

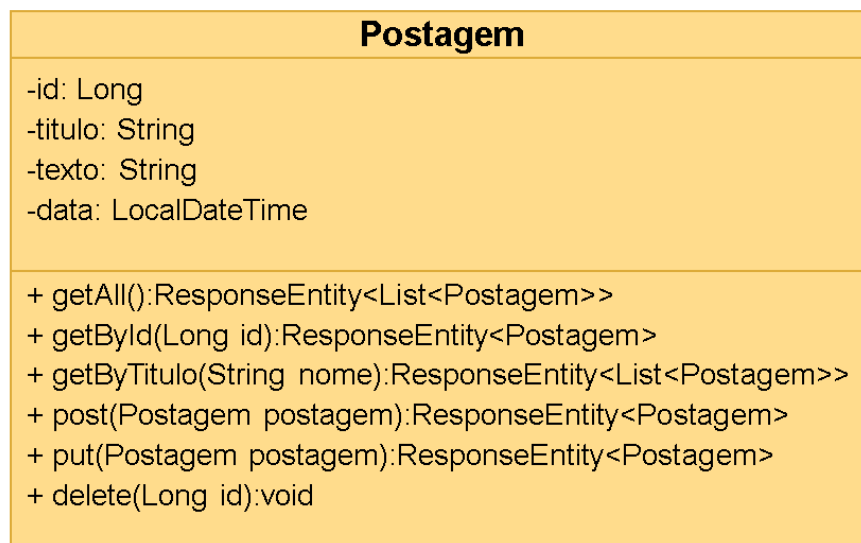
Projeto 02 - Blog Pessoal - CRUD 02

O que veremos por aqui:

1. Apresentação do Recurso Postagem
2. Criar a estrutura de Pacotes
3. Criar a Classe Model Postagem

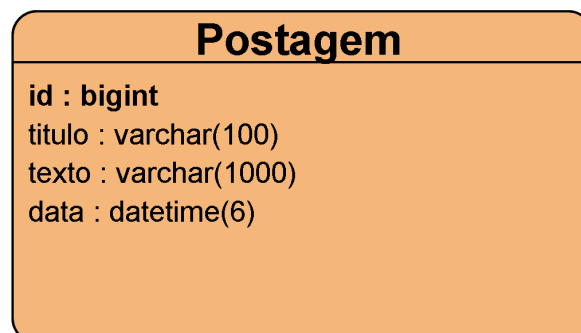
1. O Recurso Postagem

Nesta etapa vamos começar a construir o Recurso Postagem. Veja o Diagrama de Classes abaixo:



Primeiro vamos construir a **Classe Postagem**, que servirá de modelo para construir a tabela **tb_postagens** (Entidade) dentro do nosso Banco de dados **db_blogpessoal**. Os campos (Atributos) da tabela serão os mesmos que estão definidos no Diagrama de Classes acima. Na próxima etapa vamos construir a **Interface PostagemRepository**, que irá nos auxiliar na interação com o Banco de dados. Os métodos descritos no Diagrama de Classes serão implementados na etapa seguinte, na **Classe PostagemController**.

Depois de criar a Classe Model Postagem, vamos executar o projeto Blog Pessoal no STS. Após a execução veremos que será criado no MySQL o Banco de dados **db_blogpessoal** e a tabela **tb_postagens**. Veja abaixo como ficará a estrutura da nossa tabela através do **Diagrama de Entidade e Relacionamentos (DER)**:



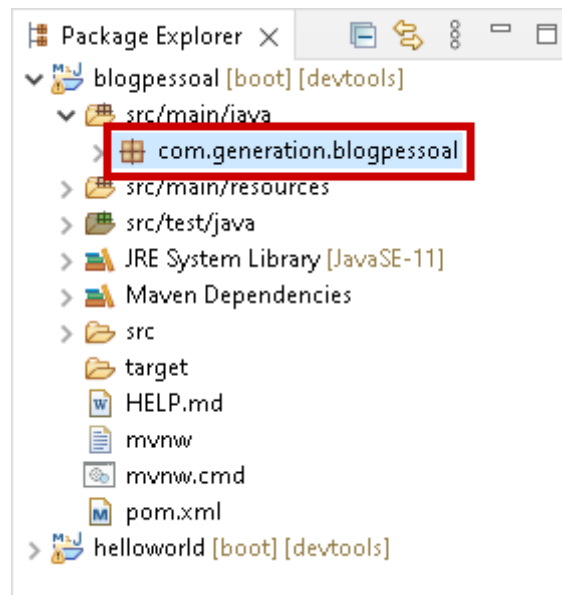
O **Dicionário de dados** da nossa tabela **tb_postagens** será o seguinte:

Atributo	Tipo de dado	Descrição	Chave
id	bigint	Identificador único	PK
titulo	varchar(100)	Título da postagem	
texto	varchar(1000)	Conteúdo da postagem	
data	datetime(6)	Data e hora da publicação/atualização da postagem	



Passo 01 - Criar o Pacote Model

Na Source Folder Principal (**src/main/java**), observe que foi criado o pacote principal da nossa aplicação (**com.generation.blogpessoal**), onde todo o nosso código será desenvolvido. Na figura abaixo, podemos visualizar o pacote:

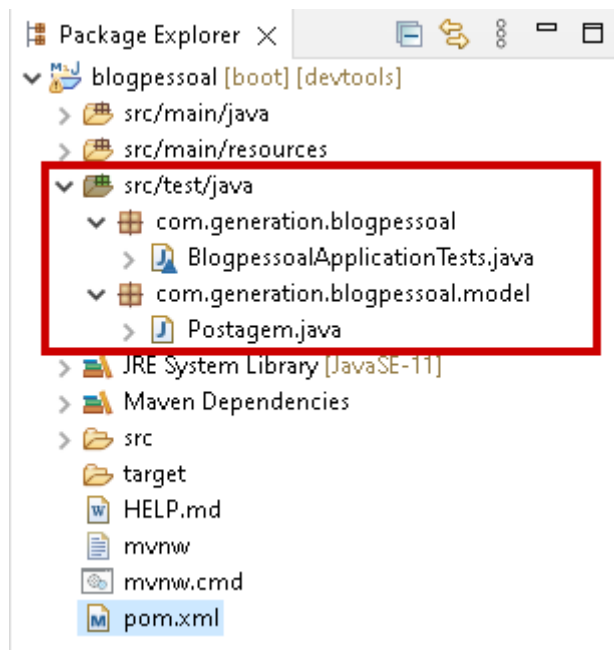


Vamos criar dentro do pacote principal, sub pacotes que chamaremos de **Camadas**. Dentro destes sub pacotes, iremos criar as Classes e Interfaces da nossa aplicação, seguindo o modelo MVC. Para criarmos o Recurso Postagem, vamos precisar de 3 Camadas:

Camada	Descrição
Model	Camada responsável pela abstração dos nossos Objetos em registros das nossas tabelas, que serão geradas no Banco de dados. As Classes criadas nesta camada representam os objetos que serão persistidos no Banco de dados.
Repository	Camada responsável por implementar as Interfaces, que contém diversos métodos pré-implementados para a manipulação de dados de uma entidade, como métodos para salvar, deletar, listar e recuperar dados da classe. Para criar estas Interfaces basta Herdar (extends) a Interface JpaRepository.
Controller	Camada responsável por receber todas as Requisições HTTP (HTTP Request), enviadas por um Cliente HTTP (Postman ou o Front-end da API), para a nossa aplicação e responder (HTTP Response) as requisições de acordo com o resultado do processamento da requisição no Back-end.

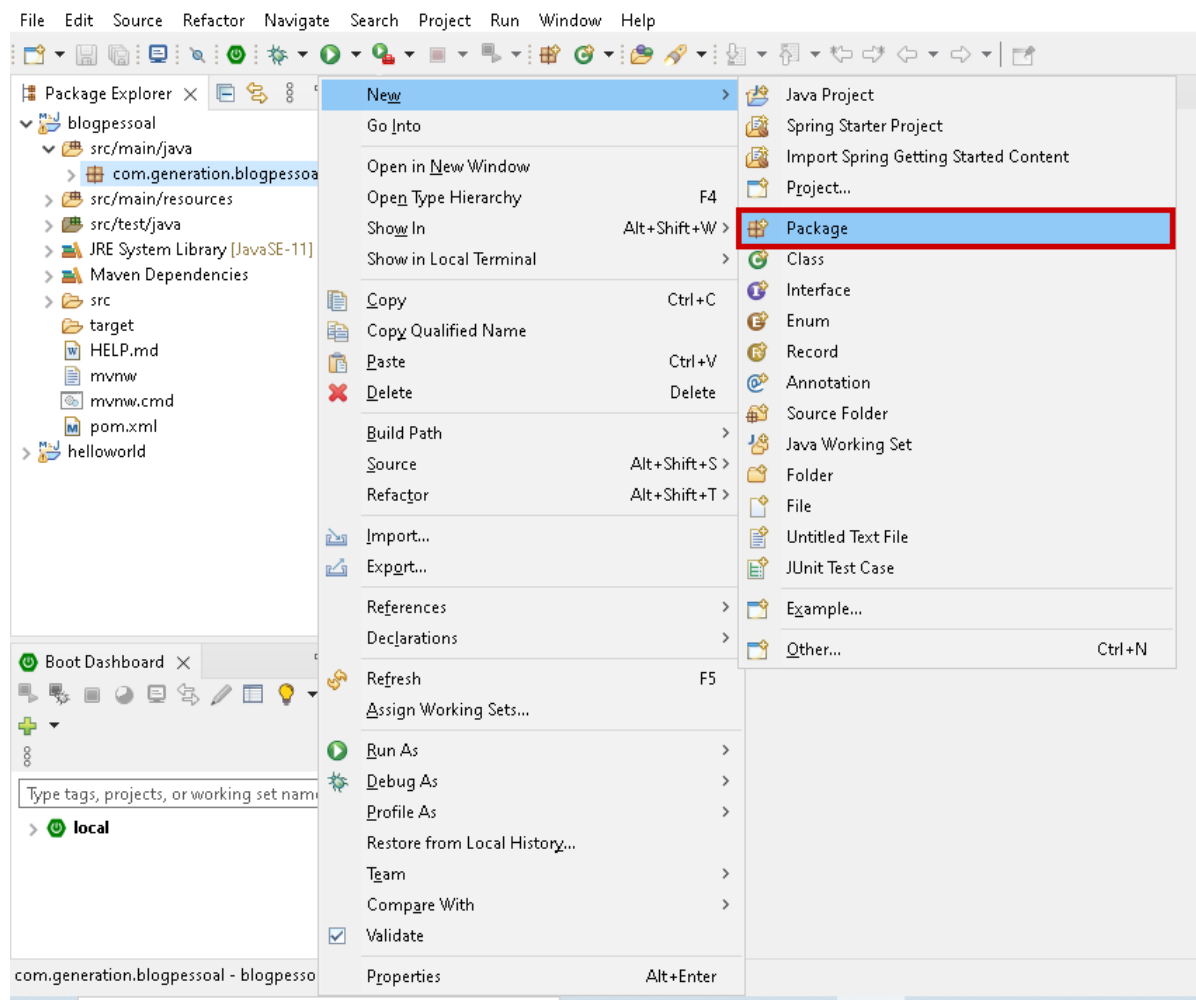


ALERTA DE BSM: Mantenha a Atenção aos Detalhes ao criar a Camada Model. Um erro muito comum é criar o pacote na Source Folder de Testes (imagem abaixo), ao invés de criar na Source Folder Principal.

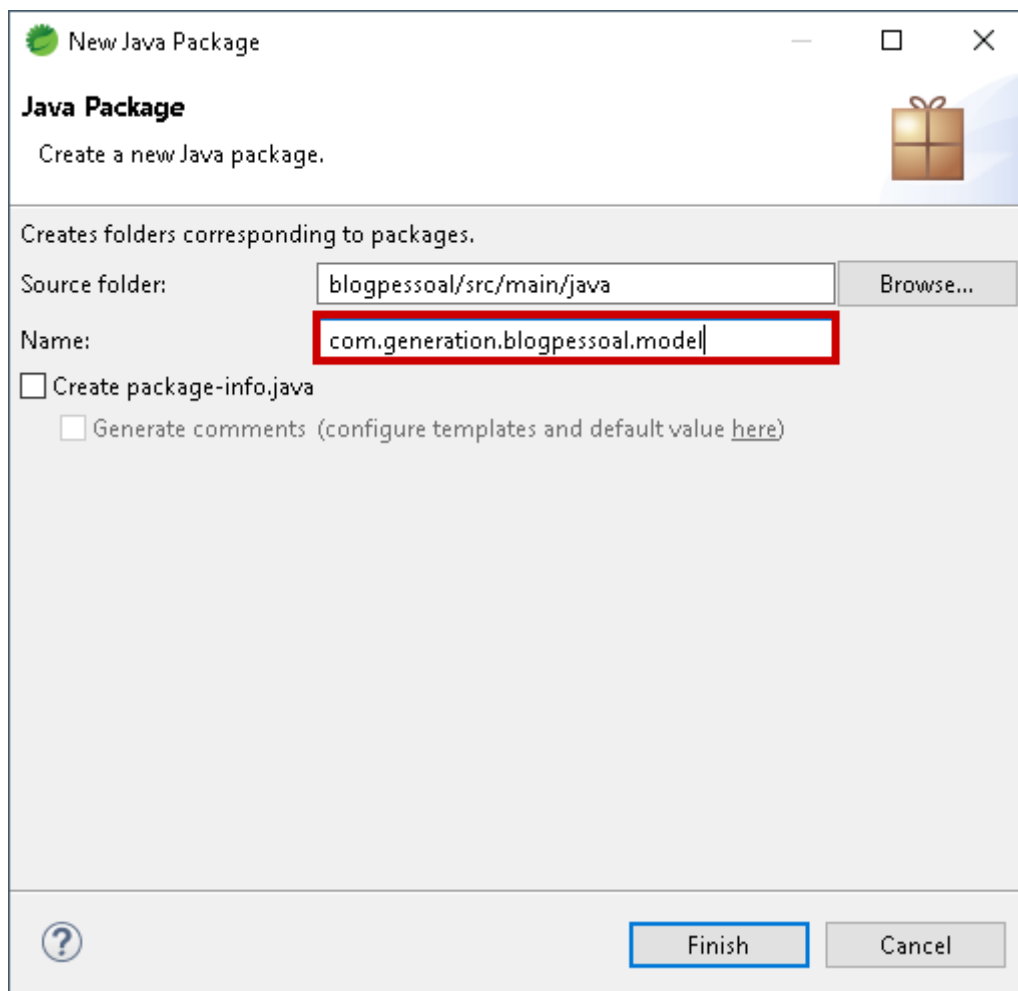


Vamos começar criando a Camada Model:

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**.

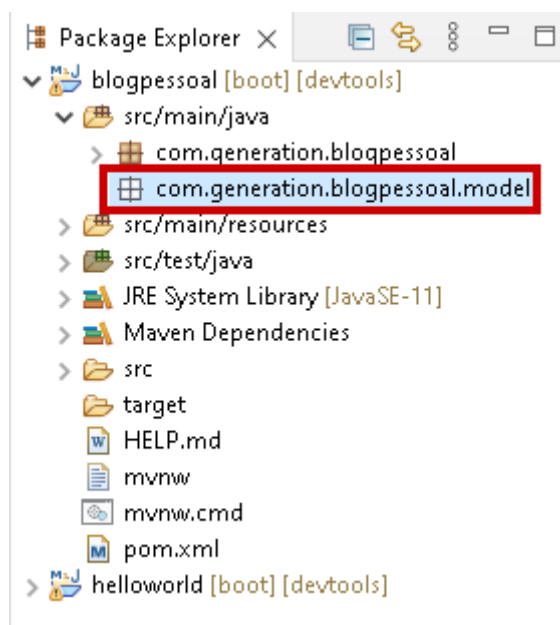


2. Na janela **New Java Package**, no item **Name**, acrescente no final do nome da Package **.model**, como mostra a figura abaixo:



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

A estrutura de pacotes da aplicação ficará igual a figura abaixo:



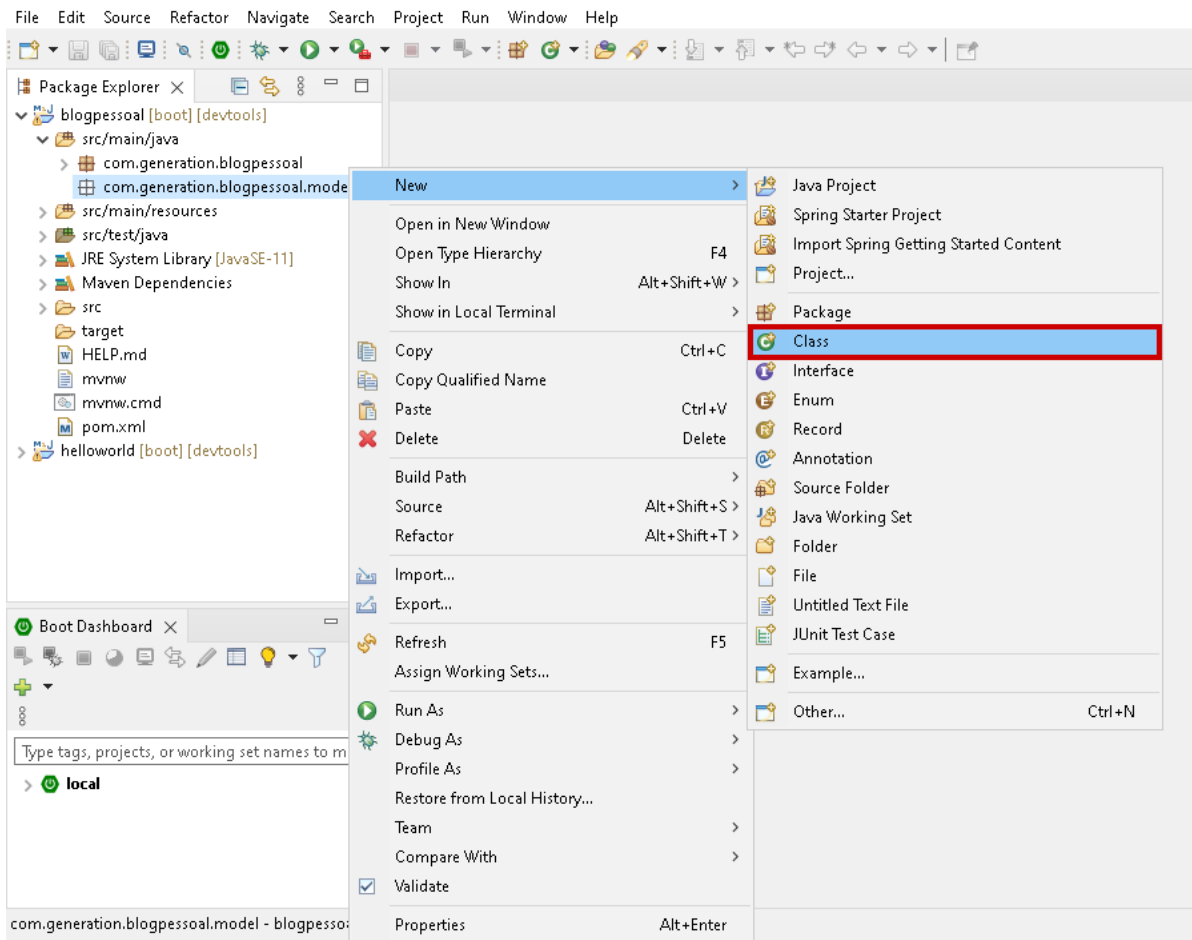
Passo 02 - Criar a Classe Postagem na Camada Model

Agora vamos criar a Classe Model que chamaremos de **Postagem**.

1. Clique com o botão direito do mouse sobre o **Pacote Model** (**com.generation.blogpessoal.model**), na Source Folder Principal (**src/main/java**), como

mostra a figura abaixo:

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



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

New Java Class

Java Class
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

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

Agora vamos criar o código da **Classe Model Postagem**, como mostra a figura abaixo:

```

14
15 @Entity
16 @Table(name = "tb_postagens")
17 public class Postagem {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     private Long id;
22
23     @NotBlank(message = "O atributo título é Obrigatório!")
24     @Size(min = 5, max = 100, message = "O atributo título deve conter no mínimo 05 e no máximo 100 caracteres")
25     private String titulo;
26
27     @NotBlank(message = "O atributo texto é Obrigatório!")
28     @Size(min = 10, max = 1000, message = "O atributo texto deve conter no mínimo 10 e no máximo 1000 caracteres")
29     private String texto;
30
31     @UpdateTimeStamp
32     private LocalDateTime data;
33

```

Vamos analisar o código:

Linha 15: A Anotação (Annotation) **@Entity** indica que esta classe define uma entidade, ou seja, ela será utilizada para gerar uma tabela no Banco de dados da aplicação.

Linha 16: A Anotação **@Table** indica o nome da Tabela no Banco de dados. Caso esta anotação não seja declarada, o Banco de dados criará a tabela com o mesmo nome da Classe Model (Postagem). Observe que o nome da Tabela segue o padrão **tb_nome-da-tabela** (tb_postagens). O prefixo **tb** indica que se trata de uma Table (Tabela). O nome da Tabela é recomendado que seja **o mesmo da Classe Model** (postagem), em **letras minúsculas, sem espaços em branco ou caracteres especiais e acentos**.

Nas Linhas 21, 25, 29 e 32 foram criados os Atributos do Objeto Postagem, que foram definidos no Diagrama de Classes acima. Veja na tabela abaixo a conversão de **Tipo de dados Java** → **MySQL**

Atributo	Tipo de dado Java	Tipo de dado MySQL
id	<u>Long</u>	bigint
titulo	<u>String</u>	varchar(100)
texto	<u>String</u>	varchar(1000)
data	<u>LocalDateTime</u>	data

Observe que acima de cada atributo foram adicionadas algumas Anotações. Estas anotações tem a função de configurar os parâmetros do Banco de dados e criar validações para os dados que serão inseridos no objeto (tamanho, formato e etc)

Linha 19: A Anotação **@Id** inidica que o atributo anotado será a **Chave Primária** (Primary Key - PK) da Tabela **tb_postagens**.

Linha 20: A Anotação **@GeneratedValue** indica que a **Chave Primária** será gerada pelo Spring Data JPA. O parâmetro **strategy** indica de que forma esta **Chave Primária** será gerada. A Estratégia **GenerationType.IDENTITY** indica que a Chave Primária será gerada pelo Banco de dados através da opção **auto-incremento** (auto-increment), que gera uma sequência numérica iniciando em 1.



ATENÇÃO: Não confundir o auto-incremento do Banco de Dados que inicia em 1 com o índice de um Array (Vetor ou Matriz) que inicia em 0.

Nas linhas 23 e 27: A anotação **@NotBlank** não permite que o atributo seja **Nulo ou contenha apenas Espaços em branco**. Você pode configurar uma mensagem para o usuário através do atributo **message**.

Nas linhas 24 e 28: A anotação **@Size** define o valor **Mínimo (min)** e **Máximo (max)** de Caracteres do atributo. Não é obrigatório configurar os 2 parâmetros. Como o parâmetro **max** foi configurado, observe que o mesmo valor informado será inserido na definição dos atributos **titulo** (**varchar(100)**) e **texto** (**varchar(1000)**) na tabela **tb_postagens** no Banco de dados. Você pode configurar uma mensagem para o usuário através do atributo **message**.

Na linha 31: A anotação **@UpdateTimestamp** configura o atributo **data** como **Timestamp**, ou seja, o Spring se encarregará de obter a data do Sistema Operacional e inserir no atributo **data** toda vez que um Objeto da Classe Postagem for criado ou atualizado.



[Documentação: @Entity](#)

[Documentação: @Table](#)

[Documentação: @Id](#)

[Documentação: @GeneratedValue](#)

[Documentação: @NotBlank e @Size](#)

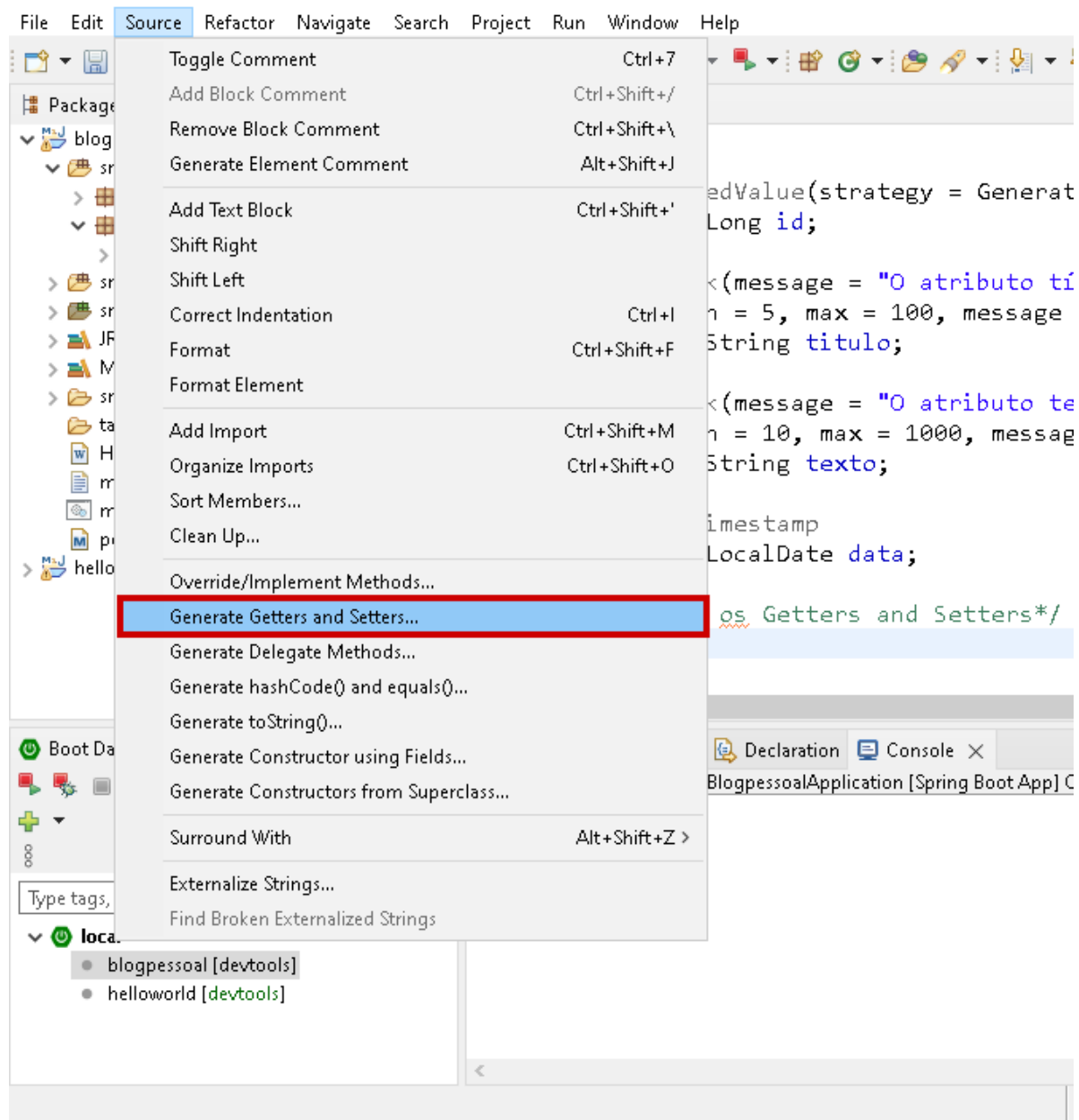
[Documentação: @UpdateTimestamp](#)



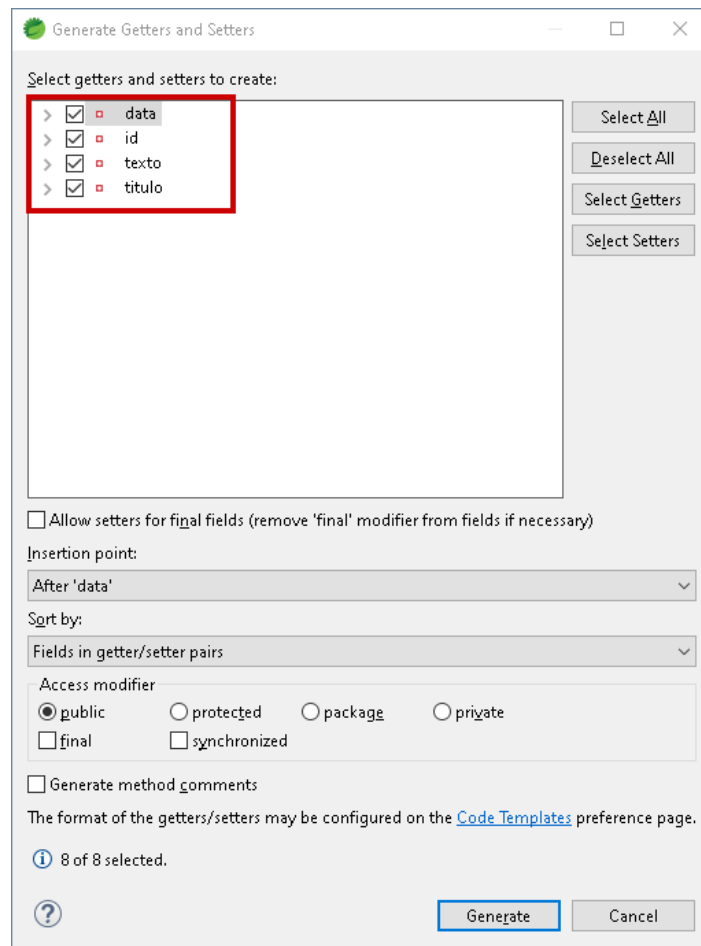
Passo 03 - Métodos Get e Set

Depois de criarmos os atributos, precisamos criar os **Métodos Get e Set** para todos os atributos da Classe. O Método Construtor não será necessário porque o Spring utiliza um recurso chamado **Injeção de Dependência** (veremos na Classe PostagemController).

1. Posicione o cursor do mouse no ponto onde será criado os métodos Get e Set.
2. No menu **Source**, clique na opção **Generate Getters and Setters...**



3. Na tela **Generate Getters and Setters**, Clique no botão **Select All** para selecionar todos os atributos e clique no botão **Generate**.



4. A geração dos Métodos ficará igual a imagem abaixo:

```

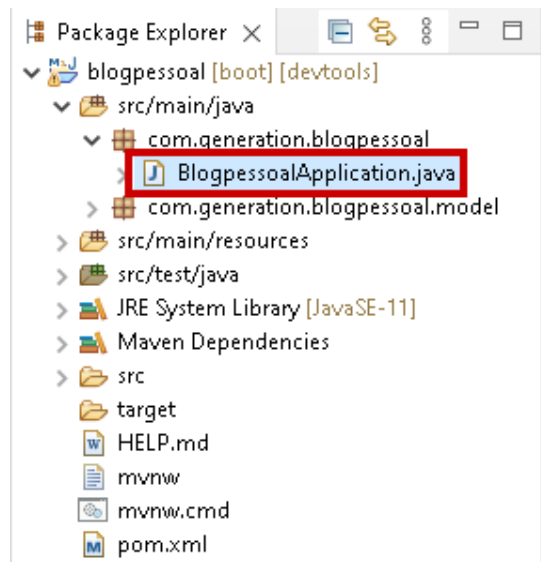
33
34  /*Insira os Getters and Setters*/
35
36  public Long getId() {
37      return this.id;
38  }
39
40  public void setId(Long id) {
41      this.id = id;
42  }
43
44  public String getTitulo() {
45      return this.titulo;
46  }
47
48  public void setTitulo(String titulo) {
49      this.titulo = titulo;
50  }
51
52  public String getTexto() {
53      return this.texto;
54  }
55
56  public void setTexto(String texto) {
57      this.texto = texto;
58  }
59
60  public LocalDateTime getData() {
61      return this.data;
62  }
63
64  public void setData(LocalDateTime data) {
65      this.data = data;
66  }

```

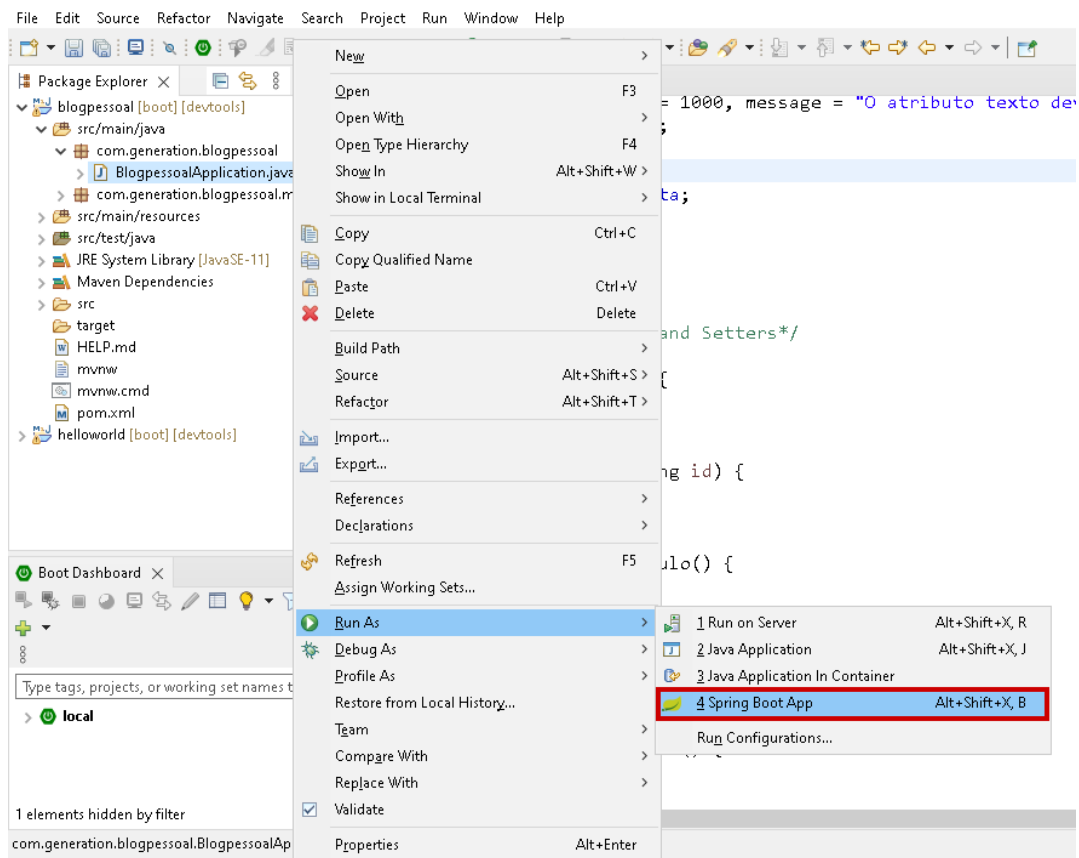


Passo 04 - Executar o projeto

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**.



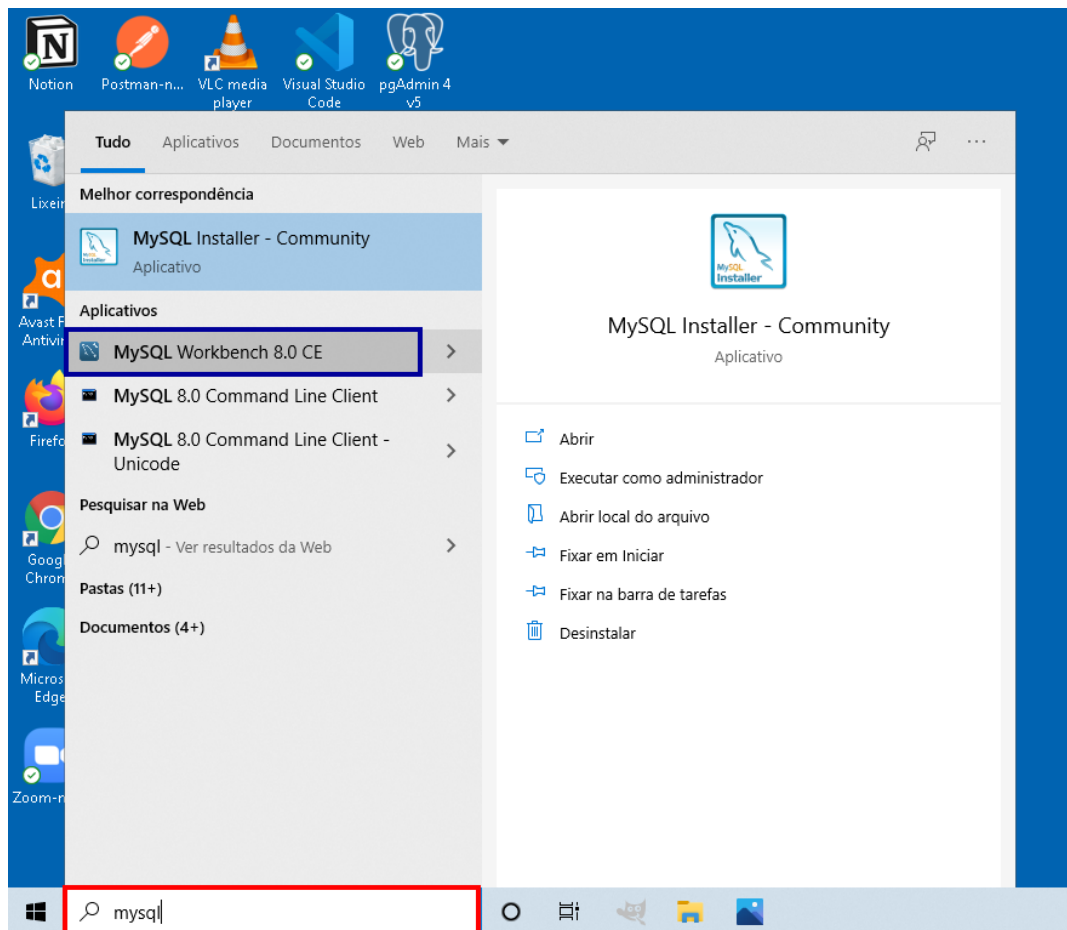
3. No menu que será aberto, clique na opção **Run AS → Spring Boot App** como mostra a figura abaixo:



Passo 05 - Checando o Banco de Dados

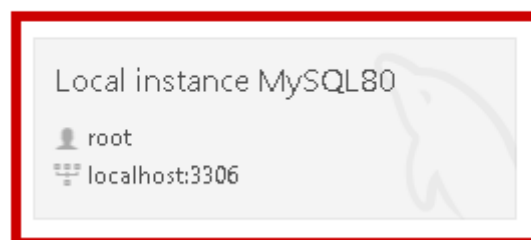
Vamos checar se o Banco de Dados **db_blogpessoal** e a Tabela **tb_postagens** foram criados no MySQL.

1. Na Caixa de pesquisas, localize o **MySQL** (Marcado em vermelho na imagem), e clique no **MySQL Workbench** (Marcado em azul na imagem).

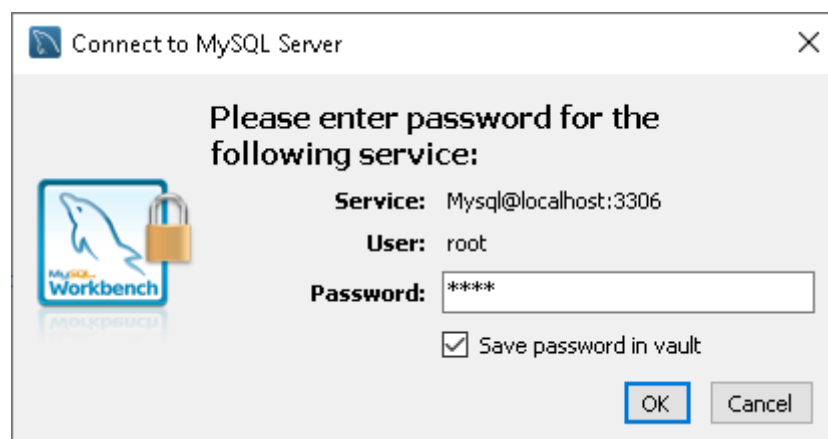


2. No **MySQL Workbench**, Clique sobre a **Conexão Local instance MySQL80**

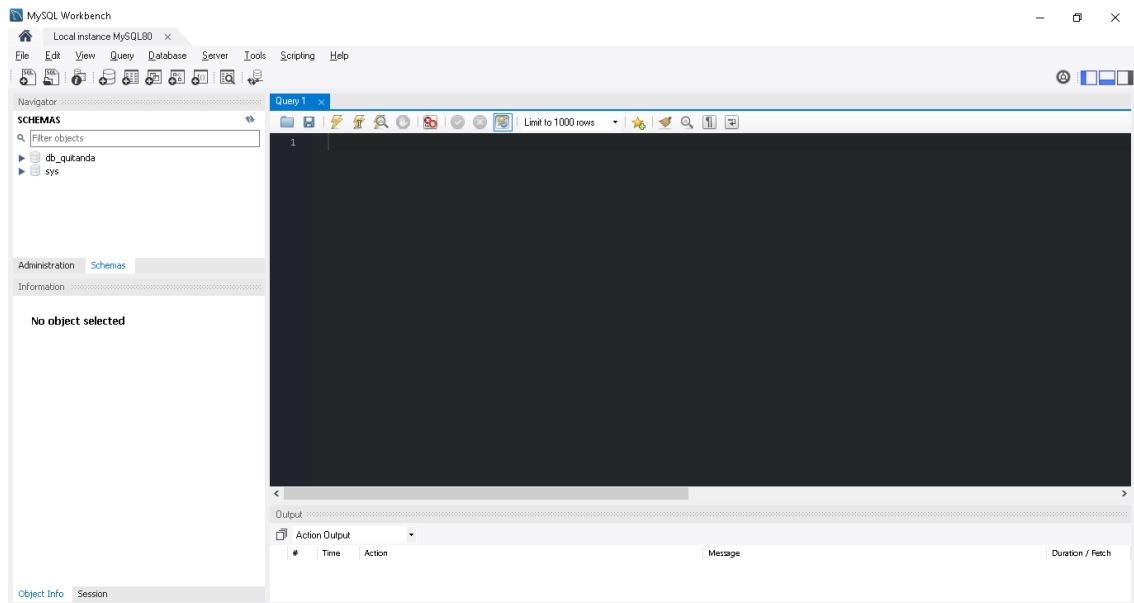
MySQL Connections + ⓘ




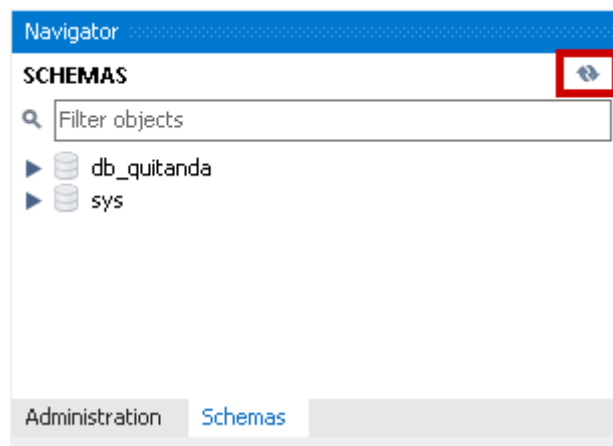
3. Caso seja solicitada a senha, **digite a senha do usuário root** e marque a opção **Save password in vault** para gravar a senha e não perguntar novamente.



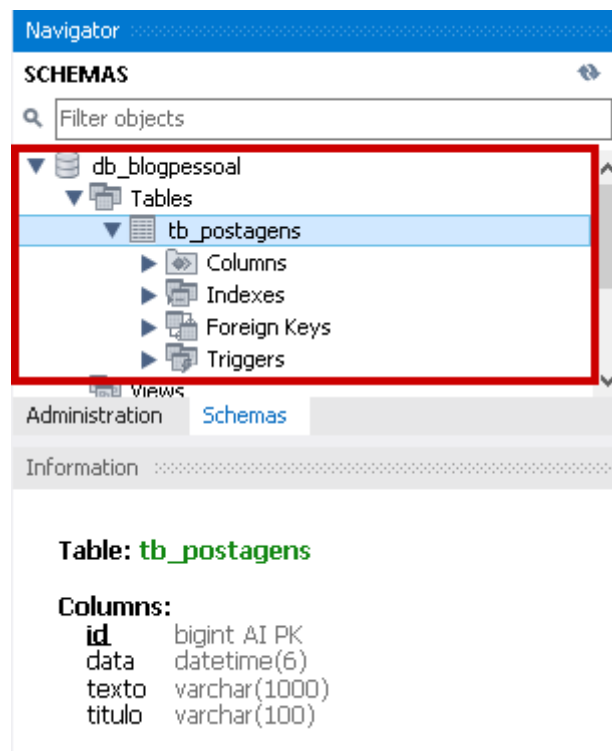
4. Será aberta a janela principal do **MySQL Workbench**.



5. Na janela **Schemas**, Clique no botão  (Atualizar)



6. Verifique se o Banco de dados **db_blogpessoal** e se a tabela **tb_postagens** foram criados, como mostra a figura abaixo:



[Código fonte do Projeto](#)

Anexo I - Estratégias para geração da Chave Primária

Estratégia	Descrição
GenerationType.AUTO	Valor padrão, deixa com o provedor de persistência a escolha da estratégia mais adequada de acordo com o Banco de dados.
GenerationType.IDENTITY	Informamos ao provedor de persistência que os valores a serem atribuídos ao identificador único serão gerados pela coluna de auto incremento do banco de dados. Assim, um valor para o identificador é gerado para cada registro inserido no banco. Alguns bancos de dados podem não suportar essa opção.
GenerationType.SEQUENCE	Informamos ao provedor de persistência que os valores serão gerados a partir de uma sequence. Caso não seja especificado um nome para a sequence, será utilizada uma sequence padrão, a qual será global, para todas as entidades. Caso uma sequence seja especificada, o provedor passará a adotar essa sequence para criação das chaves primárias. Alguns bancos de dados podem não suportar essa opção, como o MySQL por exemplo.
GenerationType.TABLE	Com a opção TABLE é necessário criar uma tabela para gerenciar as chaves primárias. Por causa da sobrecarga de consultas necessárias para manter a tabela atualizada, essa opção é pouco recomendada.

