

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”»  
по образовательной программе подготовки бакалавров

по направлению 02.03.01 «Математика и компьютерные науки»

профиль «Сквозные цифровые технологии»

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

г. Владивосток

2024

Оглавление

[Актуальность 3](#_Toc171008744)

[Задачи 3](#_Toc171008745)

[Используемые технологии 4](#_Toc171008746)

[База данных 5](#_Toc171008747)

[Введение 5](#_Toc171008748)

[Установка базы данных 5](#_Toc171008749)

[Модели 5](#_Toc171008750)

[Введение 5](#_Toc171008751)

[Task 5](#_Toc171008752)

[DTO 6](#_Toc171008753)

[Введение 6](#_Toc171008754)

[Примеры DTO 6](#_Toc171008755)

[Использование DTO 7](#_Toc171008756)

[Mapper 7](#_Toc171008757)

[Введение 7](#_Toc171008758)

[Назначение 7](#_Toc171008759)

[Реализация 8](#_Toc171008760)

[Миграции базы данных 8](#_Toc171008761)

[Введение 8](#_Toc171008762)

[Использование библиотеки Flyway для миграций 8](#_Toc171008763)

[Взаимодействие с базой данных 9](#_Toc171008764)

[Введение 9](#_Toc171008765)

[Репозиторий 9](#_Toc171008766)

[TaskRepository 9](#_Toc171008767)

[Сервис 9](#_Toc171008768)

[Введение 9](#_Toc171008769)

[TaskService 9](#_Toc171008770)

[Исключения 10](#_Toc171008771)

[Введение 10](#_Toc171008772)

[ApiException 10](#_Toc171008773)

[TaskCreateException 10](#_Toc171008774)

[TaskNotFoundException 10](#_Toc171008775)

[TaskControllerExceptionHandler 10](#_Toc171008776)

[ApiExceptionHandler 11](#_Toc171008777)

[Заключение 11](#_Toc171008778)

[Обработка HTTP-запросов 12](#_Toc171008779)

[Введение 12](#_Toc171008780)

[Контроллеры 12](#_Toc171008781)

[Введение 12](#_Toc171008782)

[Запрос без тела 12](#_Toc171008783)

[Запрос с телом 12](#_Toc171008784)

[Заключение 13](#_Toc171008785)

[Тестирование 14](#_Toc171008786)

[Введение 14](#_Toc171008787)

[Тестирование с помощью Postman 14](#_Toc171008788)

[Тестирование с помощью IntelliJ IDEA 14](#_Toc171008789)

[Результаты тестирования 14](#_Toc171008790)

[Заключение 15](#_Toc171008791)

[Список используемых источников 16](#_Toc171008792)

[Приложение 17](#_Toc171008793)

[Приложение 1 17](#_Toc171008794)

[Приложение 2 18](#_Toc171008795)

[Приложение 3 19](#_Toc171008796)

[Приложение 4 19](#_Toc171008797)

[Приложение 5 20](#_Toc171008798)

[Приложение 6 20](#_Toc171008799)

[Приложение 7 21](#_Toc171008800)

[Приложение 8 21](#_Toc171008801)

[Приложение 9 22](#_Toc171008802)

[Приложение 10 22](#_Toc171008803)

[Приложение 11 23](#_Toc171008804)

[Приложение 12 24](#_Toc171008805)

[Приложение 13 24](#_Toc171008806)

[Приложение 14 25](#_Toc171008807)

[Приложение 15 25](#_Toc171008808)

[Приложение 16 26](#_Toc171008809)

[Приложение 17 27](#_Toc171008810)

[Приложение 18 28](#_Toc171008811)

**Введение**

## Актуальность

В современном мире веб-приложения играют ключевую роль в автоматизации различных бизнес-процессов и улучшении пользовательского опыта. С ростом числа пользователей и объемов данных возникает необходимость в разработке масштабируемых, надежных и безопасных систем. Проект SpringAPI направлен на решение задач, связанных с обработкой и хранением данных, предоставляя разработчикам удобный и гибкий инструмент для создания веб-сервисов на основе архитектуры REST.

## Задачи

Целью проекта SpringAPI является создание API, обеспечивающего эффективное взаимодействие между клиентскими приложениями и серверной частью. Основные задачи проекта включают:

1. Разработка структуры базы данных для хранения информации о задачах.

* Создание схемы базы данных с таблицей для задач.
* Определение полей таблицы и настройка их типов данных.
* Написание скриптов для создания и миграции базы данных с использованием Flyway.

1. Обеспечение CRUD-операций (создание, чтение, обновление, удаление) для различных сущностей.

* Разработка контроллеров для обработки запросов GET, POST, PUT и DELETE для сущностей "Task".
* Создание DTO для передачи данных между клиентом и сервером.
* Настройка сервисного слоя для обработки бизнес-логики и взаимодействия с репозиторием.

1. Обеспечение безопасности и обработки исключений.

* Реализация централизованной обработки исключений с помощью @ControllerAdvice.
* Создание пользовательских исключений и соответствующих обработчиков для корректного ответа на ошибки.
* Внедрение аннотаций для обработки исключений в контроллерах.

1. Проведение тестирования для проверки корректности работы API.

* Использование Postman для тестирования всех методов контроллеров, включая тестирование обработки ошибок.

## Используемые технологии

Для реализации проекта были выбраны следующие технологии:

1. **Spring Boot:** Фреймворк для создания производительных и надежных приложений на языке Java.
2. **Hibernate:** ORM-решение для работы с базой данных.
3. **PostgreSQL:** Реляционная база данных для хранения данных.
4. **Maven:** Инструмент для управления зависимостями и сборки проекта.
5. **Flyway:** Инструмент для управления миграциями базы данных. Flyway позволяет версионировать схему базы данных и автоматизировать процесс применения изменений. Это помогает поддерживать согласованность базы данных и упрощает развертывание изменений в различных средах. Flyway интегрируется с Spring Boot и может автоматически выполнять миграции при запуске приложения.

# База данных

## Введение

В данной главе описан процесс запуска базы данных, описание созданных моделей, процесс миграций и использование объектов передачи данных (DTO). База данных является основой для хранения информации о пользователях и узлах, обеспечивая надежное и масштабируемое хранилище данных для выполнения различных операций.

## Установка базы данных

Для запуска системы управления базой данных PostgreSQL используется Docker. Процесс установки и запуска СУБД в контейнере включает следующие шаги:

1. Создание файла docker-compose.yaml, в котором прописываются атрибуты образа (модель базы данных, которую нужно использовать), а также параметры контейнера, такие как логин, пароль, путь к контейнеру на виртуальной машине и используемые порты (Приложение 1).
2. Использование команды docker-compose up --build для загрузки нужного образа базы данных и создания контейнера. Docker находит нужный образ, скачивает его (если он еще не скачан), и на его основе создает и запускает контейнер.

## Модели

### Введение

В проекте SpringAPI используется одна модель данных Task, которая описывается как сущность с помощью библиотеки Hibernate. Использование библиотеки Lombok позволяет сократить количество строк кода, генерируя геттеры, сеттеры и другие стандартные методы.

### Task

Модель Task представляет собой задачу и включает следующие поля (Приложение 2):

* id: уникальный идентификатор задачи.
* title: название задачи.
* description: описание задачи.
* completed: статус выполнения задачи (завершена или нет).

## DTO

### Введение

DTO используются для передачи данных между слоями приложения, таких как контроллеры и сервисы, или между клиентом и сервером. Они помогают сократить избыточность данных, повышают безопасность, и упрощают сериализацию и десериализацию данных.

### Примеры DTO

#### ErrorDto (Приложение 3)

Представляет собой объект, содержащий сообщение об ошибке. Он используется для стандартизации ответов при возникновении ошибок в приложении.

#### ErrorResponseDto (Приложение 4)

Содержит объект ErrorDto и используется для возврата стандартизированного ответа при ошибках.

#### TaskCreateDto (Приложение 5)

Используется для создания новой задачи. Этот DTO содержит необходимые данные для создания задачи.

Поля:

* title: Название задачи.
* description: Описание задачи.
* completed: Статус выполнения задачи (завершена или нет).

#### TaskDto (Приложение 6)

Представляет собой объект для передачи данных задачи. Он используется для возврата информации о задаче.

Поля:

* title: Название задачи.
* description: Описание задачи.
* completed: Статус выполнения задачи.

#### TaskResponseDto (Приложение 7)

Используется для передачи данных ответа, содержащих идентификатор задачи.

### Использование DTO

DTO используются в контроллерах для получения данных из запросов и возврата данных в ответах. Примеры использования включают:

* При создании новой задачи, данные из TaskCreateDto передаются в сервис для обработки.
* При возврате информации о задаче, данные из TaskDto или TaskResponseDto используются для формирования ответа.

Использование DTO позволяет:

1. Изолировать внутренние модели данных от внешних представлений.
2. Упростить валидацию и сериализацию данных.
3. Улучшить безопасность приложения, возвращая только необходимые данные.

## Mapper

### Введение

Мапперы играют важную роль в приложениях, разделяющих бизнес-логику и слои представления данных. В контексте Spring API, мапперы используются для преобразования сущностей базы данных в DTO (Data Transfer Objects) и обратно. Это позволяет изолировать внутренние модели данных от внешних представлений и упрощает сериализацию и десериализацию данных.

### Назначение

Использование мапперов имеет несколько ключевых преимуществ:

1. **Сокращение шаблонного кода:** Мапперы автоматизируют процесс преобразования, уменьшая количество кода, необходимого для ручного маппинга полей.
2. **Повышение читаемости и поддержки кода:** Мапперы делают код более понятным и поддерживаемым, поскольку логика маппинга сосредоточена в одном месте.
3. **Безопасность:** Мапперы помогают скрывать внутренние реализации сущностей от внешних клиентов, что повышает безопасность приложения.

### Реализация

В проекте SpringAPI используется библиотека ModelMapper для автоматизации процесса преобразования. Конфигурация маппера задается в классе TaskMapper (Приложение 8).

Метод convertToDto принимает объект Task и возвращает его представление в виде TaskDto.

Метод convertToEntity принимает объект TaskDto и возвращает его представление в виде сущности Task.

## Миграции базы данных

### Введение

Миграции — это команды для базы данных, которые выполняются при запуске программы. С помощью миграций можно добавлять новые таблицы, изменять существующие поля, сохраняя старые данные, и выполнять другие операции, аналогичные прямым запросам к базе данных. Миграции делают базу данных версионной, что позволяет в случае необходимости вернуться к предыдущей версии и продолжить работу.

### Использование библиотеки Flyway для миграций

В проекте SpringAPI миграции выполняются с использованием библиотеки Flyway. Для того чтобы Flyway распознал миграцию, файл миграции должен быть назван в специальном формате по типу (Приложение 9).

При запуске Flyway автоматически применяет миграции, основываясь на их версии, и записывает информацию о выполненных миграциях в таблицу истории миграций.

# Взаимодействие с базой данных

## Введение

Взаимодействие с базой данных является ключевой частью любого приложения. В проекте для управления данными используются репозитории, сервисы и обработчики исключений. Репозитории предоставляют интерфейсы для выполнения операций с базой данных, сервисы инкапсулируют бизнес-логику, а обработчики исключений обеспечивают корректную обработку ошибок.

## Репозиторий

### TaskRepository

Репозитории предоставляют методы для выполнения операций CRUD с сущностями базы данных. В проекте используется интерфейс TaskRepository, который наследует JpaRepository для работы с сущностью Task (Приложение 10).

Методы:

1. findAll(): Возвращает список всех задач.
2. findById(Long id): Находит задачу по её идентификатору.
3. save(Task task): Сохраняет или обновляет задачу.
4. deleteById(Long id): Удаляет задачу по её идентификатору.

## Сервис

### Введение

Сервисный слой инкапсулирует бизнес-логику приложения, обеспечивая взаимодействие между контроллерами и репозиториями. Сервисы обрабатывают запросы, выполняют валидацию данных и вызывают методы репозиториев для взаимодействия с базой данных.

### TaskService

TaskService предоставляет методы для управления задачами. Он использует TaskRepository для выполнения операций с базой данных. (Приложение 11)

Методы:

1. getTasks(): Возвращает список всех задач.
2. addNewTask(Task newTask): Добавляет новую задачу.
3. findTaskById(long taskId): Находит задачу по её идентификатору.
4. deleteTask(Long id): Удаляет задачу по её идентификатору.
5. switchCompleted(Long taskId): Переключает статус завершенности задачи.

## Исключения

### Введение

Обработка исключений является важной частью разработки приложений, поскольку позволяет обеспечить корректную работу системы в случае возникновения ошибок. В SpringAPI для централизованной обработки исключений используется аннотация @ControllerAdvice.

### ApiException

ApiException является базовым классом для всех пользовательских исключений в приложении. Он расширяет RuntimeException и добавляет поле для хранения HTTP-статуса ошибки (Приложение 12).

Поля:

1. httpStatus: HTTP-статус ошибки.
2. message: Сообщение об ошибке.

### TaskCreateException

TaskCreateException выбрасывается при возникновении ошибок, связанных с созданием задач, таких как пустые поля (Приложение 13).

Поля:

* message: Сообщение об ошибке.

### TaskNotFoundException

TaskNotFoundException выбрасывается, когда задача с указанным идентификатором не найдена в базе данных (Приложение 14).

### TaskControllerExceptionHandler

TaskControllerExceptionHandler является аннотацией для маркировки контроллеров, исключения которых будут обрабатываться ApiExceptionHandler (Приложение 15).

### ApiExceptionHandler

@ControllerAdvice используется для глобальной обработки исключений, выбрасываемых контроллерами, аннотированными @TaskControllerExceptionHandler (Приложение 16).

Методы:

* handleCreationException(ApiException exception): Обрабатывает пользовательские исключения и возвращает соответствующий HTTP-статус и сообщение об ошибке.

### Заключение

Исключения и их обработка играют важную роль в обеспечении устойчивости и надежности приложения. В проекте SpringAPI пользовательские исключения, такие как TaskCreateException и TaskNotFoundException, помогают управлять ошибками, связанными с задачами, в то время как @ControllerAdvice обеспечивает централизованную обработку этих исключений.

# Обработка HTTP-запросов

## Введение

В этой главе рассматриваются аспекты обработки HTTP-запросов в проекте. Контроллеры играют ключевую роль в обработке HTTP-запросов и отправке соответствующих ответов клиентам. Они связывают клиентские запросы с бизнес-логикой, реализованной в сервисах.

## Контроллеры

### Введение

Контроллеры в проекте отвечают за маршрутизацию и обработку HTTP-запросов. Они принимают входящие запросы, взаимодействуют с сервисным слоем и возвращают соответствующие ответы. В проекте используются аннотации Spring для определения маршрутов и типов запросов.

### Запрос без тела

Запросы без тела (например, GET-запросы) используются для получения данных с сервера без необходимости отправки данных в теле запроса (Приложение 17).

Методы:

1. getTask(): Обрабатывает GET-запрос для получения списка всех задач.
2. getTask(@PathVariable long taskId): Обрабатывает GET-запрос для получения задачи по её идентификатору.

### Запрос с телом

Запросы с телом (например, POST, PUT-запросы) используются для отправки данных на сервер для создания или обновления ресурсов.

Методы:

1. CreateTask(@RequestBody TaskCreateDto taskCreateDto): Обрабатывает POST-запрос для создания новой задачи.
2. patchTask(@PathVariable("taskId") Long taskId): обрабатывает PUT-запрос для обновления задачи по её идентификатору.
3. deleteTask(@PathVariable("taskId") Long id): обрабатывает DELETE-запрос для удаления задачи по её идентификатору.

## Заключение

В данной главе рассмотрены ключевые аспекты обработки HTTP-запросов в проекте SpringAPI. Контроллеры играют центральную роль в маршрутизации запросов и обеспечении взаимодействия с бизнес-логикой через сервисы. Они обрабатывают различные типы запросов, такие как GET, POST, PUT и DELETE, и возвращают соответствующие ответы клиентам.

Использование аннотаций Spring позволяет легко настраивать маршруты и управлять логикой обработки запросов. Контроллеры и связанные с ними компоненты обеспечивают надежное взаимодействие между клиентом и сервером, способствуя созданию эффективного и масштабируемого веб-приложения.

# Тестирование

## Введение

Тестирование является важной частью разработки программного обеспечения, обеспечивающей качество и надежность приложения. В проекте тестирование проводилось с использованием инструментов Postman и IntelliJ IDEA. Эти инструменты позволяют автоматизировать тестирование и проверить корректность работы API.

## Тестирование с помощью Postman

В рамках тестирования проекта с помощью Postman проверялись различные сценарии использования API, включая создание, получение, обновление и удаление задач.

## Тестирование с помощью IntelliJ IDEA

Тесты были направлены на проверку корректности работы контроллеров, сервисов и обработки исключений.

## Результаты тестирования

Тестирование показало, что все конечные точки API работают корректно и возвращают ожидаемые результаты. Были проверены различные сценарии, включая создание, получение, обновление и удаление задач. Обработка ошибок также была проверена для обеспечения надлежащей реакции на некорректные запросы. Тестирование с использованием Postman и IntelliJ IDEA позволило удостовериться в корректной работе API проекта. Оно обеспечило выявление и исправление ошибок при разработке, что способствовало повышению качества и надежности.

# Заключение

Реализация проекта SpringAPI позволила глубже изучить и применить на практике различные аспекты разработки с использованием Java и Spring Boot. В процессе работы были изучены и освоены следующие ключевые области:

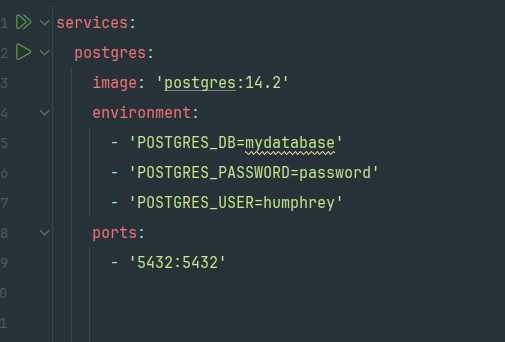
1. Проектирование и реализация модульной архитектуры позволили достичь гибкости и масштабируемости приложения. Была реализована четкая структура, разделяющая ответственность между контроллерами, сервисами и репозиториями, что упростило сопровождение и расширение функциональности.
2. Использование PostgreSQL в качестве реляционной базы данных обеспечило надежное хранение и доступ к данным. Интеграция Hibernate для ORM упростила работу с базой данных, позволив использовать объектно-ориентированный подход. Миграции с помощью Flyway обеспечили версионность базы данных и упростили процесс обновления схемы.
3. Реализация различных типов HTTP-запросов (GET, POST, PUT, DELETE) позволила создать функциональные конечные точки API для взаимодействия с клиентами. Использование аннотаций Spring упростило настройку маршрутизации и обработку запросов, обеспечив чистый и понятный код.
4. Применение объектов передачи данных (DTO) и их преобразование с помощью ModelMapper упростили процесс передачи данных между слоями приложения. Это повысило безопасность и удобство работы с данными, сократив количество шаблонного кода.

# Список используемых источников

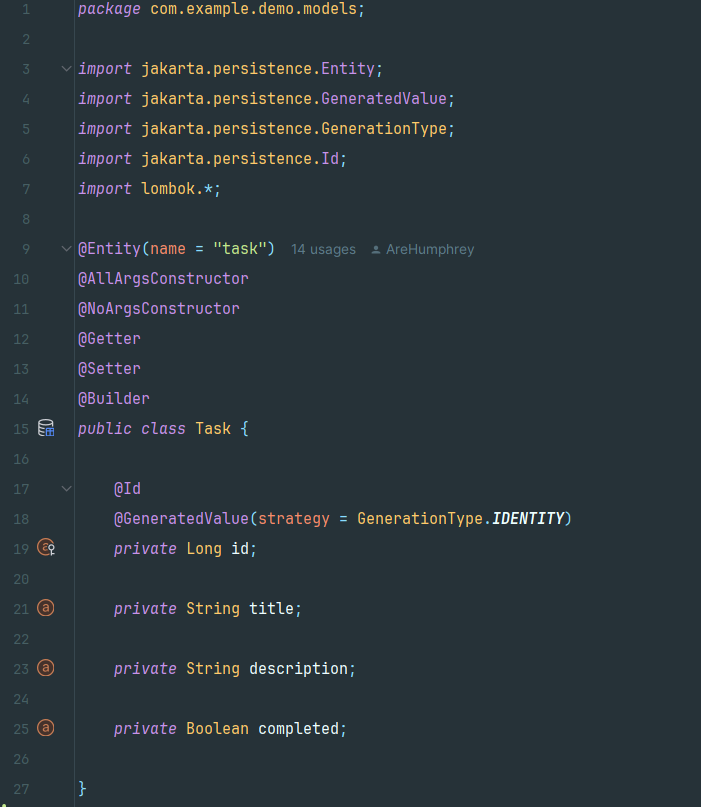
1. Spring Boot Reference Documentation [Электронный ресурс]. URL: <https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
2. Hibernate ORM Documentation [Электронный ресурс]. URL: https://hibernate.org/orm/documentation/ (дата обращения: 22.06.2024)
3. PostgreSQL Documentation [Электронный ресурс]. URL: <https://www.postgresql.org/docs/>
4. Flyway Documentation [Электронный ресурс]. URL: https://flywaydb.org/documentation/
5. ModelMapper Documentation [Электронный ресурс]. URL: http://modelmapper.org/getting-started/
6. Spring Security Reference [Электронный ресурс]. URL: <https://docs.spring.io/spring-security/site/docs/current/reference/html5/>
7. Postman Learning Center [Электронный ресурс]. URL: https://learning.postman.com/docs/getting-started/introduction/
8. Lombok - documentation [Электронный ресурс]. URL: <https://projectlombok.org/features/>

# Приложение

## Приложение 1



## Приложение 2



## Приложение 3



## Приложение 4



## Приложение 5



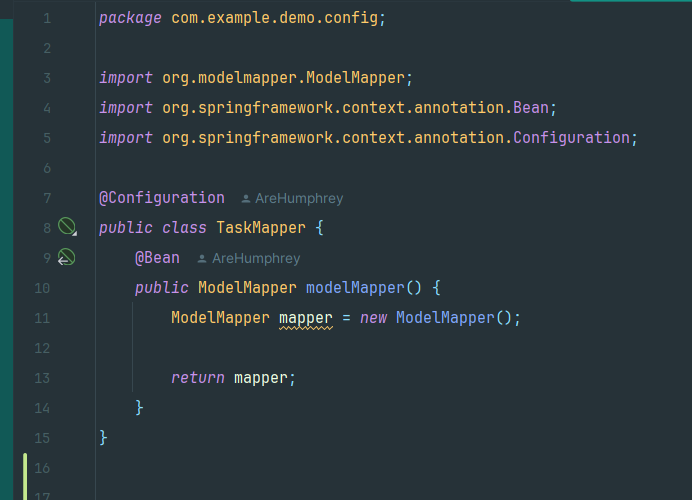
## Приложение 6



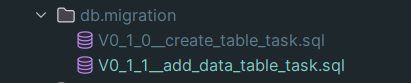
## Приложение 7



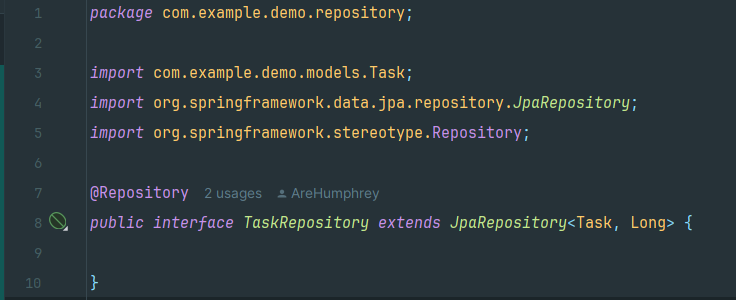
## Приложение 8



## Приложение 9



## Приложение 10



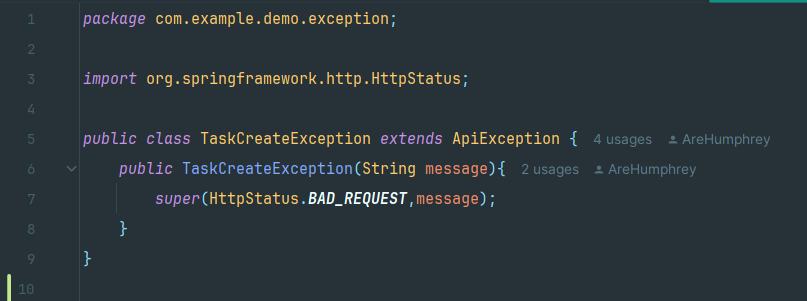
## Приложение 11

*package* com.example.demo.service;  
  
*import* com.example.demo.dto.TaskResponseDto;  
*import* com.example.demo.exception.TaskCreateException;  
*import* com.example.demo.exception.TaskNotFoundException;  
*import* com.example.demo.models.Task;  
*import* com.example.demo.repository.*TaskRepository*;  
*import* jakarta.transaction.Transactional;  
*import* lombok.RequiredArgsConstructor;  
*import* org.springframework.stereotype.Service;  
  
*import* java.util.ArrayList;  
*import* java.util.*List*;  
  
@Service  
@RequiredArgsConstructor  
*public class* TaskService {  
 *private final TaskRepository* taskRepository;  
  
 *public List*<Task> getTasks() {  
 *List*<Task> tasks = *new* ArrayList<>();  
 taskRepository.findAll().forEach(tasks::add);  
 *return* tasks;  
 }  
  
 *public* Long addNewTask(Task newTask) {  
 *if* (newTask.getTitle().strip().isEmpty()) {  
 *throw new* TaskCreateException("Поле заголовка не должно быть пустым");  
 }  
 *if* (newTask.getDescription().strip().isEmpty()) {  
 *throw new* TaskCreateException("Поле описания не должно быть пустым");  
 }  
 *return* taskRepository.save(newTask).getId();  
 }  
  
 *public* Task findTaskById(*long* taskId) {  
   
 *return* taskRepository.findById(taskId).orElseThrow(() -> *new* TaskNotFoundException(  
 "Задача с id = " + taskId + " не существует"  
 ));  
 }  
  
 *public* Long deleteTask(Long id) {  
 taskRepository.findById(id).orElseThrow(() -> *new* TaskNotFoundException(  
 "Задача с id = " + id + " не существует"  
 ));  
 taskRepository.deleteById(id);  
 *return* id;  
 }  
  
 @Transactional  
 *public void* switchCompleted(Long taskId) {  
 Task task = findTaskById(taskId);  
 task.setCompleted(!task.getCompleted());  
 taskRepository.save(task);  
 }  
}

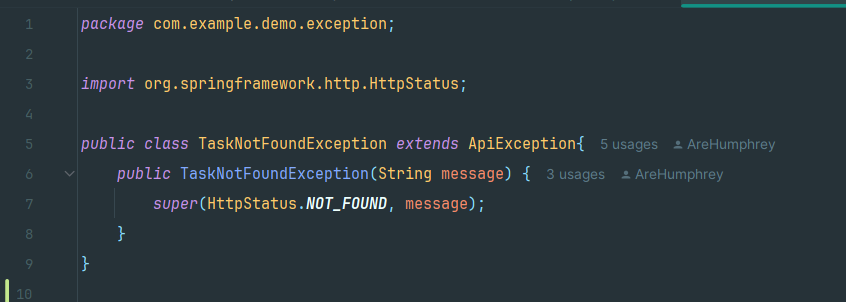
## Приложение 12



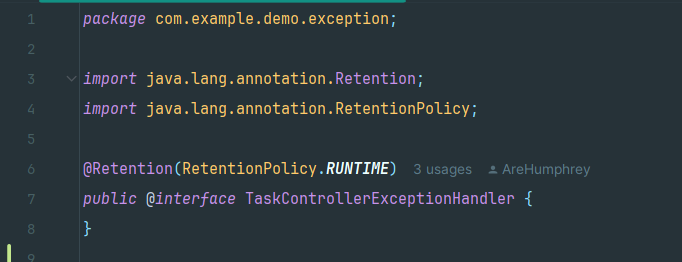
## Приложение 13



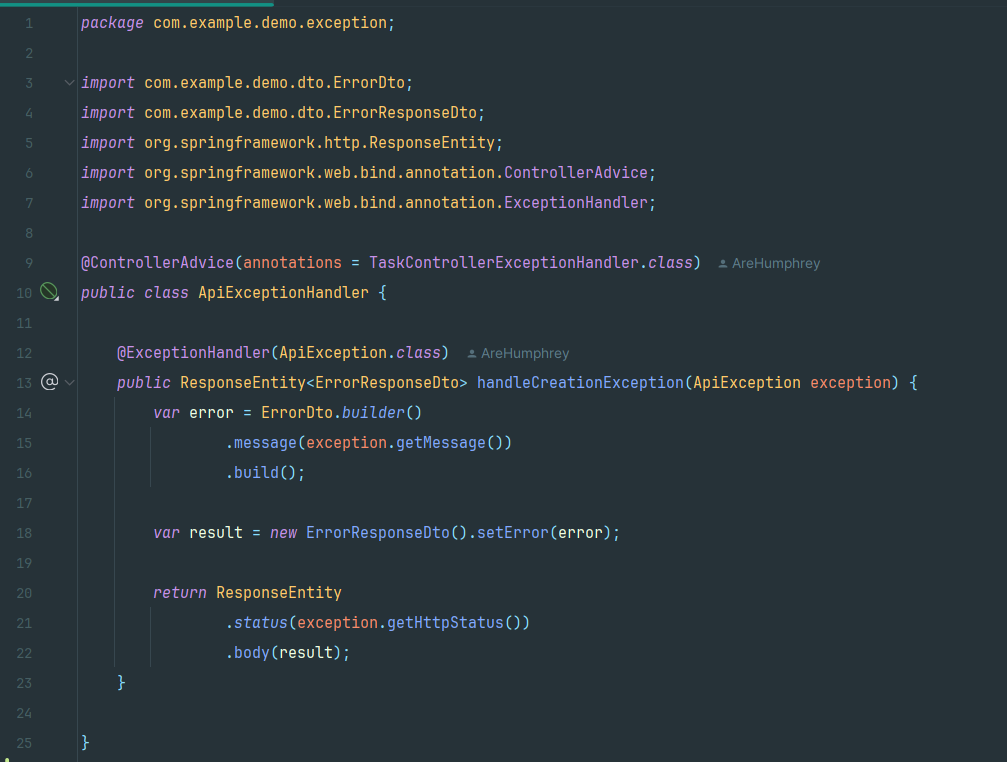
## Приложение 14



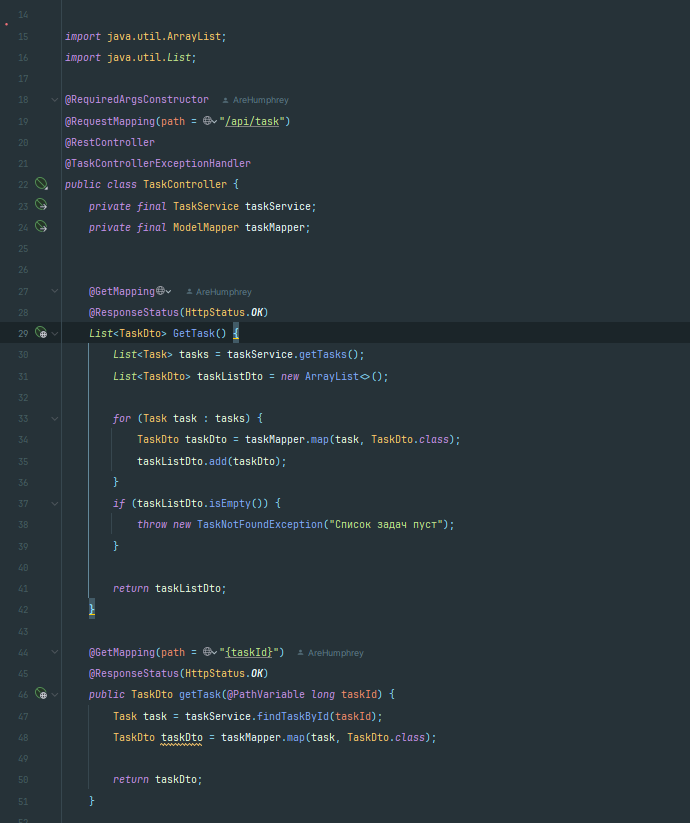
## Приложение 15



## Приложение 16



## Приложение 17



## Приложение 18

