Departamento de Engenharia Informática

Mestrado em Engenharia Informática – Computação Móvel

Mobilidade em Redes de Comunicação

Ano letivo 2012/2013

ONYX

Relatório de projeto

Documento elaborado por:
Cláudio Esperança – aluno n.º 2120917
Diogo Serra – aluno n.º 2120915

Docentes:
Sílvio Mendes

Versão 1.0.52

18 de junho de 2013

Revisões

Versão	Autor(es)	Descrição das alterações
1.0	Cláudio Esperança Diogo Serra	• Versão inicial do documento

Índice de conteúdos

Índice de imagens

Lista de acrónimos

Para referência deixamos uma lista de acrónimos utilizados ao longo do documento:

Acrónimo	Designação
API	*Application Programming Interface*
BPM	*Business Process Management*
BPMN	*Business Process Model and Notation*
CET	Curso de Especialização Tecnológica
CRUD	*Create, Read, Update and Delete*
DBAL	*Database Abstraction Layer*
DEI	Departamento de Engenharia Informática
DER	Diagrama Entidade Relacionamento
ESTG	Escola Superior de Tecnologia e Gestão de Leiria
GPL	*GNU General Public License*
HTML	*Hyper Text Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IPL	Instituto Politécnico de Leiria
JS	*JavaScript*
JSON	*JavaScript Object Notation*
LAMP	*Linux, Apache HTTP server, MySQL and PHP*
MEI	Mestrado em Engenharia Informática
MEI-CM	Mestrado em Engenharia Informática – Computação Móvel
MVC	*Model-view-controller*
ORM	*Object-relational mapping*

Acrónimo	Designação
PDO	*PHP Data Objects*
PHP	*PHP: Hypertext Preprocessor*
REST	*Representational State Transfer*
RPC	*Remote Procedure Call*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SSL	*Secure Sockets Layer*
TLS	*Transport Layer Security*
UC	Unidade Curricular
UI	*User Interface*
WSDL	*Web Service Definition Language*
XML	*eXtensible Markup Language*
XSL	*eXtensible Stylesheet Language*
XSLT	*XSL Transformations*

1. Introdução

Este documento apresenta o trabalho desenvolvido no âmbito do projeto da unidade curricular de Programação Avançada de Sistemas e Serviços do Mestrado em Engenharia Informática – Computação Móvel, da Escola Superior de Tecnologia e Gestão no ano letivo de 2012/2013.

O projeto, com a denominação ONYX, proponha o desenvolvimento de um centro de excelência BPM que demonstrasse uma arquitetura de referência SOA com base em metodologias ágeis. Pretendia-se assim a análise dos processos e modelos fornecidos no sentido de propor soluções tecnológicas para resolução dos problemas apresentados. Funcionalmente deveria ser implementado um sistema que permitisse a concretização do processo de contratação de um docente CET, através da atualização do modelo de dados fornecidos, num sistema de informação a desenvolver.

Neste documento será apresentada a a solução proposta bem como as diferentes fases no desenvolvimento do projeto. Começaremos por apresentar no capítulo 2 as informações e modelos fornecidos sobre o problema. No capítulo 3 será apresentada a solução proposta no que diz respeito à sua arquitetura e soluções tecnológicas adotadas, sendo descrito o processo de desenvolvimento no capítulo 4. Sendo as habituais conclusões apresentadas no capítulo 5.

2. Modelos fornecidos

No sentido de contextualizar o projeto, foram fornecidas algumas informações sobre o problema e o tipo de solução pretendida. Na imagem 1 é apresentado o diagrama BPMN que descreve as etapas para a contratação de um docente para um curso de especialização tecnológica.

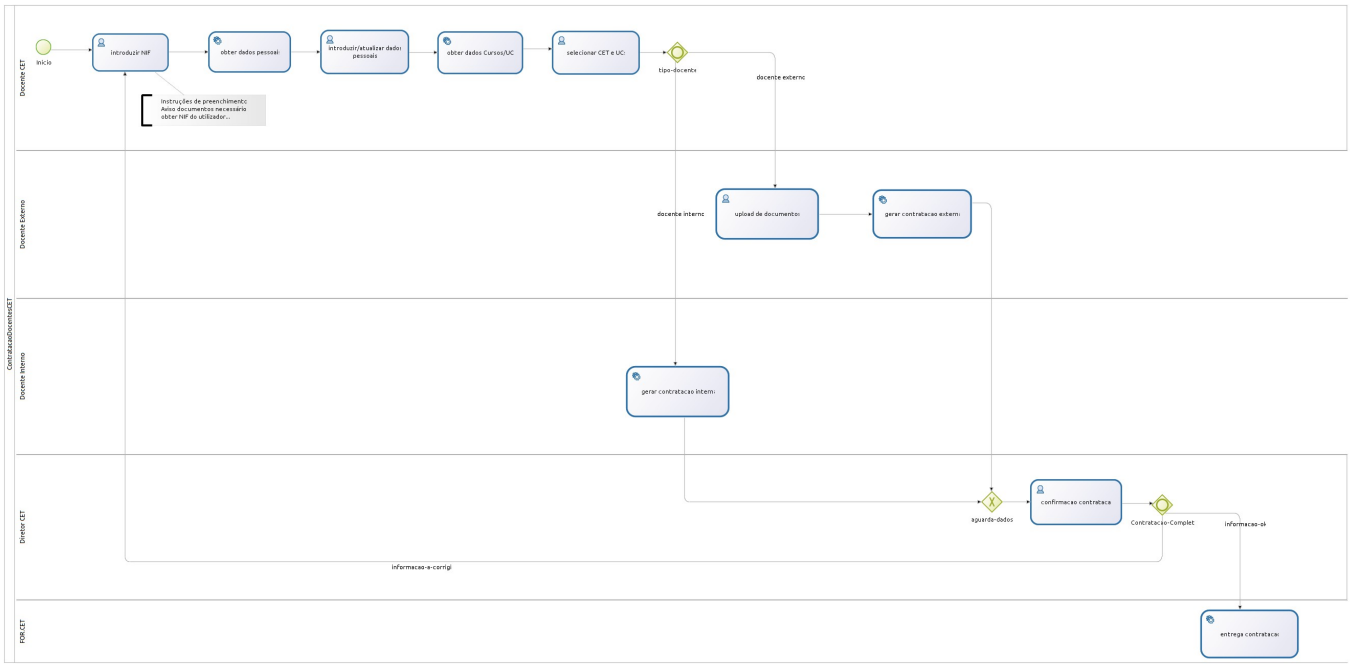

Imagem 1: Diagrama BPMN para a contratação de docentes para um CET

Com base nas especificações, este processo de contratação deveria utilizar o modelo de dados que foi também fornecido e que se encontra representado na imagem 2.

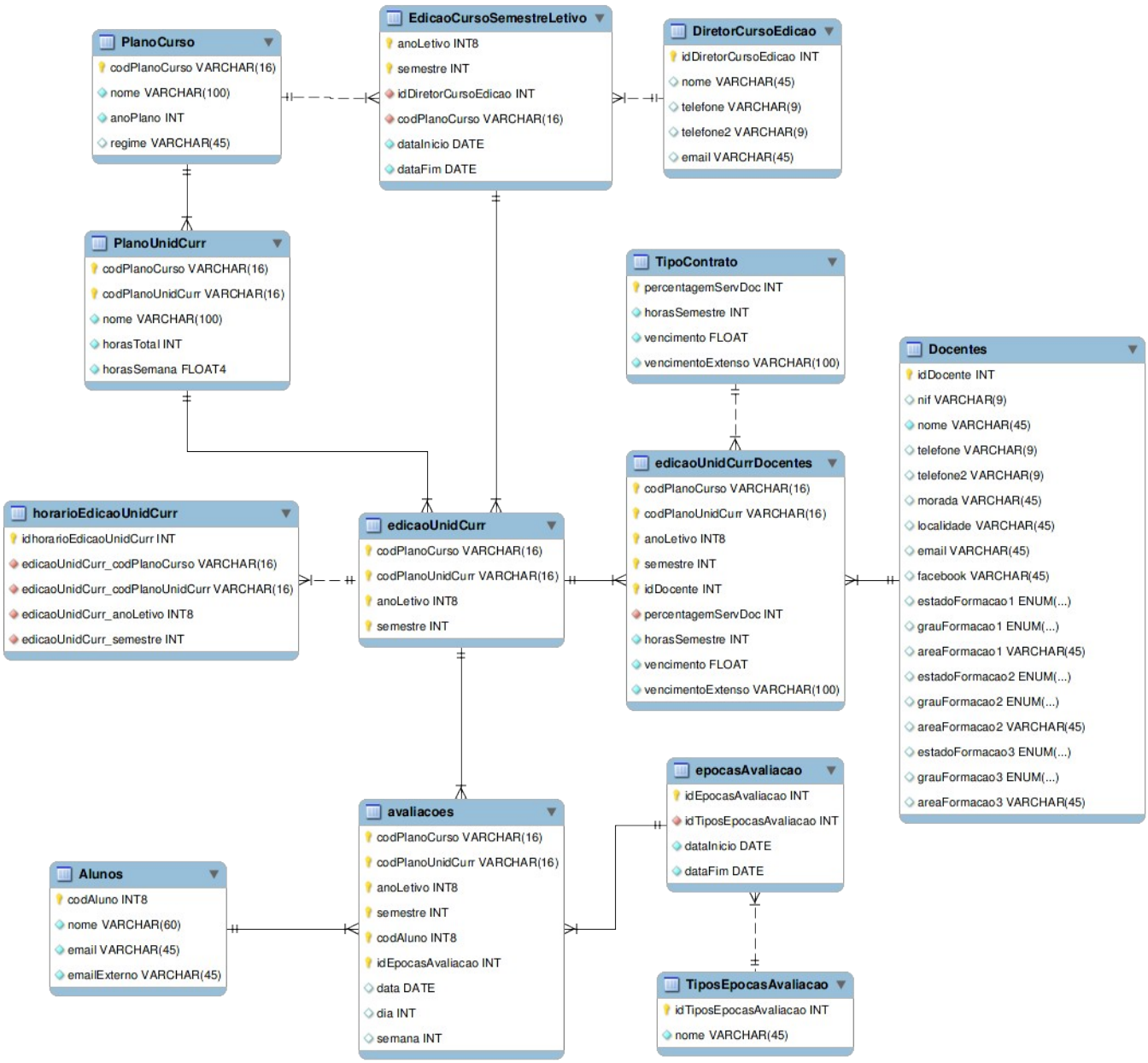

Imagem 2: DER de suporte à solução

Para além destes diagramas, foram ainda fornecidos alguns modelos para os documentos que deveriam ser gerados no final do processo de contratação.

3. Solução proposta

Neste capítulo serão apresentadas as soluções de *software* propostas bem como a arquitetura que fará uso das mesmas.

3.1. Soluções de *software*

Nesta secção serão abordadas as soluções de *software* escolhidas, bem como os motivos que levaram à sua seleção.

3.1.1. *Framework* para aplicações web

No desenvolvimento de aplicações, sites dinâmicos e serviços web, a utilização de uma *framework* torna-se essencial para permitir aliviar a sobrecarga associada ao desenvolvimento de funcionalidades comuns, tais como acesso a bases de dados, gestão da sessão, segurança, etc. Como promovem a reutilização e modularização de código, são geralmente utilizadas em vários projetos distintos o que alarga a base de testes tanto em número como em extensão, melhorando a segurança e estabilidade do *software*.

A *framework* escolhida foi a *Zend Framework,* desenvolvida pela *Zend*, disponível sob a licença *New BSD License* e baseada na linguagem de programação PHP. Todos os componentes desta solução têm uma interdependência relativamente baixa e são baseados na programação orientada a objetos (com garantias de utilização das boas práticas da linguagem). Utiliza um padrão MVC extensível, com suporte a *layouts* e modelos PHP. Do ponto de vista de base de dados são suportados vários motores, incluindo *MariaDB*, *MySQL*, *Oracle*, *Microsoft SQL Server*, *PostgreSQL*, etc. Utiliza ainda um subsistema de *cache* bastante flexível e tem suporte para vários protocolos de fornecimento e consulta de serviços, tais como SOAP, JSON,

JSON-RPC, REST e XML-RPC.

3.1.2. *Framework* de ORM

As *frameworks* de ORM permitem abstração das especificidades inerentes à gestão de uma base de dados através da utilização de objetos e uma API própria que trata dos problemas associados à persistência dos dados.

Foram analisadas as seguintes soluções:

- *CoughPHP - Collection Handling Framework*
- *Propel*
- *Doctrine*

O *CoughPHP* não é uma *framework* MVC, logo não é computacionalmente exigente e não afeta o modo como as vistas de informação são construídas, permitindo a sua integração em projetos já existentes com relativa facilidade. Suporta transações e permite a integração de mecanismos de validação de dados antes da sua persistência na base de dados. Infelizmente não permite a engenharia reversa de base de dados existentes para classes que possam ser utilizadas num projeto de software. Outros dos motivos pelo qual esta *framework* foi descartada prende-se com o facto de o projeto estar a perder *momentum*, relacionado com o facto da última alteração no código-fonte ter sido efetuada à um pouco mais de 3 anos e da página oficial ter sido descontinuada.

Relativamente ao *propel* e ao *doctrine*, são duas *frameworks* com funcionalidades muito semelhantes, consideradas soluções de referência no que diz respeito à implementação de mecanismos de ORM no PHP. A escolha recaiu sobre o *doctrine* pelo facto de permitir modificar o modelo de dados através das anotações nas próprias classes que representam as entidades, não sendo obrigatória a utilização do um formato intermédio em XML para este fim. Por outro lado pareceu ser mais simples a integração do *doctrine* na *Zend Framework* através do módulo *DoctrineORMModule*[1].

1 https://github.com/doctrine/DoctrineORMModule

3.1.3. Gestor de fluxos e de processos

O ProcessMaker foi a solução escolhida como motor para gestão de processos e fluxos de trabalho. Esta solução baseada em PHP fornece os mecanismos necessários para desenhar processos de negócio, definindo etapas, desenhando formulários, gerando documentos e permitindo a integração com serviços existentes.

É uma solução extremamente completa e relativamente bem conseguida, considerada uma solução de referência como uma plataforma de gestão de processos baseada na linguagem programação PHP.

3.2. Arquitetura

A proposta de arquitetura para a solução encontra-se representada na imagem 3.

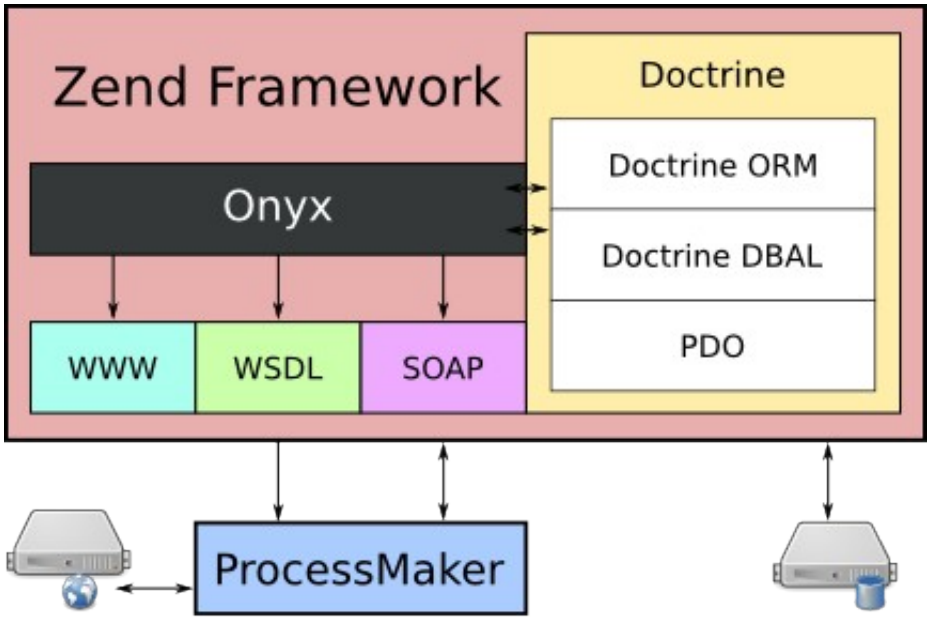

Imagem 3: Arquitetura da solução

Na imagem anterior temos o *Doctrine* como um módulo na *Zend Framework* responsável pelas operações de CRUD na base de dados da aplicação. A interligação entre o *Doctrine* e os serviços disponibilizados pelo sistema é feito pelo

componente *Onyx*, um módulo para a *Zend Framework* desenvolvido no âmbito deste projeto. Publicamente é disponibilizada uma página Web com algumas informações sobre o serviço, o WSDL que descreve o serviço disponibilizado e o interface SOAP para troca de mensagens e execução das operações fornecidas pelo serviço.

No final temos o *ProcessMaker* como a plataforma de gestão de processos que irá utilizar os serviços fornecidos, sempre que necessário, no decorrer do processo de contratação.

4. Implementação

4.1. Sistema base

O sistema base foi construído com recurso ao *Composer*[2], um gestor de dependências baseado em PHP. Através do ficheiro `composer.json` foram definidas as informações necessárias, tais como o nome, descrição e licença do módulo, bem como as respetivas dependências.

```
{
    "name": "c3/onyx",
    "description": "ONYX project from the course Advanced Programming of Systems and Services",
    "license": "GPL v3",
    "keywords": [
        "framework",
        "zf2",
        "doctrine",
        "orm",
        "onyx"
    ],
    "homepage": "http://dei.estg.ipleiria.pt",
    "require": {
        "php": ">=5.3.3",
        "zendframework/zendframework": ">2.1.3",
        "doctrine/doctrine-orm-module": "0.*",
        "doctrine/orm": ">2.3"
    }
}
```

Imagem 4: Conteúdos do ficheiro `composer.json`

Após a execução do comando `php composer.phar install` o ficheiro de configuração é processado e são instaladas todas as dependências solicitadas, devidamente integradas.

Nesta fase a base de dados fornecida foi importada para um SGBD MySQL para possa ser utilizada pela solução.

2 http://getcomposer.org/

4.2. Configuração das dependências

Depois da instalação é necessário proceder à configuração do *Doctrine* e da Zend *Framework* para que ambos os módulos possam comunicar e aceder à camada de dados.

No ficheiro `config/application.config.php` foram especificados os módulos que compõem a aplicação na Zend *Framework* (`DoctrineModule`, `DoctrineORMModule` e `Onyx`). Depois, no ficheiro `config/autoload/global.php`, são adicionadas algumas configurações especificas de configuração do módulo *Doctrine*.

```
return array(
    'doctrine' => array(
        'connection' => array(
            'orm_default' => array(
                'driverClass' => 'Doctrine\DBAL\Driver\PDOMySql\Driver',
                'params' => array(
                    'host'     => 'localhost',
                    'port'     => '3306',
                    'dbname'   => 'CETSSDB',
                    'charset'  => 'utf8'
                )
            )
        )
    )
);
```

Imagem 5: Conteúdos do ficheiro `config/autoload/global.php`

Neste ficheiro, talvez o campo mais relevante seja o campo `dbname` que especifica o nome da base de dados a ser utilizada pelo Doctrine.

As credenciais de acesso à base de dados são definidas no ficheiro `config/autoload/local.php`, através da especificação do nome de utilizador e senha, respetivamente nos valores dos campos `user` e `password`. As boas práticas sugerem a utilização de um ficheiro diferente do ficheiro de configuração global para estas configurações dada a sensibilidade das mesmas, permitindo assim que a ferramenta de controle de versões ignore este ficheiro não sendo estes dados distribuídos na publicação do código-fonte.

As últimas configurações das dependências são feitas diretamente dentro do

módulo que as vai utilizar, sendo as mesmas abordadas na secção seguinte.

4.3. Módulo *Onyx*

Este módulo foi desenvolvido especificamente para este projeto e o seu código-fonte pode ser encontrado na pasta `module/Onyx`.

4.3.1. Configuração

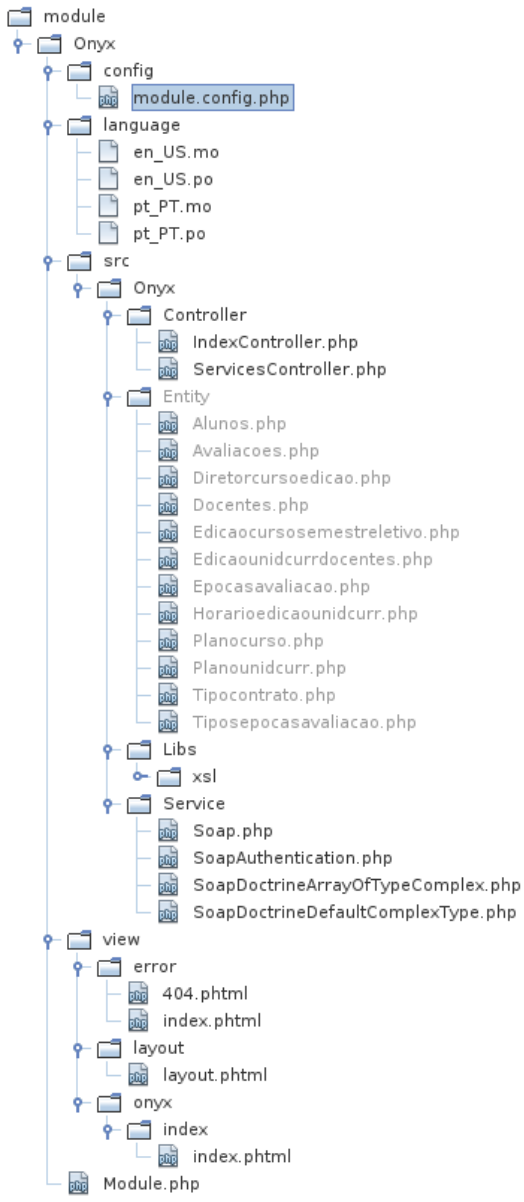

Imagem 6: Conteúdo do módulo *Onyx*

Na pasta `module/Onyx/config` temos o ficheiro de configuração do módulo, na `module/Onyx/language` os ficheiros com os pacotes de idioma disponíveis, em `module/Onyx/src` o código-fonte do módulo (com os controladores, as entidades geradas, as classes que fornecem o serviço, etc) e as vistas na pasta `module/Onyx/view`.

No ficheiro `module/Onyx/config/module.config.php` definimos então a configuração do módulo na *Zend Framework*. Começamos por definir as rotas e respetivos controladores para a vista `WWW` (denominada por `home` no ficheiro), `WSDL`, `XSL` e para os serviços (por exemplo, `SOAP`). Ao definir estas rotas, podemos aceder aos serviços disponibilizados através dos endereços:

- `http://[host]/onyx/` - serviço WWW

- `http://[host]/onyx/services/soap/wsdl` – WSDL com a descrição do serviço
- `http://[host]/onyx/services/soap/xsl` – XSL com a transformação visual a aplicar quando o WSDL é apresentado num interpretador compatível.

Ainda no ficheiro `module/Onyx/config/ module.config.php`, é definido o diretório onde estão disponíveis os ficheiros com o pacote de idioma e respetivo formato, os controladores principais fornecidos pelo módulo (neste caso o controlador para a vista WWW e o de serviços), configurações para a vista WWW e, por fim, as configurações adicionais para o módulo do *Doctrine*, como por exemplo a classe a utilizar como controlador, qual o formato da *cache* para os dados obtidos e diretório onde serão armazenadas as entidades do modelo de dados e respetivo *namespace*.

4.3.2. Geração de entidades

Depois da configuração de todos os módulos, o sistema está pronto a usar as entidades que representam a estrutura de dados da base de dados através de engenharia reversa e com recurso ao *Doctrine*. Para o efeito foi criado um *script bash* (`_generateEntities.bash`), que nos ambientes Linux que suportem esta *shell*, permite executar os comandos necessários para gerar as entidades. Os comandos são os seguintes:

- `./vendor/doctrine/doctrine-module/bin/doctrine-module orm:convert-mapping --namespace="Onyx\\Entity\\" --force --from-database annotation ./module/Onyx/src/` - permite gerar um formato intermédio a partir das informações obtidas a partir da análise de uma base de dados relacional
- `./vendor/doctrine/doctrine-module/bin/doctrine-module orm:generate-entities ./module/Onyx/src/ --generate-annotations=true` – gera as entidades propriamente ditas a partir do formato anterior

Com base nestes comandos, os ficheiros com as classes que representam as

entidades deverão ser criados na pasta `module/Onyx/src/Onyx/Entity/`.

Foram detetadas algumas limitações da *framework Doctrine*, nomeadamente no que diz respeito ao suporte a chaves primárias compostas que são simultaneamente chaves estrangeiras compostas de outras entidades. Caso existam mais do que 2 relações consecutivas deste tipo, o *Doctrine* não é capaz de suportar este modelo de dados. Outra limitação prende-se com os nomes dos campos entre entidades; verificou-se que não se pode utilizar o mesmo nome numa chave estrangeira e respetiva chave primária. Verificou-se ainda que o *Doctrine* não gerou uma das classes para uma entidade constituída exclusivamente por uma chave primária formada por quatro campos de chave estrangeira. Estas limitações levaram à alteração do DER para algo semelhante ao representado na imagem 7.

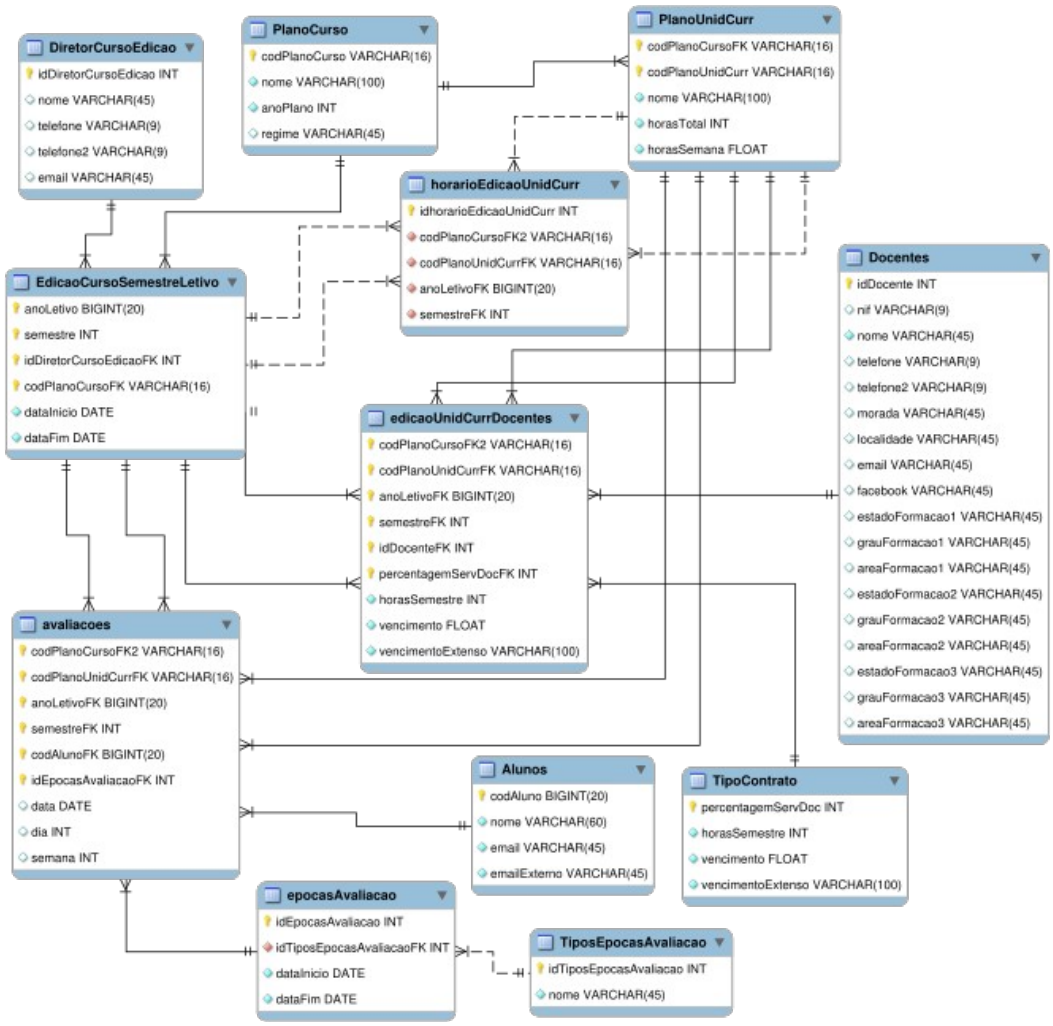

Imagem 7: DER atualizado

4.3.3. Serviço *Onyx*

Este é o serviço responsável pela disponibilização de um interface de controlo baseado em *web services*. No ficheiro `module/Onyx/src/Onyx/Controller/ServicesController.php` está especificada a lógica de funcionamento do controlador do serviço sendo que este está preparado para responder a 3 ações distintas:

- `soap`
- `wsdl`
- `xsl`

Ação SOAP

A ação `soap` é responsável por inicializar o serviço SOAP e associá-lo à classe com os métodos disponibilizados pelo sistema. Ao receber uma ação deste tipo, o serviço irá processar o pedido e invocar o método solicitado com os parâmetros fornecidos.

A concretização do serviço é feita na classe `Onyx\Service\Soap`, disponível no ficheiro `module/Onyx/src/Onyx/Service/Soap.php`. Nesta classe podem ser encontrados os doze métodos implementados no âmbito deste projeto:

- `authenticate` – método responsável pela autenticação no serviço. Foram disponibilizadas duas formas de autenticação distintas:
  - através da invocação direta deste método, que irá devolver um identificador de sessão válido (caso a autenticação seja validada com sucesso), que poderá ser utilizado como o último parâmetro em cada um dos métodos invocados para recuperar a sessão anterior e assim validar o utilizador;
  - através do envio de um cabeçalho SOAP que acompanha o pedido efetuado por um dado cliente e contém as credenciais do utilizador e a especificação de que este método deve ser invocado antes de executar o método pretendido (isto é geralmente feito de forma automática pelo

sistema, caso o cliente suporte o envio de cabeçalho SOAP no pedido).

- `deleteTeacher` – elimina um docente do sistema
- `getCourseCurricularUnits` – obtém as unidades curriculares associadas a um curso
- `getCoursesCurricularUnits` – obtém as unidades curriculares associadas a vários cursos
- `getCurricularUnits` – obtém todas as unidades curriculares existentes no sistema
- `getFutureCoursesEditions` – obtém todos os cursos cuja a data de início é posterior à atual
- `getTeacherByFin` – devolve os dados de um docente com base no seu número de identificação fiscal
- `insertTeacher` – insere um docente no sistema
- `setTeacherCurricularUnits` – associa as unidades curriculares a um docente
- `teacherExists` – verifica se um dado docente existe no sistema
- `updateTeacher` – atualiza os dados de um docente no sistema
- `verifyAuthentication` – valida o identificador de sessão, verificando se existe algum utilizador associado a esse identificador

Para interagir com a base de dados, estes métodos utilizam a API do *Doctrine* e as entidades geradas a partir do modelo de dados.

Ação WSDL

Esta ação é responsável por gerar o ficheiro XML com a especificação do serviço, nomeadamente os métodos e respetivos atributos que devem ser utilizados em cada um dos pedidos. Faz uso pleno da classe `Zend\Soap\AutoDiscover` da *Zend Framework*, responsável pela inspeção e análise da classe que concretiza o serviço, fazendo o processamento dos comentários no formato *PHPDoc* que foram utilizados

para descrever cada um dos métodos, respetivos parâmetros e tipo de dados devolvidos após a sua execução. Quando são instâncias de classes de entidades geradas pelo *Doctrine* como parâmetros ou valores de retorno de métodos do serviço, dado que esta *framework* também utiliza o formato *PHPDoc* para descrever estas classes, a informação sobre as mesmas é utilizada de forma automática pelo sistema para documentação do serviço.

Infelizmente verificou-se que as entidades geradas pelos comandos mencionados na secção 4.3.2 utilizavam o valor `private` ou `protected` para a visibilidade das propriedades das entidades geradas. Como atributos com esta visibilidade não são disponibilizados às classes de inspeção e análise de classes (pois estas fazem parte de uma área de nomes diferente – outro pacote), estas não eram documentadas na descrição do serviço. A solução passou por modificar ligeiramente o comportamento da *framework* através da edição do ficheiro (`vendor/doctrine/orm/lib/Doctrine/ORM/Tools/EntityGenerator.php`, linha 138) e a alteração do valor por omissão da visibilidade de `private` para `public`. Esta alteração não é a solução ideal pois dificulta um pouco mais o processo de atualização, mas é expetável que no futuro esta questão seja resolvida *upstream*.

Foi ainda encontrado outro problema na *Zend Framework* com origem no facto da classe `Zend\Soap\AutoDiscover` utilizar uma estratégia para definição de vetores de tipos dados complexos (ou seja, não primitivos) que não suporta interfaces, o que tornava o módulo *Doctrine* incompatível com `AutoDiscover` da *Zend Framework*. A solução passou pela reimplementação do método `addComplexType` da classe `Zend\Soap\Wsdl\ComplexTypeStrategy\DefaultComplexType` na forma da classe `Onyx\Service\SoapDoctrineDefaultComplexType` que passou também a permitir a utilização de interfaces pela adição da instrução `&&! interface_exists($type)`. A classe reimplementada foi depois utilizada na classe `Onyx\Service\SoapDoctrineArrayOfTypeComplex` em tudo semelhante à classe `Zend\Soap\Wsdl\ComplexTypeStrategy\ArrayOfTypeComplex` (com exceção da classe base que reimplementa), tendo sido esta última utilizada na definição da estratégia do `AutoDiscover`.

Depois de definir a estratégia de `AutoDiscover` e algumas definições do serviço,

esta ação adiciona uma instrução de processamento ao ficheiro XML gerado (com a descrição do serviço), para que os visualizadores compatíveis executem a ação XSL ao renderizarem o conteúdo deste ficheiro, enviando este conteúdo para a aplicação cliente.

Ação XSL

Esta ação é responsável por obter o ficheiro de transformações XSL (`module/Onyx/src/Onyx/Libs/xsl/wsdl-viewer.xsl`) que será utilizado para transformar o conteúdo do ficheiro XML num formato mais amigável para humanos, facilitando assim a leitura das especificações do serviço.

4.3.4. Vista pública

No ficheiro `module/Onyx/src/Onyx/Controller/IndexController.php` está especificado o controlador com a ação que apresenta a versão WWW do projeto. Neste ficheiro existem uma série de instruções de teste (que se encontram comentadas/inativas) e que foram utilizadas para validar o serviço durante o processo de desenvolvimento. No final da execução este controlador passa o controlo para vista que apresenta o site que foi criado.

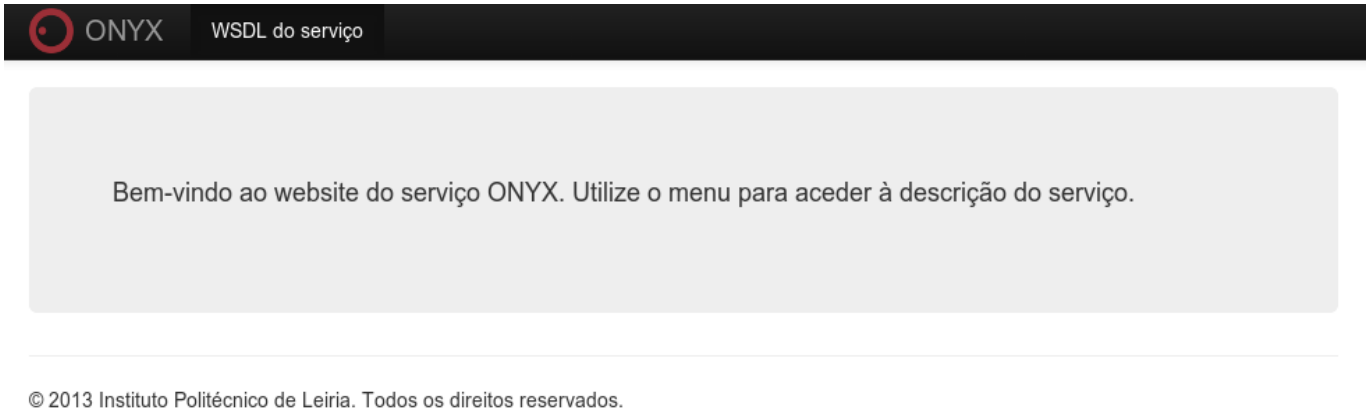

Imagem 8: Captura de ecrã com a vista WWW do site

No menu do site é disponibilizada a hiperligação de acesso à descrição em WSDL do serviço.

Dado que este site faz uso do *Twitter bootstrap*, o seu *layout* apresenta

funcionalidades de *responsive design*, adaptando a sua aparência consoante a área de visualização disponível no dispositivo cliente.

4.3.5. Idiomas

O módulo foi desenvolvido com suporte a multi-idioma. Para o efeito foi utilizada a API de internacionalização da *Zend Framework* e o suporte integrado ao motor *gettext*, através da construção os ficheiros `.po` e `.mo` necessários, devidamente traduzidos na aplicação *PoEdit*.

Os idiomas disponibilizados foi o Inglês (nativamente no código-fonte) e o Português (definido como idioma por omissão). Os resultados desta internacionalização são apenas visíveis na camada visual da aplicação, por exemplo, na vista pública abordada no capítulo anterior.

4.3.6. Controlo de versões

Para este projeto foi utilizado o *Git* como sistema de controlo de versões por permitir a gestão de código de forma distribuída e independente do acesso à Internet. O sistema de cópias de segurança foi complementado através de cópias completas com relativa regularidade e a utilização do serviço *Dropbox* para a sincronização do código-fonte entre sistemas de desenvolvimento.

4.4. *ProcessMaker*

Depois da construção do sistema modular que suporta o modelo de dados fornecido, foi tempo de implementar no *ProcessMaker* o processo de negócio para a contração de docentes para cursos de especialização tecnológica. Para o efeito a plataforma foi instalada e configurada num sistema de desenvolvimento, em base de dados totalmente independentes das utilizadas pelo modelo de dados.

Depois da instalação da solução, foram criados os grupos e os utilizadores no sistema que irão interagir com o processo de negócio.

Users

New | Summary | Edit | Status | Delete | Groups | Authentication | Enter search terr | Search

User Name	Full Name	Status	Role	Last Login	# Cases	Due Date
pass	de Sistemas e Serviços, Programa...	Active	PROCESSMAKER_ADMIN	2013-06-17 22:50:02	5	2020-01-01 00:00:00
docentecet	CET, Docente (docentecet)	Active	PROCESSMAKER_OPERATOR	2013-06-06 01:48:49	0	2014-06-06 00:00:00
docenteexterno	Externo, Docente (docenteexterno)	Active	PROCESSMAKER_OPERATOR		0	2014-06-06 00:00:00
docenteinterno	Interno, Docente (docenteinterno)	Active	PROCESSMAKER_OPERATOR		0	2014-06-06 00:00:00
diretorcet	CET, Diretor (diretorcet)	Active	PROCESSMAKER_OPERATOR	2013-06-16 10:14:12	4	2014-06-06 00:00:00
forcet	CET, FOR (forcet)	Active	PROCESSMAKER_OPERATOR		0	2014-06-06 00:00:00

Imagem 9: Lista de utilizadores

Para além dos utilizadores listados na imagem 9, foram ainda criados alguns grupos que foram depois utilizados ao longo das várias etapas do processo para atribuição de responsabilidades e definição de papeis.

Groups

New | Edit | Delete | Users | Enter search term | Search

Group Name	Status	Users	Tasks
Diretors CET	Active	1	0
Docentes CET	Active	2	1
Docentes Externos	Active	1	0
Docentes Internos	Active	1	0
employees	Active	0	0
FOR.CET	Active	1	0
users	Active	0	0

Imagem 10: Lista de grupos

Após a definição dos utilizadores e grupos, foi tempo de desenhar o processo de negócio propriamente dito.

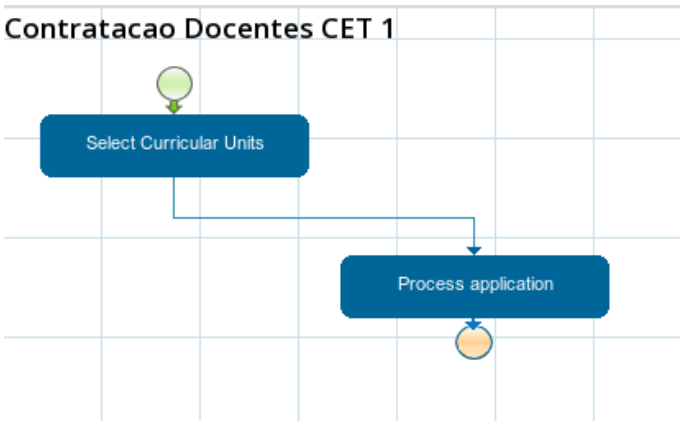

Imagem 11: Diagrama do processo de negócio

Como apenas estão envolvidos dois grupos de utilizadores ao longo deste processo, o diagrama apresenta apenas duas etapas. No entanto cada etapa tem uma série de passos, tantos quantos formulários de entrada ou de saída de dados.

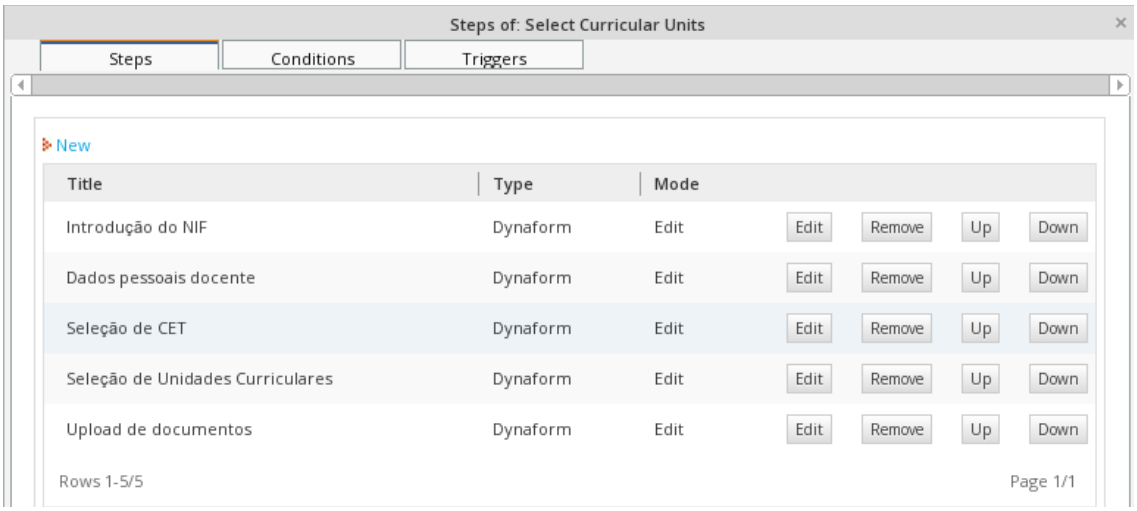

Imagem 12: Passos da primeira etapa do processo

Infelizmente o *ProcessMaker* não permite a criação automática de formulários a partir da análise de ficheiros de descrição de serviços pelo que, para cada um destes passos, tiveram de ser criados manualmente os formulários com os campos e regras para o seu preenchimento.

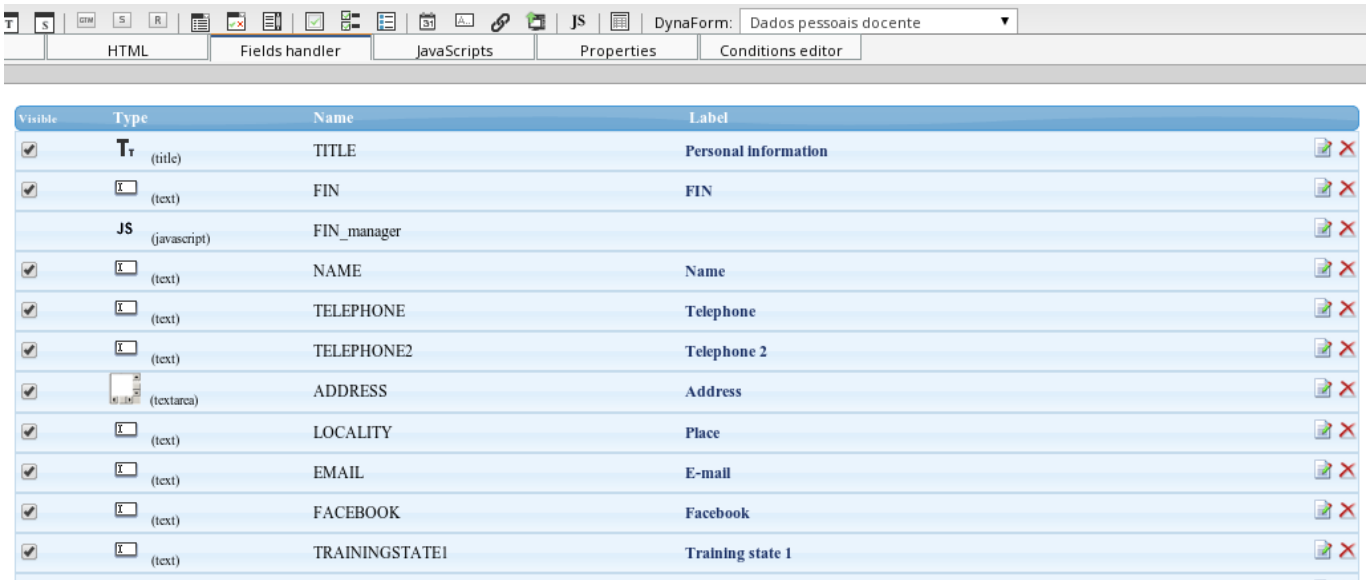

Imagem 13: Exemplo da construção de um formulário

Por outro lado, teve de ser desenvolvido o código PHP necessário para a

implementação dos *triggers* responsáveis pelas invocações dos métodos remotos disponibilizados pelo sistema de informação anterior.

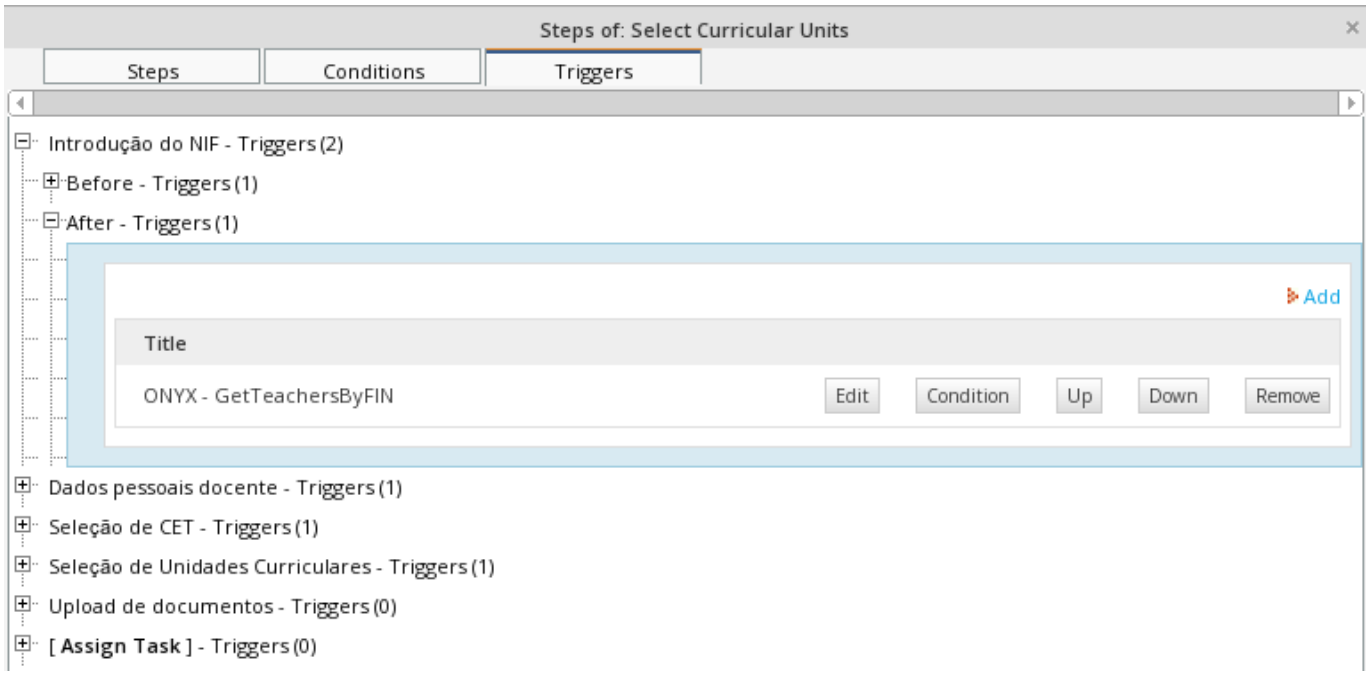

Imagem 14: Exemplo da associação de um *trigger* a um evento

No âmbito do desenho do processo foram ainda definidas as condições de transição entre passos, bem como desenhados alguns dos documentos de saída, como resultado do processo de aprovação da candidatura de um docente.

No final foi possível concluir o ciclo, permitindo a um utilizador apresentar a sua candidatura como docente, inserir/atualizar os seus dados pessoais, selecionar os cursos e unidades curriculares para as quais se pretende candidatar e submeter os documentos necessários. Ao diretor do CET permite que este receba, analise e aprove (ou negue) uma candidatura, sendo estas alterações persistidas na base de dados fornecida com recurso aos módulos desenvolvidos no âmbito deste projeto.

4.5. Instruções de instalação

O processo de instalação da solução é relativamente simples. Depois da importação da base de dados, tal como é descrito nas instruções SQL disponíveis no ficheiro `db/CETSSDB.sql`, deve ser instalado e configurado o módulo desenvolvido e a

plataforma *ProcessMaker*.

Para o módulo baseado na *Zend Framework* basta verificar se o servidor de alojamento dispõe dos requisitos necessários para suportar a aplicação[3], copiar o pacote `src` para o servidor e configurar o servidor web para servir o ficheiro `index.php` da pasta `src/public`. Um exemplo da configuração deste módulo no servidor web Apache, seria o apresentado na imagem 15.

```
pass@pass:~$ cat /etc/apache2/sites-enabled/onyx.conf
Alias /onyx /opt/onyx/public

<Directory /opt/onyx/public>
        Options FollowSymLinks
        DirectoryIndex index.php
        AllowOverride All
        Order allow,deny
        allow from all
</Directory>
```

Imagem 15: Exemplo da configuração do módulo *Onyx* no Apache

O processo de instalação do *ProcessMaker* é descrito na *Wiki* do site[4]. Depois de a plataforma estar devidamente instalada e configurada, devem ser criados os grupos tal como foi apresentado na imagem 10 e importado a última versão do processo exportado para o ficheiro `ProcessMaker/Contratacao_Docentes_CET_*.pm`, onde `*` deve ser substituído pela versão mais recente disponível. Para o efeito deve ser utilizada a opção `Import` no separador `Designer` no *ProcessMaker*.

4.6. Ambiente de testes

Com o objetivo de facilitar a análise e avaliação da solução desenvolvida, foi instalada e configurada uma máquina virtual no sistema *VirtualBox* com a *stack* LAMP onde se procedeu à instalação e configuração da solução tal como foi descrito na secção 4.5. As credenciais de administração de acesso ao sistema, ao SGBD e ao *ProcessMaker* são **`pass`** para nome de utilizador e **`pass`** como senha. No caso dos outros utilizadores existentes no *ProcessMaker*, as suas senhas são iguais ao nome de utilizador, ou seja, o utilizador `docentecet` tem como senha `docentecet`.

3 http://framework.zend.com/manual/1.12/en/requirements.introduction.html
4 http://wiki.processmaker.com/index.php/ProcessMaker_Generic_Installation

O endereço de IP deve ser apresentado na máquina virtual após a sua inicialização ou após a autenticação no sistema. Este endereço de IP pode ser utilizado num *browser* na máquina *host* para aceder ao serviço, com base nos seguintes endereços:

- `http://[endereço de ip]/` - permite o acesso direto ao *ProcessMaker*.
- `http://[endereço de ip]/onyx/` – permite o acesso à vista web do módulo *Onyx*
- `http://[endereço de ip]/onyx/services/soap/wsdl` – apresenta o WSDL do serviço

Para fazer o *deploy* de alterações ao código-fonte do módulo *Onyx*, foi desenvolvido um *script bash* de sincronização com recurso à ferramenta `rsync` e ao `SSH`. Este *script* pode ser consultado no ficheiro `_syncWithVM.bash`, e permite que utilizando um comando do tipo `_syncWithVM.bash [endereço de ip]` se faça atualização do módulo com base nas alterações implementadas na máquina de desenvolvimento.

5. Conclusões e trabalhos futuros

De um modo geral e apesar de não se ter conseguido implementar toda a solução, acreditamos que os objetivos gerais foram atingidos pela implementação de um protótipo que validou a arquitetura e soluções tecnológicas propostas. Apesar de não ter sido possível aplicar imediatamente as metodologias de desenvolvimento ágil, o sistema agora desenvolvido pode ser facilmente adaptável para outros tipos de projetos com relativa facilidade. Ao especificar os dados de acesso a uma base de dados compatível e executando os comandos mencionados neste documento, deverão ser geradas as entidades que poderão ser utilizadas em toda a *framework*, permitindo a apenas a programação da lógica de negócio e da camada de apresentação específicas para esses mesmos projetos (tal como foi demonstrado no exemplo de implementação do *web service*).

Por outro lado, o desenvolvimento de um sistema totalmente modular e a utilização de um sistema de gestão de dependências é uma grande mais valia, permitindo atualizar todas os módulos da solução com um simples comando (`php composer.phar update`). Assim, toda a solução pode beneficiar de todas as atualizações de segurança sem comprometer a sua fiabilidade (exceto se forem introduzidas alterações significativas na API).

Os testes realizados permitiram aferir a qualidade da solução proposta e demonstrar a viabilidade da mesma. Como trabalhos futuros sugerimos a implementação dos restantes processos no sentido de completar a solução e utilizar a totalidade do modelo de dados fornecido.

Este foi um projeto desafiante que permitiu o contacto com tecnologias empresariais e, apesar das dificuldades inerentes à integração dos vários componentes da solução, consideramos o resultado final bastante positivo.