

Roadmap Java Backend: Linguagem, Frameworks e Arquitetura

EDUARDO TOSTE

Checklist completo e progressivo com tudo que você precisa saber para se destacar no mercado Java em 2025. Cada tópico vem acompanhado de uma breve explicação sobre sua importância, ajudando você a entender por que cada assunto é essencial para sua formação como desenvolvedor backend.

Fundamentos da Linguagem Java

Por que aprender isso? Dominar os fundamentos é o primeiro passo para escrever códigos confiáveis e eficientes. Aqui você aprende como a linguagem funciona de verdade, lidando com tipos de dados, controle de fluxo e estruturas essenciais.

- Tipos primitivos vs Wrappers (int vs Integer)
- Operadores lógicos, relacionais e aritméticos
- Controle de fluxo (if, switch, for, while, do-while)
- Diferença entre == e .equals()
- Diferença entre String, StringBuilder, StringBuffer
- Diferença entre double, float e BigDecimal
- Arrays (criação, iteração, ordenação)
- Varargs e sobrecarga de métodos
- Palavras-chave: final, static, this, super
- Modificadores de acesso: public, private, protected, default
- Package e organização de classes

Orientação a Objetos (OOP)

Por que aprender isso? OOP é a base da arquitetura Java. Esses conceitos ajudam a modelar o mundo real com código reutilizável, limpo e de fácil manutenção.

- Conceitos: Encapsulamento, Herança, Polimorfismo, Abstração
- Criação de classes, atributos e construtores
- Sobrecarga e sobrescrita de métodos (@Override)
- Uso e diferenças entre interface e abstract class
- Composição, associação e herança entre classes
- Princípios SOLID com exemplos práticos
- Design Patterns: Singleton, Factory, Strategy, Observer, Builder

Tratamento de Exceções

Por que aprender isso? Erros acontecem. Saber tratá-los é essencial para garantir estabilidade, boa experiência ao usuário e facilidade na identificação de problemas.

- Blocos try-catch-finally
- Diferença entre throw e throws
- Exceções verificadas vs não verificadas

- Criação de exceções personalizadas
- Uso do finally mesmo após return
- Tratamento global com `@ControllerAdvice`, `@ExceptionHandler`
- Padronização de erros com estrutura (timestamp, status, path, message)
- RFC 7807 (problem+json) — opcional

Collections e Generics

Por que aprender isso? A manipulação de dados em listas, mapas ou conjuntos é algo do dia a dia. Generics evitam erros e tornam seu código mais flexível e seguro.

- Interfaces: List, Set, Map, Queue
- Implementações: ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap, PriorityQueue
- Iteração com for, forEach, iterator
- Generics: <T>, <K, V>, métodos genéricos
- Ordenação com Comparator, Comparable
- Estruturas imutáveis com Collections.unmodifiableList()

Java Moderno (Java 8+)

Por que aprender isso? Os recursos modernos do Java trazem produtividade, legibilidade e performance. O mercado exige esse conhecimento.

- Lambda expressions
- Method references
- Streams API: filter, map, collect, sorted, distinct, limit
- Collectors.toList(), joining(), groupingBy(), counting()
- Optional com encadeamentos seguros
- API de datas: LocalDate, LocalDateTime, Period, Duration
- var (Java 10+)
- Record (Java 14+)
- Parallel Streams (e cuidados com concorrência)

Spring / Spring Boot

Por que aprender isso? Spring Boot é o padrão de desenvolvimento Java moderno. Ele acelera o desenvolvimento de aplicações robustas com uma arquitetura limpa e integrada.

⚙️ Fundamentos

- Estrutura de um projeto Spring Boot
- `@RestController`, `@GetMapping`, `@PostMapping`, etc.
- `@Service`, `@Repository`, `@Autowired`
- `@RequestBody`, `@PathVariable`, `@Valid`
- Separação clara entre camadas

📦 Spring Data JPA

- `@Entity`, `@Id`, `@GeneratedValue`
- Relacionamentos: `@OneToMany`, `@ManyToOne`, `@JoinColumn`
- Lazy vs Eager loading
- Query Methods, `@Query`, Projections
- Paginação e ordenação (Pageable, Sort)
- Specification API ou Criteria API

📋 Validações e Boas práticas

- Bean Validation (`@NotBlank`, `@Email`, etc.)
- `@ControllerAdvice` para erros globais
- DTO + Mapper (ModelMapper, MapStruct, manual)

- Padronização de resposta (`ResponseEntity<>`, DTO de erro)
- Separação por módulos ou package-by-feature

📁 Configurações

- `application.yml` e `application.properties`
- Perfis com `@Profile`
- `@ConfigurationProperties`
- Externalização com variáveis de ambiente

🧪 Testes Spring

- `@SpringBootTest`, `@DataJpaTest`
- `MockMvc` para endpoints
- `TestContainers` com PostgreSQL ou MongoDB
- Coverage com JaCoCo + badge no GitHub

Spring Security

Por que aprender isso? Segurança é indispensável. Com Spring Security você protege endpoints, implementa JWT, autenticação via OAuth2 e muito mais.

- Conceitos básicos de autenticação e autorização
- JWT: geração, validação e autenticação
- Refresh Token
- Filtros personalizados (`OncePerRequestFilter`)
- Proteção baseada em roles
- OAuth2 básico (Google, GitHub)

Banco de Dados

Por que aprender isso? Toda aplicação real precisa persistir dados. Saber modelar, consultar e versionar bancos é essencial.

- SQL: SELECT, INSERT, UPDATE, DELETE
- Cláusulas: WHERE, JOIN, GROUP BY, ORDER BY, HAVING
- INNER, LEFT, RIGHT JOIN
- Transações com `@Transactional`
- Banco em memória (H2)
- Versionamento com Flyway
- Indexação, performance e normalização (nível básico)
- MongoDB (NoSQL)

APIs REST

Por que aprender isso? REST é o padrão atual para integrações. Essencial para desenvolvimento backend.

- Verbos HTTP (GET, POST, PUT, PATCH, DELETE)
- Status codes: 200, 201, 204, 400, 401, 403, 404, 409, 500
- Versionamento de rotas
- Documentação com Swagger/OpenAPI
- DTOs e `ResponseEntity` corretamente usados
- OpenAPI com `@Schema`, `@Operation`, `@Tag`

Testes (JUnit + Mockito)

Por que aprender isso? Testes automatizados garantem que seu código funcione hoje e no futuro. São fundamentais em equipes ágeis.

- JUnit 5: @Test, @BeforeEach, assertEquals, assertThrows
- Mockito: @Mock, @InjectMocks, when, verify
- Testes de integração com @SpringBootTest
- MockMvc com validações
- Teste de exceções personalizadas
- Testes de repositório com banco real (ou TestContainers)

Docker e DevOps

Por que aprender isso? Facilita o deploy, testes e padronização de ambientes. Um diferencial enorme na prática.

- Dockerfile para apps Spring Boot
- Docker Compose com PostgreSQL
- Volumes e variáveis de ambiente
- .env + docker-compose override
- Deploy com JAR + container
- Conceitos básicos de CI/CD

AWS

Por que aprender isso? A nuvem é o ambiente padrão das aplicações modernas. Entender os principais serviços da AWS permite que você implante, integre e escale aplicações de forma profissional.

- Conceitos básicos de cloud (IaaS, PaaS, regiões, disponibilidade)
- EC2: criação e deploy de instâncias Linux com Java
- S3: upload e leitura de arquivos (via SDK Java)
- RDS (PostgreSQL): banco relacional gerenciado
- SQS: fila de mensagens com integração via Spring Cloud
- IAM: políticas e permissões básicas
- Secrets Manager ou Parameter Store
- Integração com Spring Boot (via starters, AWS SDK v2)
- Deploy automatizado (Elastic Beanstalk ou Docker + ECS)
- Monitoramento com CloudWatch

Git e GitHub

Por que aprender isso? Versionamento de código é obrigatório. Saber usar Git e colaborar em times mostra profissionalismo.

- Comandos: clone, add, commit, push, pull, checkout, merge
- Fluxo de branches (main, feature/*, hotfix/*)
- Resolução de conflitos simples
- Pull Requests e revisão de código
- Rebase interativo
- Conventional Commits
- README completo com instruções de execução

Arquitetura e Práticas Avançadas

Por que aprender isso? Aprofunda seu entendimento sobre manutenibilidade, separação de responsabilidades e padrões escaláveis.

- Clean Architecture / Hexagonal (nível introdutório)
- DDD básico: Entidade, Value Object, Aggregate, Repositório
- Camadas: Domain, Application, Infrastructure
- Eventos com ApplicationEventPublisher
- @Async e execução paralela
- @Cacheable e caching local
- Retry com Spring Retry ou Resilience4j

Extras para se destacar

Por que aprender isso? Funcionalidades práticas que mostram domínio além do básico e agregam valor real aos projetos.

- Agendamento com @Scheduled
- Envio de e-mail com JavaMailSender + Thymeleaf
- Integração com APIs reais (Clima, CEP, Google Calendar)
- Geração de PDF ou CSV
- Upload de arquivos
- Log estruturado com SLF4J e Logback
- API monitorada com Spring Actuator
- Projeto deployado publicamente (Render, Railway)

Links Úteis para Estudo

- 🔗 [Documentação oficial do Java \(Oracle\)](#)
- 🔗 [Baeldung – Tutoriais completos de Java e Spring](#)
- 🔗 [Spring.io – Documentação oficial + guias práticos](#)
- 🔗 [Refactoring.Guru – Design Patterns ilustrados](#)
- 🔗 [JUnit 5 – User Guide oficial](#)
- 🔗 [Mockito – Site oficial + exemplos](#)
- 🔗 [Testcontainers – Testes com containers reais](#)
- 🔗 [Docker – Guia de introdução oficial](#)
- 🔗 [Flyway – Documentação de versionamento de banco](#)
- 🔗 [SQLBolt – Curso interativo de SQL](#)
- 🔗 [AWS para Desenvolvedores Java](#)
- 🔗 [Git – Pro Git Book \(gratuito e em português\)](#)
- 🔗 [Spring Security – Referência oficial](#)
- 🔗 [OpenAPI \(Swagger\).com Spring](#)

📌 **Recomendação final:** mantenha 1 ou 2 projetos públicos no GitHub que demonstrem sua evolução, organização, testes, arquitetura e documentação. Isso é o que mais impressiona recrutadores técnicos.