DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

LEIC-A, LEIC-T, LETI, MEIC-T, MEIC-A

Engenharia de Software

2º Semestre – 2014/2015

Enunciado da Terceira Parte do Projecto

1. Terceira Parte do Projecto ES

Na terceira parte do projecto pretende-se estender a solução desenvolvida na segunda parte do projecto com novas funcionalidades, preparar a integração das funcionalidades existentes e a desenvolver com os serviços externos SD-ID e SD-STORE e realizar a gestão do projecto aplicando a metodologia SCRUM.

2. Novas Regras de Negócio

Após análise do protótipo gerado na segunda parte do projecto, o cliente da aplicação decidiu acrescentar o seguinte novo requisito:

- O *username* de um utilizador tem que ter um número de caracteres entre 3 e 8, inclusive.

Deve ainda aproveitar esta entrega para corrigir todas as falhas do projecto relativas à primeira e segunda entregas.

3. Integração dos Serviços Externos

Na terceira parte do projecto vai-se preparar o projecto que está a ser desenvolvido para utilizar os serviços externos SD-ID e SD-STORE. Como estes serviços externos ainda não estão concretizados, eles vão ser abstraídos por duas classes *IDRemoteServices* e *StoreRemoteServices*, cuja concretização será realizada na 4ª parte do projecto. A utilização destas classes, ainda que parcialmente concretizadas, permitirá o desenvolvimento e teste de novas funcionalidades no projecto que dependam da interacção com estes serviços externos sem ser necessário esperar pela concretização da funcionalidade destas classes.

3.1. O Serviço SD-ID

A seguinte listagem apresenta a classe que abstrai o serviço externo SD-ID.

```
package pt.tecnico.bubbledocs.service.remote;

import pt.tecnico.bubbledocs.exception.DuplicateEmailException;
import pt.tecnico.bubbledocs.exception.DuplicateUsernameException;
import pt.tecnico.bubbledocs.exception.InvalidEmailException;
import pt.tecnico.bubbledocs.exception.InvalidUsernameException;
import pt.tecnico.bubbledocs.exception.LoginBubbleDocsException;
import pt.tecnico.bubbledocs.exception.RemoteInvocationException;

public class IDRemoteServices {

    public void createUser(String username, String email)
            throws InvalidUsernameException, DuplicateUsernameException,
            DuplicateEmailException, InvalidEmailException,
            RemoteInvocationException {
        // TODO : the connection and invocation of the remote service
    }

    public void loginUser(String username, String password)
            throws LoginBubbleDocsException, RemoteInvocationException {
        // TODO : the connection and invocation of the remote service
    }

    public void removeUser(String username)
            throws LoginBubbleDocsException, RemoteInvocationException {
        // TODO : the connection and invocation of the remote service
    }

   public void renewPassword(String username)
            throws LoginBubbleDocsException, RemoteInvocationException {
        // TODO : the connection and invocation of the remote service
    }
}
```

A utilização deste serviço remoto vai implicar modificações nos serviços de criação, remoção e *login* de um utilizador. Será ainda introduzido um novo serviço, o serviço de renovação de *password*. A classe *LoginBubbleDocsException* é utilizada para indicar que a *password* indicada no processo de *login* não é a correcta ou o utilizador é desconhecido. As excepções *WrongPasswordException* e *UnknownBubbleDocsUserException* ficam obsoletas. O serviço de *login* deve lançar a excepção *LoginBubbleDocsException* sempre que o par *username* e *password* for incorrecto. Finalmente, a classe *RemoteInvocationException* é utilizada para indicar que não foi possível realizar a invocação remota ao serviço externo SD-ID. As situações representadas pelas restantes classes de excepção estão perfeitamente identificadas pelo próprio nome das classes.

Caso os métodos de *IDRemoteServices* declarassem que lançavam a excepção *RemoteInvocationException* isso estaria a expor algum detalhe sobre a concretização desta classe e estariamos de certo modo a quebrar um pouco a encapsulação desta classe. Para evitar este problema, a solução usual é converter esta excepção numa outra excepção com um nível de abstracção mais alto e que já poderá ser tratada numa camada superior sem provocar quebras de encapsulamento. Assim, cada serviço deve apanhar e tratar a excepção *RemoteInvocationException*.

O tratamento corresponde a converter esta excepção noutra excepção que terá um nível mais alto de abstracção e lançar esta nova excepção. Desta forma, não se expõem os detalhes de concretização dos serviços. Neste caso, a classe da excepção a lançar é *UnavailableServiceException*. O procedimento genérico descrito é representado na seguinte listagem:

```
public void dispatch() {
    IDRemoteServices remote;
    ...
    try {
        // invoke some method on remote
    } catch (RemoteInvocationException rie) {
        throw new UnavailableServiceException();
    }
    ...
}
```

No caso do serviço de *login*, a situação de tratamento da excepção *RemoteInvocationException* é realizada de forma diferente. Este caso será descrito na secção 3.1.2.

Nota: Os serviços a desenvolver ou a actualizar têm que verificar sempre se o token indicado representa um utilizador com uma sessão activa, devendo ser actualizado a data do último acesso associada ao *token*.

3.1.1 Actualização do Serviço de Criação

A forma como se vai integrar este serviço remoto com o serviço de criação de um utilizador é a seguinte:

- A entidade *Utilizador* passa a guardar também um endereço de *email*. No processo de criação de um utilizador passa a ser necessário indicar um endereço de *email*.
- A *password* deixa de ser indicada ao serviço *CreateUser*, dado que a *password* passa a ser gerada pelo serviço externo SD-ID. Assim é necessário alterar o construtor desta classe para passar a ser: *public CreateUser(String userToken, String newUsername, String email, String name)*.
- O utilizador continua a ter uma *password*, mas a inicialização deste campo deixa de ser feita no processo de criação de um utilizador. A utilidade de manter este atributo será descrita mais à frente.
- Realize estas alterações ao domínio, serviços e testes envolvidos e teste as alterações por forma a garantir a correcta alteração.
- Alterar o serviço de criação de um utilizador por forma a utilizar o serviço externo SD-ID via *IDRemoteServices*. Caso não ocorra nenhum erro na invocação ao serviço externo, então deve ser criado um novo utilizador no sistema com o *username*, nome e *email* indicados. Relativamente às excepções que podem acontecer durante a invocação do serviço externo, elas não devem ser apanhadas, excepto a excepção *RemoteInvocationException*. Esta excepção deve ser apanhada e deve ser lançada uma excepção do tipo *UnavailableServiceException*, tal como indicado na secção 3.1.
- É necessário actualizar os testes do serviço *CreateUser* por forma a ter em conta o novo comportamento deste serviço. Dado que deve seguir uma abordagem *test first*, isto quer

dizer, que primeiro deve actualizar os testes e depois é que deverá actualizar o código do serviço com a invocação ao serviço remote *IDRemoteServices*. Dado que não há código ainda definido para esta classe deve utilizar a biblioteca *JMockit* por forma a definir o comportamento esperado desta classe em cada caso de teste.

3.1.2 Actualização do Serviço de Login

O serviço de *login* deve realizar a verificação da password de um utilizador através do método *loginUser* de *IDRemoteServices*. Por forma a aumentar a disponibilidade da aplicação, o sistema deve manter uma cópia local da *password* de cada utilizador. Na situação em que o serviço remote está indisponível (que é representado pela ocorrência da excepção *RemoteInvocationException*), o serviço de *login* deverá realizar a verificação local e aceitar o *login* do utilizador caso a *password* do utilizador seja igual à cópia local mantida no sistema.

Sempre que o *login* é feito com sucesso através do serviço externo, a cópia local da *password* do utilizador em causa deve ser actualizada caso a *password* indicada seja diferente da versão local. Desta forma garante-se a actualização da cópia local da *password* quando esta for alterada no serviço externo SD-ID. Pode acontecer que não exista uma cópia local da *password* (por exemplo, o utilizador ainda não realizou nenum *login*). Realizar a verificação local do *login* quando não exista uma cópia local da *password* corresponde a uma situação de erro que deve ser assinalada com a excepção *UnavailableServiceException*. Na situação em que a verificação local é realizada e falha também deve ser lançada a excepção *UnavailableServiceException*.

Do ponto de vista de desenvolvimento das tarefas para a concretização deste requisito, deverá proceder da mesma forma que no caso anterior[1]. Primeiro, deve actualizar os testes do serviço de *login* tendo em conta a nova funcionalidade. Novamente, será necessário utilizar a biblioteca *JMockit* para definir o comportamento esperado da classe *IDRemoteServices* em cada caso de teste. Segundo, deve actualizar a concretização do serviço de *login* por forma a ter o comportamento indicada nesta secção.

3.1.3 Actualização do Serviço de Remoção

O serviço de remoção de um utilizador também necessita de ser alterado. Este serviço agora também necessita de remover o utilizador do serviço externo SD-ID através da invocação do método *removeUser* de *IDRemoteServices*. Note que o serviço de remoção primeiro deve invocar o serviço externo e caso esta invocação corra sem problemas, então deve remover a informação local sobre o utilizador. Novamente, primeiro deve actualizar a bateria de testes associada a este serviço e só depois é que deverá actualizar a concretização deste serviço.

3.1.4 Serviço de Renovação de *Password*

Deve definir um novo serviço que consiste na geração de uma nova *password* para um utilizador. Este serviço deve utilizar o método *renewPassword* da classe *IDRemoteServices*. Este método irá associar uma nova *password* ao utilizador em causa no serviço externo de autenticação SD-ID. Caso a renovação remota da *password* ocorra com sucesso, então a cópia local da *password* deve ser invalidada. É também necessário definir os casos de teste para este serviço. Este serviço recebe apenas um argumento, o *token* do utilizador que quer renovar a sua *password*.

[1] Este procedimento deve ser seguido relativamente a todos os requisitos que é necessário concretizar.

3.2. O Serviço SD-STORE

A seguinte listagem apresenta a classe que abstrai o serviço externo SD-STORE.

```
package pt.tecnico.bubbledocs.service.remote;

import pt.tecnico.bubbledocs.exception.CannotStoreDocumentException;
import pt.tecnico.bubbledocs.exception.CannotLoadDocumentException;
import pt.tecnico.bubbledocs.exception.RemoteInvocationException;

public class StoreRemoteServices {
    public void storeDocument(String username, String docName, byte[] document)
            throws CannotStoreDocumentException, RemoteInvocationException {
        // TODO : the connection and invocation of the remote service
    }

    public byte[] loadDocument(String username, String docName)
            throws CannotLoadDocumentException, RemoteInvocationException {
        // TODO : the connection and invocation of the remote service
    }
}
```

A classe *CannotStoreDocumentException* é utilizada para indicar que não foi possível guardar o documento indicado no serviço externo SD-STORE (devido a, por exemplo, devido a ter-se excedido a capacidade máxima). A classe *CannotLoadDocumentException* é utilizada para indicar que não foi possível carregar o documento indicado. Finalmente, a classe *RemoteInvocationException* é utilizada para indicar que não foi possível realizar a invocação remota ao serviço externo SD-STORE. Tal como no caso do serviço *IDRemoteServices*, os serviços que utilizarem o serviço remoto *StoreRemoteServices* devem converter a excepção *RemoteInvocationException* em *UnavailableServiceException*.

A utilização deste serviço remoto vai implicar uma modificação do serviço de exportação. Esta novo comportamento deve ser desenvolvido utilizando sempre a abordagem *test-first*.

3.2.1 Actualização do Serviço de Exportação

O serviço de exportação já existente deve ser actualizado por forma a exportar o documento referido no serviço para o serviço externo SD-STORE(). Assim, este serviço primeiro deverá converter uma dada folha de cálculo num documento *XML* e representar este documento como um array de bytes. Isto corresponde ao funcionamento actual do serviço. Agora será necessário invocar o método *storeDocument* da classe *StoreRemoteServices*, indicando os parâmetros correctos. A excepção *CannotStoreDocumentException* que pode acontecer durante a invocação do método *storeDocument* não deve ser tratada neste serviço.

4. Gestão de Projecto

O corpo docente de Engenharia de Software prevê que a realização desta parte do projecto exigirá, em média, cerca de 10 horas de trabalho a cada aluno do grupo. Nesta previsão, o corpo docente assume que os alunos já perceberam o funcionamento da Fénix Framework, a camada de serviços e a utilização da framework de testes JUnit e JMockit.

Durante a execução da terceira parte do projecto os alunos devem seguir a metodologia SCRUM para realizarem a gestão do seu projecto. A gestão do projecto será feita utilizando as

potencialidades do *git-hub*: *issues*, etiquetas, *milestone* e wiki.

A gestão de projecto a aplicar na realização da terceira parte do projecto consiste no seguinte:

1. Definição das histórias que a concretizar para a terceira parte do projecto. Cada história (*user story*) deve descrever uma funcionalidade do sistema do ponto de vista do utilizador.
2. Cada história deve ser decomposta numa ou mais tarefas que necessitam de ser realizadas para concretizar a história.
3. Para cada tarefa deve ser estimada a sua duração, i.e. o tempo que se preve que demorará a realizar a tarefa. Uma história deverá ter uma duração máxima de 4 horas. Idealmente, a duração deverá ser entre uma e duas horas.
4. Cada história deverá ser concretizada apenas por um membro do grupo. As várias tarefas a realizar devem ser distribuídas pelos vários membros do grupo. Deve haver uma preocupação de distribuir as várias tarefas de forma uniforme pelo vários membros do grupo.
5. Registo do estado da execução de cada tarefa. Há medida que cada membro termina as suas tarefas, deve ser registado a realização das tarefas no sistema utilizado para gerir o desenvolvimento do projecto. Em qualquer momento, deverá ser possível saber que tarefas é que ainda não estão realizadas e qual o código envolvido na concretização das tarefas já finalizadas.

Por forma a realizar esta gestão do desenvolvimento do projecto, cada grupo terá de proceder da seguinte forma. Primeiro, cada grupo terá que criar o *milestone Second Sprint* e as seguintes etiquetas (*labels*) de *issues* no GitHub: *story*, *feature*, *new feature*, *test* e *bug*. Estas etiquetas permitirão distinguir os diferentes tipos de *issue* que vão ser criados durante a gestão do projecto:

- *story* deve ser utilizado para classificar *issues* que representam uma história. Deve ser aplicado quando o *issue* descreve uma funcionalidade a desenvolver do ponto de vista do utilizador. Cada história será decomposta numa ou mais tarefas de concretização.
- *feature* deve ser utilizado para classificar *issues* que descrevem uma tarefa (*task*) de concretização em que é necessário alterar código já realizado anteriormente.
- *new feature* deve ser utilizado para classificar *issues* que descrevem uma tarefa de concretização onde o programador vai adicionar novas funcionalidades ao código já desenvolvido.
- *test* deverá ser utilizado para classificar *issues* que correspondem a tarefas de concretização de casos de teste.
- *bug* deverá ser utilizado para classificar *issues* que correspondem a tarefas de resolução de *bugs* do projecto.

Crie as etiquetas *1 s*, *2 s*, *3 s*, *5 s*, *8 s*, *13 s* e *21 s*. Estas etiquetas vão ser utilizadas para representar os pontos de história associados a uma história. Deve ainda criar as etiquetas *0,5 h*, *1 h*, *2 h*, *3 h* e *4 h* que irão ser utilizadas para associar a estimativa do tempo de realização da tarefa ao *issue* que representa a tarefa a realizar.

Segundo, definir as histórias a concretizar durante o terceiro projecto. Para cada história, criar um *issue* com um dado nome colocar a descrição da história na janela de comentário do *issue*. Deverá ainda associar a etiqueta *story* e o *milestone Second Sprint* à nova *issue*.

Terceiro, cada grupo terá que criar uma página no wiki chamada *Second Sprint*. Esta página irá ter a lista de todas as histórias que devem ser concretizadas durante o 3º projecto. Isto corresponderá ao *Sprint Backlog* do SCRUM. Cada história será identificada pelo título do *issue* que representa a história e deverá ter um *link* para a página Web que descreve o *issue* em causa. Deve adicionar uma descrição breve da história na janela de comentário do *issue* associadao à história. O grupo deve ainda estimar os pontos de história desta história e deverá associar a etiqueta correspondente à estimativa ao *issue* em causa.

O quarto passo do processo de gestão corresponde à decomposição de cada história nas tarefas necessárias para atingir o objectivo da história. Assim, para cada tarefa de concretização de uma história deve realizar as seguintes operações:

1. criar um novo *issue*, indicando o título da tarefa;
2. descrever o trabalho a concretizar na janela de comentário do *issue*;
3. atribuir ao novo *issue* a etiqueta *feature*, *new feature*, *bug* ou *test* de acordo com o tipo da tarefa representada pelo *issue*. No caso de uma tarefa de teste, a lista dos casos de teste a concretizar deve ser definida na janela de comentário do *issue* correspondente à tarefa.
4. atribuir o novo *issue* ao *milestone Second Sprint*.
5. Indicar uma estimação do tempo previsto para a concretização da tarefa, associando a etiqueta que representa o tempo de duração estimado a este *issue*.
6. definir a dependência entre o *issue* que representa a tarefa e o *issue* que representa a história que contém a tarefa. A definição desta dependência será feita utilizando a funcionalidade do GitHub de um *issue* poder referenciar outros. Supondo que os identificadores de *issue* da história e da tarefa são $\#st$ e $\#tk$, respectivamente, é necessário realizar o seguinte. Na janela de comentário do *issue* da tarefa deve ser colocado uma linha contendo o seguinte texto: *Task of* $\#st$. O comentário associado ao *issue* da história também deve ser actualizado, acrescentado uma linha de texto com $\#tk$ seguido do título da tarefa.

Antes de cada tarefa começar a ser concretizada, deve ser indicado no *issue* correspondente à tarefa quem é o programador atribuído para concretizar a tarefa. Há medida que cada tarefa vai sendo finalizada, é necessário adicionar uma nova linha na janela de comentário do *issue* correspondente que indica o tempo real despendido a realizar a tarefa. Quando uma tarefa é completamente concretizada deverá ser realizado um *commit* no repositório central com todas as alterações efectuadas para concretizar a tarefa. Este commit deverá ter como mensagem *closes #id*, onde id é o identificador da *issue* que representa a tarefa em causa. Na janela de comentário desta *issue* deve ser colocado o texto *Implemented by commit* seguido pelo identificador do *commit*.

Pode consultar um exemplo da aplicação desta gestão de projecto para a aplicação *PhoneBook* em `https://github.com/tecnico-softeng-distsys-2015/phonebook/wiki/Second-Sprint`.

5. Avaliação da Gestão de Projecto

Semanalmente haverá uma avaliação da gestão do projecto. Esta avaliação será feita pelo menos uma vez durante a aula de laboratório de cada grupo. Grupos que não comparecam ao laboratório terão uma avaliação de 0 nesta componente. A avaliação terá em conta a qualidade do planeamento do projecto e da gestão do projecto que está a ser aplicada.

6. Entrega da Terceira Parte do Projecto

O prazo de entrega da terceira parte do projecto é o dia **17 de Abril de 2015** às **20h00** .

O código produzido deve ser guardado no repositório Git de cada grupo. Cada grupo, após ter concretizado esta parte do projecto e ter guardado no seu repositório o código respectivo, deverá criar a *tag* `R_3`. Esta *tag* representará a versão do código produzido para esta parte do projecto que os alunos querem submeter a avaliação.

Para facilitar o execução do código entregue, os grupos **têm** que utilizar os seguintes dados para a definição da ligação à base de dados:

- **username**: *bubble*
- **password**: *bubbl3*
- **base de dados**: *bubbledb*

6.1. Penalizações

Projectos que guardem ficheiros desnecessários no repositório terão uma penalização na nota de 2 a 4 valores. Consideram-se desnecessários os ficheiros `.class` gerados na compilação das classes Java, os ficheiros `_Base.java` automaticamente gerados na compilação da DML, ou ficheiros `.jar`. Para isso, deverão criar o ficheiro *.gitignore* no directótio base do projecto. Este ficheiro deverá indicar que o directório *target* não deve ser colocado no sistema de controlo de versões Git.