Relatório CSS 2021/2022

Projeto 02

Grupo 051

Rodrigo Branco, fc54457
Sérgio Ferreira, fc54419
Vasco Lopes, fc54410

Conteúdo

Capítulo 1

Estrutura do código

1.1 Business

Começámos por criar as interfaces remotas para os serviços guardados no `package facade`. Estes serviços são `stateless` e os `handler`'s que cada um contém possuem `@EJB`, que permite que um único `handler` seja chamado.

Dentro dos `handler`'s substituímos o `entityManager` pelos catálogos que vão ser utilizados por estes. Estes catálogos também possuem a primitiva `@EJB` que implica que apenas um catálogo é carregado. Estes catálogos também são `stateless`, o que significa que não guardam nada em memória.

Os catálogos possuem, cada um, um `entityManager` que vai ter a primitiva `@PersistenceContext`. Esta primitiva implica que a `entity` vai ser persistente, ou seja, vai saber o contexto de toda a sua execução e vai comunicar diretamente com a base de dados.

Como alterámos os `handler`'s, e já não possuem nenhum `entity manager`, precisámos de usar `@Transacional` que substitui o `getBegin()` e `commit()` do `getTransaction()`. Primeiramente aplicámos esta primitiva em todas as funções dos `handler`'s, mas depois de uma análise mais profunda escolhemos mete-la em funções onde realmente eram necessárias, como criar atividade ou criar horários nos respetivos catálogos.

Depois das transações estarem configuradas, tentámos perceber onde é que ocorreria problemas de gestão de concorrência. Estes ocorrem quando adicionamos sessões e horários às atividades, quando modificamos o número de participantes nas sessões e quando alteramos a lista de sessões no instrutor. Para corrigir ou atenuar estes problemas usámos `@Version` nas classes: `Activity`, `Session` e `Instructor`. Assim, quando é feito um `commit` da transação é incrementado o número da versão. Assim, se houver 2 alterações simultâneas sobre o mesmo objeto, a que for feita em último não vai ocorrer.

1.2 Cliente Web

Para criar o cliente WEB, usámos o servidor `Wildfly` e o esqueleto fornecido pelos professores, sendo que também modificámos o `app.properties` de forma a criar *links* com as ações pretendidas.

O `FrontController` é responsável pelos pedidos `GET` e `POST` e possuímos dois modelos para cada caso de uso utilizado nesta parte. Cada ação é responsável por uma página e direciona à página seguinte.

1.3 Cliente GUI

Para a criação do cliente GUI, tirámos partido do *Scene Builder* para construir 3 interfaces: `index.fxml`, `createActivity.fxml` e `setNewSchedule.fxml`. A primeira interface tem dois botões que ligam às duas outras interfaces, ou seja, a partir do menu inicial - `index` - é possível aceder à secção que cria atividades e à secção que cria horários para atividades regulares.

Depois da criação das interfaces, usámos o esqueleto do `Controller` gerado pelo *Scene Builder* e implementámos o código, tendo em conta que o `Controller` está ligado ao `Model`, como dita o padrão `MVC`.

Os campos que possam ser preenchidos possuem *binding*'s bidirecionais entre os seus valores e os campos do `Model`. Nos campos em que não é possivel bidirecionar o seu conteúdo, é utilizada uma `onAction` quando os mesmos são alterados: o `Controller` pede ao `Model` para atualizar o seu campo correspondente com o novo valor, simulando assim o *binding* bidirecional.

Por fim, com todos os campos preenchidos e feita a ação, o `Controller` chama o serviço remoto do servidor para o respetivo caso de uso, retornando assim uma mensagem de sucesso ou de erro.

Capítulo 2

Limitações

Devido à falta de tempo, não foi possível realizar testes para identificar todas as limitações do projeto. Temos a noção que há muitas, no entanto as seguintes foram as que conseguimos identificar.

2.1 CheckBox GUI

Uma das limitações que identificámos foi que, no GUI, no menu de definir atividade, temos que quando selecionamos e desselecionamos a *checkBox* `isRegular`, o `lock` feito no `numParticipants` deixa de existir, logo suscetível a falhas.

2.2 Mensagens GUI

Identificámos que as mensagens dadas pelo GUI, tanto de erro como de sucesso, são demonstradas apenas uma vez, ocorrendo quando é criada a primeira atividade.

2.3 Datas arbitrárias GUI

No segundo caso de uso, a seleção das datas não corresponde ao número de datas correspondentes à atividade selecionada, isto é, se a atividade tiver 3 sessões, no menu vai continuar a aparecer como 5 datas por selecionar.

2.4 Mensagens de erro GUI

No segundo caso de uso, as mensagens de erros na definição de um novo horário nem sempre correspondem ao verdadeiro erro, nem mesmo ao caso de uso correto.

2.5 Verificações GUI

Para conseguir pôr os casos de uso funcionais, tivemos que remover algumas das verificações pedidas no primeiro enunciado, pelo que é possível passar parâmetros errados, que podem levar a erros na aplicação GUI.

2.6 Mensagens de erro WEB

Quando ocorre um erro na aplicação WEB, a mensagem apresentada é a de erro geral, e não a específica do erro que ocorreu.

2.7 Verificações WEB

Para conseguir pôr os casos de uso funcionais, tivemos que remover algumas das verificações pedidas no primeiro enunciado, pelo que é possível passar parâmetros errados, que podem levar a erros na aplicação WEB.

2.8 Número de sessões WEB

No caso de uso 4, não fazemos a verificação do número de sessões, pelo que pedimos ao utilizador que insira o número correto, de acordo com a atividade que escolher.