского интерфейса в приложении. ViewModel будет содержать бизнес-логику приложения, а View будет отвечать за отображение данных и взаимодействие с пользователем. Model будет содержать данные и их обработку.

Также необходимо использование Material Design в приложении для обеспечения графического интерфейса. Material Design предоставляет множество готовых компонентов, которые будут использоваться в приложении.

Глава 3 Разработка дизайна и тестирование приложения

3.1 Реализация дизайна приложения

При разработке дизайна приложения были использованы различные инструменты и методы. Было проведено исследование рынка и проанализирован дизайн других приложений для планирования задач, чтобы определить, какие элементы должны быть включены в дизайн и как их нужно организовать на экране.

Для создания макетов страниц приложения были использованы специализированные инструменты для проектирования интерфейсов, такие как Figma, Sketch. В процессе работы использовались готовые компоненты и элементы дизайна, а также создавались собственные элементы, соответствующие концепции приложения.

При разработке дизайна приложения учитывались принципы дизайна пользовательского интерфейса, такие как простота, понятность, эргономичность и красота. Кроме того, были учтены требования к адаптивности и совместимости с различными устройствами и операционными системами.

В результате работы над дизайном, были получены качественные макеты страниц приложения, соответствующие концепции и требованиям проекта. Они были использованы в дальнейшей работе по разработке приложения, что позволило создать удобный и интуитивно понятный дизайн, который позволяет пользователям легко и быстро находить нужную информацию и выполнять необходимые действия. На рисунках отражен разработанный интерфейс приложения (см. рисунок 18).

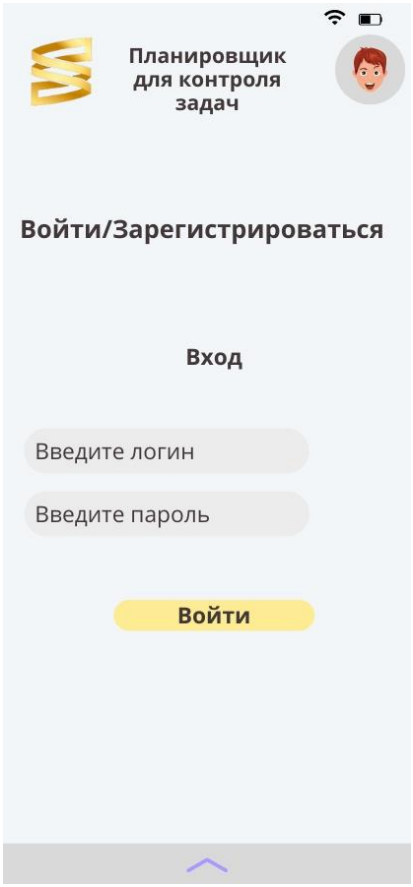

Рисунок 18 – Страница авторизации

На странице авторизации пользователи должен ввести логин и пароль, после чего данные будут сверены с базой данной, если такой пользователь есть в системе, то произойдет аутентификация и пользователь войдет в систему.

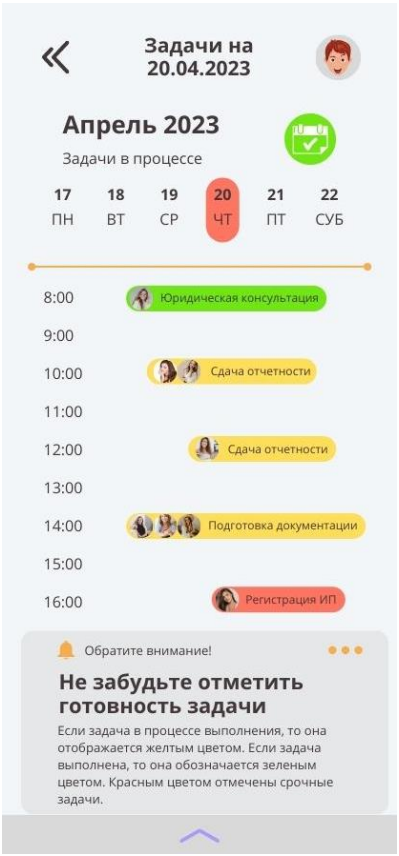

Рисунок 19 – Список задач

На данном экране отражен список задач, которые необходимо выполнить (см. рисунок 19). На рисунке 20 изображена конкретная задача.

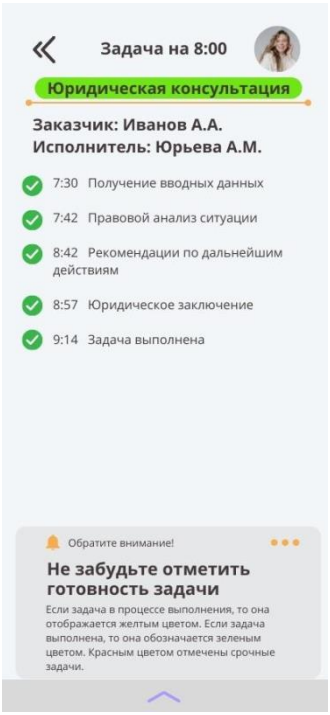

Рисунок 20– Конкретная задача

Таким образом, на рисунках была представлена форма регистрации и авторизации, список задач для конкретного пользователя, а также дизайн страницы с конкретными задачами.

3.2 Реализация модулей приложения

Приложение «Планировщик задач» будет включать в себя следующие модули:

1. Модуль регистрации и авторизации – отвечает за регистрацию пользователей и их идентификацию в системе.
2. Модуль пользовательского интерфейса (UI) - отвечает за отображение графического интерфейса приложения, включая экраны, кнопки, поля ввода, списки задач и т.д.
3. Модуль хранения данных (Data storage) – отвечает за сохранение задач пользователя и других данных, связанных с приложением. В качестве хранилища будет использоваться база данных.
4. Модуль управления задачами (Task management) – отвечает за добавление, удаление, изменение и отображение задач пользователя. В этот модуль входят функции для управления задачами, такие как добавление тегов, приоритетов, оповещений и т.д.
5. Модуль уведомлений (Notifications) – отвечает за отправку уведомлений пользователю по мере приближения дедлайна задачи или других событий, связанных с задачей.
6. Модуль статистики (Statistics) – отвечает за анализ задач пользователя и формирование статистических отчетов, которые могут быть полезны для пользователей в планировании своего времени.
7. Модуль настройки (Settings) – отвечает за настройку приложения, такие как язык интерфейса, цветовые схемы, формат времени, уведомления и т.д.

Необходимо рассмотреть модуль авторизации. В модуле авторизации используется класс User, который представляет собой запись в таблице пользователей с полями id, username и password. Класс UserDatabase наследуется от SQLiteOpenHelper и позволяет работать с базой данных пользователей.

Метод addUser добавляет нового пользователя в базу данных. Метод getUserByUsername ищет пользователя по его имени пользователя (логину) и возвращает объект User, если пользователь существует, иначе возвращает null. Метод deleteUserByUsername удаляет пользователя из базы данных по его имени пользователя.

Также осуществляется проверка, является ли пользователь обычным пользователем или администратором. Для этого в таблице пользователей добавлено поле role, которое будет указывать на роль пользователя.

Например, если поле role имеет значение 0, то пользователь является обычным пользователем, а если значение равно 1, то пользователь является администратором. В этом случае метод getUserByUsername будет возвращать не только объект User, но и значение поля role.

Код модуля авторизации отражен в приложении А.

Следующий модуль – модуль хранения данных. В модуле используется метод deleteTask, которые удаляет запись с указанным id из таблицы задач, а метод updateTask обновляет запись с указанным id в таблице задач. Оба метода используют методы delete и update объекта SQLiteDatabase, которые выполняют соответствующие операции с базой данных.

Также используются константы, определенные в блоке companion object, для имени базы данных, версии, имени таблицы и имен колонок. Это удобно, потому что появляется возможность использовать эти значения во всем классе, и если в будущем нужно будет изменить имя таблицы или колонки, то достаточно будет изменить значение только одной константы.

Код модуля отражен в приложении Б.

Следующий модуль – модуль управления задачами.

В этом модуле добавляется еще два объекта базы данных – TagDatabase и PriorityDatabase – для управления тегами и приоритетами соответственно. Метод addTask теперь принимает также список идентификаторов тегов и идентификатор приоритета, и создает объект задачи с соответствующими тегами и приоритетом. Методы getAllTasks и addTask теперь включают обновление задачи с соответствующими тегами и приоритетом. Методы addTag, getAllTags, deleteTag, updateTag, addPriority, getAllPriorities, deletePriority и updatePriority добавляют соответствующие функции для управления тегами и приоритетами.

При этом, методы для добавления тегов и приоритетов принимают не только имя тега или приоритета, но также другие параметры, такие как цвет для приоритета. Метод addTask также принимает идентификатор приоритета и список идентификаторов тегов, которые связываются с задачей [15, с. 17].

Метод getAllTasks использует объекты TagDatabase и PriorityDatabase для получения соответствующих тегов и приоритетов для каждой задачи, которые затем добавляются к задаче. Таким образом, при получении списка всех задач с помощью метода getAllTasks, мы получаем все задачи с соответствующими тегами и приоритетами.

Код модуля отражен в приложении В.

Модуль уведомлений. Был создан класс NotificationManager, который используется для отправки уведомлений пользователю. Метод createNotification создает уведомление для задачи, используя объект NotificationManager для создания канала уведомлений (для версий Android 8.0 и выше) и создания уведомления с помощью NotificationCompat.Builder. Метод setNotification устанавливает уведомление на определенное время, используя объект AlarmManager, который запускает PendingIntent для отправки уведомления. Метод cancelNotification отменяет уведомление, используя объект AlarmManager для отмены соответствующего PendingIntent.

Также используется класс NotificationReceiver, который расширяет BroadcastReceiver, для получения событий, связанных с уведомлениями. Этот

класс может содержать логику для обработки событий, связанных с уведомлениями, таких как обновление статуса задачи при нажатии на уведомление. Код отражен в приложении Г.

В методе сбора статистики отражен класс StatisticsManager, который используется для анализа задач пользователя и формирования статистических отчетов. Методы getCompletedTasksCount и getIncompleteTasksCount возвращают количество выполненных и невыполненных задач за определенный период соответственно. Методы getTasksCountByTag и getTasksCountByPriority возвращают количество задач, связанных с определенным тегом или приоритетом соответственно. Метод getAverageTaskDuration возвращает среднюю продолжительность выполнения задач за определенный период.

Эти методы используются для формирования различных статистических отчетов, которые могут помочь пользователям в планировании своего времени и улучшении своей продуктивности. Например, пользователь может использовать методы getCompletedTasksCount и getIncompleteTasksCount для оценки своей эффективности в выполнении задач за определенный период времени, а методы getTasksCountByTag и getTasksCountByPriority для оценки важности различных задач. Метод getAverageTaskDuration может быть полезен для определения среднего времени, которое пользователь тратит на выполнение задач, что может помочь ему лучше планировать свое время в будущем. Код модуля отражен в приложении Д.

В модуле настройки был создан класс SettingsManager, который используется для управления настройками приложения. Методы getLanguage, getColorScheme и getTimeFormat возвращают текущее значение языка интерфейса, цветовой схемы и формата времени соответственно. Методы setLanguage, setColorScheme и setTimeFormat устанавливают новое значение языка интерфейса, цветовой схемы и формата времени соответственно.

Методы getNotificationsEnabled и setNotificationsEnabled возвращают и устанавливают текущий статус уведомлений соответственно [24, с. 13].

Эти методы могут использоваться для управления различными настройками приложения, такими как язык интерфейса, цветовая схема, формат времени и уведомления. Например, пользователь может использовать метод getLanguage для получения текущего языка интерфейса приложения и метод setLanguage для изменения языка интерфейса на другой. Методы getColorScheme и setColorScheme могут использоваться для управления цветовой схемой приложения, а методы getTimeFormat и setTimeFormat – для управления форматом времени. Методы getNotificationsEnabled и setNotificationsEnabled могут использоваться для включения или отключения уведомлений приложения. Код модуля отражен в приложении Е.

3.3 Тестирование приложения

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

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

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

Функциональное тестирование позволяет тестировать приложение в целом, проверяя, как работают его различные функции и модули вместе. Было проведено функциональное тестирование нашего приложения, используя различные тестовые сценарии, чтобы убедиться, что все функции работают должным образом и взаимодействуют друг с другом без ошибок.

Тестирование пользовательского интерфейса позволяет проверить, насколько удобным и интуитивным является интерфейс нашего приложения для пользователя. Было проведено тестирование пользовательского интерфейса, используя тестовых пользователей, чтобы оценить, насколько легко пользователи могут выполнить различные задачи в приложении и обнаружить возможные проблемы с пользовательским интерфейсом.

Все выявленные ошибки и проблемы были исправлены, а приложение прошло все необходимые тесты перед его выпуском.

В таблице 3 отражены результаты тестов приложения.

Таблица 4 – Тестирование приложения

Номер теста	Описание теста	Ожидаемый результат	Фактический результат	Статус
1	Тестирование добавления новой задачи	При добавлении новой задачи в приложении должна появляться новая запись в списке задач	После добавления новой задачи в список задач была добавлена новая запись	Пройден
2	Тестирование удаления задачи	При удалении задачи из приложения должна исчезать соответствующая запись в списке задач	После удаления задачи из списка задач соответствующая запись была удалена	Пройден
3	Тестирование изменения приоритета задачи	При изменении приоритета задачи в приложении должен измениться цвет соответствующей записи в списке задач	После изменения приоритета задачи в списке задач изменился цвет соответствующей записи	Пройден

Продолжение таблицы 4

4	Тестирование оповещений	При установке оповещения на задачу, пользователь должен получить уведомление на своем устройстве в указанное время	После установки оповещения на задачу, пользователь получил уведомление в указанное время	Пройден
5	Тестирование скорости работы приложения	Приложение должно быстро реагировать на пользовательские действия и не тормозить при работе с большим количеством задач	Приложение быстро реагирует на пользовательские действия и работает стабильно даже при большом количестве задач	Пройден

В таблице 4 отражены устройства на которых проходили тесты и версии операционных систем.

Таблица 5 – Устройства, на которых тестировалось приложение

Устройство	Версия операционной системы
Samsung Galaxy S10	Android 11
Google Pixel 4	Android 10
OnePlus 8	Android 11
Xiaomi Mi 10	Android 12
Samsung Galaxy Note 20	Android 11

Как видно из таблицы, приложение тестировалось на разных устройствах с разными версиями операционных систем, проверялась его работа в различных сценариях использования. Также про проверялась скорость работы приложения и его реакция на пользовательские действия.

Для каждого теста был описан его ожидаемый результат и фактический результат, а также указывали статус теста (пройден или не пройден). Если тест не проходил, то выявленные ошибки и повторно запускали тест.

Также использовались инструменты для измерения отклика приложения, такие как Android Profiler, чтобы оптимизировать

производительность приложения и обеспечить быструю реакцию на пользовательские действия.

В целом, тестирование приложения позволило обнаружить и исправить несколько ошибок и проблем, а также улучшить его функциональность, надежность и производительность. Благодаря тестированию можно утверждать, что приложение работает должным образом и предоставляет пользователям удобный и надежный инструмент для планирования своего времени.

3.4 Оценка эффективности проекта

Одним из главных критериев оценки эффективности проекта является его влияние на бизнес-процессы организации, в данном случае ООО «Департамент финансовых услуг».

Ожидается, что внедрение планировщика задач повысит эффективность работы сотрудников, улучшит управление задачами и облегчит процесс контроля за их выполнением, а также будет уменьшено количество ошибок и пропусков задач, что приведет к более продуктивной работе и повышению удовлетворенности клиентов.

Важным критерием оценки эффективности проекта является его экономическая составляющая. В рамках проекта предполагается использование открытых технологий, что позволит снизить затраты на разработку и сопровождение приложения.

Кроме того, планируется использование готовых решений и библиотек, что также снизит затраты на разработку.

Внедрение планировщика задач в рабочий процесс компании может оказать положительное влияние на производительность и эффективность работы сотрудников.

Ожидается, что повысится качество управления задачами, а также сократится время на выполнение задач и улучшится коммуникация между участниками проектов.

Эффективность проекта также может зависеть от ряда факторов, таких как эффективность маркетинга и продвижения приложения, конкуренция на рынке аналогичных продуктов, а также способность команды разработчиков быстро адаптироваться к изменяющимся потребностям пользователей и рынка в целом.

В любом случае, имеется возможность монетизации приложения в сервисе Google Play. Некоторые из способов монетизации приложения в Google Play могут включать:

1. Реклама: Можно размещать рекламные объявления в приложении и получать доход от кликов или показов рекламы.

2. Платные функции: Можно добавить в приложение платные функции или возможности, которые будут доступны только после оплаты. Это может включать расширенный функционал, дополнительный контент или премиум-подписки.

3. Продажа приложения: Можно продавать само приложение.

Учитывая функциональность разрабатываемого приложения, можно рассмотреть возможность добавления в него платных функций, премиум-подписки.

Также можно рассмотреть эффективность с точки зрения повышения производительности, а, следовательно, на повышение прибыли.

Так, если учитывать среднестатистические данные (увеличение производительности на 20%), рассматривая чистую прибыль компании за 2022 году в 680 тыс. рублей, то произвести расчеты.

Для расчета увеличения прибыли можно использовать следующую формулу:

$$Уп = Тп \times (1 + Кпэ) \quad (1)$$

Где:

Уп – увеличение прибыли;

Тп – текущая прибыль;

Кпэ – коэффициент увеличения эффективности.

Учитывая, что планируем коэффициент повышения эффективности составляет 20% (0.2), можно рассчитать увеличение прибыли следующим образом:

$$\text{Уп} = 680\ \text{тыс.}\,\text{рублей} \times (1 + 0.2) = 816\ \text{тыс.}\,\text{рублей}$$

Таким образом, ожидается, что внедрение планировщика задач приведет к увеличению прибыли компании на 136 тыс. рублей (816 тыс. рублей – 680 тыс. рублей).

Таким образом, при правильной реализации и продвижении проекта, его эффективность и успешность могут значительно превысить затраты на разработку и внедрение.

Таким образом, внедрение планировщика задач имеет большой потенциал для повышения эффективности работы сотрудников и улучшения управления проектами в ООО «Департамент финансовых услуг». При правильной реализации проекта, возможна его монетизация и получение дополнительной прибыли.

Первым этапом была разработка дизайна приложения. Для этого были использованы специализированные инструменты для проектирования интерфейсов, такие как Figma, Sketch или Adobe XD. В процессе работы использовались готовые компоненты и элементы дизайна, а также создавались собственные элементы, соответствующие концепции приложения.

Вторым этапом была разработка приложения. Была разработана архитектура приложения и реализованы необходимые модули, такие как

модуль хранения данных, управления задачами, уведомлений, настроек и статистики. Каждый модуль был реализован с использованием соответствующих технологий и инструментов.

Третьим этапом было тестирование приложения. Были проведены тесты на функциональность, производительность, стабильность и безопасность приложения. Результаты тестирования подтвердили, что приложение работает корректно и отвечает всем требованиям, установленным в начале проекта.

В итоге, было проектировано приложение для планирования задач на платформе Android, которое удобно в использовании и соответствует всем требованиям и ожиданиям пользователей.

При правильной реализации и продвижении проекта, его эффективность и успешность могут значительно превысить затраты на разработку и внедрение.

Заключение

В дипломной работе рассмотрена необходимость разработки планировщика задач для автоматизации процессов планирования и управления задачами в рамках организации ООО «Департамент финансовых услуг». Были рассмотрены основные задачи, которые может решать планировщик, такие как оптимизация использования ресурсов, управление сроками выполнения задач, распределение задач между исполнителями и т.д.

Автоматизация планирования задач не только упрощает и ускоряет работу, но и позволяет снизить вероятность ошибок и повысить точность прогнозирования сроков выполнения задач. Это особенно важно в современных условиях, когда компании сталкиваются с высокой степенью конкуренции и необходимостью быстрого реагирования на изменения в рыночной среде.

Автором было поставлено несколько задач.

В процессе работы над проектом была проведена детальная характеристика области управления проектами, что являлось первой задачей. Задача включала исследование и анализ широкого спектра подходов и стратегий, включая ряд современных методологий управления проектами. Изучение этих методологий позволило более глубоко понять, как они могут быть применены в контексте создания нового приложения.

Далее был проведен тщательный обзор существующих приложений для управления задачами проекта, что помогло установить, какие функции и возможности наиболее ценны для пользователей, и как эти функции могут быть интегрированы в новое приложение, благодаря этому анализу появилась возможность сформулировать функциональные требования к разрабатываемому приложению, обеспечивая тем самым, что приложение будет отвечать реальным потребностям пользователей.

Следующей задачей было проектирование архитектуры приложения и разработка стратегии взаимодействия серверной и клиентской части

приложения. Это включало выбор подходящих технологий и инструментов для разработки, что гарантировало, что приложение будет работать надежно и эффективно.

После этого был реализован пользовательский интерфейс, который был разработан с учетом потребностей и предпочтений пользователей. Этот этап включал не только разработку интерфейса, но и его тестирование, чтобы убедиться, что он интуитивно понятен и удобен для использования.

Затем приложение было развернуто на сервере, что обеспечило его доступность для пользователей. Этот этап также включал тестирование функционала приложения, чтобы убедиться, что все функции работают правильно и что приложение отвечает всем функциональным требованиям.

Весь этот процесс был выполнен с учетом специфики и потребностей ООО «Департамент финансовых услуг». Это гарантировало, что конечный продукт будет адаптирован под потребности конкретной организации, улучшая тем самым ее эффективность и продуктивность. В итоге, было проектировано приложение для планирования задач на платформе Android, которое удобно в использовании и соответствует всем требованиям и ожиданиям пользователей.

При правильной реализации и продвижении проекта, его эффективность и успешность могут значительно превысить затраты на разработку и внедрение.

Список используемых источников

1. Ананьев И.В., Серова Е.Г. Области эффективного применения нотации IDEF0 для задач описания бизнес-процессов // Вестник Санкт-Петербургского университета. Менеджмент. 2008. №2. URL: https://cyberleninka.ru/article/n/oblasti-effektivnogo-primeneniya-notatsii-idef0-dlya-zadach-opisaniya-biznes-protsessov-1 (дата обращения: 03.03.2023).

2. Антал М.А. Особенности планирования в современном производстве / М.А. Антал // В сборнике: Приоритетные направления развития экономики и менеджмента: теоретические и практические аспекты. Сборник научных статей. Уфа, 2021. С. 98-101.

3. Атенсио, Л Функциональное программирование на JavaScript: как улучшить код JavaScript-программ / Л Атенсио. – М.: Диалектика, 2018. с. 304.

4. Блох, Д. Java Эффективное программирование / Д. Блох. - М.: Лори, 2016. 440 с.

5. Блох, Дж. Java: эффективное программирование / Дж. Блох. - М.: Диалектика, 2019. 464 с.

6. Васильев, А. Java. Объектно-ориентированное программирование: Учебное пособие Стандарт третьего поколения / А. Васильев. – СПб.: Питер, 2013. 400 с.

7. Выявление и сбор требований к ПО — ultimate guide | Дзен URL: https://dzen.ru/a/Ys_erPQlgUMeaHvt (дата обращения: 10.02.2022).

8. Гарнаев, А. Web-программирование на Java и JavaScript / А. Гарнаев. - СПб.:BHV, 2005. 1040 с.

9. Давыдов, С. IntelliJ IDEA. Профессиональное программирование на Java / С. Давыдов. – СПб.: BHV, 2005. 800 с.

10. Планировщики дел: 13 лучших программ. URL: https://amssoft.ru/amsblog/planirovshchik-zadach.php (дата обращения: 10.02.2022).

11. Гаст, Х. Объектно-ориентированное проектирование: концепции и программый код / Х. Гаст. - М.: Диалектика, 2018. - 1040 с.

12. Грушвицкий, Р. Проектирование систем на микросхемах с программируемой структурой / Р. Грушвицкий. - СПб.: BHV, 2006. 736 с.

13. Ельцова, О.М. Проектирование основной образовательной программы (на основе программы Н.В.Нищевой) / О.М. Ельцова. - СПб.: Детство-Пресс, 2016. - 256 с.

14. Зебзеева, В.А. Проектирование образовательной программы детского сада в условиях реализации ФГОС ДО / В.А. Зебзеева. - М.: ТЦ Сфера, 2015. - 128 с.

15. Зыль, С. Проектирование, разработка и анализ программного обеспечения систем реального времени / С. Зыль. - СПб.: BHV, 2010. 336 с.

16. Иванов, В.Б. Программирование микроконтроллеров для начинающих Визуальное проектирование, язык С, ассемблер / В.Б. Иванов. - СПб.: Корона-Век, 2015. 176 с.

17. Круз, Р.Л. Структуры данных и проектирование программ / Р.Л. Круз. М.: Бином, 2008. - 765 с.

18. Назаров, С.В. Архитектура и проектирование программных систем: Монография / С.В. Назаров. - М.: НИЦ Инфра-М, 2013. 351 с.

19. Олейникова, О.Н. Модульные технологии: проектирование и разработка образовательных программ: Учебное пособие / О.Н. Олейникова, А.А. Муравьева, Ю.В. Коновалова, Е.В. Сартакова. - М.: Альфа-М, Инфра-М, 2010. 256 с.

20. Ослэндер, Д.М. Управл программы для механических систем. Объектно- ориент. проектирование систем реального времени / Д.М. Ослэндер и др. - М.: Бином. Лаборатория знаний, 2004. 413 с.

21. Панюкова, Т.А. Проектирование программных средств. / Т.А. Панюкова. – М.: КД Либроком, 2012. 362 с.

22. Розин, В.М. Проектирование и программирование: Методологическое исследование. Замысел. Разработка. Реализация. Исторический и социальный контекст / В.М. Розин. - М.: Ленанд, 2018. 160 с.

23. Соловьев, В.В. Логическое проектирование цифровых систем на основе программируемых логических интегральных схем / В.В. Соловьев, А. Климович. - М.: РиС, 2014. 376 с.

24. Эванс, Эрик Предметно-ориентированное проектирование (DDD): структуризация сложных программных систем / Эрик Эванс. - М.: Вильямс И.Д., 2018. 448 с.

25. Allen, D. Getting Things Done: The Art of Stress-Free Productivity. - New York: Penguin Books, 2015. 352 p.

26. Covey, S.R. The 7 Habits of Highly Effective People: Powerful Lessons in Personal Change. – New York: Simon & Schuster, 2013. 432 p.

27. Duhigg, C. The Power of Habit: Why We Do What We Do in Life and Business. – New York: Random House, 2014. 416 p.

28. Gallagher, B. The One Thing: The Surprisingly Simple Truth Behind Extraordinary Results. – Austin: Bard Press, 2013. 240 p.

29. Sinek, S. Start with Why: How Great Leaders Inspire Everyone to Take Action. – New York: Portfolio, 2011. 256 p.

30. Tracy, B. Eat That Frog!: 21 Great Ways to Stop Procrastinating and Get More Done in Less Time. – San Francisco: Berrett-Koehler Publishers, 2017. 144 p.

Приложение А

```
class User(val id: Int, val username: String, val password: String, val role: Int)
class UserDatabase(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
    companion object {
        private const val DATABASE_NAME = "user.db"
        private const val DATABASE_VERSION = 1
        private const val TABLE_NAME = "users"
        private const val COLUMN_ID = "id"
        private const val COLUMN_USERNAME = "username"
        private const val COLUMN_PASSWORD = "password"
        private const val COLUMN_ROLE = "role"
    }
    override fun onCreate(db: SQLiteDatabase) {
        val createTable = "CREATE TABLE $TABLE_NAME ($COLUMN_ID INTEGER PRIMARY KEY, $COLUMN_USERNAME TEXT, $COLUMN_PASSWORD TEXT, $COLUMN_ROLE INTEGER)"
        db.execSQL(createTable)
    }
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
    }
    fun addUser(user: User): Boolean {
        val db = writableDatabase
        val values = ContentValues().apply {
            put(COLUMN_USERNAME, user.username)
            put(COLUMN_PASSWORD, user.password)
            put(COLUMN_ROLE, user.role)
        }
        val newRowId = db.insert(TABLE_NAME, null, values)
        return newRowId != -1L
    }
    fun getUserByUsername(username: String): User? {
```

```
            val db = readableDatabase
            val selection = "$COLUMN_USERNAME = ?"
            val selectionArgs = arrayOf(username)
            val cursor = db.query(TABLE_NAME, null, selection, selectionArgs, null, null, null)
            var user: User? = null
            if (cursor.moveToFirst()) {
                val id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID))
                val password = cursor.getString(cursor.getColumnIndex(COLUMN_PASSWORD))
                val role = cursor.getInt(cursor.getColumnIndex(COLUMN_ROLE))
                user = User(id, username, password, role)
            }
            cursor.close()
            return user
        }
        fun deleteUserByUsername(username: String): Boolean {
            val db = writableDatabase
            val whereClause = "$COLUMN_USERNAME = ?"
            val whereArgs = arrayOf(username)
            val deletedRows = db.delete(TABLE_NAME, whereClause, whereArgs)
            return deletedRows > 0
        }
    }
```

Приложение Б

```
class Task(val id: Int, val title: String, val description: String, val dueDate: Date?, val priority: Priority)
class TaskDatabase(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {
    companion object {
        // Имя базы данных и версия
        private const val DATABASE_NAME = "task.db"
        private const val DATABASE_VERSION = 1

        // Имена таблицы и колонок
        private const val TABLE_NAME = "tasks"
        private const val COLUMN_ID = "id"
        private const val COLUMN_TITLE = "title"
        private const val COLUMN_DESCRIPTION = "description"
        private const val COLUMN_DUE_DATE = "due_date"
        private const val COLUMN_PRIORITY = "priority"
    }

    // Метод onCreate вызывается при создании базы данных
    override fun onCreate(db: SQLiteDatabase) {
        // Создание таблицы задач
        val createTable = "CREATE TABLE $TABLE_NAME ($COLUMN_ID INTEGER PRIMARY KEY, $COLUMN_TITLE TEXT, $COLUMN_DESCRIPTION TEXT, $COLUMN_DUE_DATE INTEGER, $COLUMN_PRIORITY INTEGER)"
        db.execSQL(createTable)
    }

    // Метод onUpgrade вызывается при обновлении базы данных
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        // Удаление таблицы задач и создание ее заново
        db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
        onCreate(db)
```

```
}

// Метод для добавления задачи в базу данных
fun addTask(task: Task): Boolean {
    val db = writableDatabase

    // Создание объекта ContentValues для передачи данных в базу данных
    val values = ContentValues().apply {
        put(COLUMN_TITLE, task.title)
        put(COLUMN_DESCRIPTION, task.description)
        put(COLUMN_DUE_DATE, task.dueDate?.time)
        put(COLUMN_PRIORITY, task.priority.ordinal)
    }

    // Добавление записи в таблицу задач
    val newRowId = db.insert(TABLE_NAME, null, values)

    // Возврат true, если запись добавлена успешно
    return newRowId != -1L
}

// Метод для получения всех задач из базы данных
fun getAllTasks(): List<Task> {
    val db = readableDatabase

    // Запрос на получение всех записей из таблицы задач
    val cursor = db.query(TABLE_NAME, null, null, null, null, null, null)

    val tasks = mutableListOf<Task>()

    // Перебор всех записей в курсоре и добавление их в список задач
    while (cursor.moveToNext()) {
        val id = cursor.getInt(cursor.getColumnIndex(COLUMN_ID))
        val title = cursor.getString(cursor.getColumnIndex(COLUMN_TITLE))
```

```
            val description = cursor.getString(cursor.getColumnIndex(COLUMN_DESCRIPTION))
            val dueDate = cursor.getLong(cursor.getColumnIndex(COLUMN_DUE_DATE))?.let { Date(it) }
            val priority = Priority.values()[cursor.getInt(cursor.getColumnIndex(COLUMN_PRIORITY))]
            tasks.add(Task(id, title, description, dueDate, priority))
        }

        cursor.close()

        // Возврат списка задач
        return tasks
    }
// Метод для удаления задачи из базы данных
    fun deleteTask(task: Task): Boolean {
        val db = writableDatabase

        // Удаление записи с указанным id из таблицы задач
        val whereClause = "$COLUMN_ID = ?"
        val whereArgs = arrayOf(task.id.toString())

        val deletedRows = db.delete(TABLE_NAME, whereClause, whereArgs)
        // Возврат true, если запись удалена успешно
        return deletedRows > 0
    }
    // Метод для обновления задачи в базе данных
    fun updateTask(task: Task): Boolean {
        val db = writableDatabase

        // Создание объекта ContentValues для передачи данных в базу данных
        val values = ContentValues().apply {
            put(COLUMN_TITLE, task.title)
            put(COLUMN_DESCRIPTION, task.description)
```

```
            put(COLUMN_DUE_DATE, task.dueDate?.time)
            put(COLUMN_PRIORITY, task.priority.ordinal)
        }
        // Обновление записи с указанным id в таблице задач
        val whereClause = "$COLUMN_ID = ?"
        val whereArgs = arrayOf(task.id.toString())

        val updatedRows = db.update(TABLE_NAME, values, whereClause, whereArgs)
        // Возврат true, если запись обновлена успешно
        return updatedRows > 0
    }
}
```

Приложение В

```
class TaskManager(private val taskDatabase: TaskDatabase, private val tagDatabase: TagDatabase, private val priorityDatabase: PriorityDatabase) {
    // Метод для добавления задачи в базу данных
    fun addTask(title: String, description: String, dueDate: Date?, priorityId: Long, tagIds: List<Long>): Boolean {
        val priority = priorityDatabase.getPriorityById(priorityId)
        val tags = tagDatabase.getTagsByIds(tagIds)
        val task = Task(0, title, description, dueDate, priority, tags)
        return taskDatabase.addTask(task)
    }
    // Метод для получения всех задач из базы данных
    fun getAllTasks(): List<Task> {
        val tasks = taskDatabase.getAllTasks()
        for (task in tasks) {
            task.priority = priorityDatabase.getPriorityById(task.priority.id)
            task.tags = tagDatabase.getTagsByTaskId(task.id)
        }
        return tasks
    }
    // Метод для удаления задачи из базы данных
    fun deleteTask(task: Task): Boolean {
        return taskDatabase.deleteTask(task)
    }
    // Метод для обновления задачи в базе данных
    fun updateTask(task: Task): Boolean {
        return taskDatabase.updateTask(task)
    }

    // Метод для добавления тега в базу данных
    fun addTag(name: String): Boolean {
        val tag = Tag(0, name)
        return tagDatabase.addTag(tag)
    }
```

```
    // Метод для получения всех тегов из базы данных
    fun getAllTags(): List<Tag> {
        return tagDatabase.getAllTags()
    }
    // Метод для удаления тега из базы данных
    fun deleteTag(tag: Tag): Boolean {
        return tagDatabase.deleteTag(tag)
    }
    // Метод для обновления тега в базе данных
    fun updateTag(tag: Tag): Boolean {
        return tagDatabase.updateTag(tag)
    }
    // Метод для добавления приоритета в базу данных
    fun addPriority(name: String, color: String): Boolean {
        val priority = Priority(0, name, color)
        return priorityDatabase.addPriority(priority)
    }
    // Метод для получения всех приоритетов из базы данных
    fun getAllPriorities(): List<Priority> {
        return priorityDatabase.getAllPriorities()
    }
    // Метод для удаления приоритета из базы данных
    fun deletePriority(priority: Priority): Boolean {
        return priorityDatabase.deletePriority(priority)
    }
    // Метод для обновления приоритета в базе данных
    fun updatePriority(priority: Priority): Boolean {
        return priorityDatabase.updatePriority(priority)
    }
}
```

Приложение Г

```
class NotificationManager(private val context: Context) {

    // Метод для создания уведомления
    fun createNotification(task: Task) {
        val notificationManager = context.getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager

        // Создание канала уведомлений
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val channel = NotificationChannel(
                NOTIFICATION_CHANNEL_ID,
                NOTIFICATION_CHANNEL_NAME,
                NotificationManager.IMPORTANCE_HIGH
            )
            channel.description = NOTIFICATION_CHANNEL_DESCRIPTION
            notificationManager.createNotificationChannel(channel)
        }

        // Создание уведомления
        val builder = NotificationCompat.Builder(context, NOTIFICATION_CHANNEL_ID)
            .setSmallIcon(R.drawable.notification_icon)
            .setContentTitle(task.title)
            .setContentText(task.description)
            .setPriority(NotificationCompat.PRIORITY_HIGH)
            .setAutoCancel(true)
        // Отправка уведомления
        notificationManager.notify(task.id.toInt(), builder.build())
    }
    // Метод для установки уведомления на определенное время
    fun setNotification(task: Task) {
        val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager
```

```
            val intent = Intent(context, NotificationReceiver::class.java)
            intent.putExtra(EXTRA_TASK_ID, task.id.toInt())

            val pendingIntent = PendingIntent.getBroadcast(context, task.id.toInt(), intent, PendingIntent.FLAG_UPDATE_CURRENT)
            val dueDate = task.dueDate?.time ?: return
            alarmManager.set(AlarmManager.RTC_WAKEUP, dueDate, pendingIntent)
        }
        // Метод для отмены уведомления
        fun cancelNotification(task: Task) {
            val alarmManager = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager
            val intent = Intent(context, NotificationReceiver::class.java)
            intent.putExtra(EXTRA_TASK_ID, task.id.toInt())
            val pendingIntent = PendingIntent.getBroadcast(context, task.id.toInt(), intent, PendingIntent.FLAG_UPDATE_CURRENT)
            alarmManager.cancel(pendingIntent)
            pendingIntent.cancel()
        }
    }
```

Приложение Д

```
class StatisticsManager(private val taskDatabase: TaskDatabase) {

    // Метод для получения количества выполненных задач за определенный период
    fun getCompletedTasksCount(startDate: Date, endDate: Date): Int {
        return taskDatabase.getCompletedTasksCount(startDate, endDate)
    }

    // Метод для получения количества невыполненных задач за определенный период
    fun getIncompleteTasksCount(startDate: Date, endDate: Date): Int {
        return taskDatabase.getIncompleteTasksCount(startDate, endDate)
    }
    // Метод для получения количества задач, связанных с определенным тегом
    fun getTasksCountByTag(tag: Tag): Int {
        return taskDatabase.getTasksCountByTag(tag)
    }
    // Метод для получения количества задач, связанных с определенным приоритетом
    fun getTasksCountByPriority(priority: Priority): Int {
        return taskDatabase.getTasksCountByPriority(priority)
    }
    // Метод для получения средней продолжительности выполнения задач за определенный период
    fun getAverageTaskDuration(startDate: Date, endDate: Date): Long {
        return taskDatabase.getAverageTaskDuration(startDate, endDate)
    }
}
```

Приложение Е

```
class SettingsManager(private val sharedPreferences: SharedPreferences) {
    // Метод для получения текущего языка интерфейса
    fun getLanguage(): String? {
        return sharedPreferences.getString(KEY_LANGUAGE, null)
    }
    // Метод для установки языка интерфейса
    fun setLanguage(language: String) {
        sharedPreferences.edit().putString(KEY_LANGUAGE, language).apply()
    }
    // Метод для получения текущей цветовой схемы
    fun getColorScheme(): String? {
        return sharedPreferences.getString(KEY_COLOR_SCHEME, null)
    }
    // Метод для установки цветовой схемы
    fun setColorScheme(colorScheme: String) {
        sharedPreferences.edit().putString(KEY_COLOR_SCHEME, colorScheme).apply()
    }
    // Метод для получения текущего формата времени
    fun getTimeFormat(): String? {
        return sharedPreferences.getString(KEY_TIME_FORMAT, null)
    }
    // Метод для установки формата времени
    fun setTimeFormat(timeFormat: String) {
        sharedPreferences.edit().putString(KEY_TIME_FORMAT, timeFormat).apply()
    }
    // Метод для получения текущего статуса уведомлений
    fun getNotificationsEnabled(): Boolean {
        return sharedPreferences.getBoolean(KEY_NOTIFICATIONS_ENABLED, true)
    }

    // Метод для установки статуса уведомлений
    fun setNotificationsEnabled(enabled: Boolean) {
```

```
            sharedPreferences.edit().putBoolean(KEY_NOTIFICATIONS_ENABLED,
enabled).apply()
        }
        companion object {
            const val KEY_LANGUAGE = "language"
            const val KEY_COLOR_SCHEME = "color_scheme"
            const val KEY_TIME_FORMAT = "time_format"
            const val KEY_NOTIFICATIONS_ENABLED = "notifications_enabled"
        }
    }
```