2021/2022
João Rouxinol Nº44451
André Rato Nº45517

Running Events – v2.1

Introdução

No âmbito da UC de **Sistemas Distribuídos**, foi proposta a realização de um trabalho prático que consiste num serviço de registo de eventos e participantes em vários tipos de corridas. Foi pedida a implementação das seguintes componentes:

- **Aplicação Cliente**, que deve conseguir realizar:
    1. Inscrições de participantes;
    2. Registo de eventos;
    3. Consultas de eventos;
    4. Consultas de resultados.

- **Aplicação Sensores**, que deve conseguir realizar:
    1. Transmissão de leituras de chips.

- **Aplicação Servidor** que deve ser responsável por:
    1. Funcionalidades;
    2. Armazenamento.

É pedido que o armazenamento persistente deve ser feito através de **BD** em **Postgres**, e que haja abstração dos detalhes de comunicação e diferenças de plataforma.

É também dito que pode existir valorização superior caso o trabalho cumpra com as seguintes funcionalidades:

- **Redundância no backend/servidor**, para reforçar a disponibilidade do serviço;
- **Mecanismo publish/subscribe** onde a aplicação cliente pode pedir para ser notificada de passagens num ponto intermédio de um evento/corrida;
- **Segurança das mensagens sensor** – servidor, por forma a evitar adulteração dos dados (MAC).

Desenvolvimento e Solução Encontrada

De modo a desenvolver um serviço que cumpra todas as funcionalidades pedidas foi criado um programa na linguagem **Java** e foi utilizado **REST**, com a **framework Jersey**.

Classes:

As Classes criadas foram as seguintes:

- **ChipClient** – Aplicação sensores, onde são transmitidas leituras dos sensores;
- **Client** – Aplicação cliente, onde o cliente pode interagir com o serviço;
- **Chip** – Classe que representa uma leitura do chip num certo ponto;
- **Event** – Classe que representa um evento;
- **MainAppServer** - Classe que representa o servidor do serviço;

- **Participant** – Classe que representa um participante do serviço num evento;
- **PostgresConnect** – Classe responsável pela conexão a uma BD Postgres;
- **QueryManager** – Classe responsável por interações com a BD. Cada objeto QueryManager é responsável por consultas e inserções na BD;
- **ServerResource** – Classe responsável pelo tratamento e atendimento dos pedidos das aplicações cliente e sensores.

Métodos

Os métodos utilizados no serviço foram os seguintes:

Classe ChipClient

- **menu()** - Imprime o menu de opções e lê a opção escolhida pelo utilizador;
- **convertJsonStringToJsonArray()** - Converte a String passada como argumento para um JSON array;
- **registerTime()** – Faz um pedido POST ao servidor de modo a inserir um novo tempo para um sensor numa localização;
- **main()** – Responsável pela leitura dos inputs do cliente para interação com o servidor (que neste caso são as leituras dos sensores.

Classe Client

- **menu()** - Imprime o menu de opções e lê a opção escolhida pelo utilizador;
- **convertJsonStringToJsonArray()** - Converte a String passada como argumento para um JSON array;
- **postEvent()** – Faz um pedido POST ao servidor de modo a registar um novo evento com o Nome, tipo e data passados como argumento;
- **getEvents()** – faz um pedido GET ao servidor de modo a receber uma lista com todos os eventos para uma certa data passada como argumento;
- **getFileteredEvents()** - faz um pedido GET ao servidor de modo a receber uma lista com todos os eventos com um certo filtro passado como argumento (eventos futuros, passados, atuais ou passados e atuais;
- **registerParticipant()** – faz um pedido POST ao servidor de modo a registar um novo participante num evento, devolvendo depois o número de dorsal correspondente ao participante (caso o registo tenha sucesso);
- **getParticipants()** – Faz um pedido GET ao servidor de modo a receber a lista de participantes num evento passado como argumento;
- **getNumberOfParticipantsAtPoint()** – Faz um pedido GET ao servidor de modo a receber o numero de participantes que já tenham ultrapassado um certo ponto (que pode ser START, P1, P2, P3, FINISH), num evento.;
- **getClassification()** – Faz um pedido GET ao servidor de modo a receber a lista de classificações num ponto de um evento;
- **findParticipant()** – procura num JSONArray o participante com o chipID correspondente ao passado como argumento e retorna o mesmo (se for encontrado);
- **eventExists()** – procura num JSONArray o evento com o nome correspondente ao passado como argumento e retorna o **true** se for encontrado (ou false se não for);
- **decodeTier()** – usado para devolver o Escalão correspondente a um tierCode;
- **main()** - Responsável pela leitura dos inputs do cliente para interação com o servidor.

Classe Chip

- Classe contem apenas Construtor, Setters e Getters, responsáveis por definir e retornar o valor das suas 3 variáveis de classe, que são:
    1. **chipID** – O ID do chip;
    2. **section** – o ponto onde foi feita a leitura;
    3. **time** – o tempo da leitura.

Classe Event

- Classe contem apenas Construtor, Setters e Getters, responsáveis por definir e retornar o valor das suas 3 variáveis de classe, que são:
    1. **name** – o nome do evento;
    2. **type** – o tipo do evento;
    3. **date** – a data em que o evento vai ocorrer.

Classe MainAppServer

- **getBaseURI()** – Devolve o URI onde vai ficar o serviço;
- **startServer()** – Cria um servidor Http Grizzly com serviço REST.

Classe Participant

- Classe contem apenas Construtor, Setters e Getters, responsáveis por definir e retornar o valor das suas 6 variáveis de classe, que são:
    1. **name** – o nome do participante;
    2. **dorsal** – o número de dorsal associado ao participante nesta corrida;
    3. **genre** – o gênero do participante (m/f);
    4. **eventName** – o nome do evento em que o participante está inscrito;
    5. **tier** – o gênero do participante;
    6. **chipId** – o ID do chip associado ao participante neste evento.

Classe PostgresConnect

- **connect()** - Estabelece uma conexão à BD;
- **disconnect()** - Termina a conexão à BD;
- **getStatement()** - Devolve o Statement, para ser usado para criar as Queries Postgres.

Classe QueryManager

- **getFilteredEvents()**- Faz uma pesquisa na BD de eventos com o filtro passado como argumento**;**
- **getLastChipID() –** Faz uma pesquisa na BD e devolve o ultimo ID de chip atribuído**;**
- **getLastDorsal()-** Faz uma pesquisa na BD e devolve o ultimo dorsal atribuído no evento passado como argumento**;**
- **setupDataBase()-** Cria um objeto PostgresConnect pronto para aceder à BD**;**

- **closeDBConnection()-** Termina a ligação com a BD**;**
- **createEvent()-** Faz uma inserção na BD com o evento passado como argumento**;**
- **searchEvents()-** Faz uma pesquisa na BD de eventos realizados na data passada como argumento**;**
- **registerParticipant()-** Regista o participante passado como argumento num evento**;**
- **getParticipants()-** Faz uma pesquisa na BD pela lista de participantes de um evento**;**
- **registerTime()-** Regista o tempo lido por um chip num evento numa certa secção**;**
- **getLocationsByChipId()-** Faz uma pesquisa na BD pela lista de pontos por onde um chip já passou**;**
- **getChipIdByCurrentDate()-** Faz uma pesquisa na BD para obter a lista de chipid registados para uma determinada nada**;**
- **getEventByChipId()-** Faz uma pesquisa na BD para obter o evento no qual um chipid está registado**;**
- **getNumberOfParticipantsAtPoint()-** Faz uma pesquisa na BD para obter o numero de participantes que já passaram uma secção num evento**;**
- **getClassification()-** Faz uma pesquisa na BD pela lista de classificações numa secção de um evento**.**

Classe ServerResource

- **getEvents() –** retorna a lista de eventos com a data e filtros passadas como argumento, depois da pesquisa na BD ser feita por um objeto QueryManager;
- **postEvent()** – envia um pedido à classe QueryManager para a criação de um evento;
- **getParticipants()** – retorna a lista de participantes num evento;
- **getClassification()** – retorna uma lista com as classificações de um evento num certo ponto do evento;
- **getNumberOfParticipants()** – retorna o numero de participantes que já passaram num certo ponto do evento.
- **registerParticipant() –** envia um pedido à classe QueryManager para o registo de um novo participante;
- **registerTime() –** envia um pedido à classe QueryManager para o registo de uma leitura de um sensor.

Tabelas BD

Para a representação deste sistema, foi utilizada uma BD PostGres, nela foram criadas as seguintes tabelas:

- events – Representa um evento a realizar/já realizado. Inclui o nome do evento, o tipo do evento e a data do mesmo.
- registrations – Representa uma inscrição num evento. Inclui o nome do participante bem como o género e escalão do mesmo, o nome do evento, o número de dorsal e chipID associados à inscrição
- times – Representa um tempo registado numa secção. Inclui o nome do evento, o chipID que fez a leitura, a locationID referente à secção bem como o timestamp da leitura.

As Queries utilizadas para a criação das tabelas na BD para o correto funcionamento do serviço são as seguintes:

Eventos

```
CREATE TABLE events
(
    eventName varchar(255),
    type      varchar(255),
    date      date,
        PRIMARY KEY (eventName)
);
```

Inscrições

```
CREATE TABLE registrations
(
    participantName varchar(255),
    eventName       varchar(225) REFERENCES events,
    genre           char(1),
    tier            varchar(20),
    dorsal          int,
    chipID          int PRIMARY KEY
);
```

Tempos

```
CREATE TABLE times
(
    eventName  varchar(255) references events,
    chipID     int references registrations,
    locationID varchar(10),
    timestamp  TIMESTAMP,
        PRIMARY KEY (locationID, chipID)
);
```

Instruções de Utilização

De modo a utilizar este serviço é necessário fazer os seguintes passos:

1. Criar uma BD **Postgres** com as tabelas necessárias (descritas acima);
2. Compilar o projeto;
    - Feito executando o comando **mvn compile** num terminal na pasta base do projeto;
3. Alterar, no ficheiro **config.properties**, os valores de:
    - db para o nome da base de dados;
    - host para o endereço onde está a base de dados;
    - user para o nome de utilizador;
    - password para a password de acesso;
    - baseuri para o uri base do serviço.
4. Executar a classe **MainAppServer**;
5. Executar um dos cliente:
    - **Client** caso queira executar a aplicação cliente;
    - **ChipClient** caso queira executar a aplicação sensores;

Conclusões Finais

Com a conclusão deste trabalho foi possível:

- Consolidar os conteúdos aprendidos ao longo do semestre sobre arquiteturas distribuídas;
- Perceber e trabalhar com **API REST**.

Apreciação Crítica

O trabalho entregue cumpre com as seguintes funcionalidades:

- **API REST** para a aplicação sensores;
- Cliente realiza todas as operações pedidas;

As funcionalidades pedidas não implementadas são as seguintes:

- redundância no **backend/servidor**, para reforçar a disponibilidade do serviço;
- mecanismo **publish/subscribe** onde a aplicação cliente pode pedir para ser notificada de passagens num ponto intermédio de um evento/corrida;
- segurança das mensagens sensor – servidor, por forma a evitar adulteração dos dados.