

**FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA (FIAP)**

Henrique Baptista: RM 97796

Pedro Carvalho Pacheco: RM 98043

**CP6 - Domain Driven Design - 1TDSB**  
**Consulta e Criação de API**

## 1.0 Projeto de Consulta de API

A Dupla, como parte da solução para o desafio proposto neste checkpoint, desenvolveu uma aplicação Spring com Java, onde o usuário da mesma pode consultar as informações completa de um endereço, ao informar um CEP na aplicação, este será enviado para a API [viacep](#), que devolverá um JSON com as informações deste endereço.

### 1.1 Organização do Projeto

Como dito anteriormente, este é um projeto Spring, que contém apenas uma dependência, a **Spring Web**, que será responsável por fazer as requisições à API da viacep serem possíveis.

Além disso, o projeto tem duas classes principais, a primeira é uma classe **RestController**, chamada **ConsultaCepController**, que tem o método **consultaCep**, este é responsável por verificar o CEP digitado na URL de requisição, passar este CEP para a API viacep, e depois devolver o que foi retornado. O retorno, é justamente um objeto da segunda classe principal do nosso projeto, o **CepResultadoDTO**, que tem como atributo os campos retornados na requisição.

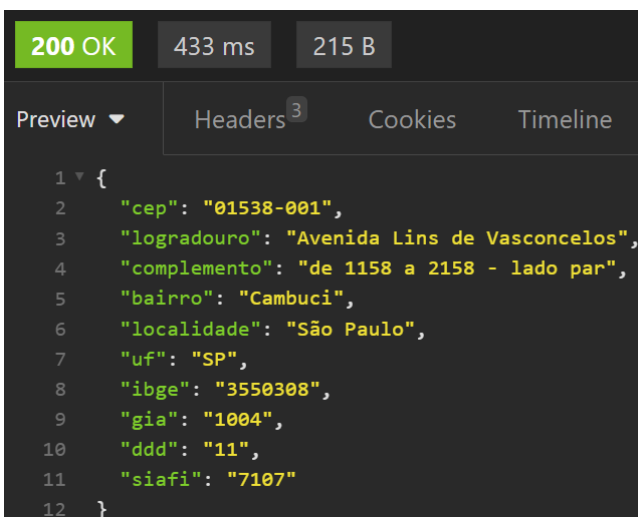
### 1.2 Como utilizar o Projeto

Antes de tudo, abra o projeto e inicialize-o. Então, em um programa de requisições http (Postman ou Insomnia, por exemplo), crie um novo campo de requisição GET e escreva a seguinte url: **localhost:8080/consulta-cep/**

- Depois disso, apenas adicione o CEP desejado ao final da URL, e faça a requisição.



GET ▼ localhost:8080/consulta-cep/01538-001



200 OK 433 ms 215 B

Preview ▼ Headers 3 Cookies Timeline

```
1 {  
2   "cep": "01538-001",  
3   "logradouro": "Avenida Lins de Vasconcelos",  
4   "complemento": "de 1158 a 2158 - lado par",  
5   "bairro": "Cambuci",  
6   "localidade": "São Paulo",  
7   "uf": "SP",  
8   "ibge": "3550308",  
9   "gia": "1004",  
10  "ddd": "11",  
11  "siafi": "7107"  
12 }
```

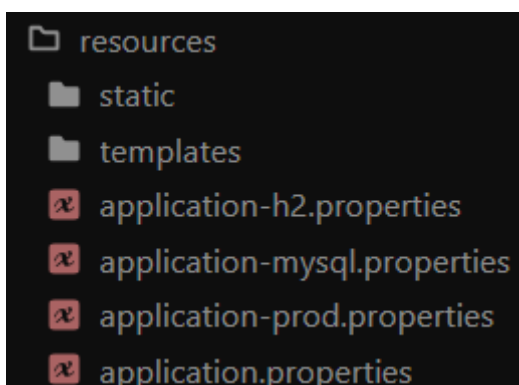
## 2.0 Projeto Criação de API

A Dupla, como parte da solução para o desafio proposto neste checkpoint, desenvolveu uma aplicação Spring com Java, onde criamos uma API, com persistência e consulta em Banco de Dados. A API, tem o tema de Jornalistas e Emissoras, onde um Jornalista tem suas informações, assim como a Emissora, e o primeiro pode fazer parte de uma Emissora. Trabalhando assim, com relacionamentos entre entidades.

### 2.1 Organização do Projeto

Como dito anteriormente, este é um projeto Spring, este, contém 5 (cinco) dependências, a **Spring Web**, que será responsável por fazer as requisições na nossa api, a **Spring Data JPA**, responsável por armazenar e consultar os dados das requisições e respostas da API no banco de dados utilizado, a **Spring Boot DevTools**, que faz com que o projeto seja reinicializado toda vez que uma mudança é feita, a **H2 Database**, que nos dá acesso a um banco de dados de testes em memória, que traz dinamismo ao nosso desenvolvimento e por último a **MySQL Connector**, que nos permite utilizar o banco de dados MySQL para armazenar e consultar os dados enquanto usamos o projeto.

O projeto tem uma pasta **resources**, responsável pelas configurações do projeto e dos bancos de dados utilizados. O projeto tem 4 arquivos .properties, sendo 3 com configurações de banco de dados, um para testes em memória (application-h2), um para conexão com um banco SQL mas localmente (application-mysql) e por último um para produção, que será utilizado para configurar um banco de dados na nuvem (application-prod). O quarto application.properties, tem as configurações gerais do projeto e define qual dos bancos de dados será utilizado.



Dando sequência, o projeto tem 3 pacotes principais. Sendo o primeiro deles o pacote **model**, com as entidades que serão usadas no projeto. Neste modelo, estão as classes padrões do projeto, com o seu mapeamento JPA e com configurações de ID. Em seguida, temos o pacote **repository**, que é responsável por armazenar as interfaces referentes às nossas entidades, estas interfaces, que estendem da `JpaRepository<>`. Cada interface tem um mapeamento de Entidade e ID (exemplo, a entidade `Jornalista` tem o campo ID como um `Long`, logo, o seu `Repository` estenderá de um `JpaRepository<Jornalista, Long>`). E por último, temos o pacote **Controller**, neste pacote, estão os `RestsControllers` da nossa aplicação, que são as classes que tem os métodos referentes às requisições http. Com essas classes, podemos salvar, deletar e listar as nossas entidades.

## 2.2 Endpoints da API

Métodos HTTP	URL da Requisição	Descrição da Requisição
GET	localhost:8080/emissoras	Retorna todas as emissoras salvas
GET	localhost:8080/emissoras/{id}	Retorna a emissora com o id informado na url
POST	localhost:8080/emissoras	Salva uma nova emissora
DELETE	localhost:8080/emissoras/{id}	Deleta a emissora com o id informado na url
GET	localhost:8080/jornalistas	Retorna todos os jornalistas salvos e as informações da sua emissora
GET	localhost:8080/jornalistas/{id}	Retorna o jornalista (e as informações de sua emissora) com o id informado na url
POST	localhost:8080/jornalistas	Salva um novo jornalista
DELETE	localhost:8080/jornalistas/{id}	Deleta a emissora com o id informado na url

## 2.3 Como utilizar o Projeto

Antes de tudo, abra o projeto e inicialize-o. Então, em um programa de requisições http (Postman ou Insomnia, por exemplo), crie um novo campo de

requisição POST e escreva a seguinte url: **localhost:8080/emissoras**

POST ▼ http://localhost:8080/emissoras

Então, defina as informações da emissora que você deseja criar, respeitando os campos **nome**, **estado** & **anoFundacao**. Depois disso, envie a requisição.

```
{
  "nome": "Globo",
  "estado": "Rio de Janeiro",
  "anoFundacao": 1965
}
```

200 OK 752 ms 68 B

Preview ▼ Headers 3 Cookies

```
1 {
2   "id": 1,
3   "nome": "Globo",
4   "estado": "Rio de Janeiro",
5   "anoFundacao": 1965
6 }
```

Depois disso, crie um novo campo no seu programa de requisições http, e escreva a seguinte url: **localhost:8080/jornalistas**

POST ▼ http://localhost:8080/jornalistas

Então, depois disso, defina as informações do jornalista, respeitando os campos **nome**, **cpf**, **rg** e **emissora.id**

```
{
  "name": "Pedro",
  "cpf": "113.245.245-56",
  "rg": "12345678",
  "emissora": {
    "id": 1
  }
}
```

```
200 OK 18.1 ms 142 B
Preview Headers Cookies
1 {
2   "id": 2,
3   "name": "Pedro",
4   "cpf": "113.245.245-56",
5   "rg": "12345678",
6   "emissora": {
7     "id": 1,
8     "nome": "Globo",
9     "estado": "Rio de Janeiro",
10    "anoFundacao": 1965
11  }
12 }
```

Com isso entendido, você já está pronto para continuar utilizando a aplicação sem um guia prático. Se alguma dúvida surgir, lembre-se de consultar a nossa tabela de endpoints.