UNIVERSIDADE FEDERAL DO CEARÁ

Relatório do Trabalho 3 – Apicultura API

Juan Pablo Rudino Mesquita
Universidade Federal do Ceará

25 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. O projeto foi implementado conforme os requisitos do Trabalho 3, que exigia comunicação via Web Services ou Application Programming Interface, sem o uso de sockets ou RMI.

2 Arquitetura do Sistema

O sistema 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}` –[1]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

3.1 Servidor Spring Boot

O servidor foi implementado em Java utilizando o framework Spring Boot. Segue um trecho do código principal:

```
package com.example.apiario.controller;

import com.example.apiario.model.Colmeia;
import com.example.apiario.service.ApiarioService;
import org.springframework.beans.factory.annotation.Autowired
    ;
import org.springframework.web.bind.annotation.*;
import java.util.List;

@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);
    }

```

```
22      @GetMapping("/colmeia_list")
23      public List<Colmeia> listarColmeias(@RequestParam String
    nomeApicultor) {
24          return apiarioService.listarColmeias(nomeApicultor);
25      }
26
27      @DeleteMapping("/colmeia_del/{id}")
28      public void deletarColmeia(@PathVariable Long id) {
29          apiarioService.deletarColmeia(id);
30      }
31
32      @PostMapping("/colmeia_add_abelhas")
33      public Colmeia adicionarAbelhas(@RequestParam Long
    idColmeia,@RequestParam int quantidade,@RequestParam
    boolean rainhaPresente) {
34
35          return apiarioService.AdicionarAbelhas(idColmeia,
    quantidade, rainhaPresente);
36      }
37  }
```

3.2 Clientes e Interpolaridade

O cliente foi desenvolvido em Python, javascript e em c para demonstrar a interoperabilidade do sistema:

3.3 Cliente em Python

```
1  import requests
2
3  BASE_IP = "192.168.0.102"  # IP do servidor
4  BASE_URL = f"http://{BASE_IP}:8080/api"
5
6  def criar_colmeia(nome_apicultor, capacidade_abelhas,
    capacidade_mel):
7      response = requests.post(
8          f"{BASE_URL}/colmeia_new",
9          params={
10             "nomeApicultor": nome_apicultor,
11             "capacidadeAbelhas": capacidade_abelhas,
12             "capacidadeMel": capacidade_mel
13         }
14     )
15     return response.json()
16
17 def listar_colmeias(nome_apicultor):
18     response = requests.get(
```

```
        f"{BASE_URL}/colmeia_list",
        params={"nomeApicultor": nome_apicultor}
    )
    return response.json()

def adicionar_abelha(id_colmeia, quantidade, rainha_presente):
    response = requests.post(
        f"{BASE_URL}/colmeia_add_abelhas",
        params={
            "idColmeia": id_colmeia,
            "quantidade": quantidade,
            "rainhaPresente": rainha_presente
        }
    )
    return response.json()

def remover_colmeia(id_colmeia):
    response = requests.delete(
        f"{BASE_URL}/colmeia_remove",
        params={"idColmeia": id_colmeia}
    )
    return response.json()

def menu():
    print("\n{ MENU }")
    print("{1} Criar colmeia")
    print("{2} Listar colmeias")
    print("{3} Adicionar abelhas")
    print("{4} Remover colmeia")
    print("{0} Sair")

if __name__ == "__main__":
    while True:
        menu()
        escolha = input("{Escolha uma op   o}: ")
        if escolha == "1":
            nome = input("{Nome do apicultor}: ")
            cap_abelhas = int(input("{Capacidade de abelhas}: "))
            cap_mel = int(input("{Capacidade de mel}: "))
            resultado = criar_colmeia(nome, cap_abelhas, cap_mel)
            print("{Resultado}: ", resultado)
        elif escolha == "2":
            nome = input("{Nome do apicultor}: ")
            resultado = listar_colmeias(nome)
            print("{Colmeias}: ", resultado)
        elif escolha == "3":
```

```
            id_colmeia = int(input("{ID da colmeia}: "))
            quantidade = int(input("{Quantidade de abelhas}:
    "))
            rainha = input("{Rainha presente? (true/false)}:
    ").lower() == "true"
            resultado = adicionar_abelha(id_colmeia,
    quantidade, rainha)
            print("{Resultado}: ", resultado)
        elif escolha == "4":
            id_colmeia = int(input("{ID da colmeia}: "))
            resultado = remover_colmeia(id_colmeia)
            print("{Resultado}: ", resultado)
        elif escolha == "0":
            print("{Saindo...}")
            break
        else:
            print("{Op   o
```

3.4 Cliente em Javascript

```
// cliente_node.js
const axios = require('axios');

const BASE_URL = "http://192.168.0.102:8080/api";

// Criar colmeia
async function criarColmeia(nomeApicultor, capacidadeAbelhas,
    capacidadeMel) {
    try {
        const response = await axios.post(`${BASE_URL}/
    colmeia_new`, null, {
            params: {
                nomeApicultor,
                capacidadeAbelhas,
                capacidadeMel
            }
        });
        return response.data;
    } catch (error) {
        console.error("Erro ao criar colmeia:", error.message
    );
        if (error.response) {
            console.error("Detalhes:", error.response.data);
        }
    }
}

// Listar colmeias
```

```
async function listarColmeias(nomeApicultor) {
    try {
        const response = await axios.get(`${BASE_URL}/colmeia_list`, {
            params: { nomeApicultor }
        });
        return response.data;
    } catch (error) {
        console.error("Erro ao listar colmeias:", error.message);
        if (error.response) {
            console.error("Detalhes:", error.response.data);
        }
    }
}

// Adicionar abelha
async function adicionarAbelha(idColmeia, quantidade, rainhaPresente) {
    try {
        const response = await axios.post(`${BASE_URL}/colmeia_add_abelhas`, null, {
            params: {
                idColmeia,
                quantidade,
                rainhaPresente
            }
        });
        return response.data;
    } catch (error) {
        console.error("Erro ao adicionar abelha:", error.message);
        if (error.response) {
            console.error("Detalhes:", error.response.data);
        }
    }
}

// remover colmeia
async function removerColmeia(idColmeia) {
    try {
        const response = await axios.delete(`${BASE_URL}/colmeia_del/${idColmeia}`);
        return response.data || "Colmeia removida com sucesso";
    } catch (error) {
        return handleError(error, "remover colmeia");
    }
}
```

```
68
69
70 // Teste
71 (async () => {
72     console.log("Criando colmeia...");
73     const colmeia = await criarColmeia("Guilherme", 1500, 75)
       ;
74     console.log("Nova colmeia:", colmeia);
75
76     console.log("\nListando colmeias...");
77     const colmeias = await listarColmeias("Guilherme");
78     console.log("Colmeias de Guilherme:", colmeias);
79
80     console.log("\nAdicionando abelha...");
81     const abelha = await adicionarAbelha(1, 10, true);
82     console.log("Abelha adicionada:", abelha);
83
84     console.log("\nRemovendo colmeia...");
85     const resultadoRemocao = await removerColmeia(1);
86     console.log("Resultado da remo   o:", resultadoRemocao);
87
88 })();
```

3.5 Cliente em C

```
1  #include <stdio.h>
2  #include <stdbool.h>
3  #include <curl/curl.h>
4  #include <string.h>
5
6  #define API_URL "http://192.168.0.102:8080/api"
7
8  size_t write_callback(void *contents, size_t size, size_t
       nmemb, void *userp) {
9      size_t total_size = size * nmemb;
10     printf("%.*s", (int)total_size, (char *)contents);
11     return total_size;
12 }
13
14 void criar_colmeia(const char *nome_apicultor, int
       capacidade_abelhas, int capacidade_mel) {
15     CURL *curl = curl_easy_init();
16     if (curl) {
17         char url[256];
18         snprintf(url, sizeof(url), "%s/colmeia_new?
       nomeApicultor=%s&capacidadeAbelhas=%d&capacidadeMel=%d",
19                  API_URL, nome_apicultor, capacidade_abelhas,
        capacidade_mel);
```

```
20
21          curl_easy_setopt(curl, CURLOPT_URL, url);
22          curl_easy_setopt(curl, CURLOPT_POST, 1L);
23          curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
    write_callback);
24          curl_easy_setopt(curl, CURLOPT_HTTP_VERSION,
    CURL_HTTP_VERSION_1_1);
25          curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "");
26          curl_easy_setopt(curl, CURLOPT_TIMEOUT, 5L);
27
28          struct curl_slist *headers = NULL;
29          headers = curl_slist_append(headers, "Expect:");
30          curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
31
32          printf("{Enviando POST para criar colmeia...}\n");
33          CURLcode res = curl_easy_perform(curl);
34          if (res != CURLE_OK) {
35              fprintf(stderr, "{Erro ao criar colmeia: %s}\n",
    curl_easy_strerror(res));
36          }
37          curl_slist_free_all(headers);
38          curl_easy_cleanup(curl);
39      }
40  }
41
42  void listar_colmeias(const char *nome_apicultor) {
43      CURL *curl = curl_easy_init();
44      if (curl) {
45          char url[256];
46          snprintf(url, sizeof(url), "%s/colmeia_list?
    nomeApicultor=%s", API_URL, nome_apicultor);
47
48          curl_easy_setopt(curl, CURLOPT_URL, url);
49          curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
    write_callback);
50
51          printf("{Enviando GET para listar colmeias...}\n");
52          CURLcode res = curl_easy_perform(curl);
53          if (res != CURLE_OK) {
54              fprintf(stderr, "{Erro ao listar colmeias: %s}\n"
    , curl_easy_strerror(res));
55          }
56          curl_easy_cleanup(curl);
57      }
58  }
59
60  void adicionar_abelha(int id_colmeia, int quantidade, bool
    rainha_presente) {
61      CURL *curl = curl_easy_init();
```

```
62      if (curl) {
63          char url[256];
64          const char *rainha_str = rainha_presente ? "true" : "
    false";
65          snprintf(url, sizeof(url), "%s/colmeia_add_abelhas?
    idColmeia=%d&quantidade=%d&rainhaPresente=%s",
66                   API_URL, id_colmeia, quantidade, rainha_str)
    ;
67
68          curl_easy_setopt(curl, CURLOPT_URL, url);
69          curl_easy_setopt(curl, CURLOPT_POST, 1L);
70          curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
    write_callback);
71          curl_easy_setopt(curl, CURLOPT_HTTP_VERSION,
    CURL_HTTP_VERSION_1_1);
72          curl_easy_setopt(curl, CURLOPT_POSTFIELDS, "");
73          curl_easy_setopt(curl, CURLOPT_TIMEOUT, 5L);
74
75          struct curl_slist *headers = NULL;
76          headers = curl_slist_append(headers, "Expect:");
77          curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);
78
79          printf("{Enviando POST para adicionar abelhas...}\n")
    ;
80          CURLcode res = curl_easy_perform(curl);
81          if (res != CURLE_OK) {
82              fprintf(stderr, "{Erro ao adicionar abelhas: %s}\
    n", curl_easy_strerror(res));
83          }
84          curl_slist_free_all(headers);
85          curl_easy_cleanup(curl);
86      }
87  }
88
89  void remover_colmeia(int id_colmeia) {
90      CURL *curl = curl_easy_init();
91      if (curl) {
92          char url[256];
93          snprintf(url, sizeof(url), "%s/colmeia_remove?
    idColmeia=%d", API_URL, id_colmeia);
94
95          curl_easy_setopt(curl, CURLOPT_URL, url);
96          curl_easy_setopt(curl, CURLOPT_CUSTOMREQUEST, "DELETE
    ");
97          curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION,
    write_callback);
98          curl_easy_setopt(curl, CURLOPT_TIMEOUT, 5L);
99
100         printf("{Enviando DELETE para remover colmeia...}\n")
```

```
    ;
101         CURLcode res = curl_easy_perform(curl);
102         if (res != CURLE_OK) {
103             fprintf(stderr, "{Erro ao remover colmeia: %s}\n"
    , curl_easy_strerror(res));
104         }
105         curl_easy_cleanup(curl);
106     }
107 }
108 
109 void menu() {
110     printf("\n{ MENU }\n");
111     printf("{1} Criar colmeia\n");
112     printf("{2} Listar colmeias\n");
113     printf("{3} Adicionar abelhas\n");
114     printf("{4} Remover colmeia\n");
115     printf("{0} Sair\n");
116 }
117 
118 int main() {
119     curl_global_init(CURL_GLOBAL_ALL);
120 
121     int escolha;
122     while (1) {
123         menu();
124         printf("{Escolha uma op   o}: ");
125         if (scanf("%d", &escolha) != 1) break;
126         getchar(); // Limpa o buffer do teclado
127 
128         if (escolha == 1) {
129             char nome[100];
130             int cap_abelhas, cap_mel;
131             printf("{Nome do apicultor}: ");
132             fgets(nome, sizeof(nome), stdin);
133             nome[strcspn(nome, "\n")] = 0;
134             printf("{Capacidade de abelhas}: ");
135             scanf("%d", &cap_abelhas);
136             printf("{Capacidade de mel}: ");
137             scanf("%d", &cap_mel);
138             getchar();
139             criar_colmeia(nome, cap_abelhas, cap_mel);
140         } else if (escolha == 2) {
141             char nome[100];
142             printf("{Nome do apicultor}: ");
143             fgets(nome, sizeof(nome), stdin);
144             nome[strcspn(nome, "\n")] = 0;
145             listar_colmeias(nome);
146         } else if (escolha == 3) {
147             int id_colmeia, quantidade;
```

```
148             char rainha[10];
149             bool rainha_presente;
150             printf("{ID da colmeia}: ");
151             scanf("%d", &id_colmeia);
152             printf("{Quantidade de abelhas}: ");
153             scanf("%d", &quantidade);
154             getchar();
155             printf("{Rainha presente? (true/false)}: ");
156             fgets(rainha, sizeof(rainha), stdin);
157             rainha[strcspn(rainha, "\n")] = 0;
158             rainha_presente = (strcmp(rainha, "true") == 0);
159             adicionar_abelha(id_colmeia, quantidade,
    rainha_presente);
160         } else if (escolha == 4) {
161             int id_colmeia;
162             printf("{ID da colmeia}: ");
163             scanf("%d", &id_colmeia);
164             getchar();
165             remover_colmeia(id_colmeia);
166         } else if (escolha == 0) {
167             printf("{Saindo...}\n");
168             break;
169         } else {
170             printf("{Op   o inv lida!}\n");
171         }
172     }
173 
174     curl_global_cleanup();
175     return 0;
176 }
```

4 Requisitos Atendidos

O projeto atende todos os requisitos especificados:

- Comunicação via API REST (HTTP/JSON)
- Não utiliza sockets ou RMI
- Interoperabilidade (Java + Python)
- Protocolo requisição/resposta
- 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.

Referências

[1] Spring Boot Documentation.
`https://spring.io/projects/spring-boot`

[2] H2 Database Engine.
`http://www.h2database.com`