**Anotações das vídeo aulas**

Vídeo 1.3

* Mostra uma ideia do que sejam as APIs
* Faz uma diferenciação entre Web Services e APIs
* Fala em “monetizar uma API”

Vídeo 1.4

* Apresenta o modelo de domínio de negócio da aplicação
* AlgaFoods – um MVP ([*Minimum Viable Product*](https://blog.sage.com.br/o-que-e-significado-mvp/)) de um sistema de vendas de comidas (tipo iFood).

Vídeo 2.1

* Responde a uma série de perguntas sobre Spring

Vídeo 2.2

* Ecossistema Spring
* Empresa responsável pelo projeto – Pivotal
* Spring.io

Vídeo 2.3

* Spring vs Jakarta EE (JEE)

Vídeo 2.4

* Spring Boot
  + Usa o conceito de Convention Over Configuration – oferece recursos que facilitam a vida do programador, de modo que este possa se concentrar mais no problema – domínio da aplicação – sem precisar se preocupar muito em ficar escrevendo códigos de configuração.
  + Usa uma visão opinativa (que pode ser substituída parcialmente – não precisa usar tudo como ele configura – mas não usar nada, não faria sentido utiliza-lo)
  + Spring não gera código ele apenas autoconfigura os projetos
* Spring Boot ou Spring MVC?
  + Spring Boot não é um framework para desenvolvimento Web, ele pode ser utilizado com este propósito, mas também para outros que não seja web
  + Usar Spring Boot, pode fazer uso do Spring MVC, mas o contrário não

Vídeo 2.5

* Mostra como criar um projeto usando o Spring Initializr dentro do *Spring Tool Suite*
  + Observação: o mesmo pode ser feito pela **Apache Netbeans IDE 11**, adicionando o plugin **NB SpringBoot.** 
    - Observação: caso seu ambiente de trabalho tenha diferente versões de Java isso pode dar conflito na IDE, para isso é necessário adicionar o valor **-J-Djdk.lang.Process.allowAmbiguousCommands=true** no final da variável **netbeans\_default\_options** do arquivo **netbeans.conf** que está na pasta “**C:\Program Files\NetBeans-11.1\_Apache\netbeans\etc**”. Caso isso não seja feito a aplicação pode não compilar, gerando o seguinte erro no console:

cd C:\project\_folder \InitializrSpringbootProject; "JAVA\_HOME=C:\\Program Files\\Java\\jdk-11.0.7" cmd /c "\"\"C:\\Program Files\\NetBeans-11.1\_Apache\\netbeans\\java\\maven\\bin\\mvn.cmd\" -Dmaven.ext.class.path=\"C:\\Program Files\\NetBeans-11.1\_Apache\\netbeans\\java\\maven-nblib\\netbeans-eventspy.jar\" -Dfile.encoding=UTF-8 clean install\""

Cannot run program "cmd" (in directory "C:\project\_folder\InitializrSpringbootProject"): Malformed argument has embedded quote: "C:\Program Files\NetBeans-11.1\_Apache\netbeans\java\maven\bin\mvn.cmd" -Dmaven.ext.class.path="C:\Program Files\NetBeans-11.1\_Apache\netbeans\java\maven-nblib\netbeans-eventspy.jar" -Dfile.encoding=UTF-8 clean install

Vídeo 2.6

* Instruções para construir e executar o projeto pelo prompt de comando
* Build do projeto
  + C:\project\_folder>mvnw package
    - Este comando fará o build do projeto, enviando o arquivo de distribuição jar para a pasta *target*
* Execução do projeto
  + C:\project\_folder>java -jar target\jar\_archive.jar
* Comando para limpar o build do projeto
  + C:\project\_folder>mvnw clean
* Comando para visualizar a árvore de dependências do projeto
  + C:\project\_folder>mvnw dependency:tree
* Comando para visualizar as dependências resolvidas
  + C:\project\_folder>mvnw dependency:resolve
* Comando para visualizar o POM efetivo (união de todos os POM do projeto)
  + C:\project\_folder>mvnw help:effective-pom
* Explorando o POM.xml
  + Hierarquia de dependências
    - Dependências transitivas
      * Dependências resolvidas
* O Maven faz um repositório local de todas as dependências que incluímos no projeto, mantendo esse repositório atualizado conforme as versões que estamos utilizando para todos os projetos.

Vídeo 2.7

* Implementação do primeiro Controller usando Spring MVC (MVC para WEB)
  + @Controller
  + @GetMapping
  + @ResponseBody
* Com isso já é possível carregar o tomcat e executar o projeto no servidor
  + Para teste do controller requisitar no navegador:
    - Localhost:8080/hello

Vídeo 2.8

* Usando DevTools do Spring para agilizar o carregamento de uma atualização
  + No STS há um atalho no menu de contexto de projeto para adicionar:
    - Spring 🡪 add DevTools
  + Em outras IDEs pode-se adicionar diretamente no pom.xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

</dependency>

Observação: o DevTools usa dois class loaders, um de base e outro de restart (para agilizar o processo de reinicialização do serviço após uma atualização)

Vídeo 2.9

* Fala da injeção de dependência (DI – *Dependency Injection*)
* Mostra o caminho para entender injeção de dependência (de forma manual) a partir de um projeto de notificação de Email e SMS usando uma situação sem polimorfismo e outra com (para implementar IoC (Inversion of Control)) de modo a implementar o baixo acoplamento. Com a IoC conseguimos fazer a injeção de dependência.
* Enfim, nesta aula é apresentada como usar polimorfismo para fazer a IoC com foco na DI
* IoC – inversão de controle – a instanciação do tipo de mensagem foi retirada do serviço, tornando as classes de serviços menos independente (acoplamento fraco), fazendo com que o método de notificação torne-se mais dinâmico, exigindo menor grau de atualização de código em eventuais manutenções.
* DI é uma forma de IoC

Vídeo 2.10 – Spring IoC Container

* SpringContext – responsável por fazer toda a mágica da injeção de dependência do Spring Framework
* Quando o IoC container inicializar ele começa a instanciar os beans que ele deve gerenciar. Bans são objetos gerenciados pelo Spring. Bean ou Dependency, são objetos gerenciados pelo SpringContext.
* Este Container de IoC do Spring é representado pela Interface ApplicationContext

Vídeo 2.11 – Definindo Beans com @Component

* No Spring os beans gerenciados são chamados de componentes, e para que possam ser criados pelo Spring é necessário declarar as classes com a anotação @Component
* Os beans gerenciáveis são utilizados para fazer a injeção de dependência.
* O Spring faz uso do recurso Component Scanning para fazer uma leitura de todas as classes anotadas com @Component e, logo, instanciá-las para o seu gerenciamento.

Vídeo 2.12 – Injetando dependências (Spring beans)

* Toda classe anotada como @Component é um bean gerenciável pelo Spring, logo, eles podem ser injetados umas nas outras.
* Nos exemplos foram mostradas as injeções de dependências pelo Spring nos construtores de classes.

construtor chamado.... NotificacaoEmail:

AtivacaoClienteService: net.map.myalgafood.di.notificacao.NotificacaoEmail@231676a4

MeuPrimeiroController: net.map.myalgafood.di.service.AtivacaoClienteService@2bd34bbe

* No exemplo, é utilizado um acoplamento forte AtivacaoClienteService e NotificadorEmail, pois o tipo é específico. Logo, é necessário definir uma estratégia para tornar num acoplamento fraco, para isso, se faz uma Interface (no Eclipse IDE é possível extrair uma interface a partir de uma classe concreta).
* Neste vídeo aprendemos como definir um bean com @Component, no próximo será visto com criar um bean de forma diferente, por meio de uma configuração de bean.

Vídeo 2.13 - Usando @Configuration e @Bean para definir beans

* Problema: O notificador de Email passa a ter duas configurações essenciais na sua inicialização – uma é a exibição (caixa alta ou não) e outra o SMTP do email. Neste caso, para fazer a injeção de dependência é necessário que esses valores sejam conhecidos, logo, como configurar esses valores em um bean gerenciável pelo Spring? Uso de @Configuration e @Bean
* Neste exemplo, haverá a necessidade de customizar a instanciação do Bean (configuração de inicialização), para isso a necessidade de implementar outras classes de configuração do Bean. Caso contrário o erro a seguir será disparado na inicialização:

APPLICATION FAILED TO START

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Description:

Parameter 0 of constructor in net.map.myalgafood.di.notificacao.NotificacaoEmail required a bean of type 'java.lang.String' that could not be found.

Action:

Consider defining a bean of type 'java.lang.String' in your configuration.

* Corrigindo o problema, deixamos de usar a anotação @Component e passamos usar @Configuration e @Bean em outras classes que servirão de base para a criação do bean NotificacaoEmail
* Uma das vantagens de fazer isso é que ao tirar o @Component a classe passa a ser uma classe normal Java e não tem mais a dependência do Spring, delegando esta tarefa para outra classe. Mas vale destacar que isso não é um problema, cada situação deve ser avaliada para definir quando ou não criar classes de configuração e inicialização de beans gerenciáveis.

Vídeo 2.14 - Conhecendo os pontos de injeção e a anotação @Autowired

* Pode ser feito de três formas
  + Pelo método construtor (é um dos melhores locais para se definir o ponto de injeção)
  + Por método setter
  + No atributo

Vídeo 2.15 Dependência opcional com @Autowired

* @Autowired(required = false)
  + Esta instrução irá dizer como ter uma dependência opcional dentro de um bean
* Para fazer o teste é necessário tirar @Component da classe NotificacaoEmail que é um componente Spring.