Tutorial Tela de Login

Instituto Federal Catarinense Campus Blumenau
Disciplina: Desenvolvimento Web
Discentes: Andrey Moraes, Yuri Teixeira, João Gabriel, Reginaldo

Back-end

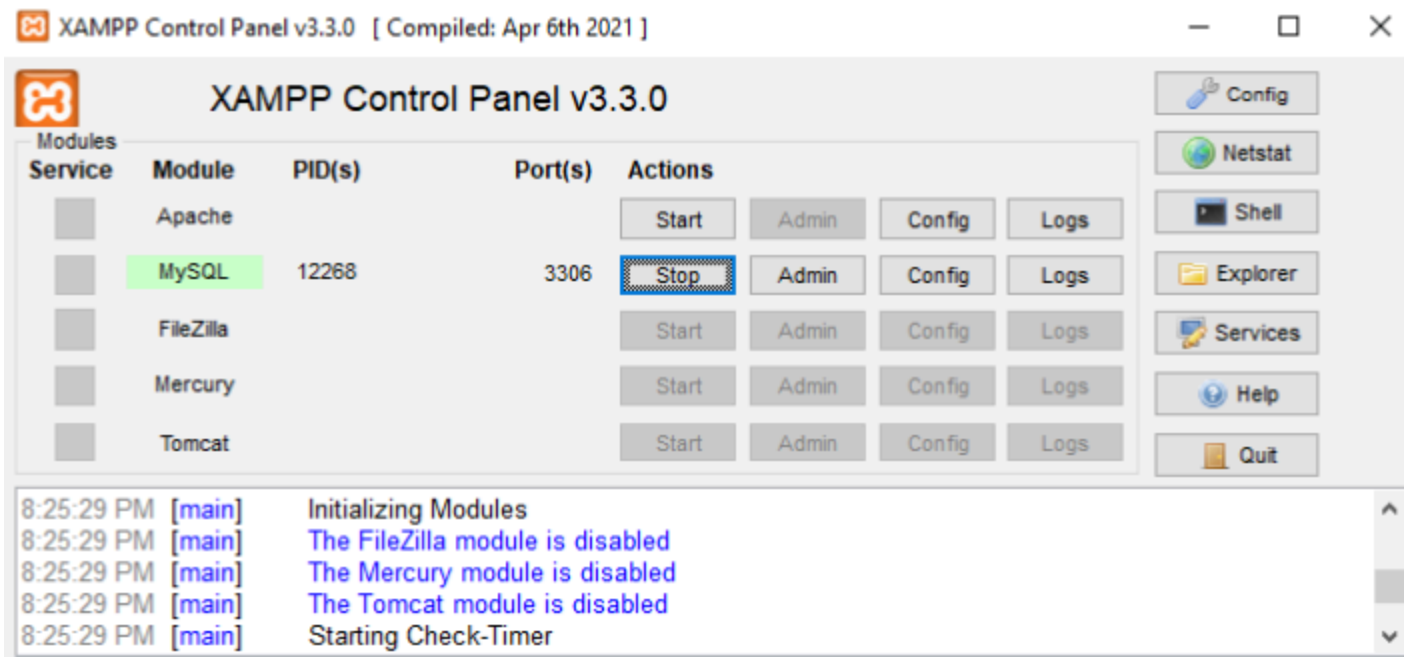
1. Instalação XAMPP

> 💡 **CONTEXTO: O PAPEL DO XAMPP**
>
> O Spring Boot é o framework da aplicação (Java), mas ele precisa de um local para armazenar os dados. O **XAMPP** é utilizado aqui pois instala facilmente um servidor de banco de dados (MySQL/MariaDB) na sua máquina local. Sem ele, a aplicação não teria onde salvar os usuários.

Baixe e instale o XAMPP no site oficial: **https://www.apachefriends.org/**

Abra o **XAMPP Control Panel** e ative o módulo **MySQL**.

[Imagem do XAMPP Control Panel]



2. Criar Projeto Spring

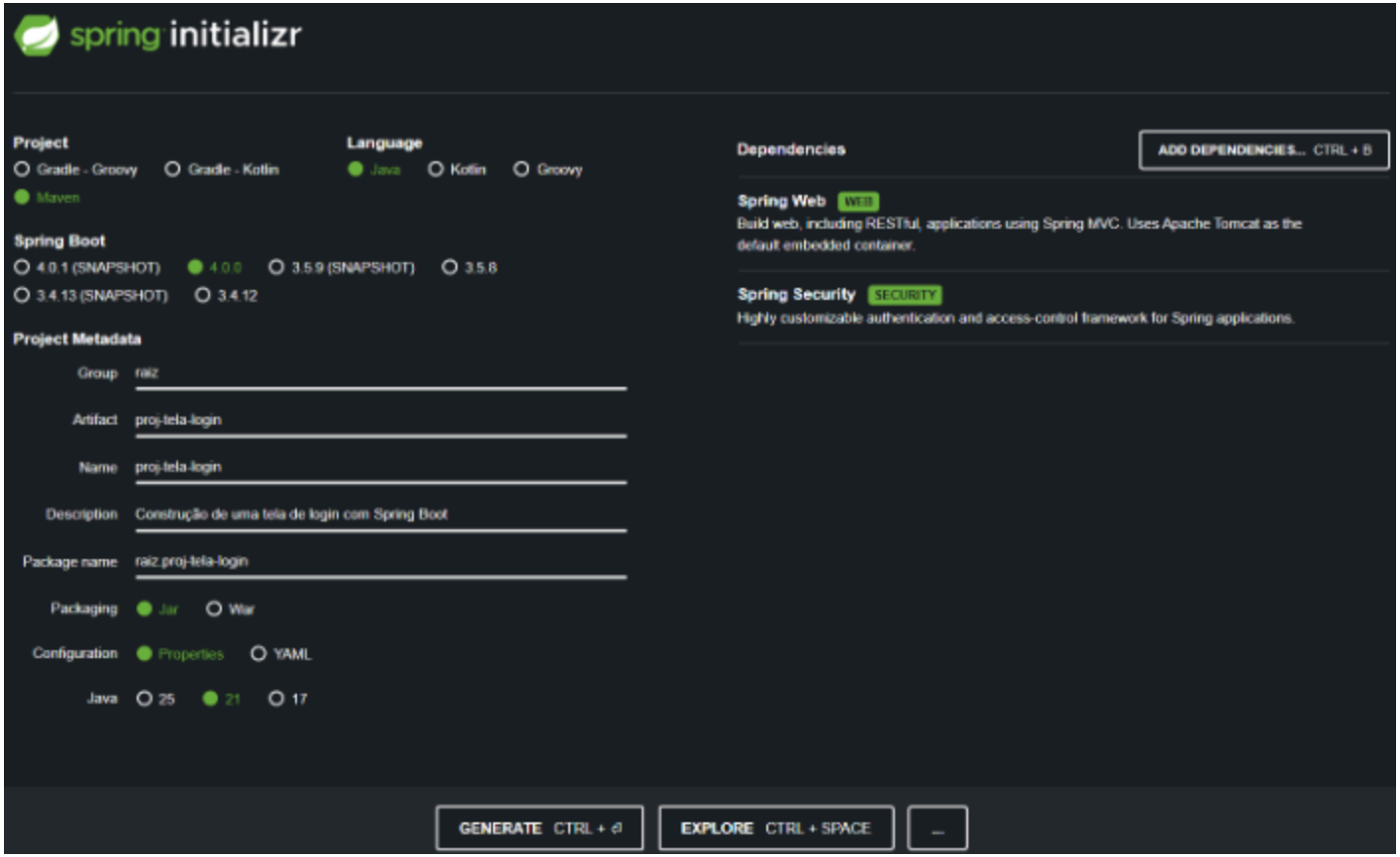
> 💡 **CONTEXTO: SPRING INITIALIZR**
>
> Configurar um projeto Java manualmente é complexo. O **Spring Initializr** cria a estrutura base do projeto, já configurando o Maven (gerenciador de dependências) e as pastas corretas, garantindo que o projeto comece padronizado.

Para construção de um projeto Spring, devemos acessar o site: **https://start.spring.io/**

Na configuração do projeto podemos definir partes do projeto como: tipo do projeto, metadados do projeto, dependências que serão usadas.

Adicione as mesma informações como está a imagem abaixo e ao final clique em **GENERATE** para obter o projeto compactado:

[Imagem do Spring Initializr Configurado]



Defina: Maven, Java, Jar, Java 21. Dependências: Spring Web, Spring Security.

3. Configuração do Ambiente (VS Code)

Descompactar o projeto e abrir no VSCODE. Adicione as seguintes extensões no VSCODE:

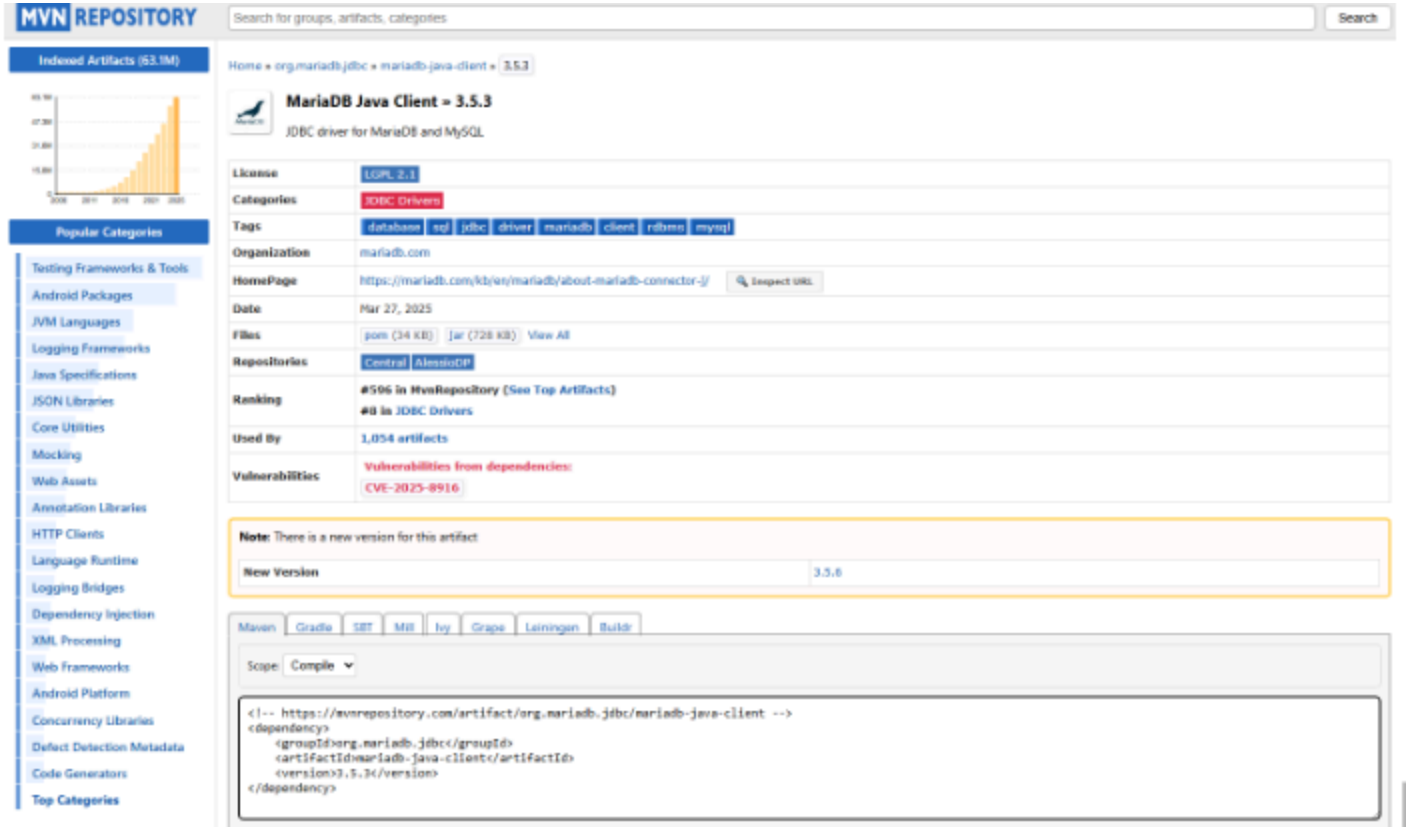
- Extension Pack for Java
- Maven for Java
- Spring Boot Extension Pack
- Live Server
- Language Support for Java(TM) by Red Hat

4. Inserir Dependência via Maven

> 💡 **CONTEXTO: DEPENDÊNCIAS MAVEN**
>
> O arquivo pom.xml gerencia as bibliotecas externas.
>
> - **Data JPA:** Traduz código Java para SQL.
> - **Security:** Gerencia login e senhas.
> - **MariaDB Client:** O driver ("motorista") que permite o Java conversar com o banco MariaDB.

Para inserir dependências no projeto, como o driver com MariaDB, acesse: **mvnrepository.com**. Exemplo MariaDB Java Client:

[Imagem do site MVN Repository]



Exemplo da busca pelo MariaDB Java Client 3.5.3

É necessário acessar o arquivo pom.xml na raiz do projeto e modificar conforme o repositório online. Adicione as seguinte dependências:

pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
```

```
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.mariadb.jdbc</groupId>
        <artifactId>mariadb-java-client</artifactId>
        <version>3.5.3</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.security</groupId>
        <artifactId>spring-security-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

5. Criação e Configuração com Banco de Dados

> 💡 **CONTEXTO: APPLICATION PROPERTIES**
>
> Aqui configuramos onde o banco está (localhost:3306) e quem pode entrar (root). O comando ddl-auto=update é muito útil: ele cria as tabelas no banco automaticamente baseado no seu código Java.

Dentro do diretório resources, acesse o arquivo application.properties, em seguida adicione:

src/main/resources/application.properties

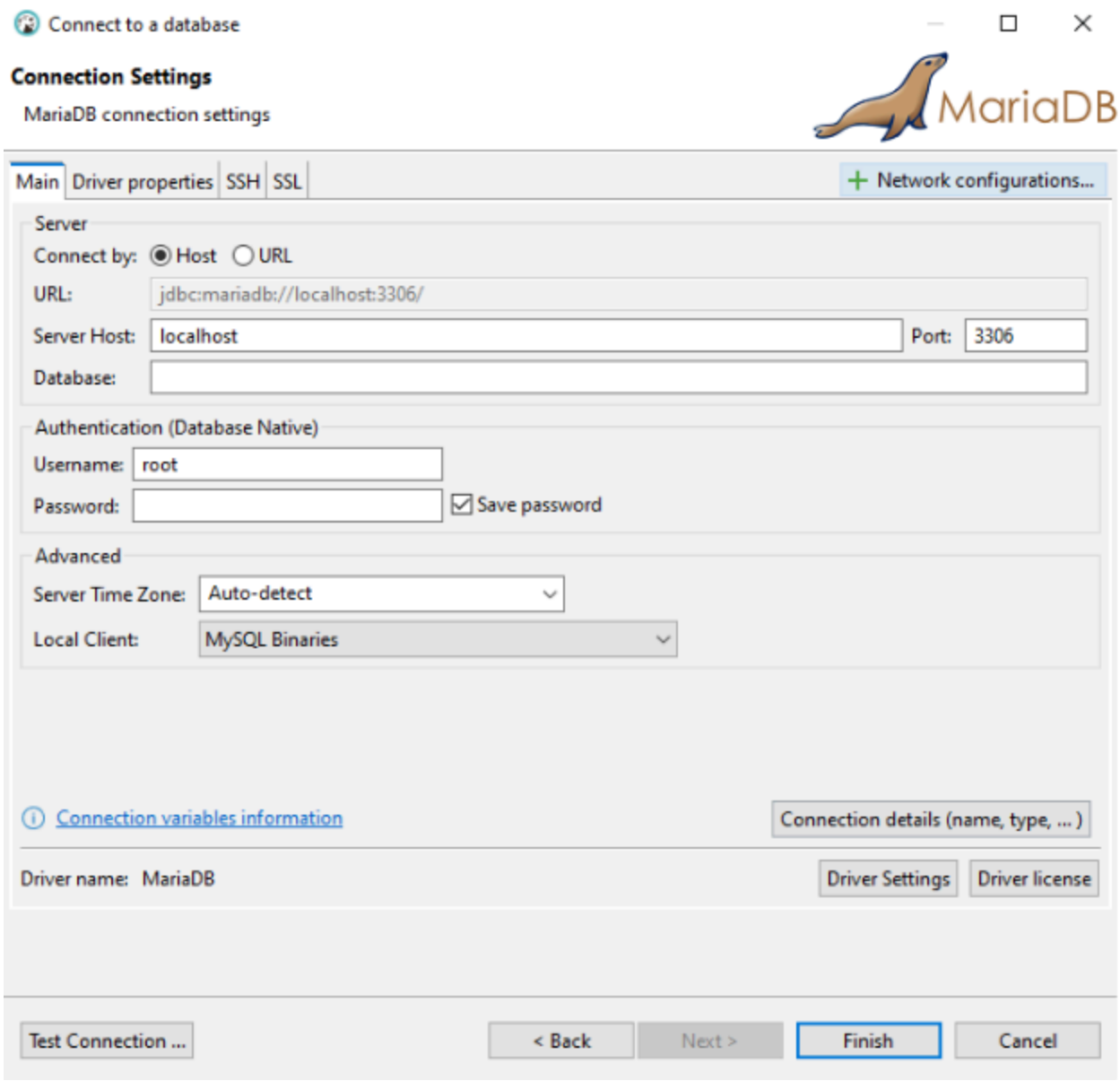
```
spring.application.name=proj-tela-login

spring.datasource.url=jdbc:mariadb://localhost:3306/teladelogin
spring.datasource.username=root
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update
```

> **Obs:** Em username e password, adicione conforme seu usuário root, deve ser igual com seu XAMPP.

Recomendação: Para construir seu banco de dados, pode utilizar Gerenciador de Banco de Dados como o DBeaver, acesse e instale: **https://dbeaver.io/**.

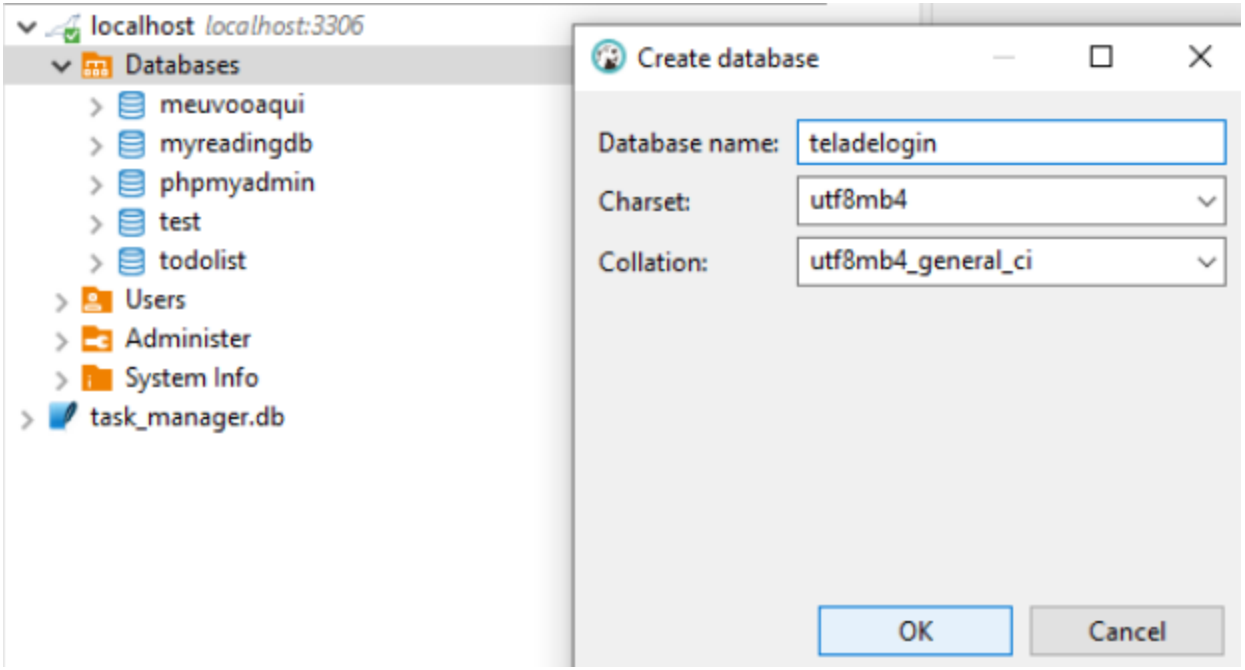
Após instalar, crie uma conexão com MariaDB:

[Imagem: Configuração de Conexão DBeaver]



Após criar a conexão, adicione o banco de dados “teladelogin” clicando em databases com botão direito e depois em Create New Database, abrirá uma tela para inserir o nome, em seguida dê OK:

[Imagem: Criação do Banco de Dados]



6. Criar Security Config

> 💡 **CONTEXTO: O GUARDIÃO**
>
> Essa classe define quem passa e quem é barrado.
>
> - permitAll(): Libera acesso (ex: para carregar o CSS ou a tela de login sem senha).
> - anyRequest().authenticated(): Bloqueia todo o resto até que o usuário logue.
> - csrf.disable(): Desativado aqui para facilitar o teste da API via JavaScript.

A partir da pasta raiz do projeto, dê o seguinte comando via terminal: cd .\src\main\java\raiz\proj_tela_login\.

Agora crie um diretório config, dentro de config crie o arquivo SecurityConfig.java, este arquivo será responsável pela permissão de acesso do projeto:

src/main/java/raiz/proj_tela_login/config/SecurityConfig.java

```
package raiz.proj_tela_login.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/**").permitAll()
                .requestMatchers("/css/**", "/js/**", "/images/**").permitAll()
                .requestMatchers("/*.html").permitAll()
                .requestMatchers("/error").permitAll()
                .anyRequest().authenticated()
            )
            .formLogin(form -> form
                .loginPage("/LoginPage.html")
                .permitAll()
```

```
            )
            .logout(logout -> logout.permitAll());

        return http.build();
    }
}
```

7. Criar Model e DTO

> 💡 **CONTEXTO: MODEL VS DTO**
>
> - **Model (User):** Representa a tabela real no banco de dados. O Spring usa isso para criar as colunas.
> - **DTO (Data Transfer Object):** É um objeto simples usado apenas para transportar dados do Front-end para o Back-end, protegendo a estrutura real do banco.

A partir do diretório proj_tela_login, crie o diretório model, dentro de model criei o arquivo User.java:

src/main/java/raiz/proj_tela_login/model/User.java

```
package raiz.proj_tela_login.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Column;
```

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @Column(unique = true)
    private String email;

    private String password;

    public User() {}

    public User(String name, String email, String password) {
        this.name = name;
        this.email = email;
        this.password = password;
    }

    public Long getId() { return id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }
```

```
    public void setPassword(String password) { this.password = password; }
}
```

Em seguida, dentro do diretório model, crie o diretório DTO, posteriormente o arquivo UserDTO.java:

src/main/java/raiz/proj_tela_login/model/DTO/UserDTO.java

```
package raiz.proj_tela_login.model.DTO;

public class UserDTO {
    private String name;
    private String email;
    private String password;

    public UserDTO(){}

    public UserDTO(String name, String email, String password){
        this.name = name;
        this.email = email;
        this.password = password;
    }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }
```

```
    public void setPassword(String password) { this.password = password; }
}
```

8. Criar Repository

> 💡 **CONTEXTO: JPA REPOSITORY**
>
> Ao estender JpaRepository, ganhamos métodos prontos para o banco de dados (salvar, buscar, deletar) sem escrever SQL manual. O método findByEmail é criado "magicamente" pelo Spring apenas lendo o nome dele.

A partir do diretório proj_tela_login, crie o diretório repository, dentro de repository criei o arquivo UserRepository.java:

src/main/java/raiz/proj_tela_login/repository/UserRepository.java

```
package raiz.proj_tela_login.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import raiz.proj_tela_login.model.User;

public interface UserRepository extends JpaRepository<User, Long> {
    User findByEmailAndPassword(String email, String password);
    User findByEmail(String email);
}
```

9. Criar Service

> 💡 **CONTEXTO: REGRA DE NEGÓCIO E HASH**
>
> O Service contém a lógica. Note o método gerarHash. É crucial para segurança: ele transforma a senha (ex: "123") em um código criptografado (SHA-256) antes de salvar no banco. Assim, nem o administrador do banco consegue ler a senha do usuário.

A partir do diretório proj_tela_login, crie o diretório service, dentro de service criei o arquivo UserService.java:

src/main/java/raiz/proj_tela_login/service/UserService.java

```
package raiz.proj_tela_login.service;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import org.springframework.stereotype.Service;
import raiz.proj_tela_login.model.User;
import raiz.proj_tela_login.repository.UserRepository;

@Service
public class UserService {

    private final UserRepository userRepository;

    public UserService(UserRepository userRepository){
        this.userRepository = userRepository;
    }

    public User createUser(String name, String email, String password) throws Exception{
        User newUser = new User(name, email, gerarHash(password, email));
        if (userRepository.findByEmail(email) != null) {
```

```
            throw new SecurityException("Email já cadastrado");
        }
        return userRepository.save(newUser);
    }

    public User logarUser(String email, String password) throws Exception{
        User user = userRepository.findByEmailAndPassword(email, gerarHash(password, email));
        if (user == null) {
            throw new SecurityException("Email ou senha inválidos");
        } else {
            return user;
        }
    }

    private String gerarHash(String senha, String email) {
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest((senha+email).getBytes());
            StringBuilder hexString = new StringBuilder();
            for(byte b : hash){
                hexString.append(String.format("%02x", b));
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("Erro ao gerar hash", e);
        }
    }
}
```

10. Criar Controller

> 💡 **CONTEXTO: API REST**
>
> O Controller é a "recepção" do Back-end.
>
> - @RestController: Indica que responde dados (JSON) e não HTML.
> - @PostMapping("/login"): Cria o endereço que o JavaScript vai chamar.
> - @CrossOrigin: Permite que diferentes origens (browsers) acessem essa API sem bloqueios.

A partir do diretório proj_tela_login, crie o diretório controller, dentro de controller criei o arquivo AuthController.java:

src/main/java/raiz/proj_tela_login/controller/AuthController.java

```
package raiz.proj_tela_login.controller;

import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
import raiz.proj_tela_login.model.User;
import raiz.proj_tela_login.model.DTO.UserDTO;
import raiz.proj_tela_login.service.UserService;

@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "*")
public class AuthController {

    private final UserService userService;
```

```
    public AuthController(UserService userService){
        this.userService = userService;
    }

    @PostMapping("/login")
    public ResponseEntity<User> login(@RequestBody UserDTO body) throws Exception{
        User user = userService.logarUser(body.getEmail(), body.getPassword());
        return ResponseEntity.ok(user);
    }

    @PostMapping("/register")
    public ResponseEntity<User> register(@RequestBody UserDTO body) throws Exception {
        User userSaved = userService.createUser(body.getName(), body.getEmail(), body.getPassword());
        return ResponseEntity.ok(userSaved);
    }
}
```

Front-end

1. Criar Página HTML

A partir do diretório raiz do projeto, dê o seguinte comando: cd .\src\main\resources\static\.

Após isso, criei o arquivo LoginPage.html, dentro do arquivo adicione:

```
src/main/resources/static/LoginPage.html
```

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login | Cadastro</title>
    <link rel="stylesheet" href="css/style.css">
</head>
<body>

<div class="auth-container">
    <div id="login-form">
        <form>
            <h2>Login</h2>
            <div class="input-group">
                <label for="login-email">Email</label>
                <input type="email" id="login-email" required>
            </div>
            <div class="input-group">
                <label for="login-password">Senha</label>
                <input type="password" id="login-password" required>
            </div>
            <button type="submit" class="auth-button">Entrar</button>
            <p class="toggle-link">
                Não tem uma conta? <a href="#" id="show-register">Cadastre-se</a>
            </p>
        </form>
    </div>

    <div id="register-form" class="hidden">
        <form>
```

```
            <h2>Cadastre-se</h2>
            <div class="input-group">
                <label for="reg-name">Nome</label>
                <input type="text" id="reg-name" required>
            </div>
            <div class="input-group">
                <label for="reg-email">Email</label>
                <input type="email" id="reg-email" required>
            </div>
            <div class="input-group">
                <label for="reg-password">Senha</label>
                <input type="password" id="reg-password" required>
            </div>
            <button type="submit" class="auth-button">Cadastrar</button>
            <p class="toggle-link">
                Já tem uma conta? <a href="#" id="show-login">Faça login</a>
            </p>
        </form>
    </div>
</div>

<script src="js/LoginPage.js"></script>
</body>
</html>
```

Após criar LoginPage.html, crie o arquivo MainPage.html:

```
src/main/resources/static/MainPage.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pagina Principal</title>
</head>
<body>
    <h1>Você chegou a pagina principal!!!</h1>
</body>
</html>
```

2. Criar CSS

A partir de static, crie o diretório css, dentro crie o arquivo style.css e adicione o seguinte código:

src/main/resources/static/css/style.css

```
:root {
    --bg-light: #F7F9FC;
    --surface-white: #FFFFFF;
    --primary-cyan: #00BFFF;
    --primary-cyan-hover: #00A3CC;
    --primary-glow: rgba(0, 191, 255, 0.2);
    --text-dark: #2C3E50;
    --text-muted: #95A5A6;
    --card-border: 1px solid #ECF0F1;
    --card-radius: 12px;
```

```
    --shadow-primary: 0 5px 15px rgba(0, 0, 0, 0.05);
}

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Space Mono', 'Consolas', 'Courier New', monospace;
}

body {
    background-color: var(--bg-light);
    color: var(--text-dark);
    line-height: 1.6;
    min-height: 100vh;
    display: flex;
    flex-direction: column;
}

body:not(:has(.main-header)) {
    justify-content: center;
    align-items: center;
}

h2 {
    color: var(--text-dark);
    font-weight: 700;
    margin-bottom: 1rem;
    text-align: center;
}

.input-group {
```

```
    margin-bottom: 1.25rem;
}

.input-group label {
    font-weight: 700;
    color: var(--text-dark);
    display: block;
    margin-bottom: 0.5rem;
    font-size: 0.9rem;
    text-transform: uppercase;
    letter-spacing: 0.5px;
}

.input-group input {
    width: 100%;
    padding: 0.9rem 1rem;
    border: var(--card-border);
    border-radius: var(--card-radius);
    font-size: 1rem;
    background-color: var(--bg-light);
    color: var(--text-dark);
    transition: all 0.3s ease;
    box-shadow: inset 0 2px 4px rgba(0,0,0,0.02);
}

.input-group input:focus {
    outline: none;
    border-color: var(--primary-cyan);
    background-color: var(--surface-white);
    box-shadow: 0 0 0 4px var(--primary-glow);
}
```

```
.auth-button {
    width: 100%;
    padding: 1rem;
    border: none;
    border-radius: var(--card-radius);
    background-color: var(--primary-cyan);
    color: #FFFFFF;
    font-size: 1rem;
    font-weight: 700;
    cursor: pointer;
    text-transform: uppercase;
    letter-spacing: 1px;
    transition: all 0.3s ease;
    box-shadow: 0 4px 6px rgba(0, 191, 255, 0.2);
}

.auth-button:hover {
    background-color: var(--primary-cyan-hover);
    transform: translateY(-2px);
    box-shadow: 0 6px 12px rgba(0, 191, 255, 0.3);
}

.hidden {
    display: none !important;
}

.auth-container {
    background-color: var(--surface-white);
    padding: 3rem;
    border-radius: var(--card-radius);
    box-shadow: var(--shadow-primary);
    width: 100%;
```

```
    max-width: 450px;
    border: var(--card-border);
    animation: fadeInUp 0.6s ease-out;
}

.auth-container h2 {
    border-bottom: 2px solid var(--primary-cyan);
    padding-bottom: 1rem;
    margin-bottom: 2rem;
}

.toggle-link {
    text-align: center;
    margin-top: 1.5rem;
    font-size: 0.9rem;
    color: var(--text-muted);
}

.toggle-link a {
    color: var(--primary-cyan);
    text-decoration: none;
    font-weight: 700;
}

.toggle-link a:hover {
    text-decoration: underline;
}

@keyframes fadeInUp {
    from { opacity: 0; transform: translateY(20px); }
```

```
    to { opacity: 1; transform: translateY(0); }
}
```

3. Criar JS

> 💡 **CONTEXTO: FETCH API**
>
> Este código usa a função fetch para mandar os dados de login para o servidor sem recarregar a tela. Ele pega o email e senha digitados, envia para /api/login e, se a resposta for positiva, redireciona o usuário para a página principal.

A partir de static, crie o diretório js, dentro crie o arquivo LoginPage.js e adicione o seguinte código:

src/main/resources/static/js/LoginPage.js

```
document.addEventListener("DOMContentLoaded", () => {
    console.log("login.js: DOMContentLoaded");

    const loginFormContainer = document.getElementById("login-form");
    const registerFormContainer = document.getElementById("register-form");
    const showRegisterLink = document.getElementById("show-register");
    const showLoginLink = document.getElementById("show-login");

    if (!loginFormContainer || !registerFormContainer) {
        console.error("login.js: containers not found", {
            loginFormContainer, registerFormContainer
        });
        return;
    }
```

```
const loginForm = loginFormContainer.querySelector("form");
const registerForm = registerFormContainer.querySelector("form");

console.log("login.js: found forms", {
    loginFormExists: !!loginForm, registerFormExists: !!registerForm
});

showRegisterLink?.addEventListener("click", (e) => {
    e.preventDefault();
    loginFormContainer.classList.add("hidden");
    registerFormContainer.classList.remove("hidden");
});

showLoginLink?.addEventListener("click", (e) => {
    e.preventDefault();
    registerFormContainer.classList.add("hidden");
    loginFormContainer.classList.remove("hidden");
});

const loginButton = loginForm?.querySelector('button[type="submit"]');
loginButton?.addEventListener("click", () =>
console.log("login.js: login button clicked"));

if (loginForm) {
    loginForm.addEventListener("submit", async (e) => {
        e.preventDefault();
        console.log("login.js: submit handler fired");

        const email = document.getElementById("login-email").value;
        const password = document.getElementById("login-password")
        .value;
```

```
console.log("login.js: credentials (email):", email);

try {
    const response = await fetch("/api/login", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ email, password }),
        credentials: "same-origin"
    });

    console.log("login.js: POST /api/login status:",
    response.status, "redirected:",
    response.redirected);

    let body = null;
    try {
        body = await response.clone().json();
        console.log("login.js: response JSON:", body);
    } catch (err) {
        console.log("login.js: response not JSON");
    }

    if (!response.ok) {
        const msg = body?.message || `Erro ${response.status}`;
        console.warn("login.js: login failed:", msg);
        alert(msg);
        return;
    }

    if (body && body.id) {
        localStorage.setItem('loggedInUserId',
        String(body.id));
```

```
                localStorage.setItem('userName', body.name);

                window.location.href = "/MainPage.html";
                return;
            }

            alert("Resposta inesperada do servidor.");
        } catch (err) {
            console.error("login.js: fetch error:", err);
            alert("Erro de rede ao tentar logar.
            Veja console/Network.");
        }
    });
} else {
    console.warn("login.js: loginForm is null —
    submit listener not attached");
}

if (registerForm) {
    registerForm.addEventListener("submit", async (e) => {
        e.preventDefault();
        const name = document.getElementById("reg-name").value;
        const email = document.getElementById("reg-email").value;
        const password = document.getElementById("reg-password").value;

        try {
            const response = await fetch("/api/register", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                body: JSON.stringify({ name, email, password }),
                credentials: "same-origin"
            });
```

```
                const data = await response.json();
                console.log("login.js: register response:", data);
                if (response.ok) {
                    alert("Usuário cadastrado com sucesso! Faça login.");
                    registerFormContainer.classList.add("hidden");
                    loginFormContainer.classList.remove("hidden");
                } else {
                    alert(data.message || `Erro ${response.status}`);
                }
            } catch (err) {
                console.error("login.js: register fetch error:", err);
                alert("Erro de rede ao tentar cadastrar.");
            }
        });
    }
});
```

Rodar o Projeto

Para rodar o projeto, de o seguinte comando via terminal: mvn spring-boot:run