

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 \_\_\_\_\_\_\_\_ работе

**«\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_»**

|  |  |
| --- | --- |
| Работа защищена  с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Регистрационный номер \_\_\_\_\_\_\_  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г. | Студент(ы) группы № Б9122-02.03.01сцт  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Поповкин А. А.  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 202\*г.  Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (должность, ученое звание)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) (ФИО)  «\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\*г. |

г. Владивосток

2024

Оглавление

[1. Введение 2](#_Toc169991803)

[1.1. Актуальность 2](#_Toc169991804)

[1.2. Цели и задачи 2](#_Toc169991805)

[1.3. Используемые технологии 2](#_Toc169991806)

[2. База данных 4](#_Toc169991807)

[2.1. Введение 4](#_Toc169991808)

[2.2. Модели 4](#_Toc169991809)

[2.2.1. Введение 4](#_Toc169991810)

[2.2.2. User 4](#_Toc169991811)

[2.2.3. Node 5](#_Toc169991812)

[2.3. DTO 6](#_Toc169991813)

[2.4. Mapper 7](#_Toc169991814)

[2.5. Миграции 8](#_Toc169991815)

[3. Взаимодействие с базой данных 9](#_Toc169991816)

[3.1. Введение 9](#_Toc169991817)

[3.2. Репозиторий 9](#_Toc169991818)

[3.3. Сервис 9](#_Toc169991819)

[3.3.1. Аутентификация 10](#_Toc169991820)

[3.4. Исключения 10](#_Toc169991821)

[4. Интернет-запросы 11](#_Toc169991822)

[4.1. Введение 11](#_Toc169991823)

[4.2. Контроллер 11](#_Toc169991824)

[4.2.1. Введение 11](#_Toc169991825)

[4.2.2. Запрос без тела 11](#_Toc169991826)

[4.2.3. Запрос с телом 12](#_Toc169991827)

[5. Тестирование 13](#_Toc169991828)

[6. Вывод 14](#_Toc169991829)

[7. Источники 15](#_Toc169991830)

# Введение

## Актуальность

В современном программировании работа над приложением или сайтом делится на две составляющие: **backend** и **frontend**.

**Frontend** разработка отвечает за визуальную составляющую программы: за расположение кнопок, за красивые иконки и т. д.

**Backend** в свою очередь отвечает за всё остальное. Создание бизнес-логики, управление базами данных, обработка интернет-запросов пользователей к серверу, аутентификация пользователей, интеграция с другими сервисами и многое другое. От того, как backend-разработчики опишут работу сервера, от оптимизации запросов к базам данных зависит безопасность и скорость работы всего приложения или сайта.

Хорошо структурированный и чистый код облегчает поддержку и дальнейшее развитие приложения, снижает вероятность возникновения багов и значительно упрощает тестирование.

Использование современных фреймворков и библиотек, таких как Spring для Java, помогает упростить и ускорить процесс разработки, при этом повысив качество конечного продукта.

## Цели и задачи

В рамках курсовой работы было необходимо реализовать приложение на языке программирования Java для создания списков задач для различных пользователей, с системой управления базами данных (СУБД) PostgreSQL.

Для достижения поставленной цели было необходимо решить следующие задачи:

1. Планирование структуры базы данных.
2. Изучение типов веб-запросов
3. Чтение документации к новым инструментам.
4. Изучение методологии написания приложений.

В функции приложения входило:

1. Добавление, чтение, изменение данных, удаление пользователей.
2. Добавление, чтение, изменение данных, удаление заметок.
3. Создание системы аутентификации для пользователей.
4. Обработка веб-запросов к приложению.
5. Обработка исключений.
6. Исполнение написанных миграций.

## Используемые технологии

Для выполнения поставленных задач были использованы различные технологии:

1. **Фреймворк Maven**[3]
   1. Инструмент для автоматической сборки проектов на Java.
2. **Фреймворк Spring**[4]
   1. Набор инструментов, предлагающий различные модули для выполнения задач, таких как доступ к данным, безопасность, транзакции и обмен сообщениями.
3. **Библиотека Hibernate**[5]
   1. Библиотека, автоматизирующая процесс перевода кода в команды для СУБД.
4. **Библиотека Lombok**[6]
   1. Библиотека, позволяющая сократить шаблонный код Java в хорошо читаемые аннотации.
5. **Система контейнеров Docker**[1]
   1. Платформа, которая предназначена для разработки, развёртывания и запуска приложений в контейнерах.
   2. Контейнеры были использованы для изолированного запуска СУБД PostgreSQL на виртуальном сервере, что позволило с удобством проводить манипуляции с базой данных.
6. **СУБД PostgreSQL**[2]
   1. Серверная система управления базами данных.
7. **Библиотека Flyway**[7]
   1. Библиотека для автоматического исполнения миграций при запуске приложения.
8. **Интегрированная среда разработки (IDE) IntelliJ IDEA Ultimate**[8]
   1. Удобная среда разработки со встроенной поддержкой **Spring** и **Maven**.

# База данных

## Введение

В данной главе будут описаны созданные модели, а также будет рассказано про процесс миграций и объекты для передачи данных.

## Модели

### Введение

В рамках задачи было создано две таблицы в базе данных: **User**, отражающей пользователя и **Node**, отражающей записи.

Библиотека hibernate даёт возможность описывать объекты базы данных как сущности, что позволяет автоматизировать перевод класса в базу данных.

Использование библиотеки Lombok позволило сильно сократить количество строк кода с помощью аннотаций, использующих шаблоны для инициализации класса с разным набором параметров, а также для описания методов получения/назначения значений из классов.

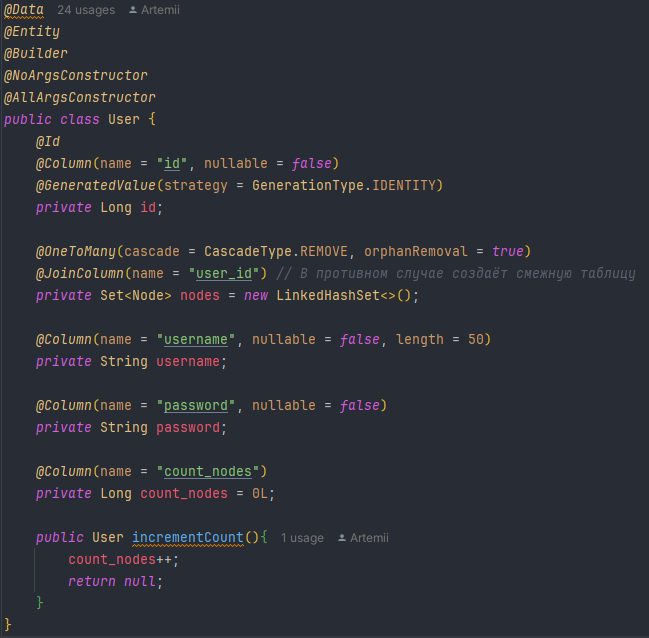
### User

User имеет следующие поля:

1. **id**
   1. Поле для идентификации пользователя
2. **username**
   1. Имя пользователя
3. **password** 
   1. Пароль
4. **count\_nodes**
   1. Количество записей

Также имеет переменную **nodes**, реализующую OneToMany связь с таблицей Node.

Функция **incrementCount** увеличивает значение count\_nodes на 1.

  
 Рис. 1. Код для таблицы User.

### Node

Node имеет следующие поля:

1. **id**
   1. Составной ключ для идентификации записи.  
      Состоит из:
      1. **User**
         1. Пользователь, кому принадлежит запись
      2. **id**
         1. Номер записи относительно пользователя
2. **name**
   1. Название записи
3. **text**
   1. Текст записи

Также имеет переменную **users**, реализующую ManyToOne связь с таблицей User.

Функция **incrementCount** увеличивает значение count\_nodes на 1.

Функция **setId** ставит нужный id, зависящий от количества записей у пользователя, и возвращает запись.

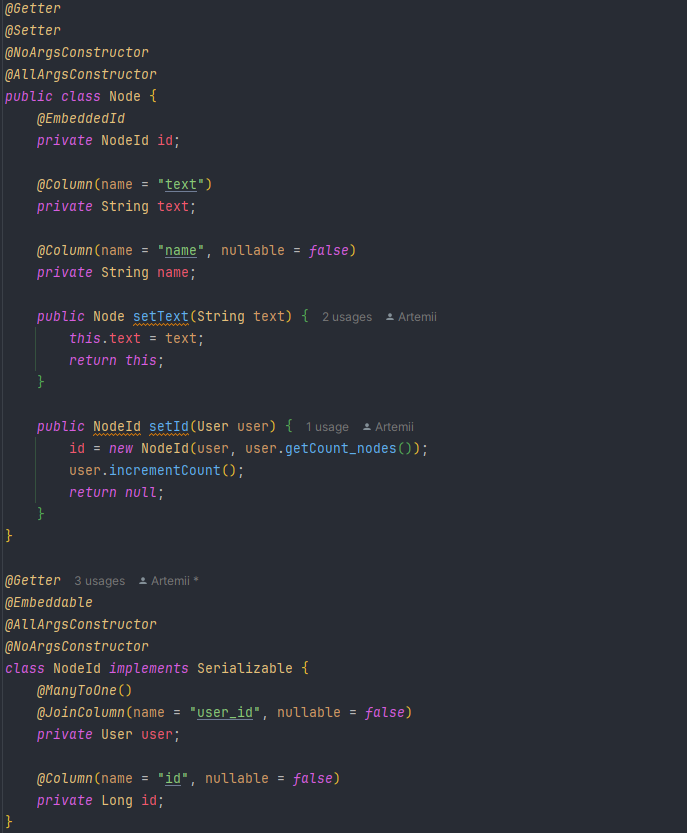


Рис. 2. Код для таблицы Node.

## DTO

**DTO — Data Transfer Object**, представляет собой объект, который содержит данные, необходимые для выполнения операции или запроса в приложении.

Их используют для увеличения безопасности при выводе данных с сервера, не передавая конфиденциальную информацию и/или не передавая данные, перехватив которые злоумышленники смогут получить доступ к аккаунту пользователя.

В приложении насчитывается 13 DTO. Часть из них используется для получения данных из интернет запроса (об этом будет рассказано далее), часть же используется для вывода информации.

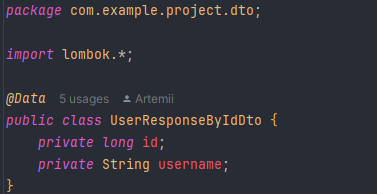


Рис. 3. Пример реализации DTO к таблице User.  
Названия переменных DTO совпадают с названиями переменных в User.

## Mapper

Mapper осуществляет конвертацию сущности (User и Node) в DTO.

Он позволяет автоматизировать процесс получения необходимых DTO параметров из сущности. Вместо того, чтобы ручками прописывать процесс переноса, достаточно назвать переменные в DTO также, как и в сущности, и mapper сможет самостоятельно перенести нужные данные.

Modelmapper по-умолчанию встроен в Spring, поэтому остаётся лишь задать стартовую конфигурацию, чтобы начать с ним работать.

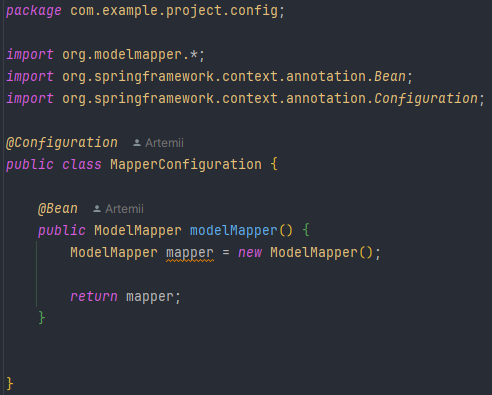


Рис. 4. Конфигурация Mapper-а.

## Миграции

Миграции — команды для базы данных, исполняемые при запуске программы. С помощью миграций можно расширить поля у таблиц, при этом сохранив старые, добавить новые таблицы и сделать всё то, что делается при помощи прямого обращения к базе данных.

Миграции позволяют делать базу данных версионной, чтобы в случае проблем можно было вернуться на старую версию и продолжить работу на ней.

В нашем случае миграции исполняются библиотекой Flyway.  
Для неё необходимо назвать файл миграций в особой форме, а именно:  
V\*номер\_версии\_через\_подчёркивания\*\_\_\*название\_миграции\*.sql

Далее останется лишь запустить программу и flyway применит новые, согласно таблице истории применения миграций, миграции к базе данных!



Рис. 5. Демонстрация наименования миграций

# Взаимодействие с базой данных

## Введение

Работа с базой данных в Spring разбита на несколько частей: Репозиторий, Сервис и Исключения. Про каждый из данных элементов будет рассказано ниже.

## Репозиторий

Класс, содержащий CRUD (Create Read Update Delete) методы для конкретной сущности.

Репозиторий имеет стандартные методы для различных операций над базой данных. Однако можно добавить свои методы, с помощью правильного синтаксиса, где:  
\*Тип возвращаемых данных\* \*действие\*\*Количество участников действия\*By\*Параметры, по которому будет происходить поиск\*(\*Аргументы\*);

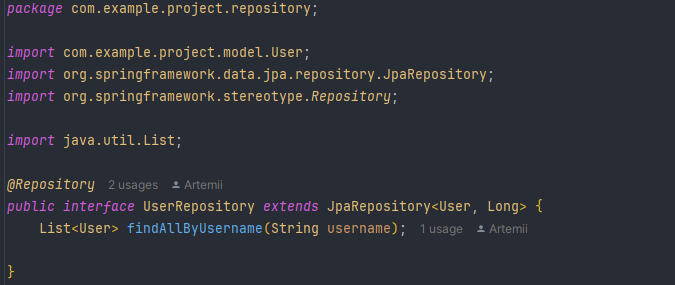


Рис. 6. Репозиторий User.  
Реализован метод поиска всех пользователей по имени.

## Сервис

Класс, отвечающий за непосредственную работу с базой данных.

Оперирует методами репозитория для исполнения команд, однако помимо этого в сервисе производятся различного рода проверки данных. В случае ошибки в данных производится выбрасывание исключения (о чём будет рассказано в следующем пункте).

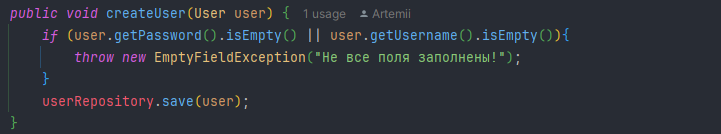


Рис. 7. Пример функции сервиса.  
Реализован метод сохранения пользователя.  
Также реализовано выбрасывание исключения

### Аутентификация

Метод UserService, сверяющий полученные логин/пароль и логин/пароль предполагаемого пользователя.

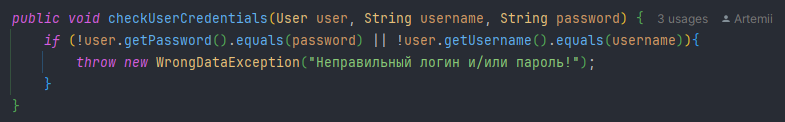


Рис. 8. Функция проверки данных.

## Исключения

Исключения вызываются при неправильных данных в сервисе.

Реализованы они через основной класс-контроллер, возвращающий DTO ошибки, основной класс ошибки, от которого наследуются исключения.

Такая система позволяет для разных типов исключений использовать разную логику.



Рис. 9. Пример исключения, наследующегося от основного класса ошибки ApiException.

# Интернет-запросы

## Введение

Существует несколько типов запросов, отличающихся по их наполнению и стандартам использования: **GET**, **POST**, **DELETE**, **PATCH**.

**GET** отвечает за получение информации с сервера, он менее надёжен, **POST** за отправку информации на сервер, более надёжен, **DELETE**, за удаление и **PATCH**, за обновление информации на сервере.

## Контроллер

### Введение

Контроллер используется для обработки информации из веб-запроса и использовании функций сервиса. В Spring-е есть несколько способов обработки информации из запроса: Взятие информации из тела запроса, из URL адреса запроса или же из заголовков.

Возвращает нужное DTO.

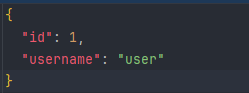


Рис. 10. Результат выполнения GET запроса по адресу localhost:8080/1

### Запрос без тела

Запрос, в котором вся информация передаётся только через URL



Рис. 11. Запрос без тела.

В Spring есть возможность получения данных из URL с помощью аннотации @PathVariable

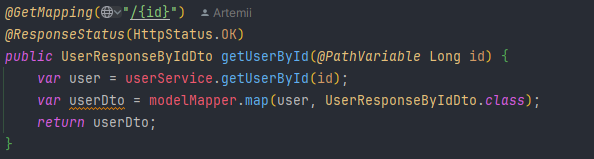


Рис. 12. Обработка запроса без тела.

### Запрос с телом

Запрос, информация в котором передаётся в теле запроса.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 13. Запрос с телом.

В Spring получение данных из тела происходит с помощью заведения DTO под нужные данные и аннотацию @RequestBody

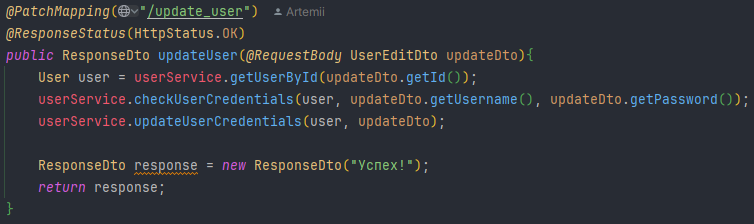


Рис. 14. Обработка запроса с телом

# Тестирование

Тестирование приложения было проведено силами InteliJ IDEA ULTIMATE.

Были созданы и запущены все виды запросов, в том числе и запросы с ошибками в данных, на что был выдан корректный результат.

Для более удобной демонстрации процесса тестирования был использован сторонний клиент Postman.

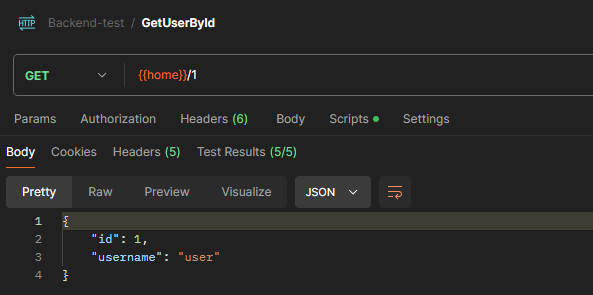


Рис. 15. Пример тестирования запроса без тела.  
Запрос должен был выдать id пользователя и его имя.

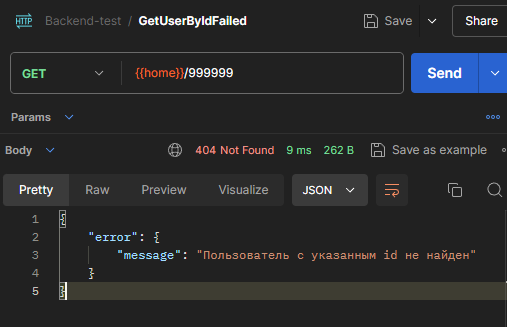


Рис. 16. Пример тестирования запроса без тела.  
Запрос должен был выдать ошибку.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рис. 17. Пример тестирования запроса с телом.  
Запрос должен был выдать список всех пользователей с именем L.

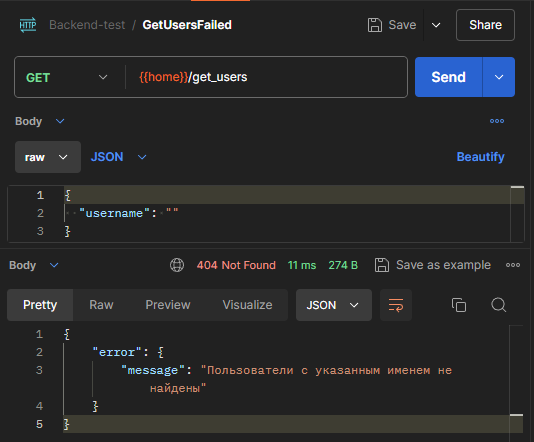


Рис.18. Пример тестирования запроса с телом.  
Запрос должен был выдать ошибку.

В таком же стиле было проведено тестирование всех остальных запросов.

# Вывод

# Источники

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

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

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

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

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

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

7. 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).

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