

Começando certo com Spring e Java em 2024

...

Eduardo Rebola
@eduardorebola / erdanielli
1º Meetup Java Dev-PR

/whoami

- Primeiro contato com Java em 2003
 - SCJP em 2005
 - SCWCD em 2007
- Graduado em Ciência da Computação (2005)
 - Primeiro projeto: www.sgp.uem.br
- Professor temporário (DIN/UEM) 2005/6
- 2006/9 atuando em empresas de Maringá
- 2010/11 na ThoughtWorks (Porto Alegre)
- 2011/19 no Ministério da Educação (Sede/FNDE/INEP)
- 2020 (full remote) em startup do Vale do Silício
- 2021+ (full remote) em empresas de Brasília



Pular Resumo

- Spring Framework 1.0 lançado em 03/2004
 - Idade medieval do Java EE
 - EJB dominavam e assustavam
 - Cânticos em XML
- Ganhou destaque pela simplicidade e por não exigir Application Server
 - IBM WebSphere / Oracle WebLogic / RedHat JBoss / Apache Geronimo / Eclipse Glassfish
 - Subia como WAR file em qualquer servlet container (Tomcat / Jetty)
- Entregava os recursos Java EE implementado pela comunidade, bem como integrava as libs mais populares
 - Hibernate / iBatis / JasperReports
- Evoluía mais rápido que o Java EE
- Muito criticado pelo excesso de boilerplate, até que...



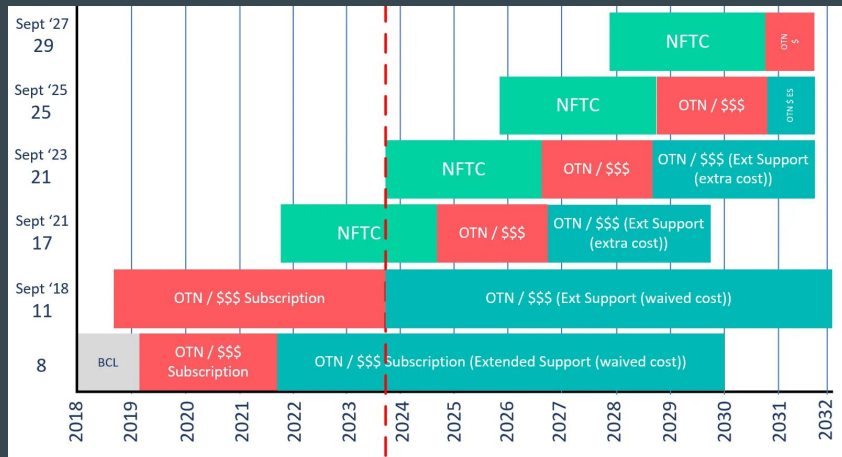
Pular Resumo

- Spring Boot 1.0 lançado em Abril/2014
- Zero XML, JAR executável (Tomcat/Jetty embutido).
- Disparou a popularidade do Spring Framework por torná-lo mais fácil de começar (convention over configuration).
 - Apache Maven também disparou em popularidade
- Java EE tentou alcançar, mas sucumbiu na versão 7 (2017). Oracle entregou pra Eclipse Foundation que rebatizou para Jakarta EE.
- Spring ainda se mantém fiel em integrar-se com as especificações mais recentes Jakarta EE
- Vários subprojetos Spring foram criados e são amplamente utilizados:
 - Spring Security
 - Spring Data
 - Spring Cloud



Pular Resumo

- 2004/Java 5: `@Annotations` (o começo do fim do XML)
- 2014/Java 8: `Instant`, `lambdas` e outros recursos de FP
- 2018/Java 11: `var`, módulos, `HttpClient`
- 2021/Java 17: `record`, `sealed types`, pattern matching, text blocks
- 2023/Java 21: `virtual threads`, + pattern matching, `Sequenced Collections`



Java morreu?

- Nunca esteve tão vivo!
- 2 entregas anuais (Março e Setembro) e versão LTS a cada 2 anos:
 - Próxima será 25 em 2025 !
- As restrições da Cloud colaboraram?
 - Project Valhalla
 - Virtual Threads (Loom)
- Tem espaço na IA?
 - Vector API
 - Project Panama (Foreign Function & Memory API)

Algumas coisas mudaram nesses 20 anos...

1. O ambiente de desenvolvimento:
 - Docker trouxe toda a infraestrutura necessária p/ a máquina do dev
 - SQL/NoSQL databases
 - Mensageria
 - Cloud stubs
2. O artefato entregável:
 - Imagem docker ao invés de um JAR
 - CaC (Configuration as Code)
3. Inteligência Artificial
 - Ajuda ou prejudica?

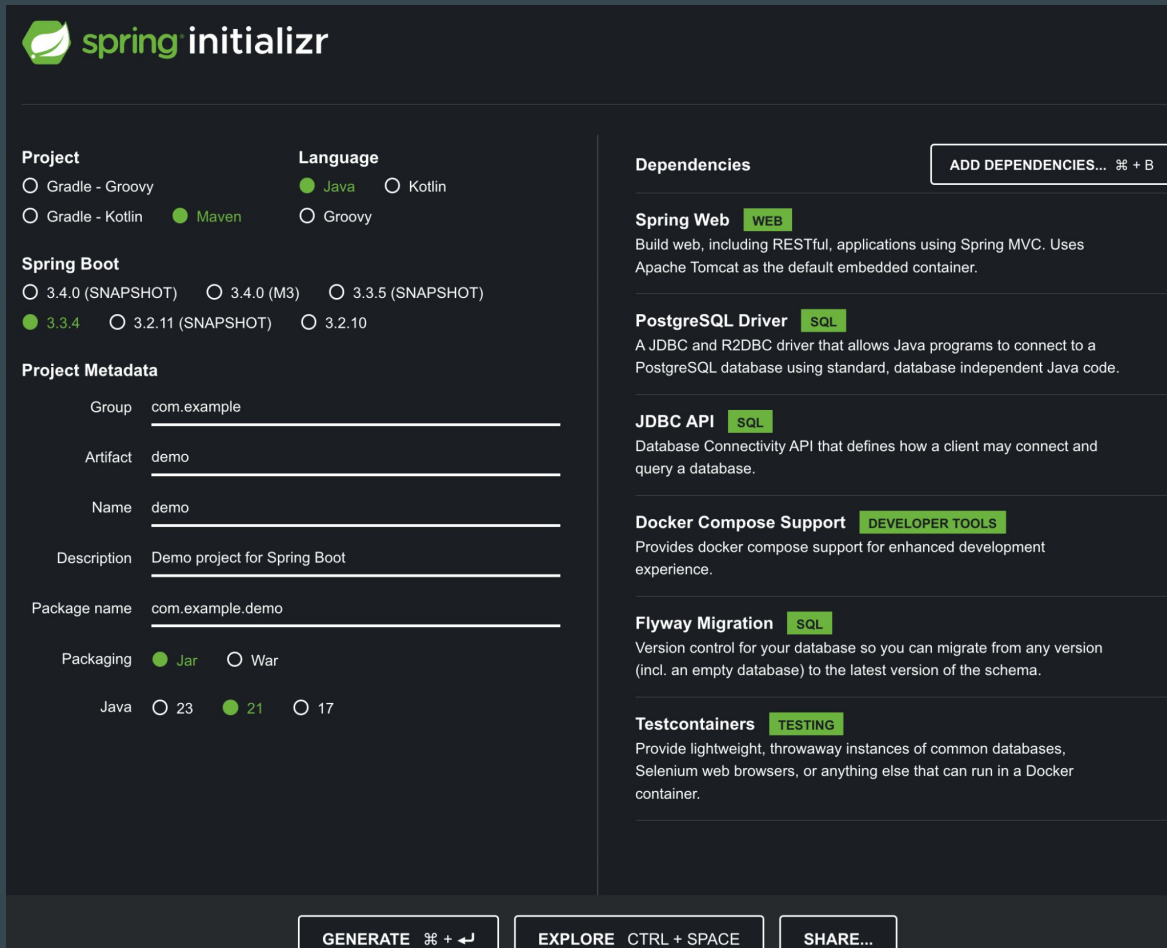
Começando pelo começo

<https://start.spring.io/>

Vamos criar uma pequena API REST de transação bancária utilizando PostgreSQL.

Serão 2 endpoints:

- POST /transferir
- POST /extrato



The image shows the Spring Initializr web application interface. It is a dark-themed form for generating a Spring project. The form is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and Testcontainers. The Project section has radio buttons for Gradle - Groovy, Gradle - Kotlin, and Maven (selected). The Language section has radio buttons for Java (selected), Kotlin, and Groovy. The Spring Boot section has radio buttons for 3.4.0 (SNAPSHOT), 3.4.0 (M3), 3.3.5 (SNAPSHOT), 3.3.4 (selected), 3.2.11 (SNAPSHOT), and 3.2.10. The Project Metadata section has input fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo). The Packaging section has radio buttons for Jar (selected) and War. The Java section has radio buttons for 23, 21 (selected), and 17. The Dependencies section has a button to add dependencies and lists several dependencies: Spring Web (WEB), PostgreSQL Driver (SQL), JDBC API (SQL), Docker Compose Support (DEVELOPER TOOLS), Flyway Migration (SQL), and Testcontainers (TESTING). At the bottom, there are buttons for GENERATE, EXPLORE, and SHARE...

Project

☐ Gradle - Groovy ☒ Gradle - Kotlin ☒ Maven

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.4.0 (SNAPSHOT) ☐ 3.4.0 (M3) ☐ 3.3.5 (SNAPSHOT) ☒ 3.3.4 ☐ 3.2.11 (SNAPSHOT) ☐ 3.2.10

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 23 ☒ 21 ☐ 17

Dependencies ADD DEPENDENCIES... ⌘ + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

PostgreSQL Driver SQL
A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

JDBC API SQL
Database Connectivity API that defines how a client may connect and query a database.

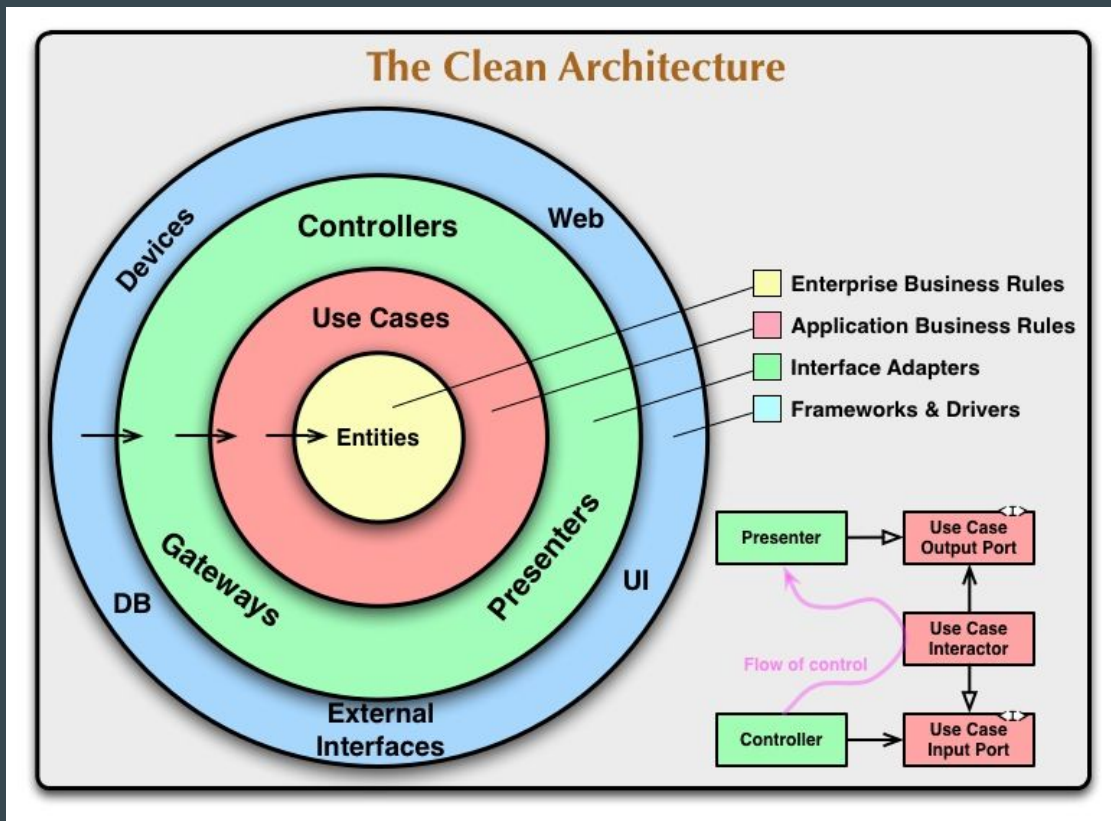
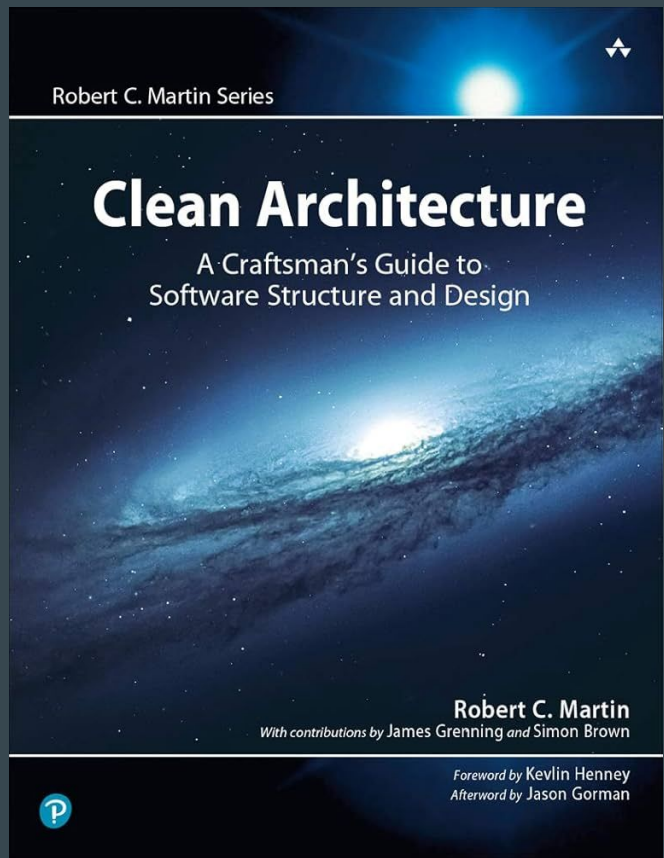
Docker Compose Support DEVELOPER TOOLS
Provides docker compose support for enhanced development experience.

Flyway Migration SQL
Version control for your database so you can migrate from any version (incl. an empty database) to the latest version of the schema.

Testcontainers TESTING
Provide lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container.

GENERATE ⌘ + ↵ EXPLORE CTRL + SPACE SHARE...

Dica #1: Comece pelo use-case



Dica #1: Comece pelo use-case

- Mapeie cada chamada de API de forma individual, como se fosse uma função
 - Uma interface por use-case
- Identifique e valide os parâmetros de entrada
 - Jakarta Beans Validation ou algo padronizado
- Identifique os parâmetros de saída. Evite não retornar nada
 - Sealed types p/ fluxos alternativos.
- Não se preocupe se o use-case irá fazer muita coisa (transação no banco, notificação em mensageria, disparo de e-mail etc). O importante é esclarecer (tipar) o seu contrato.

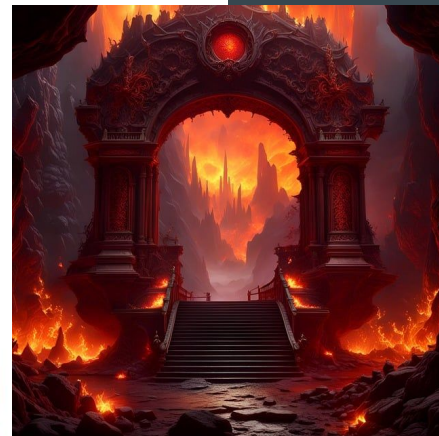
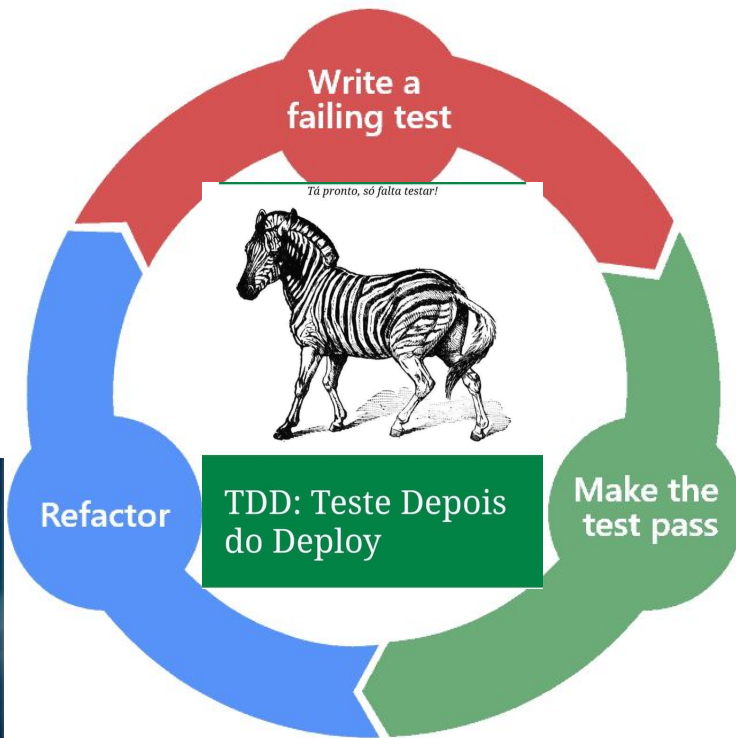
Dica #2: Valide o use-case com implementação "fake"

- Não deixe a equipe de front esperando demais
- Implemente um adapter da forma mais simples possível, que não exija nenhuma integração externa
- Garanta que todos os fluxos estejam implementados e corretos.
- Deixe documentado, quando necessário, os dados presentes para se realizar os testes de validação do protótipo. Exemplo:
 - conta 123 tem inicialmente saldo de R\$ 10,00
 - conta 234 está bloqueada com saldo de R\$ 0,00
- Não perca tempo escrevendo teste unitário p/ fake adapter. Se estiver utilizando plugin de cobertura de código, marque-o como artefato ignorado.

Dica #3: 1 RestController por use-case

- No caso de uma API REST, a exposição do use-case se dá via @RestController.
- Crie um controller por use-case e padronize o nome do método (handle / execute)
 - Padronize também um prefixo ou sufixo
- O método deve simplesmente delegar, de forma polimórfica, para o use-case responsável, aplicando as conversões necessárias de entrada/saída (JSON, fileupload, forms).
 - Toda a regra de negócio PARTE do use-case.
- Em caso de exceptions, trate-as de forma padronizada (@RestControllerAdvice)
 - Erro 400 p/ parâmetros incorretos

... mas e os testes?

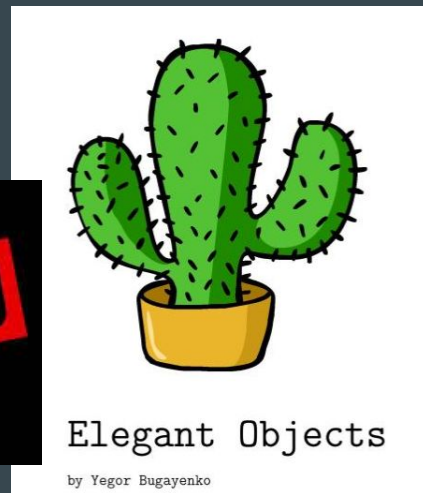


Dica #4: Implemente o use-case real após o aceite do protótipo

- Antes de começar, implemente o teste de integração da controller. Aplique o princípio Open/Closed para que possa ser estendido de acordo com as necessidades do use-case adapter (given) sem que suas validações (when/then) sejam modificadas.
- Crie uma especialização desse teste p/ cada use-case adapter começando pelo fake (p/ garantir que o teste está correto).
- Não há necessidade de escrever teste unitário destes adapters. Eles já estão sendo cobertos pelos testes de integração.

Dica #4: Implemente o use-case real após o aceite do protótipo

- Procure implementar um adapter p/ cada característica técnica do use-case
 - Um para transação do banco, outro para enviar e-mail, etc.
- Utilize *Decorator* p/ "juntar" os adapters.
 - Permita ligar/desligar adapters através de configurações externas (ex: enviar-email=true)
- <https://www.elegantobjects.org/> 🔥 🔥 🔥



Dica #5: Utilize Flyway em todos os ambientes

- Ative o clean somente nos estágios iniciais do projeto. Se entrou em sustentação, EVITE. Em ambiente de produção JAMAIS.
- Flyway só gerencia os artefatos criados PELA aplicação, e não PARA a aplicação.
 - Os pré-requisitos de banco devem estar em scripts no Dockerfile.

Dica #6: Mantenha o `application.yml` relevante e deduplicado

- Deixe o `application.yml` local configurado da forma mais completa possível.
- Se estiver utilizando **`config-server`** , coloque nele somente as exceções de cada ambiente.

Dica #7: Ajuste o Dockerfile / .dockerignore

- <https://docs.spring.io/spring-boot/reference/packaging/container-images/dockerfiles.html#page-title>
- Tente utilizar CDS, se possível.
 - <https://docs.spring.io/spring-boot/reference/packaging/class-data-sharing.html#page-title>
- Buildpacks facilitam muito!
 - <https://docs.spring.io/spring-boot/maven-plugin/build-image.html>
- Fique atento aos projetos focados em runtime mais eficiente:
 - Class Data Sharing (CDS)
 - Coordinated Restore at Checkpoint (CRaC)
 - Project Leyden
 - GraalVM native-image

Dica #8: Foi pro ar, e agora?

- Hora de apertar os cintos e garantir a evolução sustentável do projeto
- Configure ferramentas de análise estática de código
 - Cobertura / Sonar
 - Spotless: <https://github.com/diffplug/spotless>
- Se houver demandas evolutivas em um use-case, crie um novo *Decorator* com um teste especializado
- Mantenha-se firme nos princípios SOLID.

Dica #0: NÃO SUBESTIME AS REFERÊNCIAS

- <https://docs.spring.io/spring-boot/index.html>
- <https://docs.spring.io/spring-framework/reference/>
- Um dos fatores que colaboraram para o sucesso do Spring foi a qualidade da documentação, tanto externa (links acima) quanto interna (javadoc).
- Se tiver dúvida como, por exemplo, quais tipos de parâmetros podem ser utilizados no meu `@RestController`, pode ter certeza que achará a resposta na referência.
- Configure a sua IDE p/ baixar o código-fonte das dependências do projeto. Isso trará sugestões (autocomplete) mais detalhadas.

Mantenha-se atualizado com o blog oficial: <https://spring.io/blog>

Obrigado!