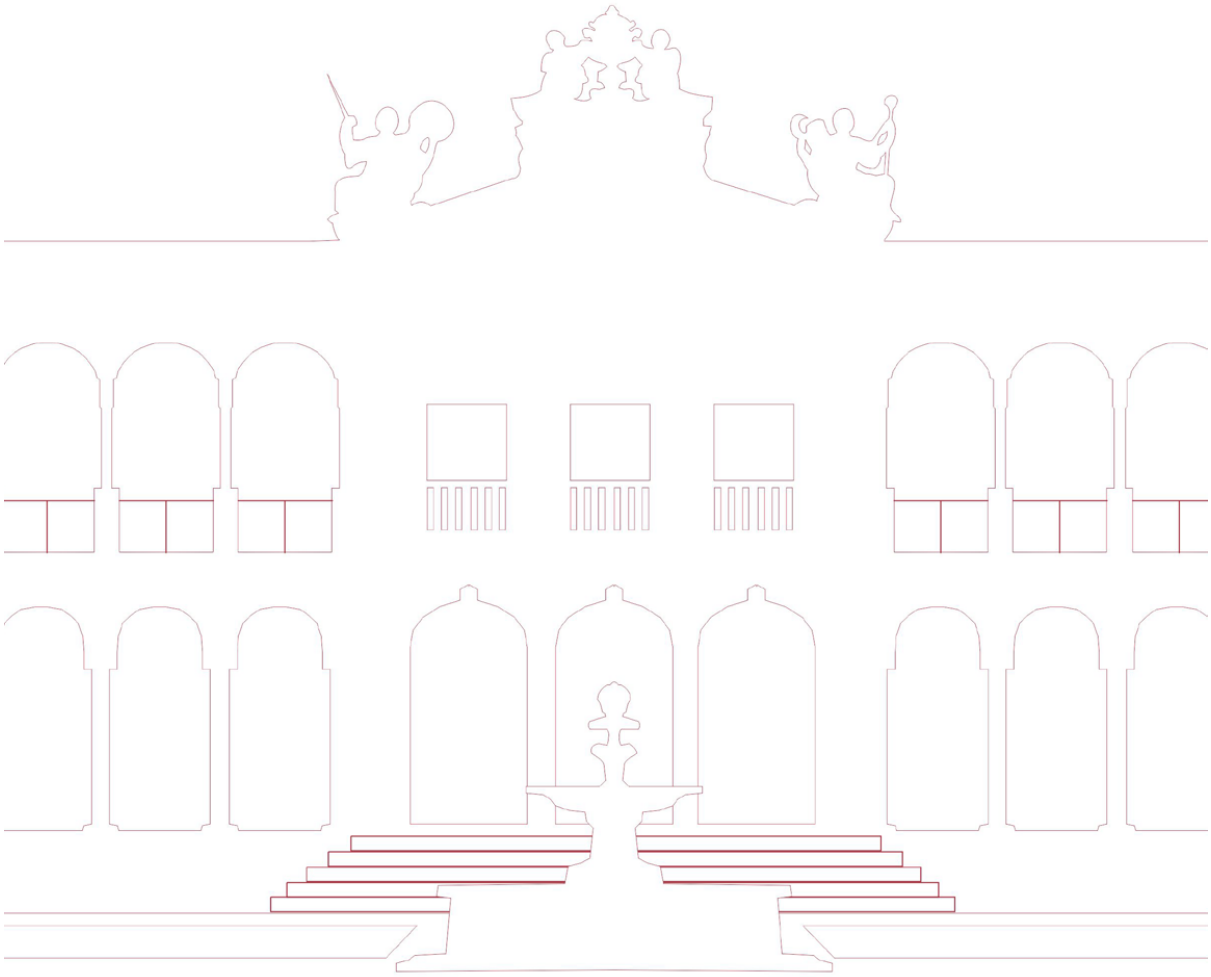
Relatório do 2º trabalho prático
Running Events

UNIVERSIDADE DE ÉVORA
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Licenciatura em Eng. Informática
Sistemas Distribuídos



Ricardo Oliveira 42647

Docente: José Saias

Évora, 17 de janeiro de 2022

Conteúdo

1 Introdução

A segunda parte do trabalho prático da Unidade Curricular de Sistemas Distribuídos consiste na implementação de um sistema com arquitetura distribuída para a gestão de eventos desportivos, nomeadamente, a respetiva cronometragem dos tempos de provas. Para tal, a aplicação deve estabelecer a comunicação entre o cliente e o servidor, de forma semelhante à parte anterior deste trabalho, de modo a serem executadas diversas operações, nomeadamente o registo, tanto de eventos, como dos participantes do mesmo; obter a lista de eventos num determinado dia e a lista de participantes inscritos num determinado evento e todos os dados a eles associados.
Por outro lado, nesta fase, será necessário implementar também, um *REST API*, que será responsável por transmitir, ao servidor, os dados dos participantes registados ao atravessarem os sensores de prova. Poderá este também, de acordo a opção de cada cliente, enviar-lhe notificações, com informações relativamente à passagem de cada participante por cada sensor.

De modo a estabelecer a comunicação conjunta dos três módulos (*backend*, *client* e *REST API*), foi recorrido ao *framework Spring Boot*.

2 Desenvolvimento e Implementação

2.1 Pensamento

Numa fase inicial, analisou-se qual seria a melhor abordagem para implementar os diferentes métodos e de que modo se iriam transmitir dados entre si. Para tal, criaram-se três aplicações distintas, recorrendo em todas à ferramenta *Apache Maven* e à *framework Spring Boot*. Quanto à transmissão de dados, o método mais adequado seria utilizando os métodos de *HTML* e/ou objetos *JSON*.
No que toca à possível implementação a cada um dos módulos, para o do cliente e sensor, chegou-se à conclusão que somente duas classes para cada um deles, seria suficiente. No entanto, utilizando o *framework* já referido anteriormente, em relação ao módulo de *backend*/servidor, uma vez que será necessário criar controladores e restantes métodos, mais classes irão ser necessárias.

2.2 Implementação

Numa fase inicial, tal como já foi referido, o módulo do servidor deverá ter todos os controladores, serviços e repositórios para consultar e modificar todos os dados relativamente aos eventos. Para tal, foram implementadas duas clases (com os respetivos controlador, serviço e repositório para ambas), onde foram iniciados os objetos do tipo participante e evento, bem como o seu construtor.
Após terminada esta etapa, o módulo do cliente foi implementado, onde, através de uma interface básica (terminal), podem ser executadas diversas tarefas de modo a controlar e gerir os eventos. Para cada opção selecionada pelo cliente, pode estar ser do tipo *POST* (se o objetivo for escrever na base de dados) ou do tipo *GET* (se o objetivo for a consulta de dados), no entanto, para cada um deste métodos, foram utilizadas diverentes formas de enviar os dados, tanto de consulta como de alteração, sendo o primeiro deles os próprios parâmetros de *HTML*, por outros lado, para o método *POST*, foram enviados por um objeto *JSON*. A cada pedido do cliente, o servidor (através dos controladores), verifica que funcionalidade o cliente pretende fazer, passando então os dados para o serviço, que irá posteriormente aceder ao repositório, concluindo assim a tarefa determinada.
Numa fase final, foi implementado o *REST API* para os sensores, que, de uma forma semelhante ao cliente, irá enviar para o servidor, os dados que pretende alterar para cada participante, aquando da passagem num sensor.

2.3 Compilação e Execução

Para uma correta compilação das aplicações, não utilizando um IDE, pode ser obtida através do seguinte comando (ou equivalente), executado no terminal, uma vez na directoria de cada aplicação:

```
mvn clean compile package
```

Por sua vez, para a execução das mesmas, podem ser executadas de duas formas distintas, sempre em terminais distintos para cada módulo, nomeadamente:

```
[1] - Utilizando o Maven
mvn spring-boot:run
```

```
[2] - Através do ficheiro .jar
java -jar target/RunningEvents-Client-1.0.SNAPSHOT.jar
java -jar target/RunningEvents-Sensor-1.0.SNAPSHOT.jar
java -jar target/RunningEvents-Server-1.0.SNAPSHOT.jar
```

3 Tecnologias Utilizadas

Para a realização deste trabalho prático, foram utilizadas as seguintes tecnologias:

- Java 16;
- Framework Spring boot;
- JSON Objects;
- Métodos HTML POST e GET;
- NetBeans 12.5 (IDE);

4 Conclusão

Com a realização do trabalho prático descrito, foi possível adquirir de uma forma mais consolidada toda a matéria lecionada nas aulas, tanto práticas como teóricas. Ao longo do trabalho, foram encaradas algumas diversidades, nomeadamente implementar corretamente a comunicação entre os diferentes módulos, especificamente os controladores *rest server-side*.
As funcionalidades de valorização superior foram também analisadas e interpretadas corretamente de como seriam implementadas nas diversas aplicações.
Uma opção deveria ter sido implementada no cliente, de modo a ativar e/ou desativar o "serviço"de subscrição às notificações e, sempre que este valor fosse alterado, comunicar com o *REST API* informando a alteração deste decisão. Enquanto o cliente pretender receber estas notificações, a cada alteração da coluna 'participant_location', o servidor comunicaria com o módulo de sensor e posteriormente, este enviaria então uma resposta ao cliente.
Quanto à implementação de segurança de verificação das mensagens através de código MAC, seria algo mais complexo a implementar. Portanto, devido à complexidade que o trabalho requereu, não foi então possível a implementação destas funcionalidades.
No decorrer do desenvolvimento deste trabalho prático, foram aprimorados os conhecimentos nas distintas tecnologias utilizadas, bem como, o interesse pelas mesmas.