



Conteúdo do treinamento

ESPECIALISTA SPRING REST

Spring e Injeção de Dependências

Você vai conhecer os principais projetos do ecossistema Spring para desenvolvimento de REST APIs, como Spring Framework, Spring MVC, Spring Boot, Spring Data JPA, Spring Security, Spring Security OAuth e Spring HATEOAS.

Vai também entender definitivamente como funciona a injeção de dependências com Spring, o IoC Container, definição de beans com `@Component`, `@Configuration` e `@Bean`, pontos de injeção, desambiguação com `@Primary`, `@Qualifier` e anotação customizada.

Além disso, você vai aprender a trabalhar com Spring Profiles, publicação e consumo de eventos customizados, configuração de projetos com `application.properties` e diferenciação por profile, criação e acesso de propriedades customizadas com `@Value` e `@ConfigurationProperties`, etc.

JPA, Hibernate e Flyway

Você vai aprender o que é ORM, como fazer mapeamento de entidades e relacionamentos com JPA (OneToMany,ManyToOne, ManyToMany, OneToOne e Embedded).

Vai também entender e trabalhar com transações no Spring, usar JPQL e Criteria do JPA, fazer joins e fetch em relacionamentos, entender e configurar um pool de conexões (Hikari), etc.

Além disso, você ainda vai ver como criar e evoluir o schema do banco de dados com Flyway, criar migrações com remanejamento de dados, reparar migrações com erros, adicionar dados de testes com callback do Flyway, etc.

E ainda, vamos instalar e usar Lombok nas nossas classes para reduzir código boilerplate.

Spring Data JPA

Você vai conhecer os super poderes do Spring Data JPA, como criação de repositórios super inteligentes com a abstração do SDJ, criação de query methods com filtros, paginação e ordenação e parâmetros nomeados.

Vai também aprender a implementar repositórios customizados, externalização de consultas JPQL em arquivo XML, implementar consultas com o padrão Specifications (DDD), etc.

Domain-Driven Design (DDD)

Durante o desenvolvimento do projeto o curso, usaremos alguns conceitos e padrões do DDD, como Repository, Aggregate, Aggregate Root, Domain Event, Domain Service, Infrastructure Service, Specification e Linguagem Ubíqua.

Fundamentos avançados de REST com Spring

Você vai dominar os fundamentos de REST definitivamente, entender o que é e o que não é esse estilo arquitetural e as constraints do REST.

Vai entender o que é de fato e a diferença conceitual e prática de uma API, Resource, Resource Model, Resource Representation, Resource Identifier, Resource Methods, Collection Resources, Singleton Resource e Sub-collection Resources.

Você também vai conhecer todos os níveis do Modelo de Maturidade de Richardson e conhecer a abordagem pragmática e purista de desenvolvedores de REST APIs.

E ainda, você vai entender com mais detalhes o protocolo HTTP, quais são os principais códigos de status do HTTP e quando usar e não usar cada um deles, identificar e entender os métodos seguros e idempotentes do HTTP, conhecer e entender quais métodos HTTP você deve usar, etc.

Você vai aprender a criar, configurar e desenvolver uma API do zero e passo a passo, usando vários projetos do ecossistema Spring.

Vai desenvolver serviços com diversos métodos HTTP, inclusive vai aprender a fazer atualização parcial com PATCH e fazer content negotiation, e tudo seguindo as boas práticas que serão discutidas e ensinadas nas aulas.

Validações com Bean Validation

Que tal aprender a fazer validações de entradas da sua API de forma profissional e avançada?

Você vai aprender a adicionar validações no seu modelo com as anotações do Bean Validation.

Vai aprender também a customizar mensagens de validação, criar validações compostas, criar validações customizadas com ConstraintValidator, criar validações customizadas em nível de classe, criar grupos de validações, validar associações em cascata, executar checagem de validações programaticamente, entender a diferença do Resource Bundle do Spring e do Bean Validation, etc.

Tratamento e modelagem de erros da API

Tratar exceptions é muito importante, mas tão importante quanto isso, é devolver como resposta o código de status HTTP adequado e uma representação padrão do problema. Infelizmente, pouca gente faz isso direito.

Mas nesse treinamento você vai aprender a tratar as exceptions e devolver uma resposta adequada e consistente (padronizada) para o consumidor da API.

Você vai aprender a usar a anotação `@ResponseStatus`, tratar exceções em nível do controlador com `@ExceptionHandler`, usar a exceção padrão `ResponseStatusException` e tratar exceções globalmente com `@ExceptionHandler`, `@ControllerAdvice` e `ResponseEntityExceptionHandler`.

Vamos tratar erros de validação e atributos inexistentes, capturar a exception `NoHandlerFoundException` e várias outras.

Ainda, vamos discutir, implementar e estender a modelagem de erros seguindo a RFC 7807 (Problem Details for HTTP APIs).

Testes de integração

Ninguém tem dúvidas que testes de software são muito importantes, mas ainda mais se esses testes forem automatizados. Por isso, no treinamento você vai aprender a implementar testes de integração automatizados para a API.

Para implementar os testes de integração, vamos usar Spring Boot Test, REST Assured, JUnit, AssertJ e Maven Failsafe Plugin.

Vamos também implementar a limpeza e população de dados de testes para cada teste (é importante que o banco esteja em um estado consistente para não influenciar o resultado).

Boas práticas e modelagem avançada de APIs

Muitos desenvolvedores de APIs não se preocupam ou até desconhecem algumas boas práticas de mercado, por isso um dos objetivos deste treinamento é discutir e disseminar esse tipo de conhecimento entre os alunos.

Você vai aprender as 5 leis fundamentais para trabalhar com data/hora em APIs, como por exemplo, uso e tratamento de timezone na requisição, resposta e armazenamento de datas e o padrão ISO 8601 para troca de dados.

Vamos ver também como fazer customizações de configurações do Jackson usando classes de mixin, para deixar o código das entidades (model) mais limpo.

Vai aprender também as boas práticas para nomeação de URIs de recursos, escolha e configuração de estratégia de nomes de propriedades, diferença e quando usar recursos de granulação fina ou recursos de granulação grossa e como modelar conceitos abstratos e ações não-CRUD na sua API.

Vamos estudar também sobre como modelar fluxos de transições de estados de recursos, implementar Sub-collection Resources, modelar e implementar endpoints que executam ações em massa.

Discutiremos sobre as diferenças entre usar as entidades como modelo de representação dos recursos ou DTOs, para desacoplar mais os controllers, além de implementar das duas

formas. Usaremos o ModelMapper para fazer Object Mapping e converter DTOs em entidades e vice-versa.

Analisaremos sobre quando usar ID sequencial ou UUID para identificar recursos, além de implementar também das duas formas, de acordo com a necessidade. Veremos sobre a granularidade do payload, quando usamos respostas massivas (chunkiness) ou mais conversacional (chattiness).

Vamos implementar e organizar as nossas classes usando alguns conceitos e padrões do DDD (Domain-Driven Design), além de discutir sobre as vantagens e desvantagens de criar exceptions de negócio de granularidade fina ou grossa.

Modelagem de projeções, pesquisas e relatórios

No mundo real, surgem várias necessidades que às vezes só estudando por conta própria nem vai passar por sua cabeça, como por exemplo projeções e pesquisas complexas.

Como a ideia do treinamento é preparar os alunos para o mundo real, você vai aprender como implementar projeções do modelo de representação de nossos recursos usando DTO e @JsonView.

Além disso, vamos estudar sobre como limitar as propriedades retornadas usando property filter do Jackson e também usando a biblioteca Squiggly.

Vamos modelar pesquisas complexas com filtros dinâmicos, paginação e ordenação e criar um endpoint que pode ser usado para consumidores que precisam de dados agregados para plotar gráficos ou exibir relatórios.

Implementaremos também um serializador customizado do Jackson para customizar propriedades de paginação na representação.

O foco do treinamento é desenvolvimento de REST APIs com Spring, mas como será que podemos modelar um recurso para disponibilizar um relatório em PDF?

Por isso mesmo, nós vamos desenvolver um relatório JasperReports com Jaspersoft Studio e disponibilizar para download em PDF em um serviço, tudo seguindo as boas práticas.

Upload e download de arquivos

Muitas vezes, precisamos implementar upload de imagens e documentos para a API. Por exemplo, o upload da foto de um produto ou foto do usuário é algo bem comum.

No treinamento, você vai aprender a modelar recursos para isso também, e ainda vai aprender a armazenar o arquivo no próprio servidor ou ainda no storage da Amazon S3 (ideal para aplicações cloud-native). Além de implementar, vamos discutir quando é melhor usar cada uma.

E claro, se você vai aprender a fazer upload de arquivos, vai também aprender a fazer o download deles, com o cuidado de não degradar a performance da API e também seguindo as boas práticas para as implementações serem independentes e intercambiáveis.

Envio de e-mails transacionais

Algo muito comum que quase todos os sistemas precisam é o envio de e-mails transacionais, como por exemplo, e-mails para avisar que um pedido foi confirmado, que a entrega está a caminho, etc.

Nós vamos desenvolver isso e você ainda vai aprender a montar templates de e-mails com Apache FreeMarker, para dados dinâmicos.

Ainda, vamos implementar componentes alternativos para envio de e-mails fake e sandbox, ideal para ambientes de desenvolvimento e staging, para evitar disparo de mensagens de teste para e-mails reais.

Cache de HTTP

Um assunto pouco dominado por vários programadores e que ficará na ponta da sua língua (e dos seus dedos, porque vamos implementar isso) é caching de HTTP.

Por que fazer cache? Como funciona o cache de HTTP? Quando não fazer cache? Quais são as precauções que devemos ter? Tudo isso será discutido no treinamento.

Você vai conhecer e implementar HTTP Caching com Cache-Control, entender o que são ETags (Entity Tags), implementar requisições condicionais com Deep ETags e Shallow ETags e entender como funciona a expiração e invalidação de cache.

Documentação com OpenAPI (Swagger)

APIs bem documentadas simplificam a vida dos consumidores, por isso você vai aprender como fazer isso no treinamento usando a especificação OpenAPI (antigamente conhecida como Swagger).

E para automatizar a geração do arquivo de definição baseado na especificação, usaremos a biblioteca SpringFox, que consegue escanear o nosso código e fazer isso "automagicamente".

Usaremos também o Swagger UI dentro do nosso projeto, para disponibilizar a documentação em uma página bonitinha, de fácil navegação pelos consumidores.

Mas nem tudo são flores. Documentar uma API parece simples, ainda mais quando se usa OpenAPI/Swagger e SpringFox, mas não se engane. Em projetos reais, vários problemas aparecem e a documentação começa a ficar inconsistente.

Mas precisamos garantir que a documentação esteja refletindo exatamente o funcionamento da API, ou seja, que esteja consistente. Por isso você vai aprender vários hacks e configurações para ajustar a documentação e deixá-la limpa e correta.

HATEOAS e Discoverability

Discoverability é a capacidade que a API dá aos consumidores de "navegar" em seus recursos sem conhecer previamente as suas URIs, e HATEOAS é um componente do REST que torna isso possível através da inclusão de hypermedia nas respostas.

Uma API que usa HATEOAS é classificada no nível máximo do modelo de maturidade de Richardson, o que também chamamos de "A glória do REST".

Vamos estudar as formas de adicionar hypermedia nas representações dos recursos, usar o Spring HATEOAS e a especificação HAL (Hypertext Application Language) para deixar a API "descobrível" e atrativa aos consumidores.

Estudaremos como criar novos links que referenciam os métodos dos controllers, incluindo links com templates e como implementar Representation Model Assemblers, para instanciar classes de modelo de representação de recursos que suportam HATEOAS.

Vamos ter muita implementação própria para deixar as coisas consistentes, mas no final, vai valer à pena. A API ficará linda! 🥰

CORS e consumo das APIs com Java e JavaScript

O foco do treinamento é desenvolver REST APIs, mas todo programador precisa saber como consumir sua própria API também, para se colocar no lugar do consumidor e fazer alguns testes.

Além disso, pode ser que você precise consumir APIs de terceiros, então é sempre bom saber como fazer isso.

Você vai aprender a consumir a API usando a linguagem Java e JavaScript.

E vai também entender a Política de Mesma Origem (Same Origin Policy), que os navegadores implementam, como funciona o CORS (incluindo os cabeçalhos), como habilitar CORS na API por método, controlador ou globalmente.

Segurança com Spring Security, OAuth2 e JWT

Uma coisa muito importante em qualquer API é a segurança, por isso nós vamos aprofundar bastante nesse assunto.

Para começar, vamos implementar uma autenticação básica HTTP usando Spring Security. Esse tipo de autenticação é útil em alguns casos, por isso não poderia faltar.

Mas em cenários mais complexos, precisamos de uma solução melhor para autorizar clientes a usarem a API, e por isso você vai aprender sobre OAuth2.

Vamos implementar o Resource Server e Authorization Server em projetos separados e também no mesmo projeto.

Ou seja, você aprenderá a implementar para pequenos projetos, onde geralmente eles ficam juntos e também para grandes projetos, onde geralmente eles ficam separados.

Você vai aprender a implementar e usar os seguintes fluxos de autorização do OAuth2: Resource Owner Password Credentials Flow, Client Credentials Flow, Implicit Flow e Authorization Code Flow.

Você vai aprender sobre a diferença entre access token e refresh token e como gerar um novo access token através de um refresh token, além de discutirmos sobre o cuidado que você deve ter com cada tipo de token.

E além disso, vai também aprender a implementar PKCE (Proof Key for Code Exchange), que é uma extensão para Authorization Code Flow recomendada para clientes públicos, como Single-Page Applications e aplicações nativas (desktop ou mobile).

O Authorization Server será implementado com Spring Security OAuth2, também conhecida como "stack antiga". Atualmente, é a única solução que temos no ecossistema para fazer isso.

O Resource Server será implementado usando a nova stack do Spring Security para OAuth2.

Vamos estudar sobre a diferença entre Opaque Tokens e Transparent Tokens, e vamos implementar das duas formas.

Com Opaque Tokens, temos que armazenar esses tokens em algum lugar, por isso vamos configurar o Redis, um banco NoSQL do tipo chave-valor, e configurar a aplicação Spring para usá-lo. Fica bem melhor que guardar os tokens na memória da aplicação.

Vamos configurar o Authorization Server para carregar o cadastro de clients OAuth (aplicações consumidoras) do banco de dados, para evitar que isso fique *hard-coded*.

E ainda, vamos implementar Transparent Tokens com JWT (JSON Web Tokens).

Aprenderemos o que é e como funciona o JWT e como assinar os tokens com algoritmo simétrico (HMAC SHA-256) e assimétrico (RSA SHA-256) e qual é a diferença entre eles.

Vamos também customizar o código de login, para validar o usuário e senha no banco de dados da aplicação e adicionar claims públicas no payload do JWT.

A segurança da aplicação será planejada para ter as entidades de usuário, grupo e permissão, por isso será bem granular.

Para garantir que os endpoints só possam ser acessados pelos usuários autorizados, vamos implementar a segurança usando Method Security, as anotações `@PreAuthorize` e `@PostAuthorize` e Spring Expression Language (SpEL).

Usaremos as authorities e roles para restringir os acessos dos usuários, além de conhecer e usar também os escopos (scopes) do OAuth2.

Criaremos também meta-anotações de segurança, para simplificar o código dos controllers e nos beneficiar de reaproveitamento de código.

Faremos ainda um controle de segurança contextual, ainda mais granular, como por exemplo quando um usuário só pode acessar um registro que é seu, ou seja, mesmo que tenha acesso ao endpoint, não é permitido que ele acesse o registro de um outro usuário.

E por fim, vamos implementar a inclusão de links (hypermedia) de acordo com as permissões do usuário (evitando a inclusão de links que não são acessíveis por falta de permissão).

Cloud-native APIs

A aplicação que vamos desenvolver no treinamento será cloud-native, ou seja, planejaremos desde o início pensando que ela será implantada na nuvem.

E isso faz uma grande diferença, porque desenvolver para a nuvem tem algumas peculiaridades. Temos que tomar alguns cuidados. Mas claro, a mesma aplicação poderá ser implantada fora da nuvem também, em um servidor de intranet.

Deploy em produção na nuvem

Se a nossa API é cloud-native, nós faremos o deploy dela na nuvem.

Usaremos Spring Profiles para alternar algumas configurações de desenvolvimento e produção sem precisar recompilar e reempacotar o projeto.

Não manteremos configurações sensíveis de produção (como por exemplo o host, usuário e senha do banco de dados) dentro do projeto. Essas configurações serão adicionadas em variáveis de ambiente, seguindo a recomendação do *The Twelve-Factor App*.

Adotaremos a plataforma da Pivotal Web Services (PWS) para o deploy da aplicação, por várias razões:

- 1) Ela usa Cloud Foundry, que é uma plataforma open source apoiada pela Google, IBM, Microsoft, Pivotal, SAP, etc
- 2) É gerenciada pela Pivotal, que é a empresa que está por trás do desenvolvimento do Spring, e por isso a plataforma tem muita afinidade com aplicações Spring

- 3) Roda na Amazon Web Services (AWS), que é a maior referência em infraestrutura de nuvem
- 4) Não precisamos nos preocupar com o gerenciamento da infraestrutura e sistemas operacionais (a Pivotal e Amazon tomam conta de tudo)
- 5) É escalável. Podemos ter uma ou centenas de instâncias (containers) rodando a nossa aplicação e dividindo a carga (inclusive, vamos fazer isso)
- 6) Oferece um marketplace com vários serviços que podem ser instanciados com poucos cliques e alguns até com versões gratuitas, como por exemplo MySQL, Redis, etc
- 7) Por ser um serviço baseado em Cloud Foundry, você não fica preso ao fornecedor (*vendor lock-in*), já que existem outros provedores que também suportam Cloud Foundry
- 8) Tem um trial bem generoso, que dá para testar bastante (e não exige cartão de crédito)
- 9) Tem um excelente custo benefício, caso você queira usar em produção (usamos na AlgaWorks para a nossa plataforma e nos atende muito bem)

Mas só pra ficar claro, nosso projeto não ficará dependente da PWS. Você poderá fazer deploy no Heroku, Digital Ocean, Amazon EC2 e outros provedores normalmente, sem qualquer problema.

Configuração e gerenciamento de logs

Ninguém duvida que fazer logging das mensagens importantes e erros é essencial para qualquer aplicação, mas para aplicações cloud-native, onde implantamos em containers descartáveis, temos que tomar cuidado.

Não podemos fazer logging para um appender de filesystem, ou seja, armazenar os logs na própria máquina, porque podemos descartar e instanciar uma nova a qualquer momento e os logs poderiam ser perdidos.

Por isso, vamos usar um serviço de gerenciamento de logs na nuvem, para onde vamos configurar a transmissão das mensagens de maneira assíncrona.

O legal é que essas ferramentas agregam os logs de todas as instâncias (containers) e fica muito mais fácil e seguro analisar os logs.

Versionamento de APIs

Uma hora ficará tão difícil manter a mesma API, que você vai sentir necessidade de criar ou alterar funcionalidades que quebram a compatibilidade com os consumidores.

O problema é que você já pode ter vários consumidores usando a API e pode ficar difícil ou até impossível migrar todos eles em uma pequena janela de tempo.

Nesse caso, você precisará implementar uma nova versão, e aí surgem várias dúvidas.

No treinamento, nós vamos estudar melhor sobre quando versionar uma API, quais as estratégias para separar o código-fonte da API antiga e da API nova e vamos implementar a seleção de versão por Media Types e por URIs.

Você vai aprender como evoluir a sua API sem quebrar os clientes (que é o ideal, já que criar uma nova versão é uma dor de cabeça que você deve evitar).

Vamos estudar também quando e como implementar depreciação e remoção de suporte de uma API, para que você faça uma transição de versões de forma mais suave.

Conteúdo programático

1. Introdução

- 1.1. Introdução ao treinamento
- 1.2. Como usar o suporte da AlgaWorks
- 1.3. Por que desenvolver REST APIs?
- 1.4. Conhecendo o modelo de domínio do projeto do curso
- 1.5. Preparando o ambiente de desenvolvimento: JDK e STS for Eclipse

2. Spring e Injeção de Dependências

- 2.1. Por que aprender e usar Spring?
- 2.2. Conhecendo o ecossistema Spring
- 2.3. Spring vs Jakarta EE (Java EE)
- 2.4. Conhecendo o Spring Boot
- 2.5. Criando um projeto Spring Boot com Spring Initialize
- 2.6. Conhecendo o Maven e o pom.xml de um projeto Spring Boot
- 2.7. Criando um controller com Spring MVC
- 2.8. Restart mais rápido da aplicação com DevTools
- 2.9. O que é injeção de dependências?
- 2.10. Conhecendo o IoC Container do Spring
- 2.11. Definindo beans com `@Component`
- 2.12. Injetando dependências (beans Spring)
- 2.13. Usando `@Configuration` e `@Bean` para definir beans
- 2.14. Conhecendo os pontos de injeção e a anotação `@Autowired`
- 2.15. Dependência opcional com `@Autowired`
- 2.16. Ambiguidade de beans e injeção de lista de beans
- 2.17. Desambiguação de beans com `@Primary`
- 2.18. Desambiguação de beans com `@Qualifier`
- 2.19. Desambiguação de beans com anotação customizada
- 2.20. Mudando o comportamento da aplicação com Spring Profiles
- 2.21. Criando métodos de callback do ciclo de vida dos beans
- 2.22. Publicando e consumindo eventos customizados
- 2.23. Configurando projetos Spring Boot com o `application.properties`
- 2.24. Substituindo propriedades via linha de comando e variáveis de ambiente

- 2.25. Criando e acessando propriedades customizadas com @Value
- 2.26. Acessando propriedades com @ConfigurationProperties
- 2.27. Alterando a configuração do projeto dependendo do ambiente (com Spring Profiles)
- 2.28. Ativando o Spring Profile por linha de comando e variável de ambiente

3. Introdução ao JPA e Hibernate

- 3.1. Instalando o MySQL Server e MySQL Workbench
- 3.2. O que é JPA e Hibernate?
- 3.3. Adicionando JPA e configurando o Data Source
- 3.4. Mapeando entidades com JPA
- 3.5. Criando as tabelas do banco a partir das entidades
- 3.6. Mapeando o id da entidade para autoincremento
- 3.7. Importando dados de teste com import.sql
- 3.8. Consultando objetos do banco de dados
- 3.9. Adicionando um objeto no banco de dados
- 3.10. Buscando um objeto pelo id no banco de dados
- 3.11. Atualizando um objeto no banco de dados
- 3.12. Excluindo um objeto do banco de dados
- 3.13. Conhecendo o padrão Aggregate do DDD
- 3.14. Conhecendo e implementando o padrão Repository
- 3.15. Conhecendo e usando o Lombok
- 3.16. Desafio: Lombok e repositório de restaurantes
- 3.17. Mapeando relacionamento com @ManyToOne
- 3.18. A anotação @JoinColumn
- 3.19. Propriedade nullable de @Column e @JoinColumn
- 3.20. Desafio: mapeando entidades

4. REST com Spring

- 4.1. O que é REST?
- 4.2. Conhecendo as constraints do REST
- 4.3. Diferença entre REST e RESTful
- 4.4. Desenvolvedores de REST APIs puristas e pragmáticos
- 4.5. Conhecendo o protocolo HTTP
- 4.6. Usando o protocolo HTTP
- 4.7. Instalando e testando o Postman
- 4.8. Entendendo o que são Recursos REST
- 4.9. Identificando recursos REST
- 4.10. Modelando e requisitando um Collection Resource com GET
- 4.11. Desafio: collection resource de estados
- 4.12. Representações de recursos e content negotiation
- 4.13. Implementando content negotiation para retornar JSON ou XML
- 4.14. Consultando Singleton Resource com GET e @PathVariable
- 4.15. Customizando as representações XML e JSON com @JsonIgnore, @JsonProperty e @JsonRootName
- 4.16. Customizando a representação em XML com Wrapper e anotações do Jackson
- 4.17. Conhecendo os métodos HTTP
- 4.18. Conhecendo os códigos de status HTTP
- 4.19. Definindo o status da resposta HTTP com @ResponseStatus

- 4.20. Manipulando a resposta HTTP com ResponseEntity
- 4.21. Corrigindo o Status HTTP para resource inexistente
- 4.22. Status HTTP para collection resource vazia: qual usar?
- 4.23. Modelando e implementando a inclusão de recursos com POST
- 4.24. Negociando o media type do payload do POST com Content-Type
- 4.25. Modelando e implementando a atualização de recursos com PUT
- 4.26. Modelando e implementando a exclusão de recursos com DELETE
- 4.27. Implementando a camada de domain services (e a importância da linguagem ubíqua)
- 4.28. Refatorando a exclusão de cozinhas para usar domain services
- 4.29. Desafio: modelando e implementando a consulta de recursos de restaurantes
- 4.30. Modelando e implementando a inclusão de recursos de restaurantes
- 4.31. Desafio: Modelando e implementando a atualização de recursos de restaurantes
- 4.32. Desafio: implementando serviços REST de cidades e estados
- 4.33. Analisando solução para atualização parcial de recursos com PATCH
- 4.34. Finalizando a atualização parcial com a API de Reflections do Spring
- 4.35. Introdução ao Modelo de Maturidade de Richardson (RMM)
- 4.36. Conhecendo o nível 0 do RMM
- 4.37. Conhecendo o nível 1 do RMM
- 4.38. Conhecendo o nível 2 do RMM
- 4.39. Conhecendo o nível 3 do RMM

5. Super poderes do Spring Data JPA

- 5.1. Implementando consultas JPQL em repositórios
- 5.2. Conhecendo o projeto Spring Data JPA (SDJ)
- 5.3. Criando um repositório com Spring Data JPA (SDJ)
- 5.4. Refatorando o código do projeto para usar o repositório do SDJ
- 5.5. Desafio: refatorando todos os repositórios para usar SDJ
- 5.6. Criando consultas com query methods
- 5.7. Usando as keywords para definir critérios de query methods
- 5.8. Conhecendo os prefixos de query methods
- 5.9. Usando queries JPQL customizadas com @Query
- 5.10. Externalizando consultas JPQL para um arquivo XML
- 5.11. Implementando um repositório SDJ customizado
- 5.12. Implementando uma consulta dinâmica com JPQL
- 5.13. Implementando uma consulta simples com Criteria API
- 5.14. Adicionando restrições na cláusula where com Criteria API
- 5.15. Tornando a consulta com Criteria API com filtros dinâmicos
- 5.16. Conhecendo o uso do padrão Specifications (DDD) com SDJ
- 5.17. Implementando Specifications com SDJ
- 5.18. Criando uma fábrica de Specifications
- 5.19. Injetando o próprio repositório na implementação customizada e a anotação @Lazy
- 5.20. Estendendo o JpaRepository para customizar o repositório base

6. Explorando mais do JPA e Hibernate

- 6.1. Mapeando relacionamento bidirecional com @OneToMany
- 6.2. Mapeando relacionamento muitos-para-muitos com @ManyToMany
- 6.3. Analisando o impacto do relacionamento muitos-para-muitos na REST API
- 6.4. Mapeando classes incorporáveis com @Embedded e @Embeddable

- 6.5. Testando e analisando o impacto da incorporação de classe na REST API
- 6.6. Mapeando propriedades com `@CreationTimestamp` e `@UpdateTimestamp`
- 6.7. Desafio: mapeando relacionamento muitos-para-um
- 6.8. Desafio: mapeando relacionamento um-para-muitos
- 6.9. Desafio: mapeando relacionamentos muitos-para-muitos
- 6.10. Entendendo o Eager Loading
- 6.11. Entendendo o Lazy Loading
- 6.12. Alterando a estratégia de fetching para Lazy Loading
- 6.13. Alterando a estratégia de fetching para Eager Loading
- 6.14. Resolvendo o Problema do N+1 com fetch join na JPQL

7. Pool de conexões e Flyway

- 7.1. Entendendo o funcionamento de um pool de conexões
- 7.2. Conhecendo o Hikari: a solução padrão de pool de conexões no Spring Boot
- 7.3. Configurando o pool de conexões do Hikari
- 7.4. Schema generation em produção não é uma boa prática
- 7.5. Flyway: ferramenta de versionamento de schemas de banco de dados
- 7.6. Adicionando o Flyway no projeto e criando a primeira migração
- 7.7. Evoluindo o banco de dados com novas migrações
- 7.8. Criando migrações complexas com remanejamento de dados
- 7.9. Criando migração a partir de DDL gerado por schema generation
- 7.10. Adicionando dados de testes com callback do Flyway
- 7.11. Reparando migrações com erros
- 7.12. Desafio: Criando migrações e mapeando as entidades Pedido e ItemPedido

8. Tratamento e modelagem de erros da API

- 8.1. Introdução ao tratamento e modelagem de erros
- 8.2. Lançando exceções customizadas anotadas com `@ResponseStatus`
- 8.3. Lançando exceções do tipo `ResponseStatusException`
- 8.4. Estendendo `ResponseStatusException`
- 8.5. Simplificando o código com o uso de `@ResponseStatus` em exceptions
- 8.6. Desafio: refatorando os serviços REST
- 8.7. Analisando os impactos da refatoração
- 8.8. Criando a exception `NegocioException`
- 8.9. Desafio: usando a exception `NegocioException`
- 8.10. Afinando a granularidade e definindo a hierarquia das exceptions de negócios
- 8.11. Desafio: lançando exceptions de granularidade fina
- 8.12. Tratando exceções em nível de controlador com `@ExceptionHandler`
- 8.13. Tratando exceções globais com `@ExceptionHandler` e `@ControllerAdvice`
- 8.14. Desafio: implementando exception handler
- 8.15. Criando um exception handler global com `ResponseEntityExceptionHandler`
- 8.16. Customizando o corpo da resposta padrão de `ResponseEntityExceptionHandler`
- 8.17. Conhecendo a RFC 7807 (Problem Details for HTTP APIs)
- 8.18. Padronizando o formato de problemas no corpo de respostas com a RFC 7807
- 8.19. Desafio: usando o formato de problemas no corpo de respostas
- 8.20. Customizando exception handlers de `ResponseEntityExceptionHandler`
- 8.21. Tratando a exception `InvalidFormatException` na desserialização
- 8.22. Habilitando erros na desserialização de propriedades inexistentes ou ignoradas

- 8.23. Desafio - tratando PropertyBindingException na desserialização
- 8.24. Lançando exception de desserialização na atualização parcial (PATCH)
- 8.25. Desafio: tratando exception de parâmetro de URL inválido
- 8.26. Desafio: tratando a exceção NoHandlerFoundException
- 8.27. Desafio: tratando outras exceções não capturadas
- 8.28. Estendendo o formato do problema para adicionar novas propriedades
- 8.29. Desafio: estendendo o formato do problema

9. Validações com Bean Validation

- 9.1. Validação do modelo com Bean Validation
- 9.2. Adicionando constraints e validando no controller com @Valid
- 9.3. Desafio: tratando exception de violação de constraints de validação
- 9.4. Estendendo o Problem Details para adicionar as propriedades com constraints violadas
- 9.5. Conhecendo e adicionando mais constraints de validação no modelo
- 9.6. Validando as associações de uma entidade em cascata
- 9.7. Agrupando e restringindo constraints que devem ser usadas na validação
- 9.8. Convertendo grupos de constraints para validação em cascata com @ConvertGroup
- 9.9. Desafio: adicionando constraints de validação no modelo
- 9.10. Customizando mensagens de validação na anotação da constraint
- 9.11. Customizando e resolvendo mensagens de validação globais em Resource Bundle
- 9.12. Desafio: customizando mensagens de validação
- 9.13. Resolvendo mensagens de validação com Resource Bundle do Bean Validation
- 9.14. Usando o Resource Bundle do Spring como Resource Bundle do Bean Validation
- 9.15. Criando constraints de validação customizadas usando composição
- 9.16. Criando constraints de validação customizadas com implementação de ConstraintValidator
- 9.17. Criando constraints de validação customizadas em nível de classe
- 9.18. Ajustando Exception Handler para adicionar mensagens de validação em nível de classe
- 9.19. Executando processo de validação programaticamente
- 9.20. Desafio: tratando a exception customizada de validações programáticas

10. Testes de integração

- 10.1. Introdução aos Testes de Integração e Testes de APIs
- 10.2. Preparando o projeto para testes de integração
- 10.3. Criando e rodando um teste de integração com Spring Boot, JUnit e AssertJ
- 10.4. Escrevendo bons nomes de testes
- 10.5. Desafio: escrevendo testes de integração
- 10.6. Rodando os testes pelo Maven
- 10.7. Configurando Maven Failsafe Plugin no projeto
- 10.8. Implementando Testes de API com REST Assured e validando o código de status HTTP
- 10.9. Validando o corpo da resposta HTTP
- 10.10. Criando um método para fazer setup dos testes
- 10.11. Entendendo o problema da ordem de execução dos testes
- 10.12. Voltando o estado inicial do banco de dados para cada execução de teste com callback do Flyway
- 10.13. Configurando um banco de testes e usando @TestPropertySource

- 10.14. Limpando e populando o banco de dados de teste
- 10.15. Testando endpoint passando parâmetro de URL
- 10.16. Desafio: refatorando o código de testes
- 10.17. Desafio: escrevendo testes de API

11. Boas práticas e técnicas para APIs

- 11.1. Analisando e definindo melhor o escopo das transações
- 11.2. Refinando o payload de cadastro com `@JsonIgnoreProperties`
- 11.3. Criando classes de mixin para usar as anotações do Jackson
- 11.4. Desafio: usando `@JsonIgnoreProperties` e Jackson Mixin
- 11.5. Antes de estudar sobre data/hora: lembrando as aulas de geografia e entendendo os fusos horários
- 11.6. Boas práticas para trabalhar com data e hora em REST APIs
- 11.7. Configurando e refatorando o projeto para usar UTC
- 11.8. Desafio - refatorando o código para usar `OffsetDateTime`
- 11.9. Isolando o Domain Model do Representation Model com o padrão DTO
- 11.10. Implementando a conversão de entidade para DTO
- 11.11. Criando DTOs para entrada de dados na API
- 11.12. Refatorando e criando um assembler de DTO
- 11.13. Desafio: Refatorando e criando um disassembler do DTO
- 11.14. Adicionando e usando o `ModelMapper`
- 11.15. Entendendo a estratégia padrão do `ModelMapper` para correspondência de propriedades
- 11.16. Customizando o mapeamento de propriedades com `ModelMapper`
- 11.17. Mapeando para uma instância destino (e não um tipo) com `ModelMapper`
- 11.18. Revisando e ajustando as mensagens de validação com o uso de DTOs
- 11.19. Estratégias de nomes de propriedades - snake case vs lower camel case
- 11.20. Desafio: usando DTOs como representation model
- 11.21. Corrigindo bug de tratamento de exception de integridade de dados com flush do JPA

12. Modelagem avançada e implementação da API

- 12.1. Modelando sub-recursos para relacionamentos
- 12.2. Granularidade de recursos: Chatty vs Chunky APIs
- 12.3. Modelando conceitos abstratos de negócio e ações não-CRUD como recursos
- 12.4. Implementando os endpoints de ativação e inativação de restaurantes
- 12.5. Desafio: implementando os endpoints de formas de pagamento
- 12.6. Adicionando endereço no modelo da representação do recurso de restaurante
- 12.7. Refatorando serviço de cadastro de restaurante para incluir endereço
- 12.8. Desafio: implementando os endpoints de grupos
- 12.9. Desafio: implementando os endpoints de usuarios
- 12.10. Um pouco mais sobre JPA: objeto alterado fora da transação é sincronizado com o banco de dados
- 12.11. Implementando regra de negócio para evitar usuários com e-mails duplicados
- 12.12. Implementando os endpoints de associação de formas de pagamento em restaurantes
- 12.13. Desafio: implementando os endpoints de produtos
- 12.14. Desafio: Implementando os endpoints de abertura e fechamento de restaurantes

- 12.15. Desafio: implementando os endpoints de associação de grupos com permissões
- 12.16. Desafio: implementando os endpoints de associação de usuários com grupos
- 12.17. Desafio: implementando endpoints de associação de usuários responsáveis com restaurantes
- 12.18. Implementando ativação e inativação em massa de restaurantes
- 12.19. Desafio: Implementando os endpoints de consulta de pedidos
- 12.20. Otimizando a query de pedidos e retornando model resumido na listagem
- 12.21. Desafio: Implementando o endpoint de emissão de pedidos
- 12.22. Implementando endpoint de transição de status de pedidos
- 12.23. Desafio: implementando endpoints de transição de status de pedidos
- 12.24. Refatorando o código de regras para transição de status de pedidos
- 12.25. Usando IDs vs UUIDs nas URIs de recursos

13. Modelagem de projeções, pesquisas e relatórios

- 13.1. Fazendo projeção de recursos com @JsonView do Jackson
- 13.2. Limitando os campos retornados pela API com @JsonFilter do Jackson
- 13.3. Limitando os campos retornados pela API com Squiggly
- 13.4. Implementando pesquisas simples na API
- 13.5. Modelando pesquisas complexas na API
- 13.6. Implementando pesquisas complexas na API
- 13.7. Tratando BindException ao enviar parâmetros de URL inválidos
- 13.8. Implementando paginação e ordenação em recursos de coleção da API
- 13.9. Desafio: implementando paginação e ordenação de pedidos
- 13.10. Implementando JsonSerializer para customizar representação de paginação
- 13.11. Implementando um conversor de propriedades de ordenação
- 13.12. Modelando endpoints de consultas com dados agregados (ideal para gráficos e dashboards)
- 13.13. Discutindo sobre onde implementar as consultas com dados agregados
- 13.14. Implementando consulta com dados agregados de vendas diárias
- 13.15. Desafio: adicionando os filtros na consulta de vendas diárias
- 13.16. Tratando time offset na agregação de vendas diárias por data
- 13.17. Conhecendo o JasperSoft Studio
- 13.18. Criando um layout do relatório JasperReports de vendas diárias
- 13.19. Estruturando endpoint e serviço de emissão de relatório em PDF
- 13.20. Preenchendo um relatório JasperReports com JavaBeans e gerando bytes do PDF

14. Upload e download de arquivos

- 14.1. Conhecendo soluções para upload de arquivos em REST APIs
- 14.2. Implementando upload de arquivo com multipart/form-data
- 14.3. Validando o tamanho máximo do arquivo
- 14.4. Desafio: Validando o content type do arquivo
- 14.5. Mapeando entidade FotoProduto e relacionamento um-para-um
- 14.6. Implementando serviço de cadastro de foto de produto
- 14.7. Excluindo e substituindo cadastro de foto de produto
- 14.8. Implementando o serviço de armazenagem de fotos no disco local
- 14.9. Integrando o serviço de catálogo de fotos com o serviço de armazenagem
- 14.10. Implementando a remoção e substituição de arquivos de fotos no serviço de armazenagem

- 14.11. Desafio: Implementando recuperação de foto no serviço de armazenagem
- 14.12. Desafio: Implementando endpoint de consulta de foto de produto
- 14.13. Servindo arquivos de fotos pela API
- 14.14. Checando media type ao servir arquivos de fotos
- 14.15. Desafio: implementando endpoint de exclusão de foto de produto
- 14.16. Corrigindo exception handler de media type não aceita
- 14.17. Amazon S3: conhecendo o serviço de storage da AWS
- 14.18. Criando usuário com permissões para adicionar objetos na Amazon S3
- 14.19. Criando chaves de acesso para a API da AWS
- 14.20. Criando bean de propriedades de configuração dos serviços de storage
- 14.21. Adicionando o SDK Java da Amazon S3 no projeto e criando a classe da implementação do serviço de storage
- 14.22. Definindo bean do client da Amazon S3 e configurando credenciais
- 14.23. Implementando a inclusão de objetos no bucket da Amazon S3
- 14.24. Desafio: Implementando a exclusão de objetos do bucket da Amazon S3
- 14.25. Implementando a recuperação de foto no serviço de storage do S3
- 14.26. Selecionando a implementação do serviço de storage de fotos

15. E-mails transacionais e Domain Events

- 15.1. Conhecendo soluções para envio de e-mails transacionais
- 15.2. Configurando o projeto para envio de e-mails usando servidor SMTP
- 15.3. Implementando o serviço de infraestrutura de envio de e-mails com Spring
- 15.4. Usando o serviço de envio de e-mails na confirmação de pedidos
- 15.5. Processando template do corpo de e-mails com Apache FreeMarker
- 15.6. Melhorando o texto do e-mail com FTL (FreeMarker Template Language)
- 15.7. Formatando valores monetários com FTL
- 15.8. Desafio: implementando serviço de envio de e-mail fake
- 15.9. Desafio: Implementando serviço de envio de e-mail sandbox
- 15.10. Conhecendo o padrão Domain Events do DDD
- 15.11. Publicando Domain Events a partir do Aggregate Root
- 15.12. Observando e reagindo a Domain Events
- 15.13. Reagindo a Domain Events em fases específicas da transação
- 15.14. Desafio: enviando e-mails no cancelamento de pedidos

16. CORS e consumo da API com JavaScript e Java

- 16.1. Implementando uma chamada na REST API com JavaScript
- 16.2. Testando a requisição na API com JavaScript e entendendo a Same Origin Policy
- 16.3. Entendendo o funcionamento básico de CORS e habilitando na API
- 16.4. Habilitando CORS em controladores e métodos com @CrossOrigin
- 16.5. Entendendo o funcionamento de preflight do CORS
- 16.6. Habilitando CORS globalmente no projeto da API
- 16.7. Desafio: implementando uma requisição GET com JavaScript
- 16.8. Implementando um formulário de cadastro e fazendo requisição POST com JavaScript
- 16.9. Desafio: implementando uma requisição DELETE com JavaScript
- 16.10. Implementando um client da REST API com Java e Spring (RestTemplate)
- 16.11. Tratando respostas com código de erro no client Java
- 16.12. Desafio: Implementando uma requisição POST no client Java

17. Cache de HTTP

- 17.1. Introdução ao Cache de HTTP
- 17.2. Habilitando o cache com o cabeçalho Cache-Control e a diretiva max-age
- 17.3. Desafio: adicionando o cabeçalho Cache-Control na resposta
- 17.4. Entendendo a validação de representações em cache com ETags
- 17.5. Implementando requisições condicionais com Shallow ETags
- 17.6. Adicionando outras diretivas de Cache-Control na resposta HTTP
- 17.7. Usando a diretiva no-cache no cabeçalho Cache-Control da requisição
- 17.8. Entendendo e preparando a implementação de Deep ETags
- 17.9. Implementando requisições condicionais com Deep ETags
- 17.10. Desafio: implementando requisições condicionais com Deep ETags

18. Documentação da API com OpenAPI, Swagger UI e SpringFox

- 18.1. Introdução à documentação de REST APIs
- 18.2. Conhecendo a OpenAPI (antes conhecida como Swagger)
- 18.3. Gerando a definição OpenAPI em JSON com SpringFox
- 18.4. Gerando a documentação da API em HTML com Swagger UI e SpringFox
- 18.5. Selecionando os endpoints da API para gerar a documentação
- 18.6. Descrevendo informações da API na documentação
- 18.7. Descrevendo tags na documentação e associando com controllers
- 18.8. Descrevendo as operações de endpoints na documentação
- 18.9. Descrevendo parâmetros de entrada na documentação
- 18.10. Descrevendo modelos de representações e suas propriedades
- 18.11. Descrevendo restrições de validação de propriedades do modelo
- 18.12. Descrevendo códigos de status de respostas de forma global
- 18.13. Desafio: descrevendo códigos de status de respostas de forma global
- 18.14. Descrevendo o modelo de representação de problema
- 18.15. Referenciando modelo de representação de problema com códigos de status de erro
- 18.16. Descrevendo códigos de status de respostas em endpoints específicos
- 18.17. Desacoplando anotações do Swagger dos controladores
- 18.18. Desafio: descrevendo documentação de endpoints de grupos
- 18.19. Descrevendo media type da resposta nos endpoints
- 18.20. Corrigindo documentação com substituição de Pageable
- 18.21. Corrigindo documentação com substituição Page
- 18.22. Desafio: descrevendo documentação de endpoints de cozinhas
- 18.23. Ignorando tipos de parâmetros de operações na documentação
- 18.24. Desafio: descrevendo documentação de endpoints de formas de pagamento
- 18.25. Descrevendo parâmetros globais em operações
- 18.26. Descrevendo parâmetros implícitos em operações
- 18.27. Desafio: descrevendo documentação de endpoints de pedidos
- 18.28. Descrevendo parâmetros de projeções em endpoints de consultas
- 18.29. Desafio: descrevendo documentação de endpoints de restaurantes
- 18.30. Desafio: descrevendo documentação de endpoints de estados
- 18.31. Desafio: descrevendo documentação de endpoints de fluxo de pedidos
- 18.32. Desafio: descrevendo documentação de endpoints de associação de restaurantes com formas de pagamento
- 18.33. Desafio: descrevendo documentação de endpoints de associação de restaurantes com usuários

- 18.34. Desafio: descrevendo documentação de endpoints de produtos
- 18.35. Desafio: descrevendo documentação de endpoints de fotos de produtos
- 18.36. Corrigindo documentação no Swagger UI para upload de arquivos
- 18.37. Desafio: descrevendo documentação de endpoints de associação de permissões com grupos
- 18.38. Desafio: descrevendo documentação de endpoints de usuários
- 18.39. Desafio: descrevendo documentação de endpoints de associação de grupos com usuários
- 18.40. Desafio: descrevendo documentação de endpoint de estatísticas

19. Discoverability e HATEOAS: A Glória do REST

- 19.1. Introdução à Discoverability e HATEOAS
- 19.2. Adicionando a URI do recurso criado no header da resposta
- 19.3. Adicionando o Spring HATEOAS no projeto
- 19.4. Atualizando o projeto para Spring Boot 2.2 (Spring HATEOAS 1.0)
- 19.5. Resolvendo conflito de dependências com Spring HATEOAS e SpringFox
- 19.6. Conhecendo especificações para formatos Hypermedia
- 19.7. Adicionando hypermedia na representação de recurso único com HAL
- 19.8. Construindo links dinâmicos com WebMvcLinkBuilder
- 19.9. Construindo links que apontam para métodos
- 19.10. Adicionando hypermedia na representação de recursos de coleção
- 19.11. Montando modelo de representação com RepresentationModelAssembler
- 19.12. Desafio: adicionando hypermedia nos recursos de usuários
- 19.13. Corrigindo link de coleção de recurso de responsáveis por restaurante
- 19.14. Desafio: adicionando hypermedia nos recursos de estados
- 19.15. Adicionando hypermedia em recursos com paginação
- 19.16. Desafio: adicionando hypermedia em recursos de pedidos (paginação)
- 19.17. Corrigindo links de paginação com ordenação
- 19.18. Adicionando links com template variables
- 19.19. Desafio: adicionando template variables do filtro de pedidos
- 19.20. Refatorando construção e inclusão de links em representation model
- 19.21. Desafio: refatorando construção e inclusão de links
- 19.22. Adicionando links de transições de status de pedidos
- 19.23. Adicionando links condicionalmente
- 19.24. Desafio: adicionando hypermedia nos recursos de restaurantes
- 19.25. Desafio: adicionando links condicionais no recurso de restaurante
- 19.26. Desafio: adicionando template variable de projeção de restaurantes
- 19.27. Desafio: adicionando hypermedia nos recursos de formas de pagamento
- 19.28. Adicionando links para desassociação de formas de pagamento com restaurante
- 19.29. Adicionando links com template variable de caminho para associação de formas de pagamento com restaurante
- 19.30. Desafio: adicionando links de associação de restaurantes com responsáveis
- 19.31. Desafio: adicionando hypermedia nos recursos de produtos
- 19.32. Desafio: adicionando links para recurso de foto de produto
- 19.33. Desafio: adicionando hypermedia nos recursos de grupos
- 19.34. Desafio: adicionando links de associação de grupos com permissões
- 19.35. Desafio: adicionando links de associação de usuários com grupos
- 19.36. Implementando o Root Entry Point da API

- 19.37. Desafio: implementando endpoint com links de recursos de estatísticas
- 19.38. Comprimindo as respostas HTTP com Gzip
- 19.39. Corrigindo as propriedades de links na documentação
- 19.40. Corrigindo a documentação dos endpoints de cidades
- 19.41. Corrigindo a paginação na documentação
- 19.42. Desafio: corrigindo a documentação dos endpoints de estados
- 19.43. Desafio: corrigindo a documentação dos endpoints de formas de pagamento
- 19.44. Desafio: corrigindo a documentação dos endpoints de grupos
- 19.45. Desafio: corrigindo a documentação dos endpoint de pedidos (paginação)
- 19.46. Desafio: corrigindo a documentação dos endpoints de produtos
- 19.47. Desafio: corrigindo a documentação dos endpoints de restaurantes e usuários
- 19.48. Removendo modelo de representação inutilizado da documentação

20. Evoluindo e versionando a API

- 20.1. Evoluindo a API com gestão de mudanças
- 20.2. Evitando quebrar os clientes: nova propriedade no modelo
- 20.3. Evitando quebrar os clientes: exclusão de propriedade do modelo
- 20.4. Evitando quebrar os clientes: alteração de tipo de propriedade do modelo
- 20.5. Evitando quebrar os clientes: alteração na estrutura de dados do modelo
- 20.6. Evitando quebrar os clientes: alteração de URL de recurso
- 20.7. O que é e quando versionar uma API?
- 20.8. As principais técnicas de versionamento de APIs
- 20.9. As principais abordagens para manter a base de código de APIs versionadas
- 20.10. Preparando o projeto para versionamento da API por Media Type
- 20.11. Implementando o versionamento da API por Media Type
- 20.12. Definindo a versão padrão da API quando o Media Type não é informado
- 20.13. Implementando o versionamento da API por URI
- 20.14. Desafio: Refatorando controladores para adicionar /v1 nas URIs
- 20.15. Desafio: adicionando os recursos de cozinhas na v2 da API
- 20.16. Gerando documentação das versões da API com SpringFox e Swagger UI
- 20.17. Desafio: revisando documentação da v2 da API
- 20.18. Depreciando uma versão da API
- 20.19. Desligando uma versão da API

21. Logging

- 21.1. Introdução ao Logback e SLF4J
- 21.2. Desafio: registrando logs de exceptions não tratadas
- 21.3. Criando uma conta no Loggly: serviço de gerenciamento de logs na nuvem
- 21.4. Configurando o appender do Loggly no Logback
- 21.5. Configurando o Logback para alternar as configurações por Spring Profiles

22. Segurança com Spring Security e OAuth2

- 22.1. Introdução à segurança de REST APIs
- 22.2. Adicionando segurança na API com Spring Security
- 22.3. Configurando Spring Security com HTTP Basic
- 22.4. Configurando autenticação de usuários em memória
- 22.5. Introdução ao OAuth2
- 22.6. Soluções para OAuth2: nova stack do Spring Security vs Spring Security OAuth

- 22.7. Conhecendo o fluxo Resource Owner Password Credentials
- 22.8. Criando o projeto do Authorization Server com Spring Security OAuth2
- 22.9. Configurando o fluxo Authorization Server com Password Credentials e Opaque Tokens
- 22.10. Configurando o endpoint de introspecção de tokens no Authorization Server
- 22.11. Configurando o Resource Server com a nova stack do Spring Security
- 22.12. Conhecendo o fluxo para emitir e usar Refresh Tokens
- 22.13. Configurando o Refresh Token Grant Type no Authorization Server
- 22.14. Configurando a validade e não reutilização de Refresh Tokens
- 22.15. Conhecendo o fluxo Client Credentials
- 22.16. Configurando o Client Credentials Grant Type no Authorization Server
- 22.17. Revisando o fluxo Authorization Code
- 22.18. Configurando o Authorization Code Grant Type
- 22.19. Testando o fluxo Authorization Code com um client JavaScript
- 22.20. Conhecendo o fluxo Implicit
- 22.21. Configurando o fluxo Implicit Grant Type
- 22.22. Mais segurança com PKCE e Authorization Code Grant
- 22.23. Implementando o suporte a PKCE com o fluxo Authorization Code
- 22.24. Testando o fluxo Authorization Code com PKCE com o método plain
- 22.25. Testando o fluxo Authorization Code com PKCE com o método SHA-256
- 22.26. Testando um client JavaScript com PKCE e Authorization Code
- 22.27. Decidindo qual fluxo OAuth2 usar

23. OAuth2 avançado com JWT e controle de acesso

- 23.1. Armazenando tokens no Redis: um banco de dados NoSQL
- 23.2. Configurando o RedisTokenStore
- 23.3. Entendendo a diferença entre Stateful e Stateless Authentication
- 23.4. Transparent Tokens: conhecendo o JSON Web Tokens (JWT)
- 23.5. Gerando JWT com chave simétrica (HMAC SHA-256) no Authorization Server
- 23.6. Configurando o Resource Server para JWT assinado com chave simétrica
- 23.7. Entendendo a diferença entre assinatura com chave simétrica e assimétrica
- 23.8. Gerando um par de chaves com keytool
- 23.9. Assinando o JWT com RSA SHA-256 (chave assimétrica)
- 23.10. Desafio: criando bean de propriedades de configuração do KeyStore
- 23.11. Extraindo a chave pública no formato PEM
- 23.12. Configurando a validação de JWT no Resource Server com a chave pública
- 23.13. Revisando o fluxo de aprovação do Authorization Code com JWT
- 23.14. Autenticando usuário com dados do banco de dados
- 23.15. Desafio: refatorando serviços de usuários para usar BCrypt
- 23.16. Adicionando claims customizadas no payload do JWT
- 23.17. Obtendo usuário autenticado no Resource Server
- 23.18. Definindo e criando as permissões de acesso
- 23.19. Carregando as permissões concedidas na autenticação
- 23.20. Carregando as Granted Authorities e restringindo acesso a endpoints na API
- 23.21. Method Security: Restringindo acesso com @PreAuthorize e SpEL
- 23.22. Desafio: tratando AccessDeniedException no ExceptionHandler
- 23.23. Simplificando o controle de acesso em métodos com meta-annotações
- 23.24. Entendendo os escopos do OAuth2

- 23.25. Carregando Granted Authorities dos escopos do OAuth2 no Resource Server
- 23.26. Restringindo acesso a endpoints por escopos do OAuth2
- 23.27. Desafio: restringindo acesso dos endpoints de restaurantes
- 23.28. Restringindo acessos de forma contextual (sensível à informação)
- 23.29. Restringindo acessos com @PostAuthorize
- 23.30. Desafio: restringindo acessos ao endpoint de pesquisa de pedidos
- 23.31. Desafio: restringindo acessos aos endpoints de transição de status de pedidos
- 23.32. Desafio: restringindo acessos aos endpoints de formas de pagamentos
- 23.33. Desafio: restringindo acessos aos endpoints de cidades e estados
- 23.34. Desafio: restringindo acessos aos endpoints de usuários, grupos e permissões
- 23.35. Desafio: restringindo acessos aos endpoints de estatísticas
- 23.36. Configurando os clientes OAuth2 em um banco de dados SQL
- 23.37. Cadastrando clientes OAuth2 no banco de dados e testando a emissão de tokens
- 23.38. Corrigindo lógica de restrição de acessos para Client Credentials Flow
- 23.39. Gerando links do HAL dinamicamente de acordo com permissões do usuário
- 23.40. Desafio: gerando links do HAL dinamicamente de acordo com permissões
- 23.41. Juntando o Resource Server com o Authorization Server no mesmo projeto
- 23.42. Ajustando a documentação da API para suporte a OAuth2
- 23.43. Customizando a página de login
- 23.44. Customizando a página de OAuth2 Approval

24. Deploy em produção

- 24.1. Conhecendo o Twelve-Factor App
- 24.2. Onde fazer deploy?
- 24.3. Conhecendo a PWS (Pivotal Web Services)
- 24.4. Instalando o CLI do Cloud Foundry
- 24.5. Criando o bucket de produção na Amazon S3
- 24.6. Criando uma conta na PWS
- 24.7. Conhecendo o CLI do Cloud Foundry
- 24.8. Instanciando serviços com o marketplace da PWS
- 24.9. Configurando serviços externos com CUPS
- 24.10. Separando as configurações dos ambientes de produção e desenvolvimento com Spring Profiles
- 24.11. Adicionando propriedades adicionais com CUPS
- 24.12. Gerando o Fat JAR
- 24.13. Implantando a aplicação na PWS
- 24.14. Fazendo binding dos serviços
- 24.15. Subindo a aplicação e fazendo downgrade do Flyway
- 24.16. Inserindo dados no banco de produção
- 24.17. Testando o upload de fotos
- 24.18. Testando o envio de e-mail
- 24.19. Retirando as chaves de segurança do projeto
- 24.20. Conhecendo o Registro.br e Cloudflare
- 24.21. Registrando um domínio no Registro.br
- 24.22. Configurando a conta no Cloudflare
- 24.23. Escalando a aplicação
- 24.24. Blue-Green Deployment: implantando novas versões com zero downtime
- 24.25. Resolvendo problema de sessão com Spring Session e Redis

24.26. Conclusão e próximos passos

