**Instituto Superior de Engenharia do Porto**

**Mestrado em Engenharia Informática**

# ODSOFT
## Organização e Desenvolvimento do Software

# Relatório Técnico de Projecto


*José Adriano Ferreira*
*1970390@isep.ipp.pt*

*Paulo Henrique Russo*
*1150285@isep.ipp.pt*


Docentes:
Alexandre Bragança
Nuno Betencourt

Porto
3 de Janeiro de 2016

## 1. Introdução

Este relatório técnico descreve o plano de um projecto de desenvolvimento de software, seguindo a metodologia de “continuous delivery”.

O projecto consiste na implementação de um pipeline de “continuous integration” que permita o desenvolvimento de quatro componentes, da aplicação designada por “Memorix”:

- Server application baseado no OFBiz
- Web application
- IOS application
- Android Application

A aplicação Memorix é integrante do OFBiz e é uma “mobile application” que permite que os utilizadores registem notas sob a forma de texto, áudio, fotos, vídeos, etc. e as possam partilhar através de redes sociais. Os dados das notas devem ser registados nos servidores da Memorix.com, mas também devem ser acessíveis via interface web.

Resumidamente o pipeline deverá contemplar:

- A compilação do OFBiz.
- Fase de testes unitários do OFBiz, incluindo a geração de um relatório de “Cobertura”.
- Fase de testes de aceitação semi-automatizada, que implica a realização de testes de UI (user interface) e aceitação manual do utilizador de testes.
- Fase de testes UAT (User Acceptance Tests), que a implica a aceitação do utilizador final. Estes devem ser realizados no ambiente do utilizador final (para o efeito será utilizada uma máquina virtual).

## 2. Project Plan

### Pipleline – Server Application no OFBiz

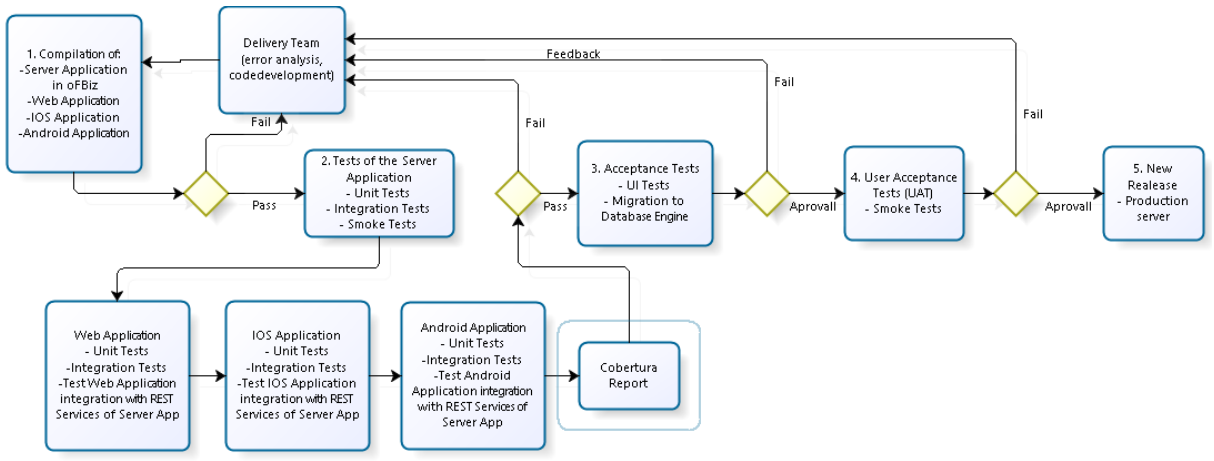



**Figura 1- Diagrama do pipelina da Server Application**

Repositório: **Bitbucket** - mei-isep/mei-rep-2015-g047

Tipo de Repositório: **GIT with Sourcetree**

Linguagem: **Java** (jdk1.8.0_60)

Build tool / compilation tool: **Ant**

Database Engine: **MySql**

New Component: **Memorix – no hot-deploy do Ofbz**

Será necessário o recurso a:

- Continuous Integration Server (ex. Jenkins)
- IDE (ex. Eclipse)
- Control version system (ex. GIT with Sourcetree)
- VirtualizationSoftware (ex. Virtualbox)
- Framework: OFBiz

Componentes do OFBiz:

- A aplicação será desenvolvida no componente Hot-deploy do OFBiz.
- Serviços REST que suportem o acesso das restantes compoenentes, web e mobile, ao backend no servidor: ***noteresource.java*** (para suportar os casos de uso criar e ver nota, que devem contemplar pelo menos os métodos GET, PUT e POST), ***pingresource.java.***
- Entidades: Notes

Testes de aceitação:

- Introdução de uma nova nota com o texto “teste de aceitação #teste” Durante a introdução o utilizador deverá validar que surge no ecrã a data e hora do sistema e o número de caractéres do texto introduzido que deverá estar correcto.
- Verificar se consegue encontrar a nota introduzida no passo anterior procurando-a por hash tag “#teste”. Depois verificar se o conteúdo, data e hora em que foi criada estão correctos.

As fases a seguir indicadas correspondem aos pontos 1, 2, 3 e 4 referidos na representação do pipeline da figura 1.

O início é despoletado pela Delivery Team com desenvolvimento, por exemplo de uma nova feature. Sempre que haja erros ou ocorra a não aceitação de testes de aprovação manual (tester ou end user), o feedback é dado à delivery team para que possa analisar a situação e decidir sobre a correcção do código.

Na **fase 1** é compilado o código da aplicação no OFBiz (main code e o código dos respectivos testes) e das restantes componentes (web, ios e android).

Na **fase 2** são realizados os testes unitários à aplicação, incluindo testes aos métodos de GET, PUT, POST e DELETE do serviço REST e testes de integração dos restantes componentes (web application, ios application e android application). Estes testes de integração devem servir para garantir que as restantes aplicações continuam a funcionar, apesar das eventuais alterações da estrutura da aplicação servidor

(entidades, serviços, etc.). Também são realizados Smoke Tests (testar o ambiente). Só se todos os testes forem bem sucedidos é que o package produzido pela compilação é copiado para a Virtual Machine de testes (VM1). Será realizado um relatório de cobertura.

As fases 1 e 2 são automáticas, ou seja, são despoletadas por um commit dos developers. Caso falhe, é dado o respectivo feedback aos developers.

Na **fase 3**, são realizados testes de interface por um utilizador de testes (tester). Deve usar a aplicação, testando entre outros, os casos de uso: “register a text note” e “view a note”. Esta fase deve prever a migração para um database engine a definir. O avanço para a fase seguinte requer aceitação do utilizador de testes. Caso aceite, o package é copiado para a Virtual Machine de um End-user (VM2).

A **fase 4** é realizada num ambiente de um utilizador final que fará os testes de aceitação (UAT). Inclui a realização de smoke tests e a verificação de casos de uso, tal como na fase anterior. O avanço para a fase seguinte requer aceitação deste utilizador. Caso aceite, é efectuado ao deploy para a máquina de produção.

**RoadMap**

- **Server application development – until 13/12/2015**
- **Web Application – until 20/12/2015**
- **IOS Application – until 27/12/2015**
- **Android Application – until 20/02/2016**

**Piplelines adicionais:**

- **Web Application**
- **IOS Application**
- **Android Application**

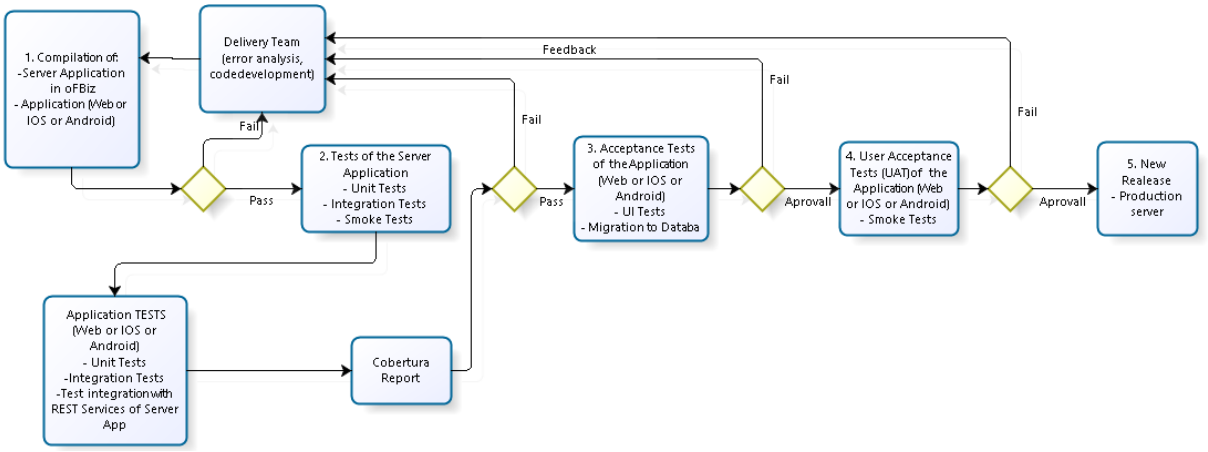



**Figura 2- Diagrama conceptual dos 3 pipelines, um para cada uma das aplicaçõeso web, ios e android.**

Para cada um dos 3 componentes adicionais está previsto um pipeline semelhante, em que a principal diferença reside na fase 1 e 2. Apenas apresentamos um diagrama que serve para representar cada um dos pipelines para cada aplicação (web, ios ou android).

As fases a seguir indicadas correspondem aos pontos 1, 2, 3 e 4 referidos na representação do pipeline da figura 2.

O início é despoletado pela Delivery Team com desenvolvimento, por exemplo de uma nova feature. Sempre que haja erros ou ocorra a não aceitação de testes de aprovação manual (tester ou end user), o feedback é dado à delivery team para que possa analisar a situação e decidir sobre a correcção do código.

Na **fase 1** é compilado o código da aplicação em causa e da server application.

Na **fase 2**, ao nível dos testes de integração, apenas são efectuados testes de integração entre a aplicação servidor e a aplicação em causa (web application por exemplo). Isto porque, se por exemplo não se estiver a conseguir fazer o deploy de

uma nova versão da web application, não significa que uma nova versão IOS não seja produzida e publicada. A única aplicação que tem que estar conforme e compatível com as restantes é a Server Application.

No caso da aplicação Android, uma vez que só se prevê o seu desenvolvimento mais tarde, será feito um desenvolvimento de um código básico (tipo “Hello world”), apenas para que que esteja previso no pipeline da aplicação servidor desde já.

**Diagrama de casosde uso:**

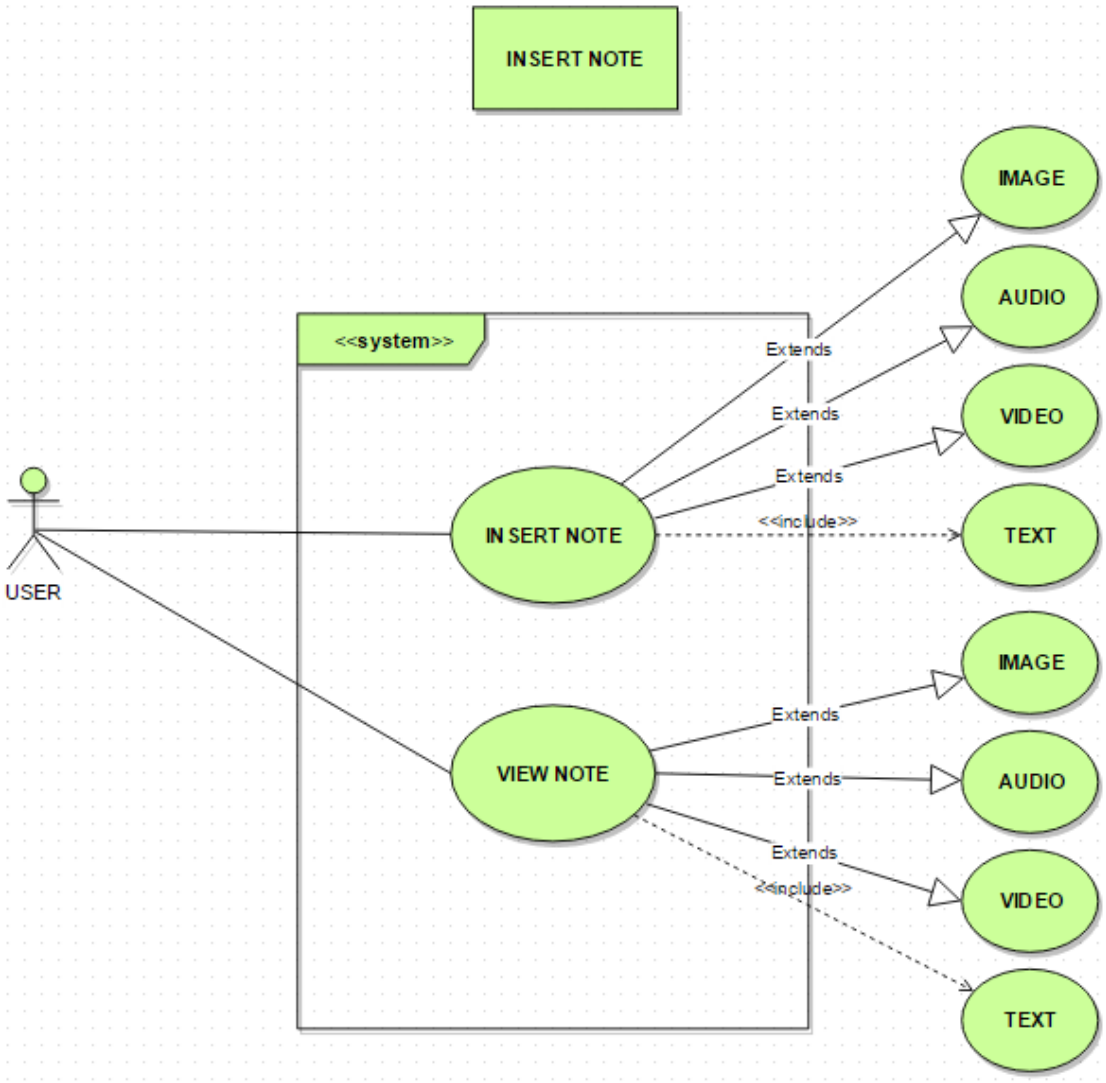



**Figura 3- Diagrama de casos de uso *"Insert Note"***

## 3. Desenvolvimento do Projecto

### 3.1 Componente MEMORIX

Foi criado um novo componente chamado Memorix no Hot-deploy do Ofbiz. Este no componente utiliza uma entidade que foi criada a e definida da seguinte forma:

- **memorix\entitydef\entitymodel.xml**

```
<entity entity-name="Note"
package-name="org.ofbiz.memorix"
title="Memorix Notes">
<field name="noteId" type="id-ne"></field>
<field name="loginName" type="id-long"></field>
<field name="note" type="id-long"></field>
<prim-key field="noteId"/>
</entity>
```

Nota: por razões de simplificação reduzimos o número de atributos da entidade, uma vez que o enunciado do trabalho referia mais atributos.

Foram criados os serviços **createnote** e ***updatenote*** que foram previamente registados em:

- **memorix\src\memorix\servicedef\services.xml**

```
<service name="createNote" engine="java"
location="memorix.Memorix" invoke="createNote">
<description>
A service to create new notes in the memorix component.
</description>
<attribute name="note" type="String" mode="IN" optional="false"/>
<attribute name="loginName" type="String" mode="IN" optional="false"/>
<attribute name="noteId" type="String" mode="OUT" optional="false"/>
</service>
```

```
<service name="updateNote" engine="java"
location="memorix.Memorix" invoke="updateNote">
<description>
A service to update notes in the memorix component.
</description>
<attribute name="note" type="String" mode="IN" optional="false"/>
```

```
<attribute name="loginName" type="String" mode="IN" optional="false"/>
<attribute name="noteId" type="String" mode="IN" optional="false"/>
<attribute name="noteId" type="String" mode="OUT" optional="false"/>
</service>
```

Foi criada a aplicação ***memorix.java*** que invoca os serviços **createnote** e ***updatenote***

- **memorix\src\memorix\Memorix.java**

```
public class Memorix{

@SuppressWarnings("null")

public static Map<String, Object> createNote(DispatchContext dctx,
Map<String, Object>context) {

GenericDelegator delegator = (GenericDelegator)
DelegatorFactory.getDelegator("default");
String noteId = delegator.getNextSeqId("Note");
String loginName = (String)context.get("loginName");
String notes = (String)context.get("note");

GenericValue note = delegator.makeValue("Note");
note.set("noteId", noteId);
note.set("loginName", loginName);
note.set("note", notes);

GenericValue myNewNote = null;
try {
   myNewNote = delegator.create(note);
} catch (GenericEntityException e) {
   e.printStackTrace();
}
Map<String, Object> result = null;
if(myNewNote != null)
{
    result = ServiceUtil.returnSuccess () ;
   result.put("noteId", noteId);
   return result; }
else
{
    return ServiceUtil.returnFailure();
 }
}

public static Map<String, Object> updateNote(DispatchContext dctx,
Map<String, Object>context) {

   // Lets update a note
   GenericDelegator delegator = (GenericDelegator)
DelegatorFactory.getDelegator("default");
```

```
    String noteId = (String)context.get("noteId");
    String loginName = (String)context.get("loginName");
    String notes = (String)context.get("note");

    GenericValue note = delegator.makeValue("Note");
    note.set("noteId", noteId);
    note.set("loginName", loginName);
    note.set("note", notes);

    try {
          delegator.createOrStore(note);
    } catch (GenericEntityException e) {
          // TODO Auto-generated catch block
          e.printStackTrace();
    }
    Map<String, Object> result = null;
        result = ServiceUtil.returnSuccess () ;
          result.put("noteId", noteId);
          return result;
}
}
```

## 3.2 Serviços REST

No componente **RESTCOMPONENT** do Ofbiz foi criada uma nova classe **NoteResource.java** que contém os serviços GET (all), PUT e POST que interagem com a entity **Note**.

- **Rescomponent\src\restcomponent\Noteresource.java**

### Método GET (all) – devolve todas as notas existentes

```
@GET
@Produces("application/json")
public Response getAllNotes() {
	String username = null;
	String password = null;

	try {
		username = httpRequest.getHeader("login.username");
		password = httpRequest.getHeader("login.password");
	} catch (NullPointerException e) {
		return Response.serverError().entity("Problem ")
				.build();	}

	if (username == null || password == null) {
		return Response.serverError().entity("")
				.build();	}

	GenericDelegator delegator = (GenericDelegator)
	DelegatorFactory.getDelegator("default");
	List<GenericValue> notes = null;

	try {
		notes = delegator.findAll("Note", false);
	} catch (GenericEntityException e) {
		return Response.serverError().entity(e.toString()).build();}

	if (notes != null) {

		String response = Util.convertListGenericValueToJSON(notes);

		if (response == null) {
			return Response.serverError().entity("Erro na conversao
			do JSON!").build(); }

		return Response.ok(response).type("application/json").build();
	}
	throw new RuntimeException("Invalid ");
}
```

## Método POST – cria uma nova nota

```
@POST
@Produces("application/json")
public Response createNote() {
	String username = null;
	String password = null;

	try {
		username = httpRequest.getHeader("login.username");
		password = httpRequest.getHeader("login.password");
	} catch (NullPointerException e) {
		return Response.serverError().entity("Problem")
				.build();}

	if (username == null || password == null) {
		return Response.serverError().entity("Problem ")
				.build();}

	JsonReader jsonReader;
	try {
		jsonReader = Json.createReader(httpRequest.getReader());
	} catch (IOException e) {
		return Response.serverError().entity("Problem ").build();
	}

	JsonObject jsonObj = jsonReader.readObject();

	// Lets now invoke the ofbiz service that creates a note
	GenericDelegator delegator = (GenericDelegator)
	DelegatorFactory.getDelegator("default");
	LocalDispatcher dispatcher =
	org.ofbiz.service.ServiceDispatcher.getLocalDispatcher
	("default", delegator);

	Map<String, String> paramMap = UtilMisc.toMap(
			"loginName", jsonObj.getString("loginName"),
			"note", jsonObj.getString("note"));

	Map<String, Object> result = FastMap.newInstance();
	try {
		result = dispatcher.runSync("createNote", paramMap);
	} catch (GenericServiceException e1) {
		Debug.logError(e1, PingResource.class.getName());
		return Response.serverError().entity(e1.toString()).build();
	}

	String noteId = result.get("noteId").toString();
	String note = Util.getNote(noteId);
	if (note != null) {
		return Response.ok(note).type("application/json").build();
	} else {
		return Response.serverError().entity("Problem reading the new
		note after created!").build();
	}
}
```

## Método PUT – altera o conteúdo de uma dada nota

```
@PUT
@Produces("application/json")
@Path("{id}")
public Response updateNoteById(@PathParam("id") String noteId) {
	String username = null;
	String password = null;

	try {
		username = httpRequest.getHeader("login.username");
		password = httpRequest.getHeader("login.password");
	} catch (NullPointerException e) {
		return Response.serverError().entity("")
				.build();}

	if (username == null || password == null) {
		return Response.serverError().entity("Problem ")
				.build();}

	JsonReader jsonReader;
	try {
		jsonReader = Json.createReader(httpRequest.getReader());
	} catch (IOException e) {
		return Response.serverError().entity("Problem ").build();}

	JsonObject jsonObj = jsonReader.readObject();

	// Lets now invoke the ofbiz service that updates a note
	GenericDelegator delegator = (GenericDelegator)
	DelegatorFactory.getDelegator("default");
	LocalDispatcher dispatcher = org.ofbiz.service.ServiceDispatcher.
	getLocalDispatcher("default", delegator);

	Map<String, String> paramMap = UtilMisc.toMap(
			"noteId", noteId,
			"loginName", jsonObj.getString("loginName"),
			"note", jsonObj.getString("note"));

	Map<String, Object> result = FastMap.newInstance();
	try {
		result = dispatcher.runSync("updateNote", paramMap);
	} catch (GenericServiceException e1) {
		Debug.logError(e1, PingResource.class.getName());
		return Response.serverError().entity(e1.toString()).build();}

	if (result.get("responseMessage").toString().compareTo("success") == 0) {
		String note = Util.getNote(noteId);

		if (note != null) {
			return Response.ok(note).type("application/json").build();
		} else {
			return Response.serverError().entity("Problem ").build();
		}
	} else {
		return response.serverError().
		entity(result.get("responseMessage").toString()).build();
	}
}
```

Estes métodos recorrem também ao uso de do serviço **getnote()** existente na classe **Util.java** existente no **RESTCOMPONENT** do Ofbiz.

- **Rescomponent\src\restcomponent\Util.java**

```
    synchronized static String getNote(String noteId) {
        GenericValue note = null;
        JsonObject object=null;

        GenericDelegator delegator = (GenericDelegator)
        DelegatorFactory.getDelegator("default");

        try {
               note = delegator.findOne("Note",
                                    UtilMisc.toMap("noteId", noteId), false);
               JSON json=null;
               json = new GenericValueToJSON().convert(note);
               JsonReader jsonReader = Json.createReader(new
               StringReader(json.toString()));
               object = jsonReader.readObject();
               jsonReader.close();
        }
        catch (GenericEntityException e) {
                      return null;
        }
        catch (ConversionException e) {
                             return null;}

        return object.toString();
        }
```

A classe Noteresource.java foi previamente registada no OFBizRESTApplication.

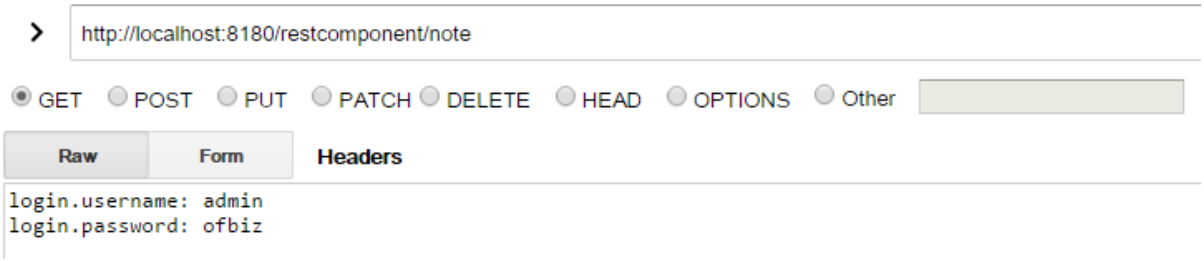
- **Rescomponent\src\restcomponent\ OFBizRESTApplication.java**

```
public class OFBizRESTApplication extends Application {
  @Override
  public Set<Class<?>> getClasses() {
      Set<Class<?>> classes = new HashSet<Class<?>>();
      classes.add(PingResource.class);
      classes.add(ProductResource.class);
      classes.add(RecipeResource.class);
      classes.add(NoteResource.class);
      return classes;
  }
}
```

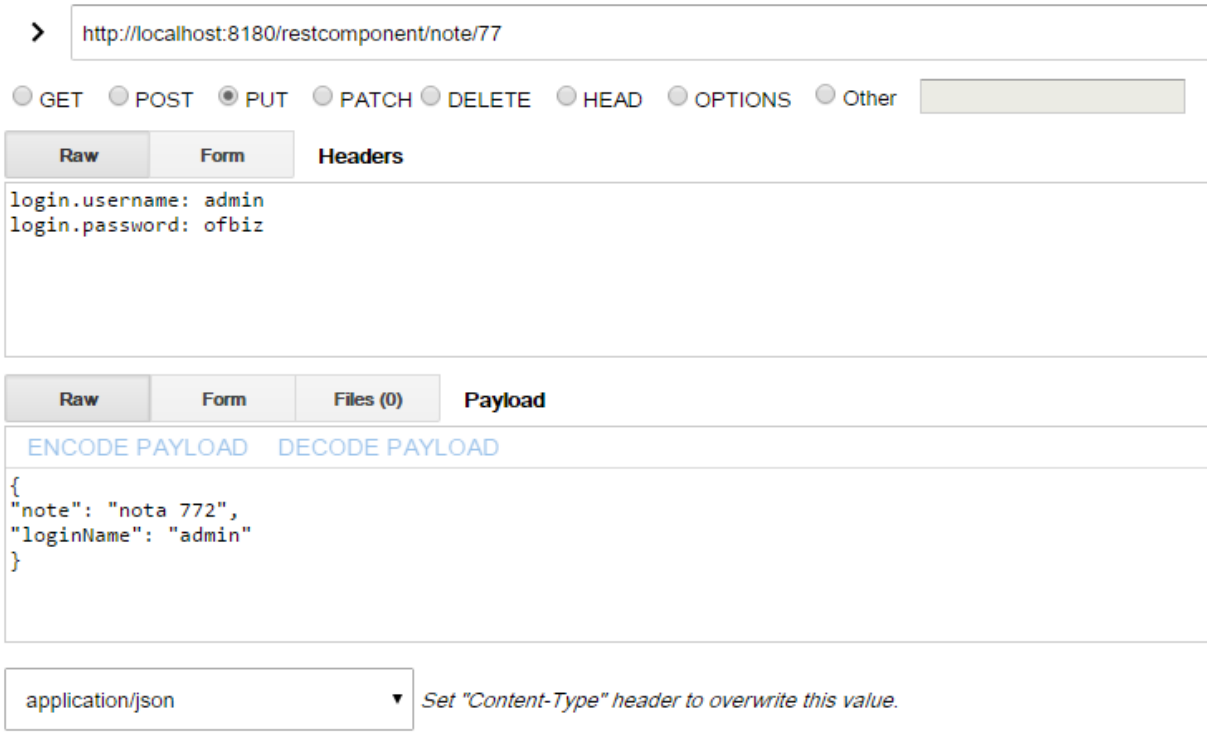
## Verificação dos Serviços Rest criados

Através da Advanced Rest Client Application (extensão do Google Chrome) foi testado o funcionamento dos serviçsoo rest. Os parâmetros utilizados foram os seguintes:

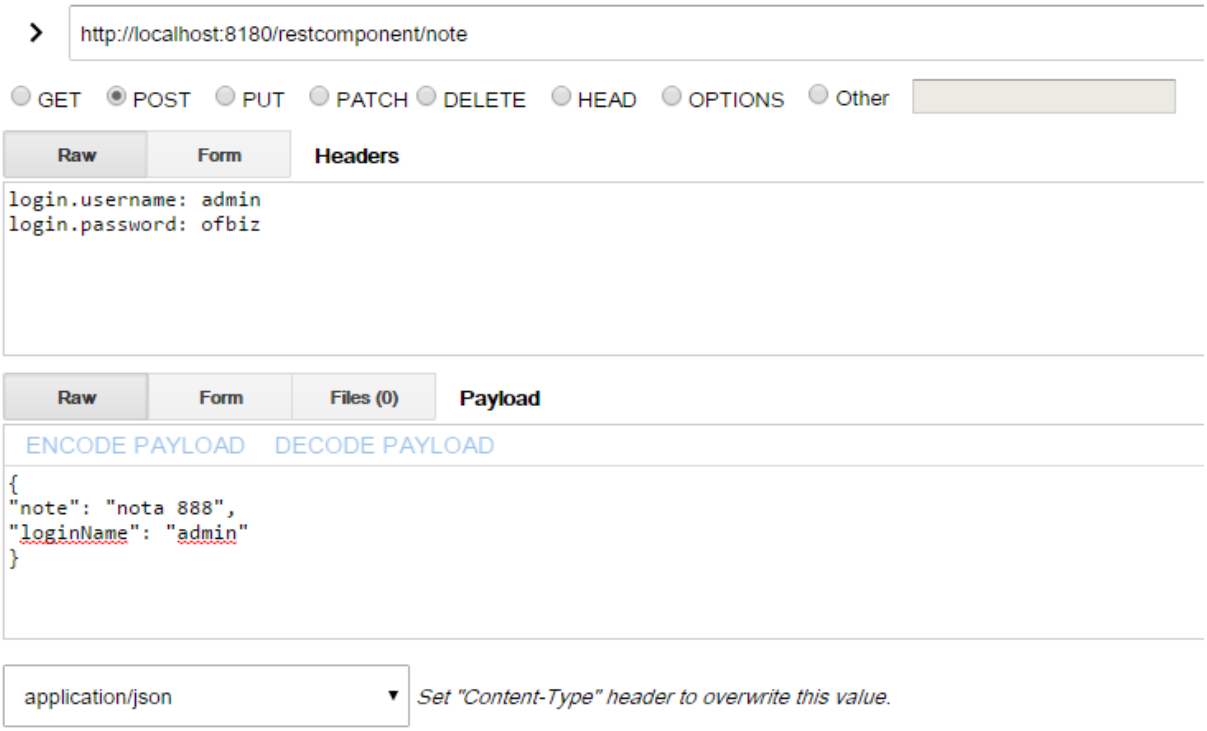
## Método GET




## Método PUT



## Método POST


http://localhost:8180/restcomponent/note
GET POST PUT PATCH DELETE HEAD OPTIONS Other
Raw Form Headers
login.username: admin
login.password: ofbiz
Raw Form Files (0) Payload
ENCODE PAYLOAD DECODE PAYLOAD
{
"note": "nota 888",
"loginName": "admin"
}
application/json
Set "Content-Type" header to overwrite this value.

## 3.4 BUILD.XML do memorix

O build.xml do memorix foi criado automáticamente aquando da criação do novo componente. Foram efectuadas alterações para permitir a inclusão no build das classes de testes e para configurar o relatório de cobertura.

**Linhas adicionadas para garantir a inclusão no build das classes de testes:**

```
<target name="jar" depends="classes">
	<main-jar/>
	<test-jar/>
	</target>
```

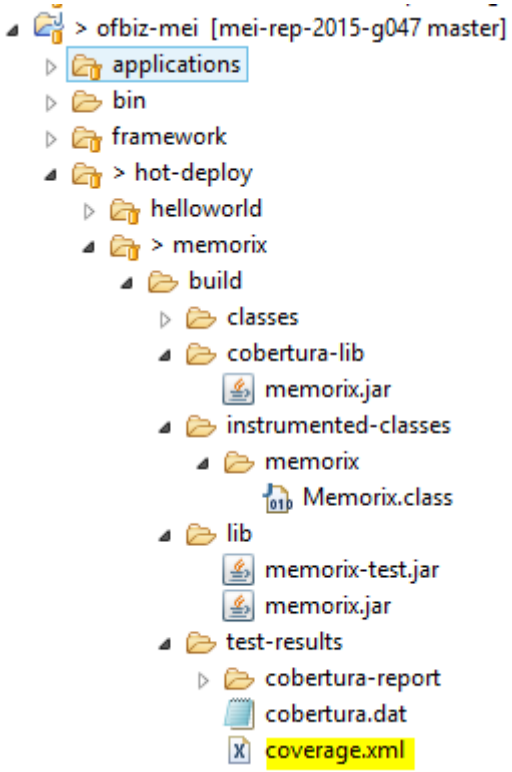
**Linhas adicionadas para configurar o relatório de cobertura:**

```
<property name="cobertura.dir" value="${ofbiz.home.dir}/tools/cobertura" />
<property name="instrumented.dir" value="${build.dir}/instrumented-classes"/>
<property name="report.xml.dir" value="${build.dir}/test-results" />
<property name="coverage.xml.dir" value="${build.dir}/test-results" />

<path id="cobertura.classpath">
	<fileset dir="${cobertura.dir}">
		<include name="**/*.jar" />
	</fileset>
</path>

<taskdef classpathref="cobertura.classpath" resource="tasks.properties" />

<target name="instrument" depends="classes" >
	<delete file="cobertura.ser" />
	<delete dir="${instrumented.dir}" />
	<cobertura-instrument todir="${instrumented.dir}">
		<fileset dir="${build.dir}">
			<include name="**/*.class" />
			<exclude name="**/*Test*.class" />
		</fileset>
	</cobertura-instrument>
</target>

<target name="test-coverage" depends="instrument">
	<junit fork="yes" dir="${basedir}" haltonfailure="no" >
		<classpath location="${instrumented.dir}" />
		<classpath refid="cobertura.classpath" />
		<classpath location="${test.classes}" />
		<classpath>
			<fileset dir="${basedir}/lib">
				<include name="*.jar"/>
			</fileset>
		</classpath>
		<formatter type="xml" />
		<test name="${testcase}" todir="${reports.xml.dir}" if="testcase"/>
		<batchtest todir="${report.xml.dir}" unless="testcase">
			<fileset dir="${build.dir}">
				<include name="**/*Test*.java" />
			</fileset>
		</batchtest>
	</junit>
</target>
```

```
<target name="coverage-report" depends="test-coverage">
       <cobertura-report   srcdir="${source}"
              destdir="${coverage.xml.dir}"
              format="xml" />
</target>
```

Para permitir o relatório de cobertura, foi efectuado download dos jar's correspondentes, que foram colocados na pasta tools do Ofbiz.

A figura seguinte apresenta a estutura criada associada à produção do relatório de cobertura (coverage.xml), quando se faz o build dos targets em causa.



## 3.5 TESTES do componenete memorix

Foram criados três conjuntos de testes unitários que foram previamente registados em MemorixTests.xml.

***Nota: A nossa preocupação incidiu na correcta configuração dos testes e na sua posterior compilação, e não nos testes em si. Ou seja não aperfeiçoamos o código de testes de forma a efectivamente testar os serviços em causa.***

- **Definição dos testes a realizar: Memorix\testdef\MemorixTests.xml**

```
<test-suite suite-name="MemorixTests" ……..

<test-case case-name="loadNoteTestData">
<entity-xml action="load"
entity-xml-url="component://memorix/testdef/data/MemorixTestData.xml"/>
</test-case>

<test-case case-name="note-tests">
<simple-method-test
location="component://memorix/script/MemorixTest.xml"/>
</test-case>

<test-case case-name="updateNoteTest">
        <junit-test-suite class-name="UpdateNoteTest"/>
</test-case>

</test-suite>
```

- **Teste loadNoteTestData - Memorix\testdef\data\MemorixTestsData.xml**

  Carrega os dados de teste registados no xml abaixo e verifica e o carregamento foi bem sucedido.

```
<entity-engine-xml>
<Note noteId="99" loginName="admin" note="memorixTextData note 99"  />
</entity-engine-xml>
```

- **Testes note-tests  - Memorix\script\MemorixTest.xml**

  Contem dois métodos de teste:

  - **testCreateNote** – deve detectar  se a criação foi bem sucedida

```
<simple-method method-name="testCreateNote" short-description="Test case for
successfully creating a Note record." login-required="false">
<entity-one entity-name="UserLogin" value-field="serviceCtx.userLogin">
      <field-map field-name="userLoginId" value="admin"/>
</entity-one>
<set field="serviceCtx.noteId" value="999"/>
<set field="serviceCtx.note" value="nota teste 999"/>
<set field="serviceCtx.loginName" value="admin"/>

<!-- Execute the service -->
<call-service service-name="createNote" in-map-name="serviceCtx">
      <results-to-map map-name="serviceResult"/>
</call-service>

<!-- Confirm the service output parameters -->
<assert>
      <if-compare-field field="serviceResult.noteId" operator="equals" to-
      field="serviceCtx.noteId"/>
</assert>

<!-- Confirm the database changes-->
<entity-one value-field="note" entity-name="Note">
      <field-map field-name="noteId" from-field="serviceCtx.noteId"/>
      <field-map field-name="note" from-field="serviceCtx.note"/>
      <field-map field-name="loginName" from-field="serviceCtx.loginName"/>
</entity-one>
<assert><not><if-empty field="note"/></not></assert>
<check-errors/>
</simple-method>
```

- **testCreateNoteFail** – deve detectar se o programa impede a criação de um noteid já existente

```
<simple-method method-name="testCreateNoteFail"
short-description="Test case for unsuccessfully creating a Note record by
attempting to use a noteId  that has already been used." login-
required="false">

<!-- Use to confirm nothing has changed at the end of the test -->
<set field="startTime" value="${date:nowTimestamp()}" type="Timestamp"/>

<entity-one entity-name="UserLogin" value-field="serviceCtx.userLogin">
<field-map field-name="userLoginId" value="admin"/>
</entity-one>

<set field="serviceCtx.noteId" value="999"/>
<set field="serviceCtx.note" value="note 999"/>
<set field="serviceCtx.loginName" value="admin"/>

<!-- Execute the service, note break-on-error is false so that the test
itself doesn't fail and we also need a separate transaction so our lookup
below doesn't fail due to the rollback -->
<call-service service-name="createNote" in-map-name="serviceCtx"
       break-on-error="false" require-new-transaction="true">
       <results-to-map map-name="serviceResult"/>
</call-service>
<!-- Clear these because break-on-error="false" doesn't seem to work as it
should at the moment -->
<clear-field field="responseMessage"/>
<clear-field field="errorMessageList"/>

<!-- Confirm the service output parameters, in this case the presence of an
error response -->
<assert><if-compare field="serviceResult.responseMessage" operator="equals"
value="error"/></assert>

<!-- Confirm the database changes, nothing should have changed -->
<entity-condition list="note" entity-name="Note">
<condition-list>
       <condition-expr field-name="lastUpdatedStamp" operator="greater-
       equals" from-field="startTime"/>
       <condition-expr field-name="noteId" from-field="serviceCtx.noteId"/>
       <condition-expr field-name="note" from-field="serviceCtx.note"/>
       <condition-expr field-name="loginName" from-
       field="serviceCtx.loginName"/>
       </condition-list>
</entity-condition>
<!--  Should be empty -->
<assert><if-empty field="note"/></assert>
<check-errors/>
</simple-method>
```

- **Teste UpdateNoteTest - Memorix\src\test\UpdateNoteTest.java**

  Contem a classe UpdateNoteTest que deve validar se um update foi bem realizado.

```
public class UpdateNoteTest{

    Map<String, Object> ctx = UtilMisc.<String, Object>toMap("1",
"admin", "test notes xyz");

        @Test
        public void testupdate() {
              Memorix.updateNote(null, ctx);
              assertEquals(1, 1, 0);
        }
```

## Resultado dos Testes

O resultado dos testes realizados pode ser consultado no ficheiro:

- **Runtime\logs\test-results\MemorixTests.xml**

```
<testsuite errors="0" failures="0" hostname="GAPC13068" name="MemorixTests"
skipped="0" tests="2" time="0.788" timestamp="2016-01-01T11:22:10">
  <properties />
  <testcase classname="org.ofbiz.testtools.EntityXmlAssertTest"
name="loadNoteTestData" time="0.0" />
  <testcase classname="org.ofbiz.testtools.SimpleMethodTest" name="note-
tests.testCreateNote" time="0.601" />
</testsuite>
```

**Nota: A compilação da suite de testes pode ser invocada através do seguinte comando, na linha de comandos:**

**ant run-test-suite -Dtest.component=memorix -Dtest.suiteName=MemorixTests**

## 3.6 Migração para Mysql / data migration scripts

Foi seguido o procedimento para mudar a base de dados ( e respectivos dados) do Ofbiz de Derby para MySQL, que se pode encontrar no seguinte link:

https://cwiki.apache.org/confluence/display/OFBIZ/How+to+migrate+OfBiz+from+Derby+to+MySQL+database

Resumidamente o procedimento passa por

4. Instalar o Mysql (mysql-5.5.23-winx64)
5. Instalar o driver MySQL JDBC driver – colocar o mysql-connector-java-5.1.14-bin.jar na pasta <ofbiz-dir>/framework/entity/lib/jdbc
6. Criar as bases de dados ofbiz, ofbizolap e ofbiztenant no mysql
7. Criar os users ofbiz, ofbizolap e ofbiztenant, com as permissões devidas (grant all)
8. No Ofbiz, exportar os dados da base de dados para ficheiros xml
9. Editar o engine.xml (<ofbiz-dir>/framework/entity/config/entityengine.xml) configurando os data sources: 'localmysql', 'localmysqlolap' e 'localmysqltenant'.
10. Substituir derby por mysql nos delegators: default, default-no-eca e test.
11. Garantir que no ficheiro DatabaseUtil.java localizado em ofbiz/framework/entity/src/org/ofbiz/entity/jdbc/ contém a instrução sqlBuf.append(" ENGINE ") e não sqlBuf.append(" TYPE ").
12. Instalar a nova configuração: ofbiz-dir>java -jar ofbiz.jar -install
13. No Ofbiz importar os dados exportados no passo 8.

**Nota: ao realizar este procedimento, ao realizar o passo 12 surgiu um erro que não conseguimos ultrapassar. Como consequência o Ofbiz deixu de arrancar. Por isso tivemos que avançar com o projecto excluido a migração para mysql.**

**Notas sobre o "data migration process"**

A implementação deste requisito passaria para a criação de scripts para alterar a estrutura da entity notes.

Por exemplo a criação de um novo campo DateTimeNoteCreation seria suportada por um script com a instrução:

```
ALTER TABLE note ADD COLUMN DateTimeNoteCreation;
```

Para retornar ao ponto anterior (rollback) deveria ser criado um script para a eliminação desse com a instrução:

```
ALTER TABLE note DROP COLUMN DateTimeNoteCreation;
```

Estas tarefas deveriam depois ser adequadamente configuradas no pipeline no Jenkins.

## 3.7 JENKINS – Build, tests e cobertura report

Foi criado um projecto “OFBIZ-MEI” no Jenkings de forma a fazer build automatizado do ofbiz e do novo componente Memorix, dos testes e do relatório de cobertura.

Previamente, foram instalados os seguintes plugins do Jenkins:

- ***GIT Plugin*** – integra o GIT com o Jenkins

- ***Ant Plugin*** - This plugin adds Apache Ant support to Jenkins.

- ***Build Pipeline Plugin*** - This plugin renders upstream and downstream connected jobs that typically form a build pipeline. In addition, it offers the ability to define manual triggers for jobs that require intervention prior to execution, e.g. an approval process outside of Jenkins.

- ***Cobertura Plugin*** - This plugin integrates Cobertura coverage reports to Jenkins.

- ***JUnit Plugin*** - Allows JUnit-format test results to be published

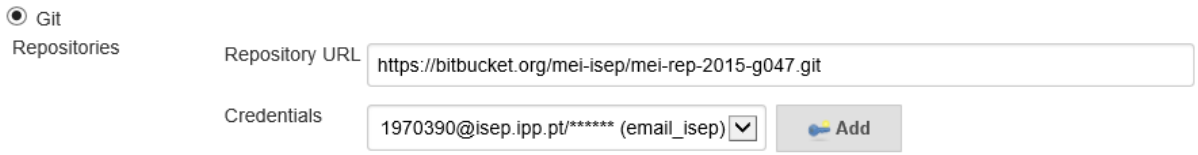
- ***Copy Artifact Plugin*** - Adds a build step to copy artifacts from another project

- ***Promoted builds plugin*** –This plugin implements a "promoted build" feature where a build of one job can be marked as "promoted" when it passes certain criteria.

- ***Publish Over SSH*** - Send build artifacts over SSH

Antes de avançar com a configuração do pipeline no Jenkins, apresentamos a configuração básica que permite fazer o build o ofbiz e do novo componente Memorix, a realização de testes e do relatório de cobertura.

- Project Name: **Ofbiz-Mei**
- Source Code Management

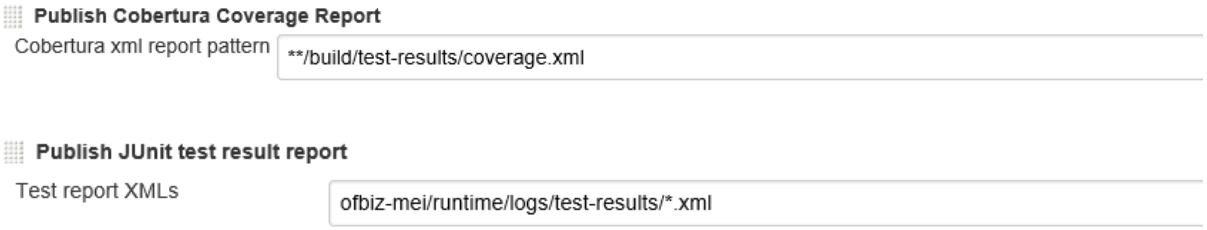



- Build do Ofbiz

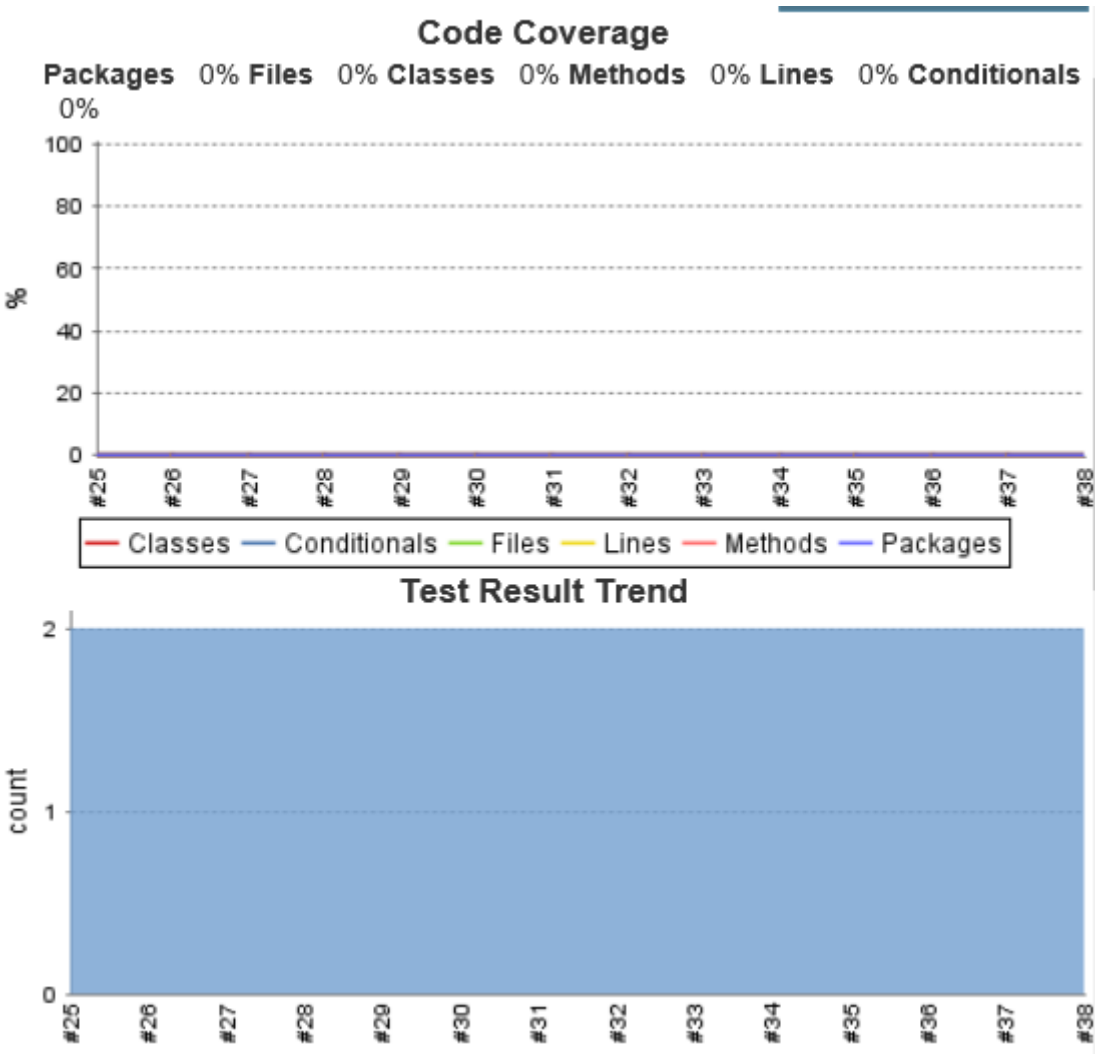

- Build da suite de testes


- Build do target que coverage.report

- Post-Build actions




**Fazendo build do projecto já é possível consultar noo Jenkins o resultado dos testes e relatório de cobertura.**



## 3.8 JENKINS Pipeline

De seguida serão apresentadas as configurações efectudas no projecto Ofbiz-Mei, no Jenkins, de forma a conseguir automatizar a fase de build do ofbiz, de testes automáticos, produção do relatório de cobertura. Se estas fases forem bem sucedidas, então o utilizador será chamado a fazer testes de aceitação (UAT). Se aceitar, o processo de deploy será concluído com a cópia para outra pasta (não foi usada uma Virtual Machine) de todos os ficheiros do ofbiz.

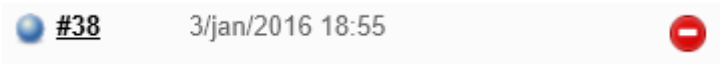
A configuração da fase de build do ofbiz, de testes automáticos, produção do relatório de cobertura foi descrita no passo anterior. No Jenkins foi activada a opção “Promote builds when” e foram configuardos dois processos (por ordem de execução):

***Promote-to-uat***

### *Promote-to-deploy*

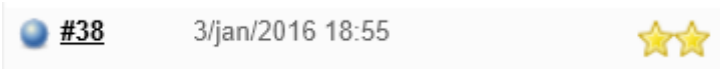
Este processo solicita a aprovação do processo anterior, mediante a qual irá executar um batch file que copia o ofbiz aqui compilado para uma nova pasta.


Fazendo build do projecto no Jenkins, o mesmo ficará pendente da aceitação do utilizador (UAT). Se não der erros, o projecto ficará neste estado:

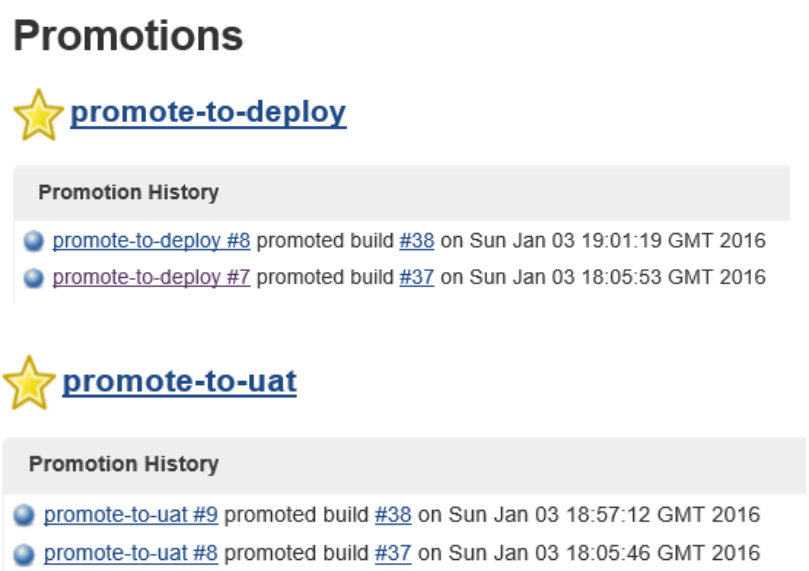


Entrando no processo pendente “promote-to-UAT” é possível aprovar para o processo seguinte que é o “promote-to-deploy”.


No final o projecto ficará neste estado:




Consultando o promotion status:



## ***Nota final***

Este trabalho prático, bem como todos os trabalhos realizados nas aulas, foram realizados em conjunto pelos dois elementos deste grupo. Contudo, devido a problemas no computador de um dos elementos, acabamos por ir fazendo todos os comits a partir do pc de apenas um dos elementos do grupo.

**FIM DO DOCUMENTO**