

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ рОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Дальневосточный федеральный университет»**

(ДВФУ)

|  |  |
| --- | --- |
| **ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ (ШКОЛА)**  **Департамент математического и компьютерного моделирования** | |
|  |

Отчёт по курсовой работе по дисциплине «Веб-программирование (Backend)»

на тему «Разработка бекенда для приложения “Todo list”»

«Математика и компьютерные науки»

профиль «Сквозные цифровые технологии»

|  |  |
| --- | --- |
| Работа защищена  с оценкой \_\_отлично\_\_\_  Регистрационный номер \_\_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г. | Студент группы № Б9122-02.03.01сцт  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чилеше.А.  (подпись) (ФИО)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024г.  Руководитель \_ассистент\_\_\_\_\_\_\_\_\_\_\_\_  (должность, ученое звание)  Охроменко Дарья Александровна\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) (ФИО)  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г. |

г. Владивосток

2024

Table of Contents

[1. Введение 2](#_Toc171273334)

[1.1 Значимость 2](#_Toc171273335)

[1.1. Цели и задачи 2](#_Toc171273336)

[1.3 Используемые технологии 3](#_Toc171273337)

[2. База данных 5](#_Toc171273338)

[2.1. Настройка базы данных 5](#_Toc171273339)

[2.2. Модели 6](#_Toc171273340)

[2.3. Объекты передачи данных (DTO) 8](#_Toc171273341)

[2.4 Mapper 9](#_Toc171273342)

[2.5. Миграции 10](#_Toc171273343)

[3. Взаимодействие с базой данных 12](#_Toc171273344)

[3.1. Репозиторий (Repository) 12](#_Toc171273345)

[3.2. Сервис (Service) 13](#_Toc171273346)

[3.3. Исключения 14](#_Toc171273347)

[4. Тестирование 15](#_Toc171273348)

[4.1 Get User 15](#_Toc171273349)

[4.2. Create User 15](#_Toc171273350)

[4.3. Delete User 16](#_Toc171273351)

[4.4. Update User 16](#_Toc171273352)

[5. Вывод 17](#_Toc171273353)

[6. Источники 18](#_Toc171273354)

# Введение

## 1.1 Значимость

В современном разработке программного обеспечения создание приложений или веб-сайтов обычно делится на два основных компонента: фронтенд и бэкенд. Фронтенд отвечает за пользовательский интерфейс и визуальные элементы, такие как размещение кнопок и эстетику иконок. Бэкенд, однако, управляет внутренними операциями, включая бизнес-логику, взаимодействие с базой данных, обработку пользовательских запросов, аутентификацию и интеграцию с внешними сервисами.

Производительность и безопасность бэкенда имеют решающее значение для успешности приложения. Хорошо написанный и оптимизированный бэкенд-код обеспечивает эффективную и безопасную работу приложения. Чистый и организованный код улучшает поддержку, снижает количество ошибок и упрощает тестирование. Использование современных фреймворков и библиотек, таких как Spring для Java, может значительно упростить процесс разработки и привести к более качественным результатам.

## Цели и задачи

Целью данной курсовой работы является разработка бэкенд-части приложения на языке Java с акцентом на управление списками задач для различных пользователей и использованием PostgreSQL в качестве системы управления базами данных (СУБД). Для достижения этой цели были поставлены следующие задачи:

1. **Проектирование базы данных:**

Планирование и разработка масштабируемой и эффективной схемы базы данных для управления данными пользователей и списками задач.

1. **Понимание веб-запросов:** CRUD (создание, чтение, обновление, удаление)

Изучение и понимание различных типов веб-запросов (GET, POST, PUT, DELETE) и их обработка в приложении.

1. **Ознакомление с инструментами:** Чтение и понимание документации по различным инструментам и библиотекам, используемым в проекте.
2. **Лучшие практики разработки:** Изучение и применение лучших практик и методологий для написания эффективного и поддерживаемого кода бэкенда.

Приложение разработано для выполнения следующих функций:

1. Управление пользователями: Добавление, чтение, обновление и удаление информации о пользователях.
2. Управление заметками: Добавление, чтение, обновление и удаление заметок.
3. Аутентификация пользователей: Реализация безопасной системы аутентификации для управления доступом пользователей.
4. Обработка веб-запросов: Эффективная обработка веб-запросов и обеспечение правильных ответов.
5. Обработка ошибок: Реализация надежной обработки исключений для управления ошибками.
6. Миграции базы данных: Выполнение миграций базы данных для обеспечения согласованности схемы в разных средах.

## 1.3 Используемые технологии

Для выполнения этих задач были использованы различные технологии и инструменты:

* Maven Framework

Инструмент автоматизации сборки, который упрощает процесс сборки проектов на Java, управления зависимостями и конфигурации проектов.

* Spring Framework

Всеобъемлющий фреймворк, предлагающий различные модули для выполнения задач, таких как доступ к данным, безопасность, управление транзакциями и обмен сообщениями.

* Hibernate Library

Библиотека ORM, автоматизирующая преобразование объектов Java в таблицы базы данных, что упрощает операции с базой данных.

* Lombok Library

Библиотека, сокращающая шаблонный код в Java путем создания общих методов (геттеров, сеттеров, конструкторов) с помощью аннотаций.

* Docker Container System

Платформа для разработки, развертывания и запуска приложений в контейнерах, обеспечивающая согласованность в различных средах. Используется для запуска PostgreSQL в изолированной среде, что упрощает управление базой данных.

* PostgreSQL DBMS

Мощная, открытая объектно-реляционная система управления базами данных, известная своей производительностью и масштабируемостью.

* Flyway Library

Инструмент миграции базы данных, автоматизирующий выполнение SQL-скриптов для версионирования схемы базы данных.

* IntelliJ IDEA Ultimate

Интегрированная среда разработки (IDE), поддерживающая разработку на Java с встроенными инструментами для Spring и Maven, что повышает производительность и качество кода.

Интеграция этих технологий позволила разработать надежное, эффективное и легко поддерживаемое бэкенд-приложение, соответствующее современным практикам разработки программного обеспечения.

# База данных

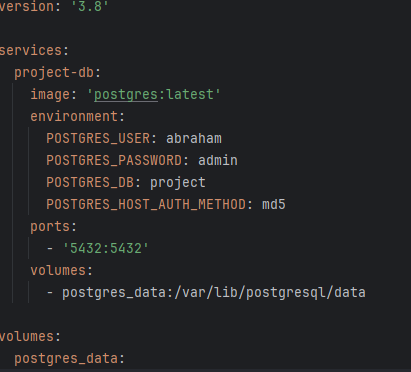
В данной главе подробно рассматриваются модели базы данных, используемые в проекте. Описывается структура и назначение каждой модели, процесс миграции базы данных и роль объектов передачи данных (DTO). Понимание этих компонентов важно для того, чтобы понять, как данные управляются, передаются и поддерживаются в приложении.

## Настройка базы данных

Система управления базами данных PostgreSQL развертывается с использованием Docker. Чтобы установить и запустить СУБД в контейнере Docker, выполните следующие шаги:

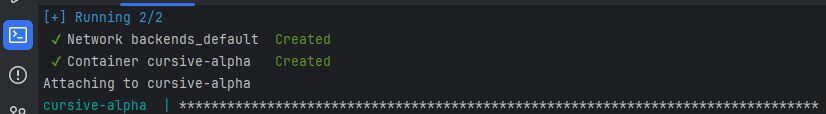
Создание файла docker-compose.yaml:

* Определите необходимые атрибуты для образа PostgreSQL, включая модель базы данных, которая будет использоваться.
* Укажите параметры контейнера, такие как имя пользователя, пароль, путь в виртуальной машине и используемые порты



Сборка и запуск контейнера:

* Выполните команду docker-compose up --build. Эта команда загрузит указанный образ базы данных, если он еще не загружен локально.
* Docker создаст и запустит контейнер, используя загруженный образ.



## Модели

#### Обзор

В рамках проекта в базе данных были созданы две основные таблицы: User и Node. Таблица User предназначена для хранения информации о пользователях, а таблица Node — для хранения записей, связанных с этими пользователями.

Библиотека Hibernate использовалась для отображения классов Java в таблицы базы данных, что облегчает управление операциями с базой данных через объектно-ориентированное программирование. Определяя объекты базы данных в виде классов, Hibernate автоматизирует преобразование этих классов в соответствующие таблицы базы данных, упрощая взаимодействие с базой данных.

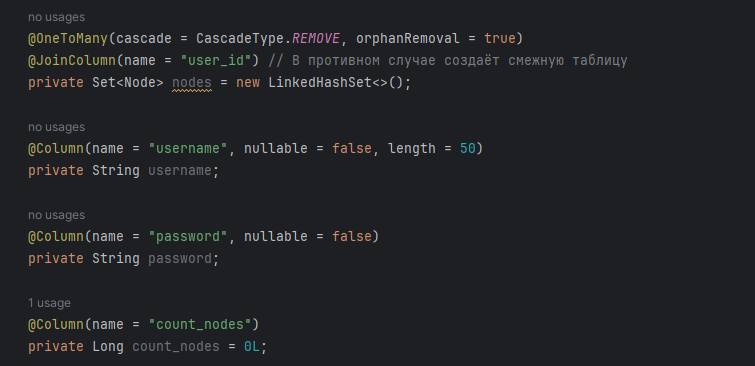
Кроме того, была использована библиотека Lombok для сокращения шаблонного кода. Аннотации Lombok автоматически генерируют общие методы и конструкторы, что улучшает читаемость и поддерживаемость кода. Это значительно уменьшает необходимость в повторяющемся коде, позволяя разработчикам сосредоточиться на бизнес-логике.

#### Модель User

Таблица User состоит из следующих полей:

1. **id:** Уникальный идентификатор пользователя.
2. **username:** Имя пользователя.
3. **password:** Пароль пользователя.
4. **count\_nodes:** Количество записей, связанных с пользователем.

*User Table snippet*

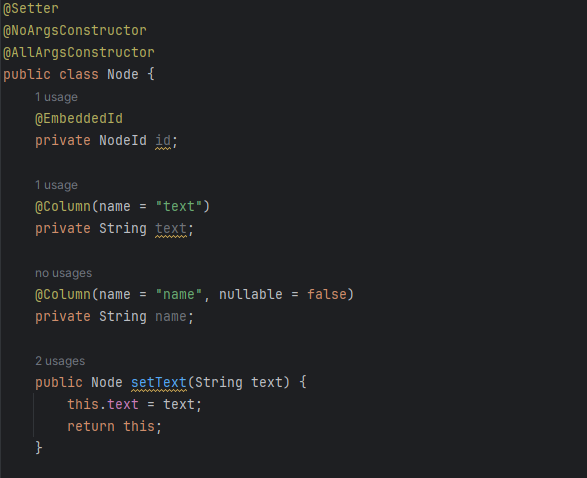


Кроме этих полей, модель **User** включает переменную **nodes**, которая определяет отношение "один ко многим" с таблицей Node. Это отношение важно для связывания каждого пользователя с его записями. Функция incrementCount предназначена для увеличения значения count\_nodes на 1, что полезно для динамического отслеживания количества записей, связанных с пользователем.

#### Node

Таблица Node предназначена для хранения отдельных записей и включает в себя следующие поля:

* **id:** Композитный ключ, используемый для уникальной идентификации каждой записи. Этот композитный ключ состоит из:
  + **User:** Пользователь, которому принадлежит запись.
  + i**d:** Уникальный идентификатор записи относительно пользователя.
* **Name:** Заголовок записи, представляющий тему или заголовок записи. •
* **Text:** Основное содержание записи, содержащее детали или основной текст записи.

Кроме того, модель Node включает переменную users, которая устанавливает отношение ManyToOne (многие к одному) с таблицей User. Это отношение важно для связывания каждой записи с ее владельцем, что позволяет эффективно управлять данными и извлекать их на основе принадлежности пользователю.

## Объекты передачи данных (DTO)

В разработке программного обеспечения Data Transfer Objects (DTO) играют важную роль как контейнеры для данных, необходимых для выполнения операций или удовлетворения запросов в приложении. Они выполняют несколько функций, сосредотачиваясь в первую очередь на повышении безопасности и эффективности в процессе обмена данными между различными слоями приложения.

#### Цель DTO

DTO предназначены для инкапсуляции данных, необходимых для конкретных задач, исключая при этом чувствительную или излишнюю информацию. Это практическое решение обеспечивает передачу только актуальных данных между различными частями приложения, минимизируя риск раскрытия конфиденциальной информации или предоставления возможностей злоумышленникам для компрометации учетных записей пользователей.

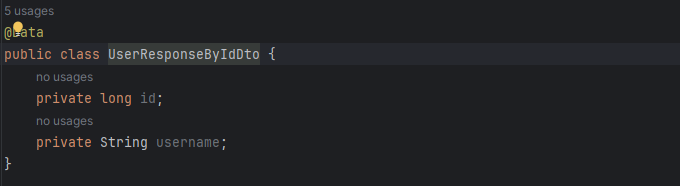
#### Применение в практике

В рамках архитектуры приложения DTO выполняют различные функции. Во-первых, они используются для получения данных из внешних источников, таких как интернет-запросы. Этот процесс включает в себя сбор и структурирование данных, полученных из удаленных серверов или API, с тем чтобы передавать в приложение только необходимую информацию.

Во-вторых, DTO играют ключевую роль в представлении информации в пользовательском интерфейсе. Благодаря организации данных в DTO разработчики могут оптимизировать процесс отображения информации для пользователей, повышая производительность и улучшая пользовательский опыт.

#### Аспекты безопасности

Одной из основных целей использования DTO является повышение безопасности приложения. Путем внимательного выбора и передачи только необходимых полей данных DTO снижают риск раскрытия чувствительной информации в процессе обмена данными. Это соответствует лучшим практикам в области кибербезопасности, защищая конфиденциальность пользователей и предотвращая несанкционированные попытки доступа.



## Mapper

Mapper преобразует сущности (например, Пользователь и Узел) в DTO (объекты передачи данных).

Он позволяет автоматизировать процесс получения необходимых параметров DTO из сущности. Вместо того чтобы вручную описывать процесс передачи данных, достаточно назвать переменные в DTO таким же образом, как в сущности, и маппер сможет самостоятельно выполнить передачу необходимых данных.

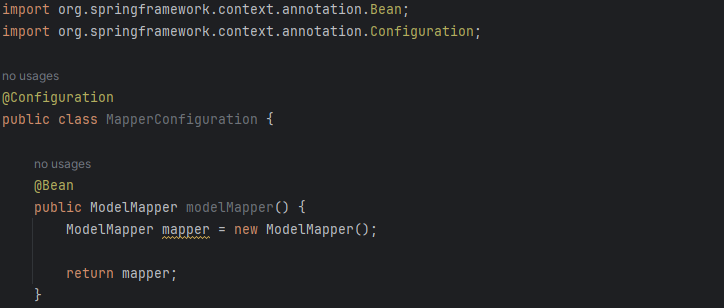
ModelMapper по умолчанию встроен в Spring, поэтому все, что требуется, это настроить начальную конфигурацию для начала работы с ним.

**Применение в практике**

Мапперы играют ключевую роль в разработке программного обеспечения, особенно в контексте архитектурных паттернов MVC (Model-View-Controller) и RESTful API. Они позволяют значительно упростить кодирование и снизить вероятность ошибок при передаче данных между уровнями приложения.

**Пример использования**

Допустим, у нас есть класс Пользователь (User) с определенными полями, такими как имя, электронная почта и роль. Для передачи данных о пользователе через API в клиентское приложение нам необходимо создать DTO Пользователя (UserDTO). Маппер позволяет автоматически скопировать значения этих полей из объекта Пользователь в объект UserDTO, если имена полей в обоих классах совпадают.



## Миграции

Миграции относятся к командам базы данных, которые выполняются при запуске программы. Они позволяют вносить изменения, такие как расширение полей таблиц, при этом сохраняя существующие, добавление новых таблиц и выполнение задач, обычно решаемых с прямым доступом к базе данных.

#### Цель и преимущества

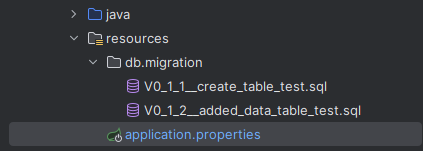
Миграции обеспечивают контроль версий для баз данных, позволяя систематически вносить изменения, которые можно откатить в случае возникновения проблем. Эта возможность версионирования гарантирует, что разработчики могут вернуться к предыдущим состояниям базы данных и продолжить работу без проблем.

#### Реализация с помощью Flyway

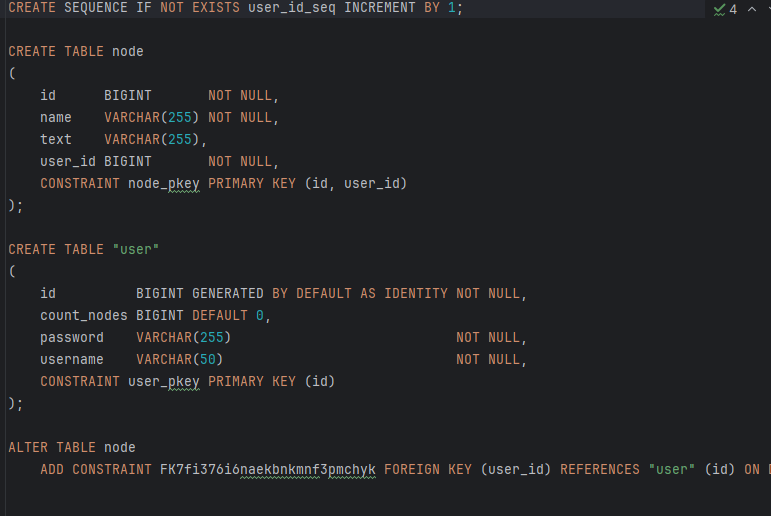
В нашей конфигурации миграции управляются с использованием библиотеки Flyway. Для запуска миграции файл миграции должен быть назван в определенном формате: V<номер\_версии>\_\_<имя\_миграции>.sql. Это соглашение о наименовании гарантирует, что Flyway правильно упорядочивает и выполняет миграции на основе их номеров версий и имен.

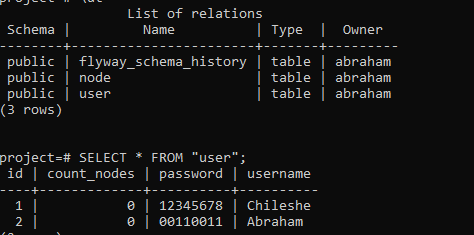
#### Преимущества использования Flyway

* Автоматизирует выполнение миграций баз данных, сокращая ручные усилия и минимизируя ошибки.
* Согласованность: Гарантирует, что все экземпляры приложения имеют одну и ту же схему базы данных, повышая согласованность в различных средах.
* Версионирование: Упрощает совместную работу и развертывание за счет отслеживания изменений базы данных во времени, что критично для поддержания целостности приложения.



*Example of user table migration*





# Взаимодействие с базой данных

Взаимодействие с базами данных в приложениях на Spring включает структурированные компоненты: Repository (Репозиторий), Service (Сервис) и обработку исключений (Exception handling). Эти компоненты совместно обеспечивают эффективное управление данными и их обработку в архитектуре приложения.

## Репозиторий (Repository)

Репозиторий в Spring играет ключевую роль как интерфейс между приложением и базой данных. Он инкапсулирует логику доступа и манипулирования сущностями данных, обычно с использованием операций **CRUD** (Create, Read, Update, Delete).

*Основные характеристики Репозитория:*

Операции CRUD:

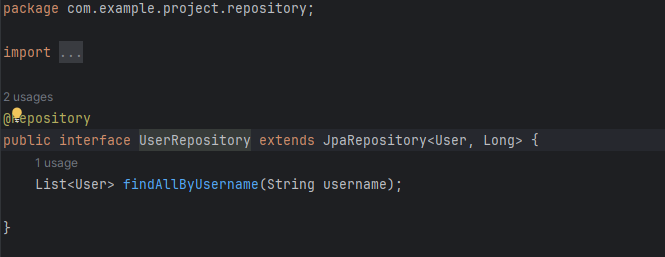
* Основная функция репозитория заключается в предоставлении методов для выполнения операций CRUD над конкретной сущностью. Эти операции позволяют разработчикам создавать новые записи, извлекать существующие данные, обновлять записи и удалять записи из базы данных.

Стандартные методы:

* Репозитории Spring Data предоставляют встроенные методы для распространенных операций с базой данных, таких как сохранение сущностей, поиск сущностей по идентификатору, удаление сущностей и запросы сущностей на основе определенных критериев.

Пользовательские методы:

* В дополнение к стандартным операциям CRUD, репозитории могут определять пользовательские методы, адаптированные под конкретные бизнес-требования. Эти методы используют соглашение о наименовании, которое описывает действие, количество затрагиваемых сущностей и параметры, используемые для фильтрации или поиска.

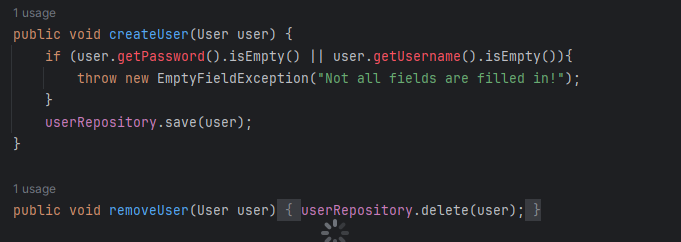


## Сервис (Service)

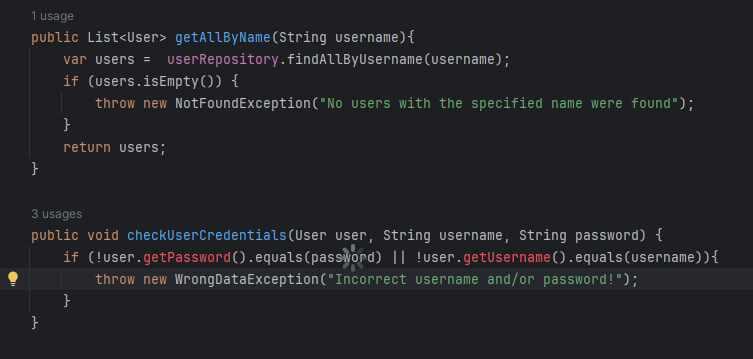
В Spring сервис (Service) представляет собой класс, ответственный за непосредственное взаимодействие с базой данных.

***Обязанности:***

* *Прямое взаимодействие с базой данных:* Класс Service использует методы репозитория для выполнения команд в базе данных. Он служит связующим звеном между бизнес-логикой приложения и уровнем хранения данных.
* *Проверка данных:* Помимо выполнения методов репозитория, Service также выполняет различные проверки данных. Эти проверки обеспечивают целостность и корректность данных перед их обработкой или сохранением в базе данных.
* *Обработка исключений:* В случае неверной проверки данных или ошибок во время операций с базой данных, Service выбрасывает соответствующие исключения. Это обеспечивает правильную обработку и управление ошибками в рамках приложения.



*Authentication*



## Исключения

Исключения возникают при некорректной обработке данных на уровне сервиса.

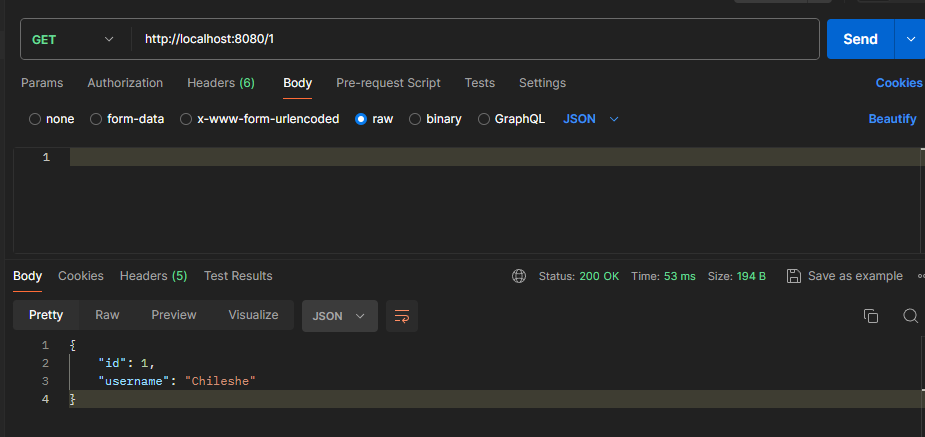
**Реализация:**

Исключения в приложениях на Spring обычно обрабатываются через основной класс контроллера, который возвращает объект DTO ошибки. Этот DTO часто является основным классом ошибки, от которого наследуются специфические классы исключений. Такой подход позволяет осуществлять дифференцированную обработку и реализацию логики для различных типов исключений.

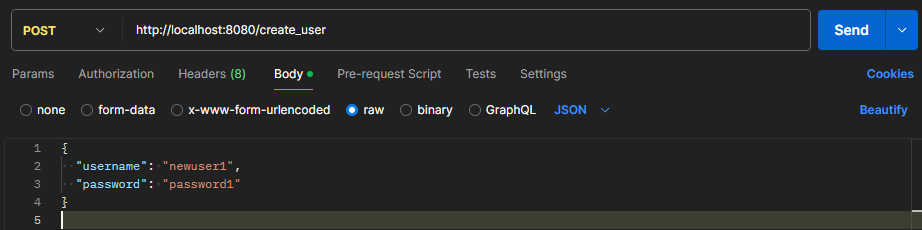


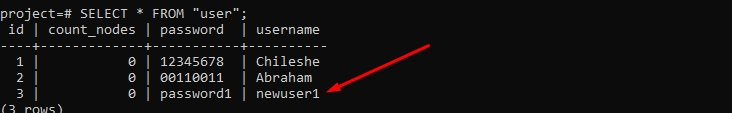
# Тестирование

## Get User

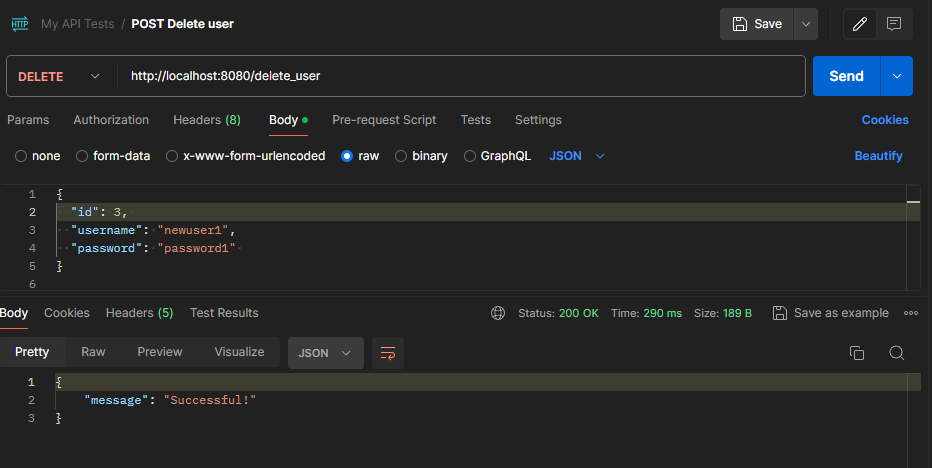


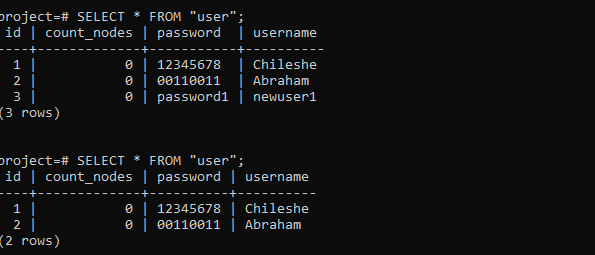
## Create User



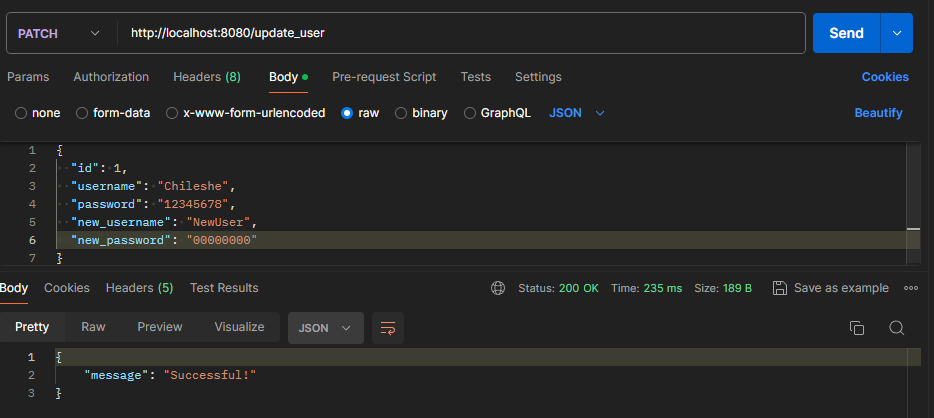


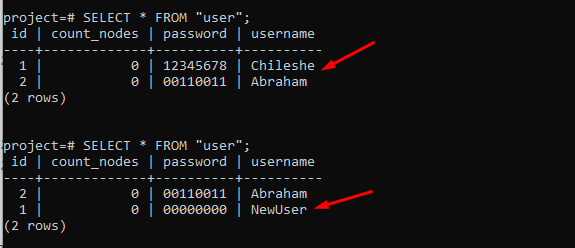
## Delete User

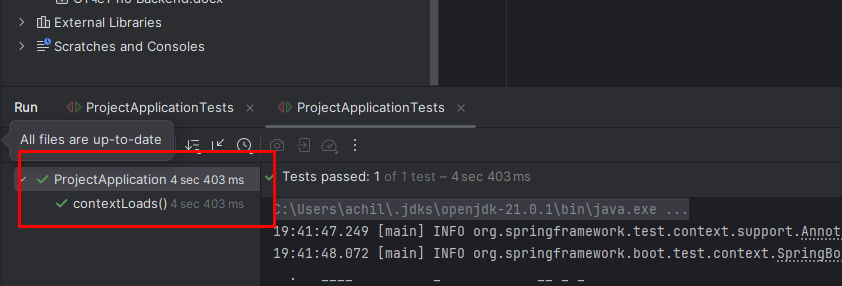




## Update User







# Вывод

В результате проделанной работы был разработан backend компонент приложения для управления списками задач различных пользователей. Этот компонент способен обрабатывать различные HTTP-запросы к серверу и эффективно перехватывает внутренние исключения, предоставляя сообщения об ошибках в понятном и удобном для пользователя формате.

# Источники

Documentation Overview :: Spring Boot [Электронный ресурс]. URL: <https://docs.spring.io/spring-boot/documentation.html> (дата обращения: 18.06.2024).

Maven – Maven Documentation [Электронный ресурс]. URL: <https://maven.apache.org/guides/index.html> (дата обращения: 22.06.2024).

Docker - Reference documentation // Docker Documentation [Электронный ресурс]. URL: <https://docs.docker.com/reference/> (дата обращения: 18.06.2024).

Getting started | IntelliJ IDEA Documentation [Электронный ресурс]. URL: <https://www.jetbrains.com/help/idea/getting-started.html> (дата обращения: 22.06.2024).

PostgreSQL 16.3 Documentation // PostgreSQL Documentation [Электронный ресурс]. URL: <https://www.postgresql.org/docs/16/index.html> (дата обращения: 22.06.2024).

Flyway CLI and API - Flyway - Product Documentation [Электронный ресурс]. URL: <https://documentation.red-gate.com/flyway/flyway-cli-and-api> (дата обращения: 18.06.2024).

Lombok - documentation [Электронный ресурс]. URL: <https://projectlombok.org/features/> (дата обращения: 22.06.2024).

Documentation - 6.5 - Hibernate ORM // Hibernate [Электронный ресурс]. URL: <https://hibernate.org/orm/documentation/6.5/> (дата обращения: 18.06.2024).