UNIVERSIDADE FEDERAL DO CEARÁ

Relatório do Trabalho 3 – Apicultura API

Juan Pablo Rudino Mesquita - 509982
Professor: Dr. Antônio Rafael Braga

26 de julho de 2025

1 Introdução

Este relatório descreve o desenvolvimento de um sistema de gerenciamento de apiários utilizando uma arquitetura cliente-servidor baseada em API REST, onde é aplicado o protocolo HTTP. O projeto foi implementado conforme os requisitos do documento do trabalho 3, que exigia apliicar comunicação via Web Services ou Application Programming Interface,usando como referencia no prejeto anterior, que utilizava de RMI.

2 Arquitetura do Sistema

Como o nosso objetivo é aprimorar o projeto onde substuiremos o RMI, o sistema novo é composto por dois componentes principais:

2.1 Servidor (Backend)

- Linguagem: Java com Spring Boot
- Entidades: `Apicultor`, `Colmeia`
- Endpoints REST:
  - `POST /colmeia_new` – Cria uma nova colmeia
  - `GET /colmeia_list` – Lista todas as colmeias
  - `DELETE /colmeia_add_abelhas` – Adiciona abelhas e define se há rainha ou não
  - `PUT /colmeia_del/{id}` – Remove Colmeia

2.2 Cliente (Frontend)

- Linguagem: Python / Javascript / c
- Biblioteca: `requests` para chamadas HTTP
- Funcionalidades:
  - Cadastro de colmeias
  - Consulta de colmeias
  - Adição de abelhas
  - Remoção de colmeias

3 Implementação

Com o servidor implementado em Java e framework Spring Boot, segue trechos do código principal onde representa cada parte

3.1 Camada de Modelo (Entity)

Representa as entidades do domínio da apicultura, onde representa a estrutura de dados e regras do dominio do sistema.Também é responsavel para o mapeamento do projeto. Segue um trecho do arquivo da camada de modelo colmeia.java abaixo:

```java
@Entity
public class Colmeia {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private int capacidadeAbelhas;
    private int abelhas = 0;
    private boolean rainha_presente = false;

    private int capacidadeMel;

    @ManyToOne
    @JoinColumn(name = "apicultor_id")
    private Apicultor apicultor;

    // Getters e Setters
}
```

Funcionalidade no projeto:

- Mapeia a tabela ‘colmeia‘ no banco H2
- Define relacionamento com ‘Apicultor‘ (N colmeias para 1 apicultor)

3.2 2. Camada de Repositório (Repository)

Projetada para simplificar drasticamente as operações de persistência (CRUD) e consultas aos dados das colmeias e apicultores. Interface para operações CRUD:

```
public interface ColmeiaRepository extends JpaRepository<
    Colmeia, Long> {
    List<Colmeia> findByApicultor(Apicultor apicultor);
}
```

Funcionalidade no projeto:

- Herda métodos básicos (‘save‘, ‘findAll‘, ‘delete‘)
- Consulta personalizada para listar colmeias por apicultor

3.3 3. Camada de Serviço (Service)

Contém a lógica de negócio, atuando para as requisições do protocolo HTTP e o banco de dados H2:

```
@Service
public class ApiarioService {

    @Autowired
    private ApicultorRepository apicultorRepo;

    @Autowired
    private ColmeiaRepository colmeiaRepo;

    public Colmeia criarColmeia(String nomeApicultor, int
    capacidadeAbelhas, int capacidadeMel) {

        Apicultor apicultor = apicultorRepo.findByNome(
    nomeApicultor);

        if (apicultor == null) {
            apicultor = new Apicultor();
            apicultor.setNome(nomeApicultor);
            apicultor = apicultorRepo.save(apicultor);
        }
        Colmeia colmeia = new Colmeia();
        colmeia.setCapacidadeAbelhas(capacidadeAbelhas);
```

```
        colmeia.setCapacidadeMel(capacidadeMel);
        colmeia.setApicultor(apicultor);
        return colmeiaRepo.save(colmeia);
    }
```

Funcionalidade no projeto:

- Valida capacidade máxima de abelhas
- Verifica existência do apicultor
- Conversão DTO → Entity

3.4 4. Camada de Controlador (Controller)

Expõe os endpoints REST,sendo a porta de entrada da API REST no spring-boot para a comunicação do usuario e das outras camadas:

```
@RestController
@RequestMapping("/api")
public class ApiarioController {
    @Autowired
    private ApiarioService apiarioService;

    @PostMapping("/colmeia_new")
    public Colmeia criarColmeia(@RequestParam String nomeApicultor,
                                @RequestParam int capacidadeAbelhas,
                                @RequestParam int capacidadeMel) {
        return apiarioService.criarColmeia(nomeApicultor, capacidadeAbelhas, capacidadeMel);
    }

    @GetMapping("/colmeia_list")
    public List<Colmeia> listarColmeias(@RequestParam String nomeApicultor) {
        return apiarioService.listarColmeias(nomeApicultor);
    }
    // ...
}
```

Funcionalidade no projeto:

- Recebe requisições HTTP (POST, GET, etc.)
- Retorna status code apropriados (200 OK, 404 Not Found)
- Mapeia JSON para objetos Java

3.5 application.properties no projeto

O arquivo de configuração application.properties foi essencial para definir o ambiente de execução da aplicação, especialmente durante a fase de desenvolvimento e testes. Temos como suas configurações principais:

- Facilidade de configuração: Eliminou a necessidade de configurar um banco de dados tradicional (como MySQL ou PostgreSQL) durante o desenvolvimento.
- Isolamento de testes: Garantiu que os testes unitários e integração não afetassem um banco de dados real.
- Banco de dados rápido e efêmero: Ideal para os testes, pois não exigir instalação externa e é reiniciado a cada execução.
- Uso do log de SQL no console: Atualização automática do esquema: A opção ddl-auto=update atualizou as tabelas do banco automaticamente quando as entidades eram modificadas, acelerando o desenvolvimento.
- Acesso remoto: Permitiu que a API fosse acessada por outras máquinas na rede.

3.6 Clientes e Interpolaridade

O cliente foi desenvolvido em diferentes linguagesn de programação para demostrar o quao flexivel é o projeto em relação ao anterior de RMI que apeas funcionaria em java. Em seguida vemos em testes em Python, javascript e em c para demonstrar a interoperabilidade do sistema. Os arquivos frontend segem a mesma logica de programação, onde possuem um menu de escolha de qual serviço irá ser chamado.

4 Requisitos e Resultados Atendidos

O projeto atende todos os requisitos especificados, onde:

- Comunicação via API REST (HTTP/JSON), sendo capaz de fazer a comunicação
- Não utiliza sockets ou RMI, obtendo um maior nivel de abstração.
- Interoperabilidade (Java + Python + C), onde apesar das discrepancias de linguagens, elas são capazes de interagir com o sistema.

- Protocolo requisição/resposta. Todos os testes de cada chamada de serviço retorna o que é esperado.
- Organização em repositório único

5 Conclusão

O sistema desenvolvido demonstra com sucesso a aplicação de Web Services utilizando uma API RESTful. A arquitetura escolhida proporciona flexibilidade, interoperabilidade e facilidade de manutenção, atendendo plenamente aos objetivos do trabalho. Como trabalho futuro, sugere-se a implementação de autenticação JWT e um frontend web.