

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** | | |
|  | |
|  |
|  |

**КУРСОВОЙ ПРОЕКТ**

по дисциплине «Веб-программирование (Backend)»

на тему «Backend разработка приложения для списка членов»

по образовательной программе подготовки бакалавров

по направлению 02.03.01 «Математика и компьютерные науки»

профиль «Сквозные цифровые технологии»

|  |  |
| --- | --- |
| Оценка \_\_\_\_\_\_\_  Регистрационный номер \_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г. | Выполнил студент группы  № Б9121-02.03.01сцт  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Иннокентьев П.В.  (подпись)  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024г.  Руководитель \_\_\_ассистент\_\_\_\_  (должность, ученое звание)  \_Охроменко Дарья Александровна\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) (ФИО)  «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2024г. |
|  |  |

г. Владивосток

2024

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc172386833)

[ЦЕЛИ И ЗАДАЧИ 4](#_Toc172386834)

[БАЗА ДАННЫХ 7](#_Toc172386835)

[УСТАНОВКА БАЗЫ ДАННЫХ 7](#_Toc172386836)

[МОДЕЛИ 8](#_Toc172386837)

[ОБЪЕКТЫ ПЕРЕДАЧИ ДАННЫХ (DTO) 9](#_Toc172386838)

[МАППЕРЫ 12](#_Toc172386839)

[МИГРАЦИИ БАЗЫ ДАННЫХ 13](#_Toc172386840)

[ВЗАИМОДЕЙСТВИЕ С БАЗОЙ ДАННЫХ 14](#_Toc172386841)

[РЕПОЗИТОРИЙ 14](#_Toc172386842)

[СЕРВИС 15](#_Toc172386843)

[ИСКЛЮЧЕНИЯ 16](#_Toc172386844)

[ИНТЕРНЕТ ЗАПРОСЫ 18](#_Toc172386845)

[КОНТРОЛЛЕРЫ 19](#_Toc172386846)

[ТЕСТИРОВАНИЕ 21](#_Toc172386847)

[ЗАКЛЮЧЕНИЕ 23](#_Toc172386848)

[ИСТОЧНИКИ 24](#_Toc172386849)

# ВВЕДЕНИЕ

Backend-разработка является неотъемлемой частью создания современных веб-приложений и сервисов, обеспечивая функциональность и логику, необходимые для работы фронтенда и интеграции с другими системами. В отличие от frontend-разработки, которая фокусируется на внешнем виде и взаимодействии с пользователем, backend-разработка отвечает за обработку данных, управление базами данных, обеспечение безопасности и масштабируемости, а также интеграцию с внешними сервисами и API.

Backend-разработка может включать использование различных языков программирования, таких как Java, Python, Ruby, PHP, а также фреймворков и библиотек, которые упрощают процесс разработки и обеспечивают высокое качество кода. Среди наиболее популярных backend-фреймворков можно выделить Spring (Java), Django (Python), Ruby on Rails (Ruby) и Laravel (PHP).

Backend-разработчики отвечают за обработку запросов от клиента, выполнение необходимых операций с данными и формирование ответа, который будет возвращен клиенту.Также они отвечают за проектирование и реализацию схем баз данных, а также за внесение и извлечение данных из баз данных. Разработчики должны обеспечить защиту данных от несанкционированного доступа и неправомерного использования, а также реализовать механизмы аутентификации и авторизации. Они должны стремиться к созданию масштабируемых и высокопроизводительных систем, которые могут обрабатывать большие объемы данных и обеспечивать быстрый отклик для пользователей.

В целом, backend-разработка является сложной и ответственной задачей, которая требует глубокого понимания технологий и методов, а также умения работать в команде и сотрудничать с другими специалистами. Однако, успешное решение задач backend-разработки может обеспечить высокое качество и функциональность веб-приложений и сервисов, которые пользуются популярностью у пользователей и клиентов.

# ЦЕЛИ И ЗАДАЧИ

Целью курсовой работы была реализация backend части приложения на языке программирования Java для работы со списками задач различных пользователей, с системой управления базами данных (СУБД) PostgreSQL.

ОСНОВНЫЕ ЗАДАЧИ ПРОЕКТА:

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.

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

• Разработка контроллеров для обработки запросов GET, POST, PUT и DELETE для сущностей "Member".

• Создание DTO для передачи данных между клиентом и сервером.

• Настройка сервисного слоя для обработки бизнес-логики и взаимодействия с репозиторием.

3. 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.

4. Проведение тестирования для проверки корректности работы API

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

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.

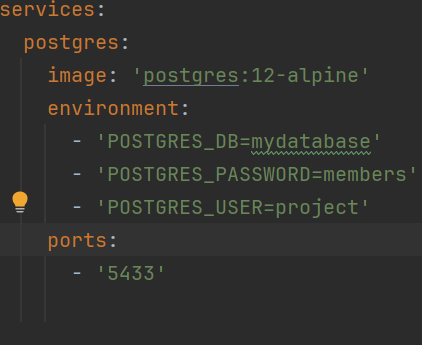
# БАЗА ДАННЫХ

В данной главе 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.



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

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

MEMBER

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

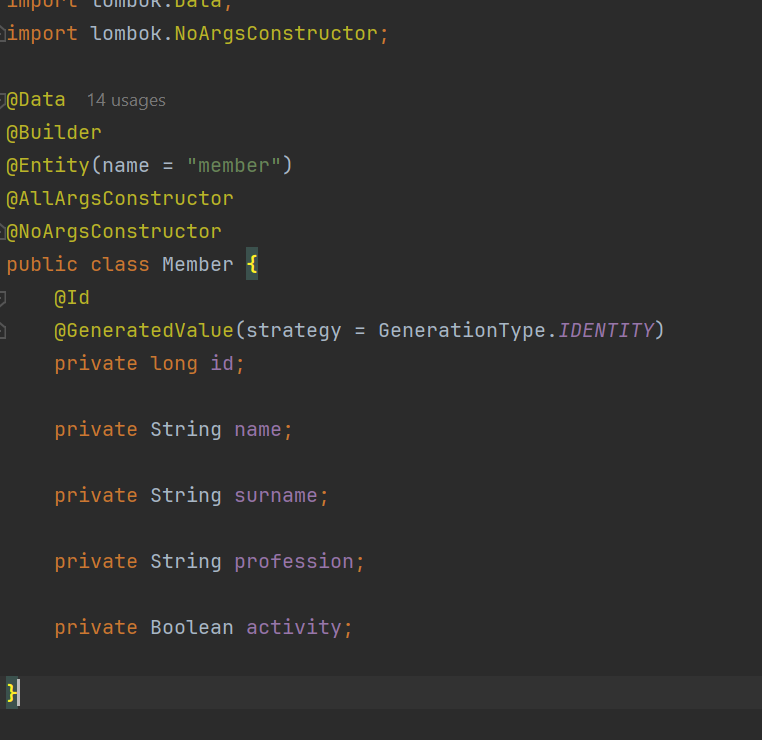
• id: уникальный идентификатор задачи.

• name: имя пользователя.

• surname: фамилия пользователя.

• profession: профессия пользователя.

• activity: статус активности.



# 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. 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.

Примеры DTO из проекта:

1. ErrorDto

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



2. ErrorResponseDto

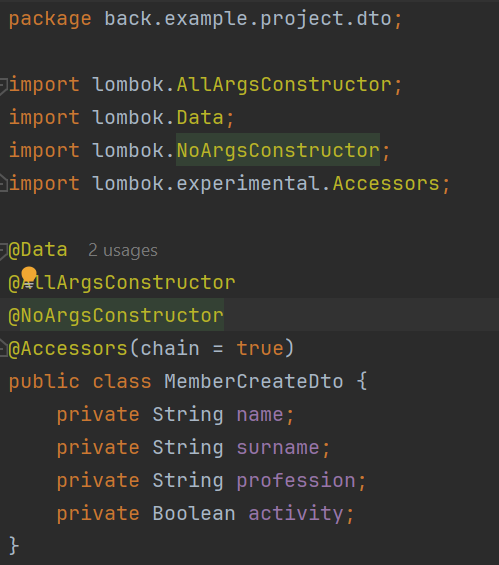
Содержит объект ErrorDto и используется для возврата стандартизированного ответа при ошибках.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

3. MemberCreateDto

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



4. MemberDto

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

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

5. MemberResponseDto

Используется для передачи данных ответа, содержащих идентификатор задачи.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Основные преимущества использования dto в backend-разработке:

Использование DTO позволяет избежать прямого доступа к слоям доступа к данным, что уменьшает связанность между слоями приложения и позволяет легче вносить изменения в код. DTO позволяет ограничить доступ к данным, предоставляя только те данные, которые необходимы для выполнения определенной операции. Это помогает защитить данные от несанкционированного доступа и неправомерного использования. Также DTO упрощает интеграцию с внешними системами и API, позволяя определять общий формат данных для обмена информацией между системами. Использование DTO может улучшить производительность приложения, поскольку они могут быть настроены для передачи только необходимых данных, что уменьшает объем передаваемых данных и ускоряет процесс передачи.

В общем DTO позволяет уменьшить связанность между слоями приложения, улучшить безопасность и производительность, а также упростить интеграцию с внешними системами и API.

# МАППЕРЫ

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.

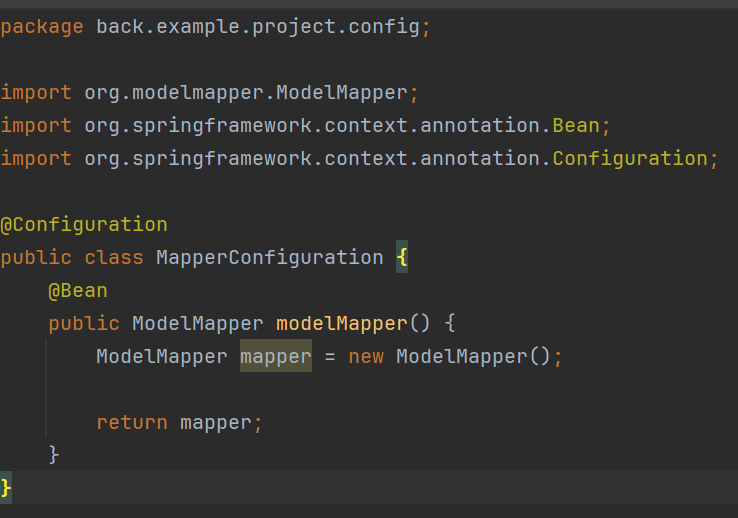
Использование мапперов имеет несколько ключевых преимуществ:

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.

В проекте SpringAPI используется библиотека ModelMapper для автоматизации процесса преобразования. Конфигурация маппера задается в классе MapperConfiguration.



Метод convertToDto принимает объект Member и возвращает его представление в виде MemberDto.

Метод convertToEntity принимает объект MemberDto и возвращает его представление в виде сущности Member.

# МИГРАЦИИ БАЗЫ ДАННЫХ

Миграции базы данных (database migrations) - это процесс изменения структуры базы данных, такой как добавление, изменение или удаление таблиц, столбцов, индексов и других объектов базы данных. Миграции базы данных являются важной частью разработки backend-приложений, поскольку они позволяют вносить изменения в структуру базы данных без потери данных и с минимальными проблемами совместимости. Основные преимущества использования миграций базы данных в backend-разработке:

Миграции базы данных позволяют вносить изменения в структуру базы данных на этапе разработки и тестирования, что упрощает процесс и позволяет обнаруживать ошибки на ранних стадиях. Также миграции позволяют вносить изменения в структуру базы данных без потери данных и с минимальными проблемами совместимости, что позволяет поддерживать стабильную работу приложения. Они позволяют автоматизировать процесс изменения структуры базы данных на этапе развертывания, что упрощает процесс и уменьшает время выполнения. Миграции улучшают отслеживание изменения в структуре базы данных и обеспечивать версионирование изменений, что позволяет легко откатить изменения или перейти к предыдущим версиям.

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:



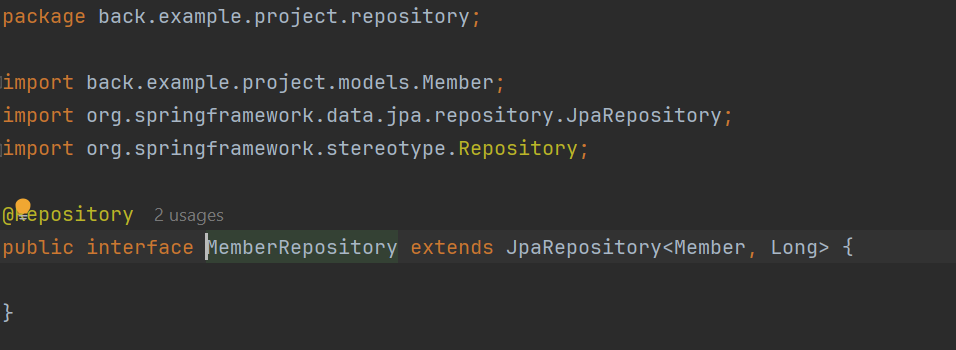
При запуске Flyway автоматически применяет миграции, основываясь на их версии, и записывает информацию о выполненных миграциях в таблицу истории миграций.

# 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

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). В проекте используется интерфейс MemberRepository, который наследует JpaRepository для работы с сущностью Member.



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

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).

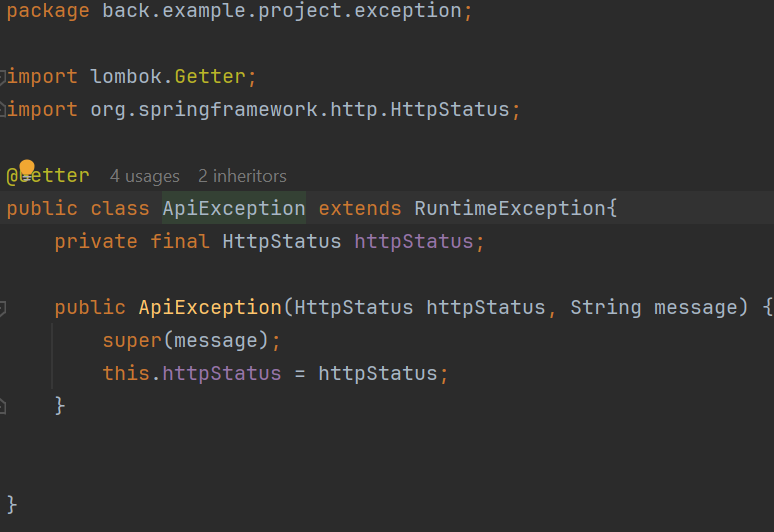
MemberService предоставляет методы для управления задачами. Он использует MemberRepository для выполнения операций с базой данных.



# ИСКЛЮЧЕНИЯ

Исключения (exceptions) – это механизмы, которые позволяют обрабатывать ошибки и необычные ситуации, возникающие во время выполнения программы. Исключения позволяют отделить логику обработки ошибок от основной логики программы, что улучшает читаемость и поддерживаемость кода.

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



CreateException выбрасывается при возникновении ошибок, связанных с созданием задач, таких как пустые поля.



NotFoundException выбрасывается, когда задача с указанным идентификатором не найдена в базе данных. Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

ControllerExceptionHandler является аннотацией для маркировки контроллеров, исключения которых будут обрабатываться ApiExceptionHandler Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

@ControllerAdvice используется для глобальной обработки исключений, выбрасываемых контроллерами, аннотированными @ControllerExceptionHandler.

# ИНТЕРНЕТ ЗАПРОСЫ

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

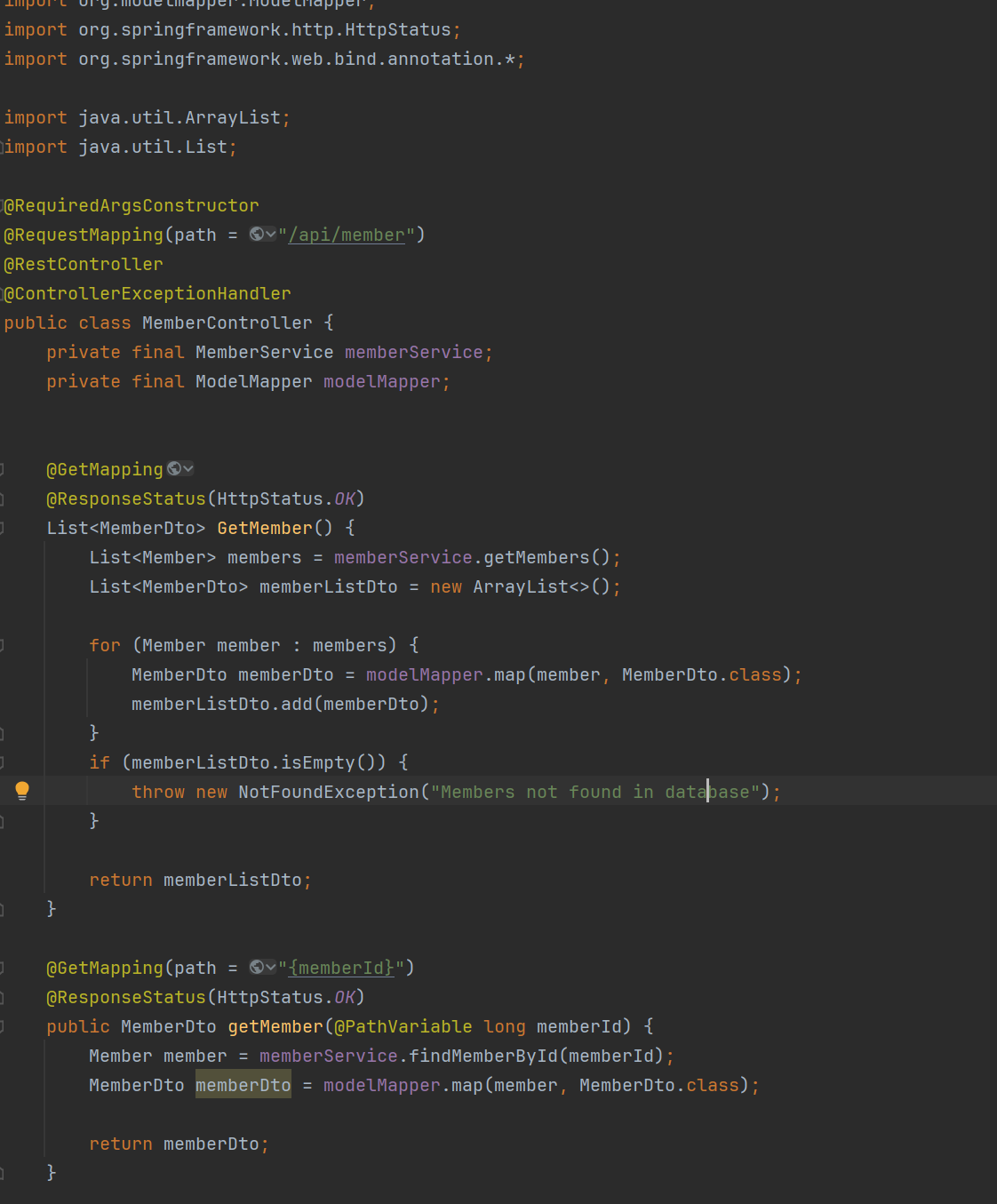
GET отвечает за получение информации с сервера, он менее надёжен, POST за отправку информации на сервер, более надёжен, DELETE, за удаление и PATCH, за обновление информации на сервере.

Всего было написано 5 запросов для User, 2 из которых имеют тип GET, и 4 запроса для Node.

# КОНТРОЛЛЕРЫ

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.

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



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

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

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

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 продемонстрировала возможности и преимущества использования фреймворка Spring для создания RESTful веб-сервисов. В ходе проекта были достигнуты следующие результаты:

Создание RESTful API: Были разработаны и реализованы основные эндпоинты для выполнения CRUD-операций (создание, чтение, обновление, удаление) над сущностями.

Использование Spring Boot: Проект был построен с использованием Spring Boot, что упростило настройку и конфигурацию приложения, а также ускорило процесс разработки.

Интеграция с базой данных: Были использованы Spring Data JPA и Hibernate для взаимодействия с базой данных, что обеспечило удобный и эффективный доступ к данным.

Обработка исключений: Были реализованы механизмы обработки исключений, что повысило надежность и устойчивость приложения.

Тестирование: Были написаны и выполнены юнит-тесты и интеграционные тесты для проверки корректности работы API и взаимодействия с базой данных.

Документация API: Для документирования API был использован Swagger, что упростило процесс тестирования и интеграции с другими системами.

В результате, проект SpringAPI продемонстрировал высокую производительность, масштабируемость и гибкость, что делает его отличным выбором для разработки современных веб-сервисов. Реализация данного проекта позволила получить ценный опыт работы с фреймворком Spring и его компонентами, а также улучшить навыки разработки и тестирования RESTful API. и удобство работы с данными, сократив количество шаблонного кода.

# ИСТОЧНИКИ

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