



Processo Seletivo Invillia

Hello World!

Meu nome é Casseb.

Este é o documento relacionado ao processo seletivo para Programador Java Pleno da empresa Invillia.



Vamos começar por um mapa.

Material divido em:

- Tasks
- Non functional requirements
- Nice to have features
- Design
- Frameworks Interessantes

Irei detalhar o que foi desenvolvido e o que foi planejado nestes aspectos.



Tasks

Criação de uma Loja

O que foi pedido

Foi solicitado rotas
Rest para criar uma
loja.

O que foi desenvolvido

Foi desenvolvido uma
rota post que recebe
as características da
loja gravando no
banco de dados.

Postman

The screenshot shows the Postman application interface. At the top, it displays a POST method and the URL `localhost:8080/store`. Below the URL, there are tabs for Params, Authorization, Headers (1), Body (highlighted in green), Pre-request Script, and Tests. Under the Body tab, there are options for none, form-data, x-www-form-urlencoded, raw, binary, and JSON (application/json) (selected). The JSON body is defined as follows:

```
1 {  
2   "name" : "StoreTest",  
3   "address":{  
4     "streetAddress" : "Rua A",  
5     "city" : "Cidade B",  
6     "state" : "SP",  
7     "zipCode" : "12226620"  
8   }  
9  
10 }
```

Teste Integrado

```
@Test
public void createStore() {
    given().
        body(getStoreTest()).
        contentType(ContentType.JSON).
    when().
        post(REST_STORE).
    then().
        assertThat().
            statusCode(200).
        body("name", equalTo(getStoreTest().getName()));
}
```

Edição de uma Loja

O que foi pedido

Foi solicitado rotas Rest para editar uma loja já existente.

O que foi desenvolvido

Foi desenvolvido uma rota put que recebe as características da loja atualizando no banco de dados.

Postman

PUT localhost:8080/store/1

Params Authorization Headers (1) **Body** ● Pre-request Script Tests ●

none form-data x-www-form-urlencoded raw binary JSON (application/json) ▾

```
1 {  
2   "name" : "StoreTest Editado",  
3   "address":{  
4     "streetAddress" : "Rua A",  
5     "city" : "Cidade B",  
6     "state" : "RJ",  
7     "zipCode" : "12226620"  
8   }  
9 }
```

Teste Integrado

```
@Test
public void updateStoreInformation() {
    Integer originalstoreId = getPersistedStore();
    String altName = "AlterName";
    Store storeEdited = getStoreTest();
    storeEdited.setName(altName);

    given().
        body(storeEdited).
        contentType(MediaType.JSON).
        pathParam("id",originalstoreId).
    when().
        put(REST_STORE+REST_STORE_PUT).
    then().
        assertThat().
            statusCode(200).
            body("name", equalTo(altName));
}
```

Consultar uma Loja

O que foi pedido

Foi solicitado rotas Rest para consultar uma loja já existente.

O que foi desenvolvido

Foi desenvolvido uma rota get que pode consultar via query parameters lojas com qualquer um dos atributos.

Postman

GET localhost:8080/store/?name=StoreTestA

Params • Authorization Headers (1) Body Pre-request Script

KEY	VALUE
<input checked="" type="checkbox"/> name	StoreTestA
Key	Value

Body Cookies (1) Headers (3) Test Results

Pretty Raw Preview JSON ▾

```
1 [  
2 {  
3   "id": 782,  
4   "name": "StoreTestA",  
5   "address": {  
6     "streetAddress": "Rua A",  
7     "city": "Cidade B",  
8     "state": "SP",  
9     "zipCode": "12226620"  
10    }  
11  }  
12 ]
```

Teste Integrado

```
@Test
public void retrieveStoreByParameters() {
    Integer originalstoreId = getPersistedStore();

    given().
        queryParam("id", originalstoreId).
        queryParam("name", getStoreTest().getName()).
    when().
        get(REST_STORE).
    then().
        assertThat().
            statusCode(200).
            body("name", equalTo(Arrays.asList(getStoreTest().getName())));
}
```

Criar um Pedido com itens

O que foi pedido

Foi solicitado rotas
Rest para adicionar
um pedido com itens.

O que foi desenvolvido

Foi desenvolvido uma
rota post que insere
um pedido e seus itens
no banco de dados.

Postman

The screenshot shows the Postman application interface. At the top, it displays a POST request to the endpoint `localhost:8080/order`. Below the request type and URL, there are tabs for `Params`, `Authorization`, `Headers (1)`, `Body`, `Pre-request Script`, and `Tests`. The `Body` tab is currently selected, indicated by an orange underline and a green dot. Under the `Body` tab, there are several options: `none`, `form-data`, `x-www-form-urlencoded`, `raw` (which is selected and highlighted in orange), `binary`, and `JSON (application/json)`. The `JSON (application/json)` option has a dropdown arrow next to it. The main area below these options contains a JSON payload with line numbers from 1 to 18. The JSON structure is as follows:

```
1  {
2    "address": {
3      "streetAddress": "RuaB",
4      "city": "Cidade B",
5      "state": "SP",
6      "zipCode": "12226620"
7    },
8    "orderItems": [
9      {
10        "description": "ItemA",
11        "unit": 1.5,
12        "quantity": 2
13      },
14      {
15        "description": "ItemB",
16        "unit": 4,
17        "quantity": 2
18      }
]
```

Teste Integrado

```
@Test
public void createOrderWithItems() {
    given().
        body(getOrderWithItems()).
        contentType(MediaType.JSON).
    when().
        post(REST_ORDER).
    then().
        assertThat().
            statusCode(200).
            body("orderItems.size()", equalTo(5));
}
```

Criar um Pedido com itens

O que foi pedido

Foi solicitado rotas
Rest para pagar um
pedido.

O que foi desenvolvido

Foi desenvolvido uma
rota post que insere
um pagamento de um
pedido no banco de
dados.

Postman

The screenshot shows the Postman application interface. At the top, it displays a POST method and the URL `localhost:8080/payment`. Below the URL, there are tabs for Params, Authorization, Headers (1), Body (highlighted with an orange underline), Pre-request Script, and Tests. Under the Body tab, there are five options: none, form-data, x-www-form-urlencoded, raw (selected), binary, and JSON (application/json). The JSON (application/json) option is also highlighted with an orange underline. The raw section contains the following JSON payload:

```
1 {  
2     "orderId" : 1,  
3     "creditCardNumber" : "12345"  
4 }
```

Teste Integrado

```
@Test
public void createPaymentForAnOrder() {
    Integer orderId = getPersistedOrder();

    given().
        body(getPayment(orderId)).
        contentType(MediaType.APPLICATION_JSON).
    when().
        post(REST_PAYMENT).
    then().
        assertThat().
            statusCode(200).
            body("status", equalTo(INITIAL_PAYMENT_STATUS.name()))
            .and()
            .body("orderId", equalTo(orderId));
}
```

Consultar um Pedido

O que foi pedido

Foi solicitado rotas
Rest para consultar
um pedido.

O que foi desenvolvido

Foi desenvolvido uma
rota get que consulta
os pedidos pelo
número do id.

Postman

GET localhost:8080/order/783

Body Cookies (1) Headers (3) Test Results

Pretty Raw Preview JSON 

```
1 {  
2   "id": 783,  
3   "confirmationDate": "2019-02-24",  
4   "status": "PROCESSING",  
5   "address": {  
6     "streetAddress": "RuaB",  
7     "city": "Cidade B",  
8     "state": "SP",  
9     "zipCode": "12226620"  
10 },  
11   "orderItems": [  
12     {  
13       "id": 786,  
14       "description": "ItemA",  
15       "unit": 1.5,  
16       "quantity": 2  
17     },  
18     {  
19       "id": 785,  
20       "description": "ItemC",  
21       "unit": 3.8,  
22       "quantity": 2  
23     },  
24   ]  
25 }
```

Teste Integrado

```
@Test
public void retrieveOrderByParameter() {
    Integer orderId = getPersistedOrder();

    given().
        pathParam("id", orderId).
    when().
        get(REST_ORDER+REST_ORDER_GET).
    then().
        assertThat().
            statusCode(200).
            body("orderItems.size()", equalTo(5));
}
```

Devolver um pagamento

O que foi pedido

Foi solicitado rotas Rest para devolver o pagamento de um pedido ou itens.

Para devolver era necessário que o pedido estivesse pago e até 10 dias após a confirmação.

O que foi desenvolvido

Foi desenvolvido uma rota post realiza a devolução somente do pedido como um todo (Não foi desenvolvido devolução por itens).

Devolução de items

Proposta de solução

Para realizar seu desenvolvimento seria necessário usar a mesma lógica que a devolução do pedido como um todo aplica par conferir se o pedido pode ser devolvido. Além disso os itens deveriam ser enviados no post para que o back-end possa saber exatamente qual item deve ser devolvido.

Uma sugestão seria aplicar status para os itens, para não precisar remove-los do pedido mantendo um histórico.

Postman

POST ▼ localhost:8080/payment/refund/783

Params Authorization Headers (1) Body

KEY
Key

Body Cookies (1) Headers (3) Test Results

Pretty Raw Preview JSON ▼ 🔗

```
1 {  
2   "result": "Refunded"  
3 }
```

Teste Integrado

```
@Test
public void refundOrder() {
    Integer orderPaid = getOrderWithPaymentPaid();

    given().
        pathParam("orderId", orderPaid).
        contentType(ContentType.JSON).

    when().
        post(REST_PAYMENT+REST_PAYMENT_REFUND).
    then().
        assertThat().
            statusCode(200).
            body("result",is(PAYMENT_REFUNDED));
}
```



Non functional requirements

O que foi desenvolvido

O Básico

Neste portfólio foi desenvolvimento somente até os requisitos funcionais no formato monolítico. Do ponto de vista de MVP para validar a ideia com o cliente e testes, o desenvolvimento atende os requisitos mas para uso em larga escala, real-time e resiliente faltou a aplicação de conceitos de microsserviços como o Spring Cloud.

Proposta de Solução

Spring Cloud Netflix

Para que a aplicação possa atender uma maior escalabilidade e tolerância a falhas, não podemos apostar tudo em um único cavalo, pois se ele tropeça, perdemos tudo, microsserviços trabalha exatamente neste ponto, dividindo execução e armazenamento em nós para que, caso um se sobrecarregue ou caia, outro possa tomar o lugar.

Proposta de Solução

Service Discovery

O conceito de Service Discovery pode ser aplicado na solução que eu desenvolvi sem perder sua regra de negócio, basta realizar alguns desenvolvimentos extras separados em 3 partes:

- Service Registry
- Service Provider
- Service Consumer

Proposta de Solução

Service Registry

Utilizando Eureka para gerenciar estes nós e ganhar performance, a aplicação Spring deve utilizar a anotação `@EnableEurekaServer` na classe main e configurar porta e desabilitar seu registro pelo arquivo de configuração (`application.yml`)

Proposta de Solução

Service Provider

Para criar um nó para prover os microsserviços, basta utilizar a anotação `@EnableDiscoveryClient` e apontar o caminho do Service Registry no carquivo de configuração (`application.yml`)

Proposta de Solução

Service Consumer

Para utilizar os serviços Rests de forma facilitada, basta utilizar nos nossos clientes a classe DiscoveryClient para que o consumo seja feito de forma balanceada e tolerante a falhar.

Proposta de Solução

Implantação

Após tudo programado e executando localmente, seria necessário a hospedagem destes nós em algum serviço em nuvem como o AWS e adquirir planos que forneçam a escalabilidade necessário para a aplicação.



Nice to have features

Processo Assíncrono

Proposta de solução

A solução desenvolvida não aplica de forma explícita processos assíncronos. De forma geral, estes tipos de processos organizam a divisão de tarefas para serem realizados simultaneamente e unir novamente o resultado.

Em meu [Github](#) tenho um exemplo de processo assíncrono onde o método `find()` se divide para executar mais rapidamente.

Processo Assíncrono

Proposta na aplicação challenge

Para a realizada do desafio proposto, o ideal seria realizar a consulta de pedidos de forma assíncrona, pois seria o ponto da aplicação que poderia mais necessitar de alta performance.

Outro local que poderia ser aplicado seria na devolução do pedido, o cliente poderia solicitar recebendo de imediato a confirmação que o pedido de devolução está em andamento, enquanto isso em outra thread o processo de regra de negócio oficializa a solicitação informando o cliente via e-mail.

Banco de Dados

O que foi desenvolvido

O Banco de dados do desafio utilizado foi o MySQL, para uma solução com estes requisitos um banco relacional atende as necessidades.

Em caso de uma aplicação monolítica hospedada local a escolha ideal seria o Oracle.

Em caso de uma aplicação onde há uma grande variedade de diversidade nos campos ou necessidade de muitos inserts (como é o caso de IoT onde é realizado muitas inserções por segundo) o recomendável seria um banco não-relacional.

Docker

Proposta de Solução

Docker permite o empacotamento da aplicação e ambiente com suas dependências em um container.

Utilizando este recurso no desafio seria possível empacotar o banco mysql, por exemplo, com tudo que é necessário para sua execução, desta forma um outro programador ou o ambiente em nuvem precisaria somente executar seu DockerFile para ter seu ambiente configurado.

Proposta de Solução

AWS está ganhando cada vez mais destaque com plataforma para serviços de cloud, oferecendo ambientes em nuvem auto escaláveis.

A hospedagem da solução no AWS seria parte da aplicação dos requisitos não funcionais de escalabilidade e tolerância a falhas, já que toda esta parte de hardware ficaria a cargo da Amazon.

Security

Proposta de Solução

Para que uma solução Rest seja segura o Spring possui o Spring Security.

Para aplica-lo é necessário primeiramente possuir alguma tabela que registre usuários e senhas que utilizarão a aplicação (Caso contrário a autenticação terá que ser feita em memória o que não é aconselhável em produção), com elas em mãos basta aponta-los aos Spring informando qual a consulta correta e indicar quais rotas estão livres para acesso sem credenciamento e quais são obrigatórias.

Security

Mais alguns detalhes sobre Security

É importante lembrar que, uma vez aplicado a configuração, sempre será necessário estar logado para utilizar as rotas, para isso é possível usar a guia Authorization do postman para realizar os testes.

The screenshot shows the 'Authorization' tab in the Postman interface. The 'TYPE' dropdown is set to 'Basic Auth'. A note in the sidebar says: 'Heads up! These parameters hold sensitive data. To keep environment, we recommend using variables.' Below the dropdown, it says: 'The authorization header will be automatically generated when you send the request. Learn more about authorization'. At the bottom left is a 'Preview Request' button. On the right, there are fields for 'Username' (adm) and 'Password' (*****), with a 'Show Password' checkbox.

Swagger

Proposta de Solução

Swagger fornece uma interface visual para que seja possível interagir com seu ambiente Rest, permitindo a criação de exemplos para facilitar o usuário que irá consumir suas rotas.

No Python, por exemplo, é possível modelar as rotas pelo Swagger e apontar qual método deseja que seja chamado, desta forma a própria documentação gera o serviço e o visual para interagir.

Clean Code

Utilização no Desafio

No desafio diversos pontos usaram as boas práticas de um código limpo, nos próximos slides irei dar exemplos.

Clean Code

Não comentar o óbvio

No exemplo abaixo o nome da função representa exatamente o que ela faz, dispensando a necessidade de comentários.

```
public OrderStore retrieveById(Integer orderId){  
    return orderRepository.findById(orderId).orElse(new OrderNull());  
}
```

Clean Code

Utilizar booleanos de forma implícita

Neste exemplo, em vez de usar um return true ou return false, utilizei o próprio retorno booleano, desta forma o código fica mais claro e fácil de entender.

```
private boolean orderAvailableToRefund(Integer orderId) {  
    return (orderConfirmedUntilTenDays(orderId) && isPaidOrder(orderId));  
}
```



Design

Design Patterns

Optional

Em alguns momentos o desenvolvedor precisa preparar o software para tratar objetos quando podem ser nulos, fugindo do temível nullpointer.

Dentro do Java 8 foi adicionado o Optional, implementado pelo Spring Boot versão 2+ para tratar consultar no banco que podem não trazer resultado, um Optional pode ter ou não um valor, sendo possível conferir se tem valor pelo método `isPresent()` ou dando uma outra opção caso não tenha pelo `orElse()`.

Design Patterns

Null Object

É possível criar uma classe que representa um valor nulo que extenda seu alvo sobreescrevendo os métodos get para tratar objetos nulos, como mostro abaixo, um objeto caso eu precise representar um Order nulo.

```
public class OrderNull extends OrderStore {  
    @Override  
    public Integer getId() {  
        return 0;  
    }  
}
```

Design Patterns

Null Object + Optional

Unindo os dois conceitos é possível consultar no banco um order ao mesmo tempo que trata o retorno caso não encontre o alvo, como mostrado abaixo.

```
public OrderStore retrieveById(Integer orderId){  
    return orderRepository.findById(orderId).orElse(new OrderNull());  
}
```

Constantes

Strings Constantes

No código há momentos que devemos setar Strings que não serão alteradas durante a execução do código, como exemplo temos o path de uma rota Rest, para estas situações, compensa criar uma classe para armazenar todas as constantes do sistema.

Constantes

Aplicação no desafio

Criei uma classe chamada Constants que possui as chamadas Rests e definição de Status iniciais das Orders e Payments, desta forma posso vincular testes unitários e seu uso mantendo o ponto de troca somente em um lugar, ou seja, se for necessário editar o status inicial das orders será necessário somente editar na classe Constants.

```
//Defaults
public static final OrderStatus INITIAL_ORDER_STATUS = OrderStatus.PROCESSING;
public static final PaymentStatus INITIAL_PAYMENT_STATUS = PaymentStatus.PENDING;
```

Perfil

Aplicação no desafio

No desafio foi separada o `application.properties` para 3 arquivos `yml` para seguir os seguintes propósitos:

- `Application.yml`: Contem todas as configurações gerais da aplicação.
- `Application-dev`: Contem configurações somente do ambiente de desenvolvimento. (Nela deixei a configuração do banco de dados apontando para variáveis de ambiente e log de debug).
- `Application-prod`: Contem configurações somente para ambiente de produção. (Para usa-lo basta configurar os scripts de configuração para executar `mvn spring-boot:run -Dspring.profiles.active=prod`)



EJB 3



JDBC



JUnit

Struts²



Frameworks Interessantes

Lombok

“Laziness is a virtue!” Lombok

Utilizei neste projeto este fantástico framework, com ele foi possível deixar minhas classes muito mais claras com as seguintes anotações:

- **@Data:** Aplica aos atributos de uma classe Getters, Setters, ToString, Equals eHashCode.
- **@Builder:** Aplica o Design Pattern Builder na classe assinada.
- **@Slf4j:** Permite o uso direto de Log na classe em a necessidade de criar explicitamente a variável log.

Rest Assured

Testes de Integração

Para testes de integração utilizei o Rest Assured, seu grande diferencial fica na facilidade de customizar testes integrados, seguindo o padrão Given, When, Then de forma que o resultado final se torna uma leitura do que o teste faz.

```
@Test  
public void createOrderWithItems() {  
    given().  
        body(getOrderWithItems()).  
        contentType(MediaType.JSON).  
    when().  
        post(REST_ORDER).  
    then().  
        assertThat().  
            statusCode(200).  
            body("orderItems.size()", equalTo(5));  
}
```

JHipster

A cereja do bolo

Deixei por último o framework que faz com que praticamente tudo que foi tratado neste material pareça coisa de criança.

Jhipster é um framework que auxilia na configuração inicial de um projeto Spring Boot (semelhante ao que é feito no Spring Initializr) mas leva para outro patamar.

JHipster

Possibilidades

Com ele é possível configurar desde os pontos mais básicos até os mais avançados, segue algumas das tecnologias que ele manipula.

- Spring Security
- Swagger
- Docker
- AWS

JHipster

Modelagem de entidades

Repare no slide anterior que ele cobre praticamente todos os requisitos não funcionais, além disso ainda é possível modelar pelo JDL como será sua entidades para que ele gere Repository, Services e Rest com Swagger, tudo com uma interface visual em angular com login e Security.

JHipster

Uso no desafio

O desafio tinha como propósito observar o passo a passo do desenvolvimento, mas somente para caráter de teste apliquei os requisitos utilizando Jhipster configurando como microserviço. Sendo necessário criar uma aplicação para processar e um gateway para conferir visualmente.

JHipster

Configuração Microservice

```
> Which *type* of application would you like to create? Microservice application
> What is the base name of your application? acme
> As you are running in a microservice architecture, on which port would like your server to run? It should be unique to avoid port conflicts. 8081
> What is your default Java package name? com.invillia
> Which service discovery server do you want to use? JHipster Registry (uses Eureka, provides Spring Cloud Config support and monitoring dashboards)
> Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
> Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
> Which *production* database would you like to use? MySQL
> Which *development* database would you like to use? MySQL
> Do you want to use the Spring cache abstraction? Yes, with the Ehcache implementation (local cache, for a single node)
> Do you want to use Hibernate 2nd level cache? Yes
> Would you like to use Maven or Gradle for building the backend? Maven
> Which other technologies would you like to use? Asynchronous messages using Apache Kafka
> Would you like to enable internationalization support? No
> Besides JUnit and Jest, which testing frameworks would you like to use?
> Would you like to install other generators from the JHipster Marketplace? (y/N) N
```

JHipster

Configuração Gateway

```
? Which *type* of application would you like to create? Microservice gateway
? What is the base name of your application? acmegateway
? As you are running in a microservice architecture, on which port would like your server to run? It should be unique to avoid port conflicts. 80
? What is your default Java package name? com.invillia
? Which service discovery server do you want to use? No service discovery
? Which *type* of authentication would you like to use? JWT authentication (stateless, with a token)
? Which *type* of database would you like to use? SQL (H2, MySQL, MariaDB, PostgreSQL, Oracle, MSSQL)
? Which *production* database would you like to use? MySQL
? Which *development* database would you like to use? MySQL
? Do you want to use the Spring cache abstraction? Yes, with the Ehcache implementation (local cache, for a single node)
? Do you want to use Hibernate 2nd level cache? Yes
? Would you like to use Maven or Gradle for building the backend? Maven
? Which other technologies would you like to use?
? Which *Framework* would you like to use for the client? Angular 6
? Would you like to enable *SASS* stylesheet preprocessor? Yes
? Would you like to enable internationalization support? No
? Besides JUnit and Jest, which testing frameworks would you like to use?
```

JHipster

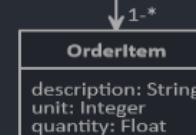
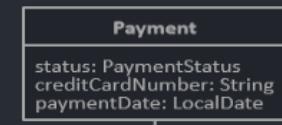
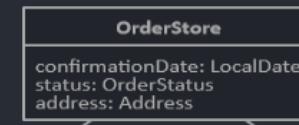
Modelagem JDL

```
// Acme entities 0.1

entity Store
{
    name String,
    address Address
}

entity Address
{
    streetAddress String,
    city String,
    state String,
    zipCode String
}

entity OrderStore
{
    confirmationDate LocalDate,
    status OrderStatus,
    address Address
}
```



JHipster

Conclusões

Após 2 horas e 30 minutos eu tinha o microservice e o gateway com Apache Kafka para execução assíncrona, Swagger, Liquibase, EhCache, Interface Visual, Spring Security, Spring Cloud com Eureka, Rotas todas funcionando e facilidades em implantação para AWS.

Confesso que ainda precisaria estudar um pouco mais a ferramenta pra extrair mais e, é claro, refatorar um pouco o código que ele gera que polui um pouco, mas ainda sim vale a pena dar uma checada.

Conclusão final



Aguardando o Bem Vindo!

Espero ter deixado claro meus pontos fortes e fracos tecnicamente com este material, foi de grande valia este estudo, com esta experiência acabei saindo maior do que entrei...

Qualquer coisa que precisem podem me contatar, pra mim programar é mais um hobby do que um trabalho pois isso me esforço para fazer meu melhor sempre.