UNIVERSIDADE FEDERAL DO CEARÁ

25, Julho 2025

Sistemas Distribuídos - 01A - 2025.1

Trabalho 3 - EJB_WS_API

Alisson Rodrigues Fernandes - 510357

Monalisa Silva Bezerra - 535614

Professor: Dr. Antônio Rafael Braga

1. Introdução

Este trabalho consiste em reimplementar um serviço de gerenciamento de reservas de passagens aéreas, originalmente feita via RMI/sockets, agora implementado em um protocolo de requisição-resposta sobre HTTP. Para isso, foi utilizada a tecnologia Spring Boot, expondo um conjunto de endpoints REST e um cliente Java que consome essas APIs.

2. Objetivos

1. Migrar a comunicação
2. Definir um protocolo HTTP
3. Garantir serialização JSON
4. Manter ou melhorar a lógica de negócio
5. Implementação de testes automatizados

3. Arquitetura e Implementação

3.1 Projeto Spring Boot

Dependências principais

- `spring-boot-starter-web`: fornece Tomcat embutido, Jackson e anotações REST (`@RestController`, `@RequestMapping`, etc.).
- `spring-boot-starter-test` (escopo `test`): JUnit 5, Spring Test, Mockito.

Pacotes

- `model`: classes `Voo`, `Passagem` (e subclasses).
- `service`: `PassagemService` com mapa concorrente de voos e métodos de negócio.
- `controller`: `PassagemController`, expõe endpoints:
  - `POST /api/passagens/primeira`
  - `POST /api/passagens/economica`
  - `DELETE /api/passagens?numeroVoo=…&codigo=…`
  - `PUT /api/passagens/transferir`
- **DTOs internos** (requests) definidos como `public static class` em `PassagemController` para serialização.

3.2 Cliente Java

- Classe `PassagemClient` utiliza `RestTemplate` para invocar cada endpoint, passando objetos DTO e recebendo instâncias de `PassagemPrimeiraClasse` ou `PassagemClasseEconomica`.

3.3 Testes

- `PassagensApiApplicationTests`: verificação de carga do contexto Spring.
- Execução por `./mvnw test`, garantiu que o contexto carregue sem erros de dependência.

4. Resultados Obtidos

API funcional

- Todos os endpoints respondem corretamente a requisições JSON e retornam objetos com as informações esperadas.

Cliente de exemplo

- Capaz de reservar, cancelar e transferir passagens por meio de chamadas HTTP, sem uso de RMI ou sockets.

Testes de contexto

- Comprovação de que o projeto configura corretamente o Spring Boot e as dependências necessárias.

Documentação mínima:

- O `README.md` pode ser facilmente estendido com instruções de uso e exemplos de `curl` ou Postman.

5. Dificuldades encontradas

Classpath e escopos Maven

- Inicialmente, o starter-web foi declarado com `<scope>test</scope>`, impedindo a resolução das anotações em tempo de execução.

Viabilidade de DTOs

- Os tipos internos do controller estavam sem `public`, tornando-se inacessíveis ao cliente. A solução foi declará-los como `public static class` ou movê-los para pacote de DTOs.

Sincronização de versões

- Uso de Java 21 com Spring Boot 3.5.3 exigiu alinhamento das versões de plugin Maven e do parent POM.

Configuração do VS Code

- Foi necessário atualizar o projeto Maven e limpar o workspace da linguagem Java para reconhecer corretamente novas dependências.

6. Detalhes adicionais da implementação

Concorrência

- O repositório `Map<String,Voo>` em `PassagemService` é um `ConcurrentHashMap`, permitindo acesso simultâneo em cenários multithread.

Serialização

- Jackson converte automaticamente as classes DTO e os modelos de passagem em JSON, dispensando configurações extras.

Tratamento de erros

- Neste escopo inicial, falhas como “voo não encontrado” geram `NullPointerException`; recomenda-se, em uma versão futura, lançar `ResponseStatusException` com código `404` ou `400`.

Extensibilidade

- Adicionar persistence (JPA + banco de dados) apenas requer adicionar o starter JPA e anotar as entidades.
- Segurança (Spring Security) pode ser integrada para proteger endpoints.

7. Conclusão e Próximos Passos

Este trabalho alcançou a migração completa de um serviço distribuído legado para uma API REST moderna, alinhada às boas práticas do ecossistema Spring. Para evolução, sugerem-se:

- **Cobertura de testes**: criar testes de integração que efetivamente disparam requisições HTTP contra o servidor em memória (usando `MockMvc` ou WebTestClient).
- **Validações**: usar anotações `@Valid` e classes de DTO com `@NotNull`, `@Size` etc.
- **Tratamento de exceções**: definir `@ControllerAdvice` para mapear erros a respostas padronizadas.
- **Documentação**: integrar Swagger/OpenAPI (via `springdoc-openapi-ui`) para documentação interativa dos endpoints.