1 Introdução

Como explicado nos trabalhos práticos anteriores, o que se propõe aos alunos de CSS este ano é o desenvolvimento de um novo módulo de um software de gestão de eventos e atividades desportivas, designado por *AulasGes*, centrado na gestão de aulas de diversas modalidades.

Com o sucesso da versão 1.0 do *AulasGes* decidiu-se avançar para a versão 1.5 que tem como objetivo permitir a utilização dos serviços do sistema a partir de diversos dispositivos com ligações à *internet*.

Mais precisamente, nesta nova versão pretende-se oferecer ao utilizador a possibilidade de aceder às operações da aplicação recorrendo a um navegador *web* e a uma aplicação cliente *desktop*. Dada a natureza do serviço oferecido e a qualidade pretendida, decidiu-se recorrer a um servidor aplicacional compatível com as especificações Java EE — o *Wildfly*. Tendo em conta a matéria lecionada na disciplina, foi também decidido que a construção da aplicação cliente *web* deve ser feita recorrendo a *Servlets* e *Java Server Pages* e que a construção da aplicação cliente *desktop* deve ser feita recorrendo ao *JavaFX*.

Dada a urgência em lançar esta nova versão para o mercado (que é como quem diz, dado que o semestre está a acabar), decidiu-se que a primeira versão dos clientes *web* e *desktop* teriam funcionalidades limitadas, cobrindo apenas algumas das operações da aplicação. Mais concretamente:

- o cliente *web* deve permitir um utilizador realizar, através de um navegador, os casos de uso *Inscrever em Aula* e *Visualizar Ocupação de Instalação*
- o cliente *desktop* deve permitir um utilizador realizar os casos de uso *Criar Aula*, *AtivarAula*

2 Que devemos fazer?

Neste trabalho — que é o último[1] — pretende-se que, baseada na vossa versão 1.0 do *AulaGes*, produzam uma nova versão preparada para ter o negócio a correr num servidor aplicacional e dois clientes remotos (uma aplicação web e aplicação desktop). Para isso devem:

- adaptar a camada de negócio da aplicação de forma a que esta possa servir pedidos concorrentes e ser acedida por vários tipos de clientes, tirando partido do *Java EE EJB Container*;
- adaptar e completar o esqueleto da aplicação web que vos é fornecida para interagir com esta camada de negócio, tirando partido do *Java EE Web Container*;
- desenvolver uma aplicação cliente que disponibilize uma interface GUI que aceda à camada de negócio utilizando RMI, recorrendo ao *Java EE Application Client Container*.

O material de apoio fornecido inclui um projeto Java EE que define a estrutura da vossa aplicação em termos de projetos Java EE. Apesar da estrutura ser a da aplicação *AulaGes*, o projeto inclui código-fonte da aplicação *SaleSys* de forma a mostrar exemplos funcionais que podem usar como base. O projeto *Enterprise Java Beans* (do tipo *maven*) é composto por cinco sub-projetos:

- **aulasges**, que agrega a informação dos restantes (não têm de programar nada neste projeto);

[1] Foi decidido juntar as duas últimas entregas previstas inicialmente numa só, de forma a reduzir a carga de trabalho.

- **aulasges-ear**, que contem informação de como empacotar a aplicação empresarial Java a instalar no servidor (não têm de programar nada neste projeto);
- **aulasges-business**, que contem a camada de negócio organizada de acordo com o padrão *Domain Model* e *Data Mapper JPA*;
- **aulasges-web-client**, que é um cliente Web desenvolvido de acordo com o padrão *Model-View-Controller* aplicado à web, que usa JSP para a visualização de informação (padrões *Server Side Template* e *Template View*) e *Servlets* para implementar o controlador (padrão *Front controller*). Este cliente acede à camada de negócio utilizando EJB de sessão remotos;
- **aulasges-gui-client**, que é um cliente com uma GUI programada em *JavaFX* que acede à camada de negócio via RMI recorrendo ao *Java EE Application Client Container*.

Em concreto, devem:

- No projeto correspondente à camada de negócio (*aulasges-business*):
  - Substituir as classes que vos fornecemos pelas classes que desenvolveram na primeira fase deste projeto (ou melhoramentos das mesmas).
  - Identificar quais são os *session beans* a fazer e anotá-los convenientemente.
  - No caso de escolherem usar apenas *stateless session beans* (muito recomendado) efetuar, se necessário, as modificações necessárias em classes que têm estado.
  - Identificar quais os *session beans* que precisam de interfaces remotos, identificar o que estes devem incluir, definir estas interfaces e suas implementações.
  - Transformar a interação com a camada JPA de forma a passar a ter *container-managed persistence*, i.e., de forma a tirar partido dos serviços oferecidos pelo servidor aplicacional. Em particular, devem tirar partido do controle de transações (*Java Transaction API*) e da injeção de contexto e dependências.
  - Identificar se e onde é necessário fazer gestão de concorrência, utilizando as primitivas oferecidas pelo JPA.

- No projeto correspondente à apresentação concretizada pelo cliente web (*aulasges-web-client*):
  - Adaptar a aplicação web de forma a utilizar a vossa camada de negócio e disponibilizar as funcionalidades indicadas anteriormente. O acesso à camada de negócio deverá ser feito através das interfaces dos *session beans*.
  - O controlador, que segue o padrão *Front Controller*, é totalmente reutilizável, mas devem listar a correspondência entre endereços web e as vossas ações no ficheiro *app.properties*. Só precisam de proceder ao desenvolvimento das ações.
  - A visualização (que recorre a JSP) tem de ser refeita para ir de encontro aos vossos casos de uso e o mesmo acontece com os *helpers ou view models*. As modificações que efetuarem devem continuar a respeitar os padrões escolhidos.

- No projeto correspondente à apresentação concretizada pelo cliente desktop (*aulasges-gui-client*):
  - Recorrer a uma variante do MVC apropriada recorrendo a propriedades e *data binding*.
  - Desenhar as cenas utilizando a aplicação *Scene Builder*.
  - Programar os controladores seguindo o padrão *Page Controller*.

- O acesso à camada de negócio deve ser feito através da injeção de *session beans* remotos, mas seguindo as regras do *Java EE application client container*.

A execução da aplicação *AulasGes* vai ser distribuída pelas máquinas dos utilizadores, um servidor Java EE, e um servidor de base de dados. A base de dados idealmente estará numa máquina diferente da do servidor aplicacional. Como anteriormente, devem utilizar a máquina disponível no endereço http://dbserver.alunos.di.fc.ul.pt/. Para efeitos de desenvolvimento, usam um servidor Java EE (Wildfly) disponível localmente (nos vossos computadores ou nas máquinas dos laboratórios).

Devem continuar a usar o ***SonarLint*** para vos ajudar a controlar a qualidade do vosso código e o ***Git*** para vos ajudar no desenvolvimento cooperativo.

3 Por onde começar?

Um elemento do grupo de trabalho deverá seguir as instruções para fazer *fork* do repositório em https://git.alunos.di.fc.ul.pt/css000/css_meta5_1920. **Importante:** devem definir o vosso *fork* como sendo privado (outros alunos não o poderão ver) e adicionar à lista de membros do projeto o utilizador css000 com o nível de *Reporter*.

O repositório fornecido contém um projeto *maven* com o código de uma versão do sistema *SaleSys* com características semelhantes às que se pretende nesta entrega. Recordem que, no início do semestre, se seguiram as instruções em DEMO.md, já viram como se importa um projeto deste género, como se põe a correr a aplicação no servidor *Widlfly* e como se executam os clientes *web* e *desktop*. Antes de começarem o desenvolvimento devem voltar a fazer esta demo para se familiarizarem com tudo o que está envolvido.

Para começar o desenvolvimento:

- Devem proceder à importação do projeto *maven* fornecido (a raiz, ou seja, **aulasges**) para o vosso IDE recorrendo à importação de projetos *maven* que o IDE disponibiliza.
- O projeto usa JPA ligado a MySQL. Devem alterar os ficheiros que definem estas ligações para usarem a base de dados do vosso grupo, por exemplo, *aulasges-business/src/main/java/resources/aulas-ee-ds-xml*. Nota: só irá funcionar se estiverem conectados ao departamento via VPN.
- Se tudo correr como esperado, o projeto importa sem erros. Notem que devem ter acesso à internet durante o processo de importação para que o *Maven* descarregue as dependências necessárias. Se ocorrer algum problema durante a importação (e tinham rede, vpn, etc.), devem, no Eclipse, escolher a opção *Maven* > *Update Project...* e depois selecionar quais os projetos a atualizar e selecionar a opção *Force Update of Snapshots/Releases* para forçar que o plug-in do Maven repita o processo de importação de dependências e de configuração do Eclipse.
- Instalar a aplicação no servidor *Wildfly* do cliente Web fazendo por exemplo, "Run As -> Run on Server" no modulo *aulasges-web-client*.
- Aceder à aplicação usando um navegador web e o cliente *JavaFX* fornecido, certificando-se que está tudo a funcionar como é suposto. Note que foi configurado o *drop&create* do esquema da base de dados de cada vez que a aplicação é instalada (por ser o mais apropriado para fazer testes).
- Efetuar as alterações descritas na secção anterior.

4 Como e quando entregamos?

Identifiquem o *commit* como sendo a entrega 5 (`git tag entrega5`) e coloquem essa identificação no servidor gitlab (`git push origin entrega5`). O deadline é **31 maio**.

O repositório deve conter:

1. um único documento PDF, chamado `Relatorio`, com as decisões mais importantes que tenham tomado em termos de desenho da aplicação, nomeadamente a forma como resolverem cada um dos aspetos listados na secção sobre o que têm a fazer (por exemplo, que EJBs definiram, que interfaces definiram para esses EJBs, que problemas de concorrência identificaram e como são estes evitados, etc). Devem ainda mencionar modificações relevantes que tenham realizado relativamente ao que reportaram no relatório da Entrega 4.
2. O código fonte do vosso projeto (o que inclui o .pom).
3. Alguns *screenshots* que ilustrem a apresentação da aplicação em cada um dos casos de uso (à semelhança do que existe no exemplo fornecido).

Recordem que o trabalho é em grupo, mas a avaliação é individual e que será utilizado o histórico do Git para aferir o grau de participação dos diferentes elementos do grupo no trabalho desenvolvido.

Dicas para Problemas Frequentes

- Façam *reorganize imports* em todas as classes de todos os módulos do projeto de forma a não deixarem quaisquer referências a classes do *SaleSys* que vão causar erros mais tarde.
- Se obtiverem erros na instalação da aplicação **analisem os erros que aparecem na consola** para tentar perceber o que está a correr mal.
- Têm de ter a aplicação a correr no servidor para poder usar qualquer uma das aplicações cliente.
- Quem tem *Windows* e a placa gráfica Nvidia, vai ter o porto 9990 (por *default* o porto usado pelo wildfly para a consola de admin) ocupado pela placa gráfica. Devem alterar o ficheiro *"path-to-wildfly"/standalone/configuration/standalone.xml* e alterar o sítio em que é referida a porta 9990 para outra, por exemplo, 8990 e fazer *restart* do servidor.
- Se tiverem alguma outra coisa a correr no porto 8080 (onde vai ficar publicado por omissão o cliente web) terminem-na.
- Não se esqueçam que durante da execução do cliente web, o navegador pode usar o que tem na cache em vez de ir buscar o código com as alterações que acabaram de fazer. Averiguem como podem forçar o *reload* no navegador que estão a usar. É recomendado usarem o modo de navegação privada durante o desenvolvimento.
- No cliente JavaFX, se obtiverem erros na execução deste cliente **analisem os erros que aparecem na consola**. Se houver erros relativos a permissões executem o script *appclient* com *sudo*. Se houver erros relacionados como JavaFx olhem de novo para os requisitos de instalação fornecidos no início do semestre, nomeadamente para tudo o que diz respeito ao Java.