**Zup projeto backend - Teste Desenvolvedor**

**========================**

Tecnologias utilizadas no desenvolvimento do projeto:

IDE: Eclipse Java EE IDE for Web Developers Neon 4.6.

Linguagem: Java 7.

Servidor de Aplicação: JBoss Wildfly 10.

Banco de dados: MySQL.

Front-End: Angular JS e HTML.

Frameworks de testes: JUnit, Arquillian, Rest-Assured.

Outros: CDI 1.1, EJB 3.3, JPA 2.1 and Bean Validation 1.1

Requisitos para execução da aplicação e dos testes

Maven 3 ou superior instalado e configurado, Java 7 instalado e MySQL instalado.

1. MySQ

Criar um usuário no MySQL com o nome de “zup” e senha “zup” como abaixo

CREATE USER 'zup'@'localhost' IDENTIFIED BY 'zup';

Criar um database com o nome de “zupprojectdb” como abaixo

CREATE DATABASE 'zupprojectdb';

Dar permissões de escrita para o usuário criado no database “zupprojectdb”, abaixo

GRANT ALL PRIVILEGES ON 'zupprojectdb' . \* TO 'zup'@'localhost';

2. Configurar JBoss WildFly

Se não estiver configurado fazer download da versão 10 em http://download.jboss.org/wildfly/10.0.0.Final/wildfly-10.0.0.Final.zip

3. Configurar datasource (MySQL)

Fazer download do conector JDBC do MySQL no link http://repo1.maven.org/maven2/mysql/mysql-connector-java/5.1.22/mysql-connector-java-5.1.22.jar

Acesse a pasta onde está o servidor “wildfly-10.0.0.Final\modules\system\layers\base\com e crie a pasta “mysql” e dentro dela crie a pasta “main”

Copie para pasta main o conector JDBC do MySQL.

Dentro da pasta main, crie o um arquivo chamado “module.xml”. Copie o conteúdo abaixo para esse arquivo.

<?xml version="1.0" encoding="UTF-8"?>

<module xmlns="urn:jboss:module:1.0" name="com.mysql

<resources>

<resource-root path="mysql-connector-java-5.1.22.jar"/>

</resources>

<dependencies>

<module name="javax.api"/>

<module name="javax.transaction.api"/>

</dependencies>

</module>

Acessar a pasta do servidor “wildfly-10.0.0.Final\standalone\configuration” e editar o arquivo “standalone.xml”. Encontrar a tag datasources: "urn:jboss:domain:datasource" e adicionar dentro do bolco do datasource o treicho abaixo.

<datasource jndi-name="java:jboss/datasources/zupProjectBackendDS" pool-name="zupProjectBackendDS" enabled="true" use-java-context="true" use-ccm="false">

<connection-url>jdbc:mysql://localhost:3306/zupprojectdb</connection-url>

<driver-class>com.mysql.jdbc.Driver</driver-class>

<driver>mysql</driver>

<security>

<user-name>zup</user-name>

<password>zup</password>

</security>

<validation>

<validate-on-match>false</validate-on-match>

<background-validation>false</background-validation>

</validation>

<statement>

<share-prepared-statements>false</share-prepared-statements>

</statement>

</datasource>

Abaixo dos datasources existe a tag "drivers" que têm as referências ao drivers JDBC, nela adicioinar o trecho abaixo.

<driver name="mysql" module="com.mysql">

<xa-datasource-class>com.mysql.jdbc.jdbc2.optional.MysqlXADataSource</xa-datasource-class>

</driver>

4. Inicializar o JBoss Wildfly

Linux: JBOSS\_HOME/bin/standalone.sh

Windows: JBOSS\_HOME\bin\standalone.bat

5. Aplicação

Pelo Github clone do repositório xy-inc em https://github.com/ruanqueiroz/xy-inc.git

Fazer pull do projeto zup-project-backend ou descompacte o arquivo do projeto enviado por email.

6. Deploy

Após efetuar todos os passos anteriores, servidor wildfly inicializado, datasource configurado, database e user criados.

Acessar a pasta onde o "zup-project-backend" foi baixado ou descompactado. Executar o comando de deploy:

"mvn clean compile wildfly:deploy"

Após o fim da execução, acessar: http://localhost:8080/zup-project-backend/"

Com a aplicação front-end é possível criar novos produtos, e ele mostra a lista de produtos cadastrados, além de ser possível acessar pesquisar um produto por id. A atualização e remoção só podem ser feitos através da Rest-API.

6.1 UnDeploy

Para fazer undeploy da aplicação do servidor, executar o seguinte comando na pasta raiz do projeto:

"mvn wildfly:undeploy"

7. Testes

Para desenvolver os testes da aplicação foram usados os frameworks: JUnit, Arquillian e Rest-Assured.

O Arquillian é um framework de testes que faz um deploy temporário de uma versão da aplicação e também permite a configuração de outra base de dados, como configuração de bando de dados em memória. Com isso podemos testar a camada de persistencia da aplicação.

O Rest-Assured é usado para testes de integração da aplicação, ele permite desenvolver os testes para camada REST. Foi implementado apenas um teste simples, que realiza getById(), ou seja, pesquisa um produto por id, para validar o teste é necessário fazer a validação do esquema do Json do modelo de Produto retornado.

Para executar os testes da aplicação, executar o comando na pasta raiz do projeto.

Comando: “mvn clean test -Parq-wildfly-remote”

8. Teste API Rest

GET http://localhost:8080/zup-project-backend/rest/products - Lista todos os produtos cadastrados

GET http://localhost:8080/zup-project-backend/rest/products/{id} - Busca um produto por id

POST http://localhost:8080/zup-project-backend/rest/products - Adiciona um novo produto

HEADER

Content-Type: application/json

BODY

{"name":"Televisao","description":"Televisao full HD","category":"Eletroeletronico","price":1500}

PUT http://localhost:8080/zup-project-backend/rest/products - Alterar produto

HEADER

Content-Type: application/json

BODY

{"id": 1,"name":"Computador","description":"Computador com bom desempenho","category":"Informatica", "price":1000}

Nesse caso o app mobile ou qualquer outro cliente deverá enviar o id no json do produto a ser atualizado.

DELETE http://localhost:8080/zup-project-backend/rest/products/{id} - Remover produto

HEADER

Content-Type: application/json

Caso o serviço execute com sucesso ele retorna 200. Caso ocorram erros, ele o status e a mensagem do erro.

9. Informação do projeto

O projeto é uma aplicação inicial, que pode crescer muito mais e se transformar em uma aplicação muito mais completa robustas, foi criado como projeto mavem e usando o servidor de aplicação o JBoss WildFly, com algumas outras tecnologias como: CDI 1.1, Maven, EJB 3.2, JPA 2.1, JAX-RS e AngularJS, que permitem mais agilidade no desenvolvimento, simplicidade na solução, escalabilidade, manutenibilidade e etc.

Foi desenvolvida a camada REST com os serviços e a camada de persistencia dos dados.

Foi criada a classe ProductProducer para disponibilizar os dados para a aplicação front-end, nesse caso a aplicação front-end acessa a camada de persistencia através do seu controller. A camada REST também acessa a persistencia que foi desenvolvida em EJB Stateless.

Os testes da aplicação implementados com Arquillian (que faz um deploy temporário) mostra que podemos automatizar bastante a camada REST, o que gera maior qualidade a aplicação, além de agilizar o desenvolvimento da aplicação.

10. Melhorias

Desenvolver os métodos de atualização e remoção de produtos na aplicação front-end.

Criar autenticação para acessar a API Rest, através de um token que seria enviado no header das requisições.

11. Fontes:

http://wildfly.org/

https://github.com/jayway/rest-assured

http://arquillian.org/