Relatório Técnico - LPS 3

Grupo

Luís Felipe Teixeira Dias Brescia

Gustavo Pereira de Oliveira

Victor Reis Carlota

Grupo do Projeto Analisado

Pedro Negri Leão Lambert

Pedro Henrique Pires

Vinícius Rezende

Tecnologias e Arquitetura do Sistema

Tecnologias Utilizadas

Backend: Spring Boot (Java), com JPA para persistência de dados e JWT para autenticação.

Frontend: React com TypeScript.

Documentação: Swagger.

Testes: Mockito.

Outras Ferramentas: Maven e Postman.

Arquitetura do Sistema

O projeto utiliza uma arquitetura de três camadas (3-tier architecture), comum em sistemas web. Ela é composta por:

1.Camada de Apresentação (Frontend): Desenvolvida em React com TypeScript, responsável pela interface visual do usuário e comunicação com o backend via chamadas HTTP.

2. Camada de Negócio (Backend): Implementada em Spring Boot, gerenciando a lógica do negócio, autenticação (JWT) e endpoints RESTful.

3. Camada de Persistência (Banco de Dados): Utiliza JPA/Hibernate para mapeamento objeto-relacional e comunicação com o banco de dados.

Essa arquitetura separa responsabilidades, facilitando manutenção e escalabilidade.

Organização do Repositório GitHub

Pontos Positivos

1. **Diagramas bem elaborados: Diagramas claros e explicativos auxiliam na compreensão da arquitetura.**
2. **Organização de Arquivos: Estrutura bem organizada, facilitando navegação e localização de componentes.**

Pontos a Melhorar

1.Melhoria na Organização de Pastas e Arquivos

Atualmente, a organização parece básica, dificultando a escalabilidade do projeto. Sugestão: adotar um padrão modularizado, como o de feature-based architecture no frontend.

Exemplo Atual (Frontend):

Tudo centralizado em uma única pasta src.

Sugestão de Estrutura:

src/

components/

Header/

Header.tsx

Header.css

pages/

Login/

LoginPage.tsx

LoginForm.tsx

services/

api.ts

utils/

validations.ts

2.Documentação do Código

Faltam comentários e explicações em partes críticas do código. Usar JavaDoc no backend e comentários claros no frontend.

Exemplo Atual (Backend):

```
public ResponseEntity<User> getUserById(Long id) {
    return userService.findById(id);
}
```

Melhoria:

```
/**
 * Retrieves a user by their ID.
 *
 * @param id the ID of the user
 * @return the ResponseEntity containing the user or a 404 error if not found
```

```
 */
public ResponseEntity<User> getUserById(Long id) {
    return userService.findById(id);
}
```

3.Aprimoramento da Validação

Adicionar validações robustas no backend utilizando o Bean Validation.

Exemplo Atual: Validação mínima nos dados recebidos.

Melhoria:

```
public class User {
    @NotBlank(message = "Name cannot be empty")
    private String name;

    @Email(message = "Invalid email format")
    private String email;

    @Positive(message = "Age must be positive")
    private int age;
}
```

4.Cobertura de Testes

A cobertura de testes está limitada. Incluir testes de integração no backend com ferramentas como o Spring Test.

Exemplo Atual:

Testes focados apenas em métodos isolados.

Melhoria:

```
@Test
public void testGetUserById() throws Exception {
   mockMvc.perform(get("/users/{id}", 1L))
          .andExpect(status().isOk())
          .andExpect(jsonPath("$.name").value("John Doe"));
}
```

5.Melhoria de Interface

Falta feedback visual em interações no frontend. Sugestão: usar loaders e validações no formulário.

Exemplo Atual (Login):

O botão de login não dá feedback ao ser pressionado.

Melhoria com React e CSS:

```
<button disabled={isLoading}>
  {isLoading ? "Logging in..." : "Login"}
</button>
```

Essas mudanças aumentam escalabilidade, usabilidade e confiabilidade do sistema.

Considerações Finais

O repositório apresenta uma boa organização e qualidade visual. Contudo, ajustes no README e na documentação do front-end (caso aplicável) poderiam elevar o padrão profissional do sistema.