**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

**Институт (Филиал)** **№ 8 «Компьютерные науки и прикладная математика**» **Кафедра \_806\_**

**Группа М8О-406Б-21 Направление подготовки 01.03.02 Прикладная математика и информатика**

**Профиль Информатика**

**Квалификация бакалавр**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

на тему: «Серверная компонента приложения для управления миграциями в реляционных базах данных»

Автор ВКРБ: Чапкин Владислав Вячеславович (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Руководитель: Лукин Владимир Николаевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант:  (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Консультант:  (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Рецензент:  (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

**К защите допустить**

Заведующий кафедрой № 806 «Вычислительная математика  
и программирование» Крылов Сергей Сергеевич (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

Москва 2025

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

**Институт (Филиал)** **№ 8 «Компьютерные науки и прикладная математика**» **Кафедра \_806\_**

**Группа М8О-406Б-21 Направление подготовки 01.03.02 Прикладная математика и информатика**

**Профиль Информатика**

**Квалификация бакалавр**

**УТВЕРЖДАЮ**

Заведующий кафедрой 806 Крылов С.С. (№ каф.) (подпись) (инициалы, фамилия)

\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_ 20 г.

**ЗАДАНИЕ**

**на выпускную квалификационную работу бакалавра**

**Обучающийся** Чапкин Владислав Вячеславович

(фамилия, имя, отчество полностью)

**Руководитель** Лукин Владимир Николаевич

(фамилия, имя, отчество полностью

кандидат физико-математических наук, доцент, доцент 806 кафедры МАИ

ученая степень, ученое звание, должность и место работы)

**1. Наименование темы**  Серверная компонента приложения для управления миграциями в реляционных базах данных

**2. Срок сдачи обучающимся законченной работы** 24.05.2025

**3. Задание и исходные данные к работе**

Разработать серверную компоненту приложения для управления миграциями в реляционных базах данных. Спроектировать структуру серверной компоненты, обеспечивающую надежное выполнение операций с миграциями. Реализовать интерфейс для взаимодействия с компонентой через API.

**Перечень иллюстративно-графических материалов\***при наличии**:**

|  |  |  |
| --- | --- | --- |
| **№ п/п** | **Наименование** | **Количество листов** |
| 1 | Архитектура серверной компоненты приложения | 1 |
| 2 | Диаграмма работы API для управления миграциями | 1 |
| 3 | Пример логов выполнения миграций | 1 |
| 4 | Характеристика и демонстрация результатов разработки | 1 |

**4. Перечень подлежащих разработке разделов и этапы выполнения работы**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| п/п | Наименование раздела или этапа | Трудоёмкость в % от полной трудоёмкости ВКРБ | Срок выполнения | Примечание |
| 1 | Анализ предметной области и необходимого функционала | 15 | 23.02.2025 |  |
| 2 | Разработка архитектуры | 15 | 13.03.2025 |  |
| 3 | Разработка сервиса | 45 | 29.04.2025 |  |
| 4 | Введение | 15 | 09.05.2025 |  |
| 5 | Заключение (краткие выводы и перспективы дальнейшей разработки темы) | 10 | 24.05.2025 |  |

**5. Исходные материалы и пособия**

* Мартин Клеппман Проектирование высоконагруженных приложений. - СПб.: 2018
* Clean Architecture // by Robert C. Martin URL: <https://blog.cleancoder.com/uncle-bob/2011/11/22/Clean-Architecture.html>
* Митра Ронни, Надареишвили Иракли Микросервисы. От архитектуры до релиза. - СПб.: 2023

**6. Дата выдачи задания** 10.02.2025

Руководитель Лукин В.Н.

(подпись)

Задание принял к исполнению Чапкин В. В.

(подпись)

РЕФЕРАТ

**Содержание**

[РЕФЕРАТ 5](#_Toc195386211)

[ВВЕДЕНИЕ 8](#_Toc195386212)

[1. АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ 10](#_Toc195386213)

[1.1. Понятие миграций в реляционных базах данных 10](#_Toc195386214)

[1.2. Анализ популярных инструментов для управления миграциями 10](#_Toc195386215)

[1.3. Проблемы существующих инструментов 12](#_Toc195386216)

[1.4. Постановка требований к разрабатываемому приложению 13](#_Toc195386217)

[2. ВЫБОР АРХИТЕКТУРЫ И ПРОЕКТИРОВАНИЕ СИСТЕМЫ 15](#_Toc195386218)

[2.1. Обоснование выбора архитектуры 15](#_Toc195386219)

[2.2. Общая структура микросервисов 16](#_Toc195386220)

[2.3. Проектирование архитектуры межсервисного взаимодействия 16](#_Toc195386221)

[2.4. Проектирование структуры базы данных 18](#_Toc195386222)

[2.5. Разработка интерфейса взаимодействия (API) 23](#_Toc195386223)

[2.5. Используемые инструменты 25](#_Toc195386224)

[3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ 27](#_Toc195386225)

[3.1. Общие подходы к реализации 27](#_Toc195386226)

[3.2. Реализация модулей приложения 27](#_Toc195386227)

[3.2.2. Модуль взаимодействия с базами данных 28](#_Toc195386228)

[3.2.3. Модуль REST API 28](#_Toc195386229)

[3.2.4. Модуль логирования 29](#_Toc195386230)

[3.3. Подходы к тестированию системы 29](#_Toc195386231)

[3.3.1. Тестирование функциональности модулей 30](#_Toc195386232)

[3.3.2. Инструменты для тестирования 31](#_Toc195386233)

[3.4. Обработка ошибок и отладка 31](#_Toc195386234)

[3.5. Документация API 32](#_Toc195386235)

[3.6. Инструкция по запуску и использованию приложения 32](#_Toc195386236)

[3.7. Рекомендации по развертыванию 33](#_Toc195386237)

[ЗАКЛЮЧЕНИЕ 34](#_Toc195386238)

[1. Итоги работы 34](#_Toc195386239)

[2. Результаты исследования 34](#_Toc195386240)

[3. Перспективы дальнейшего развития 34](#_Toc195386241)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 35](#_Toc195386242)

[ПРИЛОЖЕНИЯ 36](#_Toc195386243)

# ВВЕДЕНИЕ

Разработка программного обеспечения требует гибкого и эффективного управления изменениями в структуре баз данных. При работе с реляционными базами данных (РБД) возникает необходимость модификации схемы данных. Например, добавление новых таблиц или изменение структуры существующих. Эти изменения, называемые миграциями, играют ключевую роль в процессе разработки и сопровождения приложений.

Отсутствие централизованных инструментов для управления миграциями приводит к ряду проблем: несогласованность данных между средами разработки, трудности в откате изменений и отсутствию прозрачности в применяемых операциях. Для решения этих задач используются специализированные инструменты и приложения, однако существующие решения, такие как Migrate и Goose, обладают рядом ограничений: недостаточная автоматизация, отсутствие готового API для интеграции и минимальные возможности мониторинга статуса миграций.

Актуальность разработки заключается в создании серверной компоненты приложения, которая позволит централизованно управлять миграциями в реляционных базах данных, а также предоставит разработчикам и администраторам удобный интерфейс для отслеживания их статуса.

Цель дипломной работы – разработка серверного приложения для просмотра и управления миграциями в реляционных базах данных с использованием современных подходов проектирования и инструментов.

Для достижения цели были поставлены следующие задачи:

* Анализ существующих решений для управления миграциями.
* Выбор архитектуры серверного приложения.
* Проектирование структуры базы данных для хранения информации о миграциях.
* Реализация модулей для работы с миграциями и авторизацией.
* Разработка API для взаимодействия с приложением.
* Тестирование и отладка приложения.
* Подготовка документации и инструкции по использованию приложения.

Объект исследования – процесс управления миграциями в реляционных базах данных, предмет – серверная компонента приложения, реализующая процесс управления миграциями.

В работе используются современные технологии, такие как Golang для разработки, Docker для контейнеризации, а также база данных PostgreSQL. При проектировании системы применяется микросервисная архитектура с использованием принципов чистой архитектуры, что обеспечивает модульность и масштабируемость решения.

Структура работы состоит из пяти глав, включающих анализ аналогов, проектирование и реализацию системы, ее тестирование, а также подготовку документации для конечных пользователей.

Таким образом, разработка серверной компоненты представляет собой актуальную и практическую задачу, решение которой позволит упростить процесс управления миграциями, обеспечить их прозрачность и автоматизацию, а также предоставить удобное API для дальнейшей интеграции с другими системами.

1. АНАЛИЗ СУЩЕСТВУЮЩИХ РЕШЕНИЙ

1.1. Понятие миграций в реляционных базах данных

Миграции в реляционных базах данных представляют собой управляемый процесс изменения структуры базы данных. Этот процесс включает создание, модификацию или удаление таблиц, индексов, связей и других элементов схемы базы данных. Миграции позволяют разработчикам синхронизировать изменения между различными средами разработки, тестирования и продакшн-окружением, обеспечивая консистентность данных и облегчая процесс развертывания обновлений.

Основные задачи миграций:

* Обновление схемы базы данных с добавлением новых таблиц и полей.
* Модификация существующих структур данных (переименование, изменение типа данных и т.д.).
* Откат изменений для возврата базы данных к предыдущему состоянию.
* Отслеживание истории изменений и их последовательности.

Процесс миграций особенно актуален для современных систем с непрерывной интеграцией и доставкой (CI/CD), где изменения в базе данных должны автоматически синхронизироваться с кодом приложения. Без надлежащего управления миграциями возникает риск несоответствия структуры базы данных и логики приложения, что может привести к сбоям в работе системы.

1.2. Анализ популярных инструментов для управления миграциями

В настоящее время существует ряд инструментов и библиотек, которые помогают управлять миграциями в реляционных базах данных. Рассмотрим два наиболее популярных решения на языке программирования Golang: Migrate и Goose.

Migrate  
Migrate – это открытая библиотека для управления миграциями, написанная на Go. Она предоставляет базовый функционал для создания, применения и отката миграций.

Преимущества:

Поддержка нескольких баз данных: PostgreSQL, MySQL, SQLite, SQL Server и другие.

Гибкость хранения миграций: файлы миграций могут находиться в файловой системе, а также в облачных хранилищах (S3, GCS).

Инструмент командной строки: Migrate предоставляет CLI для выполнения миграций вне основного кода приложения.

Недостатки:

Ограниченный функционал: библиотека ориентирована на минимальные операции с миграциями (применение, откат) и не предоставляет дополнительных возможностей для мониторинга и управления.

Отсутствие готового API: Migrate не предлагает API для автоматизированного управления миграциями.

Минимальная отчетность: система не предоставляет встроенных механизмов для отслеживания статуса и логирования миграций.

Goose  
Goose – еще один инструмент для управления миграциями на языке Go. Его основное преимущество – поддержка как SQL-скриптов, так и программного кода на Go для выполнения миграций.

Преимущества:

Гибкость в написании миграций: поддержка миграций на чистом SQL и в виде кода на Go.

Командная строка: Goose имеет CLI для удобного управления миграциями вручную.

Широкая поддержка баз данных: PostgreSQL, MySQL, SQLite и другие СУБД.

Легкость интеграции: Goose легко встраивается в существующие проекты на Go.

Недостатки:

Ограниченная автоматизация: отсутствие готового REST API ограничивает возможности интеграции с другими сервисами.

Слабый мониторинг: Goose не предоставляет встроенных инструментов для отслеживания состояния миграций или ведения отчетности.

Ограниченная поддержка сложных сценариев: инструмент не рассчитан на сложные схемы управления миграциями с аналитикой и отчетами.

1.3. Проблемы существующих инструментов

На основе анализа инструментов Migrate и Goose выявлены следующие проблемы и ограничения существующих решений для управления миграциями:

Отсутствие централизованного API: Большинство инструментов не предоставляют REST или gRPC API для удаленного управления миграциями, что усложняет интеграцию с другими системами.

Минимальные возможности мониторинга: Существующие решения не позволяют отслеживать состояние миграций в реальном времени или получать подробные отчеты по истории изменений.

Ограниченная автоматизация: Большинство инструментов рассчитаны на ручное управление миграциями с использованием командной строки и не поддерживают автоматическое управление в крупных системах.

Логирование и отчетность: Недостаток встроенных инструментов для логирования операций и ведения истории миграций.

Эти проблемы особенно актуальны для крупных проектов, где изменения в базах данных происходят регулярно, а управление миграциями должно быть прозрачным и автоматизированным.

1.4. Постановка требований к разрабатываемому приложению

На основании анализа существующих решений и выявленных проблем сформулированы ключевые требования к разрабатываемому приложению:

Функциональные требования:

* Возможность создавать, применять и откатывать миграции в реляционных базах данных.
* Предоставление информации о статусе миграций (выполненные, откатанные, ожидающие выполнения).
* Поддержка СУБД PostgreSQL.
* Хранение истории миграций для их анализа и аудита.

API и взаимодействие:

* Разработка REST API для автоматизированного управления миграциями.
* Поддержка основных методов: запуск миграций, откат миграций, просмотр статуса.
* Формат запросов и ответов – JSON.

Требования к безопасности:

* Внедрение аутентификации и авторизации (например, JWT).
* Ограничение доступа к данным миграций на основе ролей и политик безопасности.
* Производительность и отказоустойчивость:
* Поддержка работы приложения под высокой нагрузкой.
* Возможность горизонтального масштабирования сервиса.

Дополнительные возможности:

* Интеграция с инструментами контейнеризации (Docker).
* Логирование всех операций.
* Документация API с использованием Swagger.

Таким образом, был проведен анализ существующих решений для управления миграциями, выявлены их ограничения и сформулированы требования к разрабатываемому приложению. Полученные выводы использованы для проектирования и реализации серверной компоненты, которая устраняет недостатки аналогов и предоставляет современный инструмент для управления миграциями в реляционных базах данных.

2. ВЫБОР АРХИТЕКТУРЫ И ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Обоснование выбора архитектуры

Для реализации серверной компоненты приложения для управления миграциями в реляционных базах данных было выбрано микросервисное архитектурное решение с применением принципов чистой архитектуры (*Clean Architecture*). Описанный подход позволяет разделить приложение на независимые слои, минимизировать зависимость между компонентами и обеспечить легкость сопровождения и масштабирования системы.

Преимущества микросервисной архитектуры:

Модульность и независимость компонентов – каждый сервис разрабатывается и поддерживается независимо, что упрощает внесение изменений и добавление нового функционала.

Масштабируемость – микросервисы могут быть масштабированы отдельно в зависимости от нагрузки на каждый из них.

Устойчивость к сбоям – сбой одного компонента не приводит к отказу всей системы.

Гибкость – возможность замены или обновления отдельных компонентов без влияния на остальные части системы.

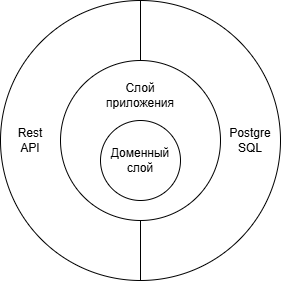
Принципы чистой архитектуры:

Четкое разделение приложения на слои: внешний слой интерфейсов, слой сервисов и бизнес-логики.

Независимость бизнес-логики от внешних компонентов: баз данных, интерфейсов и фреймворков.

Простота тестирования и изоляции: каждый слой тестируется отдельно, что повышает надежность системы.

2.2. Общая структура микросервисов



Каждый микросервис состоит из трех основных слоев:

Внешний слой: Интерфейсы и адаптеры

* Обеспечивает взаимодействие с внешним миром, например через REST API (с использованием HTTP) или gRPC.
* Обрабатывает входящие запросы и передает их в сервисный слой.
* Адаптеры для взаимодействия с реляционными базами данных, такими как PostgreSQL.

Слой приложения: Сервисы и сценарии использования (Use Cases)

* Реализует бизнес-логику работы с миграциями.
* Основные сценарии использования: запуск миграций, откат миграций, получение статуса миграций, аунтефикация и авторизация.
* Управляет потоками данных между внешним слоем и бизнес-логикой.

Доменный слой: бизнес-логика, сущности и правила

* Содержит основные модели и правила предметной области.
* Определяет сущности миграций, их состояния и логику выполнения операций.
* Полностью изолирована от внешних зависимостей.

2.3. Проектирование архитектуры межсервисного взаимодействия

В соответствии с выбранной микросервисной архитектурой, система разделена на два основных сервиса:

1. Сервис авторизации (Auth Service)

Зона ответственности сервиса управление пользователями, ролями и правами доступа.

Сервис для хранения информации о пользователях и их доступах содержит в себе таблицы:

* users
* roles
* permissions
* role\_permissions
* user\_roles

Для пользования сервисом предоставляется интерфейс для аутентификации пользователей, проверки прав доступа, управлениями ролями и разрешениями.

1. Сервис для управления миграциями (Migration Service)

Зона ответственности сервиса управление миграциями, просмотр истории применения миграций, логирование и сбор метрик.

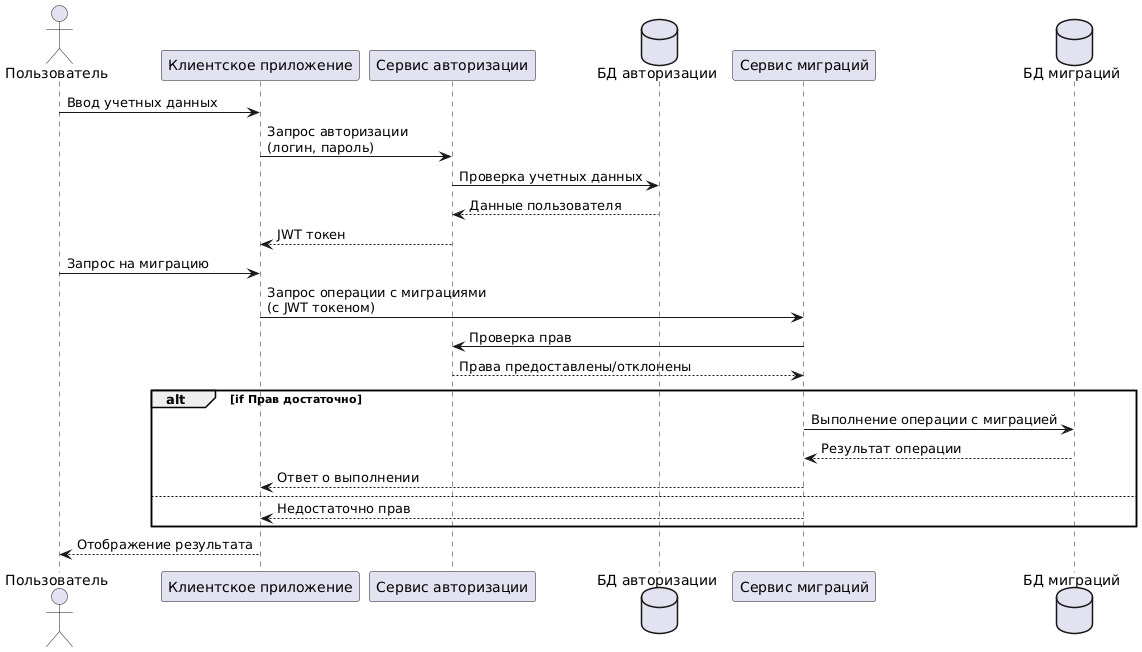
Для хранения информации о миграциях используются таблицы:

* migrations
* migration\_logs

Для пользования сервисом предоставляется интерфейс для аутентификации пользователей, проверки прав доступа, управлениями ролями и разрешениями.

В целях безопасности Migration Service в запросах принимает JWT токены, если такового нет, запрос не выполняется. При каждом запросе проверяется токен аутентификации и запрашиваются права доступа пользователя для выяснения доступных действий. Коммуникация между сервисами осуществляется gRPC, с использованием JSON Web Tokens (JWT) для аутентификации и формата JSON для передачи данных. Взаимодействие с сервисами со стороны пользователя осуществляется через HTTP REST API или gRPC. Архитектура предусматривает валидацию входящих данных и логирование всех запросов. Такое разделение обеспечивает четкое распределение зон ответственности сервисов, независимое масштабирование компонентов и устойчивость системы к частичным сбоям.

Диаграмма взаимодействия сервисов представлена на соответствующем рисунке, где подробно показаны последовательности вызовов между компонентами системы. Эта архитектура позволяет достичь высокой степени модульности и гибкости приложения, что особенно важно при дальнейшем развитии и поддержке проекта.



2.4. Проектирование структуры базы данных

Для разработки микросервиса, который управляет миграциями базы данных с учетом ролей пользователей и прав доступа, необходимо спроектировать структуру базы данных, которая будет поддерживать следующие функции:

Управление пользователями (роли и права доступа).

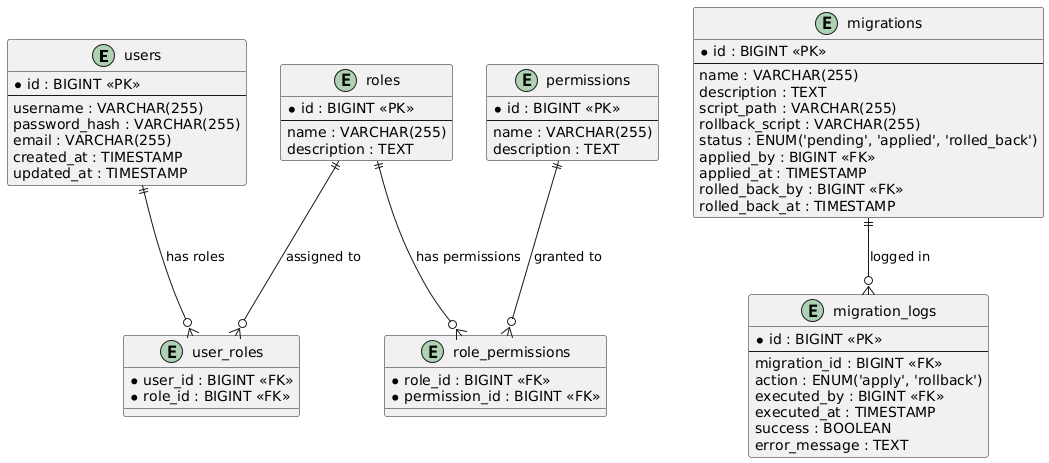
Управление миграциями (применение, откат, история выполнения).

Связь между пользователями и миграциями (кто и когда выполнил миграцию).

На основе этих требований была спроектирована следующая структура таблиц для двух микросервисов, каждый со своей базой данных:

База данных сервиса авторизации (Auth Service):

Хранит информацию о пользователях и их доступах.



1. Таблица users

Описание: хранит информацию о пользователях системы.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | BIGINT (PK) | Уникальный идентификатор пользователя. |
| username | VARCHAR(255) | Логин пользователя. |
| password\_hash | VARCHAR(255) | Хэш пароля пользователя (хранится в защищенном виде). |
| email | VARCHAR(255) | Электронная почта пользователя. |
| created\_at | TIMESTAMP | Дата и время создания записи. |
| updated\_at | TIMESTAMP | Дата и время последнего обновления записи. |

2. Таблица roles

Описание: хранит роли пользователей.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | BIGINT (PK) | Уникальный идентификатор роли. |
| name | VARCHAR(255) | Название роли (например, "admin", "developer", "viewer"). |
| description | TEXT | Описание роли. |

3. Таблица permissions

Описание: хранит права доступа.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | BIGINT (PK) | Уникальный идентификатор права. |
| name | VARCHAR(255) | Название права (например, "apply\_migration", "rollback\_migration"). |
| description | TEXT | Описание права. |

4. Таблица role\_permissions

Описание: связывает роли с правами доступа.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| role\_id | BIGINT (FK) | Идентификатор роли (ссылается на roles.id). |
| permission\_id | BIGINT (FK) | Идентификатор права (ссылается на permissions.id). |

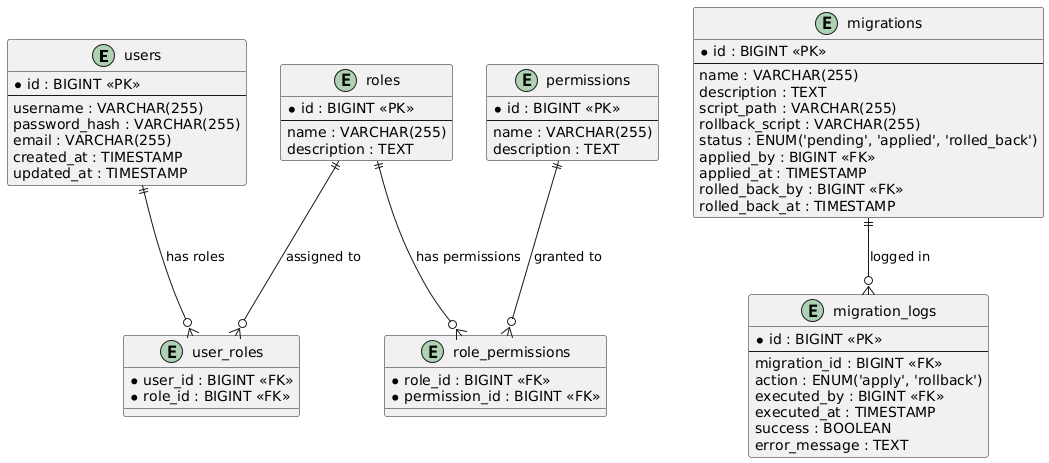
5. Таблица user\_roles

Описание: связывает пользователей с ролями.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| user\_id | BIGINT (FK) | Идентификатор пользователя (ссылается на users.id). |
| role\_id | BIGINT (FK) | Идентификатор роли (ссылается на roles.id). |

База данных сервиса миграций (Migration Service):

Хранит информацию о миграциях.



1. Таблица migrations

Описание: хранит информацию о миграциях.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | BIGINT (PK) | Уникальный идентификатор миграции. |
| name | VARCHAR(255) | Название миграции (например, "add\_users\_table"). |
| description | TEXT | Описание миграции. |
| script\_path | VARCHAR(255) | Путь к файлу скрипта миграции. |
| rollback\_script | VARCHAR(255) | Путь к файлу скрипта отката миграции. |
| status | ENUM('pending', 'applied', 'rolled\_back') | Текущий статус миграции. |
| applied\_by | BIGINT (FK) | Идентификатор пользователя, применившего миграцию (ссылается на users.id). |
| applied\_at | TIMESTAMP | Дата и время применения миграции. |
| rolled\_back\_by | BIGINT (FK) | Идентификатор пользователя, выполнившего откат миграции. |
| rolled\_back\_at | TIMESTAMP | Дата и время отката миграции. |

2. Таблица migration\_logs

Описание: Хранит историю выполнения миграций.

|  |  |  |
| --- | --- | --- |
| Поле | Тип данных | Описание |
| id | BIGINT (PK) | Уникальный идентификатор записи лога. |
| migration\_id | BIGINT (FK) | Идентификатор миграции (ссылается на migrations.id). |
| action | ENUM('apply', 'rollback') | Действие: применение или откат миграции. |
| executed\_by | BIGINT (FK) | Идентификатор пользователя, выполнившего действие (ссылается на users.id). |
| executed\_at | TIMESTAMP | Дата и время выполнения действия. |
| success | BOOLEAN | Успешность выполнения действия (true/false). |
| error\_message | TEXT | Сообщение об ошибке (если действие завершилось неудачно). |

Краткое описание связей:

Пользователи и роли: многие ко многим через таблицу user\_roles.

Роли и права доступа: многие ко многим через таблицу role\_permissions.

Миграции и пользователи: Одна миграция может быть применена или откачена одним пользователем.

Логи миграций: Каждая запись в логах связана с конкретной миграцией и пользователем, выполнившим действие.

Эта структура обеспечивает гибкость и масштабируемость для управления миграциями с учетом ролей и прав доступа.

2.5. Разработка интерфейса взаимодействия (API)

Серверное приложение предоставляет программный интерфейс REST API для взаимодействия с системой управления миграциями. Для описания интерфейса используется Swagger, что позволяет автоматически генерировать документацию.

Система предоставляет два отдельных интерфейса для взаимодействия с каждым микросервисом:

Интерфейс сервиса авторизации (Auth Service):

Для управления пользователями и правами доступа:

* Регистрация и аутентификация пользователей
  + POST /auth/register
  + POST /auth/login
* Управление ролями
  + GET /roles
  + POST /roles
  + PUT /roles/{id}
  + DELETE /roles/{id}
* Управление разрешениями
  + GET /permissions
  + POST /permissions
  + PUT /permissions/{id}
  + DELETE /permissions/{id}

Для управления миграциями базы данных:

1. Получение списка миграций

Метод: GET /api/migrations

Описание: Возвращает список всех миграций с их статусами.

Ответ: JSON-массив с информацией о миграциях.

[

{

"id": 1,

"name": "001\_initial\_schema",

"status": "applied",

"applied\_at": "2024-06-01T12:00:00Z"

},

{

"id": 2,

"name": "002\_add\_users\_table",

"status": "pending"

}

]

1. Применение миграций

Метод: POST /api/migrations/apply

Описание: запускает процесс применения миграций, переданных в аргументах.

Ответ: Сообщение об успешном применении миграций.

{

"message": "Migrations applied successfully.",

"applied\_count": 2

}

1. Откат миграций

Метод: POST /api/migrations/rollback

Описание: Откатывает последнюю примененную миграцию.

Ответ: Сообщение об успешном откате миграции.

{

"message": "Migration rolled back successfully.",

"rolled\_back\_migration": "002\_add\_users\_table"

}

1. Просмотр статуса конкретной миграции

Метод: GET /api/migrations/{id}

Описание: Возвращает информацию о конкретной миграции.

Ответ: JSON-объект с данными о миграции.

{

"id": 1,

"name": "001\_initial\_schema",

"status": "applied",

"applied\_at": "2024-06-01T12:00:00Z"

}

Оба API реализованы как RESTful сервисы и предоставляют следующие характеристики:

* Формат обмена данными: JSON
* Механизм аутентификации: JWT
* Документация: Swagger/OpenAPI
* Обработка ошибок: стандартизированные HTTP коды состояния

Межсервисное взаимодействие между Auth Service и Migration Service осуществляется через gRPC с использованием protocol buffers для сериализации данных.

Такая структура API обеспечивает четкое разделение зон ответственности между сервисами и позволяет независимо масштабировать каждый из них.

2.5. Используемые инструменты

Для реализации серверной компоненты выбраны следующие технологии и инструменты:

Язык программирования: Golang – обеспечивает высокую производительность и простоту в разработке серверных приложений.

База данных: PostgreSQL – популярная реляционная СУБД с широким функционалом и поддержкой ACID-транзакций.

Контейнеризация: Docker и Docker Compose – позволяют развертывать приложение в изолированной среде.

Документация интерфейса: Swagger – автоматическая генерация документации REST API.

Тестирование: Testify и Mockgen – фреймворки для юнит-тестирования и создания моков.

Логирование: Zerolog – эффективный инструмент для структурированного логирования.

Контроль версий: Git и GitHub – для управления исходным кодом и совместной разработки.

Подводя итог главы, был обоснован выбор микросервисной архитектуры и принципов чистой архитектуры для проектирования серверной компоненты. Продумано взаимодействие сервисов между собой и поведение пользователя. Разработана структура базы данных для хранения информации о миграциях, спроектированы основные конечные точки API для взаимодействия с приложением, а также определен технологический стек для реализации системы.

Проектирование и четкая структура системы позволили создать эффективное, гибкое и масштабируемое приложение для управления миграциями в реляционных базах данных.

3. РЕАЛИЗАЦИЯ ПРИЛОЖЕНИЯ

3.1. Общие подходы к реализации

Реализация серверной компоненты приложения для просмотра и управления миграциями в реляционных базах данных выполнялась на основе предварительно спроектированной архитектуры. Основной фокус был сделан на модульности системы, обеспечении отказоустойчивости и легкости масштабирования.

На этапе разработки были учтены следующие ключевые принципы:

Разделение ответственности – каждый слой выполняет строго определенные задачи.

Минимальная зависимость между компонентами – слои и модули взаимодействуют через четко определенные интерфейсы.

Тестируемость – реализуемые компоненты покрывались юнит-тестами для обеспечения корректности работы.

Производительность и оптимизация – фокус на минимизацию времени отклика API и оптимизацию запросов к базе данных.

Приложение было реализовано с использованием языка Golang благодаря его высокой производительности и поддержке многопоточности.

3.2. Реализация модулей приложения

3.2.1. Модуль управления миграциями

Модуль управления миграциями - ядро приложения и отвечает за выполнение следующих операций:

Применение миграций – последовательное выполнение новых миграций, сохранение их статуса в базе данных.

Откат миграций – откат последней успешно выполненной миграции.

Проверка статуса миграций – получение текущего состояния миграций (выполненные, ожидающие или откатанные).

Для работы с миграциями были внедрены механизмы управления состоянием базы данных. Основой модуля служит логика, определяющая порядок применения и отката миграций с учетом их зависимостей.

3.2.2. Модуль взаимодействия с базами данных

Модуль взаимодействия с реляционными базами данных был реализован с использованием драйвера для PostgreSQL, что обеспечивает поддержку выбранной СУБД. Основные функции модуля:

Соединение с базой данных – настройка пула соединений и их управление для минимизации накладных расходов.

Выполнение SQL-запросов – выполнение миграционных скриптов и обновление записей о состоянии миграций.

Хранение информации о миграциях – добавление и обновление записей в таблице migrations, содержащей историю миграций.

Для обеспечения отказоустойчивости добавлена логика обработки ошибок при выполнении запросов, что позволяет откатывать некорректно выполненные операции.

3.2.3. Модуль REST API

Модуль REST API обеспечивает внешнее взаимодействие с серверной компонентой приложения. Основные функции включают:

Обработку HTTP-запросов – маршрутизация запросов и вызов соответствующих обработчиков.

Формирование ответов – возврат данных в формате JSON в соответствии с требованиями клиентов.

Обработку ошибок – возвращение корректных HTTP-кодов при возникновении ошибок.

В рамках реализации модуля были определены основные конечные точки для управления миграциями:

Получение списка миграций.

Запуск новых миграций.

Откат последней миграции.

Просмотр статуса конкретной миграции.

Использование Swagger позволило автоматизировать документирование API, что значительно упростило процесс тестирования и интеграции с клиентскими приложениями.

3.2.4. Модуль логирования

Для отслеживания работы системы был внедрен модуль логирования на основе библиотеки Zerolog. Основные возможности:

Логирование информации о выполненных операциях (например, запуск или откат миграций).

Запись ошибок и критических событий.

Фиксация временных меток выполнения операций.

Логи собираются в структурированном виде, что позволяет легко анализировать работу системы и выявлять потенциальные проблемы.

3.3. Подходы к тестированию системы

Тестирование серверной компоненты приложения для управления миграциями проводилось с целью проверки корректности работы всех реализованных модулей, выявления потенциальных ошибок и обеспечения надежности системы.

Для достижения поставленных целей были применены следующие виды тестирования:

Модульное тестирование  
Проверка отдельных компонентов системы на соответствие их функциональным требованиям.

Интеграционное тестирование  
Тестирование взаимодействия между модулями приложения для проверки целостности работы системы.

Функциональное тестирование  
Проверка работы API и всей системы в целом на соответствие заявленным требованиям.

3.3.1. Тестирование функциональности модулей

Функциональные тесты были разработаны для основных компонентов приложения:

Модуль управления миграциями  
Проверялась корректность выполнения операций:

Применение миграций – успешное выполнение SQL-скриптов и сохранение статуса в таблице migrations.

Откат миграций – откат последнего изменения базы данных и обновление статуса.

Проверка статуса миграций – возврат актуального состояния миграций (выполненные, ожидающие или откатанные).

Модуль взаимодействия с базами данных  
Тестировалось выполнение SQL-запросов на создание и изменение структур данных, а также устойчивость модуля к ошибкам (например, при некорректных скриптах миграций).

Модуль REST API  
Проверялась работа конечных точек (endpoints), включая:

Корректность ответов на запросы.

Обработку ошибок при некорректных входных данных.

Соответствие форматов запросов и ответов требованиям документации Swagger.

3.3.2. Инструменты для тестирования

Для проведения тестирования использовались следующие инструменты:

Testify – для написания модульных тестов.

Mockgen – для создания мок-объектов при тестировании взаимодействия компонентов.

Postman – для проверки работы REST API.

Docker и Docker Compose – для создания тестовой среды, содержащей экземпляр базы данных PostgreSQL.

3.4. Обработка ошибок и отладка

В процессе разработки и тестирования системы были выявлены и устранены следующие проблемы:

3.5. Документация API

В разделе представлено описание интерфейсов взаимодействия приложения с пользователем и другими сервисами. Используется формат документации Swagger для наглядного представления доступных методов и запросов API.

Основные компоненты документации API:

Описание конечных точек (*endpoints*), их назначения и формата вызовов.

Примеры запросов и ответов в формате JSON.

Коды ответа сервера с расшифровкой возможных ошибок.

Поля запросов и ответов: типы данных и требования к ним.

3.6. Инструкция по запуску и использованию приложения

В разделе представлена пошаговая инструкция по настройке и запуску разработанного приложения.

Шаги установки и развертывания:

Подготовка окружения:

Необходимые зависимости (например, Docker, Go).

Версии поддерживаемой СУБД (PostgreSQL).

Настройка конфигурационных файлов:

Описание параметров конфигурации приложения (база данных, порты, параметры безопасности).

Запуск сервера:

Команды для развертывания через Docker и локальный запуск.

Использование API:

Описание взаимодействия с приложением через предоставленные конечные точки API.

Примеры использования с Postman и другими клиентами для работы с API.

Для удобного развертывания приложения используется Docker и Docker Compose. В разделе представлены инструкции по созданию и запуску контейнеров.

Основные этапы:

Создание образа Docker:

Команды для сборки приложения в контейнере.

Запуск приложения с Docker Compose:

Описание файла docker-compose.yml и его параметров.

Команды для запуска всех зависимостей и сервисов.

Проверка статуса работы контейнеров:

Инструкции по мониторингу и логированию контейнеров.

Остановка и перезапуск приложения:

Основные команды для управления жизненным циклом контейнеров.

3.7. Рекомендации по развертыванию

В разделе даны рекомендации для безопасного и эффективного развертывания приложения на рабочем сервере.

Ключевые моменты:

Настройка переменных окружения и секретов (токенов безопасности).

Масштабирование приложения для поддержки высокой нагрузки.

Мониторинг и логирование работы сервера с использованием сторонних инструментов.

Настройка резервного копирования данных и отказоустойчивости.

ЗАКЛЮЧЕНИЕ

1. Итоги работы

Достижение цели и выполненные задачи.

2. Результаты исследования

Описание готового приложения и его преимуществ.

3. Перспективы дальнейшего развития

Возможные улучшения: расширение функционала и поддержка новых СУБД.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

Научные статьи, документация по инструментам (Golang, Docker, PostgreSQL, Swagger), ресурсы по микросервисной архитектуре.

ПРИЛОЖЕНИЯ

Примеры запросов API.

SQL-схемы и структура базы данных.

Результаты тестов и логи выполнения.