Aplicações Móveis

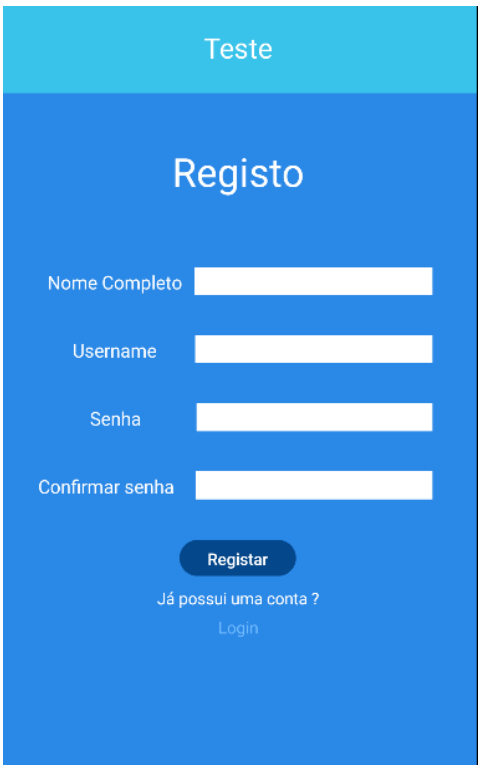
Memorando – Lab4

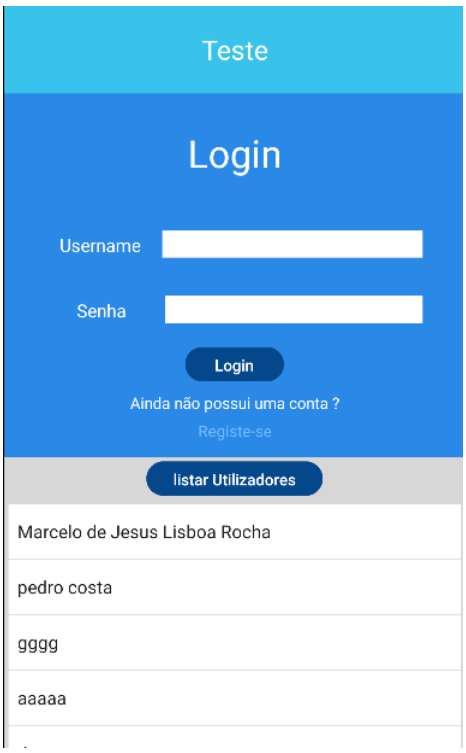
Coordenação de Engenharia Informática

Departamento de Engenharias e Tecnologias

Instituto Superior Politécnico de Tecnologias e Ciências

Nome : Marcelo Rocha - 20210032





- **API (SPRING BOOT JAVA)**

```java
package com.binasjc.spring_server_binasjc.model;

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

@Entity
public class Utilizador {

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

    private String nomecompleto;
    private String username;
    private String senha;

    public Utilizador() {

    }


    public Utilizador(String nomecompleto, String username, String senha) {
        this.nomecompleto = nomecompleto;
        this.username = username;
        this.senha = senha;
    }

    public int getId() {
        return id;
    }

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

    public String getNomecompleto() {
        return nomecompleto;
    }

    public void setNomecompleto(String nomecompleto) {
        this.nomecompleto = nomecompleto;
    }

    public String getUsername() {
        return username;
    }
```

```java
package com.binasjc.spring_server_binasjc.repositories;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.binasjc.spring_server_binasjc.model.Utilizador;


@Repository
public interface UtilizadorRepository extends CrudRepository<Utilizador , Integer> {

    public Utilizador findByUsername(String username);
    public Utilizador findByUsernameAndSenha(String username, String senha);

}
```

```java
import org.springframework.stereotype.Service;

import com.binasjc.spring_server_binasjc.model.Utilizador;
import com.binasjc.spring_server_binasjc.repositories.UtilizadorRepository;

@Service
public class UtilizadorService {

    @Autowired
    private UtilizadorRepository repository;

    public Utilizador save(Utilizador utilizador) {
        return repository.save(utilizador);
    }

    public List<Utilizador> getAllUtilizadores() {
        return (List<Utilizador>) repository.findAll();
    }

    public void delete(Utilizador utilizador) {
        repository.delete(utilizador);
    }

    public Utilizador findByUserName (String username){
        return repository.findByUsername(username);
    }

    public Utilizador findByUserNameAndSenha (String username , String senha){
        return repository.findByUsernameAndSenha(username, senha);
    }
```

```java
@RestController
@RequestMapping("/Utilizador")
public class UtilizadorController {

    @Autowired
    private UtilizadorService utilizadorService;

    @GetMapping("/get-all")
    public List<Utilizador> getAllUtilizadores() {
        return utilizadorService.getAllUtilizadores();
    }

    @GetMapping("/getUtilizadorByUsername")
    public ResponseEntity<Utilizador> getUtilizadorByUsername(@RequestParam String username) {
        Utilizador utilizador = utilizadorService.findByUserName(username);
        if (utilizador == null) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build(); // Retorna 404 se não encontrado
        }
        return ResponseEntity.ok(utilizador);
    }

    @GetMapping("/getUtilizadorByUsernameAndSenha")
    public ResponseEntity<Utilizador> getUtilizadorByUsernameAndSenha(@RequestParam String username , @RequestParam String senha) {
        Utilizador utilizador = utilizadorService.findByUserNameAndSenha(username , senha);
        if (utilizador == null) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).build(); // Retorna 404 se não encontrado
        }
        return ResponseEntity.ok(utilizador);
    }

    @PostMapping("/save")
    public Utilizador saveUtilizador(@RequestBody Utilizador utilizador) {
        return utilizadorService.save(utilizador);
    }

    @DeleteMapping("/delete")
    public void deleteUtilizador(@RequestBody Utilizador utilizador) {
        utilizadorService.delete(utilizador);
    }

}
```

Estrutura da API:

1. **Modelo: Utilizador**
    - A entidade Utilizador representa o usuário da aplicação, com atributos:
        - id: identificador único (gerado automaticamente).
        - nomecompleto: nome completo do usuário.
        - username: nome de usuário utilizado para login.
        - senha: senha do usuário.
    - Métodos: Getters e setters para todos os atributos, além de um construtor padrão e um que inicializa todos os atributos.
2. **Repositório: UtilizadorRepository**
    - Interface UtilizadorRepository que estende CrudRepository<Utilizador, Integer>, fornecendo métodos básicos de persistência e métodos personalizados:
        - findByUsername(String username): busca usuário por nome de usuário.
        - findByUsernameAndSenha(String username, String senha): busca usuário por nome e senha.
3. **Serviço: UtilizadorService**
    - Classe UtilizadorService para lógica de negócios e manipulação dos dados de Utilizador. Possui:
        - save(Utilizador utilizador): salva um novo utilizador.
        - getAllUtilizadores(): retorna todos os utilizadores.
        - delete(Utilizador utilizador): deleta o utilizador.
        - findByUserName(String username): busca utilizador por nome.
        - findByUserNameAndSenha(String username, String senha): busca utilizador por nome e senha.
4. **Controlador: UtilizadorController**
    - Classe UtilizadorController, controladora das requisições HTTP, expõe os endpoints REST:
        - **GET /Utilizador/get-all**: retorna uma lista de todos os utilizadores.
        - **GET /Utilizador/getUtilizadorByUsername**: busca um utilizador pelo nome.
        - **GET /Utilizador/getUtilizadorByUsernameAndSenha**: busca um utilizador pelo nome e senha.
        - **POST /Utilizador/save**: adiciona um novo utilizador.
        - **DELETE /Utilizador/delete**: deleta um utilizador.

Telas de Registro e Login no Android Studio

1. **Tela de Registro (Registrar.class)**

A tela de registro é responsável por captar os dados de um novo usuário e enviá-los ao servidor via API REST para serem salvos. Os principais pontos abordados no fluxo de registro são:

- **Campos**:
  - *nomeCompleto*: para o nome completo do usuário.
  - *userName*: nome de usuário exclusivo.
  - *senha* e *confirmarSenha*: campo para a senha e confirmação da senha.

- **Processo de Validação**:
  - Campos obrigatórios são verificados. Se algum estiver vazio, uma mensagem é exibida através de um Toast.
  - A senha deve ser confirmada com o campo de confirmação para garantir consistência.

- **Verificação de Disponibilidade do Username**:
  - Antes de enviar o registro, a aplicação verifica com o servidor se o nome de usuário já existe usando a função verificarUsuarioExistente(). Caso o nome já esteja registrado, um Toast notifica o usuário, solicitando um novo nome.

- **Registro do Usuário**:
  - Após passar pela verificação, os dados são enviados ao servidor usando o método utilizadorApi.save(), que registra o novo usuário no banco de dados.
  - Em caso de sucesso, os campos são limpos, e um Toast confirma o registro.
  - Em caso de falha, uma mensagem de erro é exibida, indicando falha na conexão com o servidor.

2. **Tela de Login (Login.class)**

A tela de login permite a autenticação do usuário e possibilita também a listagem de todos os usuários registrados. Os principais recursos são:

- **Campos**:
  - *userName*: campo para o nome de usuário.
  - *senha*: campo para a senha.

- **Autenticação**:
    - Ao pressionar o botão de login, os valores dos campos são verificados. Caso algum esteja vazio, um Toast solicita o preenchimento.
    - Após preenchimento, os dados são enviados ao servidor para verificação. Caso o usuário seja encontrado, um Toast indica o sucesso com o nome do usuário encontrado; caso contrário, informa que o usuário não foi localizado.

- **Listagem de Usuários**:
    - Um botão permite listar todos os usuários cadastrados.
    - Ao clicar, a aplicação solicita os dados ao servidor. Caso a requisição seja bem-sucedida, os nomes dos usuários são carregados em uma ListView (listaUtilizadoresView).
    - Em caso de falha na conexão, um Toast informa o erro de rede.

Navegação Entre Telas Ambas as telas permitem navegação entre si:

- No Registrar.class, o botão de navegação leva o usuário para a tela de login.
- No Login.class, há um botão de navegação para a tela de registro.

Essas funcionalidades são essenciais para a integração do aplicativo com o backend, permitindo o gerenciamento de usuários com registro e autenticação completos.