INSTITUTO SUPERIOR TÉCNICO

# ForkExec – T02

## Entrega 2 – Guião de Demonstração


| Nome | Número |
|---|---|
| Catarina Pedreira | 87524 |
| Miguel Coelho | 87687 |
| Ricardo Silva | 87700 |


Git Repository: https://github.com/tecnico-distsys/T02-ForkExec

# Instalação e configuração (em Ubuntu):

1. **Fazer clone do repositório**
    a. git clone https://github.com/tecnico-distsys/T02-ForkExec
    b. cd T02-ForkExec

2. **Instalar as dependências e compilar o projecto**
    a. mvn clean install -DskipTests compile

3. **Lançar os servidores de Hub, Restaurant e Points**
    a. lançar novo terminal ( ir para a pasta root do projecto)
        i. cd rst-ws
        ii. mvn exec:java
    b. lançar novo terminal ( ir para a pasta root do projecto)
        i. cd pts-ws
        ii. mvn exec:java
    c. lançar novo terminal ( ir para a pasta root do projecto)
        i. cd pts-ws
        ii. mvn exec:java -Dws.i=2
    d. lançar novo terminal ( ir para a pasta root do projecto)
        i. cd pts-ws
        ii. mvn exec:java -Dws.i=3
    e. lançar novo terminal ( ir para a pasta root do projecto)
        i. cd ../hub-ws
        ii. mvn exec:java

4. **Verificar que não existe problemas**
    a. mvn verify

# Cenário F1

## Passos para executar:

1. **Correr o teste para o F1**
    a. cd hub-ws-cli
    b. mvn -Dtest=F1Scenario test
2. **Verificar o seguinte output:**

```
Definir o default inicial balance para: 100

Criada conta de email: testtest@test.com

Conta criada com um balanço de: 100

É feito um pedido de carregar a conta com: 50 euros (5500 pontos).

Novo saldo da conta: 5600

Exiting.

Process finished with exit code 0
```

### Descrição do Cenário F1

```
client.ctrlInitUserPoints(100);
System.out.println("Definir o default inicial balance para:" + 100);

int amount = client.accountBalance(VALID_EMAIL);
System.out.println("Criar conta de email com default balance:"  + amount );

Assert.assertEquals(amount, 100);
System.out.println("Conta criada com um balanço de:" + amount);
```

Um PointsClient estabelece uma ligação com o Hub. De seguida activa a sua conta através da função loadBalance e retorna o saldo da sua conta.
O Front End chama, assincronamente, a operação de leitura (no nosso caso, a função pointsBalance) para todos os N gestores de réplicas do servidor de Points disponíveis naquele momento - três no caso do nosso projeto. A partir daqui, é chamada a operação de leitura de cada gestor de réplica. Esta operação recebe um userEmail e retorna um tuplo <Tag, Saldo>.

```
int add = 50;
client.loadAccount(VALID_EMAIL, add, VALID_FAKE_CC_NUMBER);
System.out.println("É feito um pedido de carregar a conta com:" + add);

amount = client.accountBalance(VALID_EMAIL);
Assert.assertEquals(amount, DEFAULT_INITIAL_BALANCE + add);
System.out.println("Novo saldo da conta:" + amount);
```

Os três servidores estão disponíveis para responder. No entanto, quando o Front End obtém respostas da maioria (no nosso caso, dois servidores), prossegue e devolve o valor que acabou de ler, garantido estar consistente.

# Cenário F2

## Passos para executar:

1. **Correr o teste para o F2**
    a. cd hub-ws-cli
    b. mvn -Dtest=F2Scenario test

2. **Desligar o terceiro servidor manualmente**
    a. Selecionar o terminal e clicar na tecla enter
3. **Verificar o seguinte output:**

```
Definir o default inicial balance para: 100

Criada conta de email: testtest@test.com

Conta criada com um balanço de: 100

É feito um pedido de carregar a conta com: 50 euros (5500 pontos).

Novo saldo da conta: 5600

Desligue um dos gestores de réplica do servidor Pontos

É feito um pedido de carregar a conta com: 30 euros (3300 pontos).

Novo saldo da conta: 8900

Exiting.

Process finished with exit code 0
```

**Descrição do Cenário F2**

Um PointsClient começa por efetuar os mesmos passos do que o PointsClient do cenário 1.
No entanto, poucos segundos depois de as operações de escrita terem sido chamadas para cada gestor de réplica, o terceiro servidor falha (no nosso teste utilizamos a função de sleep para dar tempo para desligar um servidor manualmente)

```
      try {
            Thread.sleep(20000);
        } catch (InterruptedException ie) {
            ie.printStackTrace();
        }
```

Como o Front End apenas precisa da resposta de uma maioria, espera apenas pela resposta de dois (os outros dois servidores) e prossegue, sem se ter sequer apercebido que o terceiro servidor falhou.

De seguida efetua mais algumas operações para verificar que o código continua consistente:

```
        add = 30;
        pointsAdded= 3300;
        client.loadAccount(VALID_EMAIL, add, VALID_FAKE_CC_NUMBER);
        System.out.println("É feito um pedido de carregar a conta com: " + add + "
euros (3300 pontos).");
        System.out.println();

        amount = client.accountBalance(VALID_EMAIL);
        Assert.assertEquals(amount, DEFAULT_INITIAL_BALANCE + 5500 + pointsAdded);
        System.out.println("Novo saldo da conta: " + amount);
        System.out.println();
        System.out.println("Exiting.");
```

Quando o terceiro servidor volta a estar ativo, atualiza os seus valores sem necessitar de intervenção do Front End.