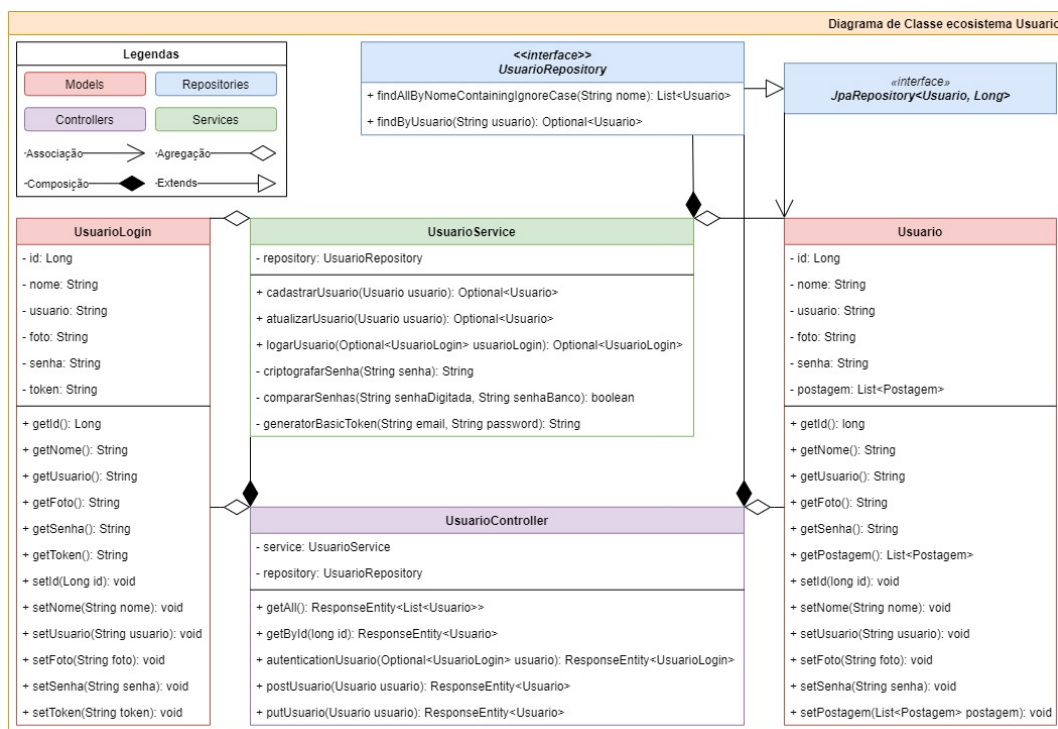


Projeto 02 - Blog Pessoal - Spring Security 01

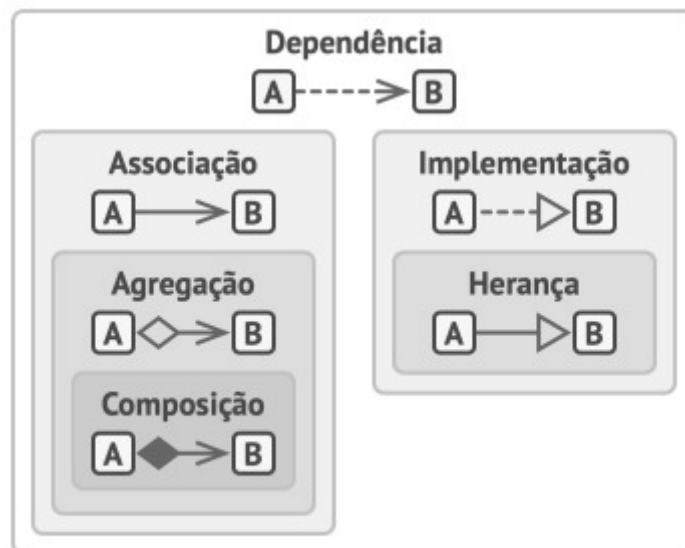
O que veremos por aqui:

1. O Ecossistema do Usuário
2. Apresentação do Recurso Usuário
3. Criar a Classe Model Usuario relacionada com a Classe Postagem
4. Criar a Classe UsuarioLogin
5. Criar a Interface UsuarioRepository

1. Ecossistema do Usuario



No Diagrama de Classes acima, é possível visualizar por completo o ecossistema do Usuário, que representa uma das principais implementações, que permitem que o sistema de autenticação baseado nos detalhes do usuário (UserDetail) funcione. A cor de cada uma das classe representa o pacote onde cada Classe deve ser construída, segundo as informações da Legenda. Para melhor compreensão das relações entre as Classes, veja abaixo uma breve definição sobre as relações entre os Objetos:



Relações entre objetos e classes: da mais fraca a mais forte.

Relação	Descrição
Dependência	Classe A pode ser afetada por mudanças na classe B;
Associação	Objeto A sabe sobre objeto B. Classe A depende de B;
Agregação	Objeto A sabe sobre objeto B, e consiste de B. Classe A depende de B;
Composição	Objeto A sabe sobre objeto B, consiste de B, e gerencia o ciclo de vida de B. Classe A depende de B;
Implementação	Classe A define métodos declarados na interface B. objetos de A podem ser tratados como B. Classe A depende de B;
Herança	Classe A herda a interface e implementação da classe B mas pode estende-la. Objetos de A podem ser tratados como B. Classe A depende de B.

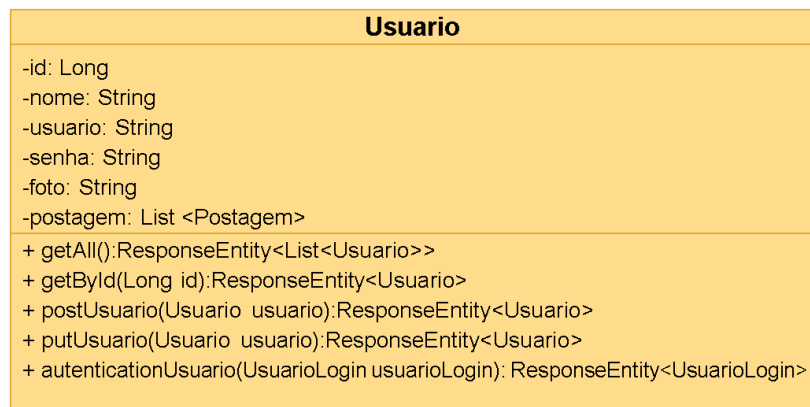
Após compreender o Diagrama de Classes acima e aprender um pouco sobre UML, segue a aplicação do código em cada uma de suas classes.



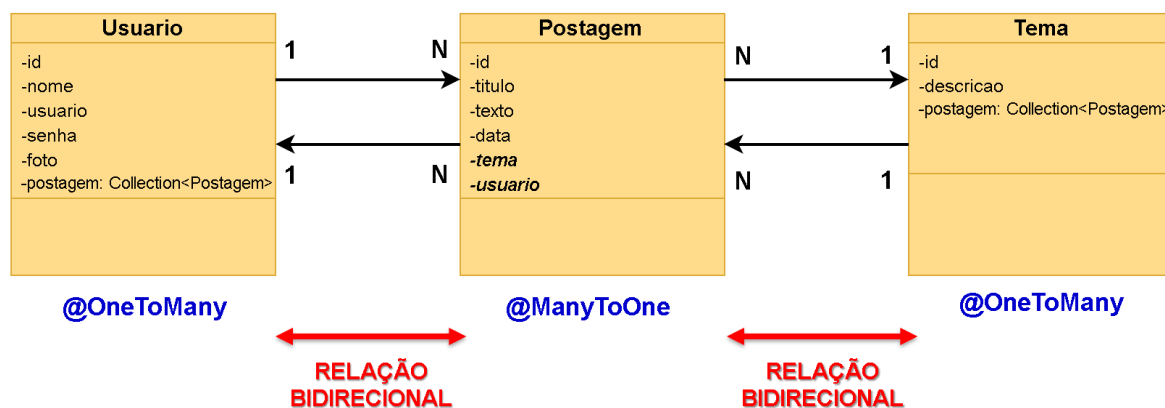
ATENÇÃO: Aproveite para validar seu código do Blog Pessoal, o projeto abaixo serve como espelho para o projeto. Tenha bastante atenção aos detalhes, pois existem implementações fundamentais para o perfeito funcionamento do sistema. Caso surjam dúvidas, não hesite em buscar os Instrutores Generation para obter orientações.

2. O Recurso Usuario

Nesta etapa vamos começar a construir o Recurso Usuario, que será composto por 2 Classes: **Usuario** e **Usuario Login**.

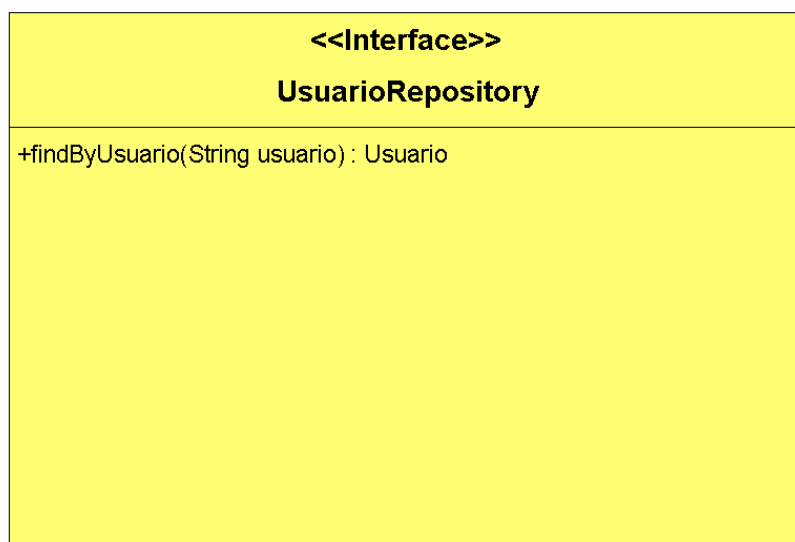


A **Classe Usuario** servirá de modelo para construir a tabela **tb_usuario** (Entidade) dentro do nosso Banco de dados **db_blogpessoal**. Os campos (Atributos) da tabela serão os mesmos que estão definidos no Diagrama de Classes acima. Além de construirmos a Classe Usuario, também faremos o Relacionamento com as Classe Postagem, construída anteriormente. Veja como ficará o Relacionamento entre as 3 Classes, após a implementação, no Diagrama de Classes abaixo:

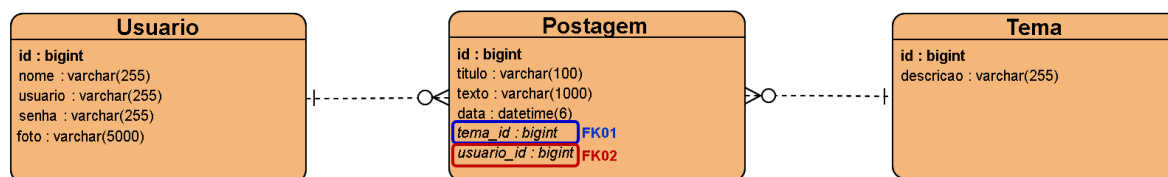


Na sequência vamos construir a **Interface UsuarioRepository**, que irá nos auxiliar na interação com o Banco de dados e com as Classes **UsuarioController**, onde serão implementados os 5 métodos descritos no Diagrama de Classes acima e com a Classe de Serviço (Service), **UsuarioService**, onde serão implementadas as **Regras de Negócio** exigidas pela **Spring Security**.

O Diagrama de Classes da nossa Interface **UsuarioRepository** será igual a imagem abaixo:



Depois de criar a Classe Model Usuario e Relacionamento com a Classe Postagem, a estrutura do nosso Banco de dados **db_blogpessoal** ficará igual ao **Diagrama de Entidade e Relacionamentos (DER)** abaixo:



Observe que a entidade Postagem (tb_postagem) passará a ter **2 Chaves Estrangeiras (FK)**: *tema_id* e *usuario_id*.

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

Atributo	Tipo de dado	Descrição	Chave
id	bigint	Identificador único	PK
nome	varchar(255)	Nome do usuário	
usuario	varchar(255)	E-mail do usuário	
senha	varchar(255)	Senha do usuário	
foto	varchar(5000)	Foto do usuário	



ALERTA DE BSM: Mantenha a Atenção aos Detalhes ao criar o Recurso Usuario. Todas as Camadas (Pacotes: Model, Repository e Controller), já foram criadas nos Recursos Postagem e Tema.



DICA: Caso você tenha alguma dúvida sobre como criar a Classe, executar o projeto, entre outras, consulte a Documentação dos outros Recursos.

Vamos criar a **Classe Usuario Login**, que embora seja criada na Camada Model, ela não irá gerar uma tabela no Banco de dados. Esta Classe será uma Classe auxiliar para o usuário efetuar login no sistema.

Usuario Login
-id: Long -nome: String -usuario: String -senha: String -foto: String -token: String

Passo 01 - Checar as Dependências

No arquivo, **pom.xml**, adicione as linhas abaixo (caso ainda não tenha adicionado):

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
</dependency>
```



[Código fonte: pom.xml](#)



Passo 02 - Criar a Classe Usuario na Camada Model

Agora vamos criar a terceira Classe Model que chamaremos de **Usuario**.

1. Clique com o botão direito do mouse sobre o **Pacote Model** (**com.generation.blogpessoal.model**), na Source Folder Principal (**src/main/java**), e clique na opção **New → Class**
2. Na janela **New Java Class**, no item **Name**, digite o nome da Classe (**Usuario**), e na sequência clique no botão **Finish** para concluir. A seguir, veja a sua implementação:

```
package com.generation.blogpessoal.model;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.validation.constraints.Email;
```

```

import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Entity
@Table(name = "tb_usuarios")
public class Usuario {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull(message = "O atributo Nome é Obrigatório!")
    private String nome;

    @Size(max = 5000,
    message = "O link da foto não pode ser maior do que 5000 caracteres")
    private String foto;

    @NotNull(message = "O atributo Usuário é Obrigatório!")
    @Email(message = "O atributo Usuário deve ser um email válido!")
    private String usuario;

    @NotBlank(message = "O atributo Senha é Obrigatório!")
    @Size(min = 8, message = "A Senha deve ter no mínimo 8 caracteres")
    private String senha;

    @OneToMany(mappedBy = "usuario", cascade = CascadeType.REMOVE)
    @JsonIgnoreProperties("usuario")
    private List<Postagem> postagem;

    /* Insira os Getters and Setters */
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getFoto() {
        return foto;
    }

    public void setFoto(String foto) {
        this.foto = foto;
    }
}

```

```

public String getUsuario() {
    return usuario;
}

public void setUsuario(String usuario) {
    this.usuario = usuario;
}

public String getSenha() {
    return senha;
}

public void setSenha(String senha) {
    this.senha = senha;
}

public List<Postagem> getPostagem() {
    return postagem;
}

public void setPostagem(List<Postagem> postagem) {
    this.postagem = postagem;
}
}

```



ALERTA DE BSM: Mantenha a Atenção aos Detalhes, o atributo senha definido acima, não pode conter o atributo size max, que limita seu tamanho máximo. Dependendo do limite inserido, pode provocar um erro 500, devido ao limite de caracteres que serão utilizados na criptografia. Deixe apenas com size min configurado.



ALERTA DE BSM: Mantenha a Atenção aos Detalhes, o atributo de relacionamento definido como postagem, possui a anotação @OneToMany com a opção cascade type = REMOVE. Se o seu projeto estiver com ALL, ao deletar uma postagem você apagará também o usuário, pois este relacionamento tem uma dependência de pai e filho. Mantenha como REMOVE para que isso não aconteça

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



Passo 03 - Criar a Relação ManyToOne na Classe Postagem

A Classe Postagem será o lado N:1, ou seja, **Muitas Postagens podem ter apenas Um Usuario**. Para criar a Relação vamos inserir depois do último atributo da Classe Postagem (tema), as 3 linhas abaixo:

```

@ManyToOne
@JsonIgnoreProperties("postagem")
private Usuario usuario;

```

Não esqueça de acrescentar os **Métodos Get e Set** para o novo atributo (usuario), que foi adicionado na Classe Postagem. Veja o código completo abaixo:



DICA: Toda vez que você adicionar um novo Atributo na sua Classe, não esqueça de criar os Métodos GET e SET do Atributo. Caso contrário, você não conseguirá visualizar ou atualizar os dados do Atributo.

```
@Entity
@Table(name = "tb_postagens")
public class Postagem {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "O atributo título é obrigatório!")
    @Size(min = 5, max = 100, message = "O atributo título deve conter no mínimo
05         e no máximo 100 caracteres")
    private String titulo;

    @NotBlank(message = "O atributo texto é obrigatório!")
    @Size(min = 10, max = 1000, message = "O atributo texto deve conter no mínimo
10         e no máximo 1000 caracteres")
    private String texto;

    @UpdateTimeStamp
    private LocalDateTime data;

    @ManyToOne
    @JsonIgnoreProperties("postagem")
    private Tema tema;

    @ManyToOne
    @JsonIgnoreProperties("postagem")
    private Usuario usuario;

    public Long getId() {
        return this.id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitulo() {
        return this.titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getTexto() {
        return this.texto;
    }
```



```

    }

    public void setTexto(String texto) {
        this.texto = texto;
    }

    public LocalDateTime getData() {
        return this.data;
    }

    public void setData(LocalDateTime data) {
        this.data = data;
    }

    public Tema getTema() {
        return this.tema;
    }

    public void setTema(Tema tema) {
        this.tema = tema;
    }

    public Usuario getUsuario() {
        return this.usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

}

```

Passo 04 - Criar a Classe UsuarioLogin na Camada Model

Agora vamos criar a quarta Classe Model que chamaremos de **UsuarioLogin**.

A classe **UsuarioLogin** é responsável por definir que o cliente ao tentar autenticar (fazer login) no sistema, forneça apenas o usuário (e-mail) e a senha. Essa classe também pode ser definida como uma DTO (*Data transfer object*), que é uma classe que é utilizada para transitar dados do sistema sem revelar sua Classe Model para o cliente.

1. Clique com o botão direito do mouse sobre o **Pacote Model** (**com.generation.blogpessoal.model**), na Source Folder Principal (**src/main/java**), e clique na opção **New → Class**
2. Na janela **New Java Class**, no item **Name**, digite o nome da Classe (**UsuarioLogin**), e na sequência clique no botão **Finish** para concluir.

A seguir veja sua implementação:

```

package com.generation.blogpessoal.model;

public class UsuarioLogin {

    private Long id;
    private String nome;

```

```
private String usuario;
private String foto;
private String senha;
private String token;

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public String getUsuario() {
    return usuario;
}

public String getFoto() {
    return foto;
}

public void setFoto(String foto) {
    this.foto = foto;
}



public String getToken() {
    return token;
}

public void setToken(String token) {
    this.token = token;
}

public void setUsuario(String usuario) {
    this.usuario = usuario;
}

public String getSenha() {
    return senha;
}

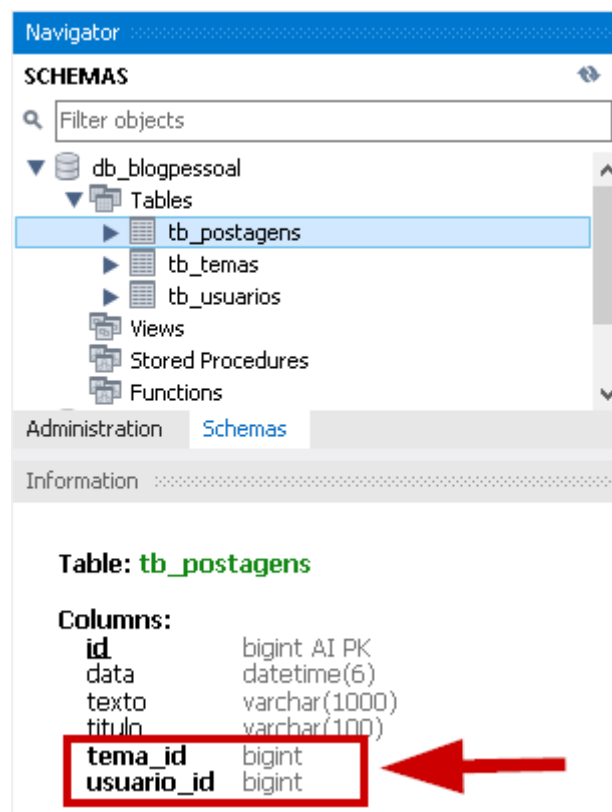
public void setSenha(String senha) {
    this.senha = senha;
}
}
```

	ALERTA DE BSM: Mantenha a Atenção aos Detalhes, não se esqueça de passar o id do usuario como atributo da classe, pois este atributo será utilizado como credencial pelo front-end para pegar o usuario pelo id. Todos os atributos presentes na Classe Model Usuario devem estar presentes na Classe UsuarioLogin.
	ALERTA DE BSM: Mantenha a Atenção aos Detalhes, não esquecer também do atributo token, pois o mesmo será passado no cabeçalho de todas as requisições que do front-end. Este atributo é fundamental para o funcionamento do consumo da api.

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

Passo 05 - Executar o projeto e Checar o Banco de dados

1. Execute o projeto e verifique no **MySQL Workbench** se a tabela **tb_usuarios** foi criada no Banco de dados **db_blogpessoal** com o respectivo relacionamento com a tabela **tb_postagens**, onde foi criada a Chave Estrangeira **usuario_id**.



Passo 06 - Criar a Interface UsuarioRepository na Camada Repository

Agora vamos criar a Interface Repository que chamaremos de **UsuarioRepository**.

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

2. Na sequência, clique na opção **New → Interface**
3. Na janela **New Java Interface**, no item **Name**, digite o nome da Interface (**PostagemRespository**), e na sequência clique no botão **Finish** para concluir.

A seguir veja sua implementação:

```
package com.generation.blogpessoal.repository;

import java.util.Optional;

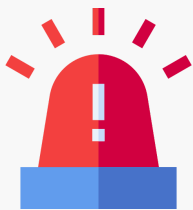
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.generation.blogpessoal.model.Usuario;

@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long>{

    public Optional<Usuario> findByUsuario(String usuario);

}
```



ALERTA DE BSM: Mantenha a Atenção aos Detalhes, tome muito cuidado pois se a escrita dos métodos findBy ou findAllBy estiver errada, pode aparecer um erro no console do STS.

Observe que foi criada uma Query Method na Interface UsuarioRepository, conforme detalhado abaixo:

Query Method

```
public Optional<Usuario> findByUsuario(String usuario);
```

Instrução SQL equivalente

```
SELECT * FROM tb_usuario WHERE usuario = "usuario";
```

Palavra		Instrução SQL
find	→	SELECT
By	→	WHERE
Usuario	→	Atributo da Classe Usuario
String usuario	→	Parâmetro do Método contendo o e-mail do usuário que você deseja procurar.



ATENÇÃO: A instrução *FROM tb_usuario* será inserida pelo JPA ao checar o nome da tabela gerada pela Classe Usuario.

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



[Código fonte: Camada Model e Repository](#)