Trabalho Prático de Programação Distribuída

Meta 2

Server Rest API com Consumidor

Trabalho realizado por:

Gonçalo Costa Enes, 2019138654 (Turma *P2*)

João Alves Pereira de Carvalho, 2019131769 (Turma *P1*)

Henrique Jorge Correia Barradas, 2019135835 (Turma *P2*)

Introdução

Para esta próxima meta do trabalho prático, foi-nos pedido para desenvolver uma *API Rest* e, para além disso, um consumidor *HTTP* para conseguir aceder aos *endpoints* da mesma.

Para conseguir realizar o pedido, foi necessária uma adaptação da classe previamente criada na meta anterior; a classe *DBManager*. Esta classe serve para lidar com todas as operações da nossa base de dados e serve como um "repositório" para os controladores conseguirem ter a informação que necessitam.

Finalmente, o consumidor foi criado com recurso a uma classe *Consumer* que possui um método para enviar um pedido e escrever a resposta. Este método é estático, pelo que não é necessário instanciar a classe e poderá ser utilizada onde necessário para devolver a informação contida no pedido *HTTP*.Foi também adaptada a *interface* de utilizador criada na meta anterior, exceto que, ao invés de utilizar *sockets TCP*, os pedidos são enviados, mais uma vez, como *HTTP*.

Tabela com endpoints

Método	***URI***	**Operação**	**Descrição**	**Corpo do pedido**	**Corpo da resposta**
GET	/event/	Read	Devolve uma pequena mensagem alusiva ao trabalho	Não possui	Uma mensagem (*string*)
GET	/isAdmin	Read	Devolve se o utilizador que está atualmente a usar a plataforma é administrador	Não possui	Uma mensagem que avisa o utilizador se tem ou não privilégios de administrador
GET	/event/admin/	Read	Devolve uma lista (com ou sem filtros)	Uma variável pesquisa que contém a pesquisa do utilizador	Uma lista (*string*) com todos os eventos criados (com ou sem filtros)
GET	/event/admin/presences/{eventId}	Read	Devolve uma lista com todas as presenças de um evento passado por parâmetro	Identificador único do evento que deseja procurar	Uma lista (string) com ou sem filtros de todas as presenças de um determinado evento
GET	/event/	Read	Devolve uma lista (string) com todos os eventos em que o utilizador tem presença	Uma variável pesquisa que contém a pesquisa do utilizador	Uma lista (string) com ou sem filtros de todas as presenças do utilizador
POST	/login	Create	Autentica o utilizador com os detalhes inseridos pelo mesmo	As informações de login do utilizador	Gera um novo *Token JWT*

POST	/register	Create	Cria uma nova conta, novo registo de utilizador, na base de dados, e permite que o mesmo efetue um login mais tarde com esses detalhes	As informações de registo do utilizador, classe Utilizador	Uma mensagem a indicar se o registo foi, ou não, bem sucedido
POST	/event/admin/	Create	Cria um novo evento na base de dados e permite que o mesmo seja, mais tarde, os utilizadores conseguirem assistir e marcar presença	As informações do evento, classe Evento	Uma mensagem a indicar se a criação do evento foi, ou não, bem sucedida
POST	/event/	Create	Permite aos utilizadores enviarem um código de 6 dígitos para registarem a sua presença num evento	O código inserido pelo utilizador	Uma mensagem a indicar se foi, ou não, registado nesse determinado evento e a criação de um novo registo na tabela Presença com o id do utilizador e o id do evento
PUT	/event/admin/{eventId}	Update	Permite ao administrador gerar um código aleatório para	O identificador do evento e o tempo de validade	Uma mensagem a indicar se esse código foi, ou não,

			um evento com tempo de validação do mesmo	(em minutos) do código	adicionado à tabela do Evento
DELETE	/event/admin/{eventId}	Delete	Permite ao administrador apagar um evento à sua escolha, desde que o mesmo não contenha presenças	Não possui	Uma mensagem que indica que o evento foi removido, ou não

Consumidor (*HTTP*)

Como já referido na introdução, foi também necessária a criação de uma aplicação cliente – aplicação esta, que, para efeitos de conveniência, foi adaptada da meta anterior, tendo apenas alterado a maneira como a mesma comunica com o servidor principal (*TCP* para pedidos *HTTP*).

Classe *Consumer*

A classe *Consumer* é composta por apenas um método, de seu nome: *sendRequestAndShowResponse()*. O método aceita vários parâmetros:

String *URI* – como, por exemplo: http://localhost:8080/event/

String Verb – "verbo" do pedido, por exemplo: POST, PUT, GET

String authorizationValue – onde é armazenado a autorização e o token

String body – o corpo da mensagem, necessário para criar eventos, p.ex

Para além desse método, possui uma pequena *UI* que serve para tratamento de dados vindos do utilizador antes de aceder aos *endpoints* com a informação desejada. Esta *UI* possui imensas funções, todas elas servem o propósito de obter a informação necessária para realizar o pedido à *API*, garantindo que os dados chegam de forma correta e que sejam válidos.

```
1 usage   GonzoEnes
public boolean submitEventCode() throws IOException {

    int eventCode = InputProtection.readInt( title: "\tInsira o código do evento: ");

    String submitEventCodeUri = "http://localhost:8080/event/submit?eventCode=" + eventCode;

    String response = Consumer.sendRequestAndShowResponse(submitEventCodeUri,  verb: "POST",  authorizationValue: "bearer " + this.token,

    System.out.println(response);

    return true;
}
```

Figura 1- Exemplo de função da UI com chamada ao método sendRequestAndShowResponse()

Conclusão

O presente trabalho teve, como objetivo, a consolidação de conhecimentos na implementação de uma *API Rest* com recurso à *framework Springboot*.

Apesar de, nesta meta, tudo ser novo, considera-se que o grupo conseguiu realizar um trabalho bastante sólido, implementando tudo o que foi requirido no enunciado e ultrapassando inúmeros obstáculos.

Em jeito de conclusão, o grupo considera que o trabalho teve um balanço geralmente positivo, tendo todos os membros trabalhado de forma semelhante, utilizando, para o efeito, ferramentas como o *Code Together*.