Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
Образовательное учреждение высшего образования
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(МОСКОВСКИЙ ПОЛИТЕХ)

ДОПУЩЕН К ЗАЩИТЕ

Заведующий кафедрой

_____ / Пухова Е. А. /

Руководитель образовательной программы

_____ / Даньшина М. В. /

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по теме:

АВТОМАТИЗИРОВАННАЯ ПЛАТФОРМА РАЗВЕРТЫВАНИЯ КОНТЕЙНЕРИЗОВАННЫХ ФУНКЦИЙ В СРЕДЕ KUBERNETES

по направлению 09.03.01 Информатика и вычислительная техника
Образовательная программа (профиль) «Веб-технологии»

Студент: _____ / Журавлев Давид Александрович, 211–321/
подпись *ФИО*

Руководитель ВКР: _____ / Гонтовой Сергей Викторович , к.н./
подпись *ФИО, уч. звание и степень*

Москва 2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное
Образовательное учреждение высшего образования
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(МОСКОВСКИЙ ПОЛИТЕХ)

ЗАДАНИЕ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
по направлению 09.03.01 Информатика и вычислительная техника
Образовательная программа (профиль) «Веб-технологии»

Тема ВКР	Автоматизированная платформа развертывания контейнеризованных функций в среде Kubernetes
ПРАКТИЧЕСКИЙ РЕЗУЛЬТАТ	
Назначение	Система предназначения для автоматизации процессов развертывания и управления бессерверными вычислениями в изолированной среде.
Основные функции	1. Регистрация и авторизация пользователей. 2. Предоставление функционала создания групп. 3. Предоставление функционала создания кодов приглашения в группу. 4. Предоставление управления правами членов групп / доступом к запущенным задачам. 5. Развертывание и управление задачами в контейнерах. 6. Настройка выполнения задач по расписанию, на webhook-событию, ручным запуском. 7. Мониторинг выполнения задач в реальном времени. 8. Просмотр логов выполнения задач.
Используемые технологии и платформы	Java 21, Kotlin, Spring, Keycloak, Postgres, Kubernetes, Docker, Vue3, TypeScript, Pinia, TailwindCSS, DaisyUI, Git

ВЫПОЛНЕНИЕ РАБОТЫ	
Решаемые задачи	1. Провести анализ предметной области. 2. Сравнить существующие аналогичные решения. 3. Провести анализ целевой аудитории веб-приложения. 4. Определить функциональные требования к веб-приложению. 5. Разработать пользовательские сценарии. 6. Спроектировать архитектуру веб-приложения. 7. Разработать дизайн-макеты страниц и компонентов веб-приложения. 8. Спроектировать схему базы данных. 9. Разработать серверную часть веб-приложения. 10. Разработать клиентскую часть веб-приложения. 11. Провести различные виды тестирования веб-приложения.
Состав технической документации	1. Техническое задание. 2. Пояснительная записка.
Состав графической части	1. Презентация. 2. Схема организационной структуры: 1 экз. 3. Диаграмма IDEF0 AS-IS: 2 экз. 4. Диаграмма IDEF0 TO-BE: 2 экз. 5. DFD-диаграмма: 1 экз. 6. Алгоритм решения задачи определения уровня тревожности: 1 экз. 7. Схема взаимодействия компонентов веб-приложения: 1 экз. 8. Экраны интерфейса: 20 экз. 9. Примеры структур проекта: 3 экз.

ПЛАН РАБОТЫ НАД ВКР

Этапы	Недели семестра																	
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Провести анализ предметной области.																		
Сравнить существующие аналогичные решения.																		
Провести анализ целевой аудитории веб-приложения.																		
Определить функциональные требования к веб-приложению.																		
Разработать пользовательские сценарии.																		
Спроектировать архитектуру веб-приложения.																		
Разработать дизайн-макеты страниц и компонентов веб-приложения.																		
Спроектировать схему базы данных.																		
Разработать серверную часть веб-приложения.																		
Разработать клиентскую часть веб-приложения.																		
Провести различные виды тестирования веб-приложения.																		

РУКОВОДИТЕЛЬ ОП:
«___»_____2025, _____/ Даньшина Марина Владимировна. _____ /
подпись *ФИО, уч. звание и степень*

РУКОВОДИТЕЛЬ ВКР:
«___»_____2025, _____/ Гонтовой Сергей Викторович _____, к.н./
подпись *ФИО, уч. звание и степень*

СТУДЕНТ:
«___»_____2025, _____/Журавлев Давид Александрович, 211–321/
подпись *ФИО, группа*

АННОТАЦИЯ

Наименование работы: автоматизированная платформа для развертывания и управления лямбда-функциями «Lambda».

Цель работы: разработка и исследование платформы для автоматизированного развертывания и управления вычислительными задачами в контейнерах, обеспечивающей удобное взаимодействие пользователей с инфраструктурой и оптимизацию ресурсов.

Объект исследования: процессы автоматизированного развертывания, исполнения и мониторинга контейнеризированных вычислительных задач.

Предмет исследования: архитектурные, программные и алгоритмические решения, обеспечивающие эффективное управление контейнерами, балансировку нагрузки, обработку задач и анализ их выполнения в распределенной среде.

Объем работы составляет X страниц. Работа включает в себя 3 главы, X рисунков, X таблиц, X листингов кода. Библиография включает X источников.

Во введении раскрываются актуальность темы и практическая значимость разработки.

Первая глава посвящена анализу предметной области, формулировке целей и задач исследования, рассмотрению существующих аналогов.

Во второй главе пределяются функциональные возможности платформы и выибается технологический стек, описываются архитектура системы, пользовательский интерфейс и технические аспекты реализации.

Третья глава включает процесс тестирования, развертывания платформы, анализ её масштабируемости и возможных путей развития.

В заключении подводятся итоги работы, оценивается её практическая ценность и формулируются основные выводы.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире информационных технологий все больше задач требует выполнения в автоматизированном и распределенном режиме. Малые и средние компании, исследовательские группы и отдельные разработчики сталкиваются с необходимостью периодически запускать нерегулярные ресурсоемкие вычислительные процессы, такие как обработка данных, генерация отчетов, обучение моделей машинного обучения и других ресурсоемких операций. Однако традиционные серверные решения требуют сложной настройки, постоянного администрирования и значительных финансовых вложений, что делает их использование не всегда целесообразным.

Существующие облачные платформы, большинство из которых не представлены на Рооссийском рынке, такие как AWS Lambda, Google Cloud Functions и Azure Functions, предлагают serverless-вычисления, позволяя запускать код по требованию без необходимости управления серверами. Однако эти решения ориентированы в первую очередь на крупный бизнес и имеют ограничения по конфигурации окружения, стоимости вызовов и способам взаимодействия с системой. Кроме того, использование этих сервисов в частных облаках или внутрикорпоративных инфраструктурах часто оказывается невозможным. В качестве альтернативы некоторые компании развертывают собственные решения на базе Kubernetes, но это требует значительных инженерных ресурсов и опыта работы с контейнеризацией.

В данной работе рассматривается разработка платформы Lambda, предназначенной для автоматизированного развертывания и управления контейнеризированными вычислениями. Платформа позволяет пользователям загружать и исполнять задачи в виде Docker-контейнеров с возможностью настройки параметров запуска, ограничений по ресурсам, управления окружением и интеграции с другими системами.

Предлагаемая платформа ориентирована на малые и средние компании, которым требуется удобное и экономически эффективное решение для выполнения вычислений без необходимости разворачивания и поддержки сложных серверных инфраструктур.

Таким образом, разработка платформы Lyambda является актуальной задачей, способной упростить доступ к автоматизированным вычислениям и снизить затраты на поддержку серверных мощностей.

1. ПРЕДМЕТНАЯ ОБЛАСТЬ

1.1. Предметная область разработки бессерверных приложений

Бессерверные архитектуры представляют собой облачную модель исполнения программного кода, при которой используется модель запуска серверных вычислений «Backend as a service» (BaaS). [1]

Бессерверность - это общая технология, позволяющая не связываться с базовой инфраструктурой. Двумя областями применения бессерверной технологии являются серверная часть как услуга (BaaS) и Функция как услуга (FaaS) или серверная часть как услуга и функция как услуга соответственно.

BaaS (Серверная часть как услуга) — это облачное решение, предоставляющее готовую инфраструктуру для серверных приложений. Оно включает в себя набор сервисов, таких как аутентификация пользователей, база данных, файловое хранилище, push-уведомления и другие стандартные серверные компоненты. BaaS позволяет разработчикам сосредоточиться на логике приложения, не беспокоясь о развертывании и обслуживании серверной части. Примеры BaaS: Firebase, Parse, Back4App.

FaaS (Функция как услуга) - это вычислительная модель, в которой разработчик записывает отдельные функции, которые выполняются по запросу (например, при запуске события). Эти функции выполняются в ответ на такие события, как HTTP-запросы или изменения в базе данных, и масштабируются автоматически. В отличие от BaaS, который предоставляет более широкий спектр серверных сервисов, FaaS фокусируется исключительно на выполнении кода. Примеры FaaS: AWS Lambda, облачные функции Google, функции Azure. [2]

Применение бессерверных вычислений, в отличие от традиционных серверных, позволяет не заботиться о даступности и масштбировании инфорастуктуры, а так же снижать издержки возникающие из-за простоев оборудования в периоды малой нагрузки. [3]

Несмотря явное указание отстуствия серверов при применении бессерверных вычилений, сервера при использовании данного подхода импользуются, но управление ими передается облачному провайдеру.

Изначально термин «Serverless» использовался в контексте rich-client [4] приложений, одностраничные веб-приложения (SPA) [5] или мобильных приложений, отличительной чертой которых является использование обширной экосистемы доступных в облаке данных, таких как удаленные базы данных (например Firebase), службы аутентификации и так далее.

К serverless приложениям также относятся приложения, в которых серверная логика по-прежнему реализуется разработчиком приложения, но, в отличие от традиционных архитектур, она выполняется в вычислительных контейнерах без сохранения состояния, которые запускаются по событиям. Такие контейнеры недолговечны, полностью управляются третьей стороной, так же существуют только в рамках одного вызова.

Один из способов реализовать такой подход это - “Function as a service” или “FaaS”. AWS Lambda в настоящее время является одной из самых популярных реализаций платформы "Function as a service но существует также множество других.

Несмотря на все преимущества которые дает применение бессерверных вычилений, необходимо учитывать особенности разработки под бессерверные системы и знать о налогах которые накладывает применение бессерверных вычислений.

Наиболее важной может оказаться пролема холодного запуска [6]. При масшатбировании до 0 при получении запроса на выполнение функции значительное время может быть затрачено на первоначальный запуск конейнера с необходимым сервисом [7].

1.2. Анализ аналогов и конкурентов

Первое появление концепции бессерверных вычислений связано с AWS Lambda, запущенной в 2014 году Amazon Web Services. Это решение позволило разработчикам загружать небольшие функции, которые выполнялись в ответ на такие события, как HTTP-запросы, изменения в базе данных или сообщения в очереди.

После успеха AWS Lambda, другие крупные облачные провайдеры представили свои serverless-решения:

— 2016г. — Google Cloud Functions;

— 2016г. — Microsoft Azure Functions;

— 2017г. — IBM Cloud Functions;

— 2019г. — Google Cloud Run.

Ни один из крупнейших поставщиков Serverless решений на данный момент не доступен в России.

В России на данной момент доступны поставщики serverless инфраструктуры:

— Yandex Cloud;

— Cloud.ru

С развитием контейнерных технологий появилась возможность объединять Serverless и Container-based подходы, что позволило избежать жестких ограничений FaaS.

Для возможности плавной смены поставщика serverless инфораструктуры существуют open-source фреймворки позволяющие разрабатывать решения одинаково работающие на инфраструктуре различных поставщиков, реализующих поддержку данных решений. К таким решениям можно отнести:

— Apache Openwhisk https://openwhisk.apache.org/

— Fission https://fission.io/

— Fn https://fnproject.io/

— Knative https://knative.dev/docs/

— Kubeless https://github.com/vmware-archive/kubeless

— Nuclio https://nuclio.io/

— OpenFaaS https://www.OpenFaaS.com/

Рассмотрим основных игроков на рынке serverless решений.

1.2.1. AWS Lambda

AWS Lambda является флагманом подхода «Function as a Code», одним из первых и самых популярных решений для запуска облачных функций. Сервисы работает не с контейнерами, а напрямую с програмным кодом, что накладывает ограничения на стек технологий которые могут быть использованы при интеграции с данным сервисом.

К плюсам системы AWS Lambda можно отнести:

— Полная интеграция с AWS-экосистемой (S3, DynamoDB, API Gateway).

— Автоматическое масштабирование.

— Поддержка множества языков (Python, Node.js, Java, Go и др.).

— Гибкая система триггеров (HTTP, очередь сообщений, события облака).

Минусами системы являются:

— Ограничение времени выполнения (15 минут).

— Задержки из-за холодных стартов.

— Высокая стоимость при интенсивных нагрузках.

— Недоступность в России.

1.2.2. Google Cloud Run

Google Cloud Run, продукт разарботанный компанией Google, является представителем класса BaaS систем которые используют в качестве единицы развертывания контейнер. В отличии от систем использующих в качестве единицы развертывания код, подход с конетейнерами не накладывает никаких ограничений на стек используемых технологий и сложность разворачиваемого сервиса. Единственным требованием к запускамому юниту является соответствие спецификации Open Container Initiative (OCI) [8].

К плюсам системы Google Cloud Run можно отнести:

— Поддержка любых языков и сред (т.к. работает с контейнерами).

— Простота развертывания (автоматически масштабируемые контейнеры).

— Гибкость в выборе окружения.

Минусами системы являются:

— Более высокая цена по сравнению с FaaS-решениями.

— Холодные старты при отсутствии постоянной нагрузки.

— Сильная интеграция с Google Cloud (ограниченная независимость).

— Недоступность в России.

1.2.3. OpenFaaS, OpenWhisk, Knative и т.д.

OpenFaaS, OpenWhisk, Knative являются популярными open-source [9] решениями для развертиывания локальных FaaS сервисов. В отличие от проприетарных облачных FaaS-платформ такие решения, при использовании для управления локальной инраструктурой позволяют избежать ограничений на время выполнения и более точно пронозировать и управлять нагрузкой. Некоторые из представленных на рынке систем могут развернуты на любых можностях: Kubernetes, Docker Swarm, локальные сервера. Все представленные сисемы работаю с контейнерами как с единицей развертывания, а значит так же накладывают ограничение соответствия стандарту OCI.

К плюсам open-source систем разворачиваемым локально можно отнести:

— Полный контроль над инфраструктурой.

— Оптимизация и прогнозируемость затрат.

— Технологическая независимость.

Минусами систем являются:

— Необходимость наличия команды поддержки и технических компетенций.

— Потребность в ресурсах.

— Необходимость самостоятельной настройки сервисов.

— Отсутствие поддержки вендора.

1.2.4. Аналоги

Аналогом же разрабатываемого решения будет являться использования классических решений размещения приложений на собственных или арендованных мощностях, таких как виртульные машины, выделенные серверы, управляемые Kubernetes или Docker Swarm кластеры.

Использование традиционных решений для размещения приложений на собственных или арендованных мощностях, таких как виртуальные машины (VM), выделенные серверы, а также управляемые кластеры Kubernetes или Docker Swarm, имеет свои преимущества и недостатки.

Преимущества:

— Полный контроль над инфраструктурой.

— Гибкость в выборе технологий.

— Предсказуемость затрат.

Недостатки:

— Необходимость наличия команды поддержки и технических компетенций.

— Потребность в ресурсах.

— Необходимость самостоятельной настройки сервисов.

— Высокие капитальные затраты при простоях вычислительных мощностей.

— Сложности с масштабированием.

— Ограниченная гибкость.

1.2.5. Гибридный подход к управлению инфраструктурой

В следствие наличия плюсов и минусов у каждого подхода к управлению инфраструктурой, большинству компаний подойдет гибридный подход, объединяющий локальные ресурсы с облачными сервисами, позволяет достичь

баланса между контролем над инфраструктурой, гибкостью в масштабировании и эффективным управлением затратами.

При исопльзовании гибридного подхода сервисы испытывающие постоянную нагрузку, а так же сервисы время отклика которых критично для операций которые они выполняют не должны быть развернуты с использованием классических подходов усправления доставкой и развертыванием. В это же время использование FaaS сервисов для функционала не не находящегося под постоянной нагрузкой и не имеющего строгих трбований к времени ответа, может помоч оптимизировать расходы на серверную инфораструктуру и простои оборудоания.

1.3. Анализ целевой аудитории

1.3.1. Сегментация целевой аудитории

На начальных этапах разработки платформы целевой аудиторией разрабатываемой системы являются стартапы, разработчики, малые и средние кампании, а так же образовательные учереждения, у которых есть потребность в выполнении задач, лежащих в прикладной области платформы.

Важным аспектом целевой аудитории является отсутствие собсевнной сервеной инфраструктуры или отсутствие возможности избежать ее простоев. Целевая аудитория системы не имеет возможности и компетенций что бы использовать существующие open-source решения, такие как OpenFaaS, OpenWhisk, Knative на мощностях собсевнной инфраструктуры.

Для целевой аудитории целью использования платформы является автоматизация и отпимизация процессов компании, выполнение требовательных вычислений, лежащих в области деятельности комании.

1.3.2. Анализ потребностей

Исходя из выявленных сегментов целевой аудитории, ключевые потрбности можно условно разделить на несколько направлений:

— компенсания отстуствия собсевнной инфораструктуры;

— оптимизация затрат на развертывание и эксплуатацию;

— выполнение ресурсоемких вычислений.

1.3.3. Выводы и рекомендации для разработки

На основании проведенного анализа целевой аудитории, можно сделать следующие выводы:

— предавители сегментов целевой аудитории нуждаются в локализованном, легком в управлении решениии для запуска бессерверных вычислений, которое позволит избежать зависимости от зарубежных вендоров;

— существующие open-source решения не являются прямыми конкурентами, но поддержка спецификаций, позволяющих совместимость с другими существующими решениями будет конкурентным преимуществом;

1.4. Проблема, цель и задачи разработки

Современные решения задач в области выполнения бессерверных вычислений в настоящее время сопряжены с радом проблем и компромисов, таких как недоступность крупных FaaS сервисов в России, органичения по формату, времени выполнения, нагрузке выполняемых функций.

Разрабаотываемая плаотаорма автоматизации управления бессерверными вычислениями призвана устранить часть этих барьеров, предоставив удобное и эффективное решение для выполнения задач выполнения бессерверных вычислений различных типов.

Целью разработки является создание платформы управления бессерверными вычислениями, обеспечивающую балланс между простотой испольования, гибкостью конфигурации и предстказуемостью затрат.

Для достижения целей разработки необходимо выполнить следующие задачи:

— провести анализ предметной области;

— сравнить существующие аналогичные решения;

— провести анализ целевой аудитории веб-приложения;

— определить функциональные требования к веб-приложению;

— разработать пользовательские сценарии;

— спроектировать архитектуру веб-приложения;

— разработать дизайн-макеты страниц и компонентов веб-приложения;

— спроектировать схему базы данных;

— разработать серверную часть веб-приложения;

— разработать клиентскую часть веб-приложения;

— провести различные виды тестирования веб-приложения.

1.5. Требования к разрабатываемой системе

Данный раздел описывает ключевые требования к проектируемой системе. Эти требования определяют ожидаемую функциональность и качественные характеристики системы, служа основой для дальнейшего проектирования и реализации.

1.5.1. Функциональные требования

Платформа должна предоставлять пользователям необходимые инструменты для работы с учетными записями и вычислительными задачами. Пользователи должны иметь возможность самостоятельно регистрироваться в системе, входить под своей учетной записью и восстанавливать доступ при утере пароля. Также необходим просмотр базовой информации своего профиля и членства в группах.

Центральной функцией системы является управление контейнерами с исполяемыми приложениями. Пользователи должны определять параметры задачи: имя, образ среды выполнения, команду запуска и переменные окружения. Система должна поддерживать разные типы запуска: однократный, периодический по расписанию, с возможностью настройки этого расписания или по внешнему сетевому вызову. Задачу можно создать, как в пространстве пользователя, так и при необходимости связать с группой для совместной работы.

Для контроля за задачами пользователи должны иметь возможность просматривать список доступных им задач и их текущее состояние, также получать детальную информацию о конфигурации и истории запусков конкретной задачи. Важно обеспечить доступ к результатам выполнения каждого запуска, включая статус, логи и код завершения. Платформа должна позволять перезапускать существующие задачи, отменять активные запуски и удалять задачи, прекращая их дальнейшее выполнение. Для webhook-задач требуется механизм их активации по уникальному URL.

Система должна поддерживать создание групп для совместного использования ресурсов и управления задачами. Пользователи должны иметь возможность присоединяться к группам и покидать их. Администраторы групп должны иметь функционал исключения участников и управления правами доступа внутри группы (просмотр, редактирование, запуск, администрирование). Все участники могут просматривать состав группы и права доступа, а администраторы — управлять процессом приглашения.

Необходимо реализовать логирование ключевых событий и ошибок для диагностики и анализа, а также предоставлять пользователям доступ к логам их задач.

1.5.2. Нефункциональные требования

Для безопасного взаимодействия с ситемой, должна быть разработана система аутентификации пользователей и контроля доступа на основе ролей. Среды выполнения пользовательских задач должны быть надежно изолированы друг от друга и от системных компонентов. Необходимо обеспечить защиту чувствительных данных при хранении и передаче информации, а также защиту от стандартных сетевых угроз.

Производительность системы должна обеспечивать быстрый отклик интерфейсов при ожидаемой нагрузке и эффективное использование вычислительных ресурсов. Масштабируемость архитектуры важна для обработки растущего числа пользователей и задач путем добавления ресурсов.

Удобство использования системы достигается через интуитивно понятный пользовательский интерфейс и логичный, хорошо документированный API.

Поддерживаемость и расширяемость должны обеспечиваться качествен-

ной архитектурой и кодом, что упрощает внесение изменений и добавление функционала.

1.5.3. Требования к данным

Система оперирует несколькими основными категориями данных, которые необходимо корректно хранить и обрабатывать. К ним относятся данные пользователей, данные групп, данные задач и данные результатов выполнения. Также используются справочные данные. Важно обеспечить логическую целостность и непротиворечивость этих данных в хранилище.

1.5.4. Требования к интерфейсам

Основным интерфейсом для пользователей является графический пользовательский интерфейс, представляющий веб-приложение. Для автоматизации и интеграции предоставляется программный интерфейс (API), построенный на стандартных веб-технологиях.

Предоставляемые интерфейсы должны быть спроектированы с учетом удобства пользователя (UI/UX). Графический интерфейс должен быть интуитивно понятным, логически структурированным и визуально последовательным. Программный интерфейс (API) должен быть четко определен, логичен и удобен для интеграции, следуя современным практикам проектирования API.

1.6. Вывод

В первой главе дипломной работы был проведен анализ предметной области управления бессерверными вычислениями. Были рассмыотрены существующие подходыы к кправлению вычислениюю, проанализированы существующие крупные решения, их преимущества и ограницения.

В ходе проведения анализа было вявлено что существующие решения в обласи FaaS и BaaS накладывают существенные ограничения на пользователей, делая неаозможными выполнение большого спектра задач.

Был проведен анализ существующих техничеких решений с открытым исходным кодом позволяющих добиться схожих возможностей управления бессерверными вычислениями на инфраструктуре пользователя. Было выяв-

лено что такие решения требуют от пользователя повышенных затрат и компетенций для поддержания собственной инфраструктуры.

Целевая аудитория была разделена на неасколько сегментов, в том числе малые и средние предприятия, образовательные учереждения. Анализ потребностей, рыночных трендов и конкурентного окружения позволил выделить ключевые требования целевой аудитории. На основе проведенного анализа сформулированы выводы и рекомендации к разработке.

В результате была сформирована цель разработки и сформулированы задачи необходимые для ее выполнения, а так же описаны функциональные и нефункциональные требования к разрабатываемой системе.

2. ТЕХНИЧЕСКАЯ РЕАЛИЗАЦИЯ

2.1. Описание бизнес-процессов

Главные процессы связаны с выполнением вычислительных задач и развертыванием контейнеров, что требует ручной настройки серверной инфраструктуры и управления вычислительными ресурсами. Это обеспечивает выполнение клиентских запросов, но сопровождается значительными временными затратами и сложностями администрирования. На рисунке 2.1 показан сам бизнес-процесс выполнения вычислительных задач и развертывания контейнеров до автоматизации, а на рисунке 2.2 представлена декомпозиция данного процесса.

Среди основных проблем неавтоматизированного бизнес-процесса можно выделить:

— получение запросов от клиентов через ручные каналы связи, что увеличивает время на их обработку и уточнение технических требований;

— ручное конфигурирование серверной инфраструктуры под каждую задачу, что требует значительных временных и трудовых затрат от администраторов;

— отсутствие централизованной системы мониторинга выполнения задач, что приводит к необходимости постоянного ручного контроля и увеличению вероятности ошибок;

— передача результатов клиентам осуществляется вручную, что замедляет процесс и увеличивает вероятность задержек.

После автоматизации значительно минимизируется участие специалистов в ручной настройке серверной инфраструктуры и мониторинге выполнения задач, что упрощает и ускоряет процесс развертывания и выполнения контейнеров. На рисунке 2.3 показан бизнес-процесс выполнения лямбда-функций после автоматизации, а на рисунке 2.4 представлена декомпозиция данного процесса.

Основными преимуществами автоматизированного бизнес-процесса являются:

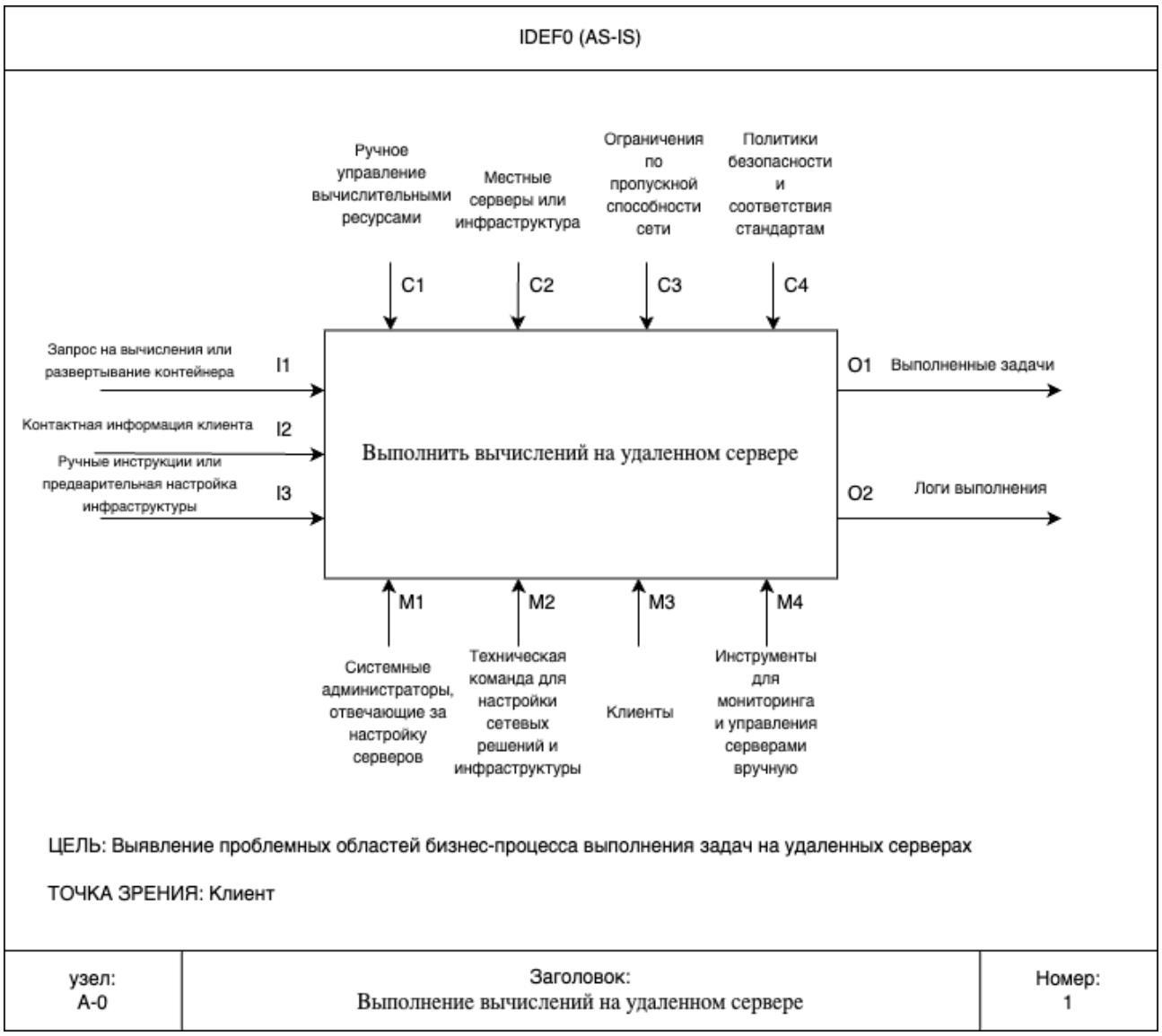

Рис. 2.1. Процесс выполнения вычислений на удаленном сервере

— автоматическое развертывание контейнеров с минимальным участием специалистов;

— гибкость настройки выполнения задач благодаря удобному интерфейсу и API;

— автоматическое формирование отчетов по выполнению задач и сбор статистики о загрузке ресурсов;

— возможность одновременного выполнения множества задач с оптимизацией использования инфраструктуры.

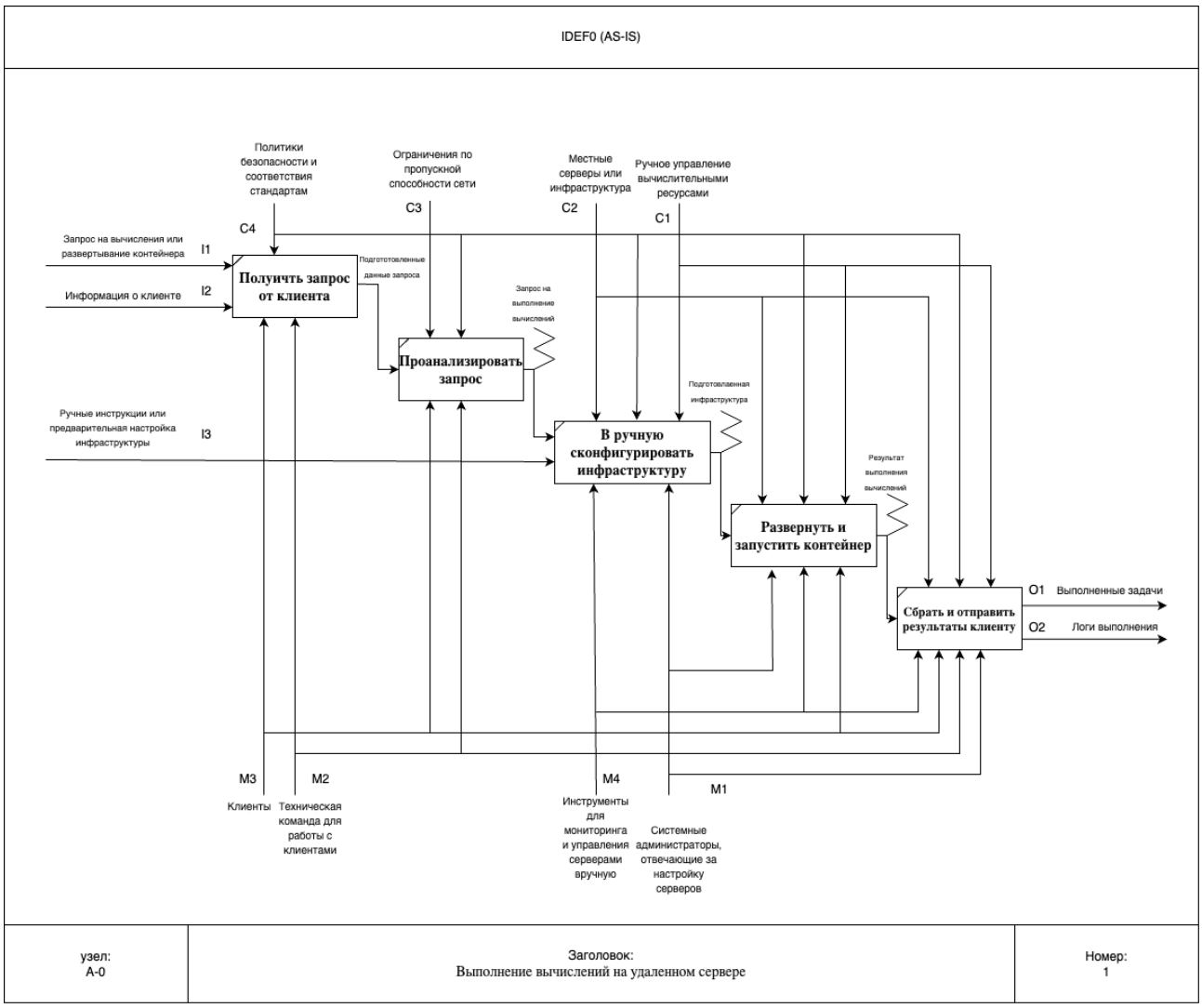

Рис. 2.2. Декомпозиция процесса выполения удаленных вычислений

2.2. Описание архитектуры решения

Описание архитектуры приложения является ключевым этапом в проектировании програмного решения. Описанная архитектура позволяет разделить систему на независимые компоненты, определить взаимосвязи между ними, правила их мастабирования а так же обеспечить согласованность в их разработке.

Разрабатываемое приложение представляет собой моногокомпонентное клиент-серверное решение и стоит незавыисимых функциональных блоков:

— Серверверная часть приложения

— Клиентская часть приложения

Серверная часть в свою очередь состоит из таких компонентов как:

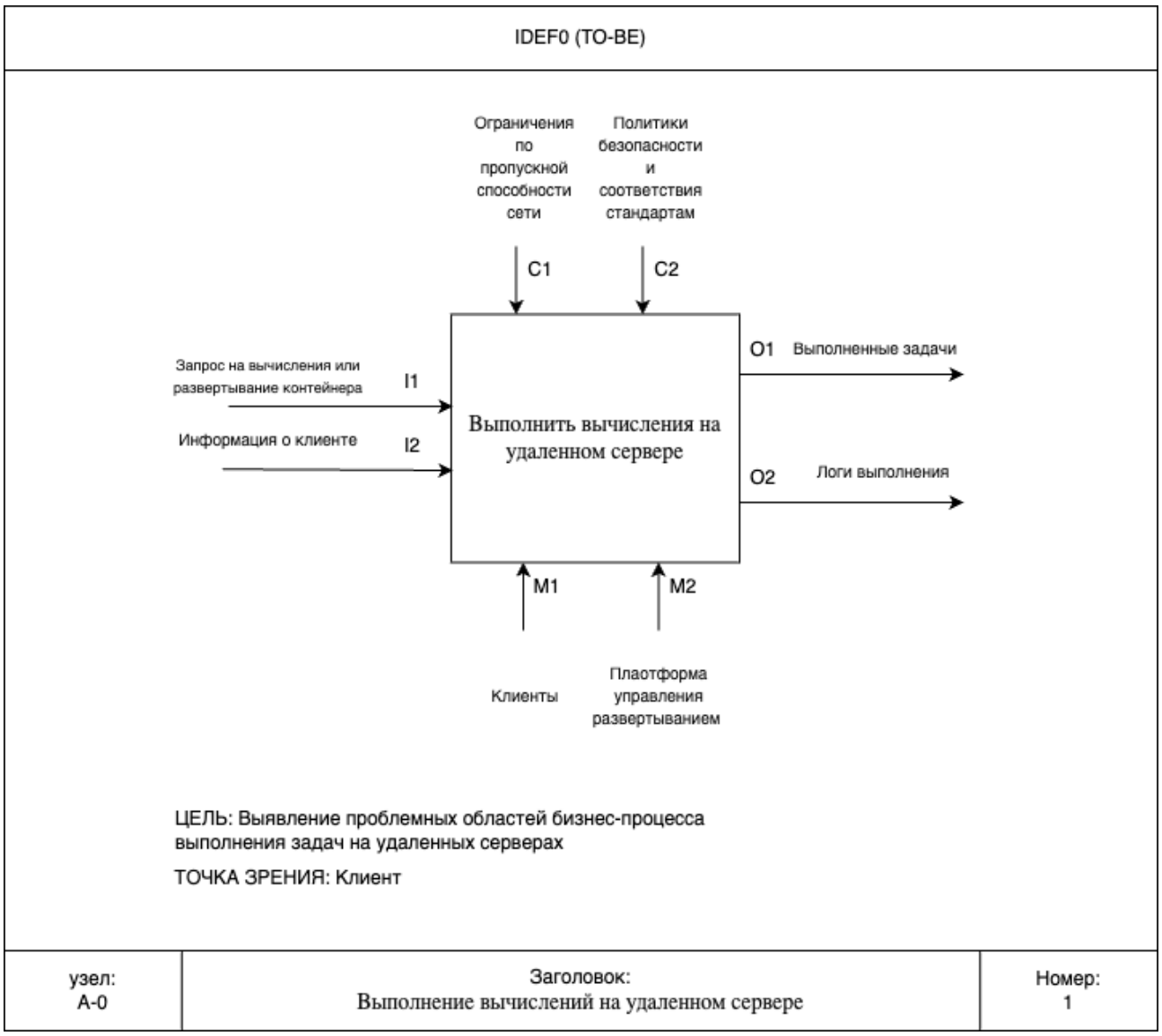

Рис. 2.3. Процесс выполнения вычислений в бессерверной среде

— Reverse proxy [10];

— Internal API Gateway

— External API Gateway

— Сревер авторизации

— Брокер сообщений

— СУБД

— Слой платформенных сервисов

— Подчиненный кластер Kubernetes

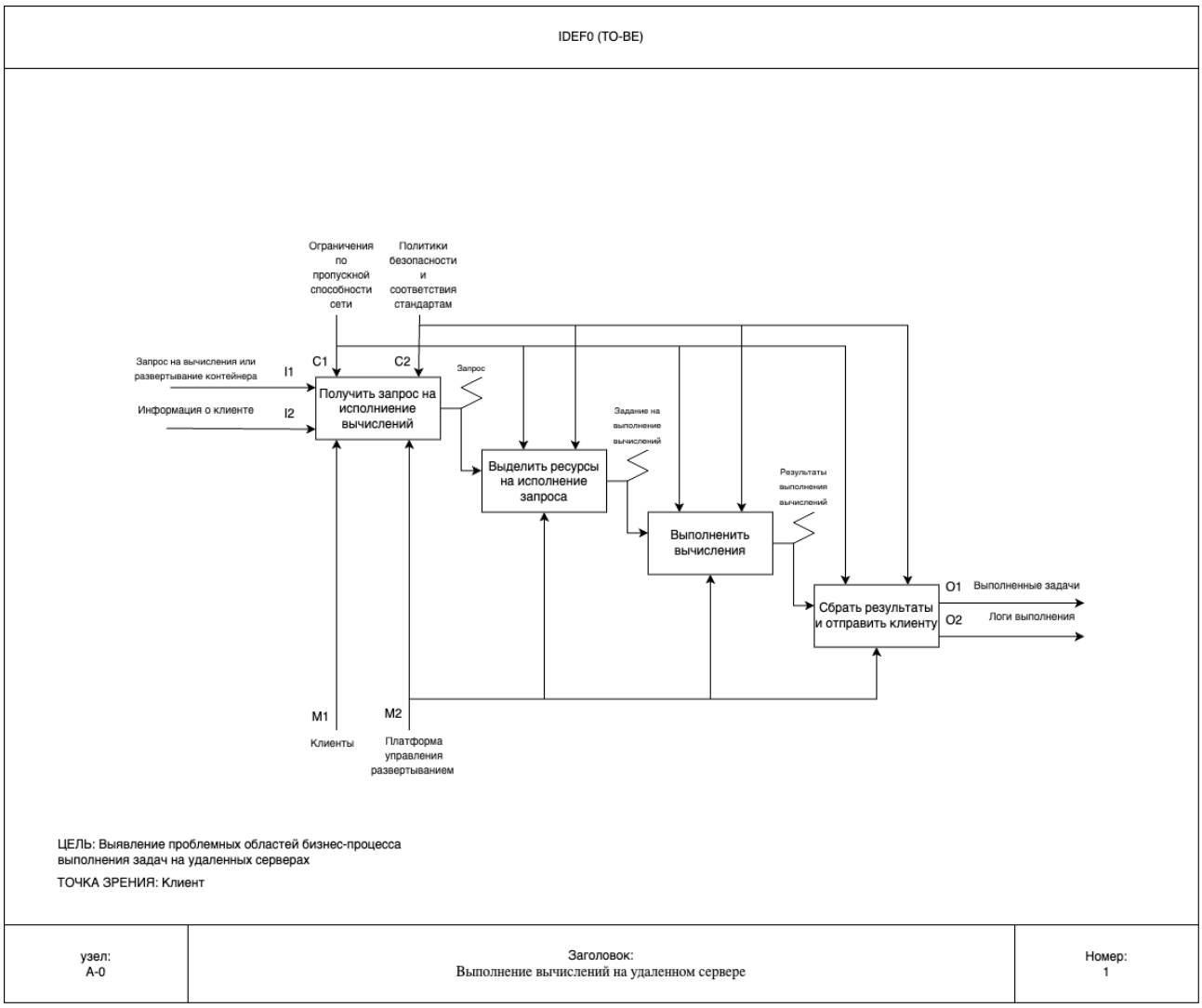

Рис. 2.4. Декомпозиция процесса выполения вычислений в бессерверверной среде

На рисунке 2.5 представлена схема взаимодействия и размещения компонентов системы.

2.2.1. Среда развертывания

Сейчас на рынке существует несколько подходов к развертыванию сервисов. Каждая из представленных сред имеет свои плюсы и минусы.

Физические серверы

В настоящее время развертывание веб-сервисов на физических серверах является устаревшим подходом. Такой подход имеет большое число минусов, таких как необходимость управления зависимостями разворачиваемых серво-

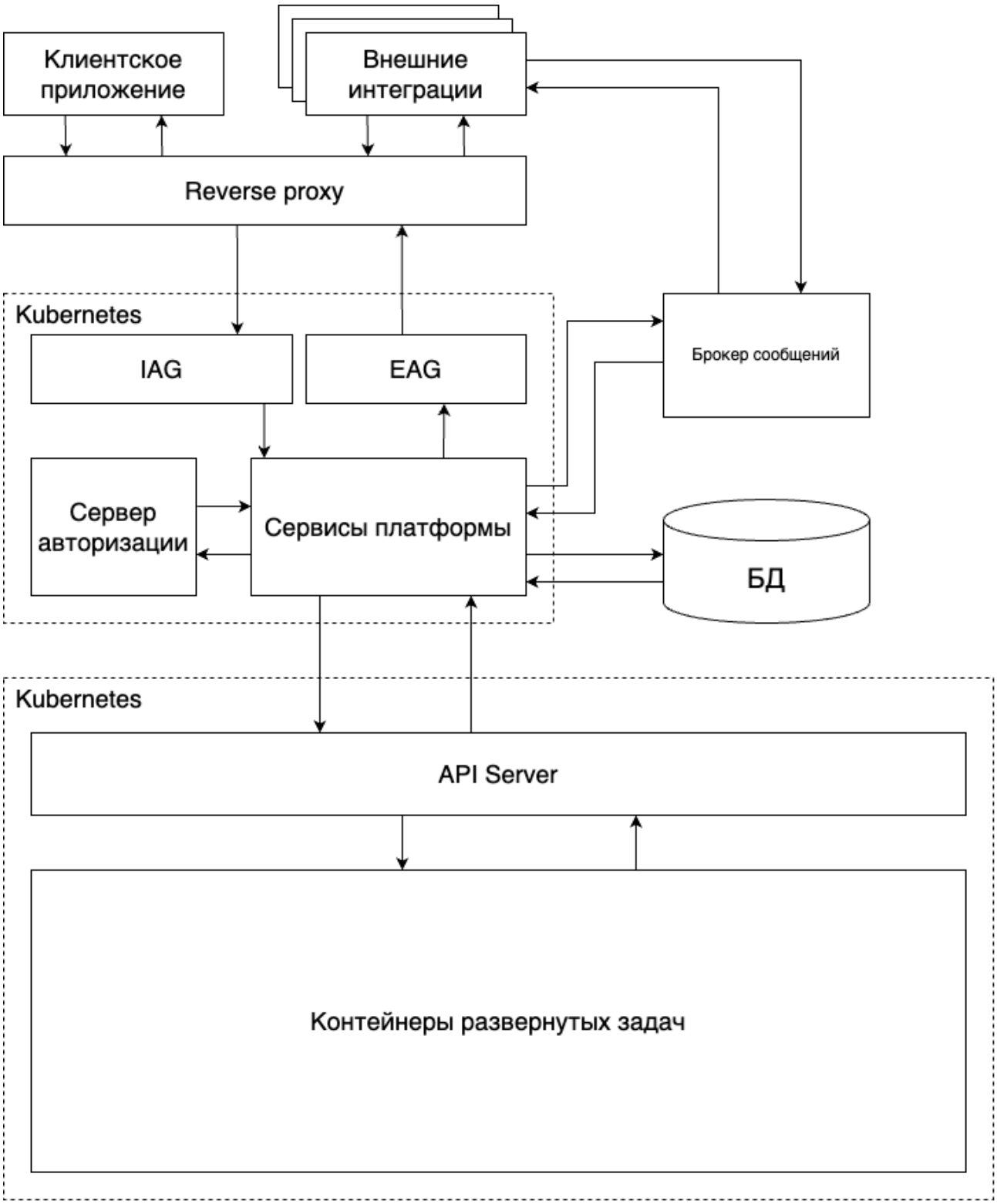

Рис. 2.5. Компоненты плфтормы

сов, сложности с масштабированием.

Виртуальные машины

Вируальные машины позволяют запускать сервисы в изолированных виртуальных средах, хорошо масштабируются но требуют ручного управления ресурсами, сильно уступают контейнерным решениям по эффективности использования ресусов и чаще всего исопльзуются для узкоспециализироанных сервисов.

Контейнеры без оркестрации

Развертывание в контейнерах без оркестрации хорошо подходит для разворачивания небольших сервисов а так же для локального запуска, но при использовании в промышленных средах требование ручного управления контейнерами делает такие системы неприменимыми.

Контейнерные оркестраторы

Контейнерные оркестраторы предоставляют возможность объединения нескольких физических серверов в один кластер с едиными ресурсами, таким образом осуществляется удиное эффективное управление всеми ресурсами системы, достигается ее масштабируемость и отказоустойчивость.

Выбор оркестратора

Для разворачивания разрабатываемой платформы оптимальным выбором является Kubernets, так как он обеспечивает гибкое и эффективное управление ресурсами клстера при помощи yaml манифестов и является стандартом в индустрии для выполнения задач оркестрации контейнеров.

2.2.2. Reverse proxy

Размещение веб-сервера или сервера приложений непосредственно в Интернете дает злоумышленникам прямой доступ к любым уязвимостям базовой платформы (приложения, веб-сервера, библиотек, операционной системы). Для того что бы избежать появления такого рода уязвимостей широко распространен паттер Reverse-proxy [10].

Reverse-proxy представляет собой промежуточный слой между внутренними сервисами приложения и внешними клиентами. Так же веб-сервер, выступающий в роли reverse-proxy может выолнять следующие функции:

— Маршрутизация запросов;

— Балансировка нагрузки;

— SSl-терминация;

— Фильтрация запросов;

— Кеширование;

— Сжатие контента;

— Управление статическими ресурсами.

В качестве reverse-proxy для реализации платформы был выбран веб-сервер nginx. Этот компонент будет отвечать за SSL-терминацию и раздачу статического контента.

2.2.3. Ingress, Egress

В качестве реализацией для Internal Api Gateway(IAG) и External Api Gateway(EAG) выбраны соотвествующие реализации Istio Ingress и Egress.

Главная задача которую выполняет IAG - распределение входящего трафика по репликам сервисов для обеспечения равномерного и эффективного использования ресурсов.

В то же время EAG отвечает за маршрутизацию исходящего трафика. Он служит точкой выхода трафика, направленного на получение внешних ресурсов.

2.2.4. Сревер авторизации

Сервер авторизации в плфторме отвечает за авторизацию и аутентификацию пользователей, а так же запросов от входящих интеграций, выполняет централизованное управление доступом к сервисам платформы.

К возможным реализациям сервера авторизации можно отнести такие сервисы как:

Среди бошаого колличества реализаций сервера авторизации для платформы выбран сервис Keycloak. Keycloak по сравнению с конкурентами обладает следующими преимуществами:

— Решение с открытым исходным кодом;

— Поддержка большого колличество протоколов авторизации и SSO;

— Наличие системы управления польователями и ролями;

— Нативная интеграция с Kubernetes и масштабируемость;

— Возможность локального развертывания;

— Подержка СУБД PostgresSQL;

— Наличие Java клиента;

— Наличие адаптера Spring Sequrity;

— Наличие административной панели.

2.2.5. Брокер сообщений

Брокер сообщений это промежуточное звено в системе. Он обеспечивает асинхронное возаимодействие между компонентами системы. Основаная задача которую выполняет брокер сообщений - позволяет не дожидаться выполнения ресурсоемких операций и получить результат их выполнения асинхронно позже.

Брокеры сообщений играют ключевую роль в реализации таких системных паттернов как очередь сообщений [11], Saga [12], а так же целого семейства Enterprise Integration Pattern (EIP) [13].

На данный момент основными реализациями брокера сообщений представленными на рынке являются:

— Apache Kafka;

— RabbitMQ;

— ActiveMQ;

— Redis Streams.

В качестве основого брокера сообщений на платформе выбран брокер сообщений Apache Kafka. К основным плюсам по сравнению с другими реализациями брокеров можно отнести концептуальное отличие данного брокера отдругих представленным на рынке.

Kafka поррерживает многокартное потребление сообщений, делая возможными сценарии использования Kafka в качестве Enterprise Service Bus [14] и позволяет не терять сообщения при ошибках и прерываниях обработки.

Так же в преимуществам выбора брокера сообщений Kafka можно отнести:

— Масштабируемость и производительность;

— Высокая доступность и отказоустойчивость;

— Поддержка потоковой обработки;

— Гибкость интеграции;

— Интеграция с Kubernetes;

— Обширная экосистема и поддержка сообщества.

2.2.6. СУБД

В проектируемой системе база данных неодходима для хранения у управления пользовательскими данными. Система управления базой данных(СУБД) обеспечивает эффективную организацию данных, в отдельных случаях транзакционность, целостность и надежность. СУБД так же предоставляет возможности масштабирования и интеграций с другими системами.

Все СУБД можно разделить на две группы по способу организации данных - реляционные и нереляционные. Главное отличие реляционных баз данных от нереляционных заключается в использовании реляционными базами данных таблиц с фиксированной схемой для хранения данных и поддержка строгих отношений между таблицами с использованием внешних ключей, в

то время как нереляционные базы данных являются более гибкими и хранят данные в формате ключ - занчение и не продъявля/т каких либо требований к схеме хранимых данных.

Данные, хранимые платформой сильно структурированы и для выполнения бизнесс логики приложения критична их целостность, поэтому единственным верным выбором будет реляционная СУБД. На данный момент на рынке представлены следующие сервисы:

— PostgreSQL;

— MySQL;

— Oracle Database;

— MariaDB.

Для исполновании в реализации платформы выбрана СУБД PostgreSQL по следующим причинам:

— Поддержка сложных типов данных, возможность расширения встроенных типов данных;

— Поддержка транзакций;

— Открытый исходный код и активное сообщество;

— Масштабируемость и отказоустойчивость.

2.2.7. Слой платформенных сервисов

Слой платформенных сервисов играет ключевую роль в системе, так как именно он отвечает за функциональные возможности системы, управление процессами выполнения вычислений.

На данный момент существует два конкурируюзих подхода к организации сервсов приложений - монолит и микросервисы [15].

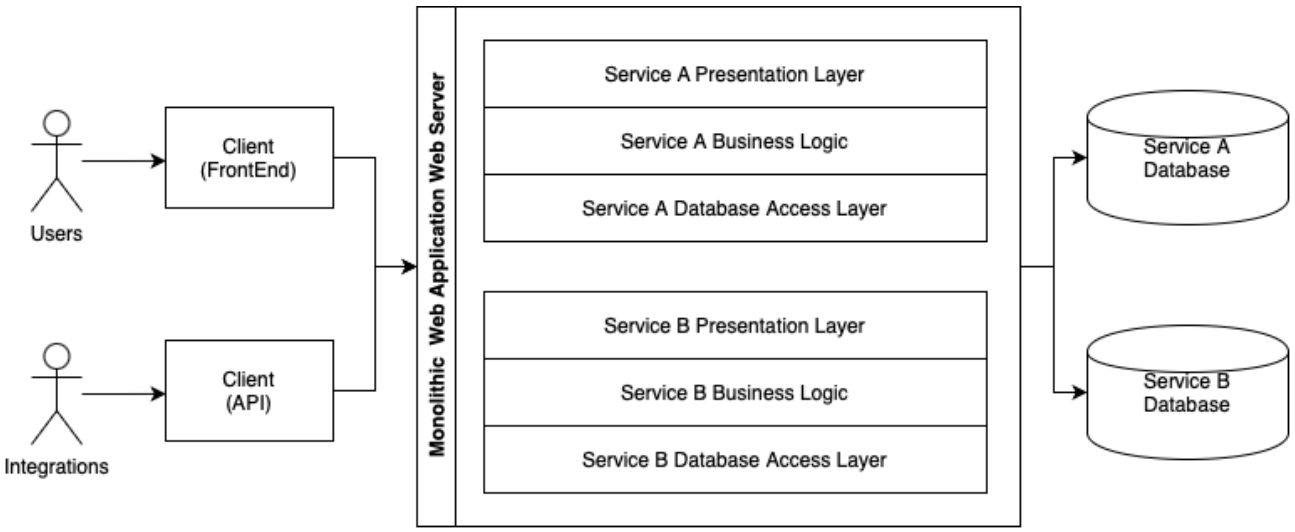

Рис. 2.6. Приложение с монолитной архитектурой

Монолитная архитектура

Монолитный сервис(Рис. 2.6) представляет собой единую единицу развертывания которая отвечает за все функции системы. При использовании такого подхода на начальных этапах можно добиться быстрого вывода на рынок основного функционала, но при длительной разработке и с использованием монолитной архтектуры можно столкнуться с проблемами со сложностью внесения изменений, увеличеии времени холодного старта, увеличением времени проходжения конвеера доставки и развертывания а так же с увеличенным расходом ресурсов при масштабировании и возможными периоднами недоступности при развертыании или неожданных отказах.

Микросервисная архитектура

Главная идея микросервисной архитектуры(Рис. 2.7) заключается в разделении системы на отдельные независимые подсистемы - микросервисы. Каждый микросервис должен отвечать за свою небольшую часть функционала.

Основыным преимуществом использования микросервисов является возможность независимого развертывания и прохожения конвеера доставки и развертывания. Микросервысы могут масштабироваться независимо, позваоля добавлять реплики нагруженным сервисам во время пиковых перидов нагрузки, а малонагруженные сервисы разворачивать с малым колличеством реплик.

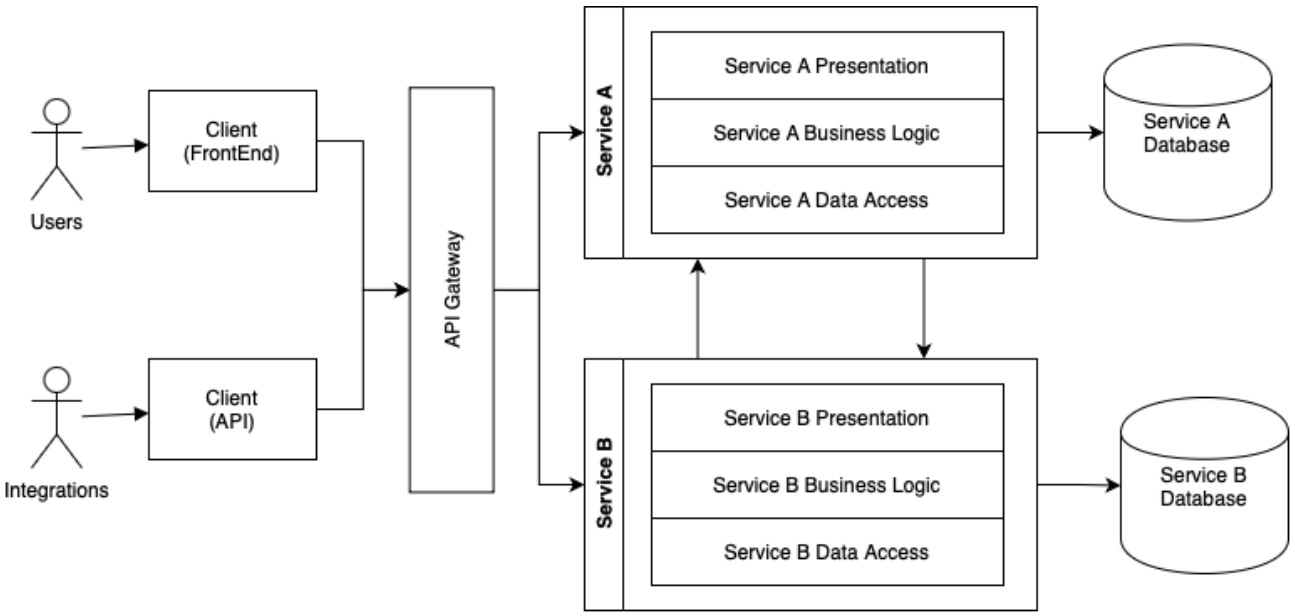

Рис. 2.7. Приложение с микросервисной архитектурой

Использование микросервисной архитектуры усложняет внесение изменений в существующую систему, а так же требует более сложного управления инфраструктурой. При использовании микросервисной архитектуры появляется необхзодимость в сохранении обратной совместимости между версиями сервисов, так как при установке поставки на стенде может находиться одновременно старая и новая версия сервиса для поддржки плавной незамтной для пользователя раскатки. Использвоание микросервисосв так же увеличивает время отклика сервисов, так как возникают дополнительные накладные расходы при выполнении межсервисных вызовов, так же при отказе отдельных сервисов могут возникат тредности с локализацией проблемы, что приводит к необзодимости внедрения распределенный трассировки вызовов.

Выбор архитектуры сервисного слоя

Для сервисного слоя платформы выбрана монолитная архитектура, при этом необходимо осуществлять проектирование функционала так, что бы в последствие он мог быть отделен в отдельные сервисы. При разработке важно соблюдать слабую связанность [16] между сервисами, используя такие подходы как Domain Driven Design(DDD) [17] или Verical Slice [18]

2.2.8. Подчиненный кластер Kubernetes

Подчиненный кластер Kubernetes в системе предназначен для разворачивания в нем задач, создаваемых пользователями. Он является выделенной изолированный средой которая обеспечивает гибкость, масштабируемость и отказоустойчивость системы.

2.3. Обоснование технологического стека

2.3.1. Клиенская часть

При разработке клиенской части приложения использован fontend фреймворк Vue3. Для управления состоянием клиентского приложения испольована библиотука Pinia. Для работы со стилизаций компонентов интерфейса использованы библиотки TailwindCSS и DaisyUI.

Такой стек технологий позволяет эффективно выполнять задачи, поставленные перед разработкой клиентской части. Фреймворк Vue3 обладает полее высокой производительностью по сравнению с предыдущей версией Vue2 и аналогами React и Angurlar.

Так же Vue3 более удобен для разработки и имеет более низкий порог вхождения. Поддрежка TypeScript позволят разрабатывать более надежные и предсказуемые компоненты.

Менеджер управления состоянием приложения Pinia выбран так как является официальной заменой Vuex в Vue3. По сравнению с предшественником новый стейтменеджер более простой и понятный так как в нет мутаций и он по умолчанию поддерживает реактивность, которая пришла на смену CompositionAPI в новой версии фреймворка.

TailwindCSS это библиотека позволяюзая избежать работы со стилями компонентов напрямую. Благодаря гибкой системе заранее предопределенных классов библиотека позволяет сократить дублирование кода и ускорить разработку.

DaisyUI является самой большой и популярной библиотекой UI компонентов постоенной на TailwindCSS. Она предлагает большой набор компонентов и тем для приложения которые гибко интегрируются с остальной системой по средствам применения классов к HTML-тегам.

2.3.2. Серверная часть

Для разработки серверной части платформы выбран язые программиования Kotlin, этот язык относится к семейству языков компилирующихся в байткод поддерживаемый виртуальной машиной Java. Для разработки так же выбрана версия Java 21 так как она является самой последней версией языка для которой на момент написания работы заявлена длинтельня поддержка (LTS).

По сравнению с Java, Kotlin имеет более выразительный синтаксис и является более безопасным так как частью языка являются механимзмы защиты от разименования $null$ указателей [19].

Фреймворком для написания серверной чати приложения выбран Spring и Spring Boot. Данное решение является стандартом в отрасти и зарекомандовало как себя удобное и эффективное для построения такого рода систем.

Экосистема Spring обладает большим колличеством библиотек позволяющими расширить функционал фемворка и интегрироваться с большинством сторонних систем. Для разработки серверной части платформы с состав дистрибутива были включены следующие библиотеки:

— keycloak-admin-client — для интеграции системы с сервером авторизации Keycloak через REST API;

— spring-data-jdbc — для интеграции с СУБД PostgreSQL;

— kubernetes-client-java — для интеграции системы с подчиненным кластером Kubernetes;

— spring-security — для обработки авторизационных токенов и создание политик доступа к предоставляемым сервисам.

Для сборки серверной касти приложения выбрана система автоматической сборки Gradle, так как она является современной заменой устаревший системе сборки Maven, дает более гибкий и удобный инструментарий, а так же на момент написания работы является стандартом в отрасли.

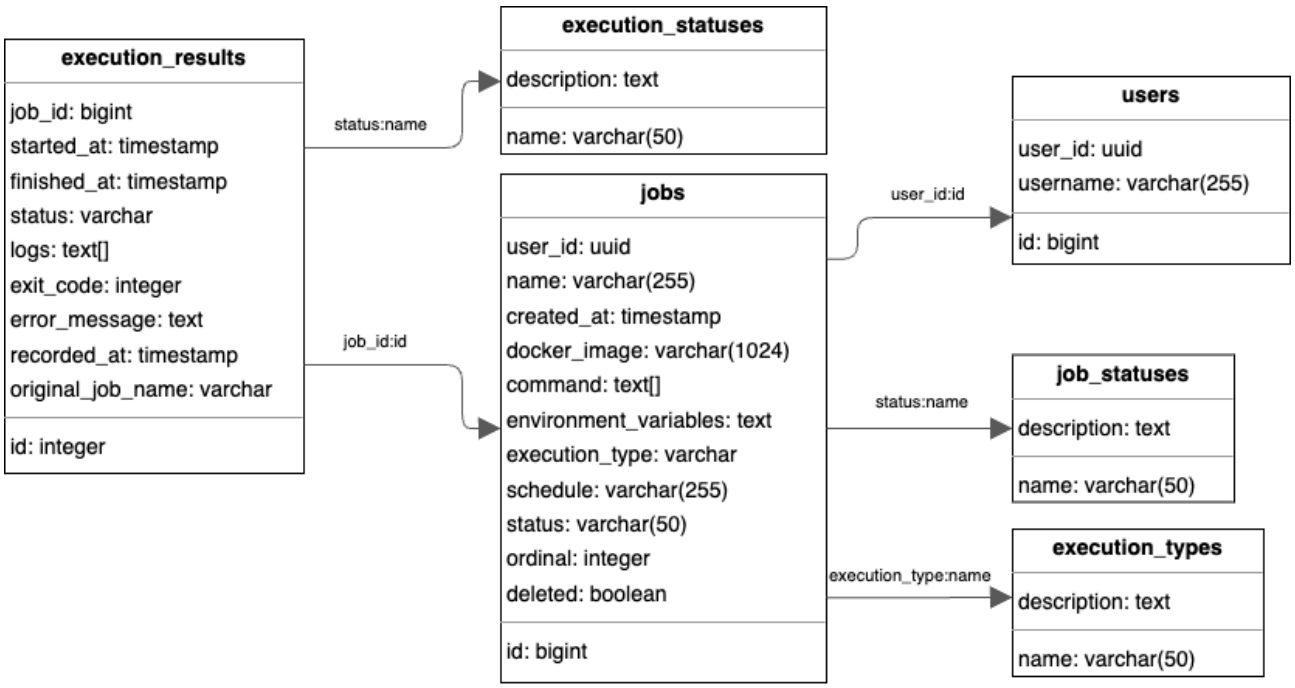

Рис. 2.8. Диаграмма отношений таблиц базы данных

2.4. Разработка модели данных

На платформе используется две базы данных - одна из них используется для хранения данных проектируемого сервиса, вторая - для хранения данных сервера авторизации.

2.4.1. Модель базы данных сервиса

В рамках предметной области основными сущностями являются пользователь, группа пользователей задача, результат выполнения задачи.

На рисунке 2.8 ER диаграмма, на которой изображены сущности базы данных и взаимосвязи между ними.

В модели базы данных испльзуется подход по реализации перечислений (enumirations) с использованием референсных таблиц, в которых первичным ключом является колонка $name$, на которую ссылаются таблицы испольузующие значения из данной таблицы в качестве перечислений. Колонка $description$ при этом является справочной и служит только для хранения человекочитаемого описания значения перечисления.

К референсным относятся таблицы $job\_statuses$, $execution\_statuses$, $execution\_types$ которые хранят статусы задач и запусков.

Таблица $jobs$ является центральной таблицу для хранения информации о

запущенных задачах. Статусом задачи является одно из значений поля $name$ референсной таблицы $job\_statuses$. Время создания записи в таблицу фиксируется автоматически а поле $created\_at$.

Таблица $jobs$ имеет сдедующие поля:

— id — $bigint$ — Уникальный идентификатор задачи.

— $user\_id$ — $uuid$ — Идентификатор пользователя, создавшего задачу.

— $name$ — $varchar(255)$ — Имя задачи.

— $created\_at$ — $timestamp$ — Время создания задачи.

— $docker_image$ — $varchar(1024)$ — Docker-образ, который используется для выполнения задачи.

— $command$ — $text$ — Команда или скрипт, который выполняется в рамках задачи.

— $environment_variables$ — $text$ — Переменные окружения, используемые для выполнения задачи.

— $execution_type$ — $varchar$ — Тип выполнения задачи;

— $schedule$ — $varchar(255)$ — Расписание выполнения задачи (cron-выражение).

— $status$ — $varchar(50)$ — Статус задачи.

— $ordinal$ — $integer$ — Порядковый номер задачи.

— $deleted$ — $boolean$ — Флаг, показывающий, удалена ли задача.

Результаты выполнения задач располагаются в таблице $execution\_result$. Для создания отношения один ко многим с таблицей $job\_statuses$, внешним ключем является поле job_id.

Таблица $execution\_result$ имеет сдедующие поля:

— id — $integer$ — Уникальный идентификатор результата выполнения.

— job_id — $bigint$ — Идентификатор задачи, к которой привязан результат.

— $started\_at — timestamp$ — Время начала выполнения задачи.

— $finished\_at — timestamp$ — Время завершения задачи.

— $status — varchar$ — Статус выполнения задачи.

— $logs — text$ — Массив логов, связанных с выполнением задачи.

— $exit\_code — integer$ — Код завершения выполнения задачи.

— $error\_message — text$ — Сообщение об ошибке.

— $recorded_at — timestamp$ — Время записи результата выполнения.

— $original\_job\_name — varchar$ — Оригинальное имя задачи.

Таблица $users$ хранит данные дублирующие информацию о зарегистрированных пользователях, хранящаюся на сервере авторизации. Она необходима для содния внешних ключей в которых значением является идентификатор пользователя в системе авторизации.

2.4.2. Модель базы данных сервера авторизации

Модель базы данных сервера авторизации играет важную роль в работе системы, так как информация о пользователях, а так же их положение в ролевой модели хранится на сервере авторизации.

Ключевыми сущностями хранения информации о пользователях являются пользователь, атрибут пользователя, группа, роль. ER диаграмма чати сервера авторизации по работе с данными пользователей и ролевой моделью представлена на рисунке 2.9.

Таблица $user\_entity$ отвечает за хранение позовой информации об учетной записи. Первичным идентификатором таблицы является идентификатор пользователя, через который происходит связыване со всеми остальными таблицами, отвечающими за хранение пользовательскиих данных. Она включает в себя такие поля как:

— id — Идентифкатор пользователя;

— $email$ — Элекстронная почта;

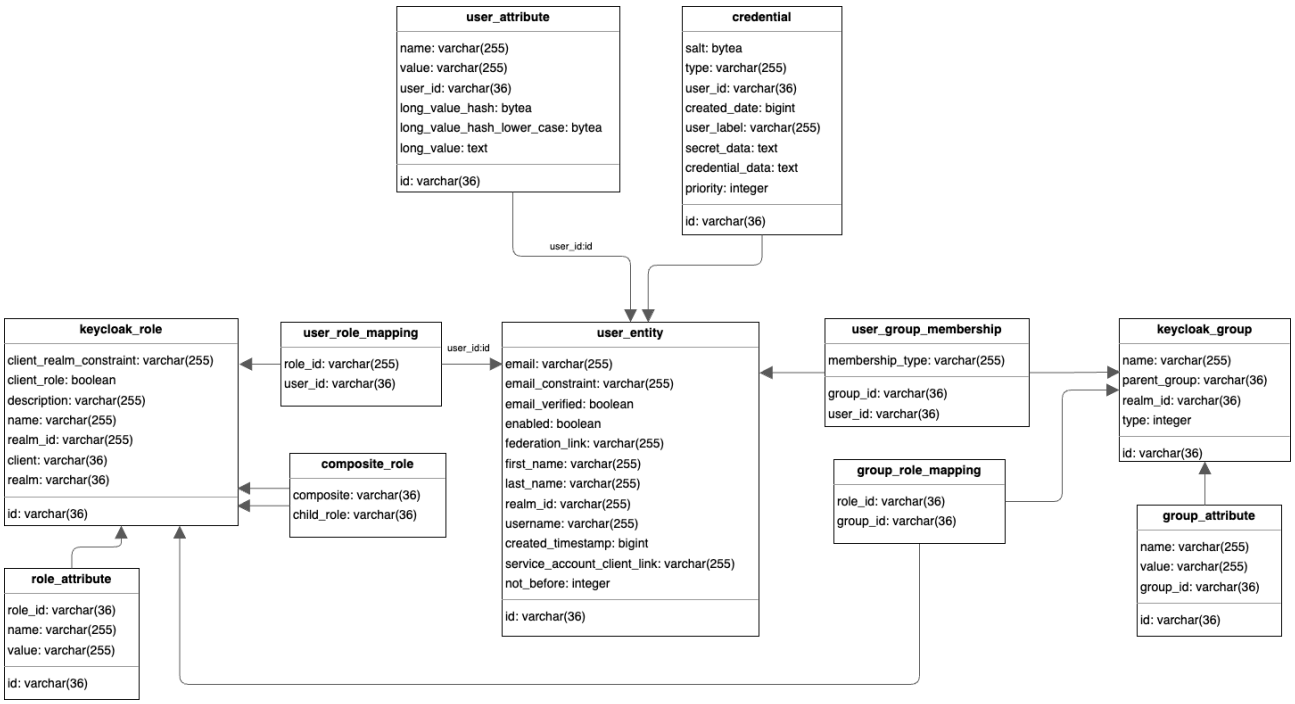

Рис. 2.9. Модель хранения данных пользователей на сервере авторизации

— $email\_verified$ — Признак подтверждения почты;

— $enabled$ — Признак активности учетной записи;

— $first\_name$ — Имя;

— $last\_name$ — Фамилия;

— $realm\_id$ — Идентификатор тенанта пользователя;

— $username$ — Имя пользователя;

— $created\_timestamp$ — Время создания учетной записи;

Пользователям можно добавлять произвольные атрибуты, которые хранятся в таблицу $user\_atrribute$.

Пароли пользователей хранятся в таблице $credetial$ в хешированном виде. Такой подход к хранению паролей пользователей является безопасным, так как хеш пароля невозможно однозначно преобразовать в пароль.

В таблице $keycloak\_role$ хранится информация о ролях пользователей, существующих в системе. К роли можно добавлять произвольные атрибуты, которые хранятся в таблице $role\_attribute$.

Сервер авторизации поддерживает созадние композитных ролей, путем связывания нескольких ролей с одной, родительской ролью. Связывание происходит серез таблицу $composite\_role$.

Роли могут быть связаны с пользователем через отношение многие ко многим, которые реализовано через третью таблицу $user\_role\_mapping$.

Более предпочтительным способ связывания пользователей с ролями является использование межанизма групп пользователей, который позволяет создать статический маппинг наборов ролей на пользователей системы, на основании их принадлежности той или иной группе.

В таблице $keycloak\_group$ хранится информация о группах пользователей. К группе пользователей можно добавлять произвольные атрибуты, которые хранятся в таблице $group\_attribute$. Группы могут быть организованы в иерархическую структуру, что позволяет выстривать сложные ролевые модели.

К группе может быть присвоено несколько ролей, при помощи связывания многие ко многим через таблицу $group\_role\_mapping$. При связывании групп в иерархическую структуру пользоватль наследует все роли родительских групп в которых находится.

Пользователи и группы пользователей связаны отношением многие ко многим — пользователи могут состоять в нескольких группах и в одной группе может состоять несколько пользователей. Для релизации такого подхода используется связывание через третью таблицу $user\_group\_membership$.

2.5. Разработка ролевой модели

Ролевая модель плфтормы реализована на механизмах управления ролями и группами Keycloak. Объединение пользователей в группы позволяет организовать сместное управление выполением задач.

2.5.1. Создание группы

Группа пользователей может быть создана пользователем на странице профиля пользователя. При создании группы, пользователь становится ее администратором.

Так как группы не опрелены заранее и создаются пользователями в процессе использования плфтформы в названии группы должна присутствовать генереририуемая компонента, гарантирующая уникальность идентивикатора группы, так же необзодимо предусмотреть возможность воода имени группы пользователем.

Для того что бы соблюсти все заявленные к имени группы требованиям идентфиикатором группы является случайный UUID идентифкатор. Для того что бы сохранить пользовательское название группы, создаятся атрибут группы с ее названием.

2.5.2. Создание ролей в рамках группы

Для каждой группы создаются роли, соответствующие необходимым уровням доступа к различным частям функционала платформы.

Идентфикатором роли так же будет являться UUID идентфикатор, гарантирующий уникальность, имя роли сохраняется атрибутом $name$.

В системе в рамках группы существуют следующие уровни доступа:

- $VIEW$ — просмотр задач, созданных в группе.
- $EDIT$ —- изменение параметров запущенных задач, их остановка и удаление.
- RUN —- создание новых задач.
- $ADMIN$ —- управление группой (добавление и удаление участников, назначение ролей, генерация токенов доступа).

2.6. Разработка каркасных макетов пользовательского итерфейса

2.6.1. Страница входа и регистрации

Страница входа и регистрации позволяет пользователям авторизоваться или создать новый аккаунт. Форма входа представлена на каркасном макете на рисунке 2.10.

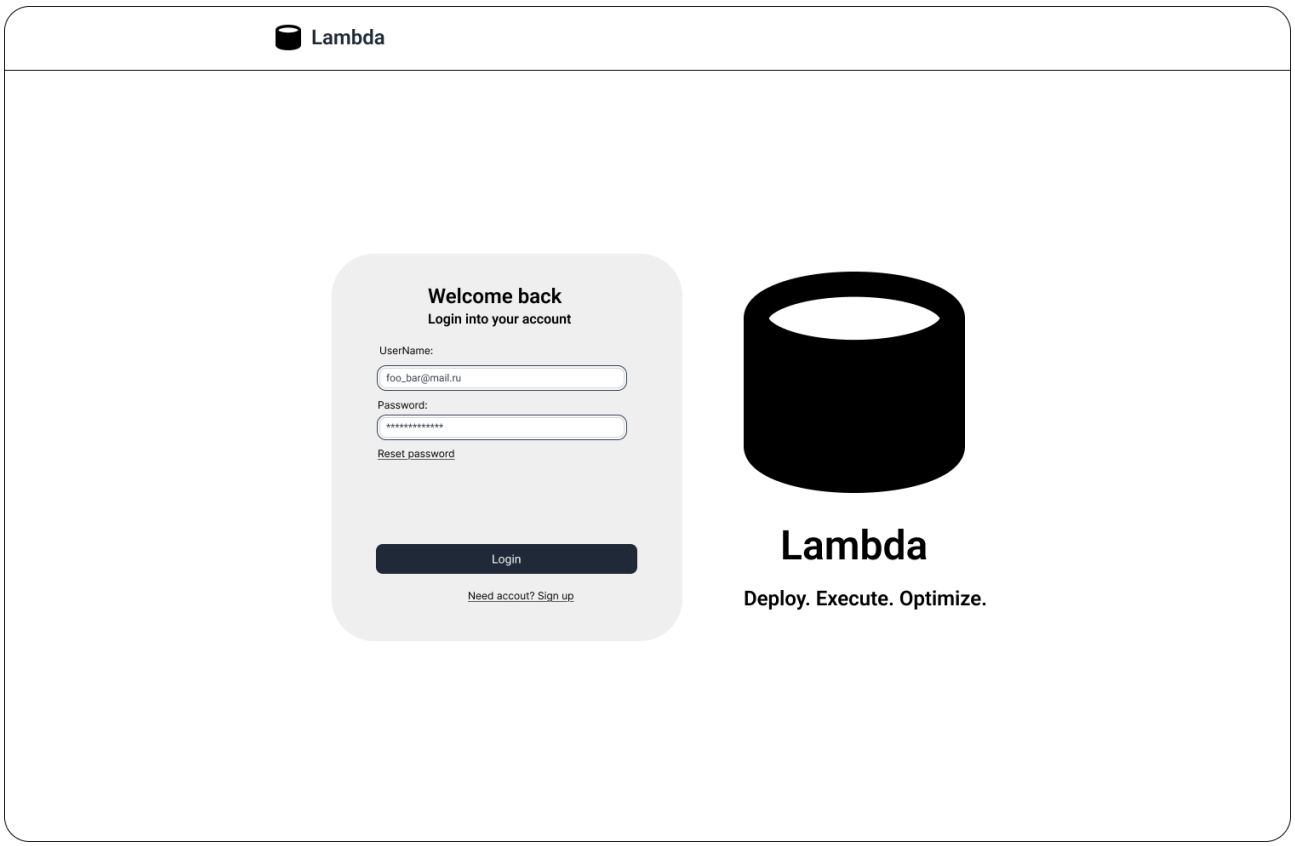

Рис. 2.10. Каркасный макет страницы входа

Для пользователей не имеющих учетной записи в системе, предстусмотре форма регистрации, которя доступная при переходе по ссылке, находящейся в нижней части формы входа. (Рис. 2.11)

2.6.2. Страница запущенных задач

На странице запущенных задач располагается информация обо всех запущенных пользователем задачах (Рис. 2.12).

В верхней части страницы располагается счетчик всех, выполненных, запущенных задач и задач запущенных неудачно. При нажатии накаждый из счетчиков применяется соответствующий фильтр к спску задач расположенному ниже.

При взаимодействии с каким либо из элементов задач, расположенных в списке задач, пользователь переходит на стриницу задачи.

2.6.3. Страница задачи

На странице задачи располгается вся информация о задаче.

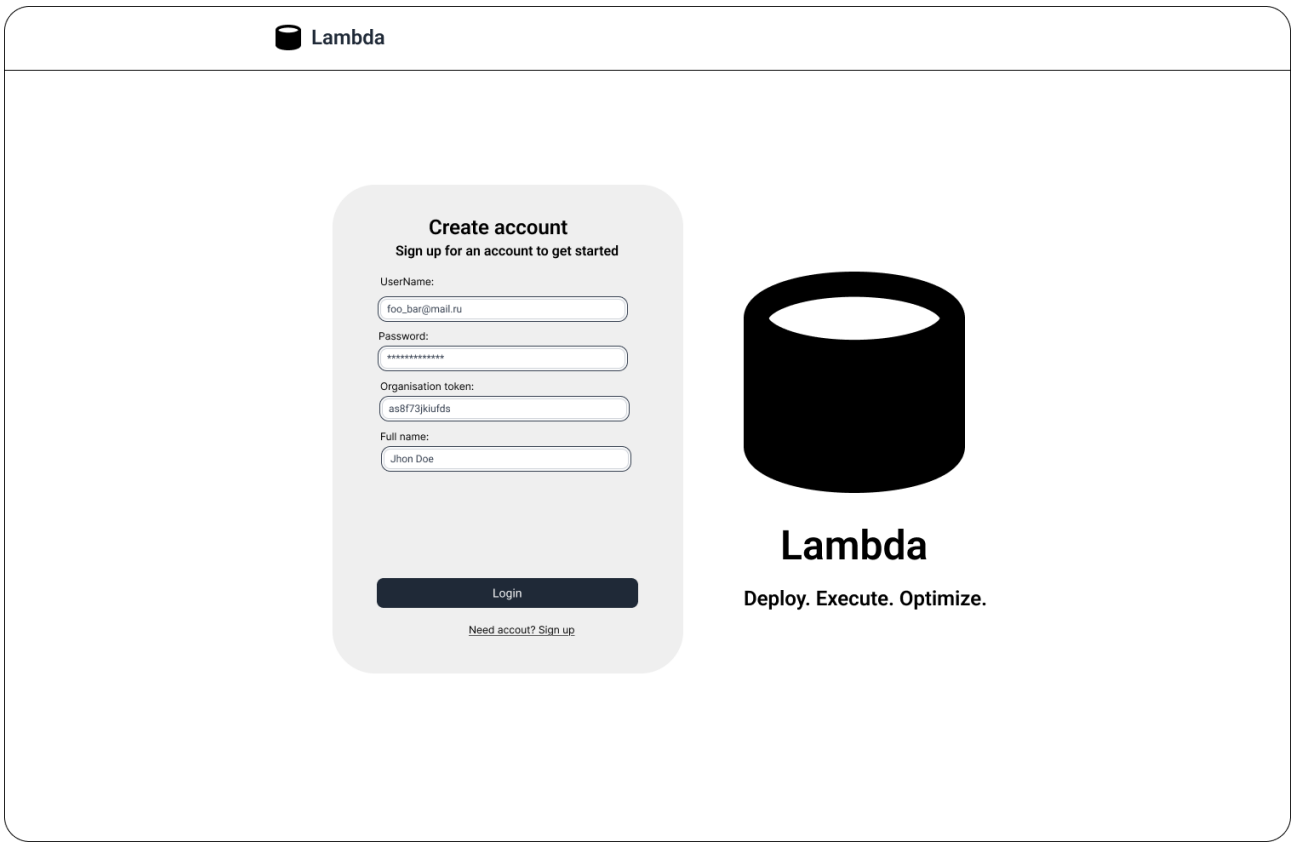

Рис. 2.11. Каркасный макет страницы регистрации пользователя

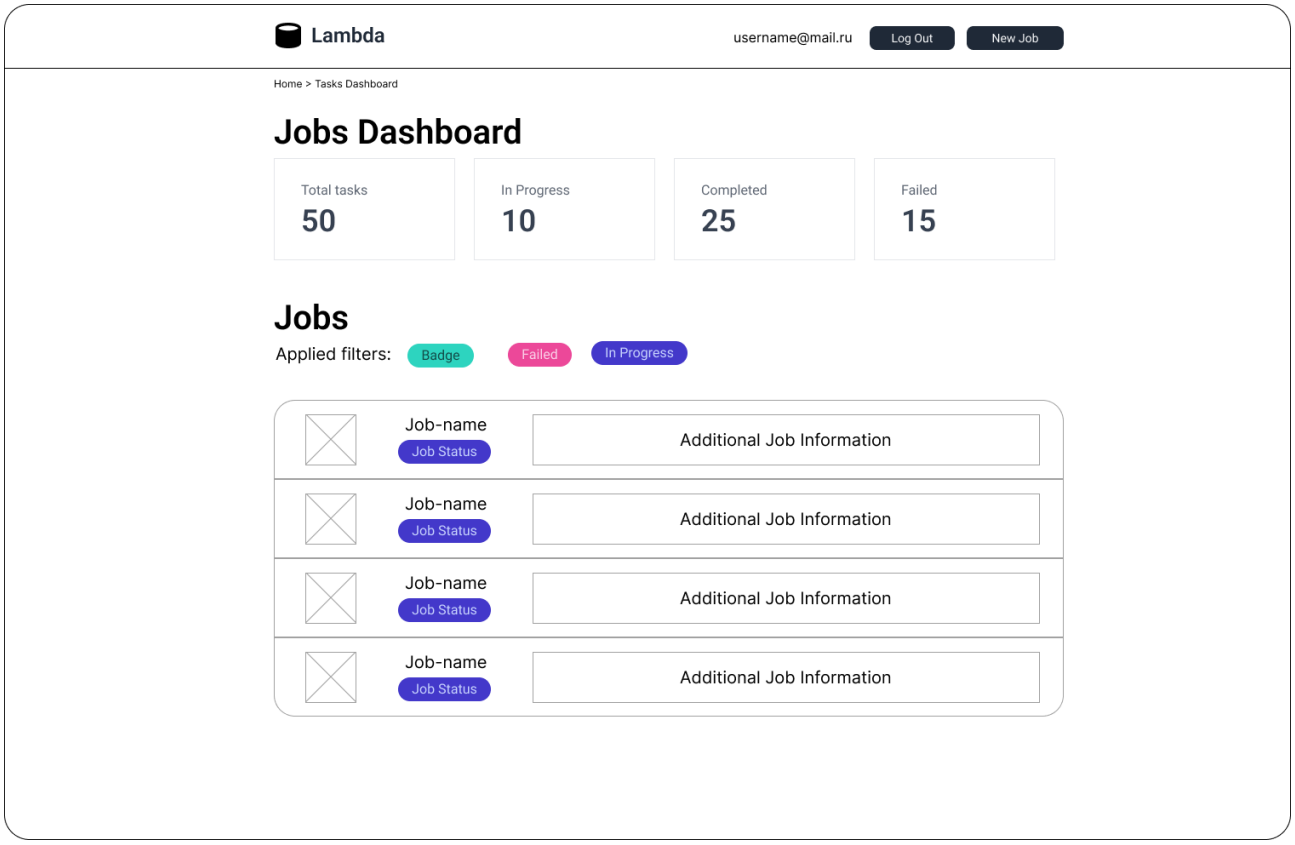

Рис. 2.12. Каркасный макет страницы запущенных задач

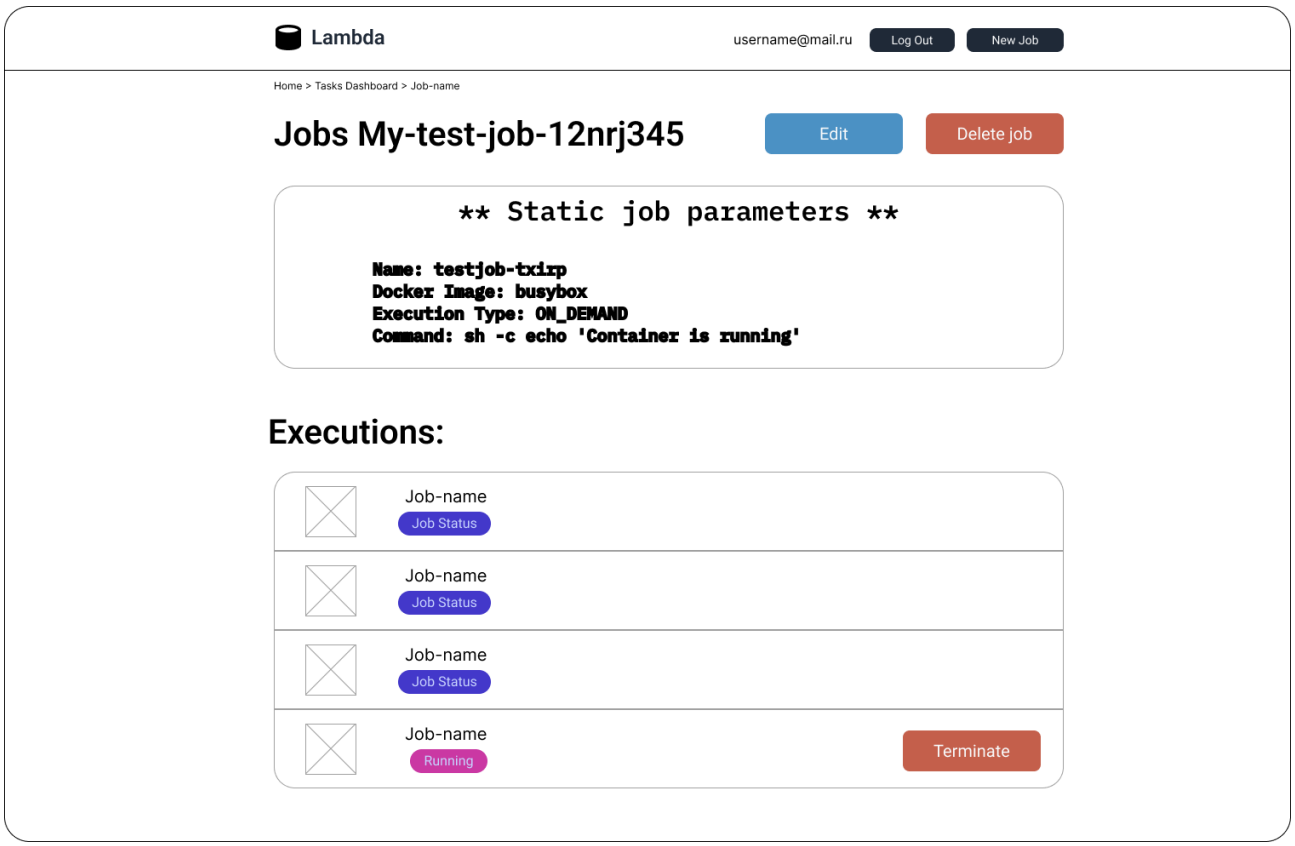

Рис. 2.13. Каркасный макет страницы просмотра запущенной задачи

В Верхней части страницу расположены кнопки редактирования задачи, каоторая позволяет отредактировать параметры запуска задачи для будущих щапусков, и кнопка удаления задачи которая позволяет прекратить выполнение последующих запусков и скрыть от пользователей удаленную задачу.

В нижней части страницы располагается список запусков задачи. Если зажача запущена в данный момент отображается кнопка завершения выполнения. При взаимодействии с каким либо из элементов в списке запусков пользователь переходит на страницу запуска задачи.

На страницу запуска задачи отображается название задачи к которой относится запуск и порядковый номер запуска, кнопки выгрузки логов запуска, завершения выполнения запуска и логи выполнения.

Окно просмотра логов продествляет собой зону просмотра текста с прокртку по осям X и Y.

2.6.4. Страница профиля пользователя

На страницу профиля пользователя отображается информация о пользователе и о группах пользователей в которых он состоит.

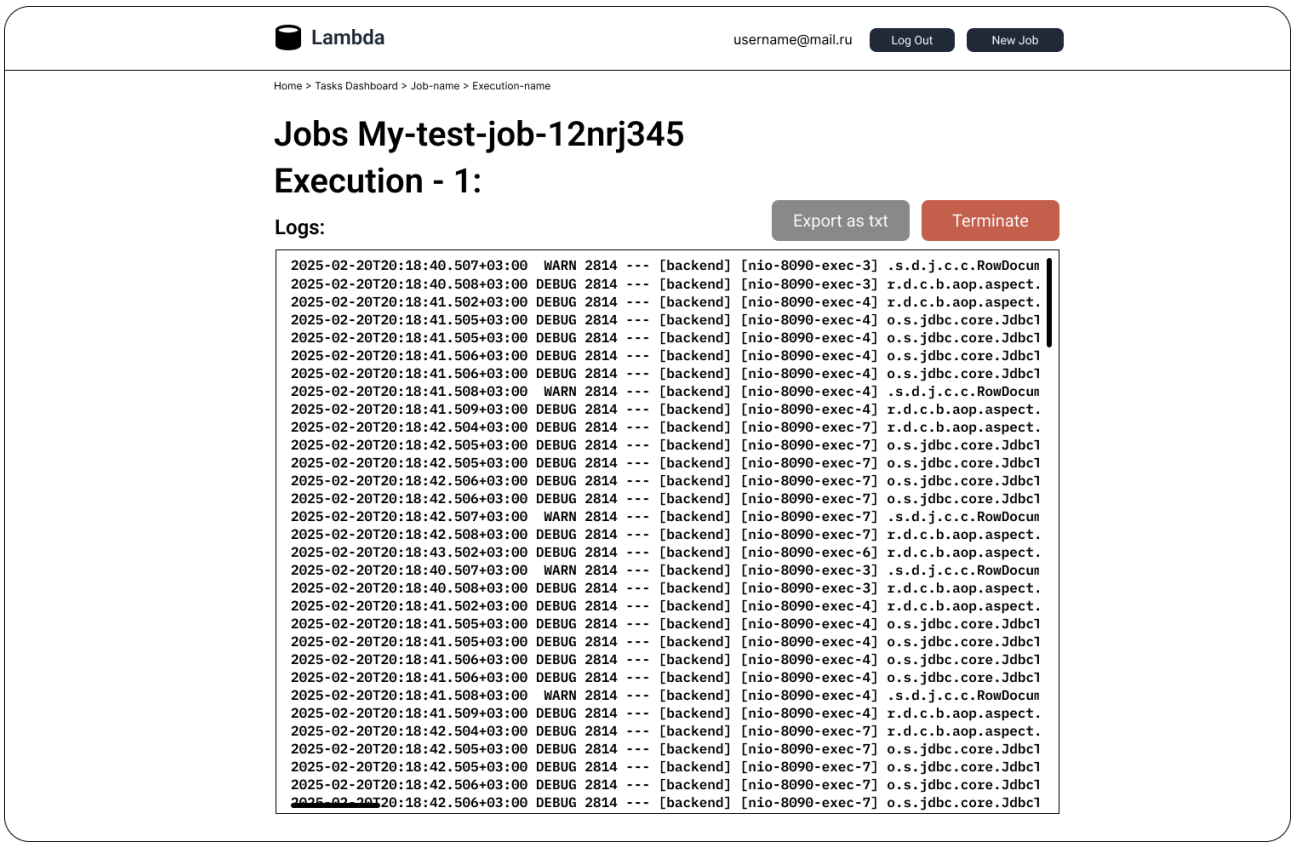

Рис. 2.14. Каркасный макет страницы просмотра запуска задачи

Если пользователь состоит из какой либо группе ему отображется кнопка выхода из группы.

Если пользователь является членом группы администраторов, ему доступна форма генерации и копирования токена доступа в рабочее пространство группы.

В нижней части страницы отображается список участников группы.

Если пользователь является членом группы администраторов, ему доступно управление правами доступа дургих членов группы. Иначе чекбоксы находятся в неактивном состоянии.

2.7. Разработка программного интерфейса сервеной части

2.7.1. Назначение и роль API в системе

Прикладной прграммный интерфейс (API) [20] серверной части системы выступает ключевым элементом архитектуры, обеспечивая возаимодействие между киентами и сервисами разрабатываемой системы.

API в системе выполняет такие такие функции как:

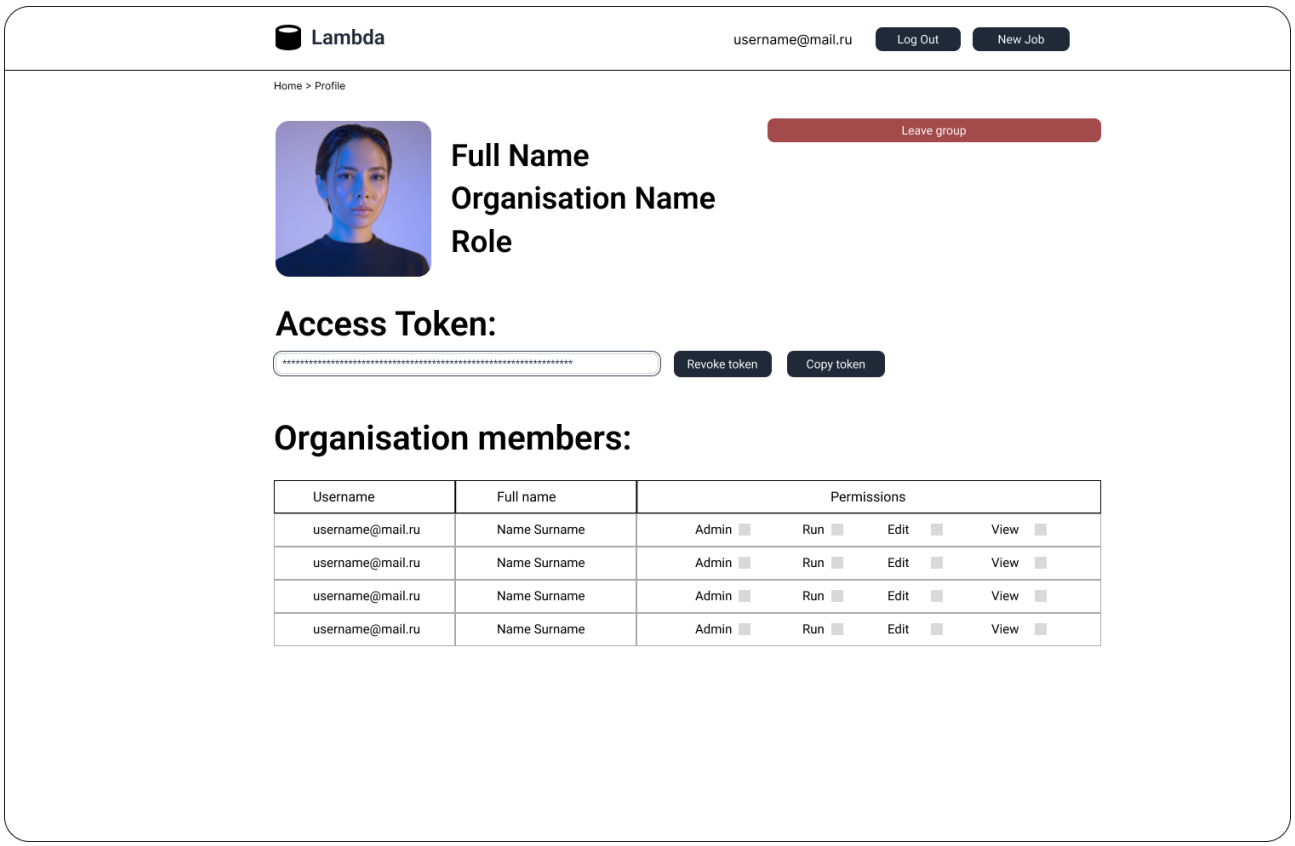

Рис. 2.15. Каркасный макет сраницы

— служит единой точной входят для всех клиентов;

— предосталяет единый способ взаимодействия с системой.

Основными потребителями програмного интерфейса разрабатываемого серверного решение является веб-клиент платформы, а так же потребители webhook [21] сервисов, предоставляемых платформой.

2.7.2. Выбор архитектурного стиля прикладного программного интерфейса

При проектировании серверной части критически важен выбор подхода и врхитектуры проектирования API. Этот выбор определяет подходы взаимодействия между клиентом и сервером, а так же влияет на поизводительность, масштабируемость и простоту интеграции с разрабатываемым сервисом. В настоящее время в проектировании API наиболее распространены 3 подхода:

— REST [22];

— SOAP [23];

— Различные RPC протоколы [24].

В качестве архитектурного подхода к проектированию программного интерфейса выбран Representational State Transfer (REST).

Ключевыми аспектами в выборе REST как архитектурного подхода к проектированию программного инерфейса стали:

— широкое распространение и поддержка: rest является отраслевым стандартом при проектировании веб-API;

— использование стандартных протоколов HTTP/HTTPS;

— простота интеграции;

— использоваение текстового кодирования данных: упрощает и ускоряет отладку.

2.7.3. Основные ресурсы и логические группы конечных точек

Прикладной программный инерфейс платформы предусмотривает набор операций, позволяющих клиентам управлять ресурсами платформы. К основыным ресурсам плфтормы можно онести:

— JwtToken - ресурс, представляющий набор данных, содержащих криптографические токены, необходимые для взаимодействия с конечными точками, защищенными механизмами авторизации;

— User - ресурс, представляющий набор данных о пользователе, в том числе информацию о пользователе, о его ролях и группах в которых он состоит;

— GroupDescription - ресурс, представляющий группу пользователей;

— Permission - доступ к той или иной части функционала платформы в рамках группы пользователей и разделенных между ними ресурсами;

— Job - ресурс, представляющий информацию о задаче, включает все параметры выполнения задачи, а так же информацию о ее запусках;

— ExecutionResult - ресурс, представляющий информацию о запуске задачи.

Логически все конечные точки программного интерфейса системы можно разделить по различным наборам признаков: по доступности из неавторизованной зоны, по ресурсу, к которому конечная точка позволяет получить доступ, по наличию вляния на состояние системы.

Из неавторизованой зоны доступны только конечные точки, позволяющие польюзователю зарегистрироваться, пройти авторизацию или сбросить пароль. К этой группе конечных точек относятся:

— POST /public/api/v1/authentication/reset-password;

— POST /public/api/v1/authentication/log-in;

— POST /public/api/v1/registration;

Все остальные конечные точки доступны только из авторизованой зоны.

По ресурсу, к которому контрольная точка предоставляет доступ конечные точки можно разделить на группы:

— Управление пользователями;

— Управление групповым досутпом;

— Управление запущенными задачами;

К группе конечных точек, позволяющих управлять пользователями относятся конечные точыеи доступные из неавторизованой зоны.

К группе конечных точек, позволяющих управлять групповым доступом пользхователей относятся конечные точки:

— POST /api/v1/role-model/refresh-join-group-token/groupId;

— POST /api/v1/role-model/leave-group/groupId;

— POST /api/v1/role-model/join-group;

— POST /api/v1/role-model/exclude-member-from-group;

— POST /api/v1/role-model/create-group;

— POST /api/v1/role-model/change-permission;

— GET /api/v1/role-model/group/groupId;

— GET /api/v1/role-model/get-join-group-token/groupId.

К группе конечных точек, позволяюших управлять запущенными задачами относятся:

— GET /api/v1/job

— POST /api/v1/job

— POST /api/v1/job/webhook/run/id

— POST /api/v1/job/rerun/id

— POST /api/v1/job/delete/id

— POST /api/v1/job/cancel/id

— GET /api/v1/job/id

К группе конечных точек, позволяющих менять состояние ресурсов системы относятся все конечные точки использующие метод пост при выполнении http зпроса.

2.7.4. Описание сценариев использования

В системе предсумотрены различные сценарии использования API для выполнения задач управления удаленными вычислениями через плфторму.

К основным сценариям можно отнести:

— Авторизация пользователя;

— Создание новой группы пользователей;

— Добавление пользователя в группу;

— Исключение пользователя из группы;

— Добавление, отнимание доступов пользователя в рамках группы;

— Просмотр списка участников группы и из доступов;

— Получение списка запущенных задач и просмотр результатов выполнения;

— Создвние новой задачи;

— Перезапуск, остановка, удаление задачи.

Авторизация пользователя

Пользователь может пройти авторизацию вызвав конечную точку POST /public/api/v1/authentication/log-in.

Если пользователь не имеет учетной записи в системе он может ее создать вызвав конечную точку POST /public/api/v1/registration.

Ответом сервера при входе и регистрации будет пара Jwt токенов [25], которые пользователь должен использовать для доступа к конечным точкам защещиенным механизмами авторизации.

Если пользователь уже имеет учетную запись, но забыл от нее пароль, он может его сбросить, при вызове эндпоинта reset-password.

Сценарии управления группами пользователей

Групповая модель доступа к ресурсам разрабатываемой системы предполагает гибкую систему групп и доступов, позволяющую разграничивать доступ к ресурсам платформы у различных пользоватлей.

По умолчанию пользователь не состоит ни в какой группе и может создавать и просматривать задачи только от своего имени.

Для того что бы создать группу пользователей необходимо вызвать конечную точку POST /api/v1/role-model/create-group. Пользователь, создавший группу, будет назначен ее администратором, а так же ему будут присвоены все дуступы в рамках созданной группы.

Для того что бы добавиьт пользователя в группу предусмотрен механизм токенов доступа в группу. Администратор группы может запросиьт токен доступа вызвав конечную точку GET /api/v1/role-model/get-join-group-token. Дальше токен любым удобным способ отправляется пользователю который должен вступить в группу.

Что бы присоединиться к группе необходимо вызвать конечную точку POST /api/v1/role-model/join-group. Пользователь будет добвален в группу с доступом на просмотр по умолчанию. Для того что бы расширить перечень доступов пользователя в рамках группы, администратор группы должен произвети соответсвующие операции.

Для добаления или удаления доступа у пользователя, являющегося членом группы, администратор должен воспользоваться конечной точкой POST /api/v1/role-model/change-permission.

Пользователь может покинуть группу используя конечную точку POST /api/v1/role-model/leave-group/{groupId}.

Администратор так же может исключить пользователя из группы используя конечную точку POST /api/v1/role-model/exclude-member-from-group.

Просмотр, создание и управление запущенными задачами

Пользователь может создать задачу доступную только ему или членам группы по его выбору вызвав эндпоинт POST /api/v1/job.

Запросить список текущих задач можно вызвав эндпоинт GET /api/v1/job.

В зависимости от членства в различных группах и наличия соответствующих доступов у пользователя, при запросе текущих задач он получит список задач доступных ему или члеам группы в которую он входит и имеет доступ на просмотр.

Отдельную задачу можно получить вызвав эндпоинт GET /api/v1/job/id.

При наличии досупа пользовать может перезапустить задачи при помощи эндпоинта POST /api/v1/job/rerun/id, отменить выполнение задачи POST /api/v1/job/cancel/id или удалить задачу POST /api/v1/job/delete/id.

Если задача является веб-хуком, то пользватель может его вызвать через интерфейс или API плфтформы возмользовавшись конечной точкой POST /api/v1/job/webhook/run/id.

2.8. Разработка сервеной части

2.8.1. Архитектура серверной части

Серверная часть система является распределенным приложеним, в котором монолитное веб приложение служит связующим компонентом между другими частями системы. Основу системы соствляет веб-приложение, напи-

санное на языке Kotlin и фреймворке для написания веб-приложений Spring Boot.

Помимо разрабатываемого приложения ключевыми компонентами системы являются кластер Kubernetes, в котором разворачиваются сервисы платформы и пользовательские задачи, база данных PostgreSQL, выполняющая функции хранения данных веб-прложения и сервера авторизации, а так же сервер авторизации Keycloak, выполнязий задачи упроавления токенами доступа пользователей и хранения информации о ролях и доступах пользователей.

2.8.2. Реализация REST API

REST API платформы спроетирован в соответсвие с подходами ресурс-ориентированной архитектуры(ROA) [26] и использует стандарт OpenAPI 3.0 для описания конечных точек.

Конечные точки платформы сгруппированы по функциональным областям - регистрация и авторизация, управление групповым доступом, управление жизненным циклом задач.

Версионирование приклодного инфтерфейса плфтформы осуществляется путем досбавления префикса v1 перед названием группы конечных точек, таким образом при дальнейшем развитии платформы, при отсутствии обратной совместимоти могут безопасно эволюционировать.

Система авторизации разрабатываемой системы использует протокол OAuth 2.0 [27] для авторизации пользователей. За генерацию и валидацию JWT-токенов [25] отвечает сервер авторизации Keycloak.

Механизмы авторизации, основанные на использовании JSON Web Token передают данные о пользователе в незашифрованном виде.

Каждый JWT токен состит из трех частей:

— заголовок - содержит название алгоритма подписи токена;

— полезная нагрузка(payload) - набор утверждений(claims) о пользователе;

— подпись токена - HMAC-хэш, вычисленный сервром авторизации с использованием приватного ключа.

Полезная так как JWT-токен передает полезную нагрузку в незашифрованном виде, утверждения о пользователе не должны содержать какой либо

чувствительной информации. Подмена утверждений содержащихся в токене невозможна так как при изменении набора утверждений, неизбежно изменится их хеш, содержащийся в последней части токена и вычисляемый с использованием приватного ключа, который известен только серверу авторизации. Таким образом гарантируется безопасность доступа к ресурсом платформы.

В листинге 2.1 предаствлен пример расшифрованного токена доступа, которым оперирует платформа.

Листинг 2.1. Токен доступа

```
{
  "exp": 1743891730,
  "iat": 1743855730,
  "jti": "59864e52-01f6-4781-a342-6e32f444eb90",
  "iss": "http://localhost:8080/realms/users",
  "aud": "account",
  "sub": "84e487cc-a864-462c-8074-bbb4d020c380",
  "typ": "Bearer",
  "azp": "backend-client",
  "sid": "9dd6e8b2-54d8-45e5-a325-2874b4090c81",
  "acr": "1",
  "allowed-origins": [
    "/*"
  ],
  "realm_access": {
    "roles": [
      "default-roles-users",
      ...
      "offline_access"
    ]
  },
  "resource_access": {
    "account": {
      "roles": [
        "manage-account",
        "manage-account-links",
        "view-profile"
      ]
    }
  },
  "scope": "profile email",
```

```
    "email_verified": true,
    "preferred_username": "a@mail.ru"
  }
```

При проектировании авторизации платформы, критическим требованием была реализация платформы без редиректов, из-за этого было принято решение отказаться от использования форм авторизации Keycloak и реализовать собственный API авторизации, делегировав серверу авторизации лишь задачи генерации и валидации токенов.

2.8.3. Технологии и архитектура веб-сервиса

Спроектированный веб-сервис реализует построен с ипользованием распространенного паттерная шестигранной архитектуры [28] и соблюдает прицины второго уронвня модели зрелости Ричардсона [29].

На уровне приложения находятся сервисы платформы, отвечающие за реализацию бизнес логики работы с доменными сущностями. К таким сервисам относятся:

— AuthenticationService;

— ExecutionService;

— JobService;

— RegisterUserService;

— UserService.

В этих сервисаз содержится бизнес-логика отвечающая за согласнованную работу системы и выполнение ей, возложенных на нее задач.

С Application слоем взаимодейтвует слой контроллеров, к оторому относятся следующие классы:

— AuthenticationController;

— JobController;

— RegistrationController;

— RoleModelController;

— UsersController.

Слой контроллеров отвечает за обработку входящих запросов и преобразование ответов сервисного слоя в сущности ResponseEntity.

Маршрутизацией трафика на конечные точки, объявленные в контроллерах занимается веб-сервер Tomcat.

Для звамодействия с базой данных спользуется слой DAO (Data Acess Object).

Для работы с информацией, хранящейся в базе данных исопльзуется порход ORM (Object Relational Mapping) [30]. Данный подход реализован с испльзованием стандарта JPA (Java Persitance API) [31], который реализуется библиоткеой Spring Data JDBC, предоставляющей лаконичный API для взаимодействия с базой данных, а так же легко интеграруемой с основным фреймворком приложения Spring Boot.

На слое DAO представлены два типа сущностей - Entity и Repository.

Entity классы предствляют собой отражение доменных сущностей хранящихся в базе данных, к ним относятся пользователи, задачи и результаты выполнения задач.

Repository в свою очередь является классом, позволяющим получить и менять состояние Entity которой он управляет. На слое DAO представлены следующие репозитории:

— ExecutionResultRepository;

— GroupAccessTokenRepository;

— JobRepository;

— UserRepository.

На интеграционнном слое приложения располагаются сервисы, обеспечивающие взаимодействие с другими сервисами платформы. К таким сервисам относятся KeycloakService, реализующий интеграцию с сервером авторизации Keycloak и KubernetesService, позволяющий взаимодейстоввать с кластером Kubernetes, в котром разворачиваются создаваемые пользователями задачи.

Для взаимодействия с Keycloak адаптер KeycloakService использует клиентскую библиотеку $org.keycloak : keycloak-admin-client$, котороя является тонким клиентом для сервера авторизации кейлок, предоставляя удобный DSL (Domain Specific Language) [32] интерфейс для взаимодействия с сервром авторизации. Библиотека берет на себя функции управления токенами досутпа, кешированными локально на стороне клиента, а так же за преобразование сущностей DSL в вызовы REST API сервера авторизации.

Адаптер KubernetesService использует библиотеку $io.kubernetes : client-java$ для взаимодействия с подчиненным кластером Kubernetes. Подключение к кластеру устанавливается при помощи файла конфигурации $.kubeconfig$, путь к которому задан в переменной окружения $\$KUBECONFIG$.

Клиентаская библиотка так же является тонким клиентом для Kubernetes и предоставляет удобный DSL API для генерации манифестов Kubernetes [33], а так же предоставляет удобные абстракции для вызова метдов $Kubernetes\ APIServer$.

2.8.4. Интеграция с Kubernetes

Для обеспечения изоляции пользовательских задач от сервисов платформы, реализован подрход разделения пространств имен. Исопльзуются два основыных пространства:

— пространство $default$ - служит для развертывания системных сервисов, таких как разработнный веб-сервис, север авторизации, база данных.

— пространство $sandbox$ - предназначено для развертывания пользовательских задач.

Такое разделение позволяет минимизировать риск несанкционированного досутпа к ресурсам платформы и предотвратить риск возможных конфликтов именования ресурсов платформы.

Для реализации возможности создания пользователь задач используются встроенные сущности Kubernetes Job и CronJob. KubernetesService управлет жизненным циклом объектов выполняя следущие операции:

— создание ресурсов;

— получение статуса выполнения;

— получение логов выполнения;

— остановка и удаление.

За генерацию манифестов отвечает фабрика [34] $V1JobFactory$. В этом классе инкапсулирована логика преобразования параметро создания задачи $JobParameters$ в манифест Kubernetes.

Ключевые аспекты конфигурации включают:

— установку $metadata.name$ для идентификации Job и $CronJob$;

— определение спецификации контейнера ($V1Container$), включая образ ($image$), команду ($command$) и переменные окружения (env);

— установку политики перезапуска $restartPolicy$ в значение $Never$ для Pod, создаваемых в рамках Job, что соответствует семантике выполнения задачи до завершения;

— конфигурацию $backoffLimit$ для $JobSpec$, определяющую количество попыток перезапуска в случае сбоя;

— для $CronJob$ дополнительно указывается поле $schedule$ и шаблон $jobTemplat$

После запуска задачи сервис $ExecutionService$ периодически опрашивает $KubernetesService$ для актаулизации статуса выполнения задачи. Собранная аггрегированная информация о статусе выполнения задачи исопльзуется для обновления информации о задаче в базе данных, откуда в последствии данная информация будет доставлена пользователям.

2.8.5. Интеграция с Keycloak и реализация механизмов ролевого доступа

Основное взаимодействие разработанного веб-сервиса с сервером авторизации Keycloak сокрыто в классе $KeycloakService$. Вся информация об учетной записи пользователя хранится на сервере авторизации. $KeycloakService$ API предосталяет достаточный функционал для управления жизненным циклом учетной записи пользователя.

Изнетграция с Keycloak осуществляется через экземляр DSL-фабрики Keycloak. В процессе формирования обращений к API Keycloak ключевыми сущносятми, в которые преобразуются доменные сущности платформы, являются:

— UserRepresentation - DTO (Data Transfer Object) пользователя Keycloak;

— CredentialRepresentation - DTO учетных данных пользователя;

— GroupRepresentation - DTO группы пользоватей Keycloak;

— RoleRepresentation - DTO роли уронвя realm Keycloak.

Для поддержания фукционалности управления ролевым доступом реализованы межанизмы генерации ролей уровня realm на основании идентификатора группы и доступа который присваивается опльзователю в рамках этой группы.

В рамках модели группа — доступ, роли, присваиваемые пользователю именуются по шаблону:

$$\{Permission\_Name\}/\{Group\_UUID\}$$

Такой подход к организации доступов пользователя позволят выдать пользователю разный набор досутпов в разных группах, исключая возможность пересечения досутпов между группами.

В последстии в токене доступа, в утверждении $realm\_access.roles$ созданные роли принимают вид представленный в листиге 2.2.

Листинг 2.2. Пример представления ролей пользователе в токене досутпа

```
...
    "realm_access": {
      "roles": [
        "RUN/1b1d286b-1a33-4c17-8dda-d9a1b4037253",
        "ADMIN/1b1d286b-1a33-4c17-8dda-d9a1b4037253",
        "EDIT/216ef3cf-aa7c-4612-85f0-fad77f3a3637",
        "VIEW/1b1d286b-1a33-4c17-8dda-d9a1b4037253",
        "EDIT/1b1d286b-1a33-4c17-8dda-d9a1b4037253"
      ]
    },
```

```
...
```

2.8.6. Обработка авторизации пользователя

Контроль за доступом к ресурсам плфотрмы яфляется критически важным спктом разработки системы. В системе реализованы механизмы валидации токенов доступа пользователей, а так же механизмы раздедедения конечных точек на авторизованную и неавторизовванную зоны.

Интеграция с Keycloak и конфигурация Spring Security

Интеграция с сервером авторизации Keycloak, для валидации токенов досутпа пользователей, осуществляется с использованием стандартных механизмов Spring Boot для конфигурации OAuth 2.0 Resource Server [35].

Пример конфигурации ресурсного сервера представлен в листинге 2.3.

Листинг 2.3. Конфигурация Resource Server

```
...
  spring:
    security:
      oauth2:
        resourceserver:
          jwt:
            issuer-uri: http://keycloak:8080/realms/users
...
```

Ключевым аспектом конфигурации является указание парметра $issuer - uri$, определяюзего адрес сервера, выпустившего JWT-токены досутпа, которым доверяет веб-сервис.

При образении на конечную точку $issuer - uri$, Keycalok предоставляет набор публичных ключей, с помощью которых веб-сервис может валидировать токен досутпа в запросе. Пример ответа сервера авторизации на запрос публичных ключей представлен в листиге 2.4.

Листинг 2.4. Ответ на запрос публичного ключа

```
...
{
  "realm": "users",
```

```
  "public_key": "MIIBIj...QAB",
  "token-service": "http://keycloak:8080/realms/users/protocol/
     openid-connect",
  "account-service": "http://keycloak:8080/realms/users/account",
  "tokens-not-before": 0
}
...
```

При получении ответа сервера авторизации с публичным ключем, Security middleware проверят токен досутпа по следующим параметрам:

— Подпись токена: Проверяется с использованием публичного ключа, полученного от Keycloak по $issuer - uri$.

— Срок действия (Expiration): Проверяется, не истек ли срок жизни токена (exp утверждение).

— Издатель (Issuer): Проверяется соответствие значения iss claim в токене сконфигурированному $issuer - uri$.

Конфигурация правил доступа в Spring Security

Для разграницения досутпа у конечным точкам, требуюзим авторизованный доступ, исопзьуется бибилитека Spring Security. Класс $SecurityConfig$ является единым местом настройки политик доступа к сервисам платформы.

Для сервисов платформы применяются следующие политики досутпа:

— Пути, начинающиеся с /public/**, пути для Swagger UI (/v3/api-docs/**), доступны всем пользователям без аутентификации.

— HTTP-метод OPTIONS разрешен для всех путей для корректной работы механизма Cross-Origin Resource Sharing (CORS) [36], который используется при взаимодействии frontend-приложения с backend API из разных источников (доменов).

— Все остальные запросы требуют наличия валидного JWT-токена в заголовке Authorization: Bearer <token> (authenticated()).

— Отключение CSRF [37]: Защита от Cross-Site Request Forgery отключается, так как JWT-аутентификация является stateless и не подвержена данному типу атак в той же мере, что и сессионная аутентификация.

— Настройка обработки JWT: Секция $oauth2ResourceServer().jwt()$ конфигурирует Spring Security для работы в режиме ресурсного сервера, ожидающего JWT-токены. Здесь же подключается пробразователь токенов KeycloakJwtTokenConverter, преобразущий утверждения токена доступа в Principal объект Spting Secutiry.

Извлечение данных пользователя и ролей из JWT

После успешной валидации токена Spring Security создает вспомогательный объект Authentication и помещает его в SecurityContextHolder, который доступен глобально в любом месте выполнения запроса.

Для удобного доступа к данным пользователя и его ролям, извлеченным из токена, используется абстракция getUserAuthentication над хранилищем SecurityContextHolder, позволяющая пребразовать его в удобный вспомогательный DTO класс UserAuthentication, хранящий информацию об идентификаторе и ролях пользователя.

2.8.7. Обработка ошибок и логгирование

В рамках платформы реализован функционал глобального логгирования вызовов метдов и обработки ошибок. Современные подходы к разработке для решение задач реализации сквозного фнукцилонала, затрагивающего большое колличество различных мест программы, но при этом реализующего схожий функционал предлагают внедрения AOP(Aspect Oriented Programming) [38] для решения такого рода задач.

Централизованная обработка ошибок REST API

Реализация единой обработки ошибок выполнена с использованием класса ErrorHandler. Аннотация @ControllerAdvice в Spring Boot позволяет классу перехватывать исключения, выброшенные из любого контроллера в приложении.

Класс ErrorHandler перехватывает все исключения возникающие в системе и производит над ними следующие действия:

— В лог записывается информация об ошибке, включая HTTP-метод, URI запроса, тип исключения и его сообщение.

— Создается объект RestError, содержащий стандартизированную структуру данных об ошибке: HTTP-статус, общее сообщение об ошибке и детальное сообщение.

— Сформированный RestError оборачивается в класс ResponseEntity с соответствующим HTTP-статусом. Это гарантирует, что клиент всегда получит ответ в предсказуемом JSON-формате, даже в случае непредвиденных сбоев на сервере.

Такой подход упрощает разработку клиентской части так как логика обработки ошибок стандартизирована.

Логирование выполнения методов с использованием АОП

Логирование является ключевым инструментом для отладки, мониторинга и анализа поведения системы. Однако, внедрение логирующих вызовов непосредственно в бизнес-логику методов может приводить к зашумлению кода и нарушению принципа единственной ответственности. Для решения этой проблемы реализован логгирующий аспект RestLoggingAspect.

Преимуществом данного подхода является декларативность. Достаточно пометить требующий логирования метод аннотацией @LogExecution, и аспект автоматически добавит необходимую логирующую обвязку без модификации исходного кода самого метода. Это значительно повышает читаемость и поддерживаемость кода бизнес-логики.

2.9. Разработка клиентской части

Для платформы развертывания контейнеризированных функций разработано клиентское веб-приложение. Оно позволяет обеспечить возаимодействие пользователя с автоматизированной системой.

Основной целью разработки разработки клиентского приложение является создание интуитивного, простого и понятного пользовательского интегрфейса (UI), позволяющего в полной мере использовать доступный функционал платформы.

2.9.1. Выбор стека технологий и инструментов

Для разработки пользовательского интерфейса выбран фреймворк Vue3. К плюсан выбранного фреймворка можно отнести поддержку реактиваности в интерфейсе, богатую экоситему библиотек, производительность и размер бандла скомпиллированного приложения.

Для управления состояниями клиентского приложения выбран стэйт-менеджер Pinia.

Pinia для фреймворка Vue3 поставлется как стейт-менеджер по умолчанию, что обеспечиват легкую интеграцию библиотеки с фреймворком, в то же время является самой удобной и гибкой реализаций менеджера состояний приложения среди аналогов.

К основным преимуществам Pinia можно отнести:

— официальная рекомендация для Vue 3;

— простота API, модульность;

— интеграция с Vue DevTools;

— поддержка TypeScript.

Для управления стилизацией HTML-элементов выбрана комбинация библиотек TailwindCSS и DaisyUI.

TailwindCSS реализует подход "utility-first". Вместо предопределенных классов компонентов, он предоставляет обширный набор низкоуровневых служебных классов, каждый из которых отвечает за одно конкретное CSS-свойство. Это позволяет разработчикам создавать полностью кастомные дизайны непосредственно в HTML-разметке, не покидая контекст и не создавая большие объемы пользовательского CSS. Такой подход обеспечивает максимальную гибкость в стилизации элементов интерфейса в точном соответствии с

требованиями дизайна. Использование TailwindCSS позволять уменьшить использование CSS в проекте а так же ускорят разработку.

DaisyUI функционирует как плагин для TailwindCSS, предоставляя набор готовых, стилизованных компонентов пользовательского интерфейса (кнопки, формы, карточки, модальные окна, оповещения и т.д.). Важно отметить, что эти компоненты построены с использованием утилит TailwindCSS. Это позволяет значительно ускорить разработку стандартных элементов UI, так как не требуется собирать их с нуля из отдельных утилит.

Использвоание DaisyUI дайет неоспоримое преимущество - консистентность дизайна интерфейса прижения, достигается минимальными усилиями разработчика. Таким образом выбор DaisyUI многократно ускоряет прототипирование и делает результат намного более качественным.

2.9.2. Архитектура клиентского приложения

Структурно компоненты клиентского приложения разделены на несколько групп, объединенных общей функциональностью.

К основным группам компонентов можно отнести следующие:

— View - компоненты отвечающие за отображение страниц приложения;

— Component - компоненты интерфейса, располагающиеся на страницах приложения;

— Api - сервисы, позволяющие взаимодействовать с серверной частью приложения;

— Store - модули ханилища состояния приложения, позволяющие различным компонентам приложения обмениваться информацией.

— Utils - утилитарные компоненты, позволяющие решать специфические задачи, такие как управление и настройка сетевых соединений, работа с сессиоными хранилищами, управление поллингом запросов и маршрутизацией.

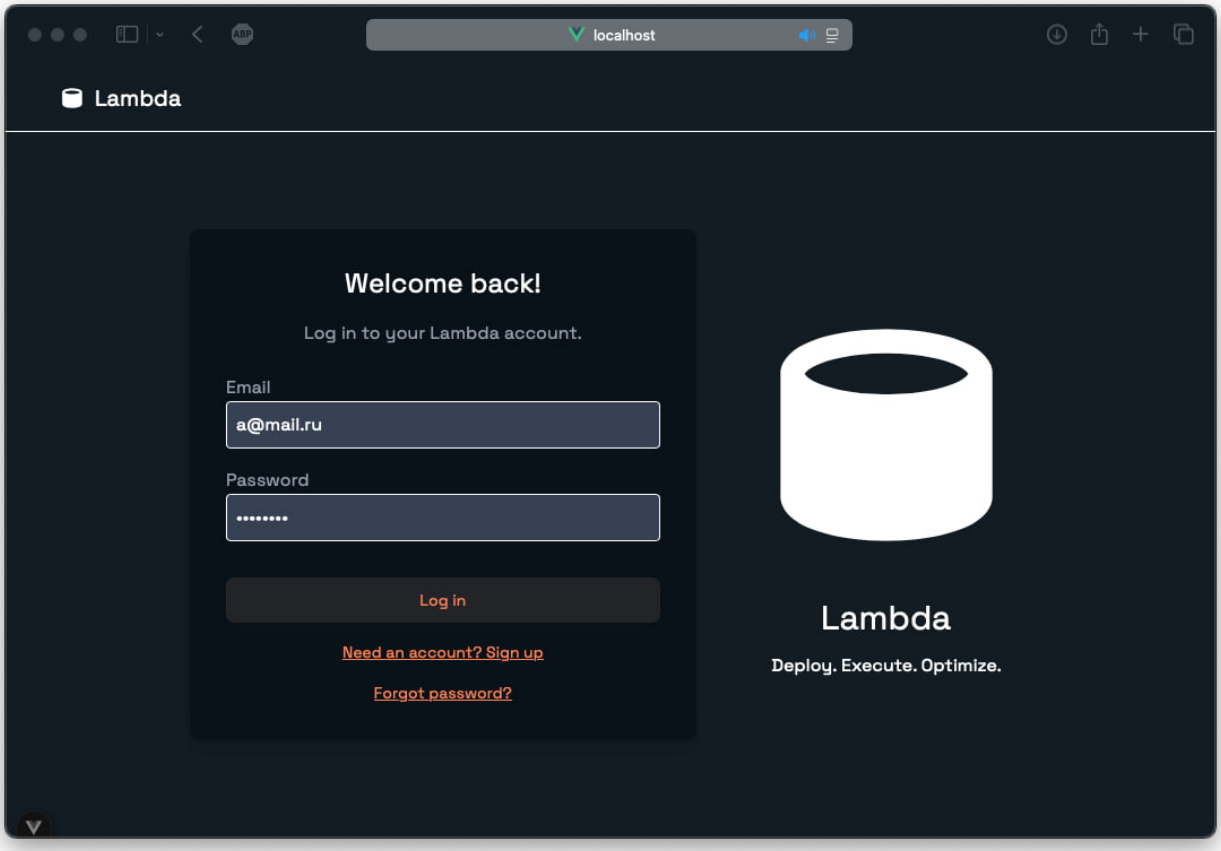

Рис. 2.16. Форма входа

2.9.3. Реализация ключевых пользовательских сценариев и компонентов

Клиентское приложение построено на основе компонентной архитектуры Vue.js, где каждый пользовательский сценарий реализуется через взаимодействие одного или нескольких компонентов, хранилищ состояния (Pinia) и сервисов для работы с API. Ниже рассмотрены основные сценарии и их реализация.

Аутентификация и Регистрация

Аутентификация и регистрация обеспечивают безопасный доступ пользователей к функциям платформы. В клиентской части эти процессы реализуются через набор компонентов представление LoginView содержит формы LoginForm, RegisterForm, ResetPasswordForm для ввода и валидации учетных данных. Компонент AuthSwitch реализует навигацию между входом и регистрацией.

Экран входа представлен на рисунке 2.16, а форма регистрации на ри-

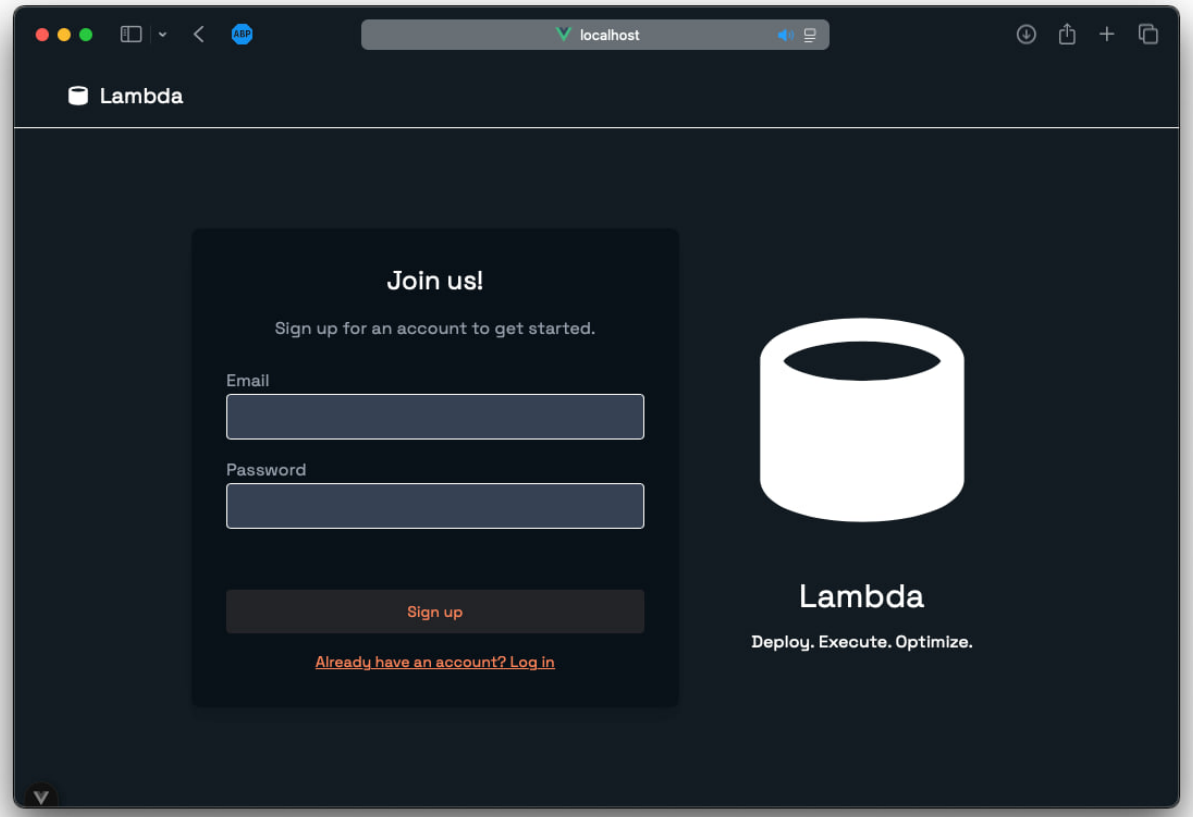

Рис. 2.17. Форма регистрации

сунке 2.17.

При нажатии кнопки войти или зарегистрироваться, введенные пользователем данные отправляются на соответствующую конечную точку. В случае успешной авторизации полученная пара JWT-токенов сохраняется в сессионное хранилище браузерasessionStorage посредством StorageService. Пользователь перенаправляется на стрницу tasks.

В последствии токен досупа пользователя используется при выполнении любых запросов к серверной части платформы. Если время действия токена доступа итекает, сетевой клиент Axios выполнят fallback обработку ошибки 401 полученной от бекенда сервиса и выпускает новый токен досутпа, обращаясь на соответствующий url с токеном refresh_token, который так же получает из сессионного хранилища.

В случае ошибки, напрмер если пользователем были введены неверные данные или выполнение запроса выло завершщено внутренней ошибкой сервера, информация обошибке, через систему уведомлений выводится на экран пользователя. Таким образом, система предоставляет пользователю обратную

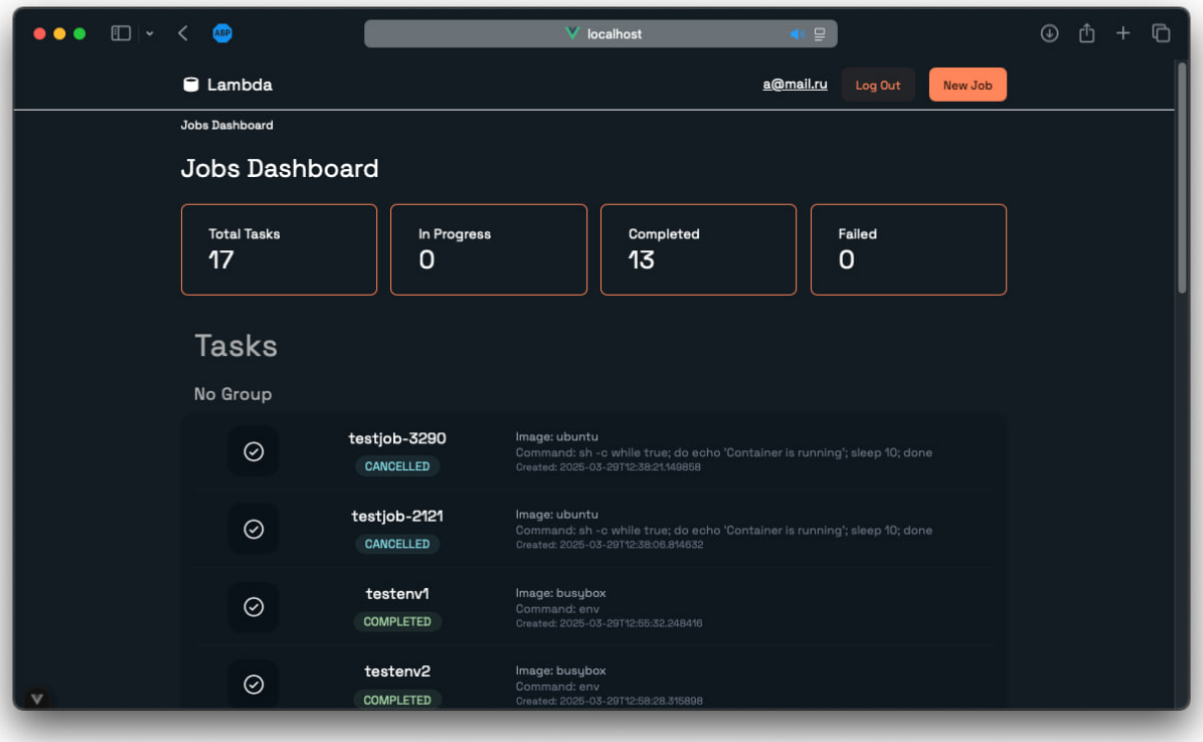

Рис. 2.18. Дашборд задач пользователя

связь и обеспечивает персистентность сессии через StorageService.ts.

Дашборд Задач

Основным интерфейсом для взаимодействия пользователя со списком задач является компонент-представление TasksView. При его инициализации (в хуке onMounted) [39] запускается процесс поллинга задач пользователя через компонет PolligService. Такая реализация позволят своевременно получать изменения статусов задач и получать новые созданные задачи без перезагрузок и каких либо действий со стороны пльзователя.

Дашборд задач представлен на рисунке 2.18.

Полученные данные сохраняются в модуле состояния tasksStore. Компонент TasksView передает этот список задач в дочерний компонент TasksList, который отвечает за их отображение. TasksList итерируется по списку, для каждой задачи создает экземпляр компонента TaskItem. Компонент TaskItem отображает ключевую информацию о задаче и использует вложенный компонент StatusBadge для визуализации ее текущего статуса.

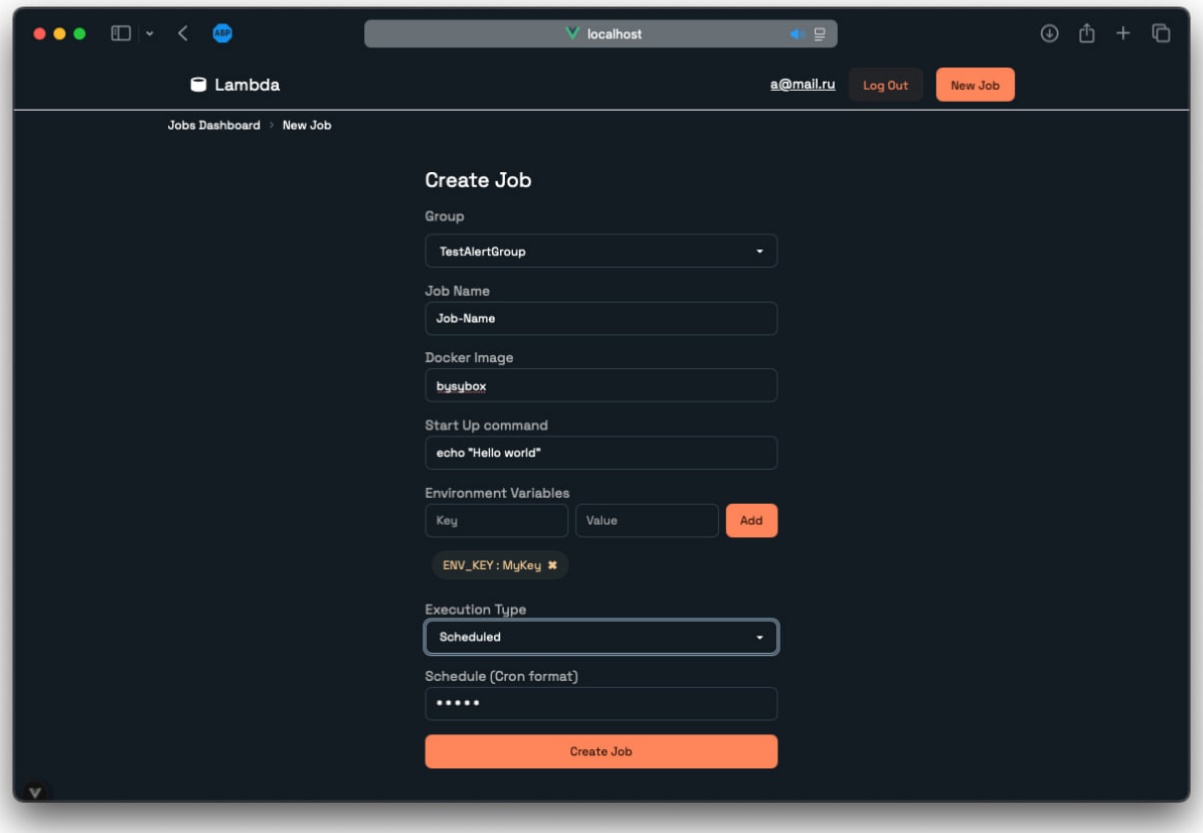

Рис. 2.19. Форма создания новой задачи

Создание Новой Задачи

Для создания новых задач предназначена страница CreateJobView, состоящая из формы ввода параметров: имени задачи, типа выполнения (однократный запуск, webhook, cron), Docker-образа, команды и переменных окружения.

Для задач типа CronJob предусмотрено поле ввода Cron-выражения, корректность которого проверяется утилитой CronValidator.ts.

Форма содания новой задачи представлена на рисунке 2.19.

После заполнения и клиентской валидации данных формы, CreateJobView инициирует действие выполняет запрос к серверной части приложения с параметрами задачи, заданными пользователем.

При успешном выполнении запроса сервер возвращает подтверждение об усешом создании задачи, пользователю отображается уведомление об успехе через alertStore.showSuccess с последующим перенаправлением на страницу соданной задачи. В случае ошибки API или серверной ошибки, пользователю выводится соответствующее сообщение через alertStore.showError.

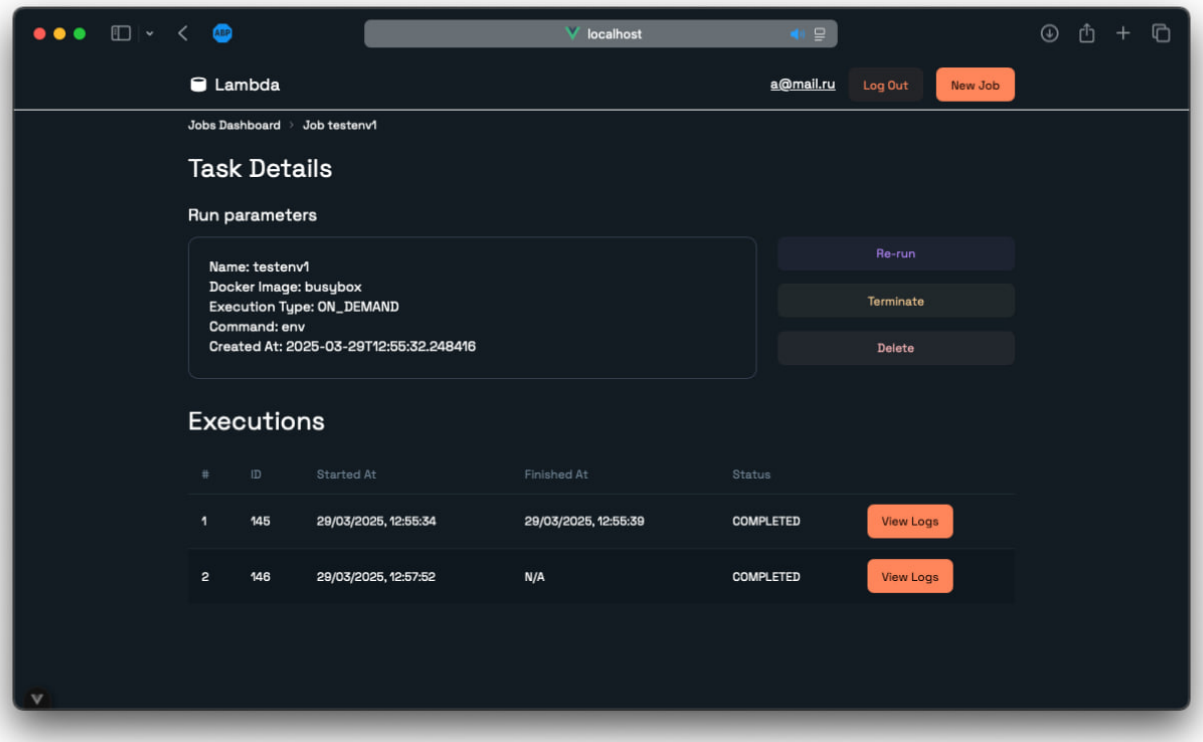

Рис. 2.20. Экран задачи

Просмотр Деталей Задачи

Для отображения подробной информации о конкретной задаче используется страница TaskView(Рис. 2.20). При переходе по роуту /task/:id компонент извлекает идентификатор задачи из URL. Дальше запускается поллиг сервис задачи, который с определенным интервалом совершает обращения к серверной части платформы для обновления информации по текущей задаче.

Полученные данные сохраняются в tasksStore и передаются дочерним компонентам: TaskDetails для отображения основной информации о задаче и TaskExecutions для вывода списка ее запусков со статусами и временными метками. При выборе конкретного запуска отображаются его логи в компоненте TaskLogs (Рис. 2.20).

Страница TaskView также предоставляет пользователю возможность выполнять действия над задачей, такие как перезапуск, отмена выполнения, удаление задачи, инициируя соответствующие вызовы API через JobsApi.

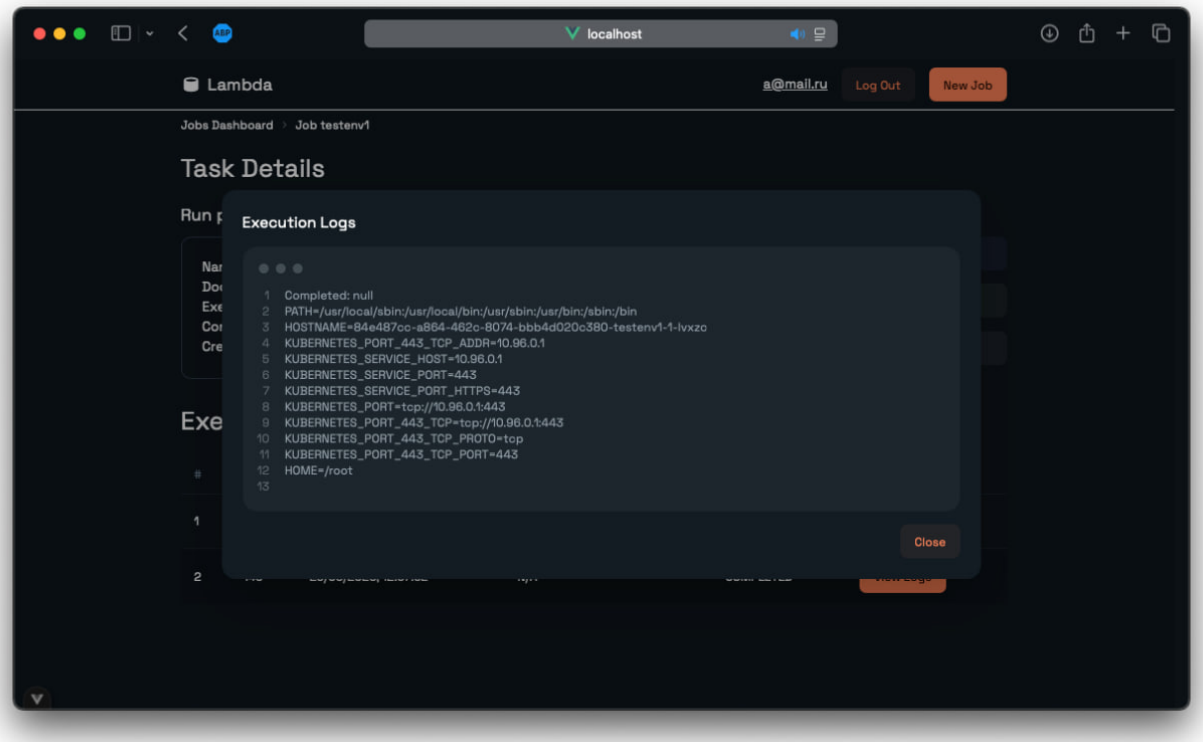

Рис. 2.21. Модальное окно просмотра логов

Управление Профилем и Группами

Раздел ProfileView(Рис 2.20) предоставляет пользователю инструменты для управления персональными данными и участием в рабочих группах. При инициализации страницы происходит загрузка информации о текущем пользователе через UserApi.ts с последующим поллингом информации о текущем пользователе и списка групп, в которых он состоит.

Данные пользователя отображаются в компоненте ProfileCard. Список доступных групп представлен в GroupSelection. Выбор конкретной группы инициирует загрузку списка ее участников и их доступов в рамках группы, которые отображается в MembersTable. Пользователям с соответствующими правами доступны функции администрирования группы: управление составом и правами доступа участников и генерация токенов приглашения в группу.

Реализованы также механизмы для создания новых групп и присоединения к существующим по токену досутпа. Управление токенами приглашений осуществляется через компонент AccessToken, который позволяет скопировать действующий токен в буфер обмена или сотозвать действующий токен и сгенерировать новый.

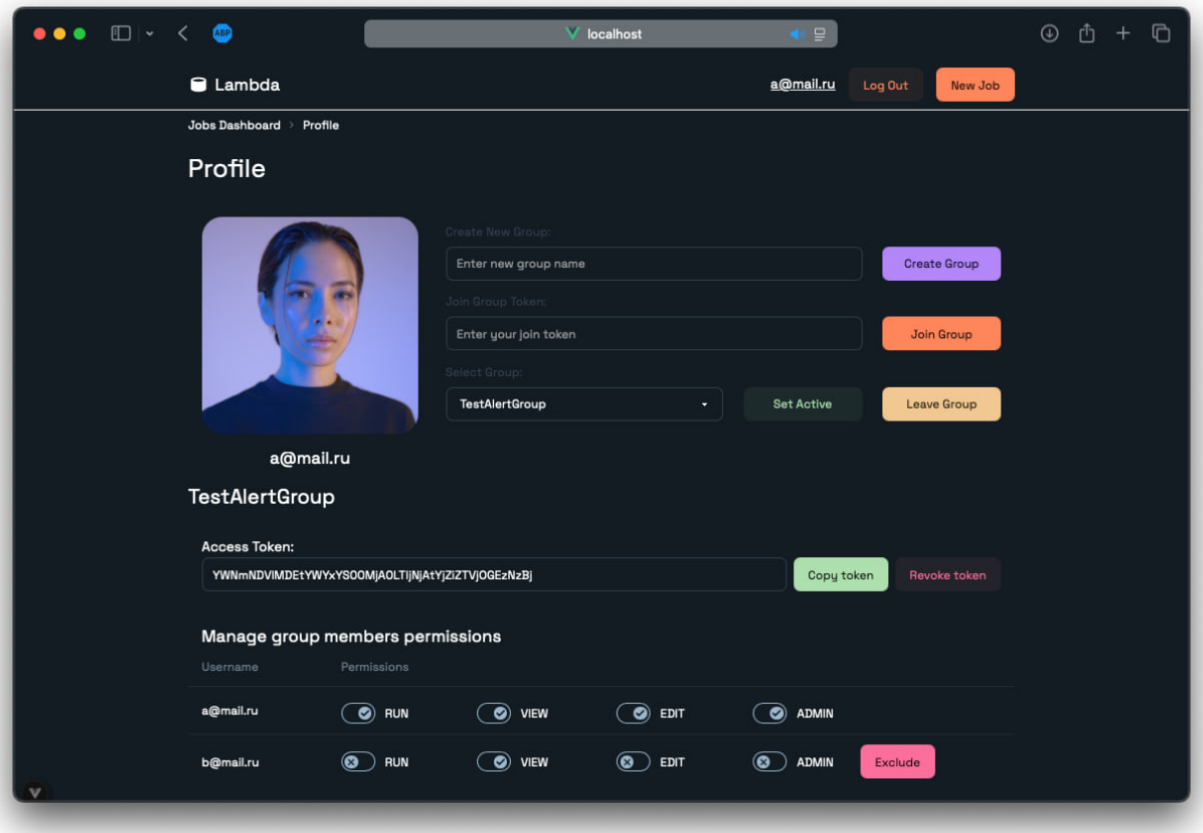

Рис. 2.22. Профиль пользователя

2.10. Вывод

В ходе технической реализации автоматизированной платформа развертывания контейнеризованных функций в среде Kubernetes были применены современные технологии и методы разработки, обеспечивающие эффективность, удобство использования и безопасность системы.

При разработке клиентской части веб-сервиса был использован язык программирования JavaScript с фреймворком Vue.js. Благодаря исопльзванию соверменных подходов к реализации пользовательского интерфейса обеспечено динамичное взаимодействие с пользователем и удобство в использовании.

Серверная часть платформы разработана с испольванием языка программирования Java и фреймворка Spring Boot для обработки запросов от пользователей, управления данными и взаимодействия с базой данных, сервером авторизации и оркестратором контейнеров.

База данных PostgreSQL использована для хранения информации о пользователях и запущенных задачах. Для работы с учетными записями пользо-

вателей использован сервер авторизации Keyloack. Он позволил реализовать авторизацию пользователей с использованием спецификации JWT, а так же предоставил механизмы для реализации гибкой ролевой моделли с динамическим набором групп и доступов.

Для развертывания приложения и обеспечения его доступности и масштабируемости был исопльзован Kubernetes, зарекомендовавший себя как надежное и удобное решение для управления контейнеризованными приложениями.

В результате разработки было создано современное решение, предоставляющее мощный и удобный пользовательский и программный интерфейс для управления распределенными контейнеризованными вычисления. Применение передовых технологий, удобный интерфейс делают этот сервис удобным инструментом для использования в большом колличестве прикладных задач.

3. ВНЕДРЕНИЕ И ЭКСПЛУАТАЦИЯ

3.1. Развертывание платформы в среде Kubernetes

Важным аспектом разработки информационной системы является правильное размещение готовго решения на целевой инфораструктуре. При выборе конфигурации серверного оборудования важно учитывать архитектурные особенности разработанной системы. В текущем разделе работы предствлены сведения об архитектуре развертывания приложения, требования к низлежащей инфраструктуре кластера Kubernetes, описание процесс установки основных компонентов платформы.

3.1.1. Диаграмма развертывания

Вся разработанная система развертывается в Kubernetes. Это обеспечивает отказоустойчивасть, масштабируемость, а так же возможность использования большого колличества серверов для развертывания пользовательских задач.

Высокоурованевая диаграмма развертывания компонентов системы представлена на рисунке 3.1.

Все ключевые компоненты системы разворачиваются в $platform - core$ нэймспейсе и каждый представляет собой отдельный Deployment компонент Kubernetes. В то же время пользовательские компоненты разворачиваются в изолированой среде $sandbox$ и являются объектами $V1Job$ и $V1CronJob$ в зависимости от конфигурации компонента.

Внешний пользовательский трафик поступает в систему через обратный прокси Nginx, который в тоже же время является SSL-терминатором [40] и корнем раздачи статических файлов.

3.1.2. Требования к инфраструктуре

Для успешного развертывания и функционирования автоматизированной платформы развертывания контейнеризованных функций необходимо обеспечить соответствие инфраструктуры следующим требованиям:

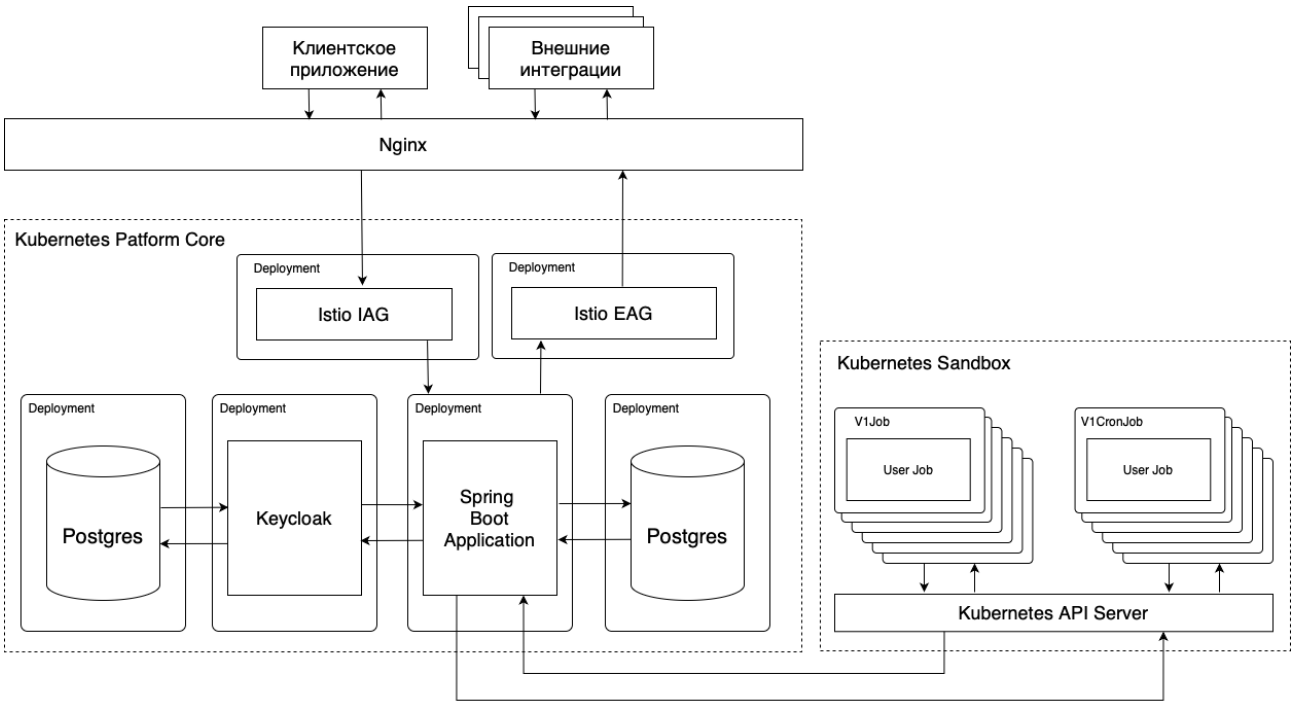

Рис. 3.1. Диаграмма развертывания

Кластер Kubernetes

Требуется наличие функционирующего кластера Kubernetes. Для использования всех возможностей и обеспечения совместимости с актуальными API рекомендуется версия 1.25 или выше.

Кластер должен иметь стандартную конфигурацию с работающими компонентами Control Plane (API Server, etcd, Scheduler, Controller Manager) и Worker Nodes (Kubelet, Kube-proxy, Container Runtime).

Серверное приложение платформы должно иметь права доступа к API Kubernetes для управления ресурсами (Jobs, CronJobs, Pods, Logs) в выделенном пространстве имен `sandbox`.

Ресурсы узлов кластера

Требования к ресурсам зависят от ожидаемой нагрузки: количества одновременно работающих пользовательских задач, их ресурсоемкости, а также активности пользователей платформы.

Минимальные требования к рабочему узлу для базового развертывания без значительной нагрузки:

— Backend: ~1 CPU, ~1-2 GiB RAM

— Keycloak: ~1 CPU, ~1-2 GiB RAM (может требовать больше при большой базе пользователей/сессий)

— PostgreSQL: ~1 CPU, ~1-2 GiB RAM (+ дисковое пространство)

— Пользовательские задачи (`sandbox`): Ресурсы зависят от самих задач. Необходимо предусмотреть достаточный запас на узлах для их запуска.

Необходимо проводить нагрузочное тестирование для определения оптимальных лимитов и запросов ресурсов (requests/limits) для подов платформы и планировать ресурсы узлов с запасом.

Доступ к образам контейнеров:

Узлы кластера Kubernetes должны иметь сетевой доступ к реестру контейнеров Docker Hub, где хранятся образы Backend, Frontend и базовые образы для пользовательских задач.

3.1.3. Процесс развертывания компонентов

Развертывание платформы осуществляется в среде Kubernetes и базируется на применении манифестов, поставляемых в составе релизного дистрибутива, описывающих требуемое состояние системы.

Процесс включает последовательное создание и настройку всех необходимых ресурсов Kubernetes для запуска основных компонентов платформы, таких как серверное приложение, система аутентификации, база данных, а также настройку сетевого взаимодействия и подготовку изолированного окружения (`sandbox`) для выполнения пользовательских задач.

Ниже описаны основные шаги, необходимые для развертывания каждого из ключевых компонентов.

Развертывание серверной части (Backend)

Развертывание серверной части, является ключевым шагом, так как этот компонент содержит основную бизнес-логику платформы. Процесс включает подготовку исполняемого артефакта, его контейнеризацию и последующее развертывание в кластере Kubernetes.

Сборка приложения и упаковка в Docker-образ.

Первым шагом является сборка приложения с использованием системы сборки Gradle. Команда сборки `./gradlew build` создает исполняемый JAR-файл в директории `build/libs`.

Далее, на основе этого JAR-файла и `Dockerfile`, создается Docker-образ приложения.

После создания образ загружается в реестр контейнеров, доступный для кластера Kubernetes. Этот процесс может быть выполен в ручную, так и автоматизирован в Ci/CD паплайне.

Подготовка Kubernetes-манифестов.

Для развертывания Backend в Kubernetes используются следующие основные типы манифестов:

— **Secret:** Используется для хранения чувствительных данных, таких как пароли и приватные ключи.

— **ConfigMap:** Используется для хранения нечувствительной конфигурации в виде пар ключ-значение.

— **Deployment:** Основной манифест, описывающий желаемое состояние для подов Backend приложения.

— **Service:** Определяет способ доступа к подам Backend внутри кластера.

Листинг 3.1. Манифест Deployment компонета

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-app
  namespace: platform-core
  labels:
    app: backend
spec:
  replicas: 2
  selector:
    matchLabels:
```

```
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      serviceAccountName: backend-service-account
      containers:
        - name: backend-container
          image: platform-registry/backend-app:latest
          ports:
            - containerPort: 8080
          envFrom:
            - configMapRef:
                name: backend-config
            - secretRef:
                name: backend-secrets
          resources:
            requests:
              memory: "1Gi"
              cpu: "500m"
            limits:
              memory: "2Gi"
              cpu: "1"
```

В листинге 3.1 предаствлен пример манифеста разворачиваемого Java приложения. Манифесты других компонентов идентичны и представлены в исходном коде разработанной системы.

Применение манифестов.

После подготовки всех YAML-файлов манифестов они применяются к кластеру Kubernetes с помощью команды:

```
kubectl apply -f <директория_с_манифестами> -n platform-core
```

Kubernetes создаст необходимые объекты, скачает Docker-образ и запустит поды Backend приложения в соответствии с описанием в Deployment.

После успешного запуска подов и прохождения readiness-проб, Service начнет направлять трафик на них. На этом этапе серверная часть готова к приему запросов от Frontend или других компонентов системы.

Развертывание клиентской части (Frontend)

Клиентская часть платформы, разработанная на Vue3, представляет собой одностраничное приложение (SPA), которое взаимодействует с пользователем и отправляет запросы к API серверной части.

Развертывание Frontend включает сборку статических ассетов и их размещение на веб-сервере, доступном для пользователей.

Исходный код Vue3 приложения компилируется в набор статических файлов (HTML, CSS, JavaScript) с помощью инструментов сборки, таких как Vite или Vue CLI. Сборка запускается командой `npm run build`.

Результат сборки (директория `dist`) содержит все необходимые файлы для работы приложения в браузере.

Для обслуживания статических файлов Frontend в Kubernetes использован веб-сервер Nginx, который раздает статические файлы потребителям.

Инициализация базы данных

После развертывания экземпляра PostgreSQL и перед полноценным запуском серверной части, необходимо инициализировать схему базы данных. Это включает создание таблиц, индексов, внешних ключей.

В разработанной системе для управления схемой базы данных используется инструмент миграций Liquibase. Этот инструмент интегрирован в Spring Boot приложение и позволяет версионировать изменения схемы БД и автоматически применять их при старте приложения.

Сервер авторизации Keycalok так же использует Liquibase для организации миграций базы данных.

Мигарции Liquibase поставляются вместе с дистрибутивом приложении и применяются при его запуске.

При последующих запусках пода Backend или при развертывании новых версий приложения с новыми скриптами миграций Liquibase будет применять только те миграции, которые еще не были выполнены, обеспечивая консистентность схемы БД.

3.2. Реализация нефункциональных требований

Помимо реализации заявленной функциональности, описанной в предыдущих разделах, для создания надежной и эффективной системы критически важно уделить внимание нефункциональным требованиям (НФТ).

В рамках разработки особое внимание было уделено трем нефункциональным требованиям: безопасности, масштабируемости и отказоустойчивости. Выбор архитектурных решений, стека технологий и конкретных механизмов реализации был во многом продиктован необходимостью обеспечения этих качеств.

3.2.1. Обеспечение безопасности

Безопасность крайне важна, поскольку система предполагает выполнение произвольного пользовательского кода, управляет доступом к ресурсам и обрабатывает пользовательские данные. В данном подразделе рассматриваются механизмы и решения, внедренные для минимизации рисков рисков безопасности.

Аутентификация и авторизация

Для реализации механизмов аутентификации и авторизации используется внешний сервер идентификации Keycloak, интегрированный с серверным приложением посредством стандартов OAuth 2.0 и OpenID Connect. Backend, выступая в роли Resource Server, отвечает за валидацию JWT-токенов доступа, получаемых от Keycloak, извлечение информации о пользователе и его ролях, а также за принудительное применение правил авторизации на уровне API эндпоинтов и отдельных операций с использованием возможностей фреймворка Spring Security.

Использование готовых решений и протоколов минимизирует риски возникновения ошибок при реализации механизмов аутентификации.

Изоляция выполнения задач

Поскольку платформа предназначена для запуска ппроизвольного пользовательского кода, критически важно предотвратить любое нежелательное

взаимодействие этого кода с компонентами ядра системы или с кодом других пользователей. Для достижения этой цели реализован механизм строгой изоляции на уровне оркестратора Kubernetes, ключевым элементом которого является выделенное, логически обособленное пространство имен.

Все пользовательские задания, создаваемые платформой в виде объектов `Job` или `CronJob`, выполняются исключительно в пределах этого изолированного окружения, что фундаментально ограничивает их область видимости и потенциальное воздействие на остальную инфраструктуру кластера.

3.2.2. Масштабируемость и производительность

Способность платформы эффективно обрабатывать возрастающие нагрузки и обеспечивать быстрый отклик API являются ключевыми нефункциональными требованиями для ее успешной эксплуатации.

Использование Kubernetes в качестве среды развертывания предоставляет возможности для горизонтального масштабирования как компонентов самой платформы, так и ресурсов, выделяемых для выполнения пользовательских задач, позволяя динамически адаптироваться к изменениям нагрузки и обеспечивать стабильную работу системы.

Горизонтальное масштабирование компонентов

Серверная часть является stateless приложением и развертывается с использованием объекта Deployment. Этот объект позволяет декларативно управлять количеством работающих экземпляров каждого компонента через параметр `spec.replicas`.

При возрастании нагрузки администратор кластера может вручную увеличить количество реплик для Backend и других компонентов платформы, и Kubernetes автоматически запустит дополнительные поды, а Service (или Ingress) обеспечит распределение входящих запросов между всеми доступными экземплярами, повышая таким образом пропускную способность и отказоустойчивость сервиса.

3.3. Тестирование платформы

Тестирование позволяет выявить ошибки, узкие места и несоответствия на ранних стадиях, что существенно снижает затраты на их исправление и повышает общее качество конечного продукта.

На текущем этапе проект находится в стадии активного прототипирования и разработки. В связи с этим, API платформы и внутренняя структура компонентов подвержены частым изменениям. По этой причине было принято решение временно отказаться от реализации полного покрытия кода модульными (unit) и интеграционными тестами. В условиях нестабильного API и быстрого развития функциональности, затраты на постоянную поддержку тестовой базы превысили бы получаемую пользу, замедляя процесс разработки прототипа.

Было проведено ручное функциональное тестирование для проверки корректности работы ключевых аспектов платформы. Это тестирование охватывало как прямое взаимодействие с программным интерфейсом (API Testing), так и проверку пользовательских сценариев через веб-интерфей, что позволило убедиться в работоспособности основных функций и интеграции между компонентами системы. Для тестирования API использовались инструменты, такие как Postman, позволяющие формировать HTTP-запросы к эндпоинтам и анализировать ответы, в то время как E2E-тестирование выполнялось непосредственно в веб-браузере.

3.4. Выводы

В данной главе было представлено описание процессов внедрения и эксплуатации разработанной систмеы. Были подробно рассмотрено влияние принятых архитектурных решений на портребильские характеристики и процессы эксплуатации продукта.

Архитектура платформы построена на основе многоуровневой модели с четким разделением ответственности. Использование Kubernetes в качестве центрального элемента для оркестрации задач и развертывания компонентов, Nginx в качестве точки входа и обратного прокси, Keycloak для централизованной аутентификации и авторизации, а также PostgreSQL для персистентного хранения данных, формирует надежную и современную основу.

Логическое разделение на пространство имен ядра платформы (platform-core) и изолированное окружение для пользовательских задач (sandbox) является ключевым архитектурным решением, повышающим безопасность и управляемость системы.

Выбранный технологический стек, включающий Kotlin и Spring Boot для Backend, Vue3 для Frontend, соответствует требованиям к производительности, удобству разработки и интеграции с экосистемой Kubernetes.

Данная глава продемонстрировала, что разработанная платформа обладает продуманной архитектурой и реализована с использованием современных технологий, обеспечивающих необходимую функциональность, безопасность и масштабируемость. Были детально описаны механизмы взаимодействия с Kubernetes, Keycloak и базой данных. Рассмотрены подходы к обеспечению нефункциональных требований, таких как горизонтальное масштабирование, а также изоляция пользовательских задач. Проведенное на этапе прототипирования функциональное тестирование подтвердило работоспособность основных сценариев использования. Представленная техническая реализация создает прочную основу для дальнейшего развития и внедрения платформы.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной выпускной квалификационной работы ожидаемые результаты были достигнуты. Был разработан и реализован действующий прототип автоматизированной платформы для развертывания контейнеризованных функций в среде Kubernetes. Созданная платформа способна существенно упростить процессы разработки, тестирования и эксплуатации приложений, предоставляя разработчикам управляемую, безопасную и масштабируемую среду для выполнения их вычислительных задач.

В процессе работы над платформой были реализованы ключевые функциональные возможности, такие как управление жизненным циклом задач, централизованная аутентификация и авторизация пользователей с использованием Keycloak и гранулярной ролевой модели на основе групп.

Реализованный функционал позволяет достичь поставленных целей по автоматизации развертывания, обеспечению изоляции выполнения пользовательского кода и контролю доступа к ресурсам.

В процессе работы были решены следующие основные задачи:

- Проведено исследование предметной области, включая технологии контейнерной оркестрации, а также существующие подходы к реализации платформ FaaS и автоматизации развертывания.
- Осуществлен анализ требований к разрабатываемой платформе, определены ключевые сущности, пользовательские роли и основные сценарии использования.
- Спроектирована архитектура системы, выбран технологический стек, определены компоненты системы и их взаимодействие, разработана спецификация API и схема базы данных.
- Разработана серверная часть и клиентская части платформы.
- Настроена среда развертывания в Kubernetes, подготовлены необходимые манифесты и конфигурационные файлы.
- Проведено функциональное тестирование разработанного прототипа для проверки работоспособности основных функций.

Тем не менее, работа над платформой не завершена, и существует потенциал для ее дальнейшего развития. На следующих этапах необходимо расширение функциональности, внедрение модульного и интеграционного тестирования, реализацию более гибких механизмов управления задачами, интеграция с системами непрерывной интеграции и доставки (CI/CD), внедрение механизмов квотирования ресурсов и детального мониторинга.

Исходя из вышеизложенного, можно заключить, что цель данной выпускной квалификационной работы — разработка прототипа автоматизированной платформы развертывания контейнеризованных функций — достигнута, а поставленные задачи успешно решены на текущем этапе, создав основу для дальнейшего развития продукта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Roberts M., Chapin J. What is Serverless? O'Reilly Media, Incorporated. 2017.
2. Elsten J. M. Exploring the potential use of FaaS within an iPaaS infrastruc¨ ture. : Master's thesis / Julian M Elsten ; University of Twente. 2023.
3. Pu Q., Venkataraman S., Stoica I. Shuffling, fast and slow: Scalable analytics on serverless infrastructure // 16th USENIX symposium on networked systems design and implementation (NSDI 19). 2019. P. 193–206.
4. Lowber P. Thin-client vs. fat-client tco // research note, Gartner. 2001.
5. Kokkonen J. Single-page application frameworks in enterprise software devel¨ opment. 2015.
6. Faaslight: General application-level cold-start latency optimization for func¨ tion-as-a-service in serverless computing / Xuanzhe Liu, Jinfeng Wen, Zhen¨ peng Chen et al. // ACM Transactions on Software Engineering and Method¨ ology. 2023. Vol. 32. No. 5. P. 1–29.
7. Silva P., Fireman D., Pereira T. E. Prebaking functions to warm the serverless cold start // Proceedings of the 21st International Middleware Conference. 2020. P. 1–13.
8. Initiative O. C. Open container initiative image format specification.
9. Bretthauer D. Open source software: A history. 2001.
10. Sommerlad P. Reverse proxy patterns. // EuroPLoP / Citeseer. 2003. P. 431–458.
11. Raje S. N. Performance comparison of message queue methods : Master's thesis / Sanika N Raje ; University of Nevada, Las Vegas. 2019.
12. Dürr K., Lichtenthäler R., Wirtz G. An evaluation of saga pattern implemen¨ tation technologies. // ZEUS. 2021. P. 74–82.
13. Hohpe G., Woolf B. Enterprise integration patterns: Designing, building, and deploying messaging solutions. Addison-Wesley Professional. 2004.
14. Menge F. Enterprise service bus // Free and open source software conference. Vol. 2. 2007. P. 1–6.
15. Newman S. Monolith to microservices: evolutionary patterns to transform your monolith. O'Reilly Media. 2019.
16. A brief survey of software architecture concepts and service oriented architec¨ ture / Mohammad Hadi Valipour, Bavar AmirZafari, Khashayar Niki Maleki,

Negin Daneshpour // 2009 2nd IEEE International Conference on Computer Science and Information Technology / IEEE. 2009. P. 34–38.
17. Evans E. Domain-driven design: tackling complexity in the heart of software. Addison-Wesley Professional. 2004.
18. Ratner I. M., Harvey J. Vertical slicing: Smaller is better // 2011 Agile Con¨ ference / IEEE. 2011. P. 240–245.
19. Samuel S., Bocutiu S. Programming kotlin. Packt Publishing Ltd. 2017.
20. The materials application programming interface (api): A simple, flexible and efficient api for materials data based on representational state transfer (rest) principles / Shyue Ping Ong, Shreyas Cholia, Anubhav Jain et al. // Computational Materials Science. 2015. Vol. 97. P. 209–215.
21. Biehl M. Webhooks–Events for RESTful APIs. API-University Press. 2017. Vol. 4.
22. Wilde E., Pautasso C. REST: from research to practice. Springer Science & Business Media. 2011.
23. Box D., Ehnebuske D., Kakivaya G. et al. Simple object access protocol (soap) 1.1. 2000.
24. Rpc: Remote procedure call protocol specification version 2 : Rep. ; Executor: Raj Srinivasan : 1995.
25. Ahmed S., Mahmood Q. An authentication based scheme for applications us¨ ing json web token // 2019 22nd international multitopic conference (INMIC) / IEEE. 2019. P. 1–6.
26. From the internet of things to the web of things: Resource-oriented architec¨ ture and best practices / Dominique Guinard, Vlad Trifa, Friedemann Mattern, Erik Wilde // Architecting the Internet of things. 2011. P. 97–129.
27. Boyd R. Getting started with OAuth 2.0. ” O’Reilly Media, Inc.”. 2012.
28. Richardson K. Microservices. Patterns of development and refactoring. Piter. 2023.
29. Salvadori I., Siqueira F. A maturity model for semantic restful web apis // 2015 IEEE International Conference on Web Services / IEEE. 2015. P. 703–710.
30. O’Neil E. J. Object/relational mapping 2008: hibernate and the entity data model (edm) // Proceedings of the 2008 ACM SIGMOD international confer¨ ence on Management of data. 2008. P. 1351–1356.
31. Yang D. Java persistence with JPA. Outskirts Press. 2010.

32. Mernik M., Heering J., Sloane A. M. When and how to develop domain-spe¨ cific languages // ACM computing surveys (CSUR). 2005. Vol. 37. No. 4. P. 316–344.
33. Poulton N. The kubernetes book. NIGEL POULTON LTD. 2023.
34. Cooper J. W. Java design patterns: a tutorial. 2000.
35. Ferry E., O Raw J., Curran K. Security evaluation of the oauth 2.0 frame¨ work // Information & Computer Security. 2015. Vol. 23. No. 1. P. 73–101.
36. Arai K., Norikoshi K., Oda M. Method resource sharing in on-premises envi¨ ronment based on cross-origin resource sharing and its application for safety-¨ first constructions. // International Journal of Advanced Computer Science & Applications. 2024. Vol. 15. No. 5.
37. Barth A., Jackson C., Mitchell J. C. Robust defenses for cross-site request forgery // Proceedings of the 15th ACM conference on Computer and com¨ munications security. 2008. P. 75–88.
38. Aspect-oriented programming / Gregor Kiczales, John Lamping, Anurag Mendhekar et al. // ECOOP'97—Object-Oriented Programming: 11th European Conference Jyväskylä, Finland, June 9–13, 1997 Proceedings 11 / Springer. 1997. P. 220–242.
39. Tikhonova A. Design and implementation of reusable component for vue. js. 2021.
40. Boisrond P. To terminate or not to terminate secure sockets layer (ssl) traffic at the load balancer // arXiv preprint arXiv:2011.09621. 2020.

ПРИЛОЖЕНИЕ А