Для запуска настроить всю необходимую инфраструктуру. Протестировать ВСЕ обработчики api status приложения.

Диаграмму классов данного проекта, цель диаграммы классов – демонстрация архитектуры – отношения между компонентами и их функциональность.

* API

Интерфейс взаимодействия приложения с внешним миром,

Реализация взаимодействия внешнего мира и модели

* MODEL

Объекты предметной области и сценарии работы с ними, не зависит от других слоев

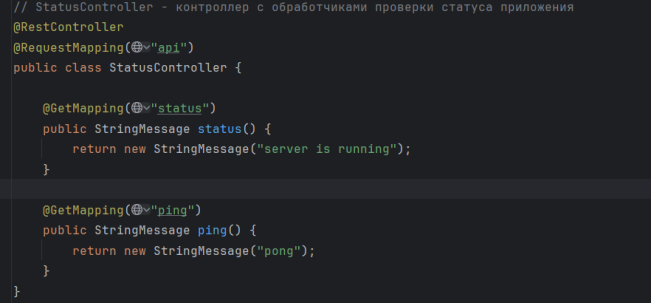
* Storage

Хранилище данных, зависит от модели посредством работы с сущностью модели и имплементации необходимых для работы модели интерфейсов

1. Api

* Controller

StatusController - контроллер с обработчиками проверки статуса приложения

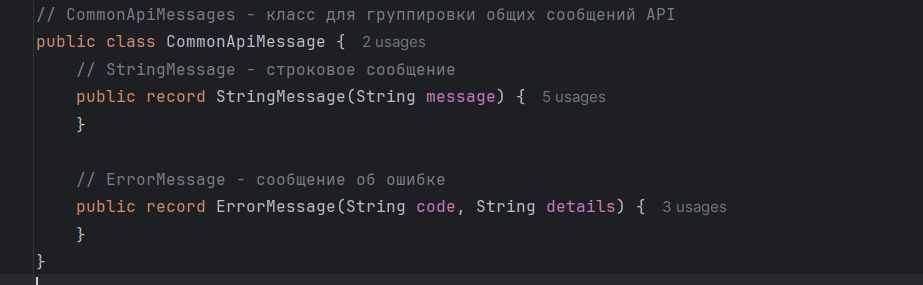


ApplicationErrorController – контроллер для обработки ошибок (собственный)



* Message

CommonApiMessages - класс для группировки общих сообщений API



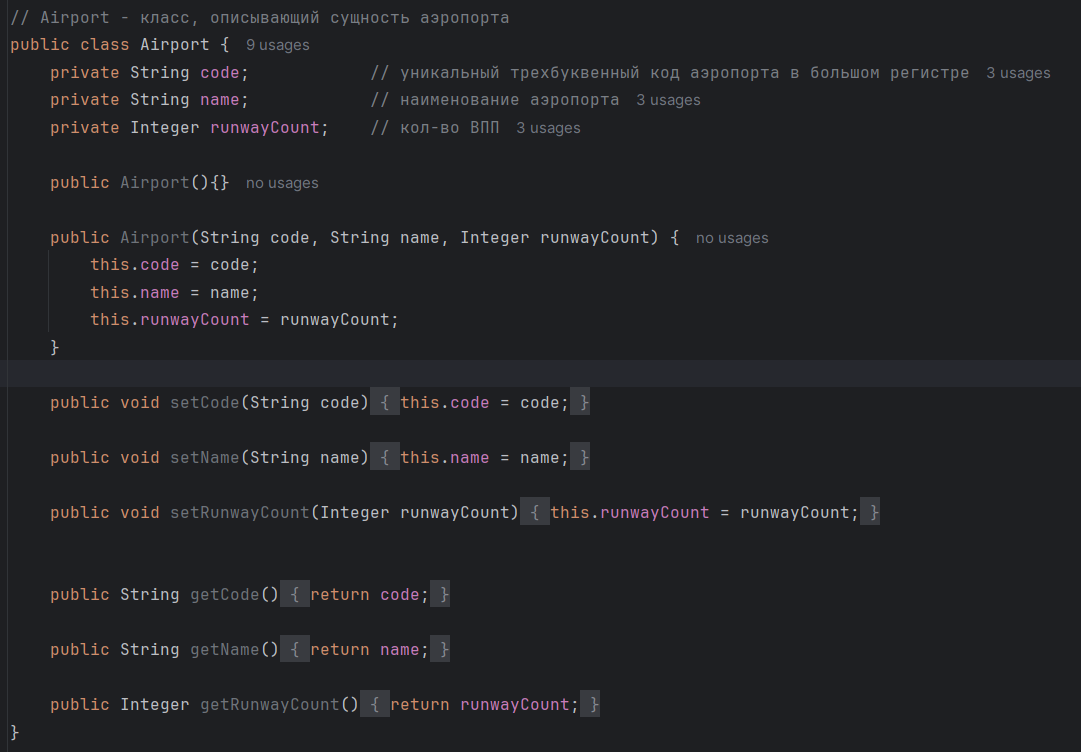
Использование RECORD

- Если вы хотите передавать неизменяемые данные между разными уровнями вашего приложения

- простые контейнеры данных с неизменяемыми свойствами и автоматически сгенерированными методами(GET, SET)

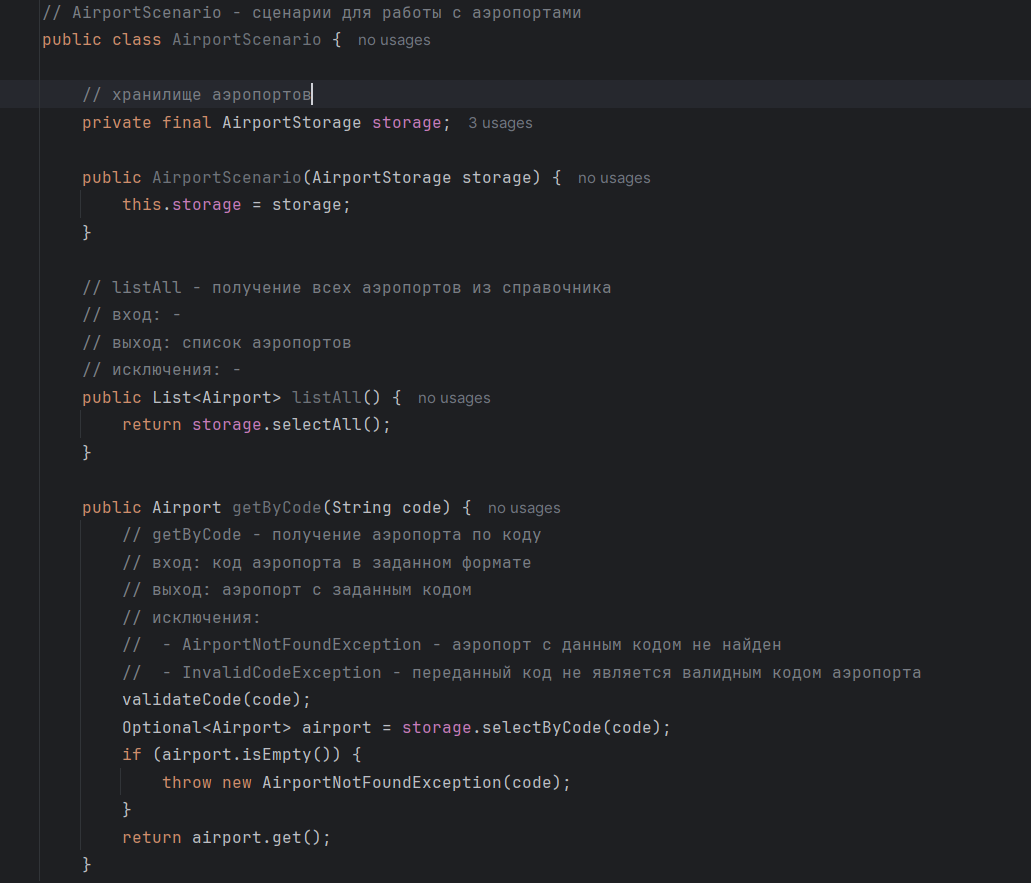
1. Model

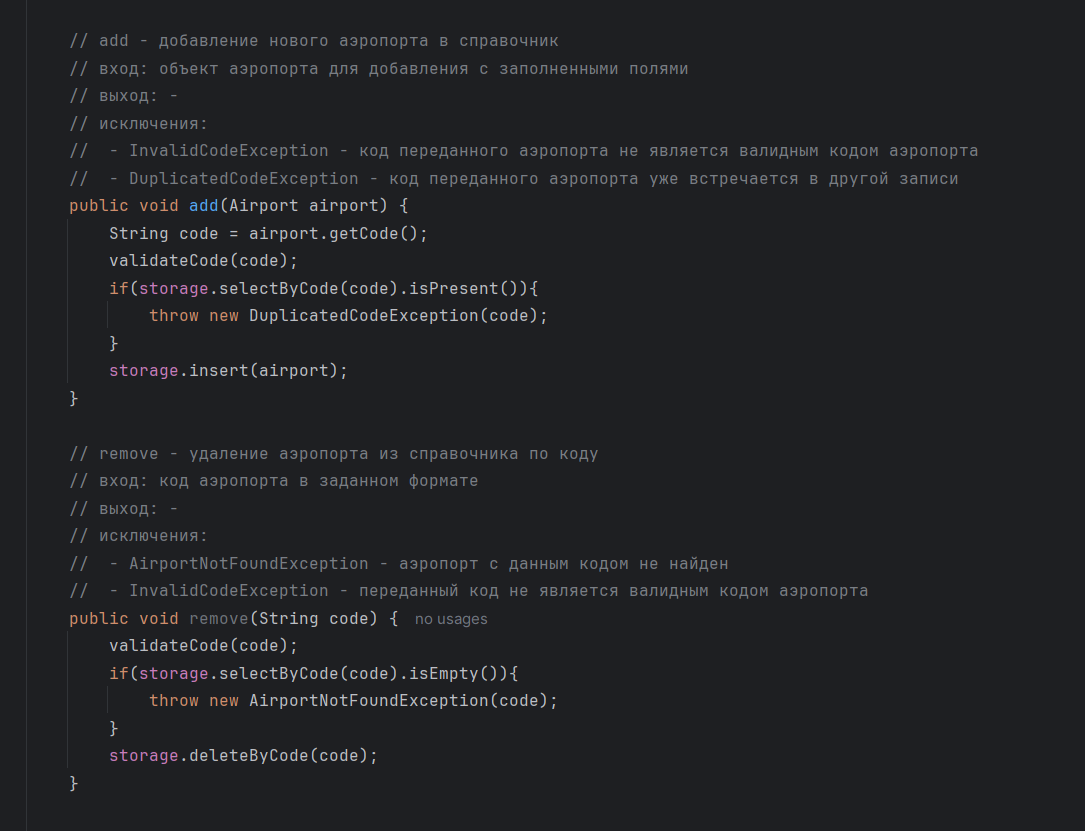
Airport - класс, описывающий сущность аэропорта

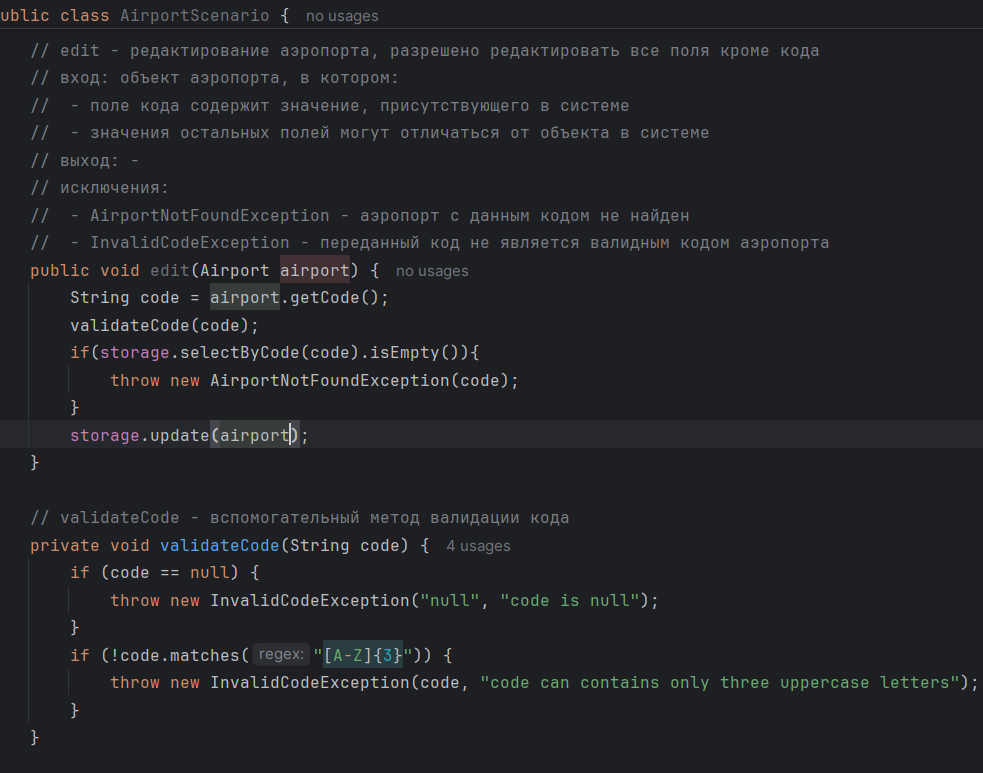


AirportScenario - сценарии для работы с аэропортами

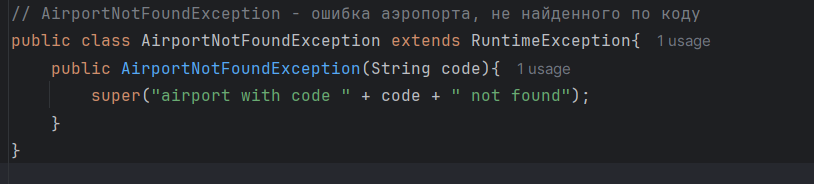
* listAll - получение всех аэропортов из справочника
* getByCode - получение аэропорта по коду
* add - добавление нового аэропорта в справочник
* remove - удаление аэропорта из справочника по коду
* edit - редактирование аэропорта, разрешено редактировать все поля кроме кода
* validateCode - вспомогательный метод валидации кода



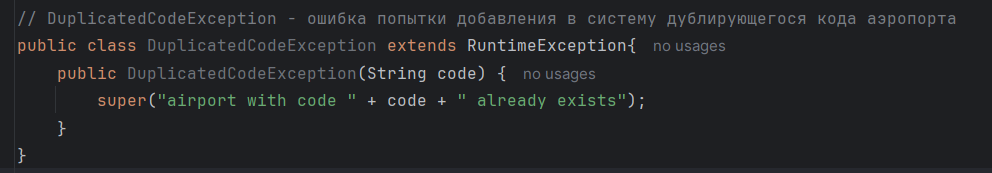




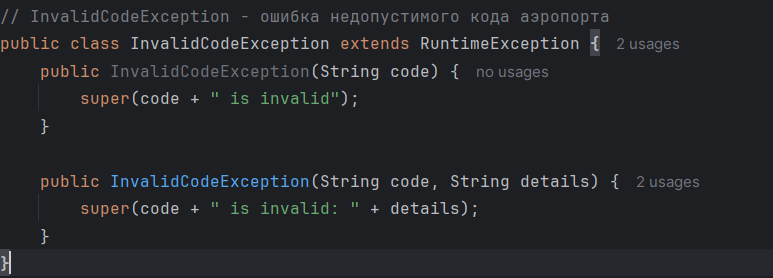
AirportNotFoundException - ошибка аэропорта, не найденного по коду



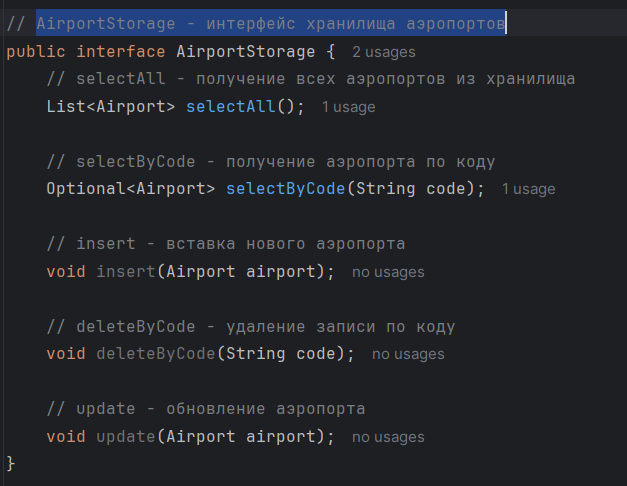
DuplicatedCodeException - ошибка попытки добавления в систему дублирующегося кода аэропорта



InvalidCodeException - ошибка недопустимого кода аэропорта



AirportStorage - интерфейс хранилища аэропортов



Контракт реализации слоя Storage и model необходимых методов, используя принцип инъекции зависимости

Далее для реализации AirportController (контроллер для работы с аэропортами) не изменяя model Airport

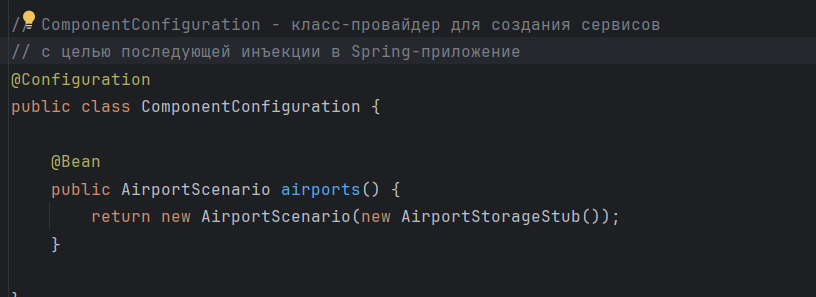
Добавим Конфигурацию

1. Configuration

ComponentConfiguration - класс-провайдер для создания сервисов с целью последующей инъекции в Spring-приложение

Главная цель создания Storage для заглушки(Stub – глупая не конфигурируемая заглушка) и проверки бизнес логики.

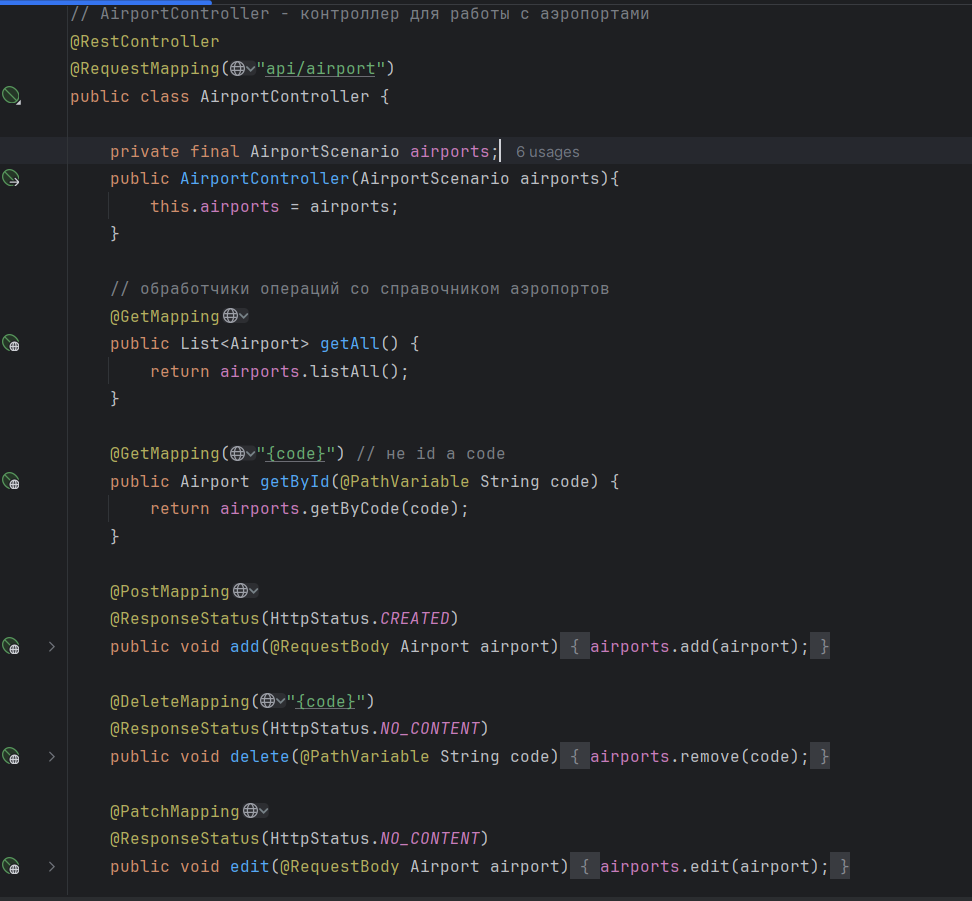
AirportStorageStub

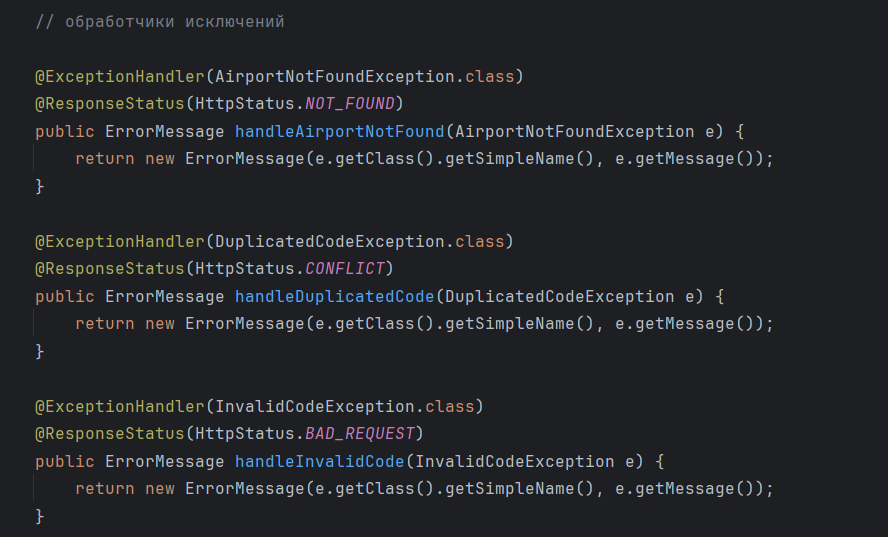


1. Для пакета API -> Controller

Добавим сам AirportController контроллер

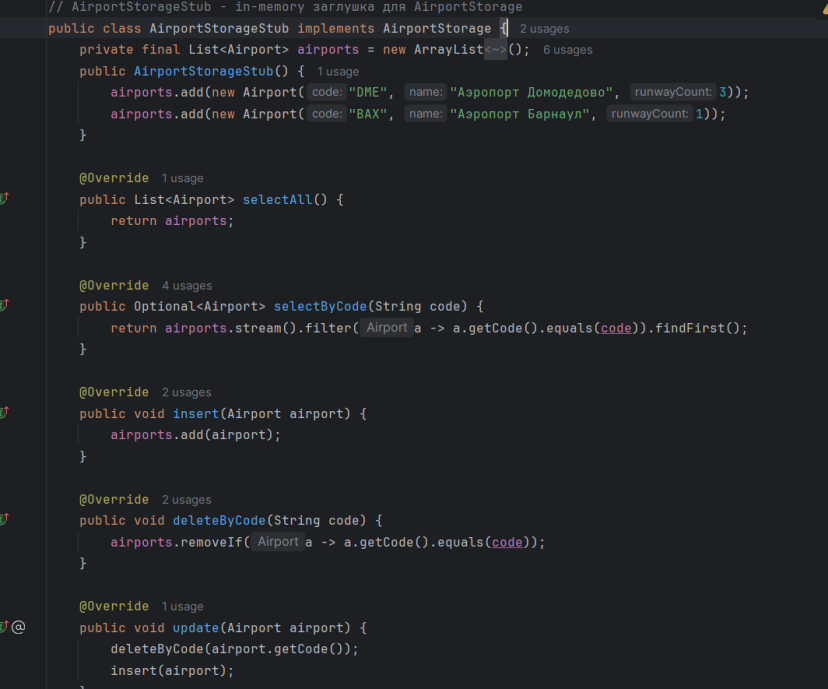
AirportController - контроллер для работы с аэропортами





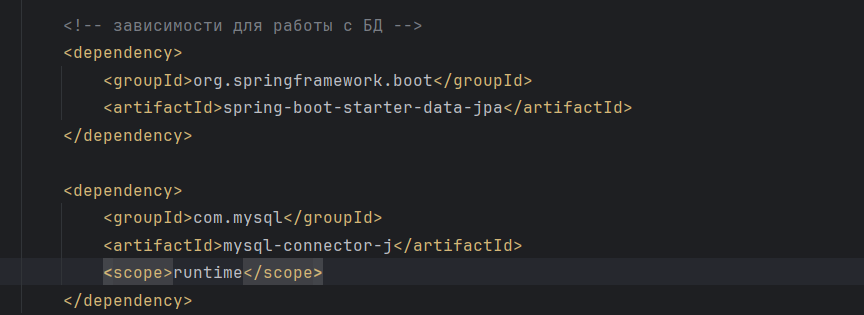
1. Stub

AirportStorageStub - in-memory заглушка для AirportStorage

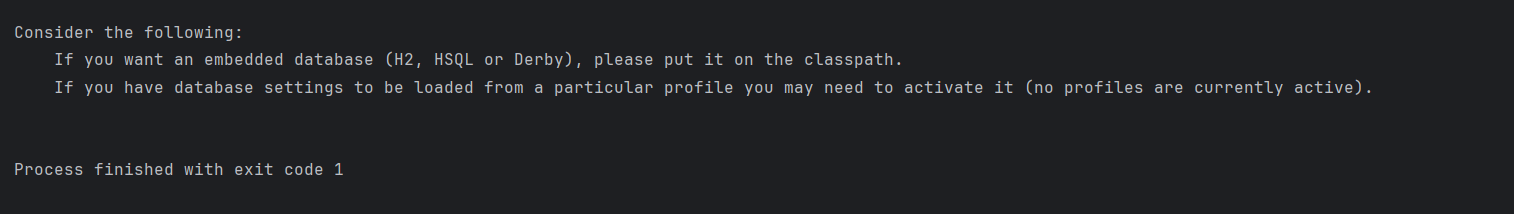


1. Добавим зависимости для работы с базой (POM.XML)

<!-- зависимости для работы с БД -->  
<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
</dependency>  
  
<dependency>  
 <groupId>com.mysql</groupId>  
 <artifactId>mysql-connector-j</artifactId>  
 <scope>runtime</scope>  
</dependency>



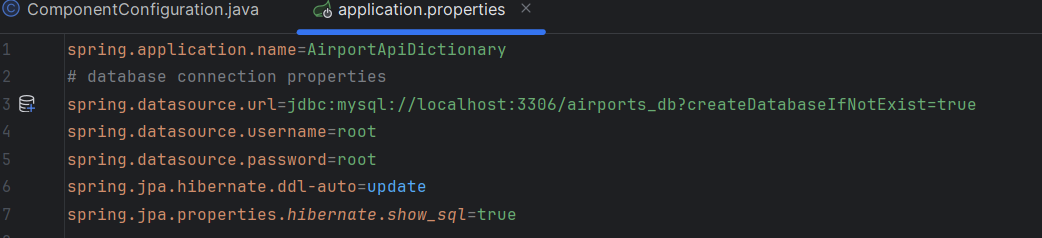
Исправим конфигурации для подключения БД

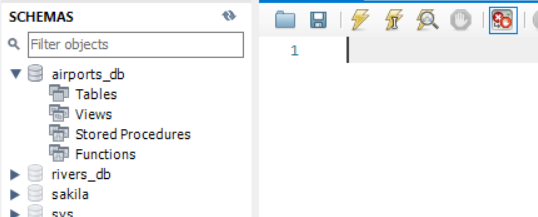


1. application.properties

Добавим настройки БД

# database connection properties  
spring.datasource.url=jdbc:mysql://localhost:3306/airports\_db?createDatabaseIfNotExist=true  
spring.datasource.username=root  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.properties.hibernate.show\_sql=true





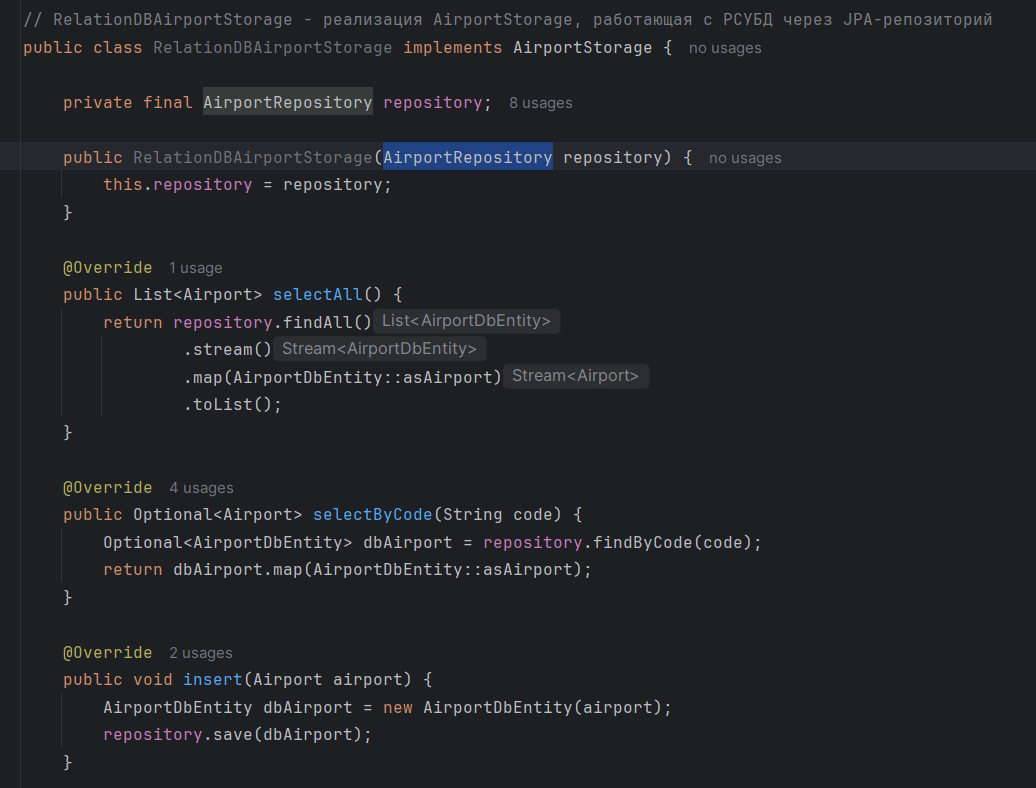
1. Storage

RelationDBAirportStorage - реализация AirportStorage, работающая с РСУБД через JPA-репозиторий, который имплементируется от AirportRepository

В котором есть конструктор и используя метод приведения к Airport -> asAirport()

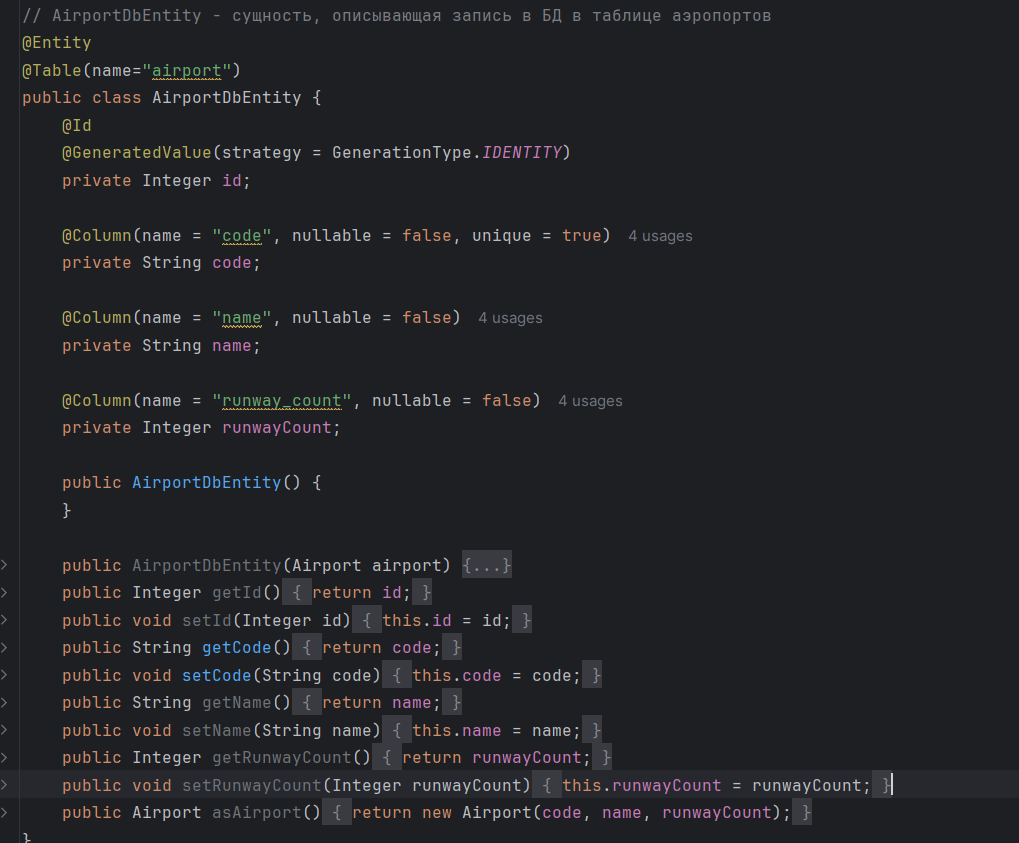


И не взирая на производительность(List)

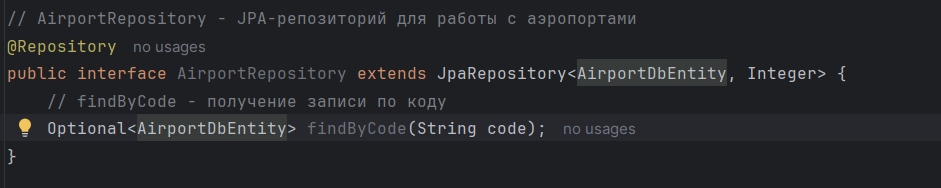




AirportDbEntity - сущность, описывающая запись в БД в таблице аэропортов, не изменяя MODEL а только используя свое представление в БД.



AirportRepository - JPA-репозиторий для работы с аэропортами не модели



1. ComponentConfiguration - класс-провайдер, добавим наш репозиторий, делая инъекцию используя Spring, он сам должен понять меняя Stub на БД

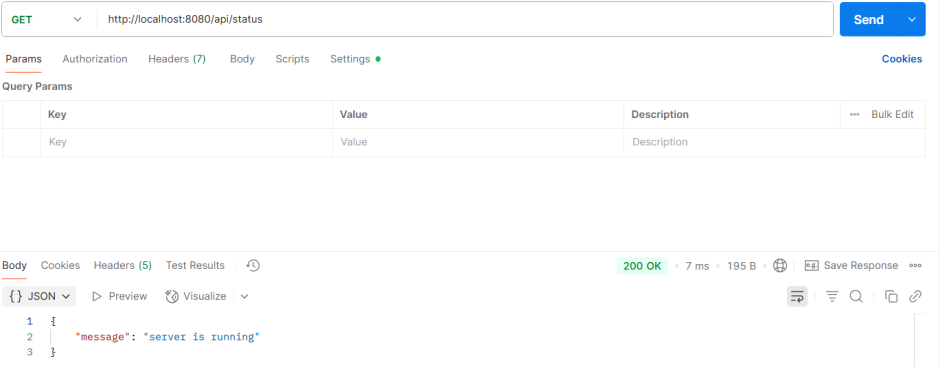
AirportRepository->RelationDBAirportStorage->AirportScenario

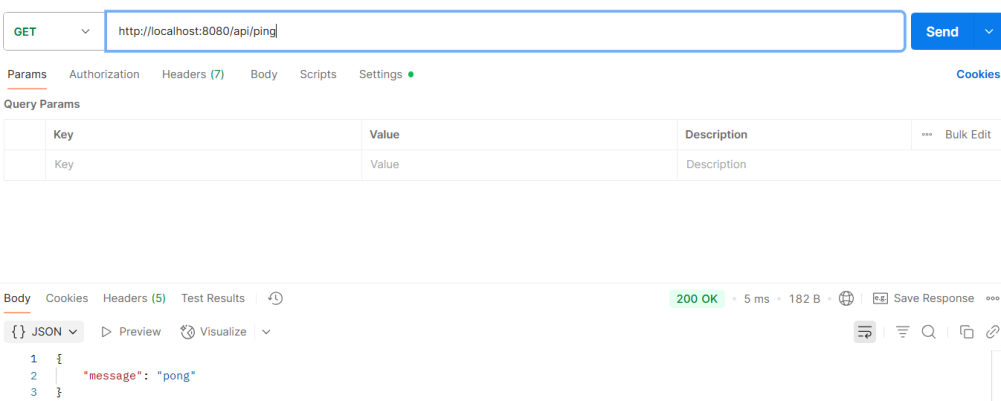
И далее в контроллеры



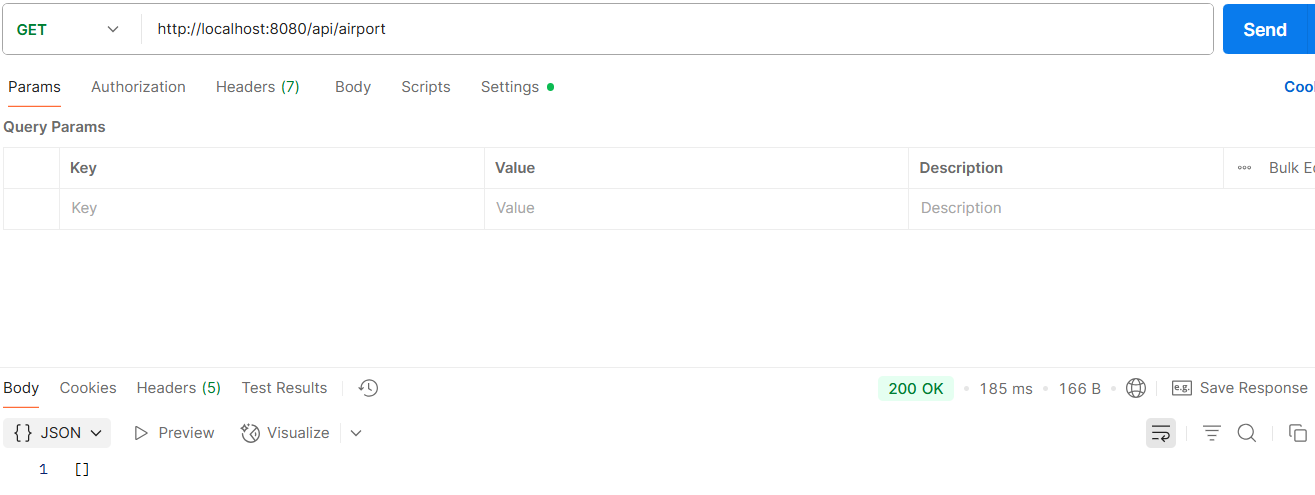
Ответы:

* Статус сервера

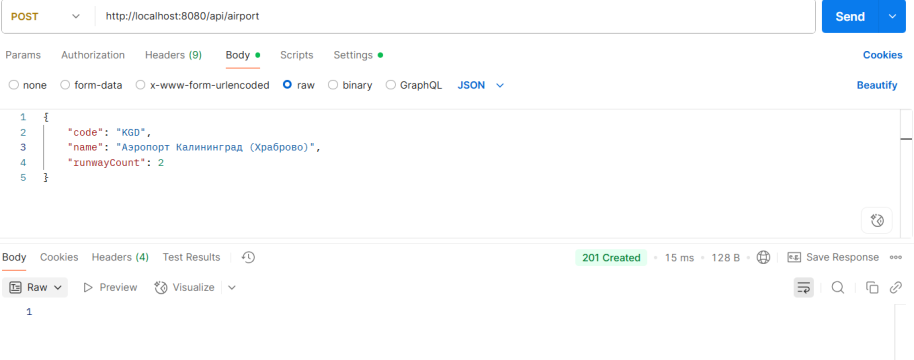


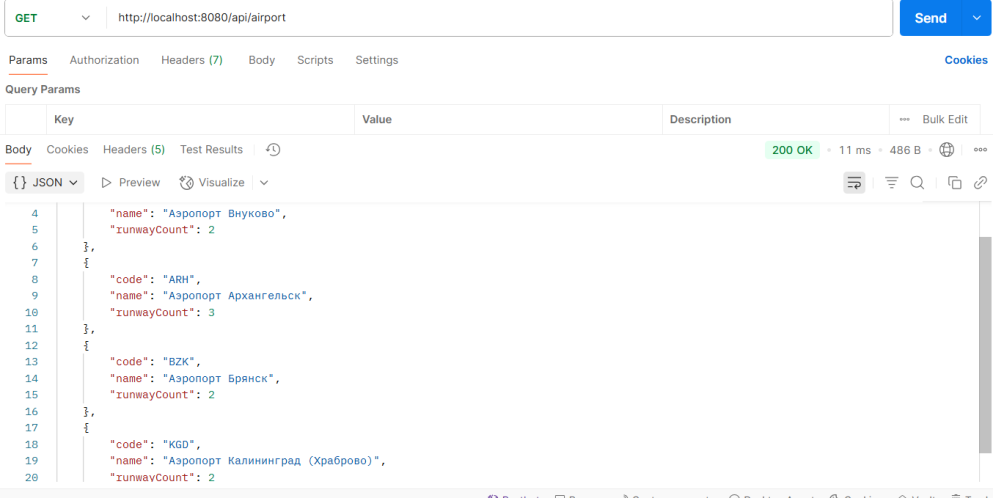


* Пустой список

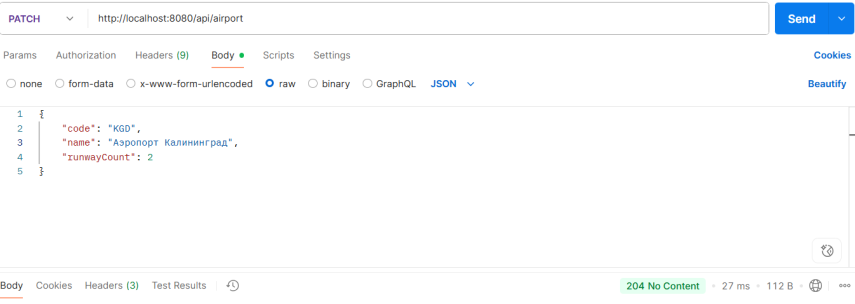


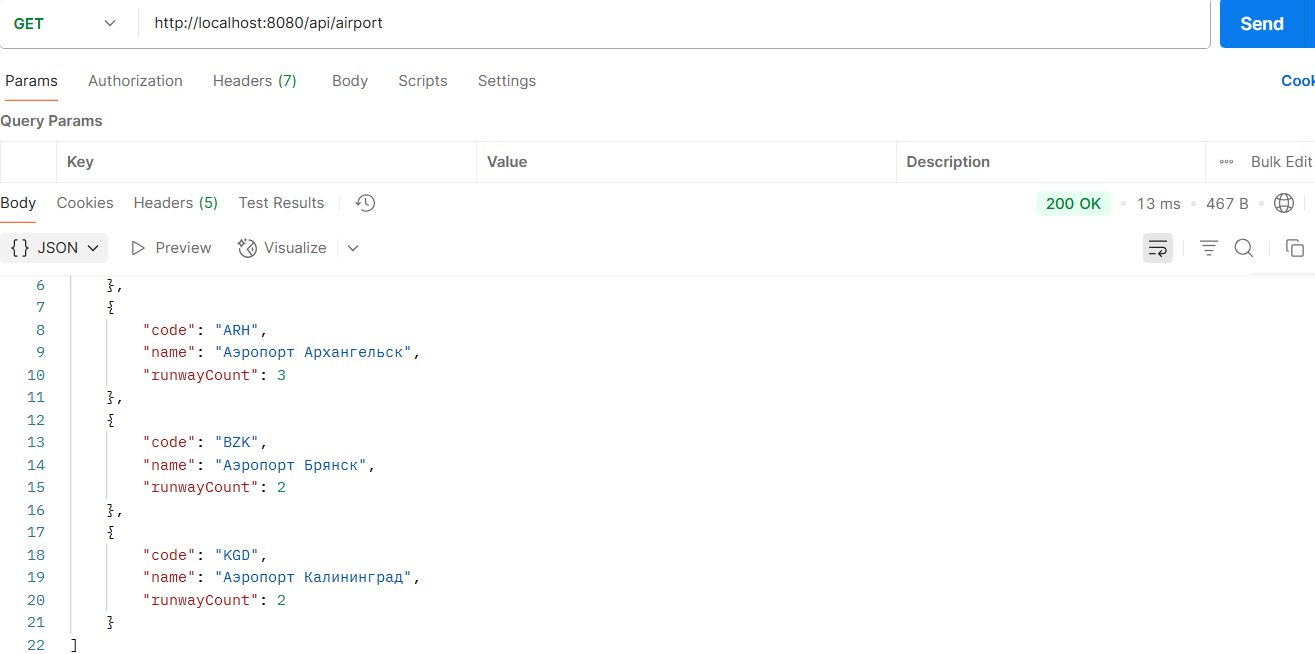
* Добавление аэропортов POST

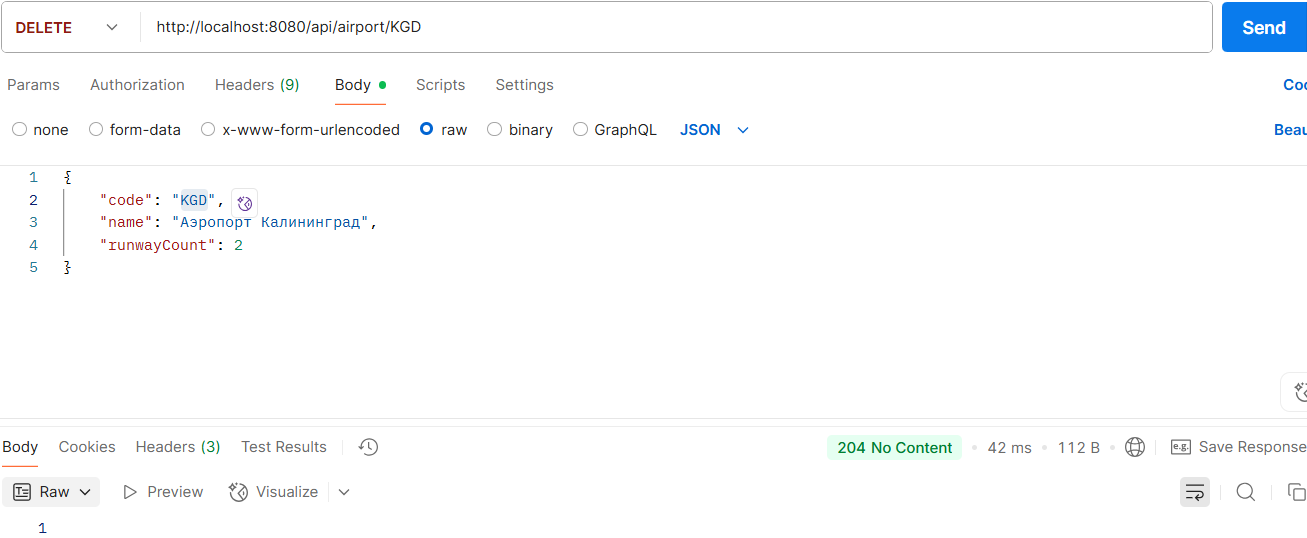
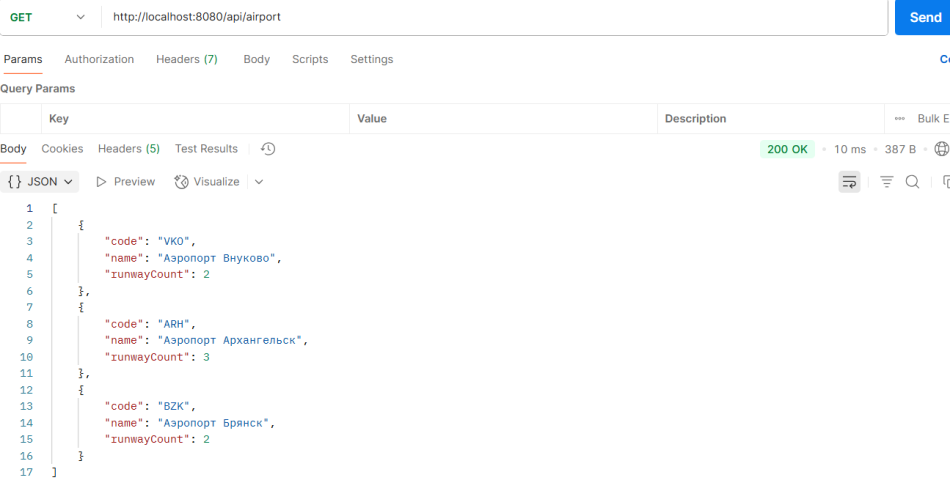
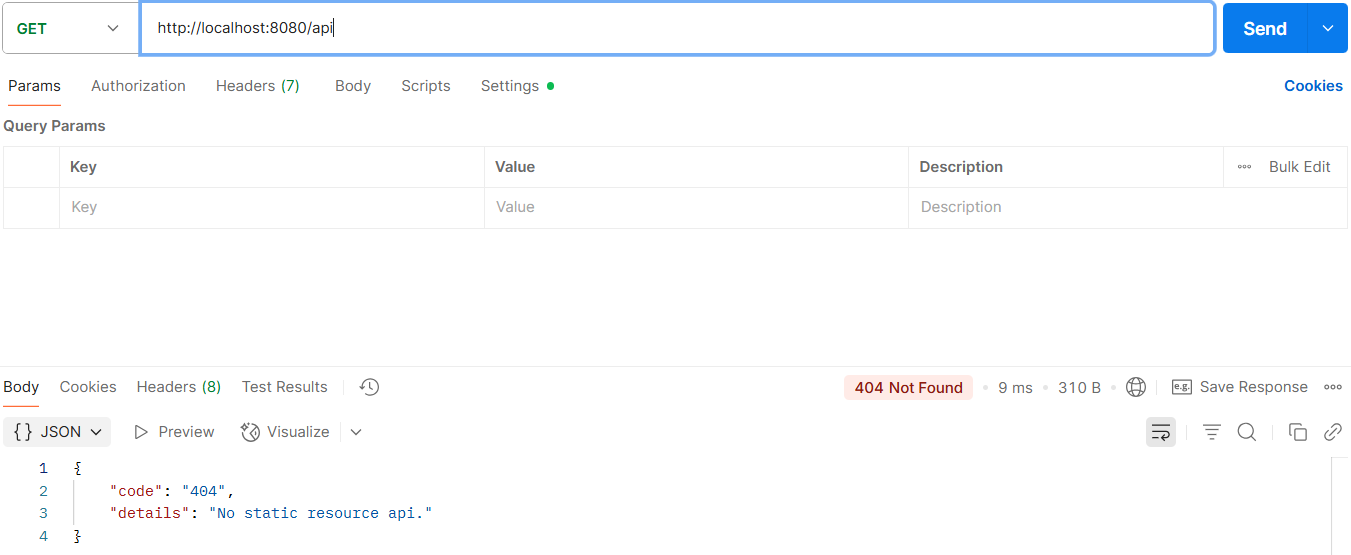
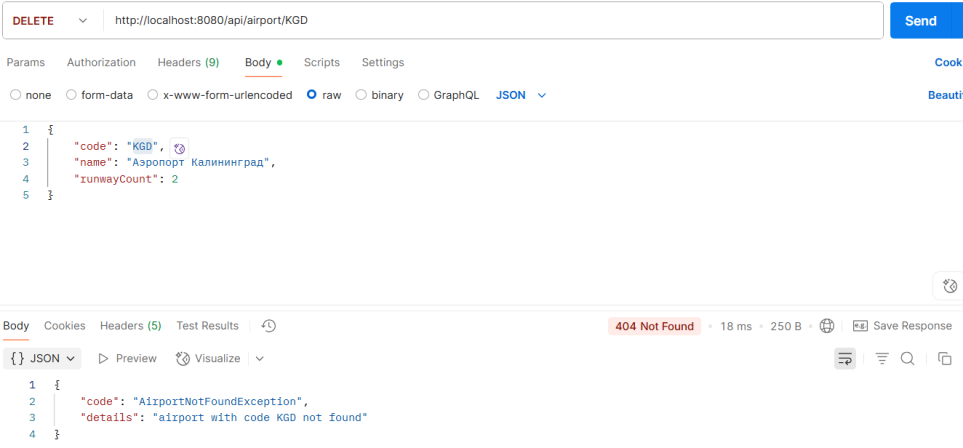
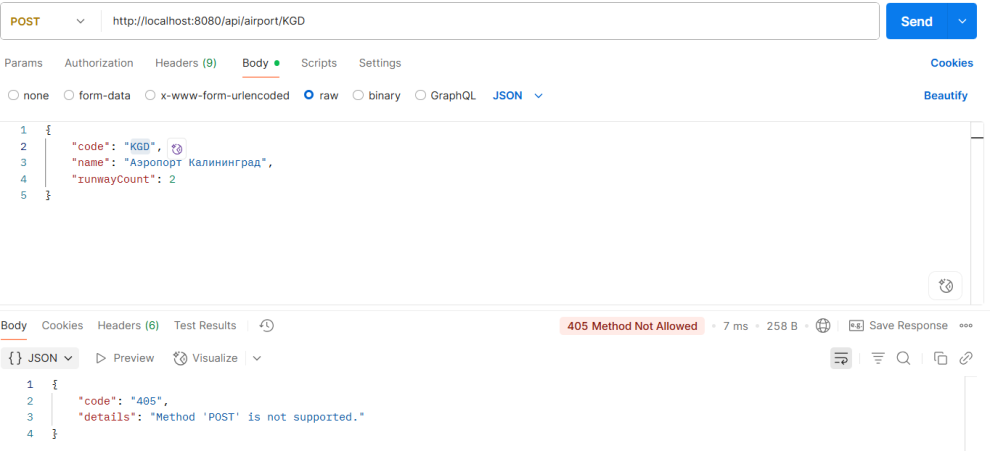




* Изменение Patch





* Удаление Delete
* 
* 
* Errors
* 
* 
* 
* 