

# Informações

Membros:

Atílio Gabriello - 11921BCC036

Flávio Vezono Filho - 11921BCC014

Guilherme Nascimento Leite - 11921BCC018

Carlos Ernani Alves de Paula Junior - 11921BCC005

OBS: Todo o back-end da aplicação foi feito utilizando spring e os dados estão sendo salvos em um banco de dados **sql (postgres)**.

Antes de iniciar a aplicação no **Spring Tool Suite**, é necessário criar uma database chamado **springPOO** no postgres. Todas as tabelas serão criadas usando **migrations** no próprio spring, que irá utilizar o fly para fazer esta tarefa.

O código back-end é feito em java utilizando spring, já o front foi feito apenas para consumir a api gerada. Não é necessário utilizá-lo para ter acesso à api.

Caso deseje rodar o front-end, é preciso ter o Node instalado. Com isso, basta rodar um yarn ou npm install dentro da pasta raiz do projeto. Depois rode yarn ou npm start.

- A api que o programa irá fornecer se encontra em ***http://localhost:8080/api/***
- Existem duas rotas básicas: ***/viagens*** e a ***/viajante*** com elas é possível inserir, deletar, e dar update nos campos da tabela.
- Para testar a aplicação, existem duas possibilidades: utilizando o próprio front-end que foi feito em ***React(typescript)*** ou o ***Insomnia/Postman***.
- Na construção da aplicação foram aplicados os conceitos de: Classes, Interfaces, Herança, List, ArrayList e Métodos.

<https://spring.io/tools>

<https://insomnia.rest/download>

O código completo está em <https://github.com/flaviozno/TrabalhoPOO>

Toda a conexão com o banco de dados é definida em **application.properties**.

Os scripts sql estão em **src/main/resources/db/migration**.

Dependências utilizadas:

- DevTools
- Flyway
- Postgresql
- Spring Boot Starter

### **Anotações utilizadas:**

**@Id:** A anotação específica a chave primária de uma entidade e prevê a especificação das estratégias de geração para os valores das chaves primárias.

**@GeneratedValue:** é utilizada para informar que a geração do valor do identificador único da entidade será gerenciada pelo provedor de persistência.

**@NotNull:** é utilizada para dizer que o conteúdo de uma string não pode ser null.

**@Valid:** é utilizada para verificar se os dados que estão sendo inseridos estão de acordo com os que a entidade recebe.

**@Entity:** é utilizada para informar que uma classe também é uma entidade. A partir disso, a JPA estabelecerá a ligação entre a entidade e uma tabela de mesmo nome no banco de dados, onde os dados de objetos desse tipo poderão ser persistidos.

**@Repository:** é uma anotação que indica que a classe é um repositório. Um repositório é um mecanismo para encapsular o comportamento de armazenamento, recuperação e pesquisa que simula uma coleção de objetos.

**@ControllerAdvice:** é uma anotação que permite lidar com exceções em toda a aplicação em um componente de manuseio global.

**@RestController:** é uma anotação para criar controladores restful. Converte a resposta para JSON ou XML.

**@RequestMapping:** é uma anotação para mapear a url gerada.

**@PathVariable:** pode ser usada para lidar com variáveis de modelo no mapeamento da solicitação e defini-las como parâmetros de método.

**@RequestBody:** é uma anotação mapeia o corpo httpRequest para um objeto de transferência ou domínio, permitindo a deserialização automática.

**@Transactional(readOnly = true):** essa anotação irá garantir que nada poderá ser alterado em uma determinada rota.

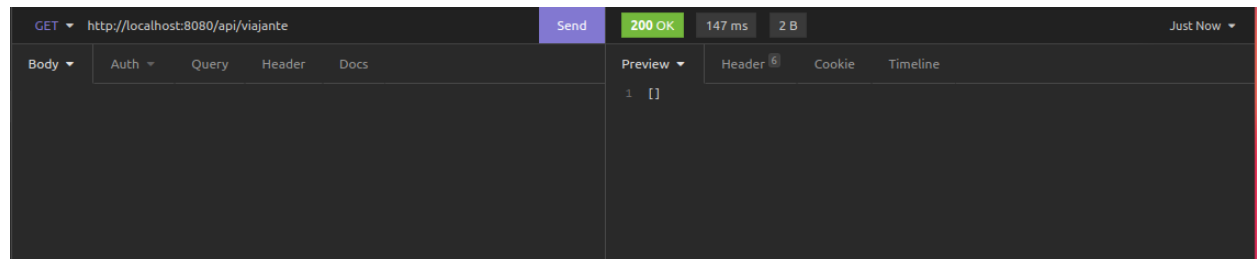
**@CrossOrigin:** é uma anotação que libera a interação entre o navegador e o servidor para lidar com segurança com as solicitações HTTP de origem cruzada. No caso está definida como **localhost:3000** pois é onde estava rodando a parte visual da aplicação em React.

**@GetMapping:** é uma anotação para rotas do tipo GET.

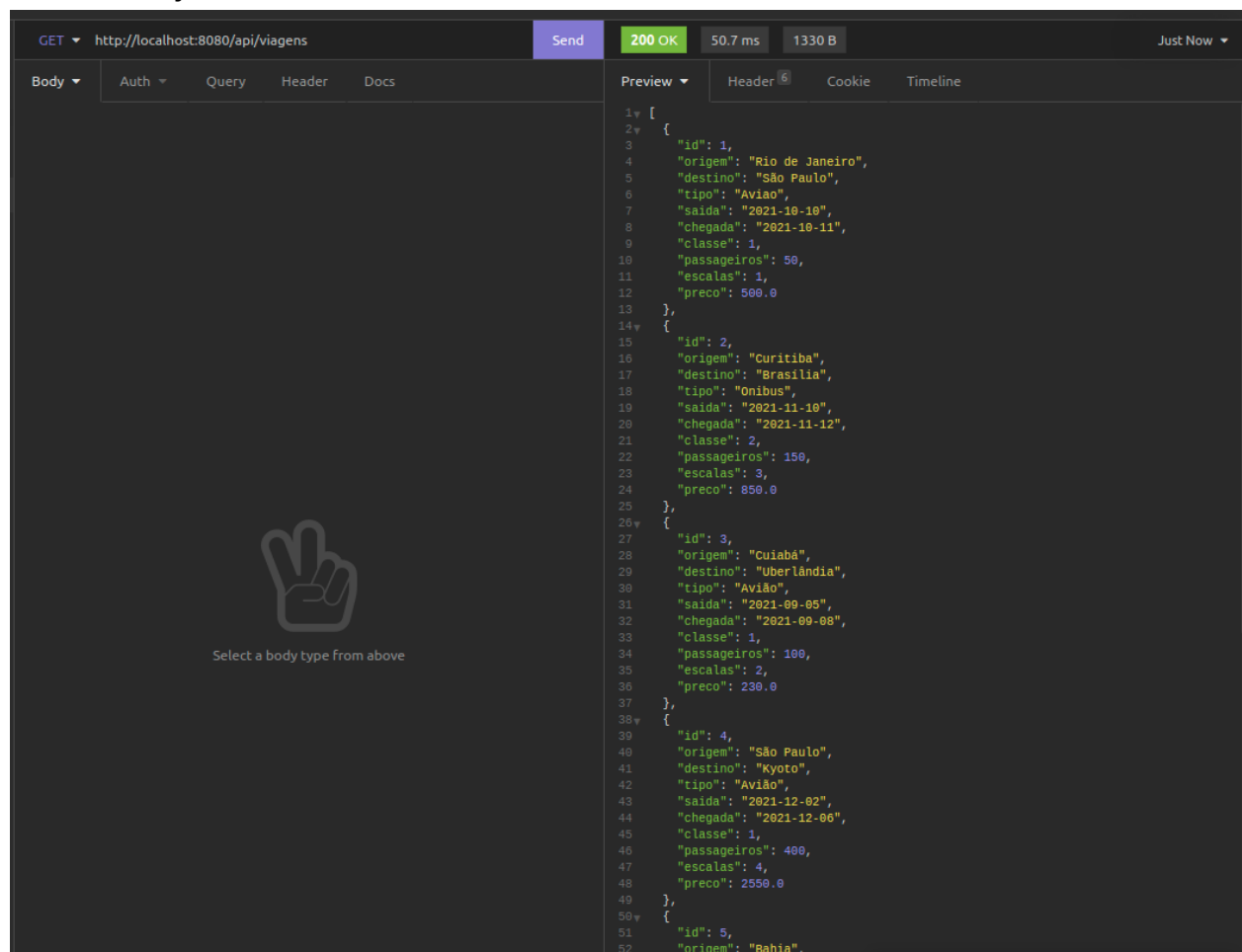
**@PutMapping:** é uma anotação para rotas do PUT/UPDATE.

**@PostMapping:** é uma anotação para rotas do PUT/UPDATE.

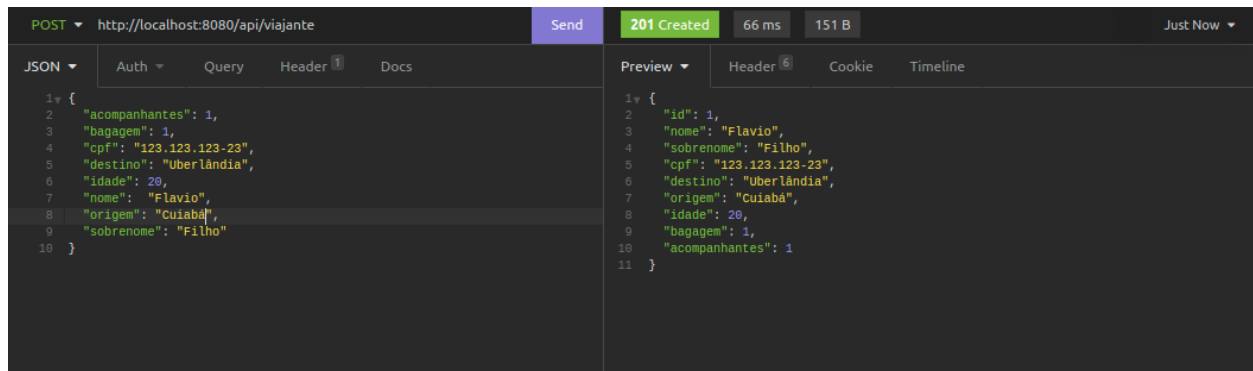
**@ResponseStatus(HttpStatus.CREATED):** é uma anotação que vai retornar o status HTTP gerado após uma interação.



Quando iniciamos a aplicação e testamos a rota <http://localhost:8080/api/viajante> a aplicação irá retornar `[]` pois ainda não existem viajantes cadastrados.



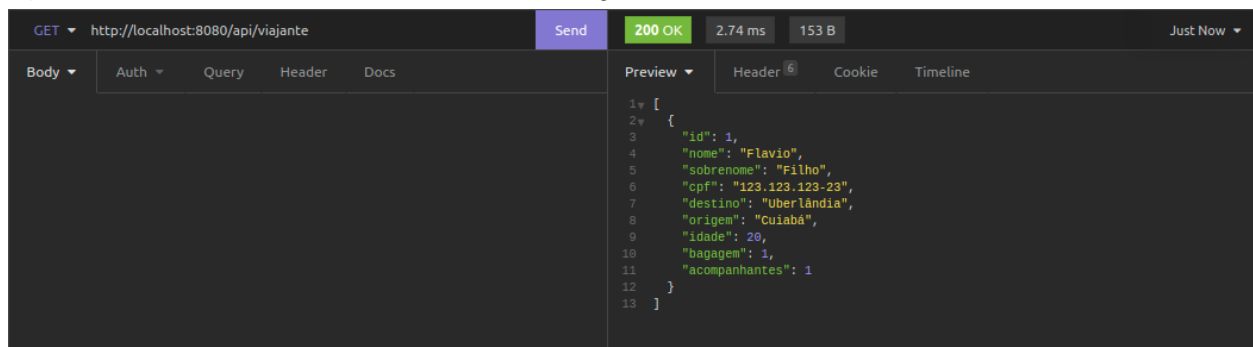
Contudo, a rota <http://localhost:8080/api/viagens> irá nos mostrar todas as viagens listadas no postgres.



```
POST http://localhost:8080/api/viajante 201 Created 66 ms 151 B Just Now
JSON Auth Query Header Docs
1 {
2   "acompanhantes": 1,
3   "bagagem": 1,
4   "cpf": "123.123.123-23",
5   "destino": "Uberlândia",
6   "idade": 20,
7   "nome": "Flavio",
8   "origem": "Cuiabá",
9   "sobrenome": "Filho"
10 }

Preview Header Cookie Timeline
1 {
2   "id": 1,
3   "nome": "Flavio",
4   "sobrenome": "Filho",
5   "cpf": "123.123.123-23",
6   "destino": "Uberlândia",
7   "origem": "Cuiabá",
8   "idade": 20,
9   "bagagem": 1,
10  "acompanhantes": 1
11 }
```

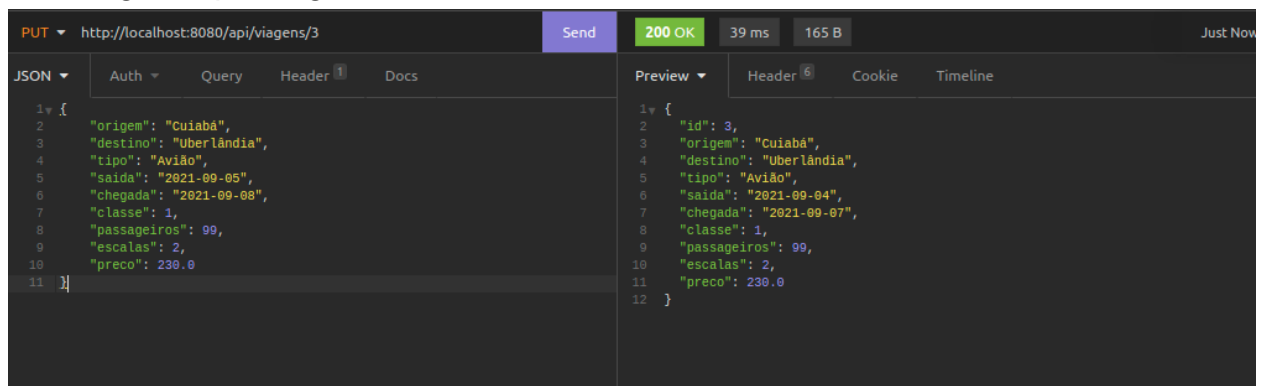
Após realizarmos o cadastro de um viajante, ele será listado no banco de dados.



```
GET http://localhost:8080/api/viajante 200 OK 2.74 ms 153 B Just Now
Body Auth Query Header Docs
1 [
2   {
3     "id": 1,
4     "nome": "Flavio",
5     "sobrenome": "Filho",
6     "cpf": "123.123.123-23",
7     "destino": "Uberlândia",
8     "origem": "Cuiabá",
9     "idade": 20,
10    "bagagem": 1,
11    "acompanhantes": 1
12  }
13 ]

Preview Header Cookie Timeline
```

Então, atualizaremos a quantidade de vamos da viagem selecionada, retirando uma vaga de passageiro.

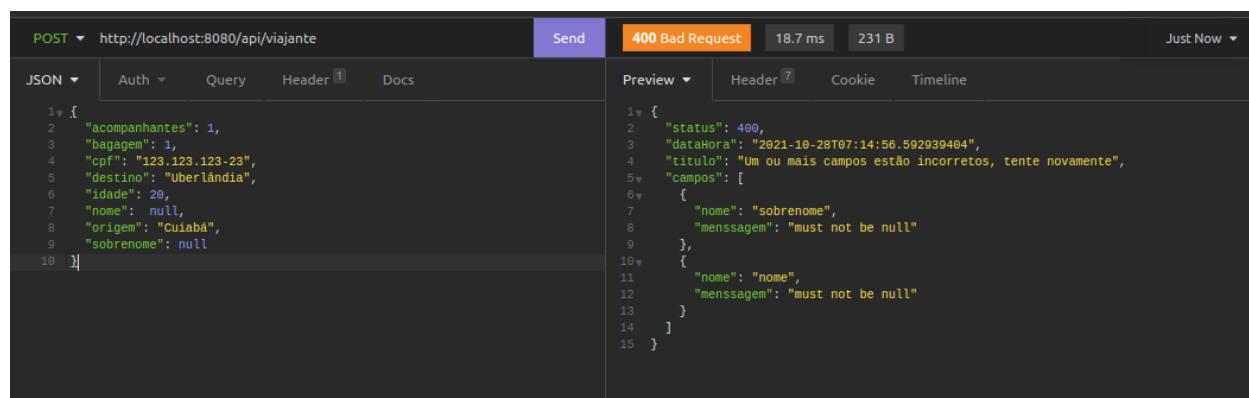


```
PUT http://localhost:8080/api/viagens/3 200 OK 39 ms 165 B Just Now
JSON Auth Query Header Docs
1 {
2   "origem": "Cuiabá",
3   "destino": "Uberlândia",
4   "tipo": "Avião",
5   "saida": "2021-09-05",
6   "chegada": "2021-09-08",
7   "classe": 1,
8   "passageiros": 99,
9   "escalas": 2,
10  "preco": 230.0
11 }

Preview Header Cookie Timeline
1 {
2   "id": 3,
3   "origem": "Cuiabá",
4   "destino": "Uberlândia",
5   "tipo": "Avião",
6   "saida": "2021-09-04",
7   "chegada": "2021-09-07",
8   "classe": 1,
9   "passageiros": 99,
10  "escalas": 2,
11  "preco": 230.0
12 }
```

Agora, a mesma viagem possui 99 vagas de passageiros, ao contrário das 100 ofertadas de início.

Caso o passageiro tente inserir os dados com algum campo em branco, ele receberá a seguinte mensagem:



Com relação ao postgres, as seguintes tabelas que foram criadas e seus respectivos dados:

Tables (3)

- flyway\_schema\_history
- viagens
- viajante

Data Output

	id [PK] bigint	origem character varying (255)	destino character varying (255)	tipo character varying (255)	saida timestamp without time zone	chegada timestamp without time zone
1	1	Rio de Janeiro	São Paulo	Aviao	2021-10-10 00:00:00	2021-10-11 00:00:00
2	2	Curitiba	Brasilia	Onibus	2021-11-10 00:00:00	2021-11-12 00:00:00
3	3	Cuiabá	Uberlândia	Avião	2021-09-04 00:00:00	2021-09-07 00:00:00
4	4	São Paulo	Kyoto	Avião	2021-12-02 00:00:00	2021-12-06 00:00:00
5	5	Bahia	Las Vegas	Avião	2021-09-03 00:00:00	2021-09-07 00:00:00
6	6	Chicago	Bruxelas	Avião	2021-08-14 00:00:00	2021-08-18 00:00:00
7	7	Monte Carmelo	Araguari	Onibus	2021-02-21 00:00:00	2021-02-21 00:00:00
8	8	Barcelona	Oslo	Avião	2021-06-12 00:00:00	2021-06-20 00:00:00

Data Output

	id [PK] bigint	nome character varying (255)	sobrenome character varying (255)	cpf character varying (255)	destino character varying (255)	origem character varying (255)	idade integer
1	1	Flavio	Filho	123.123.123-23	Uberlândia	Cuiabá	2

Data Output

	installed_rank [PK] integer	version character varying (50)	description character varying (200)	type character varying (20)	script character varying (1000)	checksum integer	installed_by character varying (100)	inst time
1	1	001	create-table-viagens	SQL	V001_create-table-viagens.sql	-648120858	postgres	202
2	2	002	insert-data-viagens	SQL	V002_insert-data-viagens.sql	496677373	postgres	202
3	3	003	create-table-viajante	SQL	V003_create-table-viajante.sql	1858357560	postgres	202