Integração de Sistemas 2020/2021

Relatório Projeto 3

Trabalho realizado por:

Hermínio Simões da Silva nº2015262754

Pedro Miguel Santos nº2017247870.

Conteúdo

Autoavaliação

a) Para esta meta conseguimos realizar tudo o que foi requisitado no enunciado, não deixando nenhuma tarefa por fazer.

b) Avaliação do grupo – 100%.

c) Hermínio Simões da Silva – Admin e métodos utilizados pelo mesmo no System Node.
Pedro Miguel Santos – User e métodos utilizados pelo mesmo no System Node.

d) Hermínio Simões da Silva – 100% Pedro Miguel Santos – 100%

e) Horas Hermínio Simões da Silva – 20h Pedro Miguel Santos – 20h

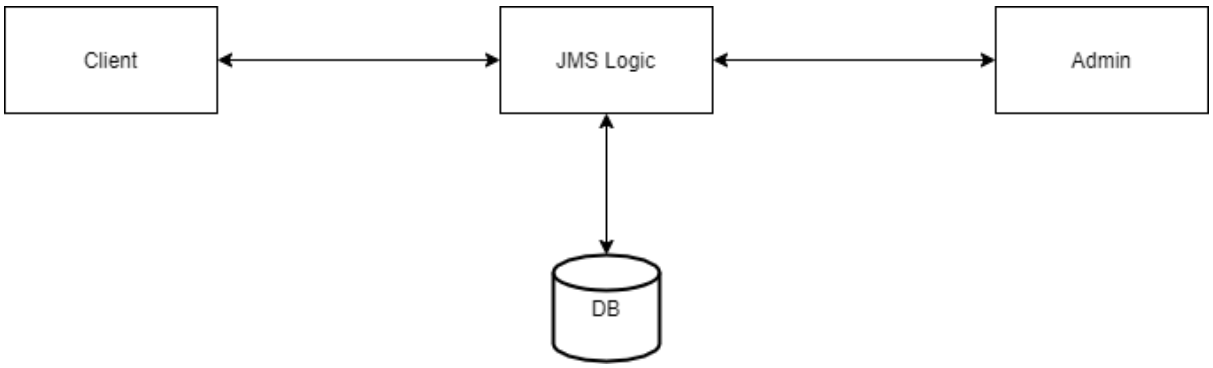
Introdução

Para este projeto foi-nos pedido que utilizássemos JMS para implementar através de uma troca de mensagens um sistema na qual um administrador e cliente coexistiam com um sistema intermédio para manipular uma base de dados de publicações. O administrador também consegue alterar uma tabela que contém os utilizadores.

Estrutura do projeto

Para realizar o pedido, começámos por implementar o System Node(no gráfico JMS Logic) que iria ter os métodos todos para realizar as operações, seja de login e registo sejam os pedidos que posteriormente poderão necessitar de intervenção do administrador e até os pedidos efetuados pelo administrador em si.

Para tal, necessitámos de criar uma ligação com uma base de dados que teria tabela de publicações e tabela com os utilizadores (clientes do sistema).



O Client e o System Node comunicam através de um Topic (playTopic), estes utilizam uma *temporary queue* para receber e responder mensagens, no caso do Client, este envia e recebe e no System Node recebe e envia. No caso do Admin, ocorre o mesmo, o Admin envia e recebe e o System Node recebe e envia.

Para a tabela de Users existente na base de dados, colocámos os parâmetros de "id", "username", "password" e um valor boolean para verificar se está ativo "active". Para a tabela de publicações, temos o "id", "título", "pubdate" e "type".

Para conseguirmos alterar os parâmetros destas tabelas, necessitámos de criar uma classe User e uma classe Publication.

Estrutura do construtor de publicações:

```java
@Entity
public class Publication implements Serializable
{
private static final long serialVersionUID = 1L;
// we use this generation type to match that of SQLWriteStudents
@OnDelete(action = OnDeleteAction.CASCADE)
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
private String name;
private String type;
private String pubdate;



public Publication(){}
public Publication(String name, String type, String pubdate)
{
    this.name = name;
    this.type = type;
    this.pubdate =pubdate;

}
```

Estrutura do construtor de User:

```java
@Entity
@Table(name = "user", schema = "public")
public class User {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
    private String username;
    private String password;
    private boolean active;

    public User() {}

    public User(String username, String password, boolean active) {
        this.setUsername(username);
        this.setPassword(password);
        this.setActive(active);
    }
```

Através destas estruturas, tanto de base de dados como classes de java, conseguimos utilizar as propriedades e métodos de JMS para comunicar com os clientes e o administrador de forma a alterar os dados e a apresentá-los corretamente.

System Node

Nesta etapa, o objetivo seria criar os métodos que iriam posteriormente ser utilizados tanto pelo administrados como pelo utilizador.

Aqui será onde os pedidos dos clientes serão recebidos e caso necessitem de aprovação do administrador, são guardados num ArrayList de tipo TextMessage (de forma a preservar os endereços de envio para mais tarde responder). Os pedidos que não necessitem de aprovação, são respondidos diretamente pelo System Node utilizando os métodos que foram criados previamente.

No System Node utilizamos a criação de threads para notificar todos os clientes quando existe uma alteração na tabela das publicações: quer seja adicionar publicações, remover publicações ou alterar o seu título, a sua data ou o seu tipo.

Client

Para o client exibimos um menu inicial para escolher se pretende realizar um registo ou um login. Após realizar um registo, terá de aguardar aprovação de administrador. Após realizar login, é-lhe apresentado um novo menu com todas as operações que pode realizar.

```
Choose an action:
 3-See All Publications
 4-Search a publication title
 5-Add New Publication
 6-Update Publication title
 7-Remove Publication
 20-Update Publication date
 21-Update Publication type
 0-Exit
```

Após ter escolhido uma opção, dependendo de qual seja, poderá ou não ter de inserir dados extra, como por exemplo, se pretender ver todas as publicações, apenas a escolha da opção é suficiente para tal, no entanto, se pretender encontrar uma publicação pelo seu título, terá de escolher a opção e posteriormente inserir o seu título. Quando a escolha e todos os parâmetros necessários para a realizar do lado do System Node estiverem preenchidos, é enviada uma mensagem para o System Node, que após ser validada é respondida. Por vezes, a opção poderá requerer uma aprovação do Admin, pelo que a resposta e o efeito da escolha não são visíveis imediatamente.

Para conseguirmos notificar todos os clientes de que uma publicação foi alterada, adicionada ou removida na base de dados, criámos uma classe “ClientThread” , onde no método “run” a thread vai estar sempre à escuta das notificações enviadas pelo System Node.

```java
public void run() {

    String msg = null;
    //          try (JMSContext context = connectionFactory.createContext("calmo", "Calmo@1997");)

    try (JMSContext context = connectionFactory.createContext("pedro", "pedro123.");){

        JMSConsumer mc = context.createConsumer(destination);
        while(true) {

            msg = mc.receiveBody(String.class);
            System.out.println("Notification from admin: "+msg);
        }


    }
        catch(JMSRuntimeException re) {

            re.printStackTrace();
        }

}
```

Para conseguir comunicar sem deixar o Client à espera de uma notificação por tempo indefinido, criámos um novo Topic “playCalmo”.

```java
public ClientThread() throws NamingException {
    this.connectionFactory = InitialContext.doLookup("jms/RemoteConnectionFactory");
    this.destination = InitialContext.doLookup("jms/topic/playCalmo");
}
```

Admin

À semelhança do User, o Admin também possui um menu onde poderá escolher o que pretende fazer:

```
Choose an action:
8-List all Users
9-List all Pending Tasks
10-Deactivate User
11-List All Publications
12- Publication Information
13-Activate User
0-Exit
```

Através das opções escolhidas, tal como no Client, poderá ter que inserir informação adicional, por exemplo, para listar os utilizadores todos, basta selecionar a opção “8”, no entanto, para desativar ou ativar um User ou visualizar informação de uma Publicação, terá de inserir dados que especifiquem qual o User ou Publicação em concreto.

À semelhança do caso de adicionar informação adicional, ao Listar as tarefas pendentes (“List all Pending Tasks”), é pedido ao administrador que responda a todas as tarefas com “y” ou “n” (sim ou não) para aprovar a sua execução, estas respostas deverão ser dadas com um espaço a separar(por exemplo :“y y”). Ao enviar este pedido para o System Node, serão feitos os métodos adequados aos pedidos que foram feitos ao administrador para responder ao cliente.

Conclusão

Em suma, este projeto permitiu-nos utilizar ferramentas que funcionam interligadas para atingir um objetivo definido. De diferentes formas e com diferentes métodos de JMS conseguimos executar funções com a mesma finalização de formas diferentes, o que nos permitiu ter um melhor conhecimento de como trabalhar com estas ferramentas.