Аннотация

Максаты: «Газпром Кыргызстан» компаниясынын калкка кызмат көрсөтүү процессин эффективдүү контролдоону жана башкарууну камсыз кылуу максатында тапшырмалардын аткарылышын эсепке алуу системасы иштелип чыккан. Ал тапшырмаларды түзүүгө жана көзөмөлдөөгө, сарпталган убакытты жана материалдарды эсепке алууга жана тандалган мөөнөттөр үчүн аткарылган тапшырмалардын статистикасын жалпылоого мүмкүндүк берет.

Өзгөчөлүк: Система компанияны кызматкерлердин жана подрядчылардын ишине ачык-айкындуулукту жана көзөмөлдү камсыз кылуучу эсепке алуу процессин автоматташтыруу үчүн инструменттерди берет. Бул уруксатсыз жабдууларды орнотуу жана мыйзамсыз киреше менен байланышкан мыйзамсыз иш-аракеттердин алдын алууга жардам берет.

Актуалдуулугу: Үйлөрдү жана батирлерди газдаштырууга муктаждыктын өсүп жаткан шартында процессти эффективдүү башкаруу жана көзөмөлдөө мындай ири компания үчүн маанилүү факторлор болуп саналат. Мыйзамсыз аракеттерге жол бербөө жана аткаруучулардын ишинин ачык-айкындуулугун камсыз кылуу зарылчылыгы барган сайын маанилүү болуп баратат. Тапшырмалардын аткарылышын көзөмөлдөө системасы компаниянын ишинин натыйжалуулугун жогорулатууга жана тапшырмаларды аткаруунун жогорку сапатын камсыз кылууга жардам берет.

Натыйжалуулугу: милдеттерди аткаруу эсепке алуу системасы компания төмөнкү натыйжаларга жетишүүгө мүмкүндүк берет:

* Объекттерди газдаштыруу процессинин айкындуулугун жана көзөмөлүн жогорулатуу.
* Мыйзамсыз иш-аракеттерди жана жабдууларды мыйзамсыз пайдаланууну алдын алуу.
* тапшырманы аткаруу убактысын жана материалды колдонууну оптималдаштыруу.
* Кызматкерлердин жана подряддык уюмдардын натыйжалуулугун жогорулатуу.
* Тапшырмаларды аткаруунун сапатын жана кардарлардын канааттануусун жогорулатуу.

Бул жыйынтыктардын баары Газпромдун натыйжалуу иштешине жана газдаштыруу боюнча кызмат көрсөтүүлөрдүн сапатын жогорулатууга алып келет.

Аннотация

Назначение: Система учета выполнения задач разработана для компании «Газпром Кыргызстан» с целью обеспечения эффективного контроля и управления процессом предоставления населению своих услуг. Она позволяет создавать и отслеживать задачи, учитывать затраченное время и материалы, а также подводить статистику выполненных задач за выбранные периоды.

Характеристика: Система предоставляет компании инструменты для автоматизации процесса учета выполнения задач, обеспечивая прозрачность и контроль над работой сотрудников и подрядчиков. Она помогает предотвратить нелегальные действия, связанные с установкой несанкционированного оборудования и незаконным получением прибыли.

Актуальность: В условиях растущей потребности в газификации домов и квартир, эффективное управление и контроль процессом являются важными факторами для такой крупной компании. Необходимость предотвращения нелегальных действий и обеспечения прозрачности работы исполнителей становится все более актуальной. Система учета выполнения задач поможет компании повысить эффективность своей деятельности и обеспечить высокое качество выполнения задач.

Эффективность: Система учета выполнения задач позволит компании достичь следующих результатов:

* Увеличение прозрачности и контроля над процессом газификации объектов.
* Предотвращение нелегальных действий и незаконного использования оборудования.
* Оптимизация времени выполнения задач и использования материалов.
* Улучшение эффективности работы сотрудников и подрядчиков.
* Повышение качества выполнения задач и удовлетворенности клиентов.

Все эти результаты приведут к более эффективному функционированию компании «Газпром» и повысят ее качество предоставления услуг газификации.

Оглавление

[Список используемых сокращений и обозначений 5](#_Toc137172499)

[Введение 6](#_Toc137172500)

[Основная часть 7](#_Toc137172501)

[Глава 1. Аналитическая часть 7](#_Toc137172502)

[1.1 Информация о предприятии 7](#_Toc137172503)

[1.1.1 Общие сведения 7](#_Toc137172504)

[1.1.2 История ОсОО «Газпром Кыргызстан» 7](#_Toc137172505)

[1.2 Описание предметной области 9](#_Toc137172506)

[1.2.1 Характеристика информационных систем 9](#_Toc137172507)

[1.2.2 Роль информационных систем 9](#_Toc137172508)

[1.2.3 Отрасли и секторы применения 10](#_Toc137172509)

[1.2.4 Особенности информационных систем 10](#_Toc137172510)

[1.2.5 Значимость информационных систем 10](#_Toc137172511)

[1.3 Обзор и анализ существующих аналогичных систем 11](#_Toc137172512)

[1.3.1 Atlassian Jira 11](#_Toc137172513)

[1.3.2 SAP (Systems, Applications, and Products) 13](#_Toc137172514)

[1.3.3 1С 15](#_Toc137172515)

[Глава 2. Проектная часть 19](#_Toc137172516)

[2.1 Обоснование необходимости и цели использования системы учета выполненных задач 19](#_Toc137172517)

[2.1.1 Декомпозиция задачи (IDEF0) 19](#_Toc137172518)

[2.1.2 Диаграмма потоков данных (DFD) 22](#_Toc137172519)

[2.1.3 Диаграмма вариантов использования (USE-CASE) 23](#_Toc137172520)

[2.2 Постановка задачи 25](#_Toc137172521)

[2.2.1 Цель и назначение создания «системы учета выполненных задач». Функциональные требования. 25](#_Toc137172522)

[2.2.2 Анализ требований к информационно-коммуникационным средствам для решения задачи. Нефункциональные требования. 27](#_Toc137172523)

[2.3 Анализ и обоснование выбора технологий 28](#_Toc137172524)

[Обоснование выбора технологий 28](#_Toc137172525)

[2.4 Окружение и среда разработки 30](#_Toc137172526)

[Visual Studio Code (VSCode) 30](#_Toc137172527)

[DataGrip 30](#_Toc137172528)

[Postman 31](#_Toc137172529)

[Git 31](#_Toc137172530)

[Docker 31](#_Toc137172531)

[2.5 Архитектура системы 32](#_Toc137172532)

[2.5.1 Компоненты системы 33](#_Toc137172533)

[2.5.2 Архитектура базы данных. ERD 34](#_Toc137172534)

[2.5.3 Структура системы. Диаграмма классов 36](#_Toc137172535)

[2.5.4 Диаграмма деятельности (Activity) 37](#_Toc137172536)

[2.6 Сторонние модули 38](#_Toc137172537)

[LinOTP 39](#_Toc137172538)

[Сервис отправки электронных писем 40](#_Toc137172539)

[Report Service 40](#_Toc137172540)

[2.7 Интеграционная часть 41](#_Toc137172541)

[Глава 3. Документация 42](#_Toc137172542)

# Список используемых сокращений и обозначений

* БД: База Данных
* (Р)СУБД: (Реляционная) Система управления базами данных
* ИС: Информационная система
* API (АПИ): Application Programming Interface - интерфейс прикладного программирования
* KPI: key performance indicators - ключевые показатели эффективности
* САПР: Система Автоматизированного Проектирования.
* ОС: Операционная Система.
* ИТ: Информационные Технологии.
* ПО: Программное Обеспечение.
* ООП: Объектно-Ориентированное Программирование.
* UI: User Interface - пользовательский интерфейс.
* UX: User Experience - пользовательский опыт.
* SQL: Structured Query Language - язык структурированных запросов.
* HTML: Hypertext Markup Language - язык гипертекстовой разметки.
* CSS: Cascading Style Sheets - каскадные таблицы стилей.
* JSON: JavaScript Object Notation - формат обмена данными, основанный на синтаксисе объектов JavaScript.
* REST: Representational State Transfer - архитектурный стиль для построения распределенных систем.
* URL: Uniform Resource Locator - унифицированный указатель ресурса (веб-адрес).
* IDE: Integrated Development Environment - интегрированная среда разработки.
* HTTP: Hypertext Transfer Protocol - протокол передачи гипертекста. HTTPS: Hypertext Transfer Protocol Secure - защищенная версия протокола HTTP.
* CRUD: Create, Read, Update, Delete - базовые операции над данными (создание, чтение, обновление, удаление).

# Введение

# Основная часть

# Глава 1. Аналитическая часть

## 1.1 Информация о предприятии

### 1.1.1 Общие сведения

ОсОО «Газпром Кыргызстан» – дочернее общество ООО «Газпром». Основным видом деятельности компании является поставка и реализация природного газа. «Газпром Кыргызстан» является монополистом по импорту природного газа в Кыргызскую Республику.

Приоритетные направления деятельности «Газпром Кыргызстан»:

1. транспортировка, распределение и реализация природного газа;
2. реконструкция и расширение действующих газопроводных систем и подземных хранилищ газа (ПХГ) в Кыргызской Республике;

Общая протяженность газопроводов, принадлежащих компании, составляет 3517,17 км, в том числе:

* магистральные - 669 км;
* высокого давления - 10,17 км;
* среднего давления - 718 км;
* низкого давления - 2120 км;

### 1.1.2 История ОсОО «Газпром Кыргызстан»

В 1959 году был образован трест «Фрунзегаз». Началось снабжение природным газом столицы Кыргызстана. На базе «Фрунзегаз» в 1963 году создан первый в Чуйской области газовый участок. В этом же году образовано Жалал-Абадское управление газового хозяйства при Госкомхозе.

В 1970 году началась газификация города Бишкек. В этот же период природный газ стал поступать в Баткенскую область.

В 1994 году приказом Министерства промышленности и торговли создается Чуйско-Таласское управление газового хозяйства в составе Республиканского производственного объединения «Кыргызгаз». В управление на правах газовых хозяйств вошли базы природного и сжиженного газа «Кантгаз» и «Кара-Балтагаз». Указом Президента Кыргызской Республики в 1997 году образована государственная акционерная компания «Кыргызгазмунайзат» на базе Республиканского производственного объединения «Кыргызгаз», Государственного производственного объединения «Чуйгазмунайзат» и Государственной акционерной корпорации «Кыргызмунайзат», установлена система корпоративного управления компанией.

23 сентября 1998 года решением общего собрания акционеров на базе части имущественного комплекса Государственной акционерной корпорации «Кыргызгазмунайзат» образовано открытое акционерное общество «Кыргызгаз».

ОсОО «Газпром» - глобальная энергетическая компания. Основные направления деятельности - геологоразведка, добыча, транспортировка, хранение, переработка и реализация газа, газового конденсата и нефти, реализация газа в качестве моторного топлива, а также производство и сбыт тепло- и электроэнергии.

ОсОО «Газпром» видит свою миссию в надежном, эффективном и сбалансированном обеспечении потребителей природным газом, другими видами энергоресурсов и продуктами их переработки.

Между Правительством Кыргызской Республики и ОАО «Газпром» 9 октября 2008 года подписан меморандум о взаимопонимании по вопросу развития сотрудничества в рамках приватизации ОАО «Кыргызгаз».

Акционерами ОАО «Кыргызгаз» 21 июня 2013 года создано ОсОО «КыргызгазПром», которому отошли все активы, права и обязательства.

26 июля 2013 года в г. Москве состоялось подписание межправительственного соглашения между министром энергетики и промышленности КР О.М. Артыкбаевым и министром энергетики РФ А.В. Новаком.

11 декабря 2013 года Жогорку Кенешом КР данное Соглашение было ратифицировано.

Президентом КР А. Атамбаевым 23 декабря 2013 года подписан Закон КР о ратификации Соглашения между Правительством КР и Правительством РФ о сотрудничестве в сфере транспортировки, распределения и реализации природного газа на территории КР за №223. Соответствующая нота направлена российской стороне.

26 декабря 2013 года Постановлением Правительства РФ №1270 от 26 декабря 2013 года одобрен законопроект о ратификации данного Соглашения.

14 апреля 2014 года в г. Бишкек подписан Договор купли-продажи между ОАО «Кыргызгаз» и ОАО «Газпром», согласно которого ОАО «Кыргызгаз» передал в собственность ОАО «Газпрома» 100% долю уставного капитала дочернего общества с ограниченной ответственностью «КыргызгазПром».

В связи со сменой собственника 11 июля 2014 года произведена государственная перерегистрация ОсОО «КыргызгазПром» и переименовано в ОсОО «Газпром Кыргызстан». С этого момента ОАО «Газпром» вступил в право собственности газотранспортной инфраструктурой Кыргызской Республики.

В структуру холдинга «Газпром Кыргызстан» входят 9 удаленных филиалов.

* Филиал «Бишкекгаз»
* Филиал «Чуйгаз»
* Филиал «Ошгаз»
* Филиал Строительно-монтажное управление «Спецгазмонтаж»
* Филиал «Жалалабадгаз»
* Филиал «Севертрансгаз»
* Филиал «Баткенгаз»
* Филиал «Югтрансгаз»
* Филиал «Метрологии и технического сервиса»

ОсОО «Газпром» занимает первое место по добыче природного газа. По объемам добычи и переработки нефти «Газпром» входит в число пяти крупнейших компаний и в число двадцати мировых лидеров бизнеса.

## 1.2 Описание предметной области

### 1.2.1 Характеристика информационных систем

Информационные системы являются специализированными программными решениями, разработанными для обработки, хранения и передачи информации в конкретной области деятельности. Они играют важную роль в автоматизации бизнес-процессов, управлении данными и обеспечении эффективности работы в соответствии с требованиями и особенностями предметной области.

### 1.2.2 Роль информационных систем

Основная цель информационных систем в предметной области состоит в обеспечении эффективности работы, улучшении качества обслуживания клиентов, оптимизации использования ресурсов, обеспечении безопасности данных и соблюдении требований соответствующих нормативных актов. Они помогают автоматизировать рутинные задачи, оптимизировать рабочие процессы и принимать обоснованные управленческие решения.

### 1.2.3 Отрасли и секторы применения

Информационные системы разрабатываются для различных отраслей и секторов, включая финансы, здравоохранение, производство, торговлю, государственное управление и другие. Каждая отрасль имеет свои специфические требования, законодательные нормы и стандарты, которые учитываются при проектировании и разработке систем.

### 1.2.4 Особенности информационных систем

Одной из ключевых особенностей информационных систем является их способность обрабатывать большие объемы данных и предоставлять оперативную и точную информацию для принятия управленческих решений. Они также обеспечивают интеграцию с другими системами и платформами, совместную работу и обмен данных.

### 1.2.5 Значимость информационных систем

Информационные системы играют важную роль в повышении эффективности и конкурентоспособности организаций. Они помогают автоматизировать бизнес-процессы, улучшить качество обслуживания клиентов, управлять данными, обеспечивать безопасность информации и эффективно использовать ресурсы. Правильное проектирование и разработка информационных систем являются важными задачами для успешного функционирования и развития предприятий и организаций.

В данном разделе было представлено общее описание предметной области информационных систем. Оно включает характеристику информационных систем, их роль и значимость, а также особенности и секторы применения. Это обеспечивает основу для более детального анализа и проектирования системы, которая будет соответствовать требованиям и потребностям предметной области.

## 1.3 Обзор и анализ существующих аналогичных систем

Существует множество систем, позволяющих осуществлять отслеживание и ведение текущих задач, многие из них обладают гибкостью и разнообразными функциональными возможностями, некоторые из них рассмотрены ниже.

### 1.3.1 Atlassian Jira

Это мощная и гибкая система управления задачами и проектами, которая предлагает широкий набор функциональных возможностей. Вот основные характеристики и функциональности Atlassian Jira:

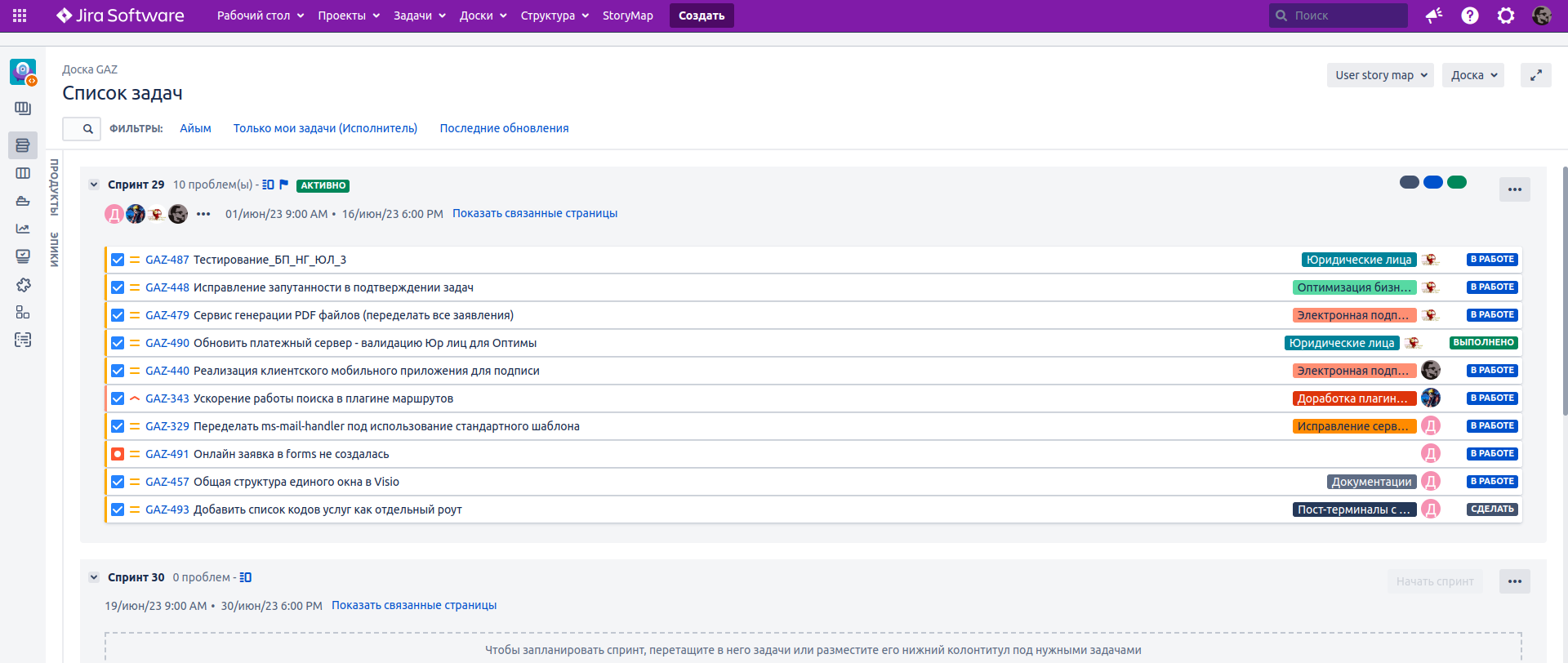


Рисунок x Скриншот списка задач (доска спринтов)

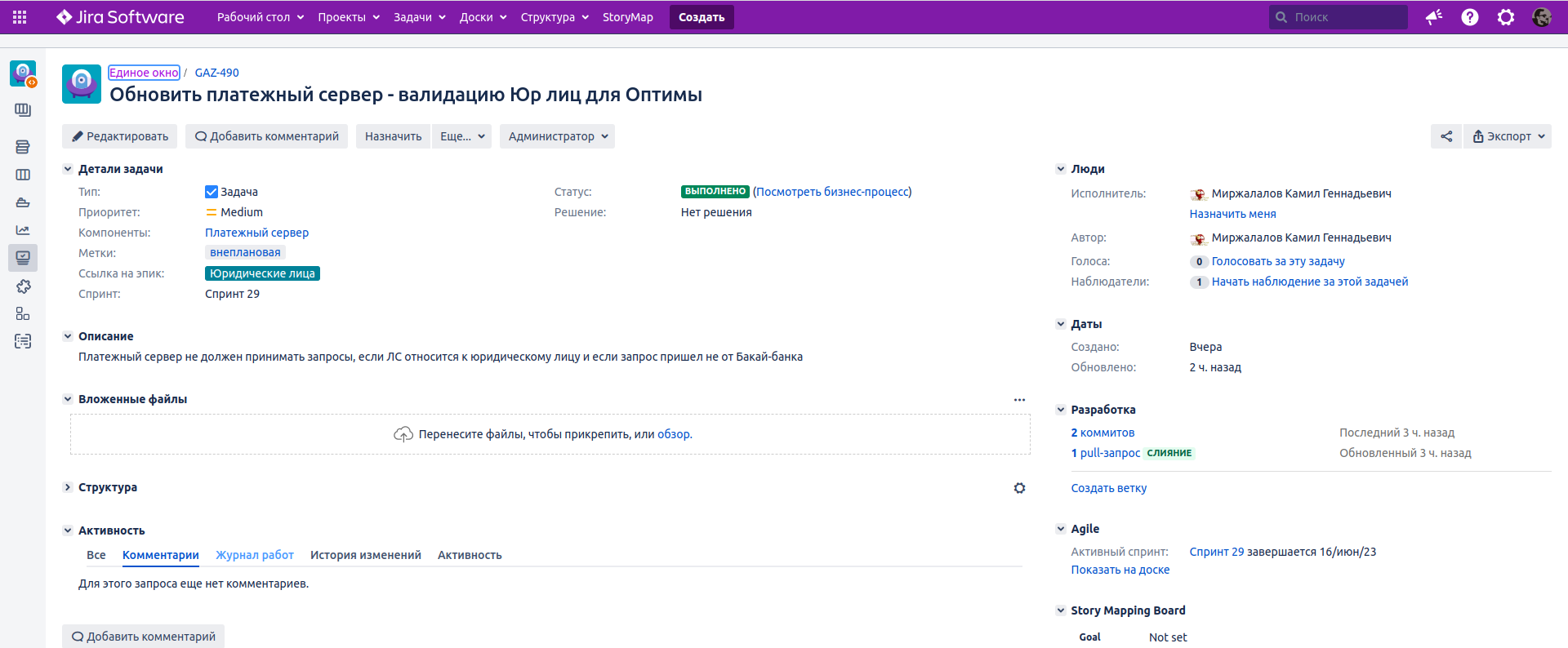


Рисунок 1.x Скриншот просмотра деталей одной задачи

1. Управление задачами и проектами: Jira предоставляет возможность создавать и отслеживать задачи, присваивать им исполнителей, устанавливать сроки выполнения и отслеживать прогресс. Можно организовывать задачи в виде проектов и структурировать их по различным категориям.
2. Отслеживание ошибок и запросов: Jira позволяет легко отслеживать и управлять ошибками, дефектами и запросами на улучшение. Можно создавать тикеты для проблем, назначать их разработчикам, отслеживать статус исправления и вести дискуссии внутри системы.
3. Гибкая система рабочих процессов: Jira позволяет настраивать и адаптировать рабочие процессы под требования вашей организации. Можно создавать собственные шаблоны задач, определять этапы выполнения и устанавливать правила перехода между ними.
4. Визуализация данных: Jira предоставляет графические диаграммы и дашборды, которые помогают визуализировать и анализировать данные о задачах и проектах. Можно создавать диаграммы Ганта, диаграммы burndown (оставшиеся задачи), сводные таблицы и другие типы отчетов для мониторинга прогресса и производительности.
5. Совместная работа и коммуникация: Jira предоставляет возможность совместной работы и коммуникации внутри команды. Можно комментировать задачи, упоминать коллег и обсуждать проблемы прямо в контексте задачи. Также, Jira интегрируется с другими инструментами для коммуникации, такими как Slack или Microsoft Teams.
6. Расширяемость и интеграция: Jira обладает широким набором плагинов и расширений, которые позволяют настроить систему под свои потребности. Она также интегрируется с другими популярными инструментами разработки, такими как Git, Confluence, Bitbucket и др.

Atlassian Jira является мощным инструментом для управления задачами и проектами, обладающим гибкостью и разнообразными функциональными возможностями. Он позволяет командам эффективно управлять рабочим процессом, отслеживать прогресс и улучшать коммуникацию внутри команды.

Несмотря на множество преимуществ, система Atlassian Jira также имеет некоторые недостатки, которые следует учитывать:

1. Сложность использования: Начальная настройка и освоение Jira могут быть сложными для новых пользователей. Интерфейс и функциональность системы могут показаться сложными для неподготовленных пользователей, особенно если они не имеют опыта работы с подобными инструментами.
2. Высокая стоимость: Jira является коммерческим продуктом, и его лицензирование может быть затратным, особенно для малых компаний или небольших проектов с ограниченным бюджетом. Дополнительные плагины или расширения также могут потребовать дополнительных затрат.
3. Необходимость настройки: Хотя Jira предлагает широкий набор функциональных возможностей, некоторые из них могут потребовать дополнительной настройки или использования плагинов. Это может повлечь за собой дополнительные затраты времени и ресурсов для настройки и интеграции системы.
4. Ограничения масштабируемости: В некоторых случаях, при работе с большим объемом данных или сложными рабочими процессами, Jira может столкнуться с ограничениями масштабируемости и производительности. Это может привести к задержкам или снижению производительности системы.
5. Необходимость обучения пользователей: Внедрение Jira в организации может потребовать дополнительного обучения пользователей. Это может быть вызовом, особенно если в организации есть сотрудники с ограниченными навыками работы с компьютером или недостаточным пониманием процессов управления задачами.

Итого, Atlassian Jira громоздкая система, с множеством возможностей, требует длительной подготовки перед ее настройкой и использованием, имеет проблемы с масштабированием и репликацией, а стоимость лицензий системы и ее плагинов в некоторых случаях, делают ее применение нецелесообразным с экономической точки зрения.

### 1.3.2 SAP (Systems, Applications, and Products)

SAP (Systems, Applications, and Products) — это крупная международная компания, предлагающая широкий спектр программных продуктов для управления бизнес-процессами в различных отраслях. Одним из ключевых продуктов SAP является платформа SAP ERP (Enterprise Resource Planning), которая предоставляет мощные функциональные возможности для ведения задач и управления бизнес-процессами. Вот некоторые из основных характеристик и функциональных возможностей SAP в плане ведения задач:

1. Интеграция процессов: SAP позволяет интегрировать различные бизнес-процессы, связанные с ведением задач, в единую систему. Это позволяет автоматизировать и упростить выполнение задач и обеспечивает единый источник данных для анализа и принятия решений.
2. Управление проектами: SAP ERP предоставляет функциональность для планирования, отслеживания и управления проектами. Это включает в себя возможности для определения задач, назначения ответственных лиц, установки сроков выполнения, контроля прогресса и оценки ресурсов.
3. Бюджетирование и учет затрат: SAP позволяет вести учет бюджетов проектов и задач, отслеживать затраты на материалы, трудовые ресурсы и другие расходы. Это помогает контролировать затраты, оптимизировать бюджеты и анализировать эффективность задач.
4. Коллаборация и коммуникация: SAP предоставляет инструменты для коллаборации и коммуникации внутри организации. Это включает в себя возможность обмена сообщениями, совместной работы над документами, управления комментариями и задачами, а также интеграцию с электронной почтой и другими коммуникационными средствами.
5. Аналитика и отчетность: SAP предлагает широкий набор инструментов для анализа данных и создания отчетов. Это позволяет анализировать производительность задач, оценивать качество выполнения работ, отслеживать ключевые показатели производительности и принимать информированные решения.
6. Персонализация и гибкость: SAP ERP позволяет настраивать и адаптировать систему под конкретные потребности организации. Это включает возможность настройки рабочего пространства, пользовательских ролей и доступа к функциональности, чтобы обеспечить оптимальное использование системы.

SAP предоставляет мощные инструменты для ведения задач и управления бизнес-процессами. Однако, конкретные характеристики и функциональные возможности могут варьироваться в зависимости от конкретной версии и конфигурации продукта SAP, а также от требований и настроек организации.

Недостатки SAP:

1. Сложность внедрения: Внедрение SAP может быть сложным и требовательным процессом. Оно может потребовать значительных усилий, времени и ресурсов, включая специалистов по SAP и консультантов. Это может повлечь высокие затраты на внедрение и обслуживание системы.
2. Высокая стоимость: SAP является мощной и расширяемой системой, однако она может быть дорогой для небольших и средних предприятий. Лицензирование и поддержка SAP могут иметь значительные затраты, что делает его недоступным для некоторых компаний с ограниченными бюджетами.
3. Сложность использования: Из-за масштабного функционала и гибкости SAP может быть сложным в использовании для новых пользователей. Необходимо обучение и опыт для эффективного использования системы. Это может потребовать времени и усилий для обучения сотрудников, и адаптации к новой среде работы.
4. Высокая зависимость от инфраструктуры: Для работы SAP требуется надежная и мощная IT-инфраструктура, включая серверы, базы данных и сетевые ресурсы. Это может потребовать дополнительных инвестиций в оборудование и поддержку IT-инфраструктуры.
5. Ограничения по настройке и гибкости: Некоторые пользователи могут столкнуться с ограничениями в настройке и гибкости SAP. Внесение изменений в систему или настройка специфических требований может быть сложным или требовать дополнительных ресурсов. Это может быть проблемой для организаций с уникальными бизнес-процессами или специфическими требованиями.

Итого, SAP еще более сложная в освоении, чем Atlassian Jira, более требовательная к инфраструктуре, также платная, имеет меньшую распространенность, за счет чего найти квалифицированного специалиста для ее поддержки (за адекватную цену) почти невозможно.

### 1.3.3 1С

Это платформа и набор программных продуктов, предназначенных для автоматизации бизнес-процессов и управления предприятием.

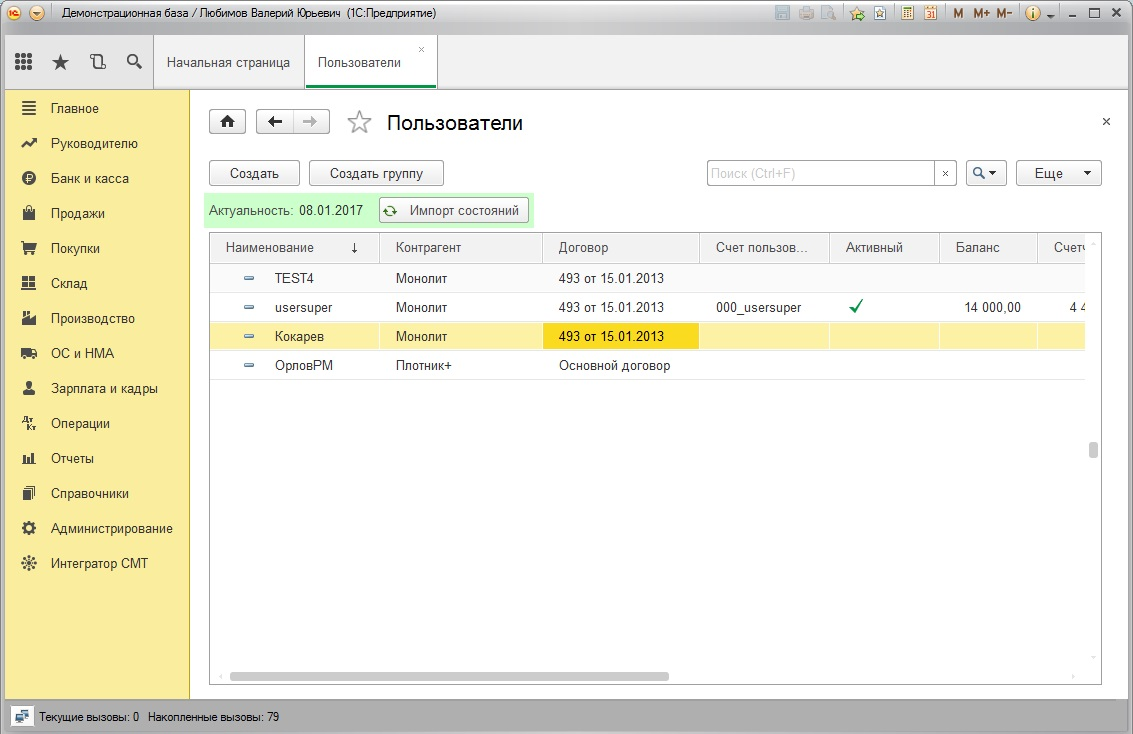


Рисунок 1.x Справочник пользователей

В контексте ведения задач, 1С предлагает следующие основные характеристики и функциональные возможности:

1. Управление задачами: 1С позволяет создавать и отслеживать задачи, определять их приоритеты, статусы, сроки выполнения и ответственных исполнителей. Можно назначать задачи конкретным сотрудникам и отслеживать их выполнение.
2. Календарное планирование: 1С предоставляет возможность создания календарей и планирования задач на определенные даты и сроки. Можно определить зависимости между задачами, устанавливать предшествующие и последующие задачи, а также распределить ресурсы и определить сроки выполнения.
3. Управление ресурсами: 1С позволяет управлять ресурсами, связанными с выполнением задач. Можно назначать сотрудников, материальные и финансовые ресурсы для каждой задачи. Это позволяет контролировать расходы и использование ресурсов в процессе выполнения задач.
4. Коммуникация: 1С предоставляет инструменты для коммуникации и совместной работы над задачами. Можно обмениваться сообщениями, комментариями, прикреплять файлы и делиться информацией с другими участниками проекта. Это способствует эффективной командной работе и обеспечивает прозрачность процесса выполнения задач.
5. Отчетность и аналитика: 1С предоставляет возможность получать отчеты и аналитическую информацию о выполнении задач. Можно анализировать статусы, сроки, затраты и другие параметры задач для принятия решений и оптимизации процесса выполнения.
6. Интеграция с другими системами: 1С обладает возможностью интеграции с другими системами и программными продуктами. Можно интегрировать 1С с почтовыми сервисами, электронными документами, системами управления проектами и другими решениями для обеспечения более полного и связанного рабочего процесса.

Недостатки 1С:

1. Сложность настройки и разработки: При работе с 1С может потребоваться определенный уровень знаний и опыта для настройки и разработки функциональности. Создание и настройка сложных бизнес-процессов может требовать углубленного понимания конфигураций и языка программирования 1С.
2. Высокая стоимость: Лицензирование и внедрение 1С могут быть затратными. Стоимость лицензий, обновлений и поддержки может быть значительной, особенно для больших предприятий или специализированных конфигураций.
3. Ограниченные возможности интеграции: Встроенные инструменты интеграции в 1С могут быть ограничены. Это может затруднять интеграцию с другими системами, особенно с технологически сложными или специализированными решениями.
4. Зависимость от поставщика: 1С является проприетарной системой, что означает зависимость от компании-разработчика. Изменение поставщика или переход на другую платформу может быть сложным и затратным процессом.
5. Ограниченная масштабируемость: Некоторые версии 1С могут иметь ограничения по масштабируемости, особенно при работе с большим объемом данных или высокой нагрузкой. Это может потребовать дополнительных усилий для оптимизации производительности и масштабирования системы.
6. Сложности при обновлении: При переходе на новые версии 1С могут возникать сложности и проблемы совместимости с предыдущими настройками и разработками. Обновление системы может потребовать тщательного планирования и тестирования.
7. Ограниченная гибкость: Некоторые пользователи могут считать, что 1С имеет ограниченные возможности для настройки под свои индивидуальные потребности и процессы. Это может быть особенно заметно в случае использования стандартных конфигураций, которые не всегда могут удовлетворять специфическим требованиям организации.

1С имеет множество возможностей, которые могут различаться в зависимости от конкретной конфигурации и модулей. Но, самый большой недостаток при выборе 1С - это негативный опыт ее использования компанией «Газпром Кыргызстан».

Есть еще множество систем, способных в той или иной степени осуществлять - учет и ведение задач компании «Газпром Кыргызстан», такие как: Redmine, GitLab, Bugzilla, Trac, которые не будут рассмотрены в данном документе.

# Глава 2. Проектная часть

## 2.1 Обоснование необходимости и цели использования системы учета выполненных задач

### 2.1.1 Декомпозиция задачи (IDEF0)

Декомпозиция решения задачи позволяет разбить сложную задачу на более простые и управляемые компоненты, что облегчает понимание и реализацию системы.  
В рамках проведения декомпозиции, мы определяем основные функции и подзадачи, которые необходимо выполнить для достижения целей системы. Затем мы структурируем эти функции в иерархическую структуру, где каждый компонент системы отвечает за определенную функциональность.

Для визуализации этой структуры и взаимосвязей между компонентами, мы используем диаграмму IDEF0. На данной диаграмме мы будем представлять функциональные блоки и связи между ними, а также входы и выходы каждого блока.  
Декомпозиция и создание диаграммы IDEF0 позволят нам лучше понять структуру системы, выделить ключевые компоненты и определить их взаимосвязи. Это будет полезной основой для дальнейшего проектирования и разработки информационной системы, а также для более эффективного управления задачами и ресурсами.

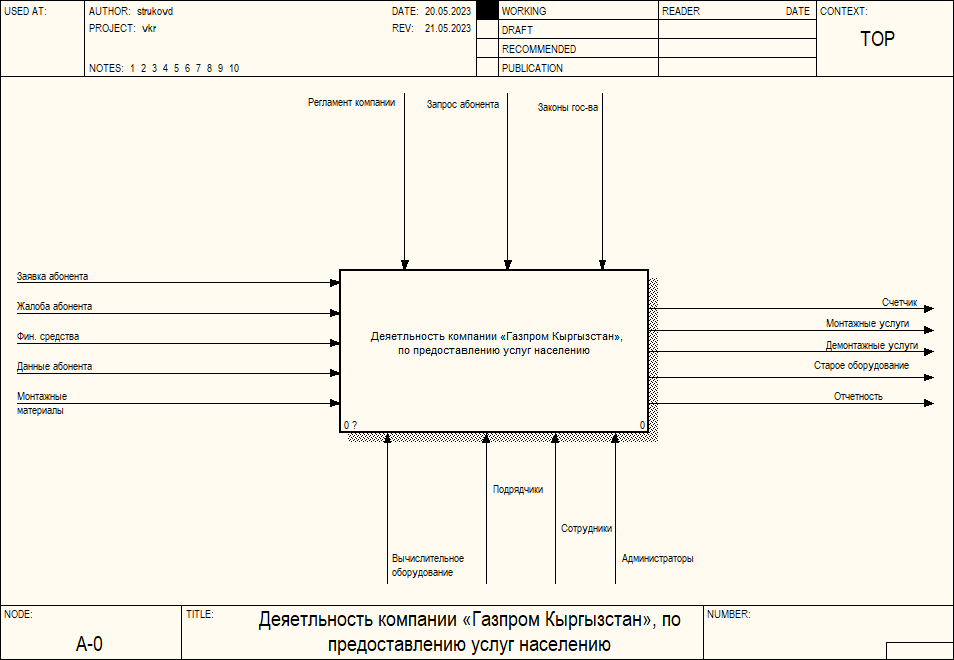


Рисунок x Общая схема работы предприятия, по части обслуживания населения

|  |  |  |
| --- | --- | --- |
| поток | описание | тип потока (вх) |
|  |  |  |
|  |  |  |

На диаграмме видны основные входящие параметры, входящие из вне системы, снизу видны механизмы, позволяющие управлять процессом (администрация, оборудование, подрядчики, и прочие сотрудники), необходимые для работы, сверху видны правила, в соответствии с которыми выполняется работа (регламент, закон, и запрос абонента), и виден результат работы (новые абонентские счетчики, демонтажные услуги, монтажные услуги, отработанное оборудование после демонтажа, отчетность).

Декомпозируем процесс, для рассмотрения деталей:

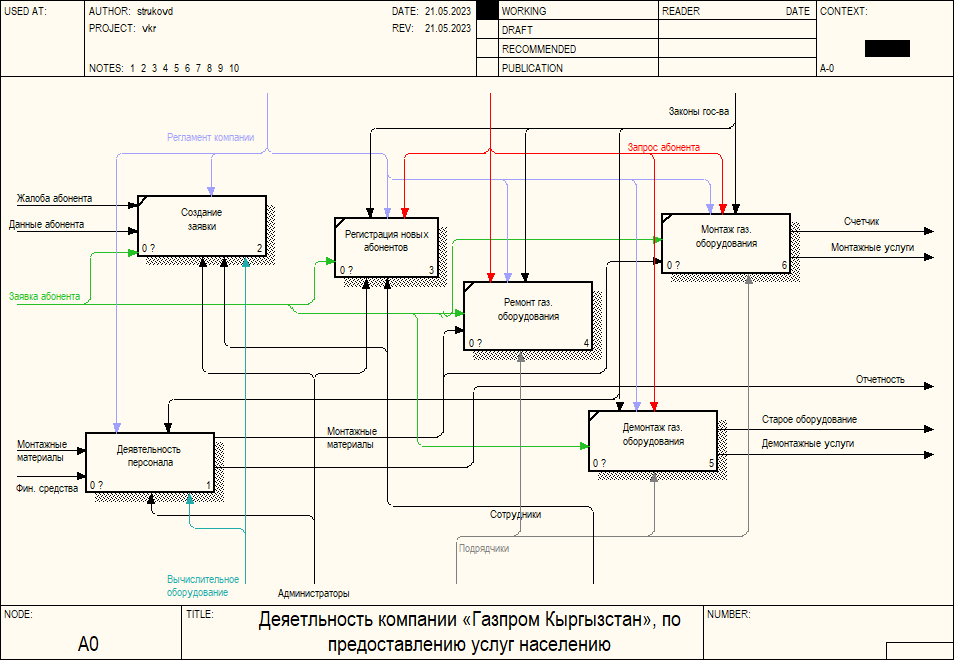


Рисунок x Декомпозиция схемы работы предприятия, по части обслуживания населения.

На схеме видны основные функции, выполняющиеся в процессе работы, в целом это работает так: - создается заявка, по которой сотрудники и подрядчики, учитывая заявку абонента выполняют некие работы. Отдельно выделена работа администрации, ее декомпозиция рассмотрена ниже:

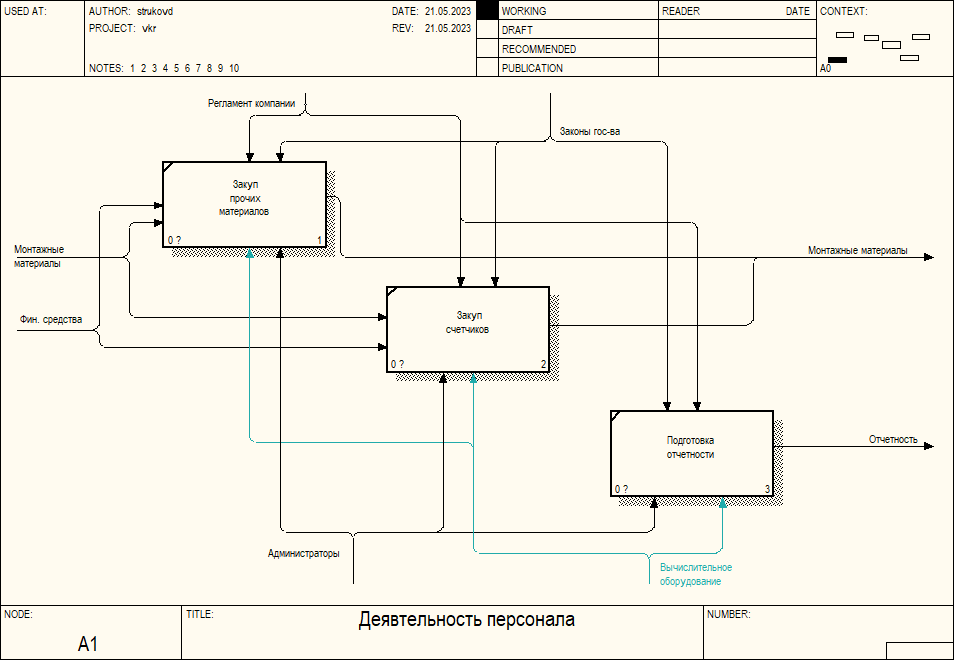


Рисунок х Декомпозиция схемы работы предприятия, по части обслуживания населения.

В целом, на схеме отображено то, что администрация снабжает сотрудников необходимыми материалами, и следит за статистикой выполнения работ.

По приведенным диаграммам можно выявить основные функциональные требования (т.е. то, какие функции от системы требуются), а именно:

* Учет заявок
* Ведение учета абонентов (или связи с системой абонентов)
* Учет используемых материалов и оборудования
* Отчетность о проделанной работе (за определенный период)

Для более рассмотрения более приближенной к требованиям информационной системы и анализа потоков данных, приведем соответствующую диаграмму в следующем подразделе.

### 2.1.2 Диаграмма потоков данных (DFD)

В то время как IDEF0 уделяет больше внимания функциональности и взаимодействию компонентов системы, DFD позволяет сфокусироваться на потоках данных и обработке информации в информационной системе, она помогает понять, какие данные используются в системе, как они передаются между компонентами и как происходит их обработка. Важной особенностью DFD является то, что она не учитывает внутреннюю логику или подробности реализации каждого компонента, а сконцентрирована на потоках данных между ними. DFD позволяет выделить основные процессы, внешние сущности и данные, а также их взаимодействие.

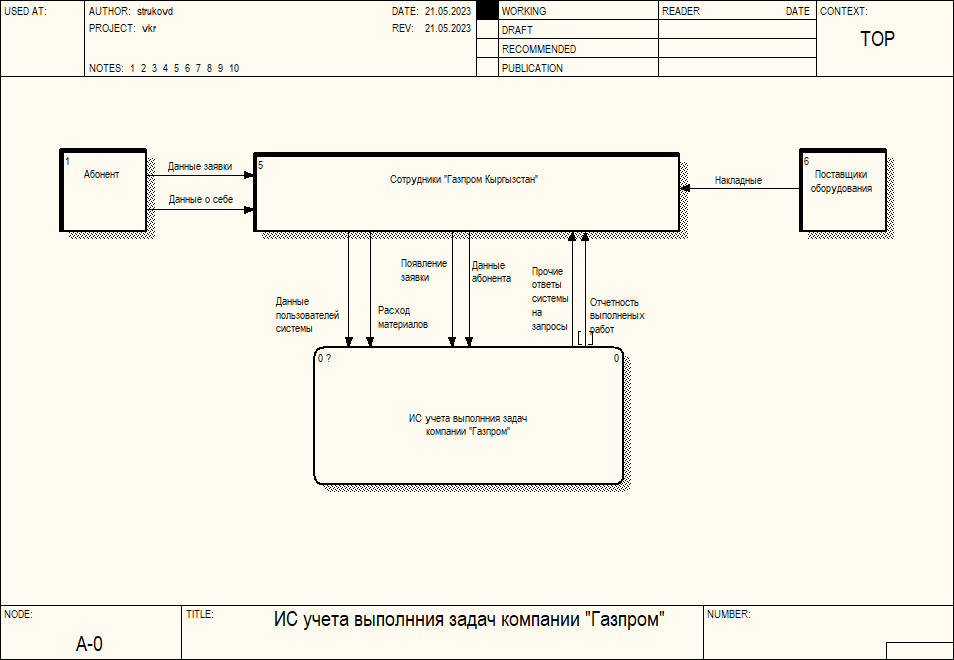


Рисунок х Внешние потоки данных

|  |  |
| --- | --- |
| описать сущности и процессы |  |
|  |  |
|  |  |

На рисунке видно примеры входящих потоков данных входящий в систему и сущности, порождающие эти потоки. Ниже представлена декомпозиция, отображающая функции, которые используют входящие потоки данных.

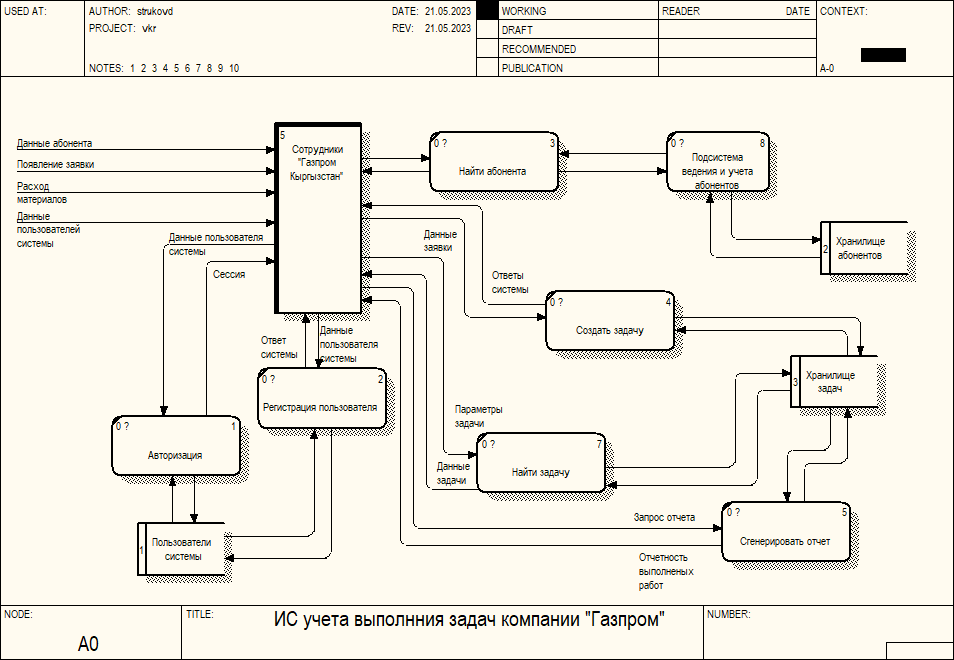


Рисунок х Декомпозиция диаграммы потоков данных

По приведенным диаграммам можно выявить основные данные которые необходимо учитывать в системе, а именно:

* Данные абонентов (через подсистему)
* Данные пользователей
* Данные задач (заявок)

В итоге, мы можем сформулировать функциональные требования в виде специальной диаграммы вариантов использования.

### 2.1.3 Диаграмма вариантов использования (USE-CASE)

Диаграмма USE-CASE является эффективным инструментом для идентификации и моделирования требований к информационной системе. Она позволяет визуально представить различные сценарии использования системы и взаимодействие ее актеров с функциональными возможностями системы.

В представленной ниже диаграмме мы сосредоточимся на основных актерах системы и их ролях во взаимодействии с системой. Актеры могут быть пользователями, внешними системами, а также другими компонентами, которые взаимодействуют с нашей информационной системой.

Каждый вариант использования (USE-CASE) представляет собой конкретную функциональную возможность системы и сценарий, в котором актеры взаимодействуют с системой для достижения определенных целей. Мы опишем основные актеры и варианты использования, а также их связи и зависимости друг от друга.

Ранее, при рассмотрении диаграммы IDEF0, был представлен список основных функциональных требований (список функций, которые требуются от системы), диаграмма вариантов использования позволяет графически (более наглядно) представить наш список функциональных требований к системе и что позволит проанализировать и выявить основные сценарии использования системы и в дальнейшем спроектировать информационную систему, основываясь на ее данных

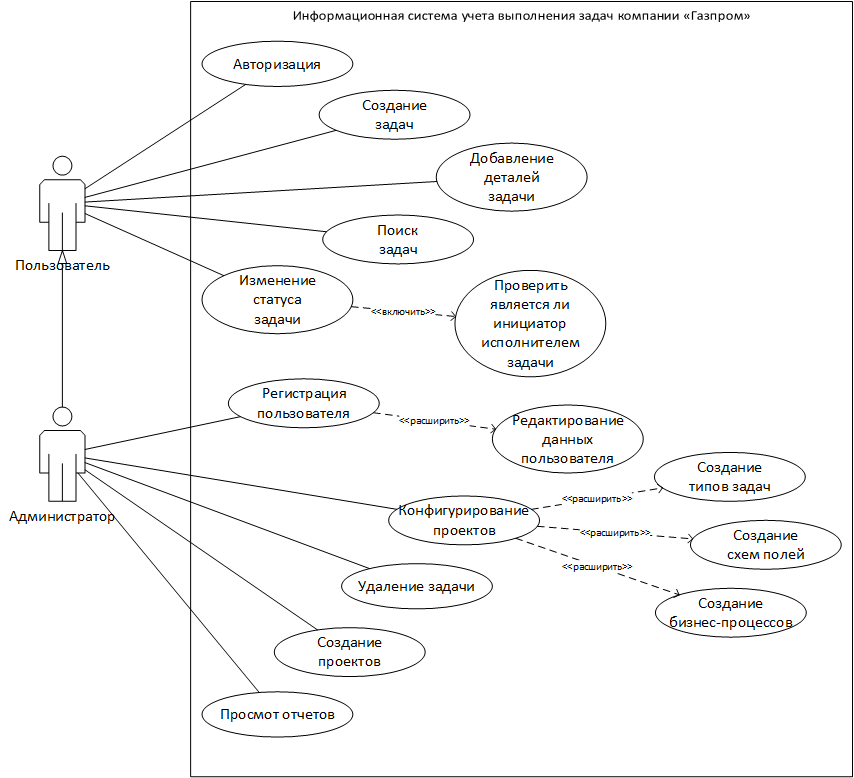


Рисунок х Диаграмма вариантов использования (USE-CASE)

|  |  |
| --- | --- |
| описание всех вариантов использованя |  |
|  |  |
|  |  |
|  |  |

На диаграмме видны основные функции, необходимые для работы системы, а также видны роли пользователей системы.

## 2.2 Постановка задачи

### 2.2.1 Цель и назначение создания «системы учета выполненных задач». Функциональные требования.

Целью разработки является создание системы учета задач (упрощенный аналог Atlassian Jira), которая будет обеспечивать эффективное управление процессом выполнения задач, контроль времени выполнения, использованных материалов и генерацию статистики по срокам выполнения задач за выбранные периоды. Главной целью является повышение эффективности работы сотрудников и подрядчиков, предотвращение нелегальных практик и обеспечение надежной отчетности перед компанией «Газпром Кыргызстан».

#### Основные задачи разработки:

1. Разработать систему, позволяющую создавать и управлять задачами, включая возможность назначения ответственных лиц, приоритетов, дедлайнов и статусов задач.
2. Реализовать функционал учета времени выполнения задач, позволяющий отслеживать затраченное время на каждую задачу и составлять соответствующую статистику.
3. Обеспечить возможность учета использованных материалов при выполнении задач, таких как оборудование, инструменты и другие ресурсы.
4. Разработать механизм генерации статистических отчетов о сроках выполнения задач за выбранные периоды, позволяющий анализировать эффективность работы исполнителей и выявлять проблемные области.
5. Обеспечить безопасность и защиту данных системы, включая механизмы аутентификации, авторизации и защиты от несанкционированного доступа.
6. Реализовать удобный пользовательский интерфейс, обеспечивающий удобство использования системы и интуитивно понятную навигацию.

#### Функциональные требования:

Ранее мы уже выявляли функциональные требования, общие требования были выявлены после формирования диаграммы IDEF0, а далее более подробно представили их в виде диаграммы USE-CASE.

После выявления цели (назначения) системы, перечислим список необходимых функций:

* Создание задач: система должна позволять пользователям создавать новые задачи, указывая информацию о них, такую как заголовок, описание, сроки выполнения, ответственные лица и приоритет.
* Управление задачами: система должна предоставлять возможность управлять задачами, включая назначение ответственных, изменение статусов, установку приоритетов, отслеживание выполнения и уведомления о сроках.
* Комментирование задач: пользователи должны иметь возможность комментировать задачи, обмениваться информацией и обсуждать детали их выполнения.
* Отчетность: система должна предоставлять возможность генерировать отчеты о выполнении задач, статусе проектов, использовании ресурсов и других важных метриках.
* Уведомления и оповещения: система должна отправлять уведомления и оповещения пользователям о новых задачах, изменениях статусов, сроках выполнения и других важных событиях.
* Фильтрация и поиск: пользователи должны иметь возможность фильтровать и искать задачи по различным критериям, таким как статус, приоритет, ответственный и другие параметры.
* Интеграция с другими системами: система должна обеспечивать возможность интеграции с другими информационными системами, такими как системы учета времени, системы управления проектами и другими, для обмена данными и совместной работы.

#### Учет недостатков аналогичных систем:

Данная система позволит избежать недостатки приведенных аналогов в предыдущем разделе, а именно:

* Расширяемость: Собственная система разработки задач может быть разработана с учетом **будущих потребностей** и возможности расширения функциональности в дальнейшем. Можно добавлять новые модули, интегрировать с другими системами и адаптироваться к изменяющимся требованиям без зависимости от вендора или ограничений стороннего ПО.
* Узкоспециализированность: Разработка собственной системы позволяет более **точно соответствовать бизнес-требованиям**, анализировать и внедрять бизнес-процессы компании. Можно создать систему, полностью соответствующую уникальным требованиям, что повышает эффективность и улучшает рабочие процессы, и не требует **длительной подготовки персонала** перед ее использованием.
* Независимость от внешних поставщиков: Собственная разработка освобождает от зависимости от вендоров и сторонних поставщиков ПО. Мы имеем контроль процесса разработки, обслуживания и поддержки системы, а также имеете полный доступ к исходному коду и **интеллектуальной собственности**.
* Более низкие затраты в долгосрочной перспективе: Хотя начальные затраты на разработку собственной системы могут быть выше, в долгосрочной перспективе это может привести к снижению общих затрат. Собственная система не требует огромных вложений на **оплату лицензий** и не зависит от регулярных обновлений или дополнительных модулей, что может сэкономить средства на протяжении всего срока эксплуатации.

### 2.2.2 Анализ требований к информационно-коммуникационным средствам для решения задачи. Нефункциональные требования.

Важно учесть не только функциональные требования, которые определяют функционал и возможности системы, но и нефункциональные требования, которые определяют качество и характеристики системы. Нефункциональные требования играют ключевую роль в обеспечении надежности, безопасности, производительности и удобства использования системы. Ниже перечислены основные нефункциональные требования, которые следует учесть при проектировании и разработке нашей информационной системы учета задач. Эти требованиями позволят нам определить приоритеты и сделать обоснованный выбор технологий и инструментов разработки.

1. Производительность: система должна обеспечивать высокую производительность и отзывчивость даже при большой нагрузке и объеме данных.
2. Надежность: система должна быть стабильной и надежной, предотвращать потерю данных и обеспечивать непрерывную работу.
3. Безопасность: система должна обеспечивать защиту данных и предотвращать несанкционированный доступ.
4. Масштабируемость: система должна быть способной масштабироваться и адаптироваться к росту объема данных и числа пользователей.
5. Удобство использования: система должна быть интуитивно понятной и легкой в использовании для пользователей разного уровня навыков.
6. Совместимость: система должна быть совместима с другими существующими системами и стандартами.

Поддержка и обслуживание: система должна быть легко поддерживаемой и обслуживаемой, с возможностью резервного копирования данных и мониторинга производительности.

## 2.3 Анализ и обоснование выбора технологий

Проведя анализ (нефункциональных) требований и степень моих навыков, было принято решение выбрать следующий стек технологий для написания модулей (сервисов):

* Node.js - основная язык программирования.
* Nest.js - удобный фреймворк для бэкенда.
* Vue.js - простой фреймворк для фронтенда.
* JSON + REST API - распространенный формат передачи данных.
* Elasticsearch - Оптимизированная СУБД для поиска больших объемов данных.
* Redis - быстрая СУБД, для хранения сессий.
* PostgreSQL - бесплатная и стабильная СУБД, для хранения данных задач и другой информации.
* Docker - контейниризация, для упрощения развертывания и безопасности.
* Linux - некоммерческая операционная система (CentOS).
* Микросервисная архитектура - для упрощения поддержки и работоспособности системы.

### Обоснование выбора технологий

#### Node.js

Является высокопроизводительной платформой для разработки серверной части. Он основан на событийно-ориентированной и неблокирующей архитектуре, что позволяет эффективно обрабатывать большое количество запросов и обеспечивать отзывчивость системы, что соответствует критерию нефункционально требования - **производительность**.

#### Микросервисная архитектура

Способствует высокой надежности и отказоустойчивости системы. Каждый модуль (сервис) может функционировать независимо, что минимизирует влияние сбоев и позволяет легко масштабировать и обслуживать отдельные компоненты. Микросервисная архитектура в сочетании с использованием Docker и Elasticsearch позволяет гибко **масштабировать** отдельные компоненты системы в зависимости от нагрузки. Мы можем горизонтально масштабировать сервисы, увеличивая количество экземпляров в контейнерах, а Elasticsearch позволяет обрабатывать большие объемы данных и запросов. Так же, в дальнейшем совместно с docker можно использовать оркестратор Kubernetes, для управления контейнеризированными сервисами, что также упростит масштабирование системы.

#### Nestjs

Позволяет удобно делить код на отдельные модули, включает множество архитектурных решений, и структурированный жизненный цикл запросов, включающий в себя шаблонные (паттерны) подходы такие как: guard, pipes, interceptors, декораторы, DI-систему, и так далее. Это позволяет удобно структурировать код, и минимизировать вероятность ошибок.

#### Docker

Обеспечивает изоляцию и контроль доступа к каждому контейнеру. Это повышает **безопасность** системы, так как в случае несанкционированного получения доступа к системе (взлома) сервисного контейнера, злоумышленник будет ограничен в своих возможностях и не сможет получить полный доступ к данным и ресурсам системы.

#### Vue.js

Предоставляет удобный и гибкий фреймворк для разработки интерфейса пользователя. Он предоставляет компонентный компонентный подход, что может позволить реализовать **удобный интерфейс**. Также, популярность Vue.js, простота в изучении и использовании позволит без особого труда найти специалиста для **поддержки** или доработки существующей системы в бедующем.

#### REST API

REST API является стандартным протоколом взаимодействия между клиентской и серверной частями системы. Это обеспечивает **совместимость с другими системами** и возможность интеграции с внешними сервисами.

Весь выбранный стек технологий имеет широкую поддержку сообщества разработчиков и предоставляет богатый выбор инструментов и библиотек для упрощения разработки, отладки и **обслуживания системы**.

## 2.4 Окружение и среда разработки

Для разработки системы я использую следующие инструменты

### Visual Studio Code (VSCode)

Расширяемость: VSCode обладает широким спектром расширений, что позволяет настроить его под нужды проекта. Я могу использовать расширения для поддержки различных языков программирования, системы контроля версий, отладки и других функций, упрощающих разработку.

Интеграция с Git: VSCode имеет встроенную поддержку Git, что облегчает работу с системой контроля версий и обеспечивает удобный механизм отслеживания изменений в коде.

Мощные функциональные возможности: VSCode обладает рядом полезных функций, таких как автодополнение, быстрые исправления, интеграция с отладчиками и т.д., которые повышают эффективность разработки.

### DataGrip

DataGrip - это инструмент управления базами данных, который обеспечивает широкий спектр функций для работы с различными СУБД. Я выбрал DataGrip по следующим причинам:

Поддержка разных СУБД: DataGrip обладает возможностью работать с различными СУБД, такими как PostgreSQL, MySQL, Oracle и др., что позволяет эффективно управлять базами данных проекта.

Удобство использования: DataGrip предоставляет удобный пользовательский интерфейс и интуитивно понятные инструменты для работы с данными, выполнения запросов, создания схем и других операций, что упрощает работу с базами данных.

Интеграция с другими инструментами: DataGrip интегрируется с другими инструментами разработки, такими как VSCode, что обеспечивает синхронизацию данных и возможность многопоточной работы над проектом.

### Postman

Postman - это инструмент для тестирования API и разработки HTTP-запросов. Я выбрал Postman по следующим причинам:

* Тестирование и отладка API: Postman предоставляет удобный интерфейс для создания, отправки и отладки HTTP-запросов, что позволяет мне эффективно тестировать и взаимодействовать с API, используемыми в проекте.
* Коллекции запросов: Postman позволяет создавать коллекции запросов, что упрощает организацию и повторное использование запросов в процессе разработки.
* Автоматизация: Postman позволяет автоматизировать процессы тестирования и интеграции API с помощью скриптов и коллекций, что повышает эффективность разработки и тестирования.

### Git

Git является распределенной системой контроля версий, которая позволяет эффективно управлять и отслеживать изменения в кодовой базе проекта. Я выбрал Git по следующим причинам:

* Контроль версий: Git обеспечивает полный контроль над версиями кода, позволяя отслеживать изменения, вносить исправления и восстанавливать предыдущие версии кода при необходимости.
* Коллаборация: Git облегчает коллаборацию в разработке, позволяя разработчикам работать над одним и тем же проектом одновременно, объединять изменения и управлять конфликтами.
* Ветвление: Git поддерживает мощные механизмы ветвления, позволяющие создавать и объединять ветки кода для параллельной разработки различных функциональностей или исправлений.

### Docker

Docker представляет собой платформу для разработки, доставки и запуска приложений в контейнерах. Я выбрал Docker по следующим причинам:

* Портабельность: Docker контейнеры обеспечивают высокую портабельность приложений, позволяя разработчикам упаковывать приложение и все его зависимости в контейнер, который может быть запущен на любой совместимой с Docker платформе.
* Изоляция: Docker обеспечивает изоляцию приложений и их зависимостей, что позволяет избежать конфликтов между различными компонентами системы и обеспечить стабильность и безопасность работы приложения.
* Масштабируемость: Docker позволяет масштабировать приложение путем запуска нескольких контейнеров с одним и тем же приложением, что обеспечивает горизонтальное масштабирование и управление нагрузкой.

## 2.5 Архитектура системы

Для проекта было принято решение выбрать архитектуру приближенную к микроервисной. Микросервисная архитектура является современным подходом к разработке и организации информационных систем. В этой архитектуре система разбивается на набор независимых сервисов, каждый из которых представляет собой отдельно работающую и развертываемую единицу. Каждый микросервис выполняет определенные функции и может быть разработан и масштабирован независимо от других сервисов.

Основные характеристики микросервисной архитектуры включают:

1. Декомпозиция на микросервисы: Система разбивается на маленькие и самодостаточные сервисы, каждый из которых отвечает за конкретную функцию или возможность.
2. Независимость и автономность: Каждый микросервис разрабатывается, развертывается и масштабируется независимо от других сервисов. Это позволяет быстро развивать и внедрять новые функции, а также упрощает поддержку и масштабирование системы.
3. Слабая связность: Микросервисы взаимодействуют друг с другом через строго определенные интерфейсы, обычно используя протоколы взаимодействия, такие как REST API или сообщения. Это обеспечивает слабую связность между сервисами и позволяет легко заменять или обновлять отдельные компоненты системы.
4. Распределенность: Микросервисы могут быть развернуты на разных серверах или даже в разных облаках, что обеспечивает гибкость и отказоустойчивость системы. Распределенность также позволяет масштабировать отдельные сервисы в зависимости от нагрузки.
5. Повышенная гибкость и инновационность: Микросервисная архитектура позволяет быстро внедрять новые функции и экспериментировать с новыми технологиями, поскольку каждый сервис может быть разработан и развернут отдельно от других.

Микросервисная архитектура для разработки информационной системы обусловлен ее преимуществами в сравнении с традиционными монолитными системами. Микросервисная архитектура обеспечивает лучшую гибкость, масштабируемость, возможность внедрения новых функций и упрощенную поддержку системы. Она также позволяет более эффективно использовать ресурсы и улучшает отказоустойчивость системы.

В данном проекте выбор микросервисной архитектуры обоснован необходимостью создания гибкой и масштабируемой системы учета задач. Микросервисы позволят независимо разрабатывать и масштабировать различные функциональные компоненты системы, такие как управление задачами, аналитика, уведомления и другие. Такой подход обеспечит более гибкое и эффективное управление разработкой и поддержкой системы, а также облегчит интеграцию с другими сервисами и системами.

Микросервисная архитектура также обеспечит возможность использования современных технологий и инструментов, таких как контейнеризация с помощью Docker, оркестрация с помощью Kubernetes и микросервисный фреймворк, такой как NestJS. Это обеспечит высокую производительность, отказоустойчивость и масштабируемость системы.

В целом, выбор микросервисной архитектуры для разработки данной информационной системы обеспечит гибкость, масштабируемость, инновационность и легкость поддержки. Этот подход позволит достичь высокой эффективности в учете задач и удовлетворить требования пользователей, а также обеспечит возможность дальнейшего развития и расширения системы в будущем.

### 2.5.1 Компоненты системы

Система состоит из нескольких компонентов (микросервисов и приложений). Компоненты системы являются ключевыми строительными блоками системы, отвечающими за выполнение конкретных функций и предоставление определенных сервисов. Архитектурное разбиение системы на компоненты позволяет нам достичь модульности, переиспользования и легкости сопровождения.

Компоненты системы представляют собой независимые логические или физические единицы, которые выполняют определенные задачи и взаимодействуют друг с другом для достижения общей цели. Каждый компонент обладает определенными интерфейсами, через которые он взаимодействует с другими компонентами и с окружающей средой. Схема позволит нам лучше понять архитектурное устройство нашей информационной системы и рассмотреть ее основные модули и компоненты. Мы сможем проанализировать их взаимодействие, выделить ключевые функциональные области и обсудить возможности расширения и модификации системы.

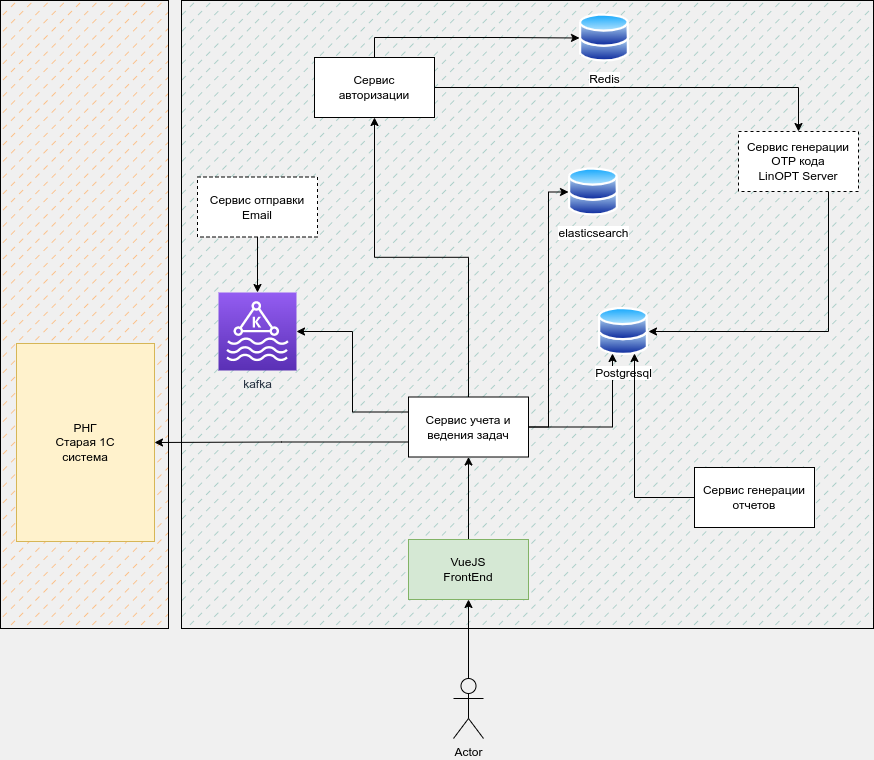


Рисунок x Схема компонентов системы

### 2.5.2 Архитектура базы данных. ERD

В данном разделе будет рассмотрена схема базы данных нашей информационной системы в виде диаграммы ERD (Entity-Relationship Diagram). Диаграмма ERD является важным инструментом для визуализации и организации структуры данных в системе. База данных является основой любой информационной системы, в которой хранятся и организуются данные. Диаграмма ERD позволяет нам логически представить сущности (entities) в системе, их атрибуты (attributes) и связи (relationships) между ними. Она помогает нам лучше понять структуру данных, их взаимосвязи и представить их в удобном и понятном формате. В данной диаграмме ERD мы сосредоточимся на основных сущностях и их связях в нашей системе. Мы рассмотрим таблицы базы данных, атрибуты каждой сущности и связи между ними. Такая визуализация позволит нам лучше организовать и структурировать данные в системе, а также понять, какие связи и зависимости существуют между различными сущностями. Анализ диаграммы ERD поможет нам определить ключевые таблицы и их атрибуты, спланировать структуру базы данных и обеспечить целостность и эффективность хранения данных. Мы сможем лучше понять взаимосвязи между различными сущностями и оптимизировать доступ и манипуляции с данными.

Рассмотрим диаграмму ERD нашей базы данных и проведем анализ ее структуры. Мы сможем лучше понять, как данные организованы в нашей системе и как они взаимодействуют друг с другом.

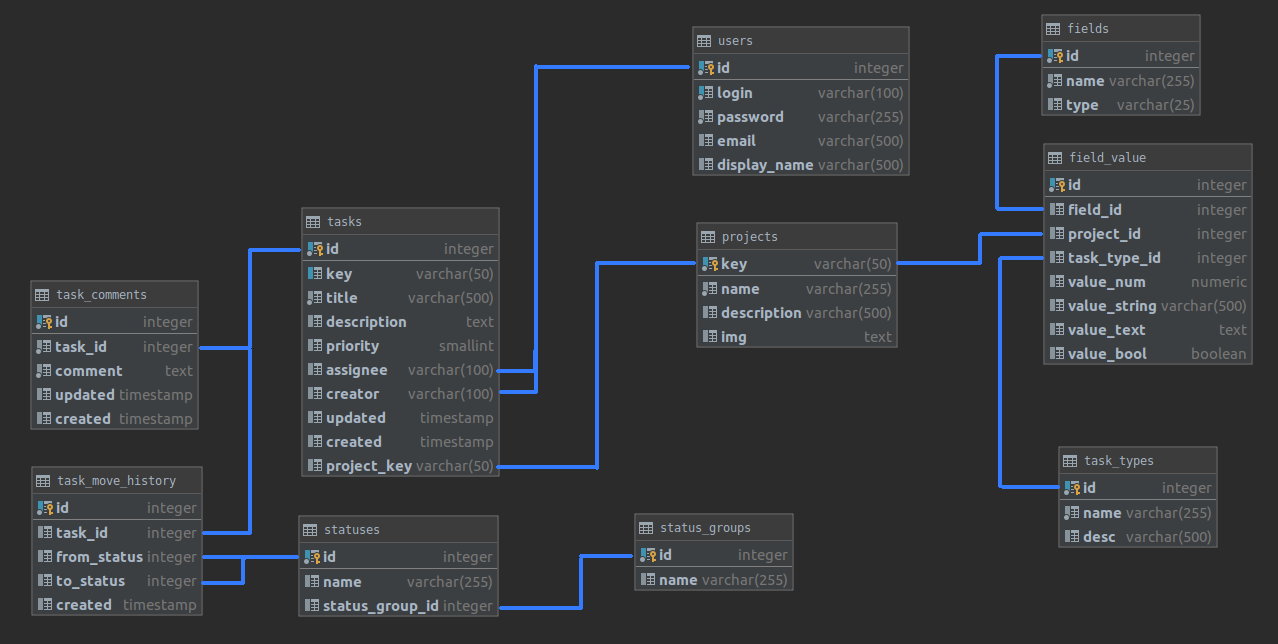


Рисунок x Схема базы данных postgresql (ERD)

|  |  |
| --- | --- |
| sasdsa |  |
|  |  |
|  |  |

На диаграмме отображены таблицы и их связи внешних ключей, в целом можно выделить основные таблицы:  
tasks - таблица содержит данные задач, к ней относятся вспомогательные таблицы, такие как feilds - дополнительные поля, project - проект к которому относится задача, task\_types - тип задачи, позволяет иметь в одном проекте несколько разнообразных задач, statuses - набор возможных статусов задачи, task\_comment - комментарии оставленные в задаче, task\_move\_history - таблица хранит в себе историю переходов по статусам, она необходима для расчета отчетов.  
user - таблица хранит в себе записи пользователей, она также относится к таблице tasks, для представления исполнителей и автором задач.

## 2.5.3 Структура системы. Диаграмма классов

Диаграмма классов является одной из ключевых диаграмм в проектировании информационных систем. Она позволяет визуализировать структуру системы, описывая классы, их атрибуты, методы и взаимосвязи. Эта диаграмма играет важную роль в проектировании объектно-ориентированных систем, помогая разработчикам лучше понять и представить компоненты системы и их взаимодействие. Ниже рассмотрены основные аспекты проектирования, связанные с описанием классов и их свойств. Мы изучим различные типы связей между классами, такие как ассоциация, наследование и агрегация, и узнаем, как эти связи отражаются на диаграмме классов.

Диаграмма классов позволяет представить визуальную модель системы, обозначить важные компоненты, определить атрибуты и методы каждого класса. Она помогает команде разработчиков иметь общее представление о структуре системы, анализировать взаимодействие классов и оптимизировать архитектуру при необходимости. Диаграмма классов является мощным инструментом для проектирования систем, и она станет фундаментальным компонентом нашего анализа и проектирования информационной системы.

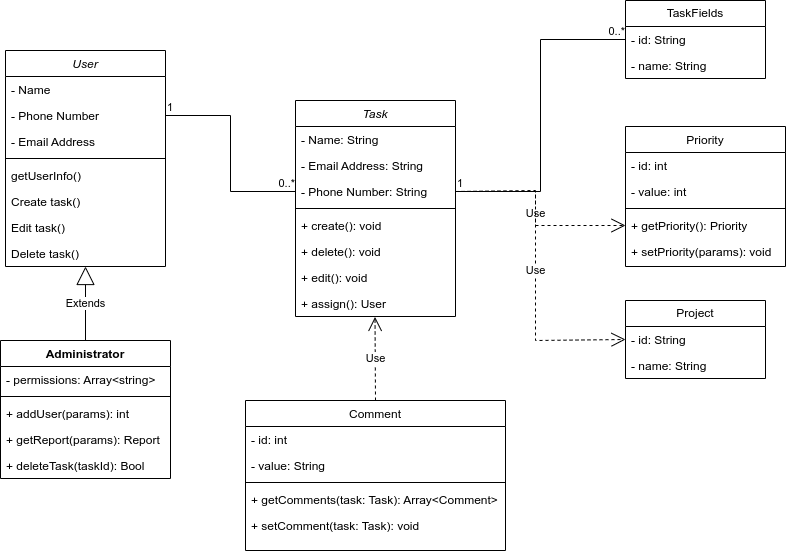


Рисунок x Диаграмма классов

На диаграмме видны основные сущности системы, и их связи. Видно что основные сущности это пользователи и задачи, и есть некоторые зависимости (неполноценные части) задач.

### 2.5.4 Диаграмма деятельности (Activity)

Диаграмма активности (Activity diagram) является одной из важных диаграмм в методологии UML (Unified Modeling Language). Она позволяет моделировать и визуализировать последовательность действий и потоки управления в процессах и системах. Диаграмма активности помогает наглядно представить, какие действия выполняются, какие решения принимаются и какие ресурсы используются в рамках определенного процесса. Диаграмма активности поможет нам более полно понять функциональность системы, процессы ее работы и связи между различными компонентами. Мы сможем проанализировать эффективность процессов, выявить потенциальные проблемы и улучшить взаимодействие между различными активностями.

В данном разделе предоставлена диаграмму активности, которая иллюстрирует процесс создания задачи в нашей информационной системе. Рассмотрим основные элементы и символы диаграммы активности. Она поможет нам лучше понять функциональность и последовательность действий в рамках нашего проекта. Ниже рассмотрен пример запроса списка задач в виде диаграммы деятельности:

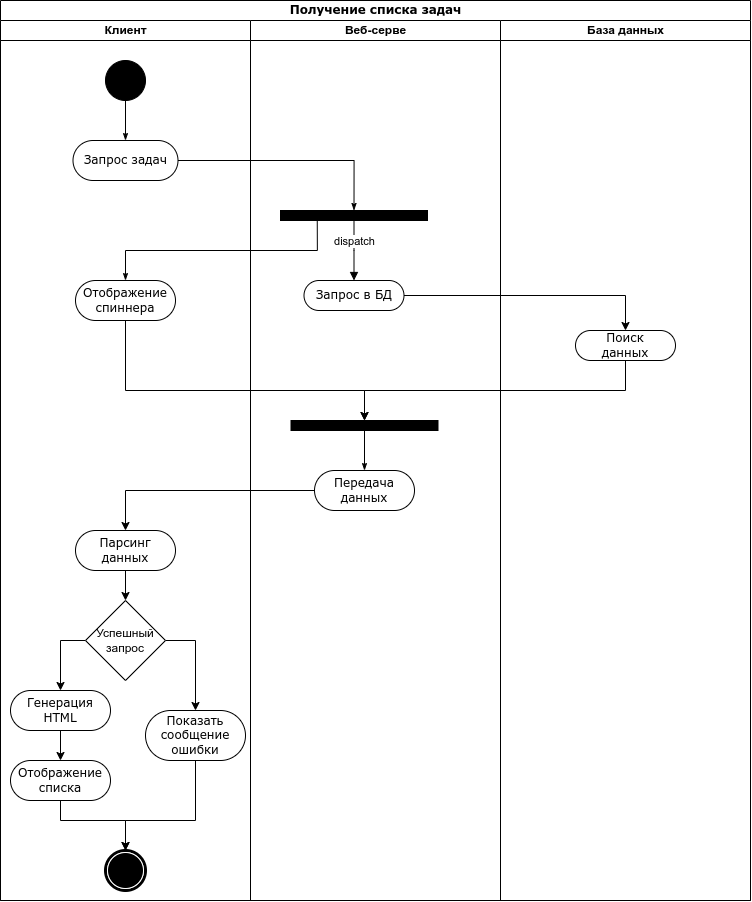


Рисунок х Диаграмма деятельности. Получение списка задач

## 2.6 Сторонние модули

В процессе разработки информационной системы мы рассмотрели различные модули и компоненты, которые не относятся к основному функционалу системы, но играют важную роль в ее работе. В данном разделе мы сосредоточимся на описании таких сторонних модулей, которые обеспечивают дополнительные функциональные возможности и улучшают производительность и эффективность системы.

Сторонние модули являются внешними компонентами или инструментами, которые интегрируются с основной информационной системой и расширяют ее возможности. Они могут предоставлять функциональность, которая не является прямой частью основного бизнес-процесса, но важна для оптимизации работы системы или обеспечения дополнительных сервисов.

В данном разделе рассмотрим несколько сторонних модулей, которые были внедрены в нашу информационную систему. Каждый модуль будет описан в контексте его функциональности, преимуществ и вклада в общую работу системы. Среди рассматриваемых сторонних модулей включен модуль генерации отчетов, модуль отправки электронных писем, а также другие модули, которые улучшают взаимодействие с пользователем, повышают безопасность или обеспечивают интеграцию с внешними системами.

Анализ и описание этих сторонних модулей позволит нам лучше понять их важность и роль в нашей информационной системе. Мы сможем оценить, как эти модули способствуют улучшению функциональности и обеспечению требуемого уровня производительности и надежности системы.

### LinOTP

LinOTP Server является одним из ключевых сторонних модулей, интегрированных в нашу информационную систему. LinOTP - это сервер аутентификации с открытым исходным кодом, который обеспечивает мощные и гибкие возможности двухфакторной аутентификации. Он предоставляет надежную защиту системы, требуя от пользователей не только пароль, но и дополнительный фактор подтверждения, такой как одноразовый пароль, код на мобильном устройстве или биометрические данные.

LinOTP Server интегрируется в нашу информационную систему для обеспечения безопасности и защиты доступа к важным ресурсам. Он предоставляет API для взаимодействия с системой аутентификации и позволяет нашим пользователям использовать двухфакторную аутентификацию при входе в систему или выполнении чувствительных операций. Это значительно повышает уровень безопасности системы, защищая ее от несанкционированного доступа и потенциальных угроз.

### Сервис отправки электронных писем

Сервис отправки электронных писем является еще одним важным сторонним модулем, внедренным в нашу информационную систему. Он предоставляет функциональность отправки электронных писем пользователям системы, а также интеграцию с внешними почтовыми провайдерами.

Сервис отправки электронных писем обеспечивает надежную и эффективную доставку сообщений, обеспечивая коммуникацию между системой и пользователями. Он позволяет генерировать и отправлять уведомления, оповещения о событиях, подтверждения и другие сообщения, необходимые для информирования пользователей о важных событиях и обновлениях.

Интеграция сервиса отправки электронных писем позволяет нам легко управлять процессом отправки и получения сообщений, а также обеспечивает возможность настройки шаблонов писем, персонализации сообщений и контроля доставки. Это значительно улучшает взаимодействие с пользователями, обеспечивая своевременную и информативную коммуникацию.

Сервис отправки электронных писем является неотъемлемой частью нашей информационной системы, обеспечивая эффективное взаимодействие с пользователями и обеспечивая своевременную доставку важных сообщений и уведомлений.

### Report Service

Сервис, разработанный на базе фреймворка NestJS, который предоставляет функциональность по генерации отчетов в форматах XLSX и DOCX. Он является ключевым компонентом нашей информационной системы и обеспечивает автоматизированное создание и экспорт отчетов для пользователей.  
Сервис генерации отчетов осуществляет интеграцию с различными источниками данных в нашей системе, включая базу данных и другие внешние системы. Он обеспечивает получение необходимых данных, их обработку и форматирование в соответствии с заданными шаблонами отчетов. С использованием мощных библиотек и инструментов, таких как "xlsx" и "docx", сервис создает качественные и профессионально оформленные отчеты.

Функциональность сервиса позволяет нашим пользователям генерировать разнообразные отчеты, включая сводные таблицы, диаграммы, списки данных и другие типы информационных документов. Сервис поддерживает гибкую настройку шаблонов отчетов, что позволяет адаптировать их под конкретные потребности и требования пользователей. Благодаря использованию форматов XLSX и DOCX, отчеты могут быть легко открыты и просмотрены на различных платформах и устройствах без необходимости установки специального программного обеспечения.

Сервис генерации отчетов на базе NestJS обеспечивает надежность и стабильность работы, а также простоту в развертывании и управлении. Он интегрируется плотно с другими компонентами нашей информационной системы, обеспечивая гармоничную работу и удобство использования для пользователей.

## 2.7 Интеграционная часть

Для получения абонентских данных, предполагается интеграция с 1С системой под названием РНГ “Расчет с населением за газ”, для получения абонентских данных.  
При интеграции с РНГ, возникла проблема, с той особенностью что РНГ система не предоставляет API методы для удобного взаимодействия. Решением было интегрироваться непосредственно с базой данных MSSQL, на которой основана система 1С. Для интеграции с базой данных 1С, которая основана на Microsoft SQL Server (MSSQL), используется язык SQL (Structured Query Language) для получения данных и взаимодействия с базой данных. Формируем SQL-запросы, которые выполняются на сервере MSSQL, чтобы получить нужные данные из базы данных 1С. При формировании SQL-запросов учитывается структуру и схема базы данных 1С. Определяются нужные таблицы, поля и связи между ними, чтобы получить необходимые данные. Затем используются драйверы базы данных, которые обеспечивают соединение с MSSQL сервером и выполнение SQL-запросов.

Полученные данные парсятся и обрабатываются в нашей системе, чтобы использовать их в соответствии с требованиями проекта. Применяются соответствующие методы и алгоритмы для извлечения нужных значений, выполнения вычислений или обновления данных в базе данных 1С. Интеграция с базой данных 1С через MSSQL позволяет нам получать актуальные данные из системы 1С и использовать их в нашей информационной системе. Это обеспечивает согласованность и обмен информацией между различными системами, что важно для эффективного функционирования нашей системы учета задач.

# Глава 3. Документационная часть

## 3.1 Руководство программиста

### 3.1.1 Требование к среде

Для успешного запуска и разработки приложений на Vue.js, Nest.js и серверных приложений на Node.js + Express.js, необходима определенная среда, которая должна быть установлена на вашей системе. В этом разделе мы опишем требования к среде разработки для каждого из этих фреймворков и технологий.

#### 3.1.1.1 Операционная система

Для максимальной стабильности и эффективности рекомендуется использовать операционную систему Linux в качестве основной платформы разработки. Все скрипты запуска написаны для bash интерпритатора, а почти все дистрибутивы линукс поддерживают этот интерпретатор. Рекомендуется использовать CentOS, благодаря своей надежности и широкой поддержке. Linux обеспечивает поддержку систем контейнеризации и нативную среду, специально настроенную, для разработки и выполнения веб-приложений, а также обладает богатым набором инструментов и библиотек для программистов.

#### 3.1.1.2 Система контейнеризации Docker

Для создания удобной и изолированной среды разработки и развертывания приложений рекомендуется использовать систему контейнеризации Docker. Docker позволяет упаковывать приложения и их зависимости в контейнеры, которые могут быть легко переносимыми и повторно используемыми на разных платформах.

Преимущества использования Docker включают:

1. Изолированное окружение

Docker обеспечивает изолированную среду, где каждое приложение работает в своем собственном контейнере. Это значит, что каждое приложение и его зависимости находятся в изолированном окружении, не влияющем на другие приложения или саму операционную систему.

1. Переносимость

Контейнеры Docker могут быть запущены на разных операционных системах и платформах, обеспечивая высокую переносимость и совместимость приложений. Это означает, что вы можете разрабатывать и развертывать свои приложения в среде разработки и затем перенести их на производственные серверы без необходимости повторной настройки и установки зависимостей.

1. Управление зависимостями.

Docker позволяет управлять зависимостями приложений и обеспечивает их легкую установку и настройку. Вы можете определить все необходимые зависимости в файле конфигурации Docker, и Docker автоматически загрузит их при создании контейнера. Это делает процесс установки и настройки зависимостей проще и более надежным.

1. Масштабируемость  
   Docker обеспечивает гибкую масштабируемость приложений. Вы можете запускать несколько экземпляров контейнеров с одним и тем же приложением, что позволяет распределить нагрузку и обеспечить высокую доступность. Docker также обеспечивает удобные инструменты для управления контейнерами и масштабирования приложений.

Для использования Docker вам необходимо установить Docker Engine на вашу систему и ознакомиться с базовыми командами Docker для работы с контейнерами. Обратитесь к официальной документации Docker для получения подробных инструкций по установке и настройке.

Использование Docker в среде разработки и развертывания приложений существенно упрощает процесс разработки, обеспечивает надежность и переносимость, а также упрощает управление зависимостями и масштабирование. Рекомендуется ознакомиться с Docker и использовать его в своих проектах для обеспечения эффективной и удобной работы.

#### 3.1.1.3 Окружение для локального запуска

При запуске сервисов в докер контейнере необходимая среда обычно устанавливается при запуске контейнера, но для локальной разработки и запуска проекта вам потребуется настроить определенное окружение.

1. **Git**

Git является распределенной системой контроля версий, неотъемлемой частью разработки программного обеспечения. Убедитесь, что у вас установлен Git на вашей локальной машине. Git позволяет вам управлять версиями кода, сотрудничать с другими разработчиками и управлять историей изменений в вашем проекте.

1. **Node.js**

Node.js является средой выполнения JavaScript на стороне сервера, и его необходимо установить для запуска проекта. Убедитесь, что у вас установлена последняя версия Node.js на вашей локальной машине. Node.js обеспечивает среду, в которой вы можете выполнять серверные операции и устанавливать зависимости для вашего проекта.

1. **NestJS CLI**

NestJS CLI - это интерфейс командной строки, предоставляемый фреймворком NestJS. Он упрощает создание, разработку и запуск проектов на NestJS. Установите NestJS CLI глобально на вашей локальной машине, чтобы использовать его для создания и запуска проекта NestJS.

1. **Vue.js CLI**

Vue.js CLI - это интерфейс командной строки для разработки приложений на Vue.js. Установите Vue.js CLI глобально на вашей локальной машине, чтобы использовать его для создания и запуска проектов на Vue.js. Vue.js CLI предоставляет удобные инструменты для разработки, сборки и тестирования приложений Vue.js.

Установите и настройте все вышеупомянутые компоненты в соответствии с инструкциями, предоставленными каждым инструментом. Убедитесь, что все зависимости установлены корректно и готовы к запуску вашего проекта на локальной машине.

После того, как вы настроили окружение для локального запуска, вы будете готовы разрабатывать и тестировать свой проект на своей локальной машине перед развертыванием на более широкой среде.

### Обзор файловой структуры, кода и модулей

Весь проект хранится в едином репозитории, вот небольшое описание того, что такое монорепозиторий:

Монорепозиторий (или монолитный репозиторий) - это подход к организации разработки программного обеспечения, при котором весь код и связанные файлы проекта хранятся в одной централизованной репозитории. В отличие от подхода с множеством отдельных репозиториев, где каждый модуль или компонент имеет свой собственный репозиторий, монорепозиторий объединяет весь код и его зависимости в единую структуру.  
В монорепозитории код разных модулей или компонентов проекта хранится вместе, обычно в одной директории или наборе вложенных директорий. Это позволяет разработчикам легко обращаться к различным частям кода и координировать изменения между ними. Вместе с тем, монорепозиторий предоставляет единое место для управления версиями, отслеживания изменений и управления зависимостями.  
Одним из преимуществ использования монорепозитория является упрощение процесса разработки и поддержки проекта. Разработчики могут легко обмениваться кодом, изучать и вносить изменения в различные модули, а также проводить общие тестирования и интеграцию кода. Это способствует улучшению сотрудничества и общей эффективности разработки.  
В контексте данного отчета использование монорепозитория означает, что весь код, модули и файлы проекта хранятся в одной централизованной репозитории. Это позволяет нам легко организовывать и анализировать структуру кода и модулей, а также обеспечивает единое место для управления версиями и изменениями.

#### 3.1.2.1 Структура проекта

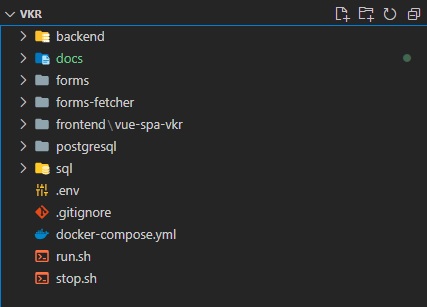


Рисунок х Общая структура папок

На рисунке видны директории сервисов, файлы скриптов, и директория с документацией. Рассмотри подробнее каждый сервис.

#### 3.1.2.2 Директория backend

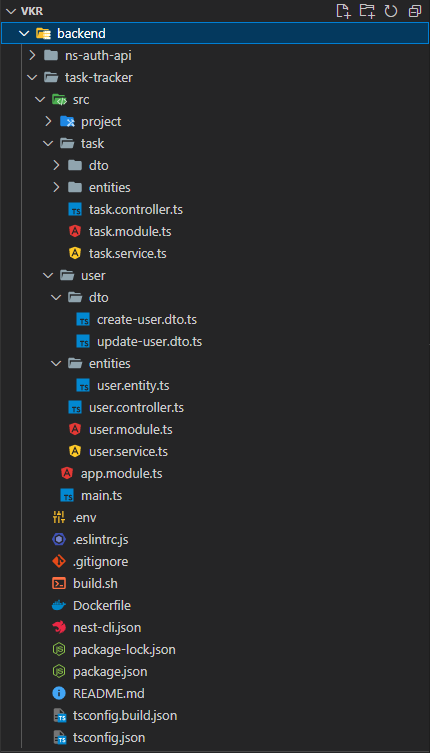


Рисунок х Директория backend

На рисунке видно, что директорий backend содержит две поддиректории: ns-auth-api и task-tracker, это директории сервисов, которые отвечают за авторизацию и управление задачами соответственно.

На рисунке развернута директория лишь одного сервиса - task-tacker. По структуре сервиса можно определить, что используемый язык Typescript.

#### 3.1.2.3 Описание сервиса task-tacker

Сервисы построены на основе фреймворка NestJS, не все комоненты фреймвокра видны на рисунке, но есть базовый набор компонентов фреймворка, о которых нужно знать перед началом работы с ним:

**Корневая директория**: Это основная директория вашего проекта, в которой находится файл package.json и другие файлы конфигурации. Здесь вы можете также разместить глобальные модули, общие утилиты и файлы, которые не относятся к конкретным модулям.

**src**: Эта директория содержит исходный код вашего приложения NestJS. В ней обычно размещаются модули, контроллеры, провайдеры, фильтры, интерсепторы и другие файлы, связанные с бизнес-логикой приложения.

**Модули (Modules)**: Директория src содержик корневой модуль app, все модульные файлы имеют характерное расширение module.ts. В данном случае, поддиректории внутри src, являются модулями. Каждый модуль является логическим блоком вашего приложения и содержит набор компонентов, связанных с определенной функциональностью. Внутри директории src обычно создаются поддиректории для каждого модуля, например, user, project, task и т.д. Каждый модуль содержит файлы, такие как контроллеры, сервисы, провайдеры и другие файлы, связанные с этим модулем.

**Контроллеры (Controllers)**: Контроллеры обрабатывают входящие HTTP-запросы и управляют потоком данных. Они служат точкой входа в ваше приложение и определяют маршруты, по которым обрабатываются запросы. Контроллеры часто используются для принятия запросов, вызова соответствующих сервисов и возврата ответов клиенту.

**Сервисы (Services)**: Сервисы предоставляют бизнес-логику вашего приложения. Они обычно выполняют операции, связанные с базой данных, внешними API или другими компонентами системы. Сервисы могут быть внедрены (injected) в контроллеры или другие сервисы для повторного использования кода и обеспечения модульности.

**Провайдеры (Providers)**: Провайдеры в NestJS предоставляют различные объекты и зависимости, которые используются в вашем приложении. Они могут включать сервисы, репозитории базы данных, конфигурационные объекты, внешние модули и многое другое. Провайдеры могут быть внедрены в контроллеры, сервисы или другие провайдеры с помощью механизма внедрения зависимостей (Dependency Injection).

**Фильтры (Filters)**: Фильтры используются для обработки исключений и ошибок в вашем приложении. Они позволяют вам глобально или локально обрабатывать исключения, возникающие в процессе обработки запросов.

**Интерсепторы (Interceptors)**: Интерсепторы позволяют вам перехватывать и изменять запросы и ответы на разных стадиях их обработки. Например, вы можете использовать интерсепторы для применения общих логических операций, таких как логирование, кэширование или преобразование данных.

**DTO (Data Transfer Objects)**: DTO представляют объекты данных, которые используются для передачи информации между различными компонентами вашего приложения. Они служат контрактом между клиентским кодом (например, HTTP-запросами) и серверной частью приложения. DTO часто используются в контроллерах для валидации, сериализации и десериализации данных.

**Сущности (Entities)**: Сущности представляют объекты и таблицы в базе данных, связанные с вашим приложением. Они представляют структуру и схему данных, которые сохраняются и извлекаются из базы данных. Сущности обычно содержат аннотации или декораторы, которые определяют их свойства и отношения с другими сущностями.

**Репозитории (Repositories)**: Репозитории предоставляют абстракцию для доступа к данным и выполнения операций базы данных. Они предоставляют методы для создания, чтения, обновления и удаления данных, связанных с сущностями. Репозитории часто используются в сервисах для взаимодействия с базой данных.

**Миграции (Migrations)**: Миграции представляют собой набор скриптов или файлов, которые обновляют схему базы данных при развертывании или изменении вашего приложения. Они обеспечивают согласованность структуры базы данных с текущей версией вашего приложения и позволяют эффективно управлять изменениями в базе данных.

#### 3.1.2.4 Директория frontend

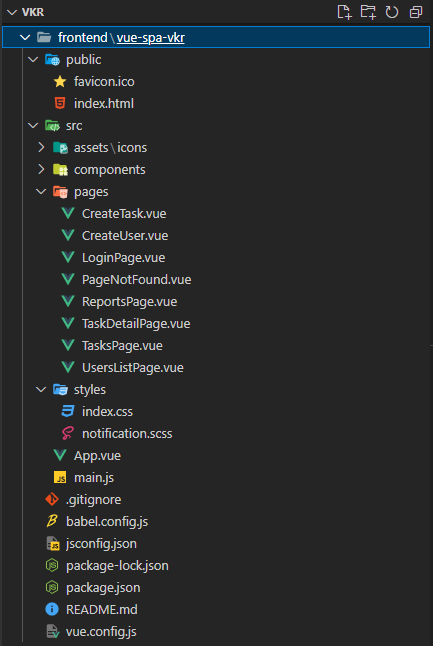


Рисунок х Директория frontend

Внешний вид (пользовательский интерфейс) построен на фреймворке Vue 3, в директории frontend имеется только одна директория vue-spa-vkr, говорящая о том, что сервис является SPA приложением, построенное на Vue 3 фреймворке, вот краткое описание того, что такое SPA приложение:

SPA (Single Page Application) — это веб-приложение, которое загружает одну HTML-страницу при инициализации и динамически обновляет ее содержимое без перезагрузки страницы при взаимодействии пользователя.

SPA-приложения на Vue.js 3 основаны на компонентах и позволяют создавать богатые интерактивные пользовательские интерфейсы. Вместо того чтобы загружать отдельные HTML-страницы для каждого действия или перехода, SPA использует клиентскую маршрутизацию и асинхронные запросы к серверу для динамического обновления контента на одной странице.

В контексте Vue.js 3, SPA-приложение строится вокруг компонента App.vue, который является главным компонентом приложения. Он содержит макет и определение основной структуры страницы.  
SPA на Vue.js 3 использует Vue Router для управления маршрутизацией. Маршруты определяются в файле router/index.js или аналогичном месте. Vue Router позволяет определить пути URL и связать их с соответствующими компонентами, которые будут отображаться при переходе по маршруту.  
Компоненты в SPA-приложении Vue.js 3 используются для описания различных частей пользовательского интерфейса и функциональности. Они состоят из шаблонов, которые определяют структуру HTML-разметки, стилей для оформления и JavaScript-кода, который определяет поведение компонента.  
SPA-приложения на Vue.js 3 также могут использовать Vuex для управления состоянием приложения. Vuex предоставляет глобальное хранилище, в котором хранятся данные и методы для их изменения и доступа из компонентов.  
Основным преимуществом SPA-приложений является более быстрый и плавный пользовательский опыт благодаря отсутствию перезагрузки страницы при каждом действии пользователя. SPA также позволяют создавать более интерактивные и реактивные пользовательские интерфейсы с помощью современных фреймворков, таких как Vue.js 3.

#### 3.1.2.5 Описание директорий сервиса vue-spa-vkr

main.js: Этот файл является точкой входа вашего приложения Vue.js 3. В нем вы можете настроить основные настройки, подключить плагины, создать экземпляр Vue и монтировать его на элемент в вашем HTML-файле.

App.vue: Это главный компонент вашего приложения. Он является родительским компонентом для всех остальных компонентов и определяет общую структуру и макет вашего приложения.

src: Это основная директория вашего проекта Vue.js 3, в которой находится исходный код.

assets: В этой директории вы можете разместить статические ресурсы, такие как изображения, шрифты или файлы стилей.

components: Здесь вы можете создавать компоненты Vue.js, которые представляют отдельные части пользовательского интерфейса. Каждый компонент может состоять из шаблона (HTML разметка), стилей (CSS или SCSS) и скрипта (JavaScript или TypeScript), который определяет логику компонента.

directives: Если в вашем проекте есть пользовательские директивы Vue.js, вы можете поместить их в эту директорию. Директивы позволяют вам манипулировать DOM-элементами напрямую.

mixins: Если в вашем проекте есть mixins, то вы можете создать их в этой директории. Миксины - это объекты, которые содержат повторно используемый код, который может быть включен в компоненты для добавления функциональности.

plugins: Если вы используете плагины в своем проекте, вы можете поместить их в эту директорию. Плагины обычно используются для добавления глобальных функциональностей, расширения функциональности Vue.js или интеграции с сторонними библиотеками.

router: Если вы используете Vue Router для управления маршрутизацией в вашем приложении, вы можете создать файлы маршрутизации в этой директории. Они определяют пути и соответствующие компоненты, которые должны быть отображены для каждого пути.

store: Если вы используете Vuex для управления состоянием вашего приложения, вы можете создать файлы хранилища (store) в этой директории. Хранилище содержит состояние приложения и методы для его изменения и доступа.

views: В этой директории можно разместить компоненты, которые представляют отдельные страницы вашего приложения. Как и компоненты, представления могут содержать шаблон, стили и скрипты, но обычно представляют целые страницы.

#### 3.1.2.6 Описание директорий сервиса forms

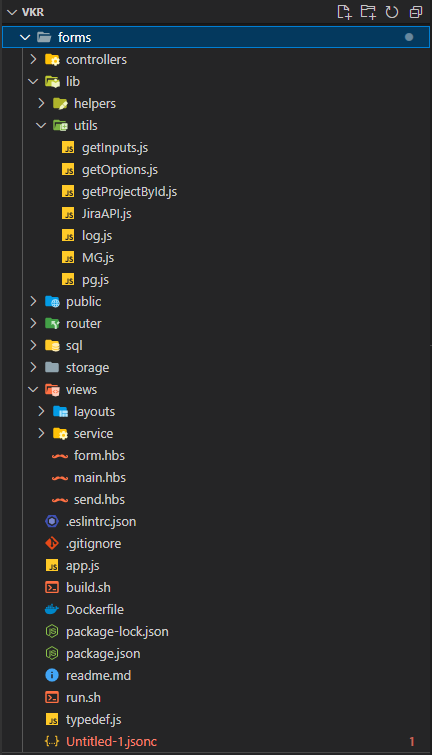


Рисунок х Директория сервиса forms

Сервис forms - является сайтом, построенным на Express.js фреймворке и использующий MVC (Model-View-Controller) подход с шаблонизатором Handlebars.

Express.js является минималистичным и гибким фреймворком, который предоставляет основу для создания веб-приложений. В сочетании с MVC подходом и Handlebars вам предоставляется структура и инструменты для разделения логики, данных и отображения, что облегчает разработку и поддержку вашего сервиса.

В данном сервисе содержатся такие сущьности, как:

Представления (Views): Представления отвечают за отображение данных на стороне клиента. В случае использования Handlebars, представления могут быть созданы в виде шаблонов, которые содержат HTML-разметку с встроенными выражениями и переменными. Handlebars позволяет вам встраивать данные в шаблоны и генерировать динамический HTML-код.

Контроллеры (Controllers): Контроллеры обрабатывают входящие запросы и взаимодействуют с моделями и представлениями. Они являются промежуточным звеном между маршрутами и моделями, координируя выполнение бизнес-логики и определяя, какие данные будут отображены в представлениях. Контроллеры содержат методы, которые обрабатывают различные действия, такие как создание, чтение, обновление и удаление данных.

Маршруты (Routes): Маршруты определяют URL-адреса и соответствующие им обработчики запросов. В Express.js маршруты определяются с помощью методов, таких как app.get(), app.post(), app.put(), app.delete() и т.д. Маршруты связываются с соответствующими контроллерами, которые выполняют необходимые действия и возвращают данные для отображения.

Статические файлы (Static Files): В сервисе на Express.js вы можете определить папку для статических файлов, таких как изображения, CSS-стили или JavaScript-скрипты. Эти файлы будут доступны напрямую по URL-адресу без необходимости обработки маршрутов.

Шаблоны Handlebars (Handlebars Templates): Шаблоны Handlebars представляют собой файлы с расширением .hbs или .handlebars, которые содержат HTML-разметку с встроенными выражениями Handlebars. В этих шаблонах вы можете использовать условия, циклы, вставку переменных и другие конструкции Handlebars для генерации динамического контента в представлениях.

Сложно описать все нюансы сервиса в данном документе, лишь чтение исходного кода даст более точное понимание логики работы сервиса.

#### 3.1.2.7 Описание директорий сервиса forms-fetchet

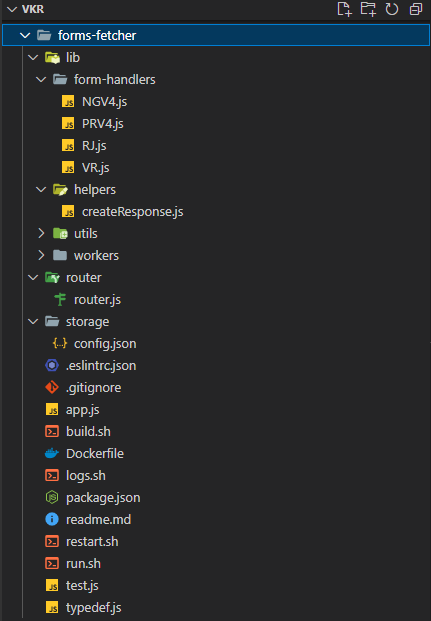


Рисунок х Структура сервиса forms-fetcher

forms-fetcher также как и forms построен на фреймворке express.js, но является фоновым микросервисом, и не имеет визуального пользовательского интерфейса (UI). Его цель - проверка базы данных на предмет наличия новых заявок, отправленных с сайта forms, для дальнейшей ее обработки, и создании заявки в система учета задач.

#### 3.1.2.8 Описание директории postgresql

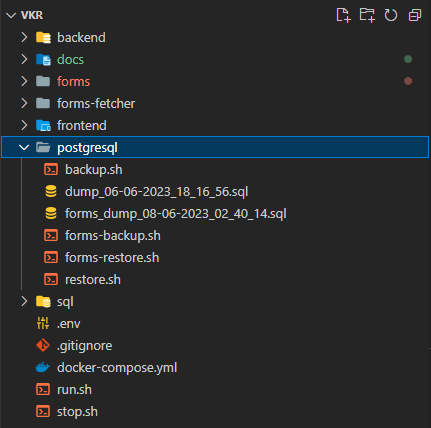


Рисунок х Содержимое директории postgresql

Директория postgresql содержит скрипты бэкапа и развертывания миграций с данными в базу данных postgresql, для корректного запуска и работоспособности микросервисов.

### 3.1.3 Запуск приложения

Запустить микросервисы можно двумя способами, в докере и локально, рассмотрим оба способа.

#### Запуск в докере

Для запуска сервисов в докер-контейнерах, предназначены корневые скрипты run.sh для запуска, и stop.sh для остановки, так же важно в корневой директории в .env файле описать переменные окружения, для доступа к БД и прочие параметры.

В случае если запуск глобальным скриптом не удался, можно собрать образы для каждого сервиса отдельно, для этого в директории каждого сервиса предусмотрен файл build.sh, который автоматически запускает сборку образа для текущей директории (на основе Dockerfile).

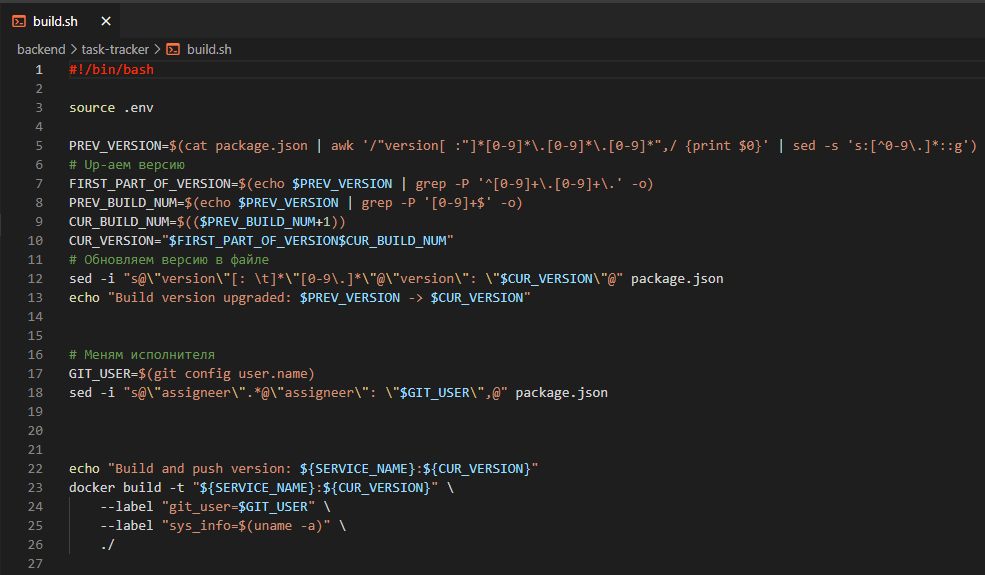


Рисунок х Пример build.sh скрипта.

Данный скрипт вычисляет текущую версию сервиса из package.json файла, инкрементирует номер сборки, добавляет в мета-данные контейнера данные о авторе сборки (на основе гит-данных), и собирает сервис в виде докер-образа.

#### Локальный запуск

Второй способ запуска более простой, но подойдет только в режиме разработки. Для запуска нужно в консоли зайти в каждую директорию, и выполнить команду запуска сервисов, скрипты запуска обычно хранятся в package.json, и они могут отличатся в зависимости от фреймворка, для нашего случая приведу таблицу:

|  |  |
| --- | --- |
| **Сервис** | **Скрипт запуска** |
| vue-spa-vkr | npm run serve |
| task-tracker | npm run start |
| ns-auth-api | npm run start |
| forms | node app.js |
| forms-fetcher | node app.js |

Перед запуском сервисов должна быть настроена база данных.

## 3.2 Руководство пользователя

### 3.2.1 Обзор основного функционала

#### Авторизация

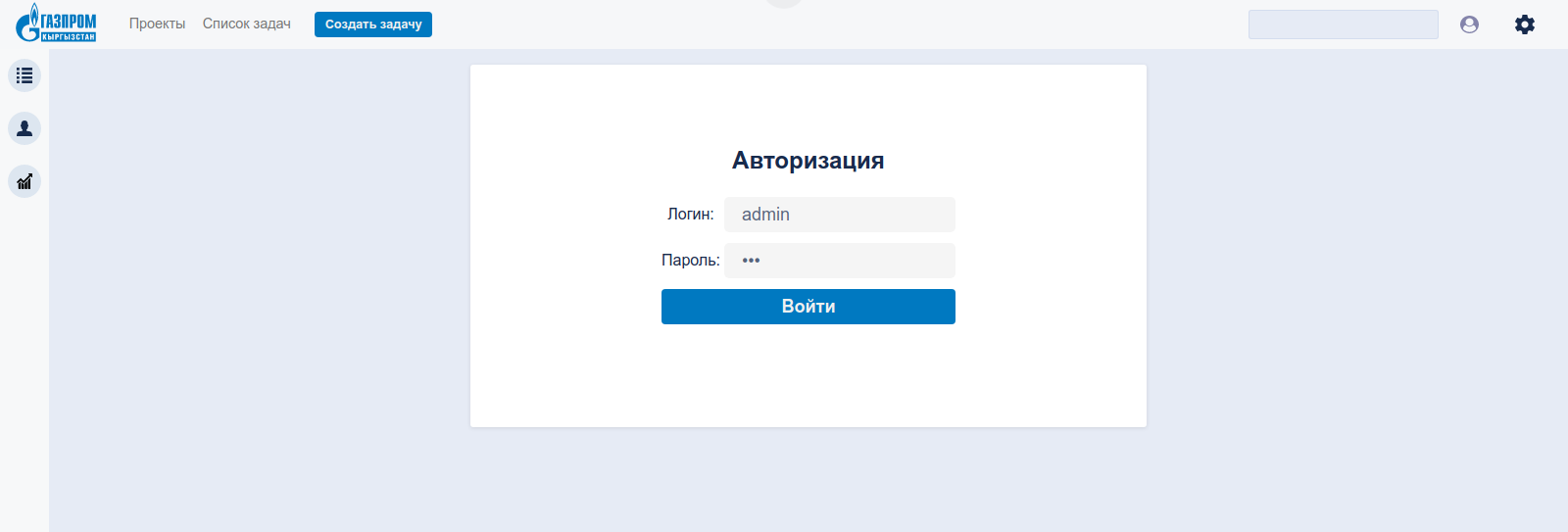


Рисунок х Страница входа в систему учета задач

При входе на домен системы пользователь попадает на страницу входа, система не отобразит данные, пока пользователь не будет авторизован.

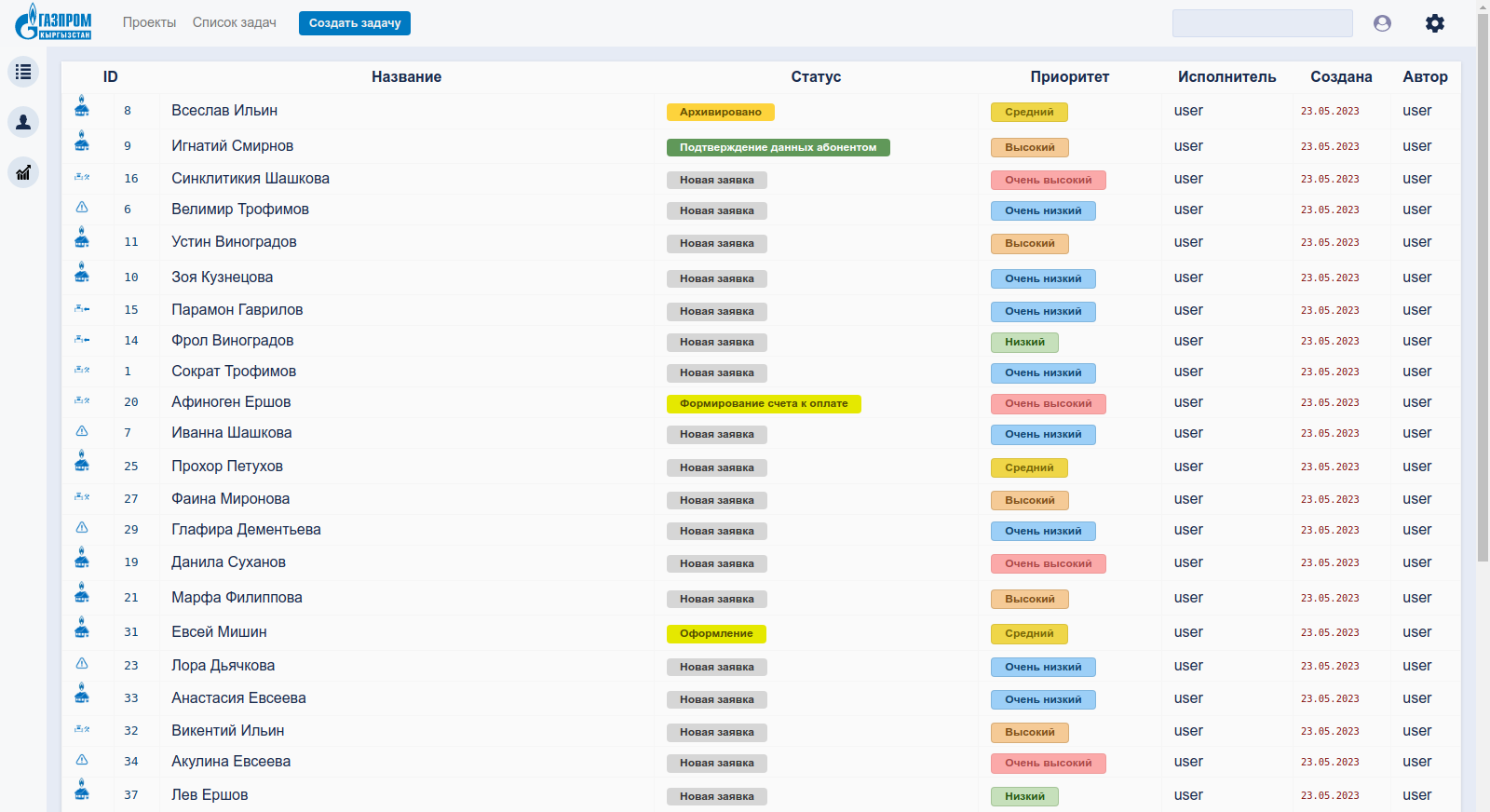


Рисунок х Пользовательский список задач

После успешной авторизации, система перенаправит пользователя на страницу со списком задач.

#### Сортировка задач

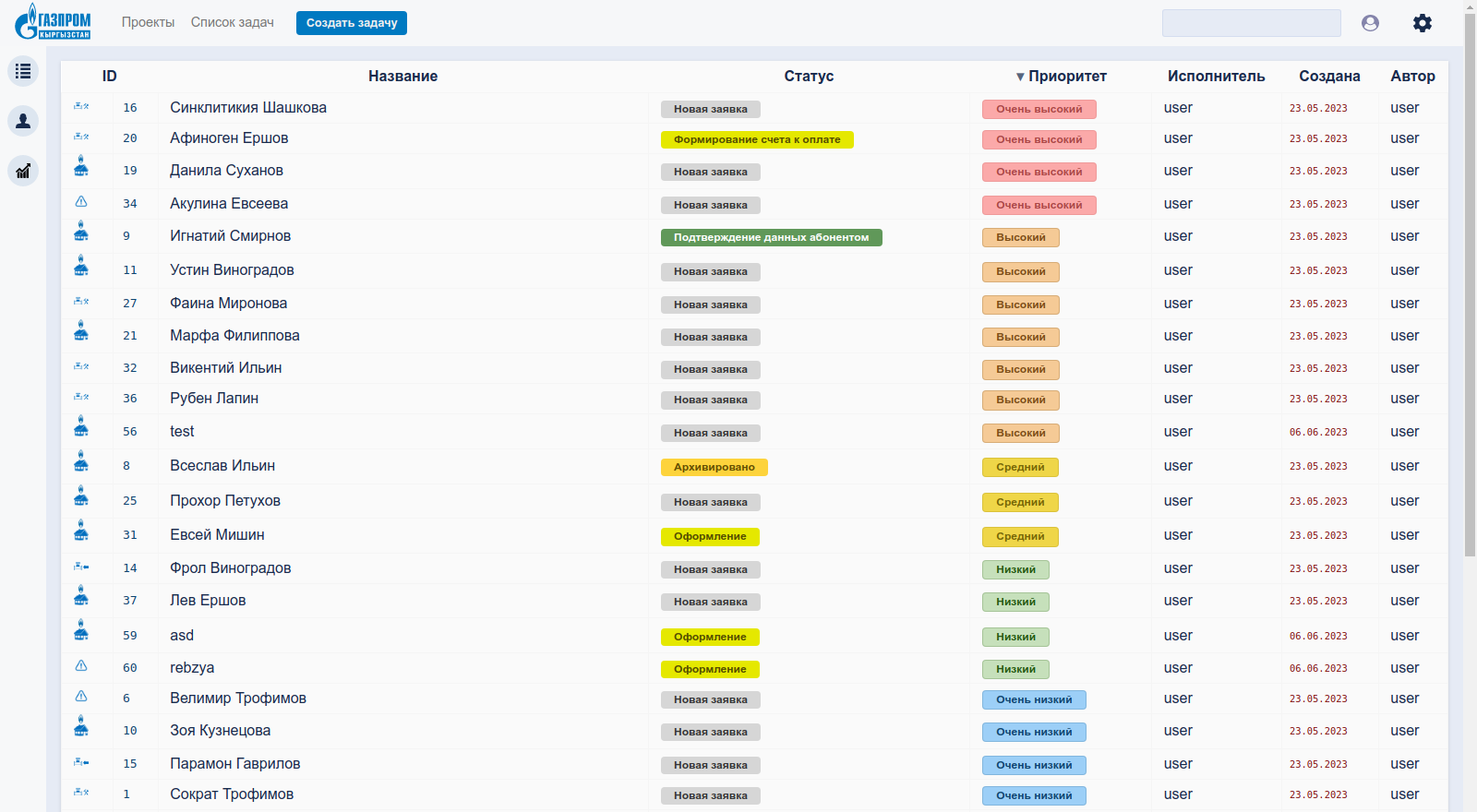


Рисунок х Отсортированный список задач по столбцу - «Приоритет»

При клике на столбец таблицы, задачи будут отсортированы по данному столбцу, при повторном клике в обратном порядке.

#### Детали задачи

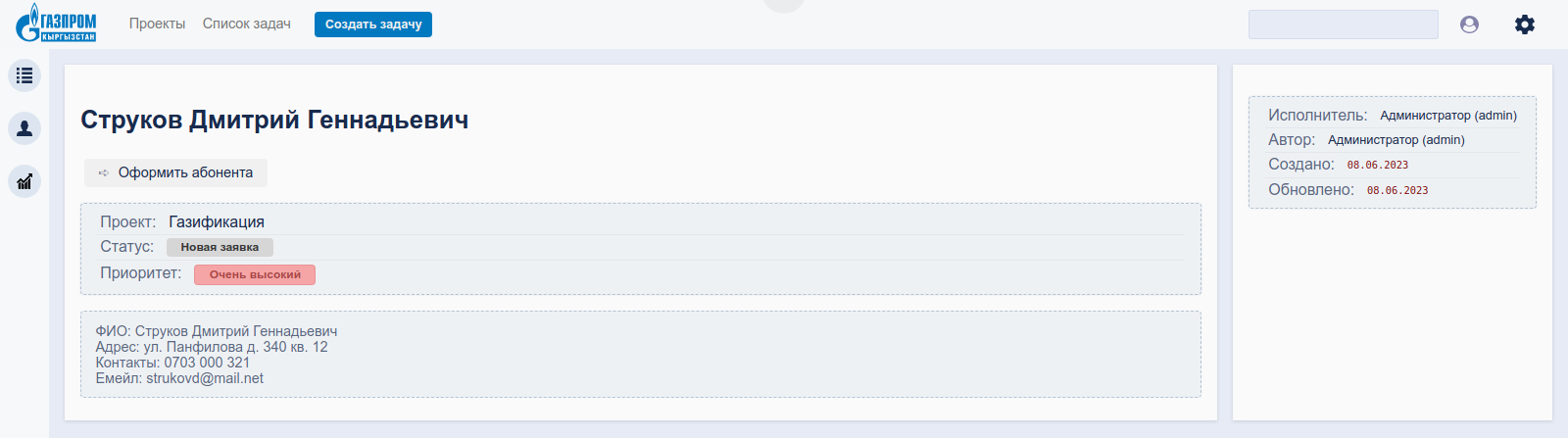


Рисунок х Детали задачи

При клике по задаче (строка таблицы), система отобразит выбранную задачу более детально.

#### Поиск

#### Создание задачи

#### Создание пользователя

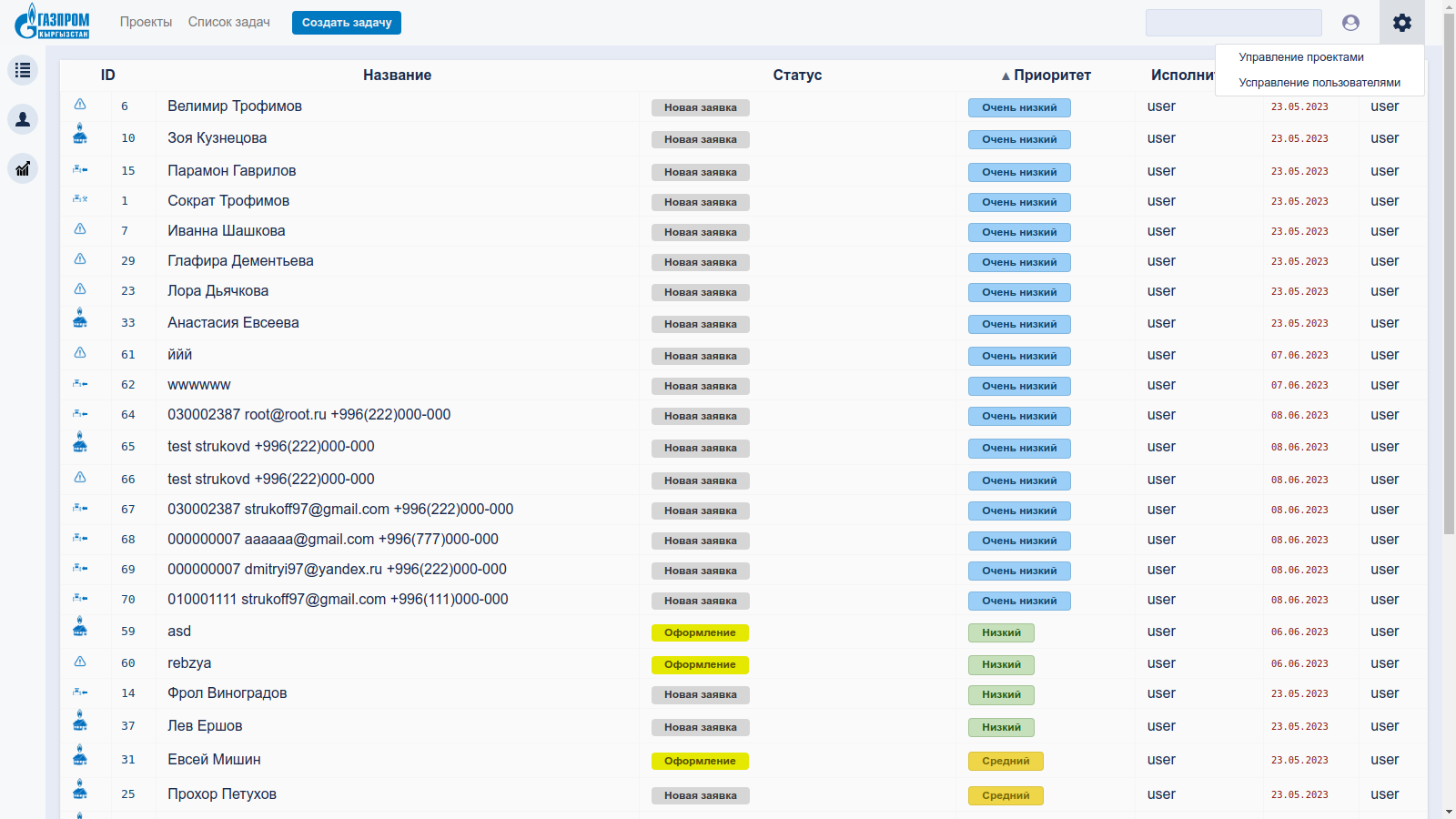


Рисунок х Контекстное меню настроект

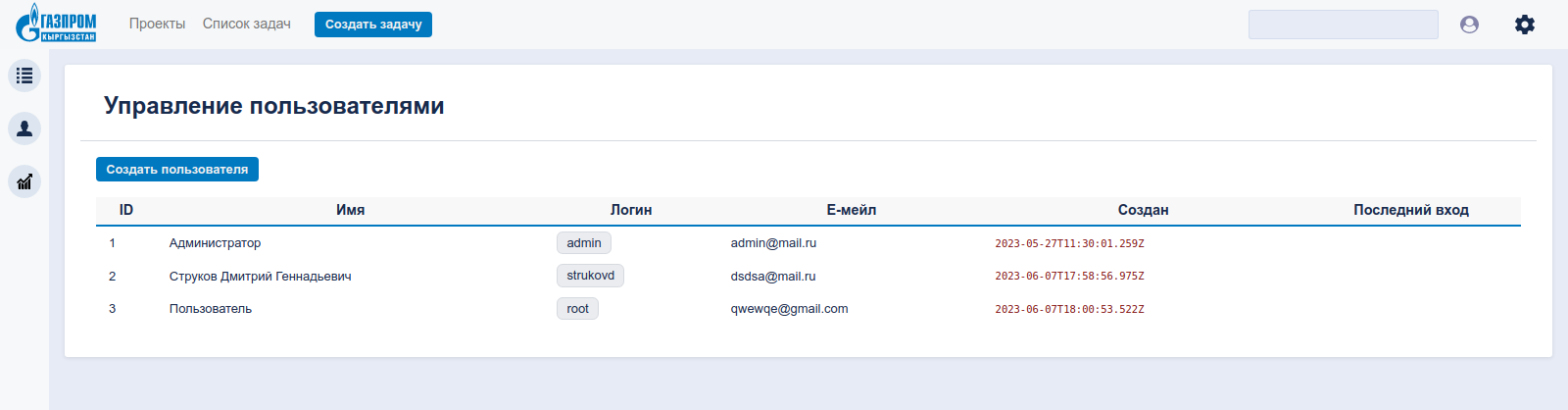


Рисунок х Страница управления пользователями

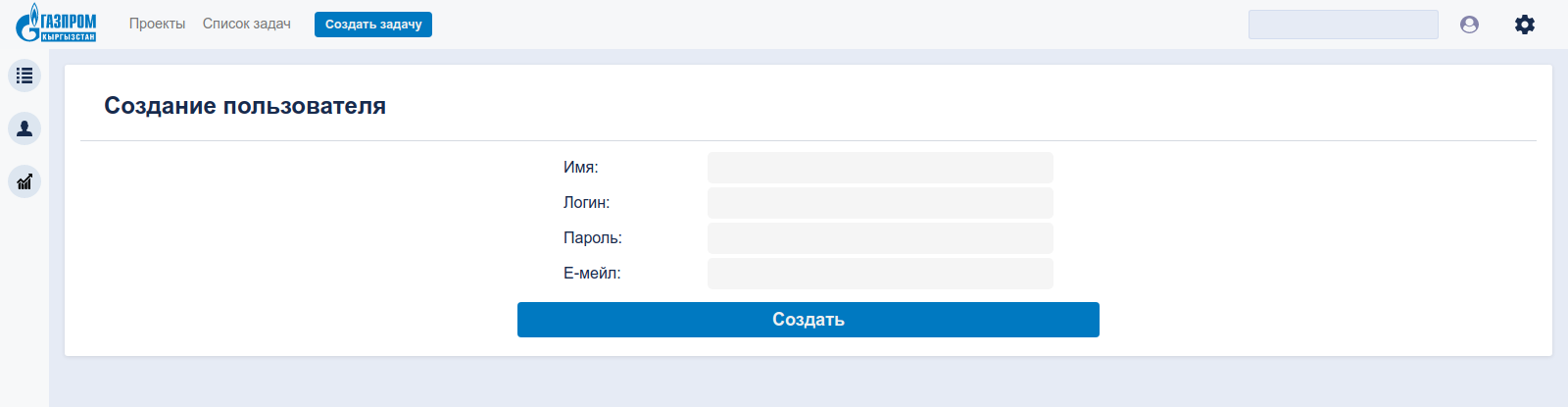


Рисунок х Форма создания нового пользователя

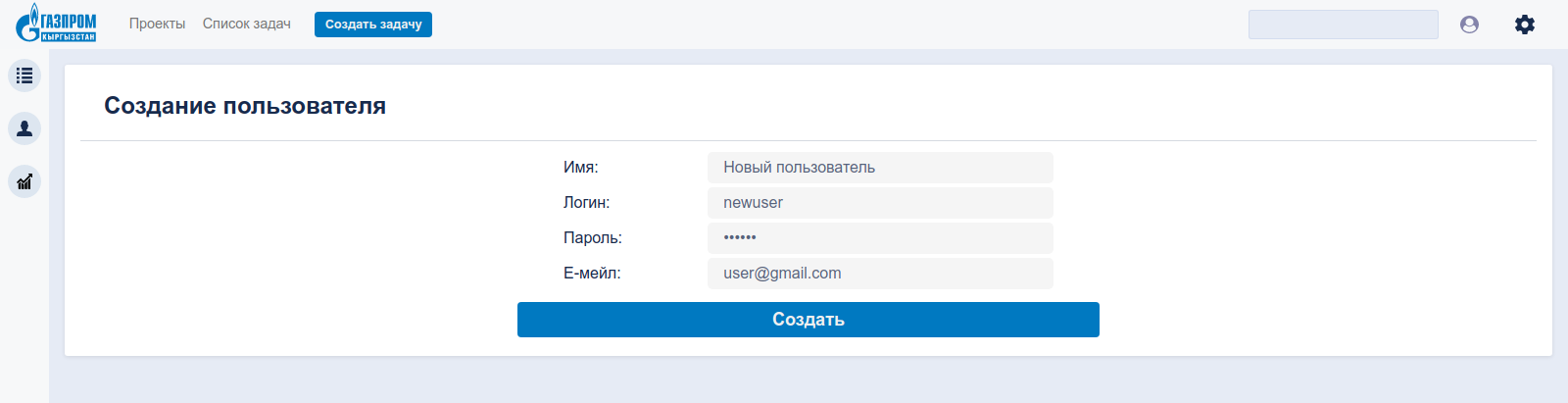


Рисунок х Заполненная форма создания нового пользователя

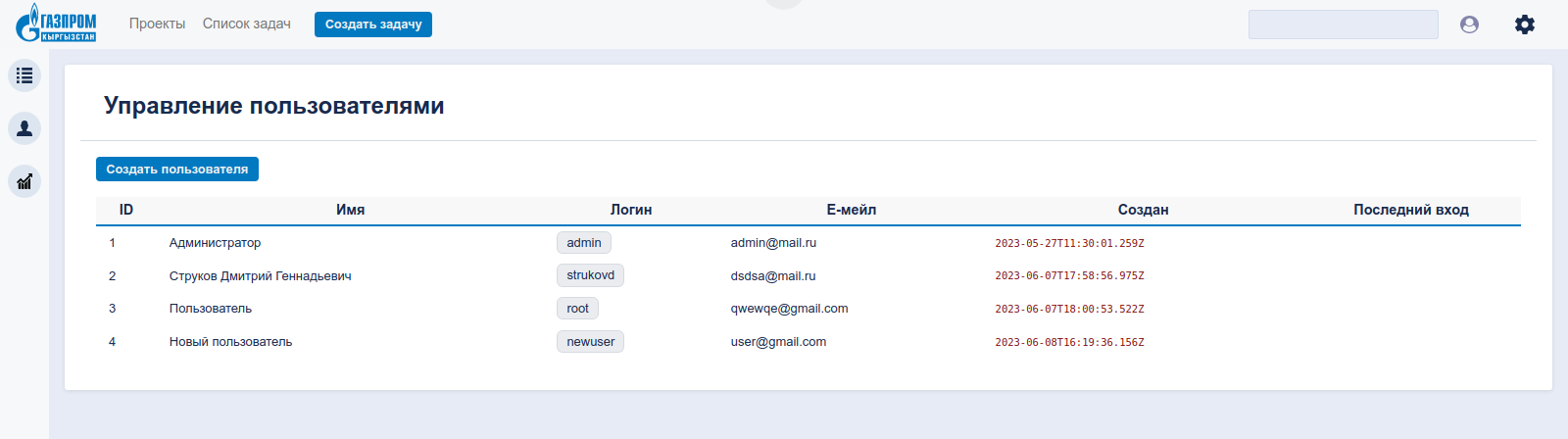


Рисунок х Новый пользователя появился в списке

## 3.3 Тестирование системы

### 3.3.1 Авторизация

#### 3.3.1.1 Не валидная попытка авторизации

3.3.1.2 Валидная попытка авторизации

### 3.3.2 Создание заявки