



Residência em Software

Residência em Tecnologia da Informação e Comunicação

SpringBoot: Persistência com JPA em Requisições

Professor:

Alvaro Degas Coelho



INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



De onde paramos

{JSON}

Operações CRUD

- Básico e fundamental
 - No fundo, tudo o que fazemos é CRUD
- ORM: Crud inclui dados complexos
 - Veremos
- Processo em duas vias
 - FrontEnd -> Aplicação -> Banco de Dados
 - Banco de Dados -> Aplicação -> Front End

Inserir um Usuario

- Um método “inserir” no Controller
 - Verbo POST (porque?)
 - Anotação @PostMapping
- Temos que fazer o objeto chegar até o método
 - A partir da camada de Interface
- Para nós: a partir da interface HTTP

Inserir um Usuario

- (Mesmo) problema de segurança: o objeto que vem como parâmetro não deve ser o mesmo da Entity
 - Não vem com Id (é gerado no SGBD)
 - Padrão DTO
- Neste caso o DTO tem uma nomenclatura padrão: UsuarioForm
- Como?
 - Criar a classe UserForm na package Form
 - Dentro da package Controller

Inserir um Usuario

- FORM (DTO)
 - Não tem Id (nome, email, senha)
 - Não esquecer dos Getters e Setters
- Importante: o parâmetro não é uma string
 - Objeto estruturado: dados no corpo da requisição.
 - Anotação `@RequestBody` identificando isso
- Importante (2): `UsuarioForm` não pode ser usado pelo repository
 - Precisamos converter para objeto do tipo `Usuario`

Novas anotações

- **@PostMapping**
 - Identifica que o método está associado ao verbo HTTP POST
- **@RequestBody**
 - Identifica que um parâmetro é obtido no corpo de uma requisição
 - JSON

Problemas

- Não temos um Front End
 - Que nos mande o JSON dentro de uma requisição HTTP POST
- Sistemas reais: não podemos esperar pelo FrontEnd
 - Android? Ios? Linux? PC Windows?
- Um Contrato
 - Requisição (com verbo) + Corpo
 - Corpo: JSON

POSTMAN

- Testar, documentar e colaborar em APIs (Application Programming Interfaces)
 - Fundamentalmente: Gera Requisições idênticas às que virão da camada de Interface
- Útil para
 - Teste de APIs: O Postman permite enviar solicitações HTTP (como GET, POST, PUT, DELETE)
 - Criação e execução de solicitações
 - Criar e enviar solicitações HTTP
 - Definir os cabeçalhos, parâmetros, corpo da solicitação
 - Obter resposta em diferentes formatos, como JSON, XML, HTML, entre outros.

Crie/Acesse sua conta

- Acesse www.postman.com

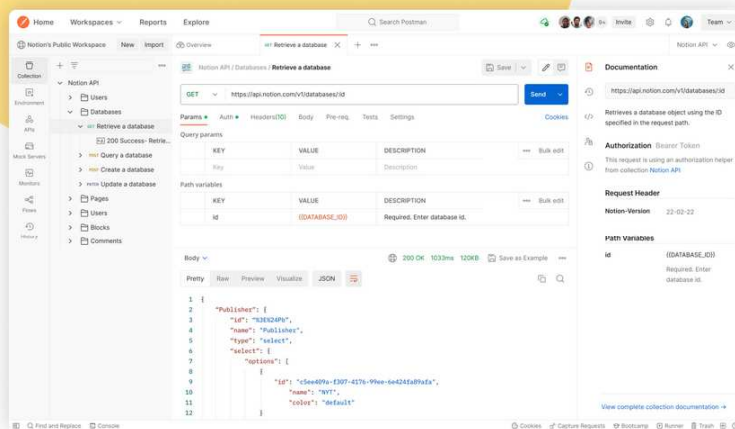


Debug _ APIs together

Over 30 million developers use Postman. Get started by signing up or downloading the desktop app.

[Sign Up for Free](#)

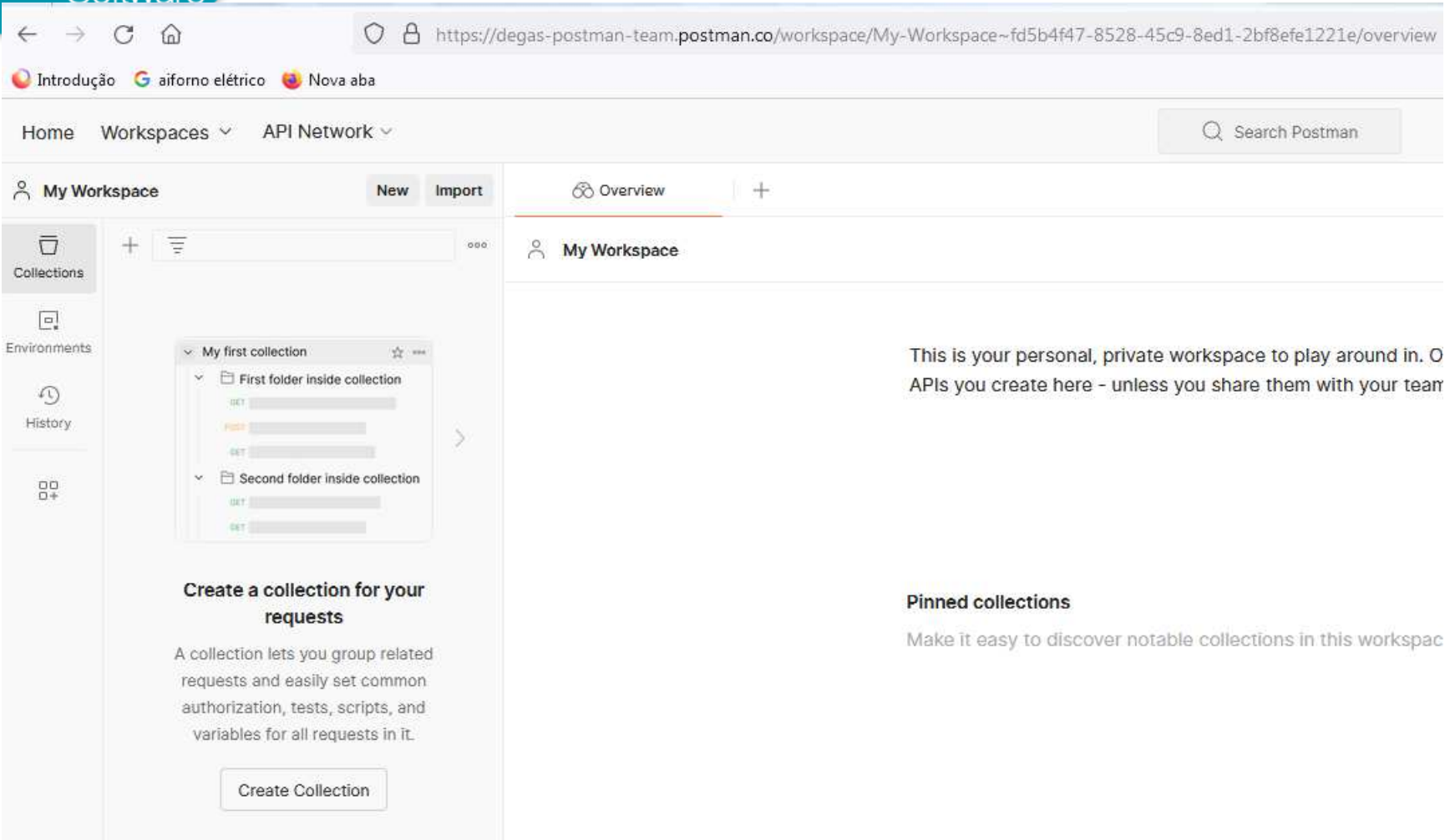
Download the desktop app for



What is Postman?



Acesse seu workspace



The screenshot shows the Postman web interface. The browser address bar displays the URL: <https://degas-postman-team.postman.co/workspace/My-Workspace~fd5b4f47-8528-45c9-8ed1-2bf8efe1221e/overview>. The navigation bar includes links for Home, Workspaces, and API Network, along with a search bar labeled "Search Postman". The main content area is titled "My Workspace" and features a sidebar with "Collections", "Environments", and "History". The "Collections" section is active, showing a tree view of "My first collection" with two folders: "First folder inside collection" and "Second folder inside collection". Each folder contains a "GET" request. Below the tree view, there is a section titled "Create a collection for your requests" with a description: "A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it." and a "Create Collection" button. The right side of the workspace overview shows a heading "My Workspace" and a description: "This is your personal, private workspace to play around in. O APIs you create here - unless you share them with your team". Below this, there is a section titled "Pinned collections" with the text: "Make it easy to discover notable collections in this workspac".

Introdução aiforno elétrico Nova aba

Home Workspaces API Network Search Postman

My Workspace New Import Overview +

Collections + Environments History

My first collection

- First folder inside collection
 - GET
- Second folder inside collection
 - GET

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

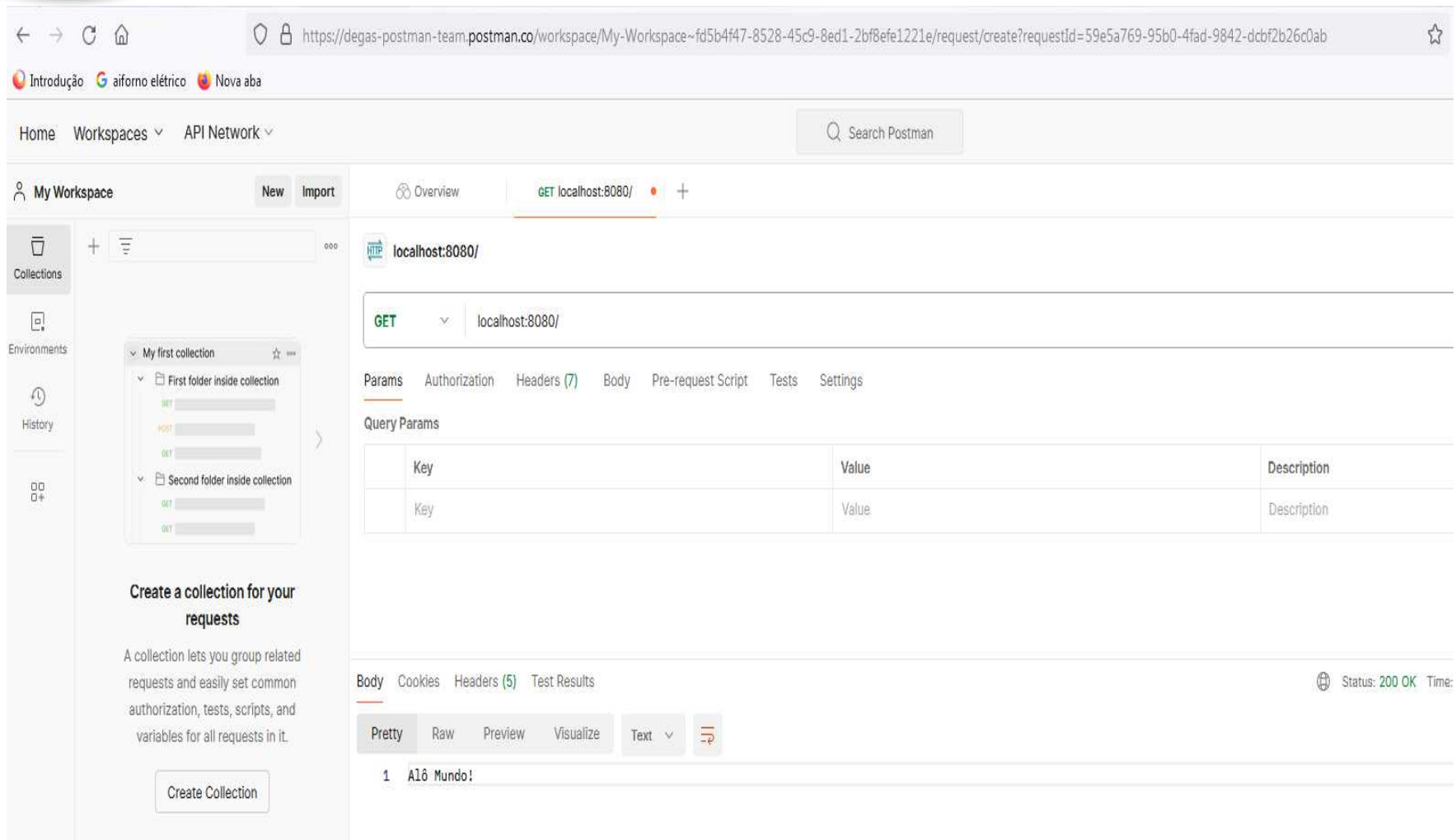
Create Collection

This is your personal, private workspace to play around in. O APIs you create here - unless you share them with your team

Pinned collections

Make it easy to discover notable collections in this workspac

Postman



The screenshot displays the Postman web interface in a browser. The address bar shows the URL: `https://degas-postman-team.postman.co/workspace/My-Workspace~fd5b4f47-8528-45c9-8ed1-2bf8efe1221e/request/create?requestId=59e5a769-95b0-4fad-9842-dcbf2b26c0ab`. The interface includes a sidebar with navigation options: Home, Workspaces, and API Network. The main workspace is titled "My Workspace" and shows a "GET localhost:8080/" request. The "Query Params" section is visible, showing a table with columns "Key", "Value", and "Description". The "Body" tab is selected, showing a "Pretty" view of the response: "1 Alô Mundo!".

Introdução aiforno elétrico Nova aba

Home Workspaces API Network

Search Postman

My Workspace New Import Overview GET localhost:8080/ +

Collections

Environments

History

My first collection

- First folder inside collection
 - GET
- Second folder inside collection
 - GET

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create Collection

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

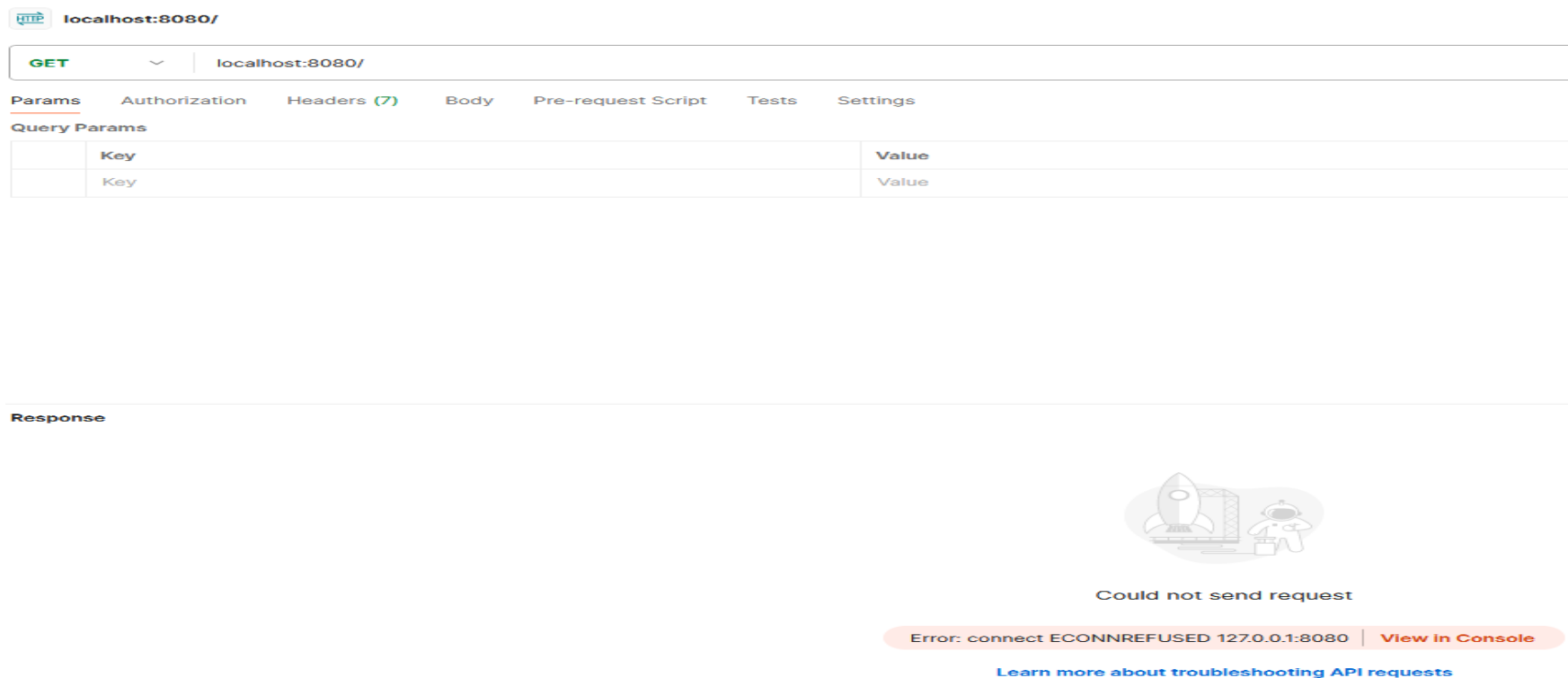
Status: 200 OK Time:

Pretty Raw Preview Visualize Text

1 Alô Mundo!

Crie uma requisição

- New -> HTTP
- Tente executar GET localhost:8080/
 - PostMan Agent

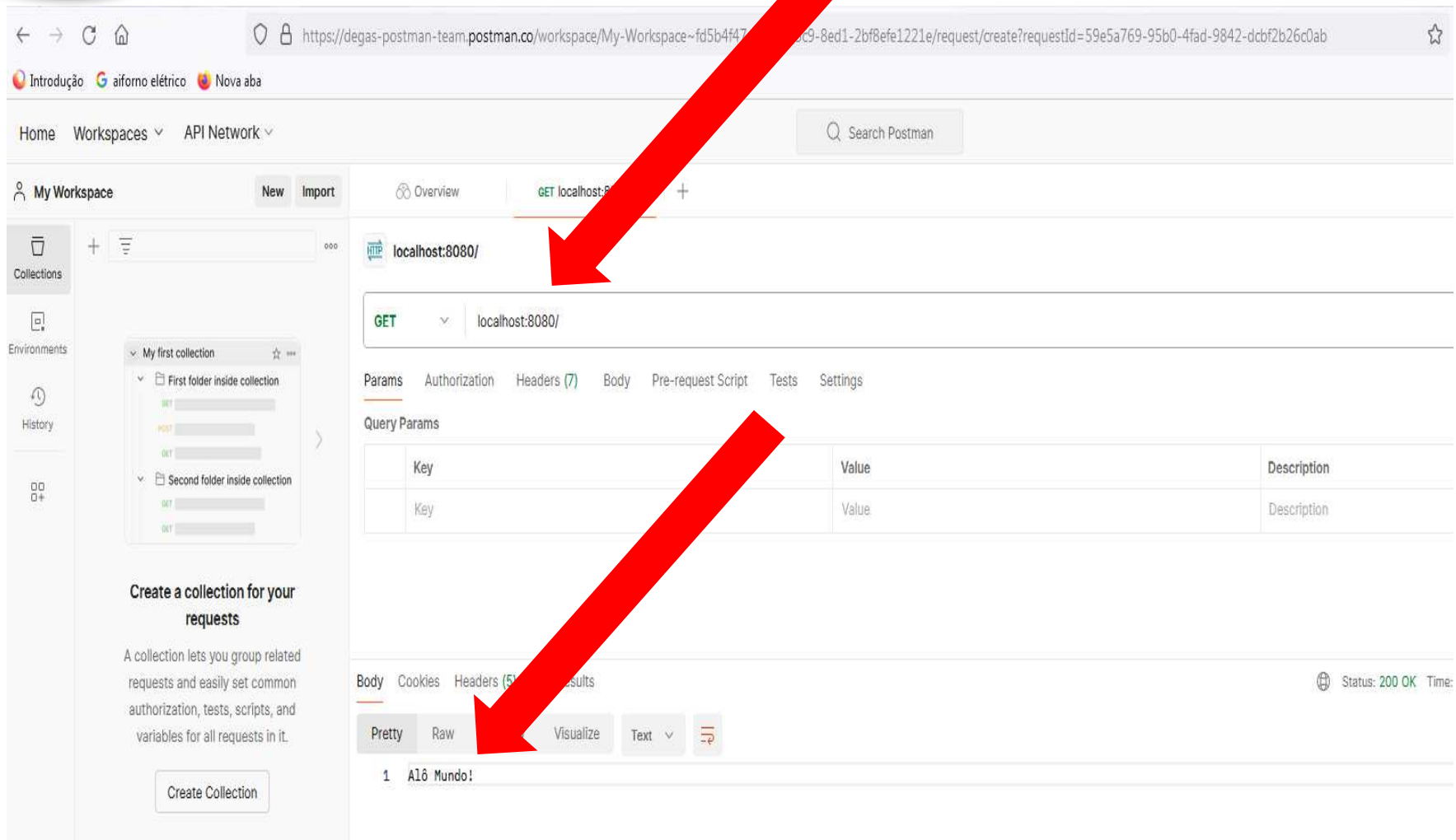




Residência
em Software

Ok. Inicie sua Aplicação

Postman



The screenshot shows the Postman web interface. The browser address bar displays the URL: `https://degas-postman-team.postman.co/workspace/My-Workspace~fd5b4f47...c9-8ed1-2bf8efe1221e/request/create?requestId=59e5a769-95b0-4fad-9842-dcbf2b26c0ab`. The interface includes a sidebar with 'My Workspace', 'Collections', 'Environments', and 'History'. The main area shows a request configuration for `localhost:8080/` with a `GET` method. The 'Query Params' section is visible, and the 'Body' tab is selected. The response is displayed in 'Pretty' view, showing a status of `200 OK` and the text `Alô Mundo!`. Two red arrows are overlaid on the image: one pointing to the URL bar and another pointing to the 'Raw' view button in the response section.

Introdução aiforno elétrico Nova aba

Home Workspaces API Network Search Postman

My Workspace New Import Overview GET localhost:8080/

Collections

Environments

History

My first collection

- First folder inside collection
 - GET
 - POST
 - GET
- Second folder inside collection
 - GET
 - GET

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create Collection

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Results

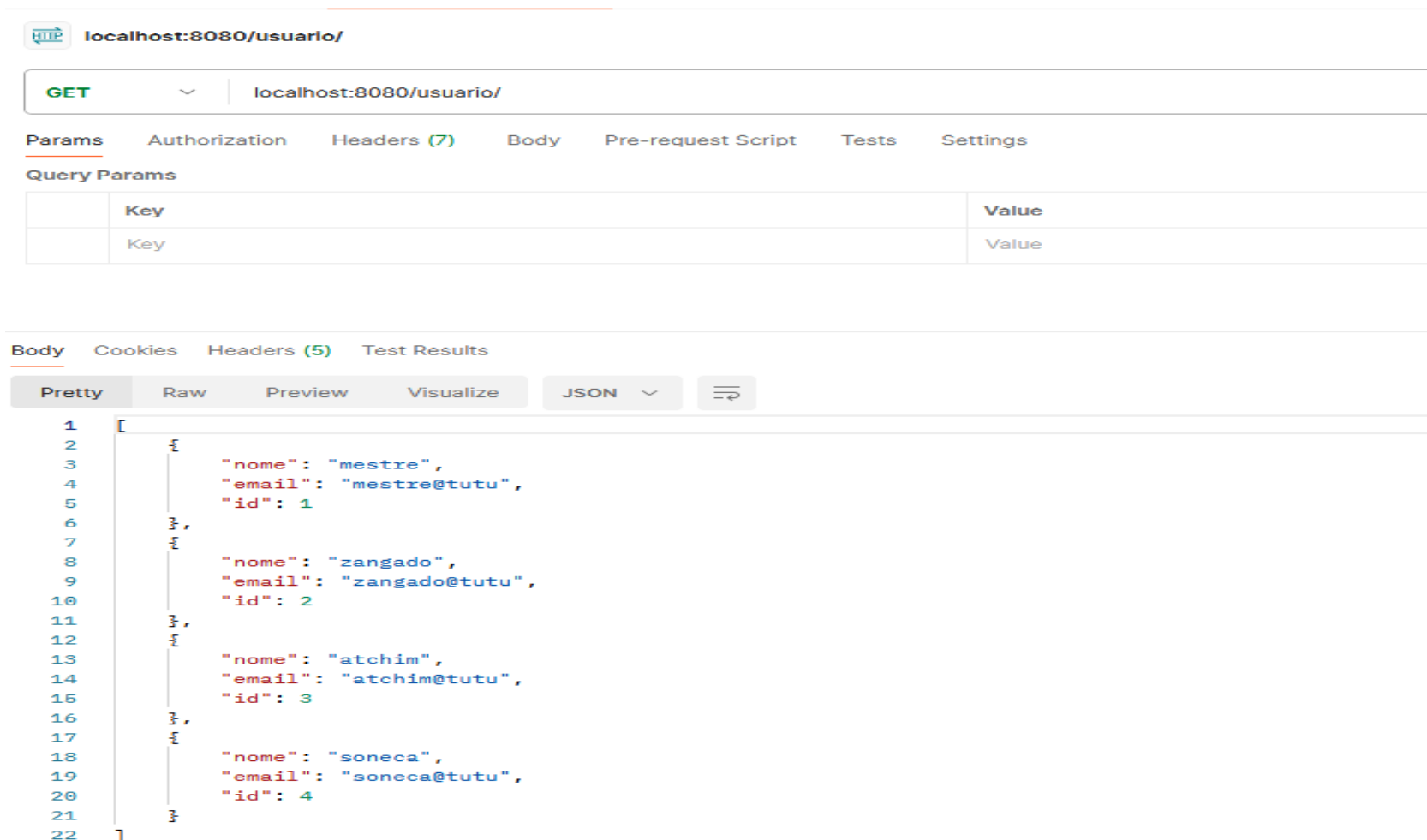
Pretty Raw Visualize Text

1 Alô Mundo!

Status: 200 OK Time:

Crie uma requisição

- (inicie sua API) New -> HTTP
- Tente executar GET localhost:8080/usuario/



The screenshot shows a REST client interface with the following components:

- URL Bar:** Shows the URL `localhost:8080/usuario/`.
- Method:** A dropdown menu set to `GET`.
- Params:** A tab labeled "Query Params" with a table containing two rows, each with "Key" and "Value" headers.
- Body:** A tab showing the response body in JSON format, with a line number indicator on the left (1-22).

The JSON response body is as follows:

```
[
  {
    "nome": "mestre",
    "email": "mestre@tutu",
    "id": 1
  },
  {
    "nome": "zangado",
    "email": "zangado@tutu",
    "id": 2
  },
  {
    "nome": "atchim",
    "email": "atchim@tutu",
    "id": 3
  },
  {
    "nome": "soneca",
    "email": "soneca@tutu",
    "id": 4
  }
]
```


Configurando um POST

- Selecione New -> HTTP
- Verbo POST localhost:8080/usuario/
- Vamos criar um novo Header
 - Key: Content-Type (deixe a interface te ajudar)
 - Value: application/json
- Isto indica que a requisição para acrescentar um usuário vai mandar seus dados através de um arquivo json

Configurando um POST


HTTP localhost:8080/usuario/

POST localhost:8080/usuario/

Params Authorization Headers (9) Body Pre-request Script Tests Settings

<input checked="" type="checkbox"/>	Host	<calculated when request is sent>
<input checked="" type="checkbox"/>	User-Agent	PostmanRuntime/7.36.1
<input checked="" type="checkbox"/>	Accept	*/*
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br
<input checked="" type="checkbox"/>	Connection	keep-alive
<input checked="" type="checkbox"/>	Content-Type	application/json
	Key	Value

Response



Configurando um POST

- Body
 - O Json com os dados do usuário
 - O Springdata vai extrair os dados e colocar no objeto tipo UserForm

Corpo da mensagem

- Na aba Body
- Escolha a opção **raw**
- Indica que os dados seguirão em linha (texto)
- Insira o dado (UsuarioFORM) no formado JSON

```
{  
    "nome": "dunga",  
    "email": "dunga@tutu",  
    "senha": "dunga"  
}
```

Requisição pronta

- Mas a aplicação não
- Precisamos programar
- Método inserir (com anotação `@PostMapping`) recebe um objeto do tipo `UsuarioFORM`
 - `@RequestBody` no parâmetro (formato JSON)
 - Cria um objeto do tipo `Usuario` (usando o parâmetro `UsuarioFORM`)
 - Executa o método `save()` do `usuarioRepository`



Residência
em Software

Programa aí!

Como fica

- Método inserir do Controller de usuario
 - Observe que precisamos programar o método toUsuario
 - Classe UsuarioFORM

```
@PostMapping
public void inserir (@RequestBody UsuarioFORM usuarioFORM) {
    Usuario usuario = usuarioFORM.toUsuario();
    usuarioRepository.save(usuario);
}
```

Como fica

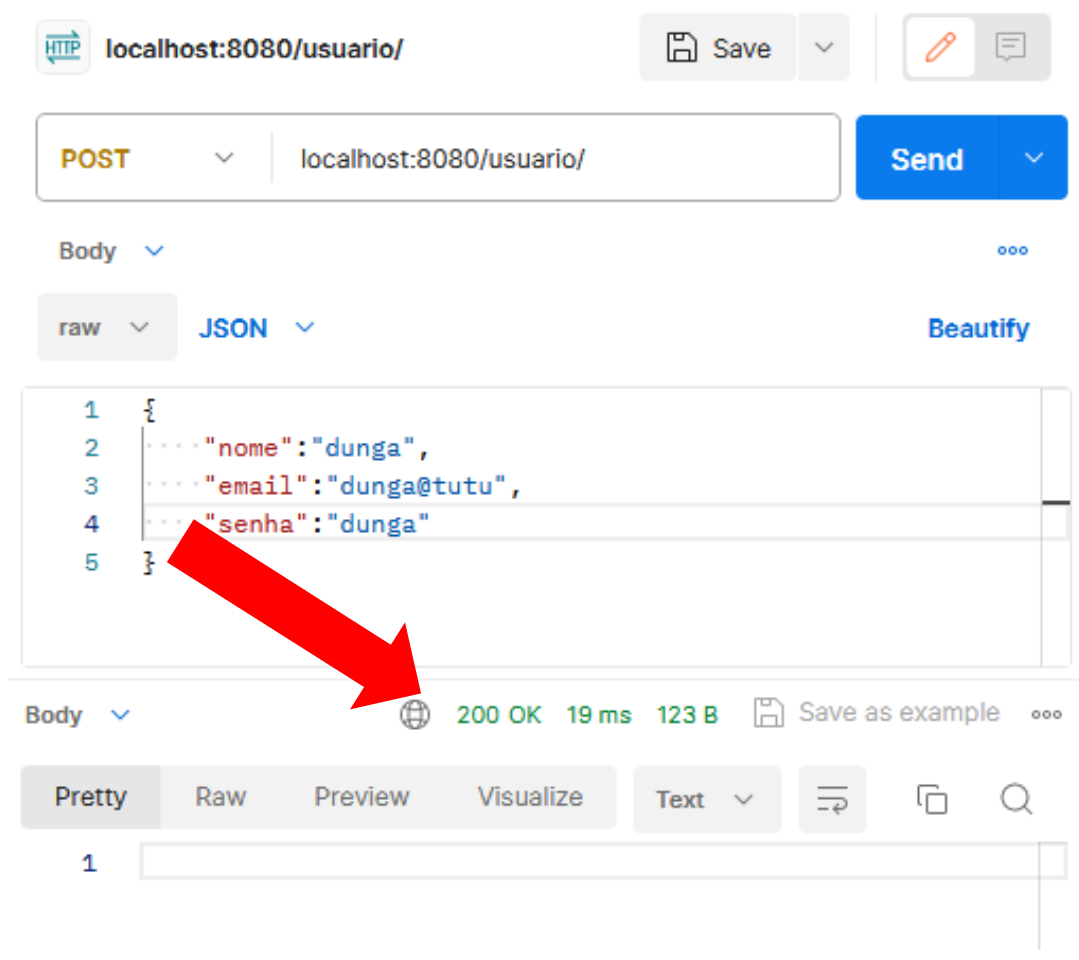
• Classe UsuarioFORM

```
public class UsuarioFORM {  
  
    private String nome;  
    private String email;  
    private String senha;  
  
    public UsuarioFORM() {  
  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getEmail() {  
        return email;  
    }  
  
    public void setEmail(String email) {  
        this.email = email;  
    }  
}
```

```
    public String getSenha() {  
        return senha;  
    }  
  
    public void setSenha(String senha) {  
        this.senha = senha;  
    }  
  
    public Usuario toUsuario() {  
        Usuario usuario = new Usuario();  
        usuario.setNome(nome);  
        usuario.setSenha(senha);  
        usuario.setEmail(email);  
        return usuario;  
    }  
}
```


Execute sua requisição

- POST localhost:8080/usuario/
 - Body: json



Verifique no H2

localhost:8080/h2-console/login.do?jsessi

Nova aba

Max rows: 1000 | Auto complete

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM USUARIO |

SELECT * FROM USUARIO;

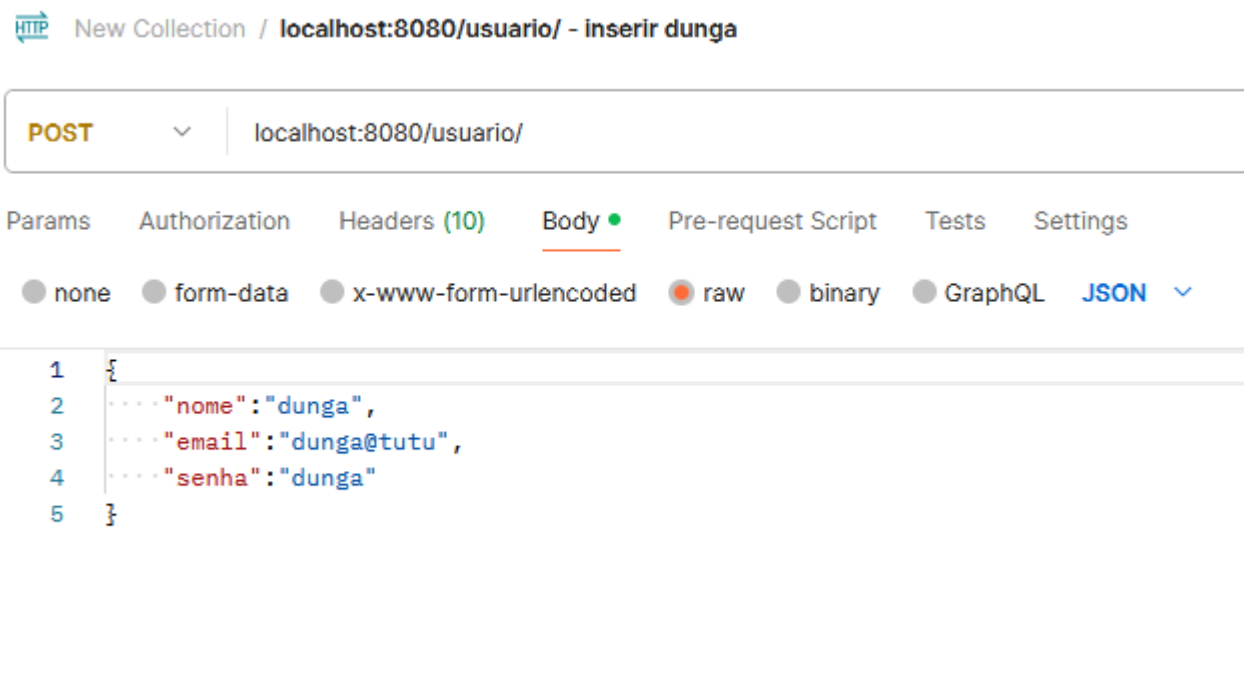
ID	EMAIL	NOME	SENHA
1	mestre@tutu	mestre	mestre
2	zangado@tutu	zangado	zangado
3	atchim@tutu	atchim	atchim
4	soneca@tutu	soneca	soneca
6	dunga@tutu	dunga	dunga

(5 rows, 0 ms)

Edit

Salve sua requisição

- É útil ter algumas já configuradas para testar



Retorno

- Por padrão, o SpringBoot vai devolver código html 200
 - Processado com sucesso
- O ideal: Status 201 Recurso criado com sucesso
 - Mais especificidade no tipo do processamento
 - Insert

ResponseEntity

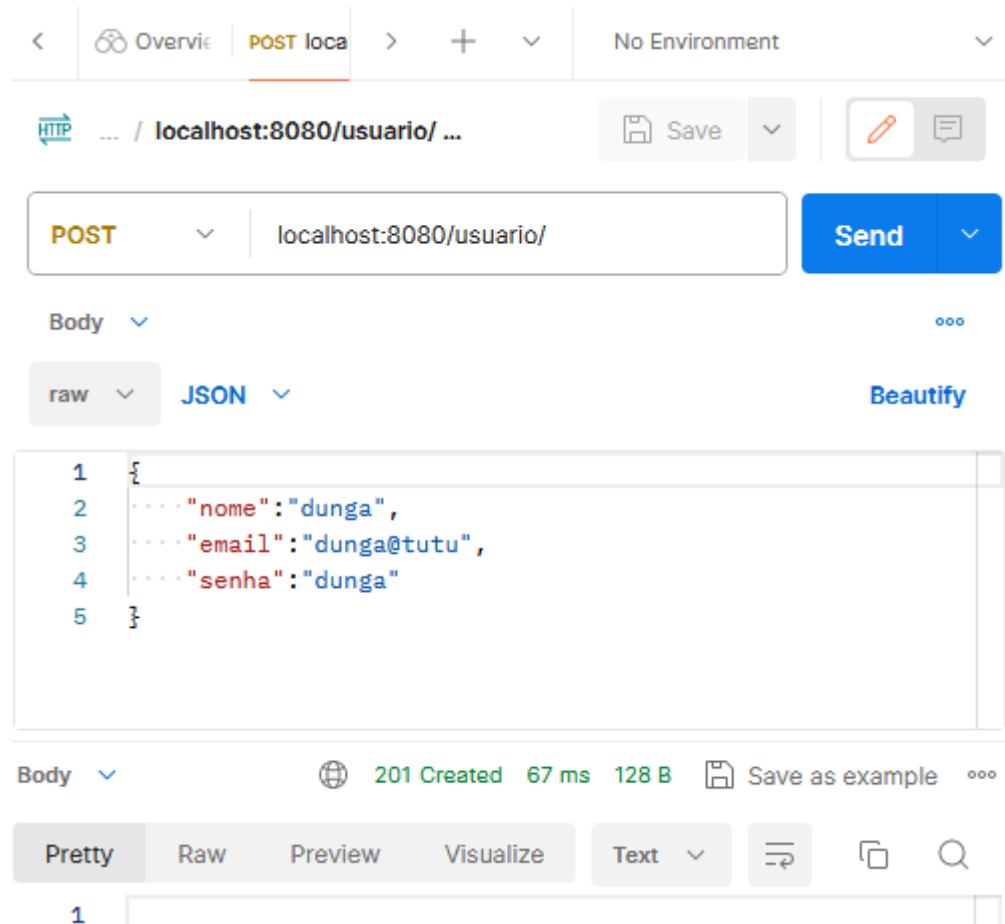
- Tipo de dados para respostas do sistema
 - Retorno de um método que atendeu uma requisição
- ResponseEntity representa a resposta HTTP completa
 - Status, cabeçalho e corpo
- No corpo incluimos objetos (preferencialmente DTO)
 - Ex; `public ResponseEntity<UsuarioDTO> inserir`
 - Método retorna um objeto que embute um dado do tipo UsuarioDTO no seu corpo
- Status: existem vários tipos de ResponseEntity
 - Exemplo: `ResponseEntity.ok`
 - Status 200
 - No Caso: `ResponseEntity.created`
 - Status 201

ResponseEntity

- Melhorando o método
 - Ainda precisaremos programar mais!

```
@PostMapping
public ResponseEntity<UsuarioDTO> inserir
    (@RequestBody UsuarioFORM usuarioFORM) {
    Usuario usuario = usuarioFORM.toUsuario();
    usuarioRepository.save(usuario);
    return new ResponseEntity<>(HttpStatus.CREATED);
}
```

ResponseEntity



The screenshot displays a REST client interface with the following components:

- Navigation:** Back, Overview, **POST local**, Forward, Add, and Environment dropdown (No Environment).
- Request Bar:** HTTP icon, URL `localhost:8080/usuario/`, Save button, Edit/Comments icons.
- Method and URL:** **POST** dropdown, `localhost:8080/usuario/` text field, **Send** button.
- Body Section:**
 - Body dropdown (expanded).
 - raw dropdown, **JSON** dropdown, Beautify button.
 - Code editor with the following JSON body:

```
1 {  
2   "nome": "dunga",  
3   "email": "dunga@tutu",  
4   "senha": "dunga"  
5 }
```
- Response Section:**
 - Body dropdown (expanded).
 - Status: 201 Created, Time: 67 ms, Size: 128 B, Save as example button.
 - Format tabs: **Pretty**, Raw, Preview, Visualize, Text dropdown.
 - Icons for expand, copy, and search.
 - Line 1 of the response body is visible in the editor.

Retorno

- Boas práticas
- Boa prática 1: Após a inserção
 - Devolver uma representação do dado que foi excluído
- Facilita para o programador da Interface

URI

- Uniform Resource Identifier (Identificador Uniforme de Recursos)
 - String que se refere a um recurso na WEB
- URL é um URI
- Boa prática 2: devolver a URI onde um recurso recém-criado pode ser encontrado
- No nosso caso: `/usuario/{Id}`
 - `{Id}` é o Id do usuário criado
 - Ex. `/usuario/5`

Construindo o ResponseEntity

- O que queremos fazer
 - Construir uma ResponseEntity
 - Inserir nela um objeto UsuarioDTO
 - Associar uma URI
 - Devolver a ResponseEntity construída
- Nosso caso: Responsentity.created
 - http status: 201

Facilidade do SpringBoot

- Podemos indicar um objeto a mais como parâmetro
 - Ele sempre está ali, mas não usamos se não precisamos
- Um objeto do tipo UriComponentsBuilder
 - Vai nos permitir definir a URI que vamos associar à ResponseEntity

Novo cabeçalho

- Nosso método agora tem dois parâmetros

```
public ResponseEntity inserir(  
    UsuarioFORM usuarioform,  
    UriComponentsBuilder uribuilder)
```

- Agora temos tudo o que precisamos para produzir o código

Inserir (Usuário)

- Parâmetros: *usuarioform* e *uribuilder*
 - Criar um objeto do tipo Usuario (que é entidade)
 - Salvar este objeto (Usando o usuarioRepository)
 - Criar um objeto do tipo UsuarioDTO
 - Configurar o *uribuilder*
 - Path: /usuario/{id} onde id na URI será o Id do usuário que salvamos
 - Criar a URI
 - Substituindo {id} pelo getId() do usuário
 - Retornar a ResponseEntity (created) associada à URI e com o UsuarioDTO no corpo

Inserir (Usuário)

- ```
uriBuilder.path("/usuario/{id}");
```

  - Salvar este objeto (Usando o `usuarioRepository`)
  - Criar um objeto do tipo `UsuarioDTO`
  - Configurar o *uribuilder*
    - Path: `/usuario/{id}` onde `id` na URI será o `Id` do usuário que salvamos
  - Criar a URI
    - Substituindo `{id}` pelo `getId()` do usuário
  - Retornar a `ResponseEntity (created)` associada à URI e com o `UsuarioDTO` no corpo

# Inserir (Usuário)

- Parâmetros: *usuarioform* e *uribuilder*
  - Criar um objeto do tipo Usuario (que é entidade)
  - Salvar este objeto (Usando o usuarioRepository)
  - `URI uri = uriBuilder.buildAndExpand(usuario.getId()).toUri();`
  - Configurar o *uribuilder*
    - Path: `/usuario/{id}` onde id na URI será o Id do usuário que salvamos
  - Criar a URI
    - Substituindo `{id}` pelo `getId()` do usuário
  - Retornar a ResponseEntity (created) associada à URI e com o UsuarioDTO no corpo

# Inserir (Usuário)

- Parâmetros: *usuarioform* e *uribuilder*
  - Criar um objeto do tipo Usuario (que é entidade)
  - Salvar este objeto (Usando o usuarioRepository)
  - Criar um objeto do tipo UsuarioDTO
  - Configurar o *uribuilder*
    - **return** `ResponseEntity.created(uri).body(usuarioDTO);`
  - Criar a URI
    - Substituindo {id} pelo `getId()` do usuário
  - Retornar a `ResponseEntity (created)` associada à URI e com o `UsuarioDTO` no corpo





Residência  
em Software

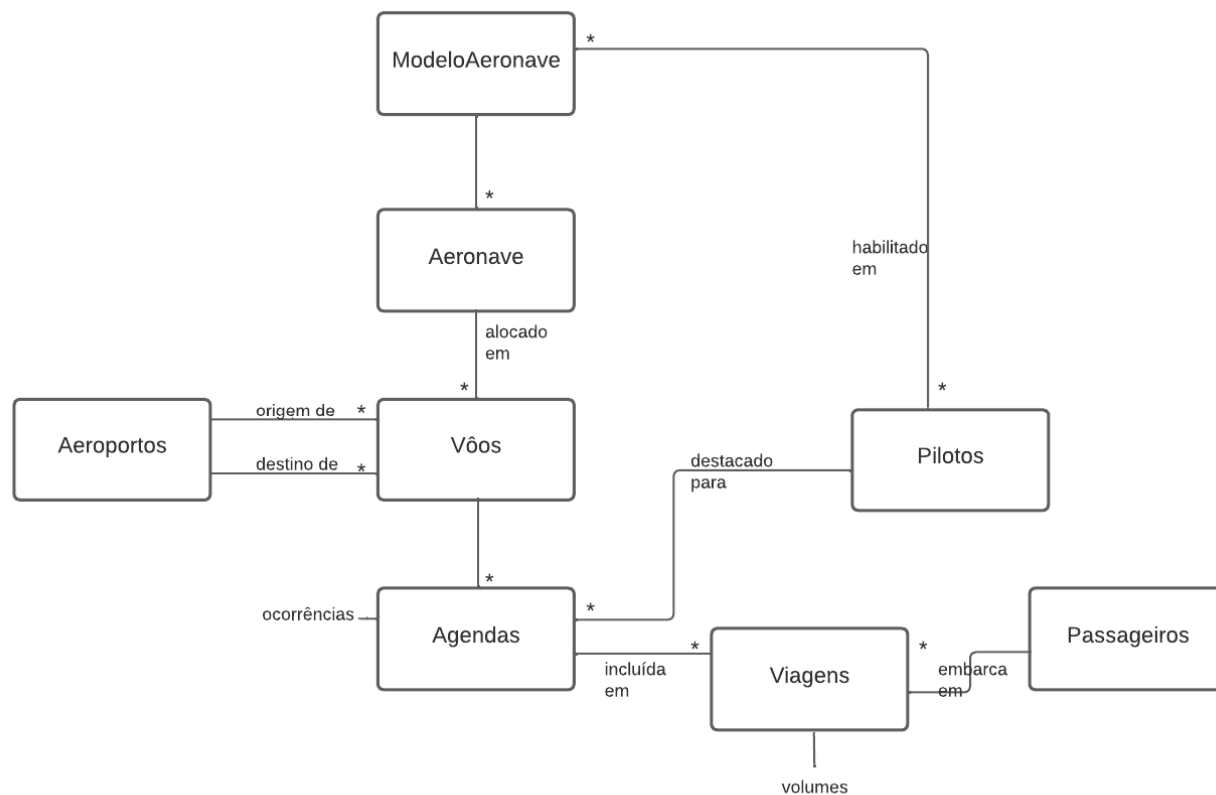
# Programa aí!

# Como fica

```
@PostMapping
public ResponseEntity<UsuarioDTO> inserir
 (@RequestBody UsuarioFORM usuarioFORM,
 UriComponentsBuilder uriBuilder) {
 Usuario usuario = usuarioFORM.toUsuario();
 usuarioRepository.save(usuario);
 UsuarioDTO usuarioDTO = new UsuarioDTO(usuario);
 uriBuilder.path("/usuario/{id}");
 URI uri = uriBuilder.buildAndExpand(usuario.getId()).toUri();
 return ResponseEntity.created(uri).body(usuarioDTO);
}
```

# Exercício

- De volta ao nosso sistema da Companhia aérea



# Inserir

- Verbo POST em todos os Endpoints
  - /pilotos/
  - /aeroportos/
  - /modelo aeronave/
- Recebe sempre JSON com dados
  - PilotoFORM (Nome, NumBreve)
  - AeroportoFORM (ICAO, Nome)
  - ModeloAeronaveFORM (Fabricante, Nome)

Não esquecer de devolver o status  
correto (HTTP 201)

Não esquecer de devolver o DTO do  
objeto inserido