

Projeto 02 - Blog Pessoal - CRUD 07

O que veremos por aqui:

1. Criar o método `post(Postagem postagem)` para persistir uma nova Postagem no Banco de Dados
2. Criar o método `put(Postagem postagem)` para atualizar uma Postagem persistida no Banco de Dados

1. O Recurso Postagem

Nas etapas anteriores, começamos a construir a Classe **PostagemController** e implementamos os Métodos:

- **getAll()** → Retorna todos os Objetos da Classe Postagem persistidos no Banco de dados.
- **getById(Long id)** → Retorna um Objeto específico da Classe Postagem persistido no Banco de dados. A Postagem é identificada pelo atributo id.
- **getByTitulo(String titulo)** → Retorna todos os Objetos da Classe Postagem persistidos no Banco de dados, cujo atributo titulo contenha a String enviada no parâmetro titulo do Método.

Vamos continuar a construção da nossa Classe Controladora implementando o **Método post(Postagem postagem)**, que persistirá um novo Objeto da Classe Postagem no Banco de dados e o **Método put(Postagem postagem)**, que atualizará um Objeto da Classe Postagem persistido no Banco de dados.

Postagem
-id: Long -titulo: String -texto: String -data: LocalDateTime
+ getAll():ResponseEntity<List<Postagem>> + getById(Long id):ResponseEntity<Postagem> + getByTitulo(String nome):ResponseEntity<List<Postagem>> + post(Postagem postagem):ResponseEntity<Postagem> + put(Postagem postagem):ResponseEntity<Postagem> + delete(Long id):void



Passo 01 - Criar o Método `post(Postagem postagem)`

Vamos implementar o Método **post(Postagem postagem)** na Classe Postagem Controller.

Traçando um paralelo com o MySQL, seria o equivalente a instrução: `INSERT INTO tb_postagens (titulo, texto, data) VALUES ("Título", "Texto", CURRENT_TIMESTAMP());`.

```

46
47 @PostMapping
48 public ResponseEntity<Postagem> post(@Valid @RequestBody Postagem postagem){
49     return ResponseEntity.status(HttpStatus.CREATED)
50         .body(postagemRepository.save(postagem));
51 }
52

```

Linha 47: a anotação **@PostMapping** indica que o Método `post(Postagem postagem)`, responderá a todas as requisições do tipo **HTTP POST**, enviadas no endereço <http://localhost:8080/postagens>.



ATENÇÃO: O Endereço do Endpoint será igual ao Endereço do Recurso (@RequestMapping). O Método `getAll()` utiliza o mesmo endereço, porém como se tratam de verbos diferentes (um é o GET e ou outro é o POST) o endereço pode ser o mesmo.

Linha 48: O Método `ResponseEntity<Postagem> post(@Valid @RequestBody Postagem postagem)` será do tipo **ResponseEntity** porque ele responderá Requisições HTTP (HTTP Request), com uma **Resposta HTTP** (HTTP Response). Observe que o Método possui um parâmetro, que é um Objeto da **Classe Postagem**, chamado **postagem**.

@Valid: Esta anotação valida o Objeto `Postagem` enviado no Corpo da Requisição (Request Body), conforme as regras definidas na Model `Postagem` (@NotNull, @NotBlank, @Size e etc).

@RequestBody Postagem postagem: Esta anotação recebe o Objeto do tipo `Postagem` enviado no Corpo da Requisição (Request Body) e insere no parâmetro `Postagem` do método `post`.

<Postagem>: O Método além de retornar um objeto da **Classe ResponseEntity** (CREATED→201), no parâmetro Body (Corpo da Resposta), será retornado o **Objeto da Classe Postagem**, que foi persistido no Banco de dados, na tabela **tb_postagens**, com o atributo `id` preenchido pelo Banco de dados (auto incremento). Observe que nesta linha também foi utilizado o recurso **Java Generics** para simplificar o retorno do Objeto.

Linha 49: return

`ResponseEntity.status(HttpStatus.CREATED).body(postagemRepository.save(postagem));`
Executa o Método padrão da Interface `JpaRepository` (`save(postagem)`) e retorna o **HTTP Status CREATED→201** se o Objeto foi persistido no Banco de dados.



[Documentação: @PostMapping](#)



[Documentação: ResponseEntity](#)



[Documentação: HttpStatus](#)



[Documentação: .save\(T entidade\)](#)



[Documentação: @RequestBody](#)



[Documentação: Java Generics](#)



[Documentação: @Valid](#)

Para concluir, não esqueça de Salvar o código (**File → Save All**) e verificar se o Projeto está em execução

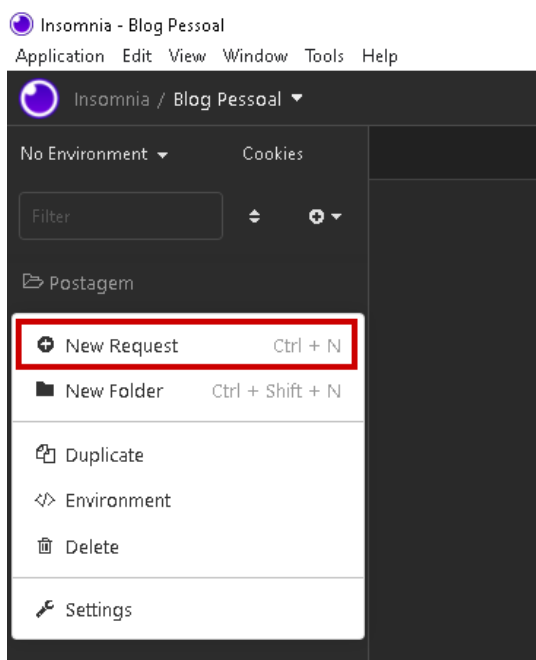


[Código fonte do projeto](#)

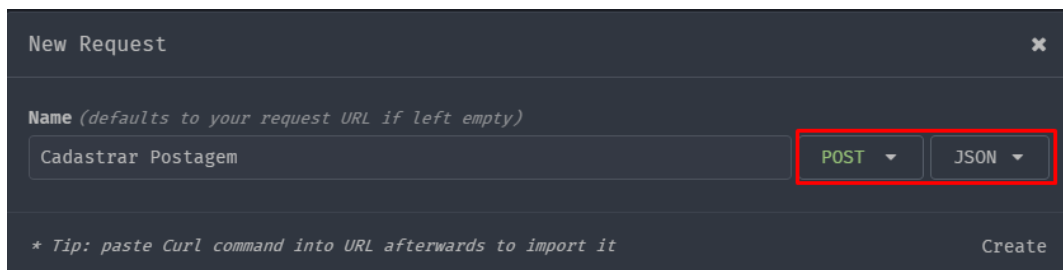
Passo 02 - Testar no Insomnia o Método Post

Agora vamos criar a Requisição para o **Método post(Postagem postagem)**:

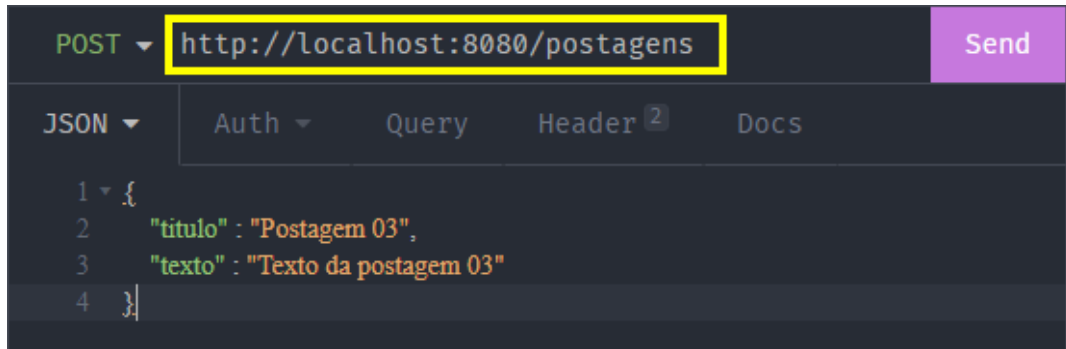
1. Clique com o botão direito do mouse sobre a **Pasta Postagem** para abrir o menu e clique na opção **New Request**.



2. Na janela que será aberta, informe o nome da requisição e o Método HTTP que será utilizado (**POST**). Depois de selecionar o Método **POST**, observe que será necessário informar o formato em que os dados serão enviados através do Corpo da Requisição. Selecione o formato **JSON**, como indicado na imagem abaixo em vermelho e clique no botão **Create** para concluir.



3. Configure a requisição conforme a imagem abaixo:



4. No item marcado em amarelo na imagem acima, informe o endereço (endpoint) da Requisição. A requisição **Cadastrar Postagem** foi configurada da seguinte maneira:

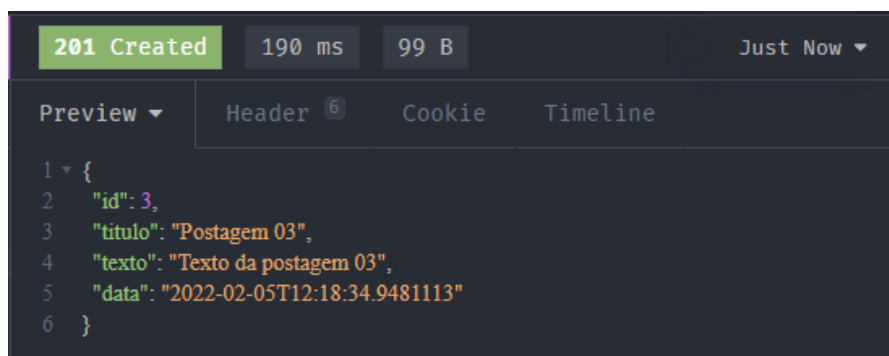
- A primeira parte do endereço (<http://localhost:8080>) é o endereço do nosso servidor local. Quando a API estiver na nuvem, ele será substituído pelo endereço da aplicação na nuvem.
- A segunda parte do endereço é o **endpoint** configurado na anotação **@RequestMapping**, em nosso caso **/postagens**.

5. Na guia **JSON**, precisamos inserir um **JSON** com os dados que serão inseridos na nova postagem. Lembrando que no padrão JSON: **o texto antes dos 2 pontos (:) é o atributo** da Classe e **o texto depois dos 2 pontos (:) é o dado** que será cadastrado no atributo. Os atributos são separados por vírgula, como mostra a imagem acima.

ATENÇÃO: Observe que no método **POST** não é necessário enviar os atributos **id** (será gerado pelo Banco de dados) e **data** (será enviada pelo Sistema Operacional) no JSON.

6. Para testar a requisição, com a aplicação rodando, clique no botão

7. O resultado da requisição você confere na imagem abaixo:



8. Observe que a aplicação retorna além dos dados que foram persistidos no Banco de dados com o **id** e a **data**, ela também retorna um **HTTP Status 201 → CREATED** (indicado em verde na imagem acima). Este Status indica que a Requisição foi bem sucedida!

Passo 03 - Criar o Método put(Postagem postagem)

Vamos implementar o Método **put(Postagem postagem)** na Classe **Postagem Controller**. Observe que ele é muito parecido com o Método **post**. Traçando um paralelo com o **MySQL**, seria o equivalente a instrução: `UPDATE tb_postagens SET titulo = "titulo", texto = "texto", data = CURRENT_TIMESTAMP() WHERE id = id;`

```

52
53 @PutMapping
54 public ResponseEntity<Postagem> put(@Valid @RequestBody Postagem postagem){
55     return postagemRepository.findById(postagem.getId())
56         .map(resposta -> ResponseEntity.status(HttpStatus.OK)
57             .body(postagemRepository.save(postagem)))
58         .orElse(ResponseEntity.status(HttpStatus.NOT_FOUND).build());
59 }
60

```

Linha 53: a anotação **@PutMapping** indica que o Método `put(Postagem postagem)`, responderá a todas as requisições do tipo **HTTP PUT**, enviadas no endereço <http://localhost:8080/postagens>.

Linha 54: O Método **`ResponseEntity<Postagem> put (@Valid @RequestBody Postagem postagem)`** será do tipo **`ResponseEntity`** porque ele responderá Requisições HTTP (HTTP Request), com uma **Resposta HTTP** (HTTP Response). Observe que o Método possui um parâmetro, que é um Objeto da **Classe Postagem**, chamado **`postagem`**.

@Valid: Esta anotação valida o Objeto Postagem enviado no Corpo da Requisição (Request Body), conforme as regras definidas na Model Postagem (@NotNull, @NotBlank, @Size e etc).

@RequestBody Postagem postagem: Esta anotação recebe o Objeto do tipo Postagem enviado no Corpo da Requisição (Request Body) e insere no parâmetro Postagem do método `put`.

<Postagem>: O Método além de retornar um objeto da **Classe ResponseEntity** (OK→200), no parâmetro Body (Corpo da Resposta), será retornado o **Objeto da Classe Postagem**, que foi atualizado no Banco de dados, na tabela **`tb_postagens`**.

Linha 55:

`return postagemRepository.findById(postagem.getId());` Retorna a execução do método **`findById(id)`**. O Método retornará um **Objeto da Classe Postagem** persistido no Banco de dados, caso ele seja encontrado a partir do parâmetro **`id`**, que foi recuperado do Objeto postagem enviado. Caso contrário, será retornado um Objeto Nulo. Observe que o **`id da postagem`** foi recuperado através do método **`getId()`** que retorna o Id da Postagem enviada no Corpo da Requisição (Request Body)

`.map(resposta →`

`ResponseEntity.status(HttpStatus.OK).body(postagemRepository.save(postagem));` Se o Objeto da Classe Postagem for encontrado, o método **`map`** (Optional), mapeia no **Objeto resposta** o retorno do método **`findById(id)`**, mas ao invés de exibir o Objeto **`resposta`** no Corpo da Resposta, vamos executar o Método **`postagemRepository.save(postagem)`**, que retornará o Objeto postagem atualizado e o HTTP Status **`OK→200`**.

O Método `save(postagem)` funciona da seguinte maneira:

- **Se o Objeto enviado possuir o atributo `id`** → Ele checa se o Objeto existe e atualiza todos os atributos. Se não existir o Objeto, ele cria um novo com os dados enviados. Por este motivo fazemos a checagem se o Objeto existe através do Método **`findById(Long id)`** antes de executar o Método **`save(postagem)`**.
- **Se o Objeto enviado não possuir o atributo `id`** → Ele cria um novo com os dados enviados.

`.orElse(ResponseEntity.notFound().build());` Se o Objeto Postagem não for encontrado (Nulo) pelo Método **`findById(id)`**, será retornado o HTTP Status **`NOT FOUND → 404`** (Não Encontrado!). O método **`build()`** constrói a Resposta com o HTTP Status retornado.



[Documentação: @PutMapping](#)

[Documentação: ResponseEntity](#)

[Documentação: HttpStatus](#)

[Documentação: .save\(T entidade\)](#)

[Documentação: @RequestBody](#)

[Documentação: Java Generics](#)

[Documentação: @Valid](#)

Para concluir, não esqueça de Salvar o código (**File** → **Save All**) .



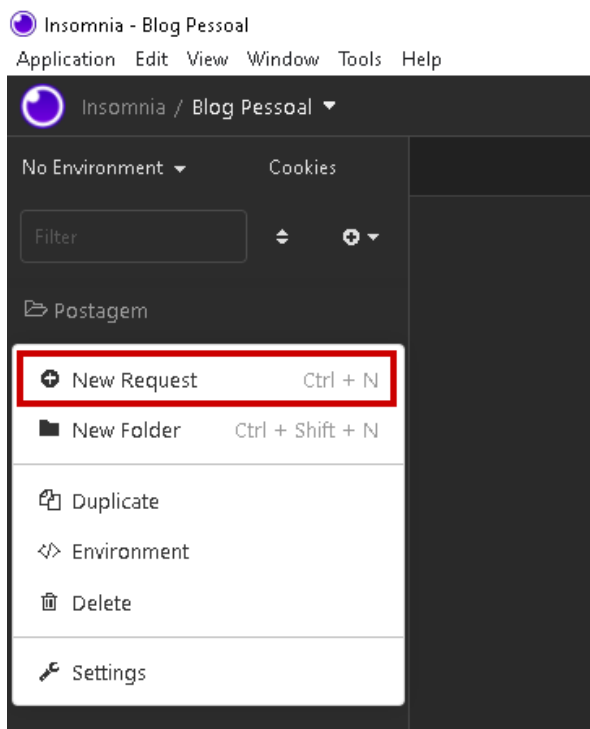
[Código fonte do projeto](#)



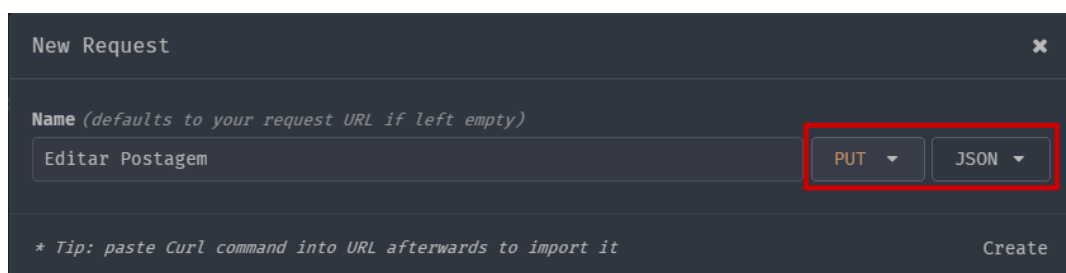
Passo 04 - Testar no Insomnia o Método put

Agora vamos criar a Requisição para o **Método put(Postagem postagem)**:

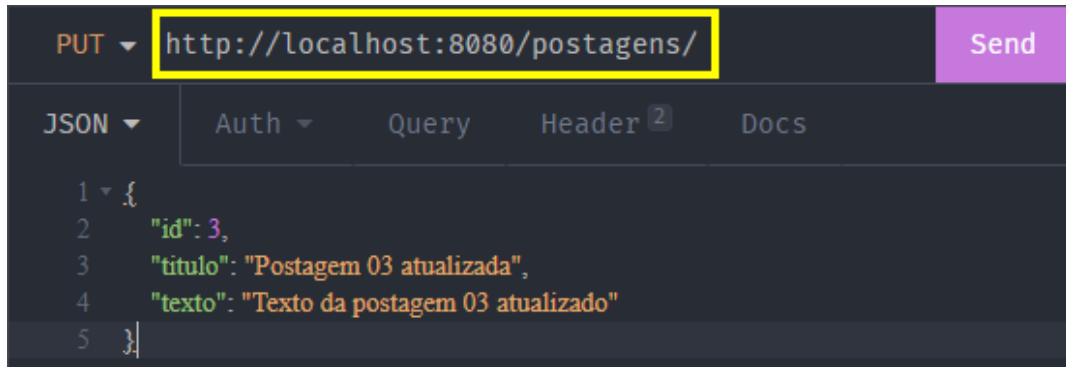
1. Clique com o botão direito do mouse sobre a **Pasta Postagem** para abrir o menu e clique na opção **New Request**.



2. Na janela que será aberta, informe o nome da requisição e o Método HTTP que será utilizado (**PUT**). Depois de selecionar o Método **PUT**, observe que será necessário informar o formato em que os dados serão enviados através do Corpo da Requisição. Selecione o formato **JSON**, como indicado na imagem abaixo em vermelho e clique no botão **Create** para concluir.




3. Configure a requisição conforme a imagem abaixo:



4. No item marcado em amarelo na imagem acima, informe o endereço (endpoint) da Requisição. A requisição **Editar Postagem** foi configurada da seguinte maneira:

- A primeira parte do endereço (<http://localhost:8080>) é o endereço do nosso servidor local. Quando a API estiver na nuvem, ele será substituído pelo endereço da aplicação na nuvem.
- A segunda parte do endereço é o **endpoint** configurado na anotação **@RequestMapping**, em nosso caso **/postagens**.

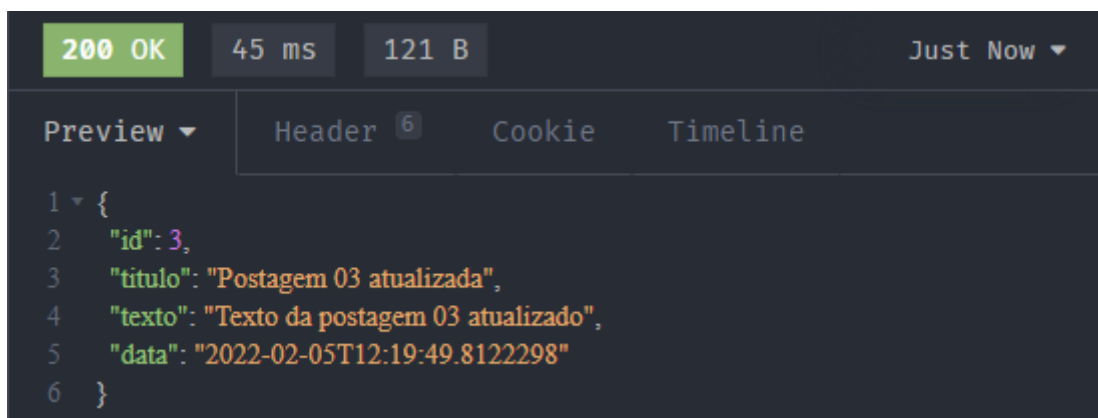
5. Na guia **JSON**, precisamos inserir um **JSON** com os dados que serão inseridos na nova postagem. Lembrando que no padrão JSON: **o texto antes dos 2 pontos (:)** é o **atributo** da Classe e **o texto depois dos 2 pontos (:)** é o **dado** que será cadastrado no atributo. Os atributos são separados por vírgula, como mostra a imagem acima.



ATENÇÃO: Observe que no método PUT é necessário enviar o atributo **id** no JSON para identificar a Postagem que será atualizada. A data não precisa ser enviada, porque ela será atualizada pela própria aplicação.

6. Para testar a requisição, com a aplicação rodando, clique no botão .

7. O resultado da requisição você confere na imagem abaixo:



8. Observe que a aplicação retorna além dos dados que foram atualizados no Banco de dados, ela também retorna um **HTTP Status 200 → OK** (indicado em verde na imagem acima). Este Status indica que a Requisição foi bem sucedida!

