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 Quarta Parte do Projecto

1. Introdução

Nesta parte final do projecto de ES e SD pretende-se:

- Estender a solução desenvolvida até ao momento com novas funcionalidades, o que pode provocar alterações em todas as camadas da aplicação desenvolvida até agora. Ver secções 2 e 4.
- Realização de testes de software. Ver secção 5.
- Aplicar a metodologia SCRUM na gestão do trabalho desenvolvido por cada grupo. No caso dos grupos que estão a realizar ES e SD, a gestão a realizar contém o trabalho a realizar para a 4ª parte do projecto de ES e o 2º projecto de SD. Ver secção 6.
- Os grupos que estão a fazer ES e SD:
  Concretizar as classes *IDRemoteServices* e *StoreRemoteServices* por forma a invocarem os serviços externos SD-ID e SD-STORE desenvolvidos no contexto da cadeira de Sistemas Distribuídos. Ver secção 3. Adicionalmente deverão fazer um teste de sistema distribuído. Ver secção 5
- Os grupos que estão a fazer apenas ES:
  Implementação de um serviço adicional: Atribuir uma função de intervalo a uma célula. Ver secção 4.

2. Nova Arquitectura de Serviços

Na implementação atual, os serviços *BubbleDocs* que invocam serviços remotos podem gerar uma situação de erro se a transação local falhar. Estes serviços, como por exemplo *CreateUserService*, começam por efetuar a invocação remota, no caso do exemplo *IDRemoteServices.createUser*, e depois invocam a camada de domínio de *BubbeDocs*. Nos casos em que a invocação remota falha toda a transação é abortada, mas se a invocação remota tiver sucesso e a transação local falhar o utilizador é criado remotamente e contudo não o é localmente. Uma solução seria pedir ao serviço remoto para apagar o utilizador criado mas provavelmente já teria sido enviado um email com a palavra passe. Uma outra solução é inverter a ordem das invocações, pois se a invocação remota falhar pode-se remover o utilizador local, uma vez que ainda ninguém foi informado da sua criação. A este tipo de estratégia chama-se compensar os efeitos da ação. Assim no caso anterior primeiro é invocada a criação local do utilizador, *CreateUserService*, seguido da sua criação remota, *IDRemoteServices.createUser*, e caso esta última falhe são desfeitos os efeitos locais através da invocação da transação local *RemoveUserService*. Note-se que se a transação de criação local falhar a criação remota nunca chega a acontecer, ou seja o email nunca é enviado. Neste quarto projeto pretende-se refactorizar dos serviços existentes para suportarem este tipo de situações. Assim deve ser criada a seguinte estrutura arquitetural:

- Criar uma camada de serviços de integração, e.g. *integration*.
- Criar um serviço de integração para cada serviço atual, e.g. *CreateUserIntegrator* para *CreateUserService*. Os novos serviços fornecem a mesma interface que os serviços atuais, delegando a invocação nestes. Aconselha-se a criação de uma classe abstrata *BubbleDocsIntegrator* que todos os serviços de integração estendem. Note que o método *execute* desta classe não deve ser *@Atomic*. Compilar e testar.
- Para os serviços atuais que contêm invocações remotas, e.g. *CreateUserService*, mudar as suas classes de teste para passarem a usar os serviços de integração e.g. *CreateUserIntegrator*. Note que a classe de teste também deverá ser renomeada de *CreateUserServiceTest* para *CreateUserIntegratorTest*. Para os serviços que não fazem invocações remotas não é necessário refactorizar os testes pois os seus serviços de integração apenas delegam nos serviços de atuais. Compilar e testar.
- Adicionar às novas classes de teste os caso de teste que sejam relevantes para as situações de compensação. Por exemplo, é necessário testar que *CreateUserIntegrator* gere o caso em que a invocação remota falha, ou seja, que o utilizar criado localmente é removido. Compilar e testar.
- Refactorizar os serviços atuais que contêm invocações remotas para os respectivos serviços de integração de modo a incluir ações de compensação. Por exemplo, ao refactorizar o serviço *CreateUserService* para o correspondente serviço *CreateUserIntegrator* deve-se compensar a criação local do utilizador. Assim, para cada serviço atual que interage com um serviço externo, retirar a invocação ao serviço externo e colocá-la como um método do serviço de integração. Assim, os serviços atuais passam apenas a interagir com o domínio da aplicação, são serviços puramente locais. No serviço de integração, pode ainda ser necessário definir uma operação de compensação da execução do serviço local. A lógica de execução do serviço de integração deve ser concretizada no seu método `execute`. Compilar e testar.

Do conjunto de serviços de integração que efetuam invocações remotas apenas *CreateUserIntegrator* e *RemoveUserIntegrator* necessitam de ações de compensação. Adicionalmente, e para simplificar, a compensação associada ao apagar de um utilizador, a sua criação, apenas necessita de re-criar o utilizador com os seus atributos (exclui-se a palavra passe), toda a restante informação, folhas que lhe pertenciam, etc, não é recuperada.

A camada de integração não pode aceder diretamente ao domínio, devendo-o apenas fazer usando a interface disponibilizada pela camada de serviços. Por esta razão será necessário proceder a algumas alterações nesta camada, por exemplo, o identificador do documento no caso do serviço local de exportação de um documento deve ser disponibilizado à camada de integração. Entre estas alterações contam-se:

- Um novo serviço *GetUserInfoService* em que dado um *username* devolve a informação do utilizador associado, o qual será usado pelo serviço *RemoveUserIntegrator*.
- Um novo serviço *GetUsername4TokenService* em que dado um *token* devolve o *username* associado, o qual será usado por diversos serviços de integração.

Tenha em atenção que, sendo estes dois serviços, *GetUserInfoService* e *GetUsername4TokenService*, auxiliares, eles não possuem um serviço correspondente na camada de integração.

3. Integração dos Serviços Externos

Os alunos que estão a realizar o projecto conjunto de ES e SD têm que realizar a concretização dos métodos das classes *IDRemoteServices* e *StoreRemoteServices* com invocações aos serviços externos SD-ID e SD-STORE na aplicação BUBBLE DOCS tendo em conta o trabalho desenvolvido no contexto do projecto de Sistemas Distribuídos.

4. Novos Serviços

Na quarta parte do projecto é necessário concretizar os seguintes serviços:

- Importar um documento;
- Atribuir uma função binária a uma célula;
- Atribuir uma função de intervalo a uma célula; (Para grupos apenas-ES)
- Obter conteúdo de uma folha

4.1. Importação de Documentos

O serviço de importação de um documento deverá ter a seguinte funcionalidade. Este serviço recebe um identificador de documento e o *token* do utilizador que quer realizar a operação de importação. Caso o *token* corresponda a um utilizador com uma sessão válida, então será necessário aceder ao serviço externo SD-STORE via invocação do método `loadDocument` da classe `StoreRemoteServices`. O *username* a indicar na invocação deste método é o *username* do utilizador que quer realizar a operação de importação. Caso o serviço remoto execute sem qualquer problema, deve então ser criado uma nova folha de cálculo com um conteúdo igual

ao representado no documento importado. Esta nova folha terá como criador o utilizador que realizou esta operação e não tem associado nenhum utilizador em modo leitura ou escrita. Deve ainda ser atribuído um novo identificador à folha de cálculo que representa o documento importado.

Note-se que um utilizador só pode importar folhas que tenha exportado anteriormente. Não é possível ter um utilizador a exportar um documento e ter outro utilizador a importar o mesmo documento, dado que tanto na operação de importação como na de exportação o documento em causa está associado ao utilizador que está a realizar a operação. Na operação de exportação, o nome do documento a indicar no método *storeDocument* da classe *StoreRemoteServices* deve ser o identificador único do documento a exportar convertido para o formato de cadeia de caracteres.

4.2. Atribuição de uma Função Binária

É necessário concretizar o serviço **AssignBinaryFunctionToCell** que irá atribuir uma função binária a uma determinada célula de uma dada folha de cálculo. Este serviço recebe o identificador da célula a alterar, uma expressão que representa a função binária a atribuir, o identificador da folha de cálculo a alterar e o *token* do utilizador que quer realizar a alteração. O serviço deve devolver o novo valor da célula alterada caso tudo corra sem problemas. O serviço deverá ter em conta se o utilizador pode alterar a folha de cálculo e se a célula em causa pode ser alterada.

4.3. Atribuição de uma Função de Intervalo - Grupos apenas ES

É necessário concretizar o serviço **AssignRangeFunctionToCell** que irá atribuir uma função de intervalo a uma determinada célula de uma dada folha de cálculo. Este serviço recebe o identificador da célula a alterar, uma expressão que representa a função de intervalo a atribuir, o identificador da folha de cálculo a alterar e o *token* do utilizador que quer realizar a alteração. O serviço deve devolver o novo valor da célula alterada caso tudo corra sem problemas. O serviço deverá ter em conta se o utilizador pode alterar a folha de cálculo e se a célula em causa pode ser alterada.

4.4. Obter Conteúdo de uma Folha

Finalmente, é necessário concretizar o serviço **GetSpreadSheetContent** que deverá devolver o conteúdo das células preenchidas de uma dada folha de cálculo. Este serviço recebe o *token* do utilizador que quer realizar a operação e o identificador da folha de cálculo a ler. Por razões de simplificação, o serviço devolve uma matriz de objectos do tipo *String*. Esta matriz tem as mesmas dimensões que a folha de cálculo a ler. O valor da avaliação de cada célula preenchida da folha de cálculo é colocado na posição equivalente desta matriz de objectos. Células da folha de cálculo não preenchidas ou sem qualquer conteúdo associado são representadas com a string vazia () na posição correspondente da matriz. Este serviço deve ter em conta se o utilizador que quer realizar a operação pode aceder à folha de cálculo.

5. Testes

Deverão ser efectuados testes de software utilizando as frameworks `JUnit` e `JMockit`. É necessário definir os casos de testes para os novos serviços. Pode também ser necessário actualizar os casos de testes de alguns serviços devido à introdução da nova arquitectura de

integração. Deve sempre seguir a metodologia *Test-driven development* durante o desenvolvimento do seu projecto.

As alterações a realizar aos serviços já existentes por forma a concretizarem a nova arquitectura devem ser feitas aplicando as técnicas de refactorização e aplicando sempre os testes de software já realizados sempre que é aplicado uma dada refactorização. Seguindo este procedimento, há uma garantia relativamente forte que possíveis erros introduzidos no código dos serviços devido à refactorização realizada serão apanhados durante a execução da bateria de casos de testes.

5.1. Testes de Sistema

As invocações que tem sido implementadas no método *main* deverão ser substituídas por um teste de sistema. Este teste exercita a invocação sequencial dos serviços de integração do sistema, simulando um cenário de utilização final por parte de um utilizador que tipicamente começa por se autenticar e de seguida realiza diversas operações sobre folhas de cálculo, sua criação, alteração, visualização, etc. Note que esta sequência corresponde a um único caso de teste e que os serviços já foram testado isoladamente. Os serviços a ser invocados em sequência são serviços de integração. Desta forma, deve no projecto de teste ser criado um *package* de nome *integration* que contém dois *sub-packages*: *component* e *system*. No *package component* estão os testes dos serviços de integração, e.g. *CreateUserIntegratorTest*, enquanto que no *package system* deve estar o teste de sequência *LocalSystemTest* que contém a sequência de invocações aos serviços de integração e usa *JMockit* para as invocações aos serviços remotos. Note que o *teardown* destes testes deve limpar a base de dados uma vez que cada teste é uma sequência de transações, não sendo pois possível executar o teste no contexto de uma única transação que faz *rollback* no fim.

Para os grupos que estão a realizar ES+SD deverão adicionalmente repetir o teste de sistema em modo distribuído. Para isso deverão criar um teste com o nome *RemoteSystemIT* que contém a mesma sequência que *LocalSystemTest* mas em que as invocações a *IDRemoteServices* e *StoreRemoteServices* não são *mocked*.

6. 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, e não tem em conta o desenvolvimento das funcionalidades que estejam relacionadas com o projecto de Sistemas Distribuídos[1]. A contabilização do tempo de concretização destas funcionalidades deve ser realizada no contexto da disciplina Sistemas Distribuídos.

Durante a execução do projecto deve ser seguida a metodologia SCRUM, aplicando o processo definido no enunciado da terceira parte do projecto de ES com as seguintes alterações:

- No caso da quarta parte do projecto, cada grupo deve criar agora o *milestone Third Sprint*. Todas as tarefas a realizar durante a quarta parte do projecto devem ser associadas a este *milestone*.

[1] Estas funcionalidades apenas dizem respeito aos alunos que estão a realizar Engenharia de Software e Sistemas Distribuídos.

- Deve ainda ser criada uma página no wiki do projecto de cada grupo com o nome *Third Sprint*. Esta página irá ter a lista de todas as histórias que devem ser concretizadas durante a quarta parte do projecto. No caso dos grupos que também estão a realizar Sistemas Distribuídos deve também ser indicado as histórias a realizar na segunda parte do projecto de SD. Cada história deve ser representada pelo seu título e pelo *link* para a página Web que descreve o *issue* associado à história, de forma semelhante ao que foi realizado na terceira parte do projecto.

Com o objectivo de conseguir realizar uma melhor gestão do projecto e ter uma noção melhor do que está a ser feito, o processo de gestão do projecto passa a ter os seguintes passos adicionais:

- No início de cada semana, cada grupo realiza o planeamento do trabalho a realizar durante a semana. Para realizar este planeamento é fundamental saber a disponibilidade de cada elemento do grupo durante a semana em causa. Assim, este planeamento será representado por uma matriz com 9 colunas e um número de linhas igual ao número de elementos do grupo. A primeira actividade do planeamento semanal do grupo é indicar o número de horas disponíveis de cada elemento do grupo para trabalhar no projecto de ES e SD nos vários dias da semana. Esta informação é representada nas primeiras sete colunas da matriz de planeamento.

- De seguida, cada elemento do grupo, em função da disponibilidade indicada para a semana, vai indicar na oitava coluna da matriz as tarefas que estima ir realizar durante a semana.

- Durante a semana, e à medida que vai realizando as tarefas atribuídas, cada elemento do grupo insere as tarefas que já terminou na última coluna da matriz. É importante actualizar esta matriz à medida que as tarefas vão sendo realizadas por forma a que o grupo consiga realizar uma gestão de projecto eficaz. Desta forma, é possível determinar tarefas que estão atrasadas e que podem ser atribuídas a outros elementos do grupo que já tenham terminado as suas tarefas.

- **Semanalmente**, no início de cada semana de trabalho, **cada grupo** deve produzir um documento de texto com a retrospectiva do trabalho realizado na semana anterior. Este documento deve indicar o que correu bem e de acordo com o planeado e o que correu mal ou conduziu a atrasos. Este documento deverá resultar da análise da matriz respeitante ao planeamento e execução da semana anterior. O documento deverá ainda conter o planeamento para a semana corrente. Este documento deverá ser colocado no sub-directório '`info`' do projecto com o nome *restrospectiva-x*, onde x deverá ser substituido pelo número da semana a que o documento diz respeito. Na primeira semana de trabalho deve ser criado o documento *restrospectiva-0* que apenas irá conter a matriz relativa ao planeamento da primeira semana de trabalho.

O planeamento e gestão do segundo *sprint* da aplicação *PhoneBook* foi actualizado para estar de acordo com o processo a aplicar na gestão e planeamento da quarta parte do projecto. A nova versão do *sprint* pode ser consultada em `https://github.com/tecnico-softeng-distsys-2015/phonebook/wiki/Third-Sprint`.

Uma componente da nota desta parte do projecto de ES dependerá da qualidade da gestão do projecto aplicada pelo grupo. Esta componente valerá 6 valores. A página *wiki* do projecto com o resultado da aplicação do SCRUM à gestão do projecto será consultada várias vezes

pelos docentes das aulas de laboratório para aferir da qualidade da aplicação da metodologia SCRUM na gestão semanal do trabalho do grupo. Um dos pontos de avaliação será a aula de laboratório de cada grupo. A não comparência semanal do grupo na aula de laboratório resultará numa avaliação negativa nesta componente de avaliação.

7. Entrega da Quarta Parte do Projecto

O prazo de entrega da quarta parte do projecto é o dia **13 de Maio 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_4`. 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*

7.1. 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.