



# **Arquiteturas de Software**

## **PA3 (Recurso)**

Ricardo Madureira nº:104624

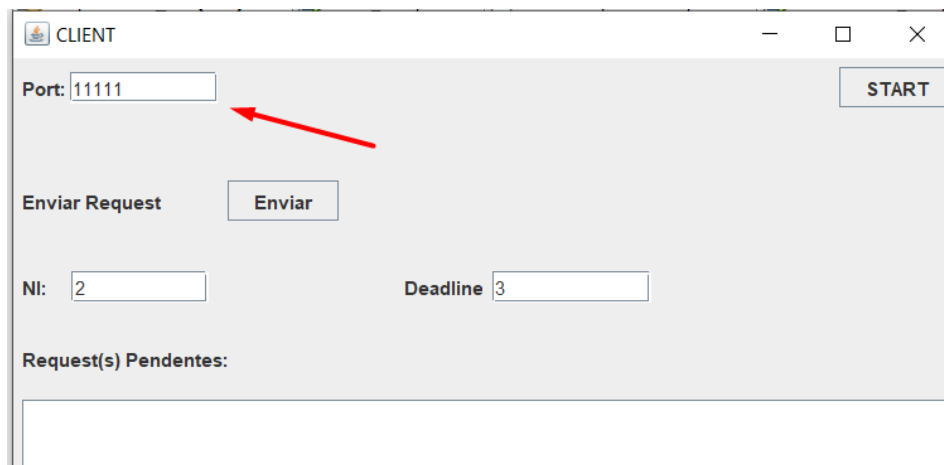
Diogo Azevedo nº:104654

# Tática de arquitetura implementada

## Implementação do Cliente

Para a realização das funções do cliente, onde este será responsável, numa parte inicial, em enviar requests para uma entidade responsável para controlar e fazer a gestão dos requests. Esta entidade ira ser o LoadBalancer.

Para realizar estas trocas de informação realizamos a conexão entre estas duas entidades através de JavaSockets com uma conexão TCP/IP, onde permitimos que o utilizador possa por o numero da porta de ligação que mais desejar.



*Figure 1 - Inserir porta para ligação cliente - LoadBalancer*

De seguida temos um botão onde será responsável por enviar requests ao nosso LoadBalancer, dentro deste botão realizamos a inicialização e execução do nosso Thread “ThreadRequestClient”.

No nosso Thread “ThreadRequestClient”, realizamos a conexão através dos Sockets e construímos o formato dos nossos requests tal como pedido no enunciado.

```
String str_forrequest = "" + ser + "|" + "" + requeste_id + "|00" + "|01" +  
    "|" + "" + ni.getText() + "|" + "00" + "|" + deadline;
```

Posteriormente, quando o tratamento dos requests for realizada, iremos receber na entidade Cliente esses requests e assim mostrar o seu resultado final.

Para realizarmos a ligação mandamos uma mensagem ao nosso LoadBalancer para ele saber que temos um cliente vivo.

```
String ms = "cliente";  
infromClient2.writeUTF(ms);  
infromClient2.flush();  
ser = infromClient.readUTF(); //ID
```

Figure 2 - Mensagem Cliente-LoadBalancer

Em que o LoadBalancer irá receber e verificar esta mensagem.

```
,  
if ("cliente".equals(mensagem_client)) {  
    System.out.println("enviar id para client " + id_client);  
    try {  
        data_outforclient.writeUTF(String.valueOf(id_client));  
    } catch (IOException ex) {  
        Logger.getLogger(LoadBalancer_GUI.class.getName()).log(Level.SEVERE, null, ex);  
    }  
    all_clientessocket_conectados.put(id_client, s_forclient);  
    id_client++;  
}
```

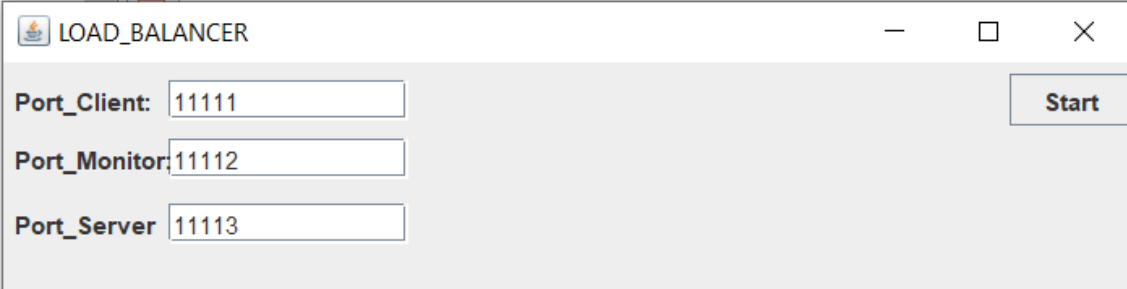
Figure 3 - Confirmação da conexão Cliente-LoadBalancer

## Implementação LoadBalancer

De seguida no nosso LoadBalancer, iremos realizar a implementações dos vários aspetos do sistema, onde esta entidade será a responsável pelo controlo dos requests que serão enviadas para o servidor.

Para realizar estas trocas de informação realizamos a conexão entre as entidades através de JavaSockets com uma conexão TCP/IP onde permitimos que o utilizador possa por o numero da porta de ligação que mais desejar.

Iremos ter três Threads que serão responsáveis por cada entidade no nosso sistema.

A screenshot of a Java Swing window titled "LOAD\_BALANCER". The window has a standard title bar with minimize, maximize, and close buttons. Inside the window, there are three text input fields. The first is labeled "Port\_Client:" and contains the value "11111". The second is labeled "Port\_Monitor:" and contains the value "11112". The third is labeled "Port\_Server:" and contains the value "11113". To the right of these fields is a button labeled "Start".

*Figure 4 - portas referentes ao cliente, monitor e servidor*

Teremos mais um Thread(ThreadLoadBalancer) que será responsável por enviar os requests através do cliente para serem processados para o servidor e outro Thread (Receber\_request) que ira receber os requests finais do servidor e enviar de volta para os clientes.

Para a entidade Clientes e Servidores, como foi mencionada anteriormente, aqui realizamos a comparação das mensagens vindas da entidade Cliente e Servidor para verificar se existe clientes/servidores vivos.

## Implementação Monitor

Esta entidade, será responsável por mostrar o estado dos servidores e o processamento que está a ocorrer em cada servidor.

Para realizar estas trocas de informação realizamos a conexão entre as entidades através de `JavaSockets` com conexão `TCP/IP` onde permitimos que o utilizador possa por o número da porta de ligação que mais desejar.

De seguida, realizamos a conexão do nosso Monitor com a nossa entidade Servidor para assim termos acesso às informações provenientes no servidor para realizarmos os objetivos mencionados anteriormente

Para tal, iremos receber uma mensagem vinda do servidor a dizer que este está a tentar se conectar com ele e realizamos a comparação da mensagem para assim estabelecermos uma conexão que nos irá permitir receber informação proveniente do servidor.

De seguida, para realizarmos os objetivos estabelecidos inicialmente para esta entidade, implementamos um Thread, denominado de “ThreadMonitor”, onde iremos explicar mais detalhadamente o seu funcionamento.

### **ThreadMonitor**

Neste Thread, inicialmente começamos por arranjar uma forma de ver o conteúdo que cada servidor está a processar e mostrar na nossa GUI. Também mostramos o servidor que está ligado.

## Implementação do Servidor

Nesta entidade o nosso objetivo é tratar os requests que recebemos provenientes no nosso LoadBalancer e voltar a enviar para o LoadBalancer com as características pedidas no enunciado.

Para realizar estas trocas de informação realizamos a conexão entre as entidades através de JavaSockets com conexão TCP/IP, onde permitimos que o utilizador possa por o número da porta de ligação que mais desejar.

Em primeiro começamos por estabelecer conexão com o nosso LoadBalancer com a mesma logica nas entidades anteriormente referidas, mandamos uma mensagem ao nosso LoadBalancer para ele saber que temos servidores vivos.

```
String msg_forload = "servidor";
try {
    dataenviar_forload.writeUTF(msg_forload);
    dataenviar_forload.flush();
} catch (IOException ex) {
    Logger.getLogger(Server_GUI.class.getName()).log(Level.SEVERE, null, ex);
}
```

Figure 5- Mensagem Servidor - LoadBalancer

Com este código enviamos uma mensagem do servidor para o LoadBalancer.

```
if ("servidor".equals(mensagem_servidor)) {
    System.out.println("enviar id para servidor " + id_servidor);
    try {
        data_outforservidor.writeUTF(String.valueOf(id_servidor));
    } catch (IOException ex) {
        Logger.getLogger(LoadBalancer_GUI.class.getName()).log(Level.SEVERE, null, ex);
    }
    allServerSocketsConnected.put(id_servidor, s_forServer);
    allRequestsOnEachServer.put(id_servidor, new ArrayList<String>());
    id_servidor++;
}
```

Figure 6 - Confirmação LoadBalancer - Servidor

No LoadBalancer recebemos esta mensagem para identificar o servidor.

De seguida realizamos uma implementação para fazer com que cada servidor processe três informações em simultâneo e ter uma queue com o máximo de 2 posições para processar mais 2 requests, caso ultrapasse estes numero rejeitamos os restantes requests.

Para implementar o processamento em simultâneo, o que fazemos foi ter uma variável que ira contar o número de requests, e uma HashMap que recebe esta contagem e os

requests inseridos, enquanto esta contagem for menor que três incrementamos o numero de requests e inserimos o numero de requests e o requests associado na nosso HashMap e posteriormente mostramos este resultado na GUI.

Também damos “parse” do deadline de cada request.

O nosso HashMap vai ser nested, em que na primeira key vai conter a contagem do request, e na o value irá ter um par key:value.

Este segundo par key:value criamos um dataholder, que irá guardar o nosso request e o deadline dele, assim o HashMap irá conter:

{0 : “request”:”deadline”, 1 : “request”:”deadline”, etc....}.

De seguida temos um Thread (ThreadServer) que será responsável por processar os requests recebidos.

### ThreadServer

Neste Thread, onde é passado como parâmetro importante o HashMap contendo os requests, e iremos realizar o seu processamento consoante o numero “Niter” fornecido pelo utilizador no lado do Cliente. Para tal, realizamos o parsing da string recebida e retiramos o numero que corresponde ao Niter e definimos o nosso PI como uma string. Se o nosso Niter que recebemos for menor que 14 (ou seja, menor que 14 pois é as casas decimais do nosso PI) realizamos a substring corresponde ao nosso PI, ou seja, deslocamos a vírgula consoante o número de Niter que recebemos. Feito este processamento, temos que ir de novo á string que processamos e mudarmos o estado desta para “02”.

Feito isto, multiplicamos o nosso Niter pelos 5 segundos definidos no enunciado e enviamos o nosso requests final.

```
String pi = "3.1415926589793";
if (niter < 14) {
    str_processed = pi.substring(0, 2 + niter);
    //System.out.println("str_processed" + str_processed);
}

StringBuilder processed = new StringBuilder();
processed.append(process[0]).append("|").append(process[1]).append("|").append(id_servidor).append("|")
    .append("02").append("|").append(process[4]).append("|").append(str_processed).append("|").append(deadline)
    .append("|");
System.out.println("process: " + processed.toString());

try {
    sleep(5000 * niter);
} catch (InterruptedException ex) {
    java.util.logging.Logger.getLogger(ThreadServer.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
}
```

Figure 7 - Processo de tratamento do request

Para enviar a informação de que os servidores estão online e o que estão a processar, como mencionamos anteriormente, enviamos para o Monitor uma mensagem para darmos sinal de que estamos vivos e enviar a informação dita anteriormente.

## UC realizadas

Não realizamos a UC1 (distribuir através de vários servidores), UC5(Shutdown do servidor), e a UC6(Múltiplos LoadBalancer) devido a falta de tempo de nossa parte.

Realizamos por completo as UC2(realizar os processos em threads individuais), UC3(3 threads processados, 2 em queue e o resto ser rejeitado) a UC4(earlier-deadline first) e a UC7(GUI)

## Como realizamos a UC4(earlier-deadline first):

```
public int min_deadile(HashMap<Integer, DataHolder.Data> controlar_dealine) {  
  
    List<Map.Entry<Integer, DataHolder.Data>> list = new ArrayList<>(controlar_dealine.entrySet());  
    //System.out.println("List -> " + list);  
    Collections.sort(list, Comparator.comparing(o -> o.getValue().value));  
  
    System.out.println("Menor -> " + list.stream().findFirst().get().getKey());  
    return list.stream().findFirst().get().getKey();  
}
```

Com este excerto de código nos comparamos os valores do deadline guardados no HashMap e enviamos o counter (posição desse request no HashMap) para o Thread que irá executar o tratamento do request, assim ele sabe qual request deve processar primeiro.



## Contribuição dos elementos do grupo

Apesar do trabalho ter sido muito dividido, ou seja, um elemento fazia uma coisa, mas ia ver o raciocínio noutra parte, onde as ideias se completavam, onde cada elemento se focou mais foi:

Ricardo Madureira: na parte dos Monitor e LoadBalancer, parte do relatório.

Diogo Azevedo: na parte do Clientes, Servidor e parte do relatório.

### **Percentagem de trabalho de cada elemento:**

Ricardo Madureira: 45%

Diogo Azevedo: 55%