Uma imagem com texto

Descrição gerada automaticamente

**Arquiteturas de Software**

**PA3**

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 onde permitimos que o utilizador possa por o numero da porta de ligação que mais desejar.

Para o LoadBalancer saber se existe clientes conectados, a estratégia que realizamos foi sempre que um cliente é criado, este envia uma mensagem ao LoadBalancer a dizer “Clientevivo!” e posteriormente no LoadBalancer este irá comparar as mensagens recebidas e caso sejam iguais, temos a informação que um cliente se conectou ao nosso sistema.

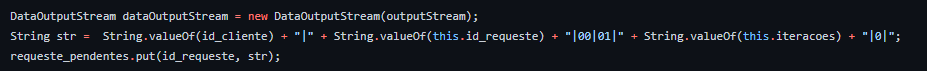
Uma imagem com texto, interior, laranja, escuro

Descrição gerada automaticamente



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 “Send\_Request”.

No nosso Thread “Send\_Request”, realizamos a conexão através dos sockets e construímos o formato dos nossos requests tal como pedido no enunciado.

Mas como podemos ter vários Clientes a enviar requests, não podemos por tudo na mesma variável pois assim só um cliente é que iria enviar requests, para tal, a estratégia utilizada foi utilizar uma HashMap onde esta irá guardar os ids dos clientes vivos e os requests que cada cliente enviar, para assim vários clientes enviarem informação.

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

Ao receber os requests finais, o que fazemos é remover da nossa hashMap o processamento final de cada requests do campo value, e percorrer esses valore para mostrar na nossa GUI.

(PRINT CODIGO, MUDAR CODIGO)

## 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 as diversas entidades.

Para realizar estas trocas de informação realizamos a conexão entre as entidades através de JavaSockets 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.

Teremos mais um Thread(ThreadLB\_Request) que será responsável por enviar os requests para serem processados para os servidores e outro Thread (ThreadLB\_ReceberRequest) que ira receber os requests finais e enviar de volta para os clientes.

Para a entidade Clientes, como foi mencionada anteriormente, aqui realizamos a comparação das mensagens vindas da entidade Cliente para verificar se existe clientes vivos. Caso haja, passamos á implementação no nosso “ThreadLB\_Request”.

### ThreadLB\_Request

Neste Thread o nosso objetivo é receber a informação de quantos servidores estão vivos, através da informação que recebemos do nosso Monitor e fazer a seleção correta de como os requests vao ser enviados para cada servidor com uma distribuição justa.

Depois de sabermos a informação de quantos servidores estão vivos, iremos observar a informação que cada servidor está a processar, para sabermos a quantidade que cada servidor pode ter. Depois de sabermos isso, temos uma variável com o valor igual 5, pois cada servidor no total só pode processar no máximo cinco requests e enquanto a informação nos diversos servidores for menor que cinco, nos inserimos os requests nesses servidores. Quando cada servidor tiver cheio (ou seja, com cinco requests a serem executados) incrementamos uma variável, inicializada a zero, que será responsável por nos informar que os servidores estão cheios.

(PRINT CODIGO ThreadLB\_Request)

De seguida ao realizarmos a escolha para onde irá cada requests, enviamos os requestes vindos de cada cliente para os servidores que estão vivos.

(PRINT CODIGO ThreadLB\_Request)

### ThreadLB\_ReceberRequest

…

Para a entidade Monitor onde nos apresentamos a informação de quantos servidores estão online e a informação a ser processada em cada servidor, a ideia para saber se o Monitor se conectou ao nosso sistema é idêntica ao dos clientes, ou seja, iremos receber por parte do Monitor uma string a dizer “monitorvivo” e aqui no LoadBalancer iremos fazer a comparação para assim termos a informação que o Monitor esta operacional.

(PRINT CODIGO Logic\_Monitor)

## Implementação Monitor

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

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

Numa primeira fase e idêntico ao que mencionamos anteriormente, enviamos para o nosso LoadBalancer uma mensagem, “monitovivo” para assim dar-mos sinal ao nosso Load que o Monitor esta a conectar-se ao nosso sistema.

(PRINT Logic\_lb)

De seguida, realizamos a conexão do nosso Monitor com a nossa entidade Servidor para assim termos acesso ás 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 das mensagem para assim estabelecermos uma conexão que nos ira permitir recebr 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.

(PRINT Logic\_recived\_fromservers)

### ThreadMonitor

…