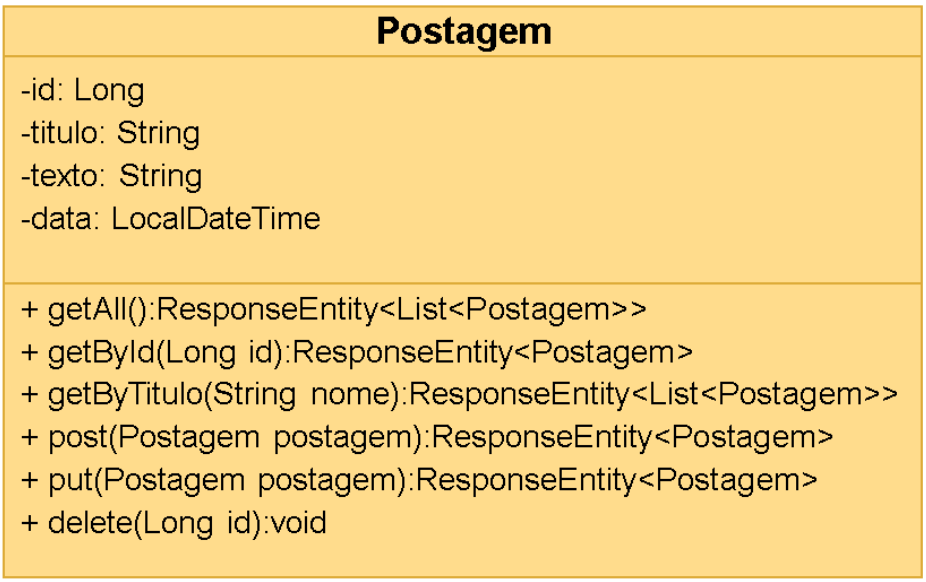
Projeto 02 - Blog Pessoal - CRUD 04

O que veremos por aqui:

1. Criar a Classe PostagemController
2. Criar o método getAll() para listar todas as Postagens

1. O Recurso Postagem

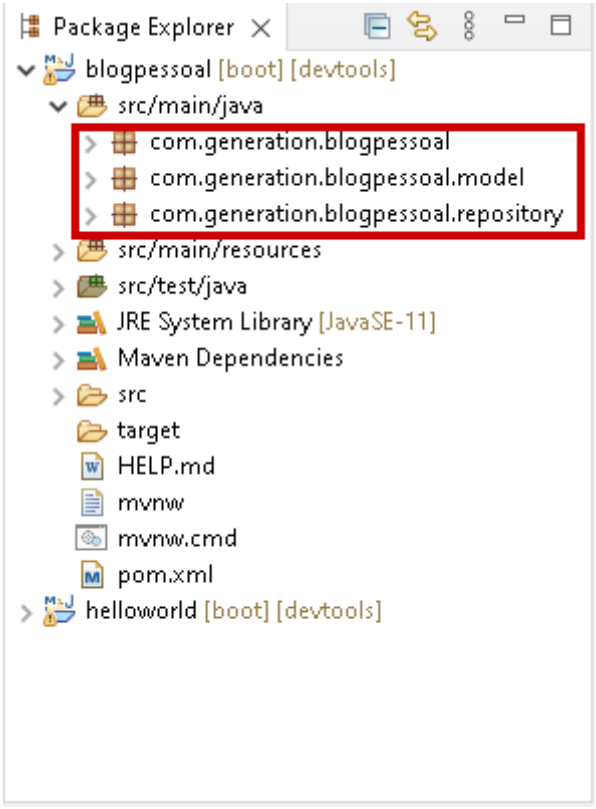
Nas 2 etapas anteriores, começamos a construir o **Recurso Postagem**, a partir da **Classe Model Postagem**, onde implementamos todos os atributos do recurso e geramos a tabela **tb_postagem** dentro do nosso Banco de dados **db_blogpessoal**. Na sequência, implementamos a **Interface PostagemRepository**, que possui todos os métodos necessários para Interagirmos com o Banco de dados. Agora vamos começar a criar os 6 Métodos da Classe Postagem, listados no Diagrama de Classes abaixo, na Classe **PostagemController**.



A Classe **PostagemController** será a Classe Controladora do Recurso Postagem, ou seja, ela irá responder toda e qualquer Requisição (HTTP Request), que for enviada de fora da aplicação para o Recurso Postagem. Dentro desta Classe iremos implementar os métodos do **CRUD (Create, Read, Update e Delete)**, que fazem parte da **Interface JpaRepository** e os Métodos Personalizados (Consultas), que serão assinados dentro da **Interface PostagemRepository**.

🦶 Passo 01 - Criar o Pacote Controller

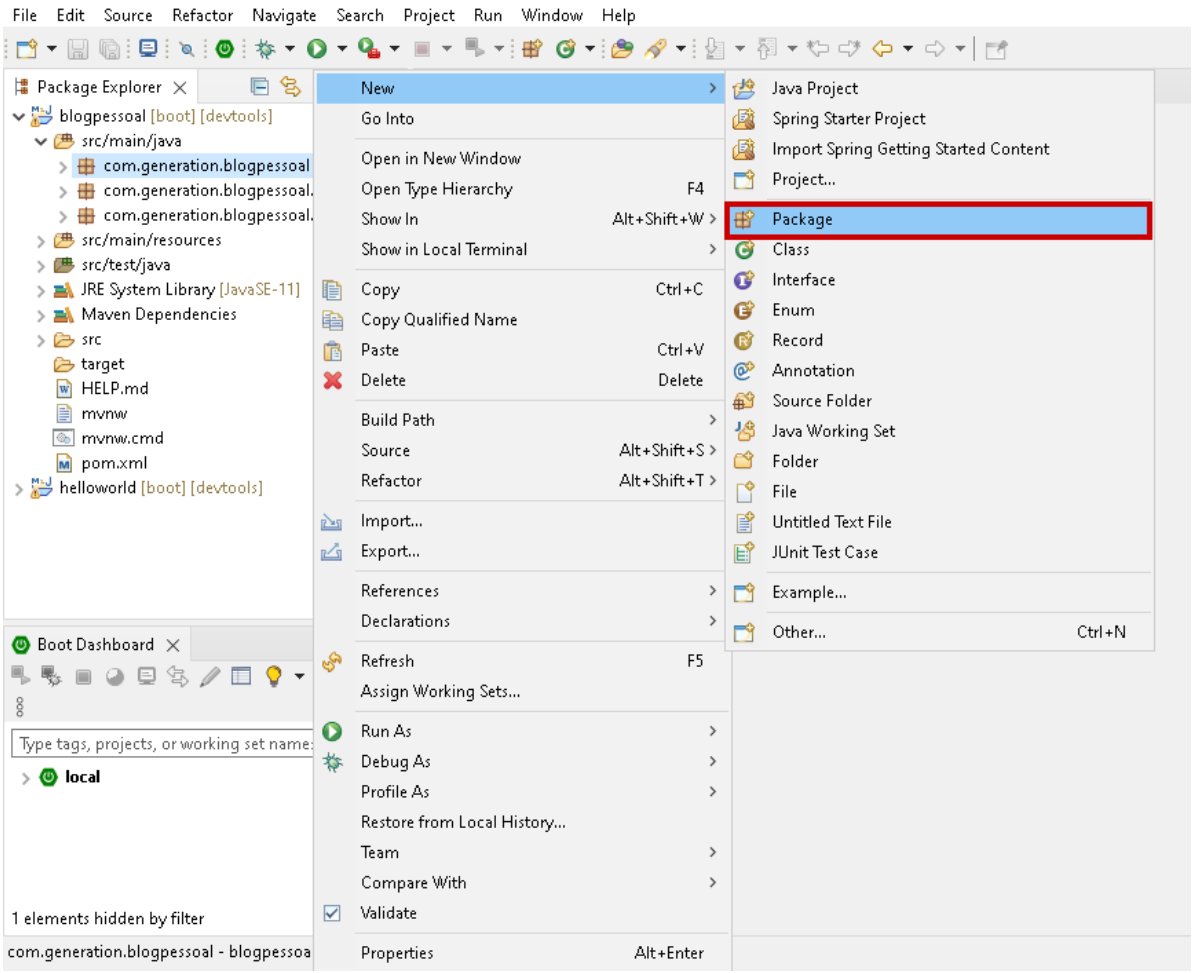
Na Source Folder Principal (**src/main/java**), observe que já foi criado o pacote Principal da nossa aplicação (**com.generation.blogpessoal**), o pacote Model (**com.generation.blogpessoal.model**) e o pacote Repository (**com.generation.blogpessoal.repository**). Na figura abaixo, podemos visualizar os 3 pacotes:



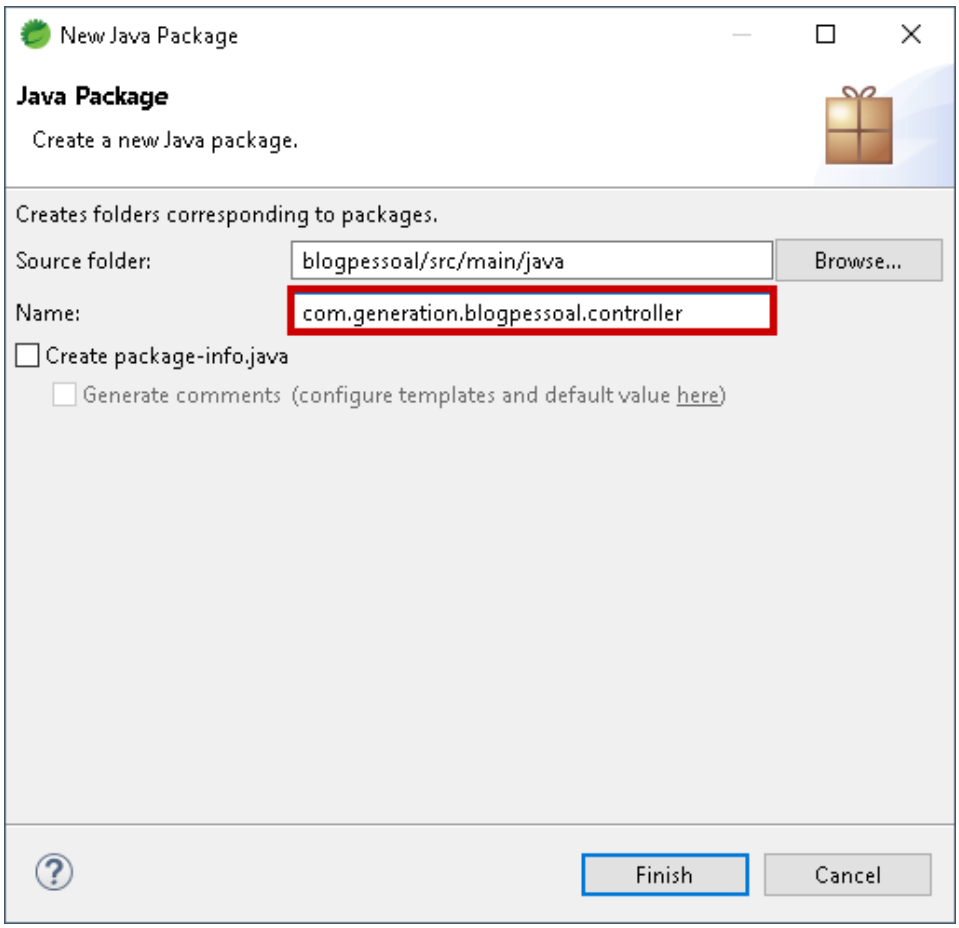
ALERTA DE BSM: ***Mantenha a Atenção aos Detalhes ao criar a Camada Controller. Um erro muito comum é criar o pacote na Source Folder de Testes, ao invés de criar na Source Folder Principal.***

Nesta etapa, vamos criar a **Camada Controller**:

1. No lado esquerdo superior, na Guia **Package explorer**, clique com o botão direito do mouse sobre a Package **com.generation.blogpessoal**, na Source Folder **src/main/java** e clique na opção **New → Package**.

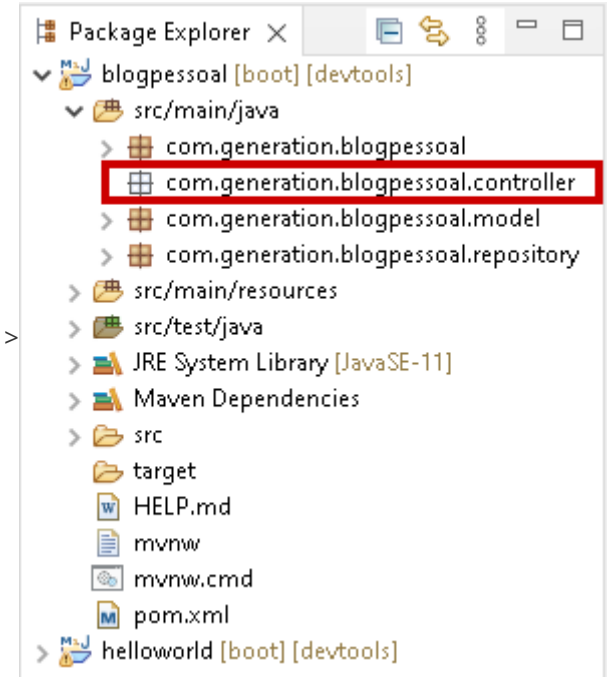


2. Na janela **New Java Package**, no item **Name**, acrescente no final do nome da Package **.controller**, como mostra a figura abaixo:



3. Clique no botão **Finish** para concluir.

Quando você terminar de criar a **Camada Controller**, a sua estrutura de pacotes ficará igual a figura abaixo:

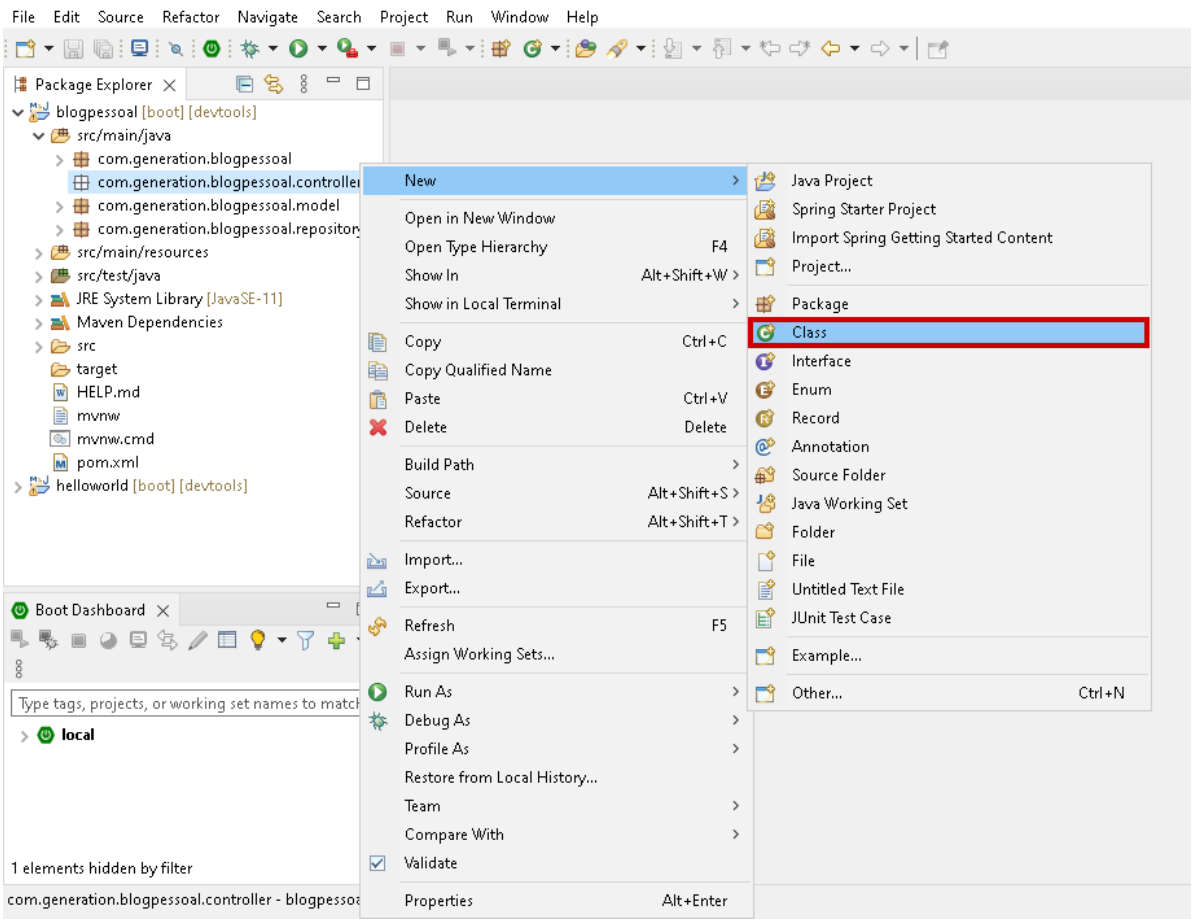


👣 Passo 02 - Criar a Classe PostagemController na Camada Controller

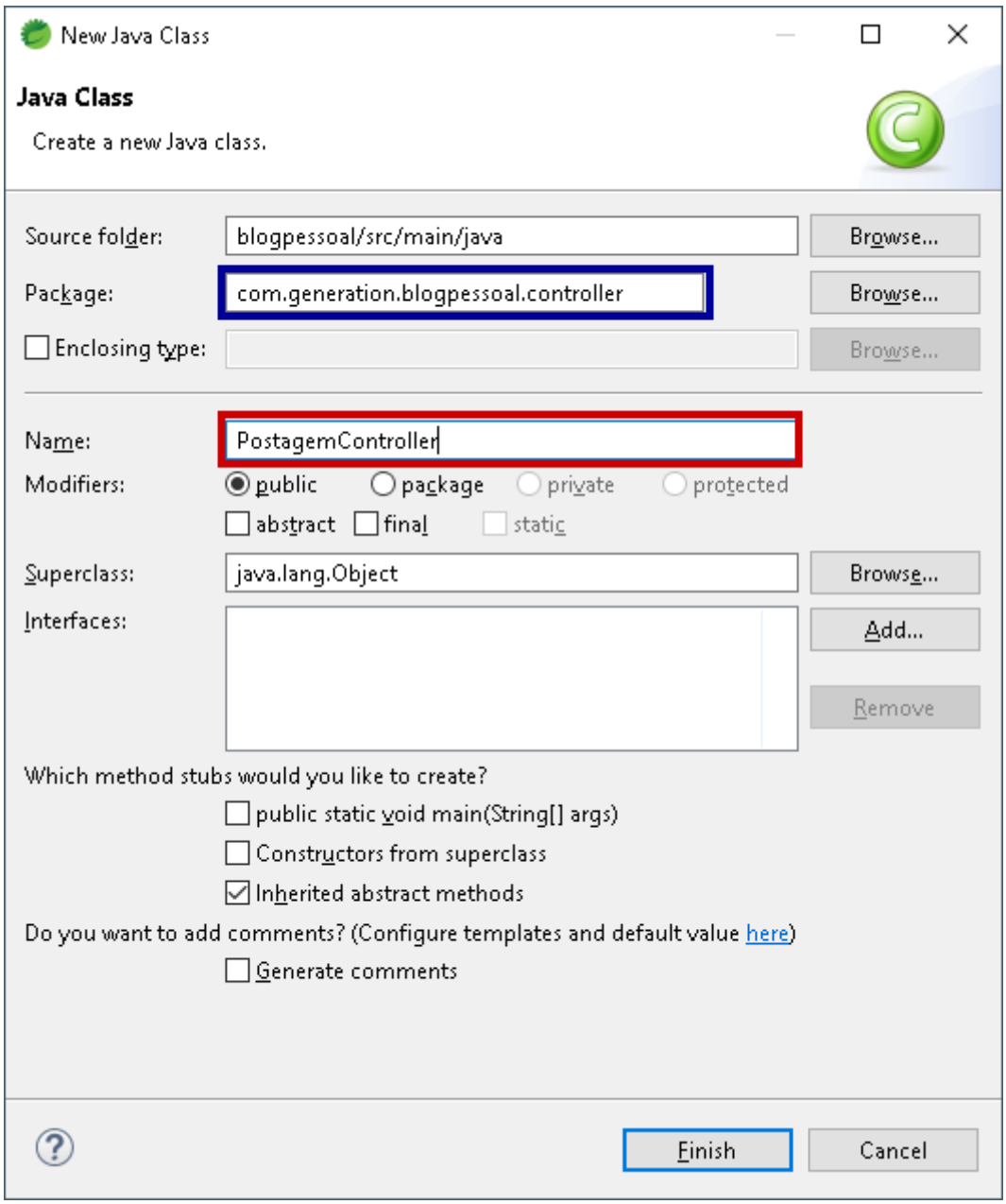
Agora, vamos criar a Classe Controladora que chamaremos de **PostagemController**.

1. Clique com o botão direito do mouse sobre o pacote controller da aplicação (**com.generation.blogpessoal.controller**).

2. Na sequência, clique na opção **New → Class**



3. Na janela **New Java Class**, no item **Name**, digite o nome da Classe (**PostagemController**), como mostra a figura abaixo:



4. Clique no botão **Finish** para concluir.

Agora vamos começar a criar o código da Classe Controladora **PostagemController**:

```
14
15@RestController
16@RequestMapping("/postagens")
17@CrossOrigin(origins = "*", allowedHeaders = "*")
18public class PostagemController {
19
20    @Autowired
21    private PostagemRepository postagemRepository;
22
```

Nesta primeira parte vamos criar a base da Classe Controladora.

Linha 15: a anotação **@RestController** define que a classe é do tipo RestController, que receberá requisições que serão compostas por:

- **URL:** Endereço da requisição (**endpoint**)
- **Verbo:** Define qual método HTTP será acionado na Classe controladora.

- **Corpo da requisição (Request Body):** Objeto que contém os dados que serão persistidos no Banco de dadas. Nem toda a requisição enviará dados no Corpo da Requisição.

Após receber e processar a requisição, a Classe Controladora Responderá a estas requisições com:

- Um **Código de Status HTTP** pertinente a operação que está sendo realizada.
- O resultado do processamento (Objetos de uma Classe, por exemplo) inserido diretamente no corpo da resposta (**Response Body**)

Linha 16: a anotação **@RequestMapping** é usada para mapear as solicitações para os métodos da classe controladora **PostagemController**, ou seja, definir a **URL** (endereço) padrão do Recurso (**/postagens**). Ao digitar a url do servidor seguida da url do Recurso (**http://localhost:8080/postagens**), o Spring envia a requisição para a Classe responsável pelo Recurso associado à este endereço.

Linha 17: a anotação **@CrossOrigin** indica que a classe controladora permitirá o recebimento de requisições realizadas de fora do domínio (localhost e futuramente do heroku quando o deploy for realizado) ao qual ela pertence. Essa anotação é essencial para que o front-end (Angular ou React), tenha acesso à nossa aplicação (O termo técnico é consumir a API). Além de liberar todas as **Origens das requisições** (parâmetro origins), a anotação libera também os **Cabeçalhos das Requisições** (parâmetro allowedHeaders), que em alguns casos trazem informações essenciais para o correto funcionamento da aplicação. Um bom exemplo é o **Token de Segurança**, que veremos mais à frente, que tem a função de liberar o acesso à um endpoint específico. Em produção, recomenda-se substituir o * pelo endereço do deploy do front-end.

Linhas 20 e 21 a anotação **@Autowired** (Injeção de Dependência), é a implementação utilizada pelo Spring Framework para aplicar a Inversão de Controle (IoC) quando for necessário. A Injeção de Dependência define quais Classes serão instanciadas e em quais lugares serão Injetadas quando houver necessidade.

Em nosso exemplo, a Classe Controladora cria um ponto de injeção da **Interface PostagemRepository** e quando houver a necessidade o Spring Framework cria um novo **Objeto da Classe Postagem** a partir da Interface PostagemRepository, permitindo o uso de todos os Métodos da Interface (métodos Padrão ou Personalizados), sem a necessidade de criar Métodos Construtores na Classe Model ou Criar/Instanciar Objetos de forma manual (igual vocês fizeram no Bloco I). Estamos transferindo a responsabilidade desta tarefa para o Spring e desta forma nos preocuparemos apenas com o processamento das Requisições.

Documentação: HTTP Methods Request

Documentação: HTTP Status Code


Documentação: *@RestController*


Documentação: *@RequestMapping*


Documentação: *@CrossOrigin*


Documentação: *@Autowired*


Artigo: *Injeção de Dependência*

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

 Passo 03 - Criar o Método getAll()

Vamos implementar o Método **getAll()** na Classe Postagem Controller, que retornará todos os Objetos da Classe Postagem persistidos no Banco de dados. Traçando um paralelo com o MySQL, seria o equivalente a instrução: `SELECT * FROM tb_postagens;`.

```
22
23      @GetMapping
24      public ResponseEntity<List<Postagem>> getAll(){
25          return ResponseEntity.ok(postagemRepository.findAll());
26      }
27
```

Linha 23: a anotação **@GetMapping** mapeia todas as Requisições **HTTP GET**, enviadas para um endereço específico, chamado **endpoint**, dentro do Recurso Postagem, para um Método específico que responderá a requisição, ou seja, ele indica que o Método getAll(), responderá a todas as requisições do tipo **HTTP GET**, enviadas no endereço http://localhost:8080/postagens/.

ATENÇÃO: ***O Endereço do endpoint será igual ao Endereço do Recurso (@RequestMapping) apenas quando a anotação @GetMapping não possuir um endereço personalizado, como um parâmetro por exemplo. Caso existam dois ou mais métodos do tipo GET será necessário personalizar o endereço de cada Método anotado com @GetMapping.***

Linha 24: O Método **getAll()** será do tipo **ResponseEntity** porquê ele responderá a **Requisição HTTP** (HTTP Request), com uma **Resposta HTTP** (HTTP Response).

✔`<List<Postagem>>`: O Método além de retornar um objeto da **Classe ResponseEntity** (OK →200), no parâmetro body (Corpo da Resposta), será retornado um Objeto da Classe **List (Collection)**, contendo todos os Objetos da Classe **Postagem** persistidos no Banco de dados, na tabela **tb_postagens**. Observe que nesta linha foi utilizado um recurso chamado **Java Generics**, que além de simplificar o retorno do Objeto da Classe **List**, dispensa o uso do **casting** (mudança de tipos). Observe que na definição do Método foram utilizados **os símbolos** , onde **T** é o **Tipo do Objeto** que será retornado no **Corpo da Resposta**.

Linha 25: return ResponseEntity.ok(postagemRepository.findAll()); Executa o método **findAll()** (**Método padrão da Interface JpaRepository**), que retornará **todos os Objetos da Classe Postagem** persistidos no Banco de dados (`<List<Postagem>>`). Como a List sempre será gerada (vazia ou não), o Método sempre retornará o **Status 200 → OK**.


[Documentação: *@GetMapping*]


[Documentação: *ResponseEntity*]


[Documentação: *HttpStatus*]


[Documentação: *.findAll()*]


[Documentação: *Collection List*]

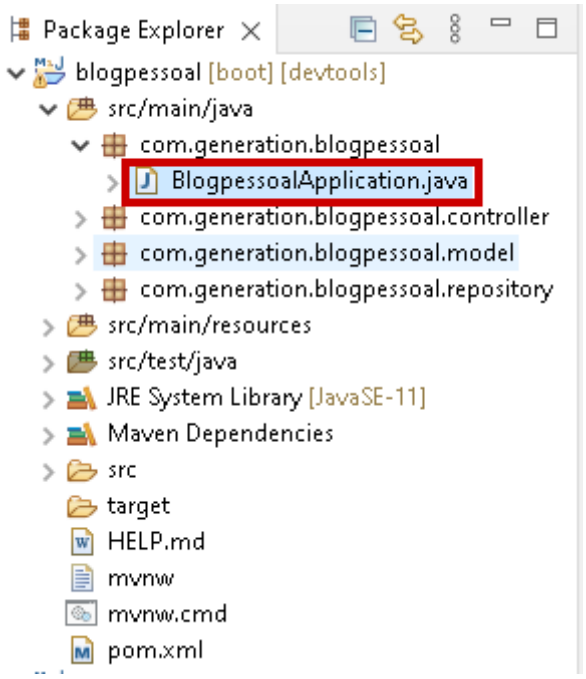

[Documentação: *Java Generics*]

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

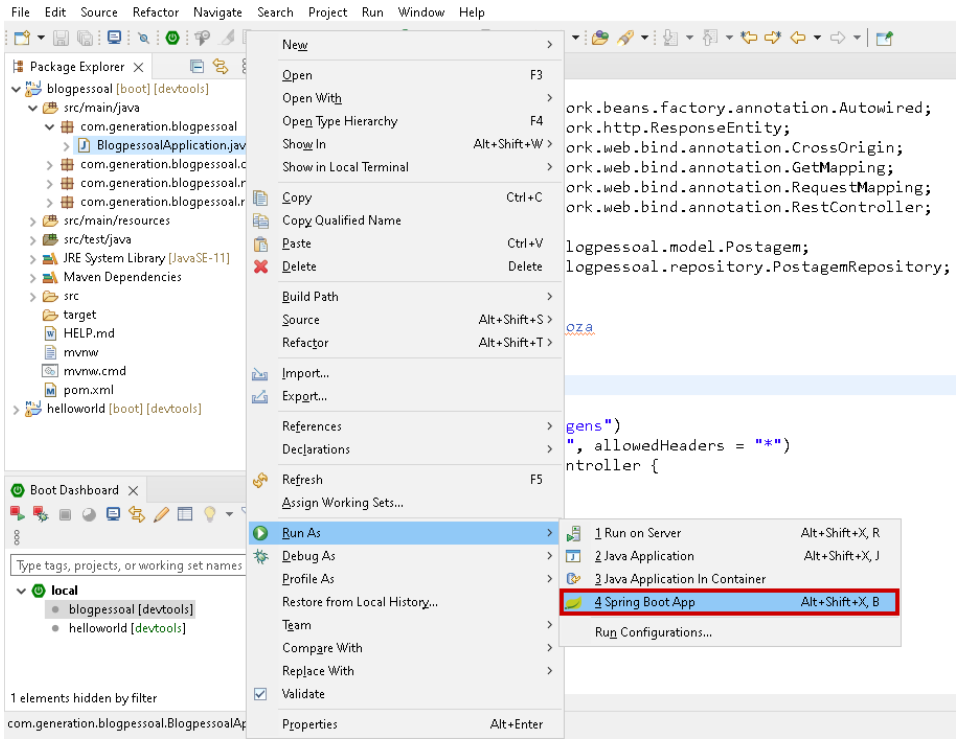

[Código fonte do projeto]

Passo 04 - Executar o projeto

1. No **STS**, na **Package Explorer**, clique na pasta **src/main/java** e na sequência clique no pacote principal **com.generation.blogpessoal**.
2. Clique com o botão direito do mouse sobre o arquivo **BlogpessoalApplication.java**.



3. No menu que será aberto, clique na opção **Run AS → Spring Boot App** como mostra a figura abaixo:



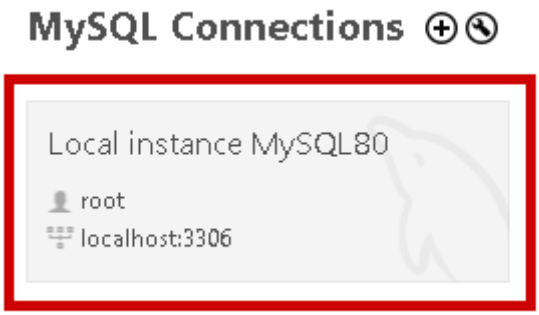
4. Observe que será aberta a janela Console com o log da execução do código. Caso esteja tudo certo, o Console exibirá no final do processamento a mensagem indicando que a **aplicação está em execução** (indicada em vermelho na imagem).

```
INFO 13208 --- [  restartedMain] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect
INFO 13208 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine
INFO 13208 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default
WARN 13208 --- [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database
INFO 13208 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
INFO 13208 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
INFO 13208 --- [  restartedMain] c.g.blogpessoal.BlogpessoalApplication   : Started BlogpessoalApplication in 8.133 seconds (JVM running for S
```

👣 Passo 05 - Inserir dados no Banco de dados

Como não temos o Método de Cadastrar Postagem criado, vamos inserir dois registros no Banco de Dados **db_blogpessoal** na Tabela **tb_postagem** para testarmos o nosso Método getAll().

1. Na Caixa de pesquisas, localize o **MySQL**, e clique no **MySQL Workbench**.
2. No **MySQL Workbench**, Clique sobre a **Conexão Local instance MySQL80**



3. Caso seja solicitada a senha, **digite a senha do usuário root** e marque a opção **Save password in vault** para gravar a senha e não perguntar novamente.

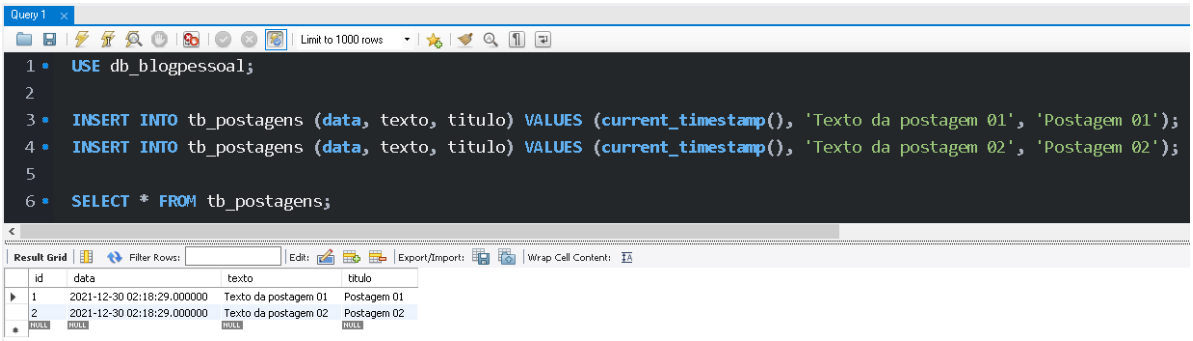
4. Será aberta a janela principal do **MySQL Workbench**.
5. Na janela **Query 1**, insira o código abaixo:

```
USE db_blogpessoal;

INSERT INTO tb_postagens (data, texto, titulo)
VALUES (current_timestamp(), 'Texto da postagem 01', 'Postagem 01');
INSERT INTO tb_postagens (data, texto, titulo)
VALUES (current_timestamp(), 'Texto da postagem 02', 'Postagem 02');

SELECT * FROM tb_postagens;
```

6. Clique no primeiro raio  para inserir os dados no Banco de dados **db_blogpessoal**.



7. A imagem acima mostra que os dados foram inseridos com sucesso!

Passo 06 - Testar no Insomnia

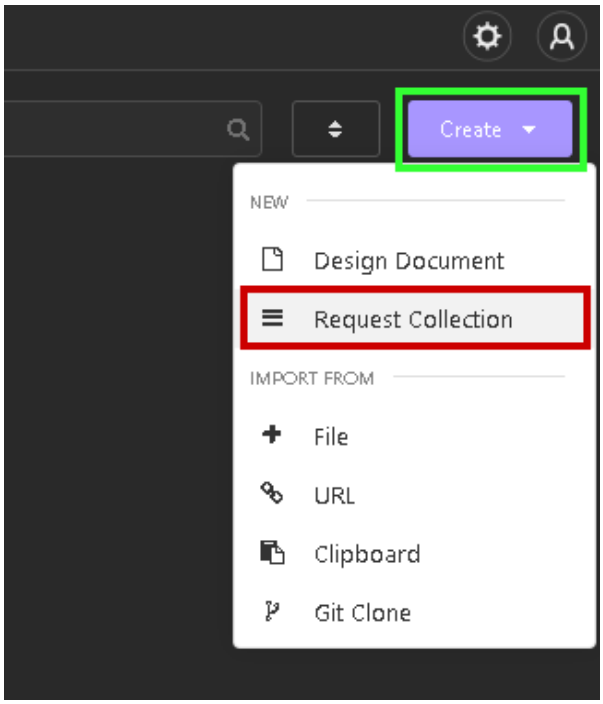
Para testar a aplicação, utilizaremos o **Insomnia**.

[Site Oficial do Insomnia]

Para testar a nossa aplicação, vamos criar uma **Collection** para guardar todas as nossas Requisições do Projeto Blog Pessoal. Na sequência vamos criar dentro da Collection uma pasta chamada **Postagem** para guardar todas as requisições do Recurso Postagem. Para concluir, vamos criar uma requisição do tipo **GET**, dentro da pasta Postagem, para testar o nosso Método **getAll()**.

6.1. Criando a Collection Blog Pessoal

1. Na janela principal do Insomnia, clique no botão **Create** e clique na opção **Request Collection**.

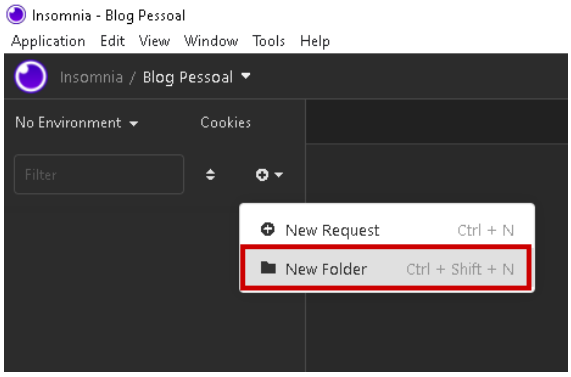


2. Na janela que será aberta, informe o nome da Collection (**Blog Pessoal**) e clique no botão **Create** para concluir.

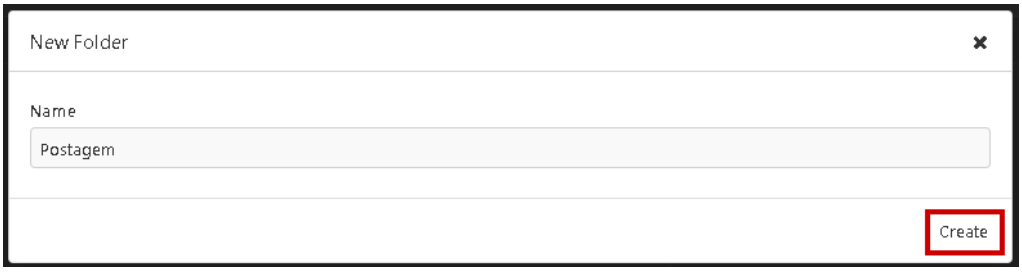
6.2. Criando a Pasta Postagem

Vamos criar dentro da **Collection Blog Pessoal** a **Pasta Postagem**, que guardará todas as requisições do **Recurso Postagem**.

1. Na **Collection Blog Pessoal**, clique no botão ⊕. No menu que será aberto, clique na opção **New Folder**.



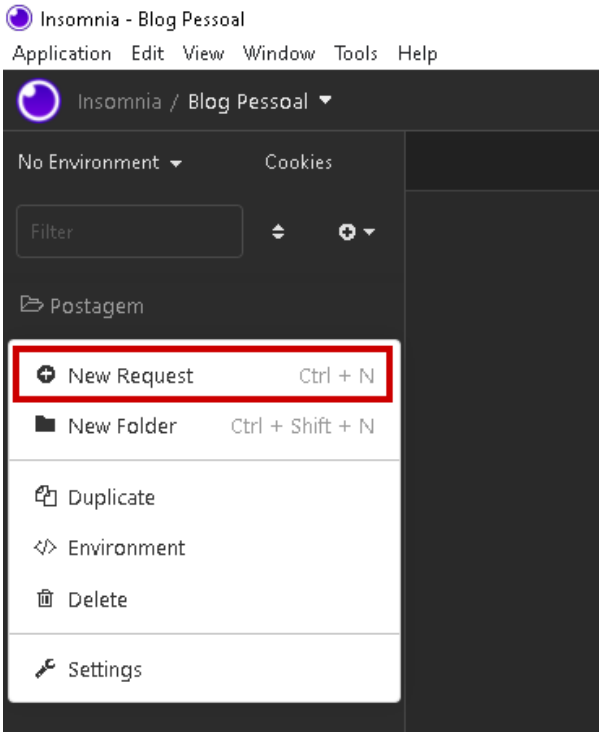
2. Na janela que será aberta, informe o nome da pasta (**Postagem**) e clique no botão **Create** para concluir.



6.3. Criando a Request - Consultar todas as postagens - findAll()

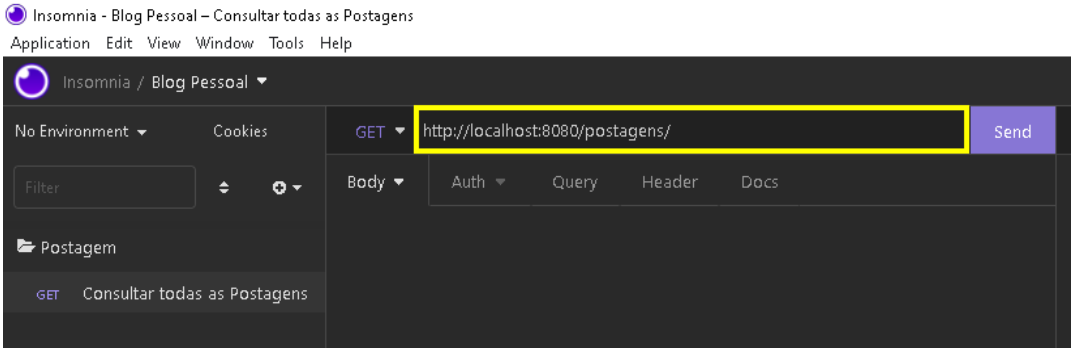
Agora vamos criar a Requisição para o **Método getAll()**:

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 (**GET**), indicado na imagem na cor azul. Clique no botão **Create** para concluir.

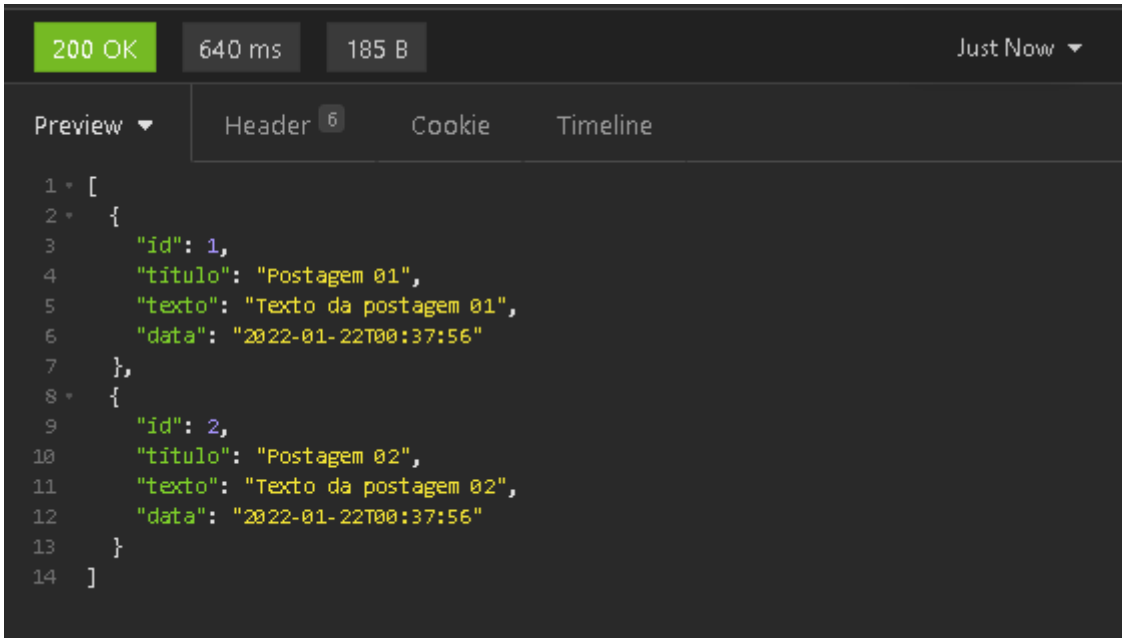
3. Configure a requisição conforme a imagem abaixo:



4. No item marcado em amarelo na imagem acima, informe o endereço (endpoint)) da Requisição. A requisição **Consultar Todas as postagens** 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 (http://nomedaaplicacao.herokuapp.com).
- A segunda parte do endereço é o **endpoint** configurado na anotação ***@RequestMapping***, em nosso caso **/postagens**.

5. Para testar a requisição, com a aplicação rodando, clique no botão Send.
6. O resultado da requisição você confere na imagem abaixo:



7. Observe que a aplicação além de exibir os dados de todos os Objetos da Classe Postagem persistidos no Banco de dados, no Corpo da Resposta, ela também retornará um **HTTP**

Status 200 → OK (indicado em verde na imagem acima), informando que a Requisição foi bem sucedida!

[Site Oficial do JSON]