# Vídeo 1 (12.10.2017) - Introdução

# SOFEA (Service-Oriented Front-End Architecture)

+ Arquitetura como todo front/end

+ tem a vantagem de, separar o back do front, o front será baixado.

Somente uma vez, depois disso, o uso da rede fica.

Somente para trafego de JSON do back

# SPA (Single Page Aplication)

Mais em front end, pra q tudo funcione em uma única página só.

#Projeto

+ Angular para FrontEND

+ Spring Boot para Back END

Vamos Gerar BUILD

+ Front ==> Recurso Estáticos HTML e CSS

+ Back ==> arquivo jar.

DEPLOY

+ Front ==> Servidor de Downloads (HTTP) Simples - como amazon

+ Back ==> executar a app e levantar o tomcat

Benefícios

+ Escalabilidade, pois front/end vão ficar bem separados.

+ Da pra ter times especialistas, eh bem separado o front/back.

+ Interoperabilidade: O front pode ser lido por desktop/celular enquanto

Que o back pode ser em qualquer linguagem, não ha dependências técnicas entre eles.

#FRONT. END-ANGULAR

ANGULAR: framework JS pra fazer FrontEND web, são varias bibliotecas, você desenvolve fazendo HTML orientado a template. Ela eh bem produtiva. O primeiro eh ANGULAR2 depois foi pra ANGULAR4, mas não existe o Angular 3.

O angular separa em componentes, onde cada um tem seu próprio HTML e CSS, isso da à vantagem de poder reaproveitar esses componentes.

// Lembre-se: Não tente reinventar a roda

Esses componentes podem ser encontrados na net:

ng-bootstrap: https://ng-bootstrap.github.io/#/home

prime-ng: https://www.primefaces.org/primeng/

A primeng eh a mais completa

TYPESCRIPT: Programação que compila e vira JavaScript, ele não roda em navegadores. Eh uma linguagem baseada em Java Script. Familiar a JAVA.

ANGULAR x ANGULARJS: apesar dos nomes serem parecidos, são bem diferentes e não são compatíveis, o ANGULARJS eh mais antigo que o ANGULAR

#BACK. END-SRPING

+ REST: Pepresentational State Transfer: EH um modelo Arquitetural

+ Benefícios:

+ organização

+ conseguimos evoluir de forma incremental

+ o SW fica altamente escalável

+ Independe de Tecnologia já que eh algo arquitetural

+ Orientações pra esse modelo pelo criador, pra dizer que a app eh REST.

+ Deve ser no modelo: cliente/servidor

+ não deve manter um estado

+ Deve ter cache, mas não em todos os pontos.

+ em camadas (Elemento intermediários de forma transparente

+ Código sob demanda: um código eh buscado na app e executado no cliente

+ REST você RESTFUL:

+ REST eh um modelo arquitetural

+ RESTFUL eh a aplicação desse modelo

+ PROTOCOLO HTTP:

+ orientado a CIENTE que faz REQUEST HTTP e SERVIDOR que retorna

RESPONSE, ou seja, CLIENTE/SERVIDOR.

+ POSTMAN

+ Nos ajuda a fazer requisições HTTP

+ SPRING

+ Eh como uma incubadora de projetos

+ SPRING MVC:

+ nos ajuda a tratar requisições e respostas

+ SPRING DATA JPA:

+ Eh como uma abstração acima do JPA

+ Nos ajuda dando algumas operações Pré-implementadas

+ SPRING BOOT:

+ facilita muito o desenvolvimento

+ ajuda na configuração

# Vídeo 2 (13.10.2017) – FrontEND

# O que eh e não eh REST

+ a URI ('u erri i') (URL da nossa APP) deve ser sempre a mesma, a

Única coisa que deve mudar eh QUAL O METODO HTTP usar

(POST APAGUE, GET).

Ex:

Posso fazer POST -> /Salvar Produto

Isso eh ruim

Eh melhor fazer POST -> /produto

Ou seja, não criar URI só pra fazer uma coisa.

Tentar reaproveitar

+ O ideal para os erros eh voltar o tipo de erro HTTP SEMPRE

# PASSOS DO VIDEO A PARTIDE DE 16 MIN

+ em application. propeties, coloca pra formatar as datas

spring.mvc.date-format=yyyy-MM-dd

+ apaga pastas static e template

+ crio a interface ItemReposity, ela ira estender JPA repository·.

Por conta disso, ela será criada em tempo de exe. pelo Spring DATA.

Então eh algo bem simples

+ Vamos criar a classe que vai mapear as requisições

+ cria o pacote patrimônio/resource

+ cria a classe Item Resource

+ colocamos os marcados @RestController pra pegar requisições HTTP

+ criamos ItemReposity como @Autowiered, pra já ter injetado uma instancia pras nossa consultas

+ criamos o método lista e mapeamos como sendo um método GET

Por isso eh @GetMapping ("/item")

+ em application.propertie, vamos criar um arq. SQL pra fazer uns inserts

Mais simples para podermos testar

...

Falta o restante

# Vídeo 3 (15.10.2017) - Angular

# vsc: Vamos usar Visual Studio Code pra editar o Time Script, chamado VS Code

# node js: plataforma pra construir app em JS que roda no servidor, o angular não precisa, pois roda no navegador, mas nos precisamos no node no desenvolvimento pra programar.

# npm: gerenciador de pacotes, eh como um maven para o Java script.

# angular cli: serve pra agilizar a programação do angular, uma instala

+ a partir do npm, vamos executar o comando, g --> global == no PC:

>npm install -g @angular/cli

+ abrir o servidor no visual studio code, procuramos o terminal e

Digitamos 'ng serve', serve pra agente testa. Vai ficar na porta 4200

> ng serve

+ criar um projeto, como no maven:

> ng new [nome-proj]

# cmd:

-verifica com node -v e nmp -v pra ver se esta instalado no shell

-clear no shell pra limpar a tela

- verifica se o angular cli esta instalado usando ng -v

# Key Short:

https://code.visualstudio.com/docs/getstarted/keybindings#\_keyboard-shortcuts-reference

# Inicio Real aos 9 minutos

+ uma app angular presisa de pelo menos um modulo, este foi o.

src/app/app.mode.js

+ Possui metas informações no @NgModule

+ ele declara AppComponent, que diz que app module tem ngmodule

+ a partir do app componente agente começa a add outros

+ em imposrt agente adiciona módulos q o modulo depende

pode ser já prontos na net ou algo nosso

#app.component

+ responsável por mostrar a view

+ ele possui uma classe que implementa a lógica

Configuramos com o decorator componente

+ usa templates

### INÍCIO DA CONSTRUÇÃO ###

# em app.component.html

# Fazer a barra azul 'navebar' e, ela é um componente isolado, pra podermos também reaproveita-la no futuro.

# nav bar

+ digitamos no terminal

> ng g c navbar --spec=false

+ o -- spec=false eh pra nao criar arq. De teste

+ g de generate e c de component

+ Um componente pode usar outro

# install prime ng == biblioteca de componentes

> npm install primeng --save

O save quer dizer que eh uma dependência do projeto

# gerar serviço

> ng g s {nome-service //coloca service na frente auto}

#ctrl+c ==> mata o cmd

# f12 pra poder vê as coisas no browser do googlechorme

# Vídeo 4 - Respondendo a Dúvidas (16.10.2017)

# Eh seguro deixar o front-end exposto

+ Resposta: sim, pois a parte de segurança deve ser feita no back-end,

Toda a parte de segurança deve ser feita pela API/

+ O front não recebe nada de segurança. Agente ate pode validar campos

No front pra ficar legal, mas, isso deve ser realmente feito.

No back-end. Agente valida no FRONT para que o usuário já saiba que.

Inserir dados errados não vai funcionar, o que melhora a.

Usabilidade do software

# Como funciona a SEGURANAÇ no RESTFUL

+ No modelo cliente servido, o usuário inicia enviando um.

Usuário e senha e o servidor retorna um cookie e que também fica

Armazenado no servidor. Enfim, eh usando SESSION.

+ Agora, na nossa API Rest (Tomcat com Spring Boot).

Agente mandaria user/senha, e, a APIRest retorna um TOKEN. A API

Rest eh StateLess, ela no mantem estado. Assim se der algo de

Errado nela, agente não perde usuário.

NAO HÁ SESSION

Quando volta uma requisição. Do cliente, o cliente manda de volta um.

token, e, ai, volta JSON.

# Se eu usar uma biblioteca de componentes eu fico preso a ela?

+ Sim e não. Todas essas bibliotecas possuem algo chamado TEMA/THEMES

Elas mudam em geral as cores e um pouco a forma visual dos componentes como um todo, como globalmente.