API WARS

A ORDEM DOS DADOS DESPERTA

LUANDERSON ARLINDO

Prefácio do Autor

Se você está lendo este livro, saiba de uma coisa: você já faz parte da Aliança do Código.

Meu nome é Luanderson Arlindo, e assim como você, um dia olhei para uma tela preta com letras brancas e me perguntei: “Como isso tudo realmente funciona?” Foi nesse momento que percebi que programar não era apenas escrever comandos — era criar universos. APIs são as naves que conectam esses mundos. Estruturas de dados, os sistemas de propulsão. E os algoritmos de ordenação? O que mantém tudo em harmonia.

Este livro nasceu da vontade de mostrar que sim, é possível aprender programação com profundidade mesmo começando do zero, sem perder o senso de diversão e aventura.

Prepare-se para sair da superfície do "Ctrl + C / Ctrl + V" e mergulhar em um código com propósito. Juntos, vamos construir sua primeira API RESTful em Java, entender como os dados são armazenados e manipulados, e aplicar algoritmos reais para ordená-los. Tudo isso com analogias, exemplos didáticos e uma pitada da Força.

Que o código esteja com você. Sempre.

Introdução – Como Usar Este Livro

Este e-book foi feito para guiar iniciantes em programação, com foco em APIs REST usando Java e Spring Boot, estruturado de forma progressiva. Aqui vai um mapa para aproveitar ao máximo:

Para quem é este livro?

- Estudantes de programação
- Iniciantes em Java
- Desenvolvedores que desejam entender APIs, algoritmos e estruturas de dados de forma prática

O que você vai aprender?

- Conceitos fundamentais de APIs REST
- Como criar uma API com Spring Boot
- Como usar estruturas de dados em projetos reais
- Como implementar e aplicar algoritmos de ordenação
- Como persistir dados em um banco de dados com Spring Data JPA

Requisitos básicos

- Ter o Java instalado (versão 17 ou superior recomendada)
- IDE como Spring Tool Suite, Eclipse ou IntelliJ
- Conhecimento básico em lógica de programação (nada avançado)
- Vontade de codar e explorar

Ferramentas utilizadas

Java + Spring Boot

Postman para testes de API

H2 Database (banco de dados em memória)

IDE com suporte ao Spring (STS ou similar)

Estrutura dos capítulos

Cada capítulo traz:

- Contexto narrativo com referências à saga (para motivação!)
- Explicação teórica didática
- Exemplo prático com código
- Missão prática (atividades para fixação)

Dicas para aproveitar melhor:

- Teste todos os códigos você mesmo
- Use o Postman sempre que for apresentado um endpoint
- Brinque com os dados: adicione personagens, mude os algoritmos
- Marque seus avanços e retome de onde parou

Se tudo isso parece animador, ótimo — essa é a ideia. Chegou a hora de embarcar na nave API Wars, com sua IDE como sabre de luz, sua lógica como escudo e sua persistência como motor.

Vamos ao Capítulo 1?

Índice – API Wars: A Ordem dos Dados Desperta

Uma jornada didática por APIs REST, estruturas de dados e algoritmos de ordenação em Java com Spring Boot

- Armazenando dados com List
- Implementando ordenações manuais na API
- Organizando tudo com boas práticas

Parte III – A Ascensão da Persistência

8. Capítulo 6 – A Ascensão da Persistência: Integrando Banco de Dados com JPA
   - O que é JPA?
   - Usando banco H2 em memória
   - Tornando sua API persistente
   - Acessando o console do H2
   - Refatorando o código com repositórios

Parte IV – Holocrons do Conhecimento

- Apêndice A – Dicas de Produtividade para Devs Iniciantes
- Atalhos no Spring Tools
- Boas práticas de organização
- Evitando os erros mais comuns
- Apêndice B – Glossário Galáctico do Programador
- Termos essenciais explicados de forma simples
- Apêndice C – Próximos Passos na Galáxia Java
- JDBC, JPA avançado, Swagger, Testes Unitários
- Comunidades e conteúdos recomendados

Encerramento

9. Agradecimentos do Autor
   Palavras finais, créditos e reconhecimento ao aprendiz da Força Java.
10. Sobre o Autor: Luanderson Arlindo
    Uma breve apresentação da sua trajetória e contato profissional.

Capítulo 1 – Introdução: Um Padawan Chamado Dev

Há muito tempo, em um computador não muito distante...

Bem-vindo à Galáxia do Código

Você está prestes a iniciar sua jornada como desenvolvedor Jedi. Aqui, o sabre de luz é o Java, sua nave é o Spring Boot e seu objetivo é dominar os caminhos da Força das APIs REST, aprender a arte ancestral das estruturas de dados e a sabedoria dos algoritmos de ordenação.

Mas antes de empunhar seu teclado e enfrentar os desafios da galáxia, é preciso entender o que está por vir.

O Que Você Vai Aprender

Este e-book foi criado para guiar iniciantes em uma trilha completa e prática:

Como criar APIs RESTful com Java e Spring Boot

O que são estruturas de dados como List, Set e Map

Como funcionam os algoritmos de ordenação e quando usá-los

Como tudo isso se conecta em um único projeto funcional

Não se preocupe se alguns nomes parecerem estranhos no começo. Vamos usar analogias, explicações passo a passo e códigos comentados para garantir que você compreenda tudo.

Para Quem É Este E-book

Este material é ideal para quem:

Está começando com Java e quer construir projetos práticos

Deseja entender o funcionamento de uma API REST de verdade

Quer aplicar estruturas de dados e ordenação de forma útil e direta

Gosta de aprender com uma abordagem divertida e temática

Ferramentas da Jornada

Você vai usar:

Ferramenta	Função
Java (JDK 17+)	Linguagem principal
Spring Boot	Framework para criar APIs REST
Postman	Testar as rotas da API
IDE (Eclipse, IntelliJ ou Spring Tools)	Para codar com conforto

Nosso Projeto: A Base Rebelde

Você construirá uma API REST capaz de:

Armazenar dados em memória com estruturas de dados

Ordenar listas enviadas pelo usuário com algoritmos simples

Expor tudo isso em endpoints organizados e bem estruturados

Ao final, você terá um projeto completo funcionando, pronto para expandir.

A Missão Começa Agora

Agora que você já pegou seu sabre e entendeu a missão, é hora de dar o primeiro passo: construir sua primeira API RESTful com Java.

No próximo capítulo, vamos criar nosso projeto com Spring Boot e construir os primeiros endpoints REST – a base da nossa rebelião de código!

Que a Força (dos dados) esteja com você.

Capítulo 2 – Primeiros Passos com a Força (API REST em Java)

"A Força está em todos os endpoints que sabem responder."
– Mestre Devwan Kenobi

Objetivo do Capítulo

Criar um projeto Java com Spring Boot

Configurar a estrutura básica

Criar o primeiro endpoint REST (GET)

Testar no Postman ou navegador

Etapa 1: Criando o Projeto Spring Boot

Use o site https://start.spring.io:

- Project: Maven
- Language: Java
- Spring Boot: 3.1 ou superior
- Project Name: apiwars
- Group: com.apiwars
- Artifact: apiwars
- Name: apiwars
- Description: API Wars - A Ordem dos Dados Desperta
- Package Name: com.apiwars
- Packaging: Jar
- Java: 17+
- Dependências:
- Spring Web
- Spring Boot DevTools
- Lombok (opcional, mas recomendado)
- Spring Boot Actuator (opcional)
- Clique em Generate, extraia o .zip e abra no Eclipse ou Spring Tools.

Estrutura Inicial Esperada

```
src
 └── main
    └── java
       └── com.apiwars
          └── ApiwarsApplication.java
 └── resources
    ├── application.properties
    └── static / templates (vazio)
```

Etapa 2: Criando Seu Primeiro Endpoint

Crie uma nova classe chamada StatusController.java:

```
package com.apiwars.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class StatusController {

   @GetMapping("/status")
   public String status() {
      return "A Força está ativa!";
   }
}
```

Etapa 3: Rodando o Projeto

Vá até a classe ApiwarsApplication.java e clique com o botão direito → Run As > Java Application.

No console, você verá algo como:

Tomcat started on port(s): 8080 (http)

Abra o navegador e acesse:

http://localhost:8080/status

Você verá:

A Força está ativa!

Testando com Postman

Se preferir, use o Postman:

Método: GET

URL: http://localhost:8080/status

Resposta:

"A Força está ativa!"

Dica Jedi

Use @RestController para APIs que retornam JSON ou String direto.
Use @Controller quando quiser retornar HTML (não usaremos aqui).

Próximos Passos

Você agora já possui uma API REST funcionando! No próximo capítulo, vamos armazenar dados em memória com estruturas como List, Set e Map, e criar endpoints para manipulá-los.

Preparado para dominar os Guardiões do Código?
Capítulo 3: Os Guardiões do Código – Estruturas de Dados vem aí!

Capítulo 3 – Os Guardiões do Código: Estruturas de Dados

“Um desenvolvedor sem estrutura... é como um droide sem circuito.”
– Mestre Yavass

Objetivo do Capítulo

Entender o que são estruturas de dados

Conhecer List, Set e Map na prática

Criar endpoints REST que usam essas estruturas

Armazenar dados em memória (sem banco de dados)

O que são estruturas de dados?

Estruturas de dados são como caixas organizadoras. Elas nos ajudam a guardar e organizar dados na memória de forma eficiente.

No Java, temos várias opções. Vamos focar em 3 principais:

Estrutura	Exemplo do mundo real	Quando usar
List	Lista de passageiros	Quando importa a ordem dos dados
Set	Ingressos únicos de um evento	Quando não pode ter dados repetidos
Map	Dicionário (chave → valor)	Quando precisa buscar por uma chave

Criando uma API que usa List

Vamos simular uma lista de personagens rebeldes.

Classe Personagem.java

```
package com.apiwars.model;

public class Personagem {
    private String nome;
    private String planeta;

    // Construtores
    public Personagem() {}
    public Personagem(String nome, String planeta) {
        this.nome = nome;
        this.planeta = planeta;
```

```
    }

    // Getters e setters
    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }

    public String getPlaneta() { return planeta; }
    public void setPlaneta(String planeta) { this.planeta = planeta; }
}
```

Classe PersonagemController.java

```
package com.apiwars.controller;

import com.apiwars.model.Personagem;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

@RestController
@RequestMapping("/personagens")
public class PersonagemController {

    private List<Personagem> personagens = new ArrayList<>();

    @PostMapping
    public String adicionar(@RequestBody Personagem p) {
        personagens.add(p);
        return "Personagem adicionado com sucesso!";
    }

    @GetMapping
    public List<Personagem> listar() {
        return personagens;
```

```
    }
}
```

Testando com Postman

Adicionando personagem (POST):

Método: POST

URL: http://localhost:8080/personagens

Body (JSON):

```
{
  "nome": "Luke Skywalker",
  "planeta": "Tatooine"
}
```

Listando personagens (GET):

Método: GET

URL: http://localhost:8080/personagens

Extras com Set e Map (opcional para este capítulo)

Se desejar, podemos criar versões com:

Set<Personagem> para evitar duplicados

Map<String, Personagem> para acessar por nome (chave)

Quer que eu inclua essas variações ainda neste capítulo?

Conclusão

Agora você aprendeu como usar estruturas de dados reais para armazenar informações dentro da sua API REST. Sem banco de dados, mas com organização e lógica.

No próximo capítulo, vamos ordenar essa bagunça: ensinar algoritmos de ordenação em Java, e como usá-los com seus dados da API!

Preparado para o próximo nível?
Capítulo 4: A Ordem Se Estabelece – Algoritmos de Ordenação vem aí!

Capítulo 4 – A Ordem Se Estabelece: Algoritmos de Ordenação

“Em todo caos, há uma ordem esperando para ser implementada.”
– Mestre Sortwalker

Objetivo do Capítulo

Compreender o que são algoritmos de ordenação

Implementar os algoritmos Bubble Sort, Selection Sort e Merge Sort em Java

Usar esses algoritmos dentro de uma API

Criar uma rota /ordenar para receber dados desordenados e devolver ordenados

O Que São Algoritmos de Ordenação?

Algoritmos de ordenação são instruções que reorganizam dados em uma determinada ordem (geralmente crescente ou decrescente).

Exemplo real:

Você tem uma lista com os seguintes nomes:

["Leia", "Anakin", "Yoda", "Luke"]

E quer ordenar:

["Anakin", "Leia", "Luke", "Yoda"]

Principais Algoritmos

1. Bubble Sort – Simples e didático

```
public List<Integer> bubbleSort(List<Integer> lista) {
   List<Integer> ordenada = new ArrayList<>(lista);
   int n = ordenada.size();
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (ordenada.get(j) > ordenada.get(j + 1)) {
            int temp = ordenada.get(j);
            ordenada.set(j, ordenada.get(j + 1));
            ordenada.set(j + 1, temp);
         }
      }
   }
```

```
    return ordenada;
}
```

2. Selection Sort – Escolhe o menor a cada passo

```
public List<Integer> selectionSort(List<Integer> lista) {
    List<Integer> ordenada = new ArrayList<>(lista);
    int n = ordenada.size();
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++) {
            if (ordenada.get(j) < ordenada.get(minIdx)) {
                minIdx = j;
            }
        }
        int temp = ordenada.get(minIdx);
        ordenada.set(minIdx, ordenada.get(i));
        ordenada.set(i, temp);
    }
    return ordenada;
}
```

3. Merge Sort – Divide e conquista (mais eficiente)

```
public List<Integer> mergeSort(List<Integer> lista) {
    if (lista.size() <= 1) return lista;

    int meio = lista.size() / 2;
    List<Integer> esquerda = mergeSort(lista.subList(0, meio));
    List<Integer> direita = mergeSort(lista.subList(meio, lista.size()));

    return merge(esquerda, direita);
}

private List<Integer> merge(List<Integer> esquerda, List<Integer> direita) {
    List<Integer> resultado = new ArrayList<>();
    int i = 0, j = 0;
```

```
    while (i < esquerda.size() && j < direita.size()) {
        if (esquerda.get(i) < direita.get(j)) {
            resultado.add(esquerda.get(i++));
        } else {
            resultado.add(direita.get(j++));
        }
    }

    resultado.addAll(esquerda.subList(i, esquerda.size()));
    resultado.addAll(direita.subList(j, direita.size()));
    return resultado;
}
```

Criando um Endpoint /ordenar

Classe OrdenacaoController.java

```
package com.apiwars.controller;

import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/ordenar")
public class OrdenacaoController {

    @PostMapping("/bubble")
    public List<Integer> ordenarComBubble(@RequestBody List<Integer> numeros) {
        return bubbleSort(numeros);
    }

    private List<Integer> bubbleSort(List<Integer> lista) {
        List<Integer> ordenada = new ArrayList<>(lista);
        int n = ordenada.size();
        for (int i = 0; i < n - 1; i++) {
```

```
        for (int j = 0; j < n - i - 1; j++) {
            if (ordenada.get(j) > ordenada.get(j + 1)) {
                int temp = ordenada.get(j);
                ordenada.set(j, ordenada.get(j + 1));
                ordenada.set(j + 1, temp);
            }
        }
    }
    return ordenada;
  }
}
```

Testando no Postman

POST http://localhost:8080/ordenar/bubble

Body (raw JSON):

[8, 3, 2, 5, 1]

Resposta:

[1, 2, 3, 5, 8]

Pronto Para o Combate!

Agora você sabe como ordenar listas manualmente, sem depender de .sort() ou recursos prontos. Isso te dá uma base sólida para entrevistas, lógica de programação e para entender como funcionam bibliotecas modernas.

Próximo Capítulo

No Capítulo 5 – Unificando o Conselho: Estrutura + Algoritmo + API, você irá unir tudo: API REST + estruturas de dados + ordenação → em um projeto funcional e interativo!

Perfeito! Agora é hora de unificar o poder de tudo o que você aprendeu até aqui. No Capítulo 5 – Unificando o Conselho: Estrutura + Algoritmo + API, você vai criar uma API REST completa que recebe personagens, os armazena numa estrutura de dados e permite ordená-los por nome ou planeta, utilizando algoritmos de ordenação manuais.

Capítulo 5 – Unificando o Conselho: Estrutura + Algoritmo + API

“A verdadeira força de uma API está na harmonia entre lógica, estrutura e organização.”
– Mestre Algorithma

Objetivo do Capítulo

Unir estrutura de dados com ordenação

Criar endpoints para adicionar e listar personagens

Criar endpoints para ordenar personagens por nome ou planeta

Utilizar algoritmos de ordenação dentro dos endpoints

Estrutura do Projeto

Você terá:

Componente	Responsabilidade
Personagem	Modelo de dados
PersonagemController	Endpoints REST
PersonagemService	Lógica de negócio (estrutura + ordenação)

Classe Personagem.java

```
package com.apiwars.model;


public class Personagem {
    private String nome;
    private String planeta;


    public Personagem() {}


    public Personagem(String nome, String planeta) {
        this.nome = nome;
        this.planeta = planeta;
    }


    public String getNome() { return nome; }
    public void setNome(String nome) { this.nome = nome; }
```

```
    public String getPlaneta() { return planeta; }
    public void setPlaneta(String planeta) { this.planeta = planeta; }
}
```

Classe PersonagemService.java

```
package com.apiwars.service;

import com.apiwars.model.Personagem;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

@Service
public class PersonagemService {

    private List<Personagem> personagens = new ArrayList<>();

    public void adicionar(Personagem personagem) {
        personagens.add(personagem);
    }

    public List<Personagem> listar() {
        return personagens;
    }

    public List<Personagem> ordenarPorNome() {
        List<Personagem> copia = new ArrayList<>(personagens);
        bubbleSortPorNome(copia);
        return copia;
    }

    public List<Personagem> ordenarPorPlaneta() {
```

```
        List<Personagem> copia = new ArrayList<>(personagens);
        bubbleSortPorPlaneta(copia);
        return copia;
    }

    private void bubbleSortPorNome(List<Personagem> lista) {
        int n = lista.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (lista.get(j).getNome().compareToIgnoreCase(lista.get(j + 1).getNome()) > 0) {
                    Personagem temp = lista.get(j);
                    lista.set(j, lista.get(j + 1));
                    lista.set(j + 1, temp);
                }
            }
        }
    }

    private void bubbleSortPorPlaneta(List<Personagem> lista) {
        int n = lista.size();
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (lista.get(j).getPlaneta().compareToIgnoreCase(lista.get(j + 1).getPlaneta()) > 0) {
                    Personagem temp = lista.get(j);
                    lista.set(j, lista.get(j + 1));
                    lista.set(j + 1, temp);
                }
            }
        }
    }
}
```

Classe PersonagemController.java

```
package com.apiwars.controller;
```

```
import com.apiwars.model.Personagem;
import com.apiwars.service.PersonagemService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/personagens")
public class PersonagemController {

    @Autowired
    private PersonagemService service;

    @PostMapping
    public String adicionar(@RequestBody Personagem personagem) {
        service.adicionar(personagem);
        return "Personagem adicionado!";
    }

    @GetMapping
    public List<Personagem> listar() {
        return service.listar();
    }

    @GetMapping("/ordenar/nome")
    public List<Personagem> ordenarPorNome() {
        return service.ordenarPorNome();
    }

    @GetMapping("/ordenar/planeta")
    public List<Personagem> ordenarPorPlaneta() {
        return service.ordenarPorPlaneta();
```

```
    }
}
```

Testando no Postman

Adicionar personagem (POST)
http://localhost:8080/personagens

```
{
  "nome": "Obi-Wan",
  "planeta": "Stewjon"
}
```

Listar todos (GET)
http://localhost:8080/personagens

Ordenar por nome (GET)
http://localhost:8080/personagens/ordenar/nome

Ordenar por planeta (GET)
http://localhost:8080/personagens/ordenar/planeta

Missão Cumprida

Você agora criou uma API REST que:

Armazena dados com List
Usa algoritmos de ordenação manuais
Cria rotas organizadas para acesso, inclusão e ordenação

Capítulo 6 – A Ascensão da Persistência: Integrando Banco de Dados com JPA

"O conhecimento que persiste é aquele que foi gravado no banco da Força."
– Mestre Databius

Objetivo do Capítulo

Integrar banco de dados H2 ao seu projeto

Persistir personagens com Spring Data JPA

Visualizar dados no console do navegador

Eliminar o uso da List manual no Service

Passo 1: Dependências no pom.xml

Se ainda não adicionou, inclua:

```
<dependencies>
  <!-- Spring Web -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <!-- Spring Data JPA -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>

  <!-- Banco de Dados H2 -->
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
  </dependency>
</dependencies>
```

Passo 2: Configurações no application.properties

```
spring.datasource.url=jdbc:h2:mem:apiwars
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Passo 3: Tornar Personagem uma Entidade

```
package com.apiwars.model;

import jakarta.persistence.*;

@Entity
public class Personagem {

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

    private String nome;
    private String planeta;

    public Personagem() {}

    public Personagem(String nome, String planeta) {
        this.nome = nome;
        this.planeta = planeta;
```

```
    }

    // getters e setters...
}
```

Passo 4: Criar PersonagemRepository

```
package com.apiwars.repository;

import com.apiwars.model.Personagem;
import org.springframework.data.jpa.repository.JpaRepository;

public interface PersonagemRepository extends JpaRepository<Personagem, Long> {
}
```

Passo 5: Atualizar o PersonagemService

```
package com.apiwars.service;

import com.apiwars.model.Personagem;
import com.apiwars.repository.PersonagemRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class PersonagemService {

    @Autowired
    private PersonagemRepository repository;

    public void adicionar(Personagem personagem) {
        repository.save(personagem);
    }
```

```
    public List<Personagem> listar() {
        return repository.findAll();
    }

    public List<Personagem> ordenarPorNome() {
        List<Personagem> lista = repository.findAll();
        lista.sort((p1, p2) -> p1.getNome().compareToIgnoreCase(p2.getNome()));
        return lista;
    }

    public List<Personagem> ordenarPorPlaneta() {
        List<Personagem> lista = repository.findAll();
        lista.sort((p1, p2) -> p1.getPlaneta().compareToIgnoreCase(p2.getPlaneta()));
        return lista;
    }
}
```

Acesso ao Banco via Navegador

Acesse:
http://localhost:8080/h2-console

JDBC URL: jdbc:h2:mem:apiwars

Clique em Connect

Resultado Final

Sua API agora:

Armazena dados em memória no H2

Utiliza Spring Data JPA

Pode ordenar dados já persistidos

Permite testar tudo com o Postman ou navegador

Conclusão

Você criou uma API REST completa, com:

Estruturas de dados

Algoritmos de ordenação

Integração com banco de dados

Arquitetura limpa e didática