**Spring Framework**

Conjunto de tecnologias para desenvolvimento de aplicações web em Java.

É composto por diversos módulos que incluem desde bibliotecas para acesso a banco de dados até recursos para desenvolvimento de aplicações para web como por exemplo projetos baseados no padrão **MVC** ou **API**.

Logotipo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

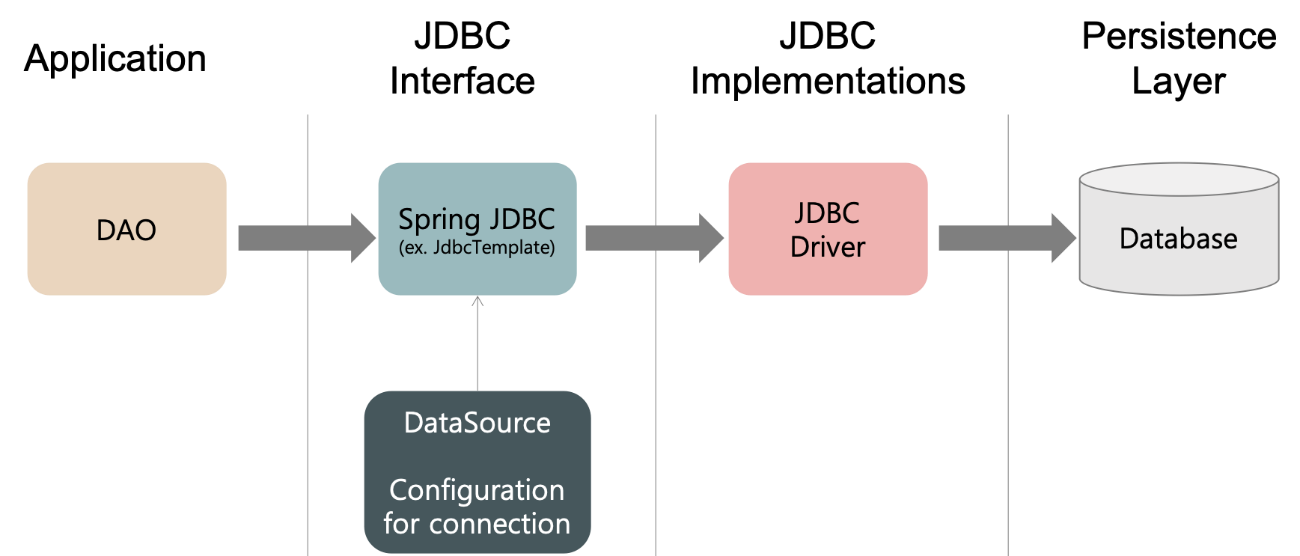
**Configurando o Framework Spring para**

**acessar uma base de dados**

Precisamos mapear no projeto os parâmetros necessários para conectar nossa aplicação em um banco de dados, neste caso do PostGreSQL.

**Data Source**

Nome dado a configuração feita no projeto Spring para definir qual o banco de dados será acessado pela aplicação. Todo projeto criado em Spring irá configurar a conexão do seu banco de dados através de um **Data Source**



**Onde configuramos o Data Source do projeto?**

Todas as configurações do projeto Spring MVC ficam dentro da classe:

/config/MvcConfiguration.java

Esta é a principal classe de configuração do projeto Spring MVC.

**@Bean**

Annotation do Spring Framework utilizada para criarmos um método de configuração do projeto.

package br.com.cotiinformatica.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.ViewResolver;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration

@ComponentScan(basePackages="br.com.cotiinformatica")

@EnableWebMvc

public class MvcConfiguration extends WebMvcConfigurerAdapter{

**@Bean**

**public ViewResolver getViewResolver(){**

**InternalResourceViewResolver resolver**

**= new InternalResourceViewResolver();**

**resolver.setPrefix("/WEB-INF/views/");**

**resolver.setSuffix(".jsp");**

**return resolver;**

**}**

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/resources/\*\*").addResourceLocations("/resources/");

}

}

**Precisamos adicionar uma configuração nesta classe para definir o Data Source do projeto** (conexão com banco de dados do projeto):

package br.com.cotiinformatica.config;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.web.servlet.ViewResolver;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration

@ComponentScan(basePackages = "br.com.cotiinformatica")

@EnableWebMvc

public class MvcConfiguration extends WebMvcConfigurerAdapter {

@Bean

public ViewResolver getViewResolver() {

InternalResourceViewResolver resolver

= new InternalResourceViewResolver();

resolver.setPrefix("/WEB-INF/views/");

resolver.setSuffix(".jsp");

return resolver;

}

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/resources/\*\*").addResourceLocations("/resources/");

}

**/\***

**\* Método para configurar o DATA SOURCE do projeto,**

**ou seja, a conexão com o**

**\* banco de dados utilizado pelo projeto**

**\*/**

**@Bean**

**public DataSource getDataSource() {**

**DriverManagerDataSource dataSource**

**= new DriverManagerDataSource();**

**dataSource.setDriverClassName("org.postgresql.Driver");**

**dataSource.setUrl**

**("jdbc:postgresql://localhost:5432/bd\_sistemacontas");**

**dataSource.setUsername("postgres");**

**dataSource.setPassword("coti");**

**return dataSource;**

**}**

}

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

**Padrão Repository**

Consiste em criarmos classes chamadas de repositórios para realizar as operações necessárias em banco de dados para cada tabela/entidade do sistema (CRUD).

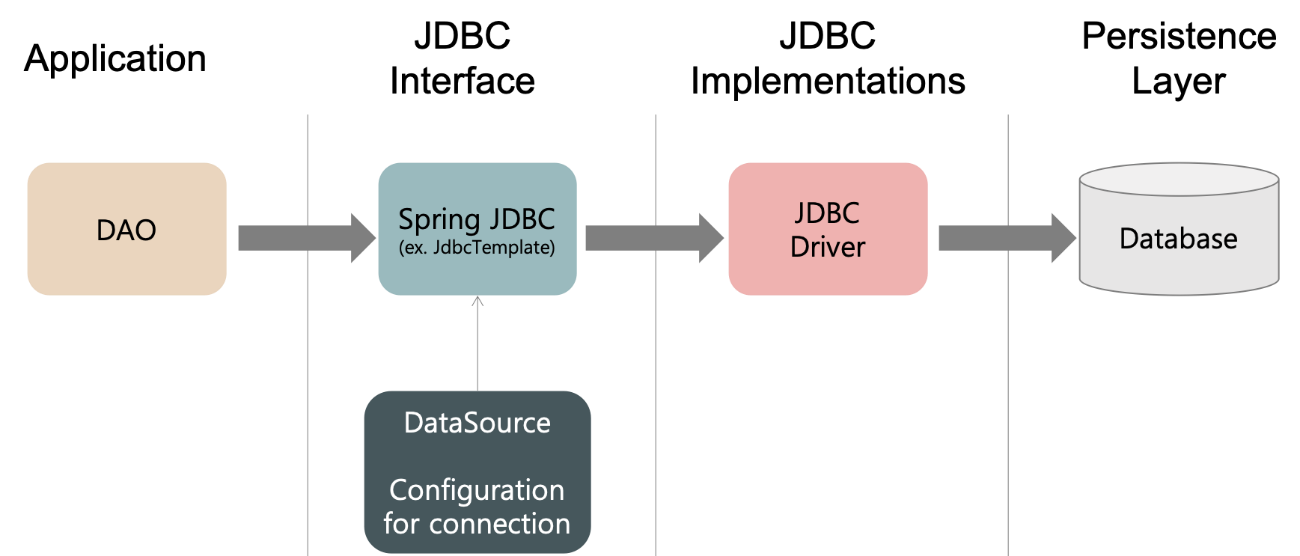
Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**JDBC Template**

É uma biblioteca do Spring Framework utilizada para simplificar a escrita de rotinas de banco de dados no projeto por meio do JDBC.

Para usarmos o JDBC Template precisamos configurar o **Data Source** (configuração do banco de dados) no projeto e fazer com que as classes de repositório possam receber esta configuração diretamente do Spring.



Criando as classes de repositório utilizando o **JdbcTemplate** do Spring e a conexão definida pelo **DataSource**:

**package** br.com.cotiinformatica.repositories;

**import** javax.sql.DataSource;

**import** org.springframework.jdbc.core.JdbcTemplate;

**public** **class** UsuarioRepository {

// atributo

**private** JdbcTemplate jdbcTemplate;

// método construtor

**public** UsuarioRepository(DataSource dataSource) {

jdbcTemplate = **new** JdbcTemplate(dataSource);

}

}

**package** br.com.cotiinformatica.repositories;

**import** javax.sql.DataSource;

**import** org.springframework.jdbc.core.JdbcTemplate;

**public** **class** ContaRepository {

// atributo

**private** JdbcTemplate jdbcTemplate;

// construtor

**public** ContaRepository(DataSource dataSource) {

jdbcTemplate = **new** JdbcTemplate(dataSource);

}

}

Se cada classe **Repository** do projeto precisa receber um **Data Source**, então precisamos configurar como este DataSource seja enviado para essas classes.

Voltando na classe de configuração do projeto:

/config/MvcConfiguration.java

package br.com.cotiinformatica.config;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.web.servlet.ViewResolver;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

import br.com.cotiinformatica.repositories.ContaRepository;

import br.com.cotiinformatica.repositories.UsuarioRepository;

@Configuration

@ComponentScan(basePackages = "br.com.cotiinformatica")

@EnableWebMvc

public class MvcConfiguration extends WebMvcConfigurerAdapter {

@Bean

public ViewResolver getViewResolver() {

InternalResourceViewResolver resolver

= new InternalResourceViewResolver();

resolver.setPrefix("/WEB-INF/views/");

resolver.setSuffix(".jsp");

return resolver;

}

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/resources/\*\*")

.addResourceLocations("/resources/");

}

/\*

\* Método para configurar o DATA SOURCE do projeto,

ou seja, a conexão com o

\* banco de dados utilizado pelo projeto

\*/

@Bean

public DataSource getDataSource() {

DriverManagerDataSource dataSource

= new DriverManagerDataSource();

dataSource.setDriverClassName("org.postgresql.Driver");

dataSource.setUrl

("jdbc:postgresql://localhost:5432/bd\_sistemacontas");

dataSource.setUsername("postgres");

dataSource.setPassword("coti");

return dataSource;

}

**/\***

**\* Configuração para que a classe UsuarioRepository**

**\* possa receber o DataSource do projeto**

**\* (conexão com o banco de dados)**

**\*/**

**@Bean**

**public UsuarioRepository getUsuarioRepository() {**

**return new UsuarioRepository(getDataSource());**

**}**

**/\***

**\* Configuração para que a classe ContaRepository**

**\* possa receber o DataSource do projeto**

**\* (conexão com o banco de dados)**

**\*/**

**@Bean**

**public ContaRepository getContaRepository() {**

**return new ContaRepository(getDataSource());**

**}**

}

Desenvolvendo o repositório de usuários:

/repositories/UsuarioRepository.java

**package** br.com.cotiinformatica.repositories;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.List;

**import** javax.sql.DataSource;

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.jdbc.core.RowMapper;

**import** br.com.cotiinformatica.entities.Usuario;

**public** **class** UsuarioRepository {

// atributo

**private** JdbcTemplate jdbcTemplate;

// método construtor

**public** UsuarioRepository(DataSource dataSource) {

jdbcTemplate = **new** JdbcTemplate(dataSource);

}

**public** **void** create(Usuario usuario) **throws** Exception {

String query = "insert into usuario(nome, email, senha)

values(?,?, md5(?))";

Object[] params = { usuario.getNome(), usuario.getEmail(),

usuario.getSenha() };

jdbcTemplate.update(query, params);

}

**public** Usuario findByEmail(String email) **throws** Exception {

String query = "select \* from usuario where email = ?";

Object[] params = { email };

List<Usuario> lista = jdbcTemplate.query

(query, params, **new** RowMapper<Usuario>() {

@Override

**public** Usuario mapRow(ResultSet rs, **int** rowNum)

**throws** SQLException {

Usuario usuario = **new** Usuario();

usuario.setIdUsuario(rs.getInt("idusuario"));

usuario.setNome(rs.getString("nome"));

usuario.setEmail(rs.getString("email"));

usuario.setSenha(rs.getString("senha"));

**return** usuario;

}

});

**if**(lista.size() == 1)

//verificando se 1 usuário foi encontrado

**return** lista.get(0);

//retornando os dados do usuário encontrado

**else**

**return** **null**; //retornando vazio

}

}

**Voltando para o MVC – Model View e Controller**

Revisando os conceitos do padrão MVC.

O padrão MVC é utilizado para desenvolvimento de aplicações web em Java (através do Spring MVC) mas também em outras plataformas de desenvolvimento.

Resumindo...

**Views**

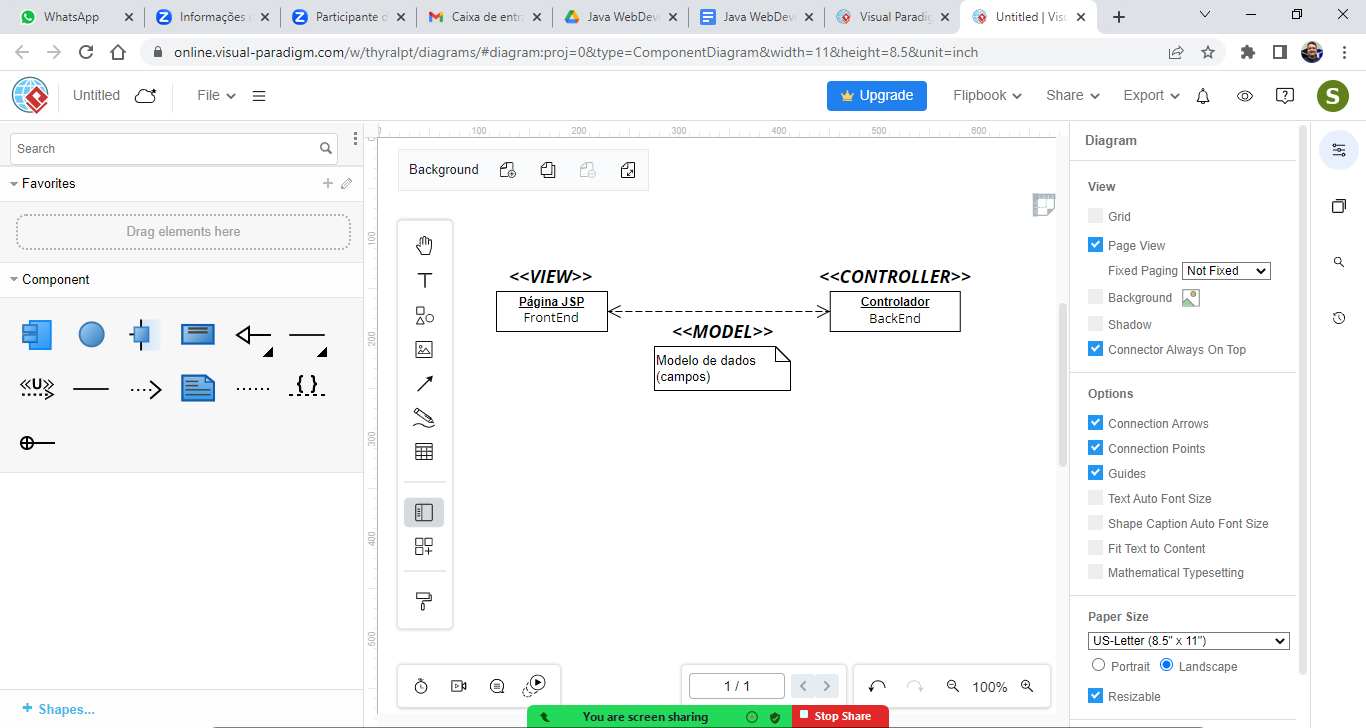
São as páginas utilizadas para definir a interface web do sistema (FrontEnd). No Spring MVC, estas páginas são arquivos de extensão **.jsp**

**Controllers**

São as classes utilizadas para mapear as rotas de navegação para as Views e capturar as requisições enviadas das páginas para o controlador.

**Models**

São classes simples utilizadas para capturar os dados de uma página e enviar estes dados para um controlador e vice versa.



Criando uma classe de modelo de dados para capturar os campos do formulário de cadastro de usuário:

Interface gráfica do usuário, Aplicativo, Excel

Descrição gerada automaticamente

Vamos criar a classe que será utilizada como

modelo de dados para capturar os campos

da página de cadastro de usuário:

/models/CriarUsuarioModel.java

**package** br.com.cotiinformatica.models;

**import** lombok.AllArgsConstructor;

**import** lombok.Getter;

**import** lombok.NoArgsConstructor;

**import** lombok.Setter;

**import** lombok.ToString;

@Setter

@Getter

@NoArgsConstructor

@AllArgsConstructor

@ToString

**public** **class** CriarUsuarioModel {

**private** String nome;

**private** String email;

**private** String senha;

}

Precisamos associar esta classe Model à página de criação de usuário, pois esta classe será utilizada para capturar os campos do formulário da página:

Interface gráfica do usuário, Aplicativo, Excel

Descrição gerada automaticamente

Voltando no controlador:

/controllers/CriarUsuarioController.java

package br.com.cotiinformatica.controller;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;

import br.com.cotiinformatica.models.CriarUsuarioModel;

@Controller

public class CriarUsuarioController {

@RequestMapping(value = "/criar-usuario") // ROTA (navegação)

public ModelAndView criarUsuario() {

// WEB-INF/views/criar-usuario.jsp

ModelAndView modelAndView = new ModelAndView("criar-usuario");

**modelAndView.addObject("model", new CriarUsuarioModel());**

return modelAndView;

}

}

Vamos para a página JSP e fazer a captura dos campos:

/WEB-INF/views/criar-usuario.jsp

Interface gráfica do usuário, Aplicativo, Excel

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!-- Biblioteca de Tags do Spring MVC -->

<%@ taglib uri=*"http://www.springframework.org/tags/form"*

prefix=*"form"* %>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

<!-- CDN da folha de estilos CSS do bootstrap -->

<link href=*"https://cdn.jsdelivr.net/npm/bootstrap*

*@5.3.0-alpha1/dist/css/bootstrap.min.css"* rel=*"stylesheet"* />

<style>

**label***.error* { color: *#df4759*; }

**input***.error* { border: *2px solid #df4759*; }

</style>

</head>

<body>

<div class=*"row"*>

<div class=*"col-md-4 offset-md-4"*>

<div class=*"card mt-3"*>

<div class=*"card-body"*>

<div class=*"text-center"*>

<h2>Sistema de Contas</h2>

<h5>Cadastro de usuários</h5>

</div>

<hr/>

**<form id=*"form\_criarusuario"***

**action=*"cadastrar-usuario"* method=*"post"*>**

<div class=*"mb-2"*>

<label>Entre com

o seu nome:</label>

<form:input path=*"model.nome"*

type=*"text"* name=*"nome"* class=*"form-control"*/>

</div>

<div class=*"mb-2"*>

<label>Entre com o seu

email:</label>

<form:input

path=*"model.email"* type=*"text"* name=*"email"* class=*"form-control"*/>

</div>

<div class=*"mb-2"*>

<label>

Entre com a sua senha</label>

<form:input

path=*"model.senha"* type=*"password"* name=*"senha"* id=*"senha"* class=*"form-control"*/>

</div>

<div class=*"mb-2"*>

<label>Confirme a sua

senha</label>

<input type=*"password"*

name=*"senhaConfirmacao"* class=*"form-control"*/>

</div>

<div class=*"mb-2 d-grid"*>

<input type=*"submit"*

value=*"Cadastrar"* class=*"btn btn-primary"*/>

</div>

<div class=*"mb-2 d-grid"*>

<a href=*"/sistemacontas/"*

class=*"btn btn-light"*>

Já possui cadastro?

<strong>Acesse aqui!</strong>

</a>

</div>

</form>

</div>

</div>

</div>

</div>

<!-- CDN do arquivo javascript do bootstrap -->

<script src=*"https://cdn.jsdelivr.net/npm/bootstrap*

*@5.3.0-alpha1/dist/js/bootstrap.bundle.min.js"*></script>

<!-- CDN do arquivo javascript do JQuery -->

<script src=*"https://code.jquery.com*

*/jquery-3.6.3.min.js"*></script>

<!-- CDN dos arquivos da biblioteca JQuery Validation -->

<script src=*"https://cdnjs.cloudflare.com/ajax/libs*

*/jquery-validate/1.19.5/jquery.validate.min.js"*></script>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs*

*/jquery-validate/1.19.5*

*/additional-methods.min.js"*></script>

<script src=*"https://cdnjs.cloudflare.com/ajax/libs*

*/jquery-validate/1.19.5/localization*

*/messages\_pt\_BR.min.js"*></script>

<script>

$(document).ready(**function**() {

$("#form\_criarusuario").validate({

rules: {

'nome' : { required: **true**, minlength: 8,

maxlength: 150 },

'email' : { required: **true**,

email : **true** },

'senha' : { required: **true**,

minlength: 8,

maxlength: 20 },

'senhaConfirmacao' : { required: **true**,

equalTo: '#senha' }

}

});

})

</script>

</body>

</html>

**No controlador, vamos criar o método para capturar o envio do formulário (SUBMIT) e então cadastrar o usuário no banco de dados.**

**<form id=*"form\_criarusuario"***

**action=*"cadastrar-usuario"***

**method=*"post"*>**

Interface gráfica do usuário, Aplicativo, Excel

Descrição gerada automaticamente

/controllers/CriarUsuarioController.java

**package** br.com.cotiinformatica.controller;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.servlet.ModelAndView;

**import** br.com.cotiinformatica.models.CriarUsuarioModel;

@Controller

**public** **class** CriarUsuarioController {

/\*

\* Método para abrir a página

\*/

@RequestMapping(value = "/criar-usuario") // ROTA (navegação)

**public** ModelAndView criarUsuario() {

// WEB-INF/views/criar-usuario.jsp

ModelAndView modelAndView = **new** ModelAndView

("criar-usuario");

modelAndView.addObject("model", **new** CriarUsuarioModel());

**return** modelAndView;

}

**/\***

**\* Método para receber o SUBMIT POST do formulário**

**\*/**

**@RequestMapping(value = "/cadastrar-usuario",**

**method = RequestMethod.*POST*)**

**public ModelAndView cadastrarUsuario(CriarUsuarioModel model) {**

**ModelAndView modelAndView = new ModelAndView**

**("criar-usuario");**

**try {**

**//TODO**

**}**

**catch(Exception e) {**

**//TODO**

**}**

**modelAndView.addObject("model", model);**

**return modelAndView;**

**}**

}

**Debugando o código fonte do projeto:**

Debugando no Eclipse:

\*\* Primeiro, precisamos criar um **BREAKPOINT**

(Local onde irá começar o Debug)

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Executando o projeto em modo Debug

**/DEBUG AS / DEBUG ON SERVER**

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Agora, estamos na linha do código

onde marcamos o BREAKPOINT:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Inspecionado uma variável:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Para que possamos cadastrar o usuário, o controlador vai precisar acessar o repositório:

Interface gráfica do usuário, Aplicativo, Excel

Descrição gerada automaticamente

**@Autowired**

Annotation utilizada pelo Spring para inicializar objetos de forma automática, utilizando um recurso nativo do framework chamado de **injeção de dependência**.

package br.com.cotiinformatica.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.servlet.ModelAndView;

import br.com.cotiinformatica.models.CriarUsuarioModel;

import br.com.cotiinformatica.repositories.UsuarioRepository;

@Controller

public class CriarUsuarioController {

**@Autowired**

**private UsuarioRepository usuarioRepository;**

(...)

}

**Implementando o fluxo para cadastro do usuário:**

package br.com.cotiinformatica.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.servlet.ModelAndView;

import br.com.cotiinformatica.entities.Usuario;

import br.com.cotiinformatica.models.CriarUsuarioModel;

import br.com.cotiinformatica.repositories.UsuarioRepository;

@Controller

public class CriarUsuarioController {

@Autowired //injeção de dependência (inicialização automática)

private UsuarioRepository usuarioRepository;

/\*

\* Método para abrir a página

\*/

@RequestMapping(value = "/criar-usuario") // ROTA (navegação)

public ModelAndView criarUsuario() {

// WEB-INF/views/criar-usuario.jsp

ModelAndView modelAndView = new ModelAndView("criar-usuario");

modelAndView.addObject("model", new CriarUsuarioModel());

return modelAndView;

}

/\*

\* Método para receber o SUBMIT POST do formulário

\*/

@RequestMapping(value = "/cadastrar-usuario", method = RequestMethod.POST)

public ModelAndView cadastrarUsuario(CriarUsuarioModel model) {

ModelAndView modelAndView = new ModelAndView("criar-usuario");

try {

//verificar se já existe um usuário cadastrado no banco de dados

//com o email informado na página

if(usuarioRepository.findByEmail(model.getEmail()) != null) {

modelAndView.addObject("erro\_email",

"O email informado já está cadastrado, tente outro.");

}

else {

Usuario usuario = new Usuario();

usuario.setNome(model.getNome());

usuario.setEmail(model.getEmail());

usuario.setSenha(model.getSenha());

//gravando o usuário no banco de dados

usuarioRepository.create(usuario);

modelAndView.addObject("mensagem\_sucesso",

"Parabéns! Sua Conta de usuário foi criada com sucesso.");

model = new CriarUsuarioModel();

}

}

catch(Exception e) {

modelAndView.addObject("mensagem\_erro", e.getMessage());

}

modelAndView.addObject("model", model);

return modelAndView;

}

}

**Exibindo as mensagens na página:**

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

<div class=*"text-success"*>

**<strong>${mensagem\_sucesso}</strong>**

</div>

<div class=*"text-danger"*>

**<strong>${mensagem\_erro}</strong>**

</div>

<form id=*"form\_criarusuario"* action=*"cadastrar-usuario"* method=*"post"*>

<div class=*"mb-2"*>

<label>Entre com o seu nome:</label>

<form:input path=*"model.nome"* type=*"text"* name=*"nome"*

class=*"form-control"*/>

</div>

<div class=*"mb-2"*>

<label>Entre com o seu email:</label>

<form:input path=*"model.email"* type=*"text"*

name=*"email"* class=*"form-control"*/>

<span class=*"text-danger"*>

**${erro\_email}**

</span>

</div>

Executando:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**No banco de dados:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**Publicando o trabalho da aula para o GITHUB:**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/sistemacontas (main)

**$ git add .**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/sistemacontas (main)

**$ git commit -m 'Desenvolvimento do cadastro de usuários'**

Samsung@DESKTOP-P9F6D9F MINGW64 ~/Desktop/COTI - Aulas/2023 - Java WebDeveloper SQS 18h as 22h (Início em 09.01)/workspace/sistemacontas (main)

**$ git push -u origin main**

Texto

Descrição gerada automaticamente