FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Projeto 3

Ano Letivo 2020 / 2021

Departamento de Engenharia Informática

Integração de Sistemas
1º Semestre

Discentes

António Lopes – 2017262466 – uc2017262466@student.uc.pt

Lucas Ferreira – 2016243439 – uc2016243439@student.uc.pt

Índice

Parâmetros de avaliação

a. De um modo geral, cumprimos com todos os requisitos e conseguimos fazer todos os pressupostos pedidos no enunciado, que irão ser abordados nas páginas seguintes.

b.

Autoavaliação do grupo
90%

c.

	António Lopes	Lucas Ferreira
Cliente	✓	✓
Admin	✓	✖
MessageBean	✓	✓
AdminBean	✓	✖
JMS	✓	✓
Relatório	✓	✓

d.

António Lopes	Lucas Ferreira
90%	70%

e.

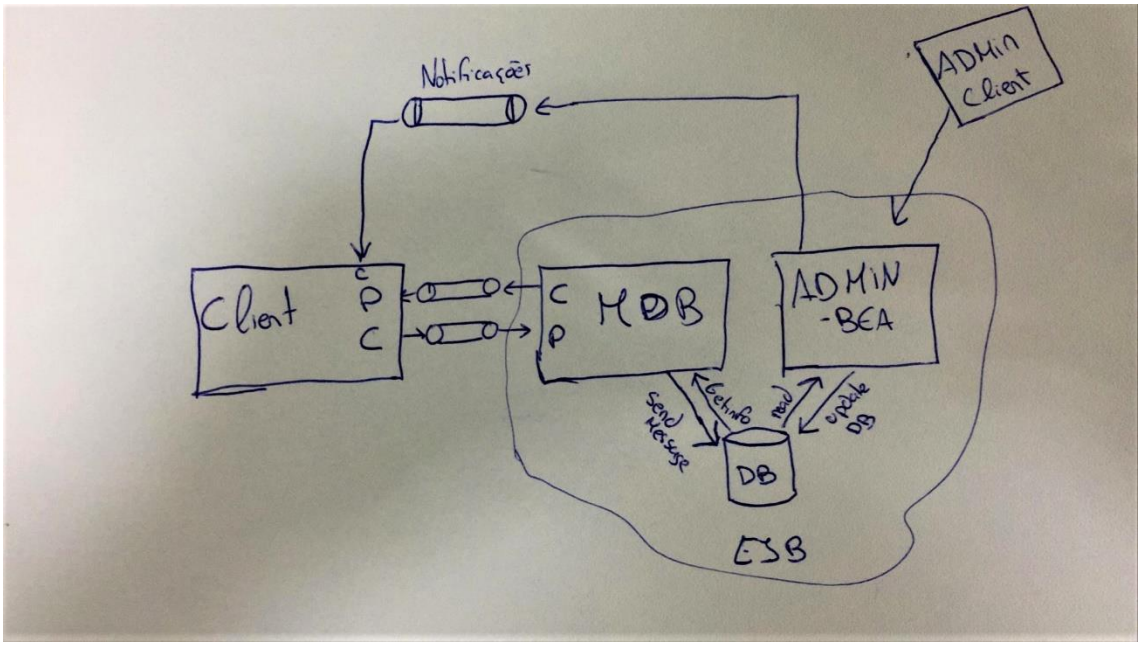
António Lopes	Lucas Ferreira
20h	14h

Nota Prévia

De acordo com o terceiro trabalho prático, solicitado a realizar na cadeia de Integração de Sistemas, vamos focar a nossa atenção nos principais objetivos deste projeto que são aprender, utilizar e criar um sistema que faça de uso do **Java Message Service**, comunicando, com a recorrência outros serviços, através de um sistema de mensagens. Para tal, de modo a estes serviços funcionarem corretamente, o JMS é acompanhado da **Java Persistence API** para acesso a uma base de dados onde se encontram todas as entidades intervenientes neste projeto, que neste caso serão administradores, investigadores, e aproveitando do projeto anterior, publicações dos investigadores.

Ainda, para permitir uma comunicação totalmente assíncrona, é feito o uso de **Message-Driven Beans** na arquitetura construída, que por sua vez são um tipo de enterprise bean, invocados num container **EJB** a partir do momento em que recebe uma mensagem de uma queue ou topic proveniente de uma receiver JMS. É ainda feito, com recurso ao **WildFly 18**, um servidor que nos permite manter e fazer as transações entre as entidades em tempo-real.
De forma a conseguir cobrir todas estas funcionalidades, serão explicados nos próximos tópicos, com a ajuda de alguns exemplos implementados.

Arquitetura



Para explicar a arquitetura utilizada, com o auxílio da imagem acima, começamos pelo Message-Driven Bean, que é o que mantém todas as transações entre o cliente, administrador e base de dados. Para tal, tanto no lado do cliente, como no do MDB são criados os producers e consumers do Java Messaging Service, de modo a conseguirem comunicar através de protocolos de mensagem e assim perfazer quaisquer ações pressupostas. Posto isto, temos a ligação entre o EJB do administrador e o MDB que comunicam entre si com o auxílio do JPA para fazer as alterações conforme necessário na base de dados. Este EJB é utilizado por um cliente Administrador que tratará de todos os pedidos do cliente através de uma consola, que por sua vez irá encaminhar os pedidos à base de dados para receber as suas tarefas pendentes, escolher o que fazer com elas e, posteriormente, direcionar as notificações através de uma fila de mensagens do tipo Publish-Subscribe, de modo a conseguir identificar corretamente o respetivo cliente.

Researcher

Começando pelo researcher, tendo em conta, que este necessitará tanto de receber, como enviar mensagens através da fila de mensagens, é criado um **Producer** e um **Consumer**, que num exemplo concreto, estes são usados para conseguir enviar pedidos ao sistema, tais como o registo, e posteriormente receber a resposta em tempo real ou de uma durable subscription (sendo entregue a mensagem de permissão após a tentativa de login, neste caso), caso o administrador não atenda ao seu pedido no momento em que está no sistema, respetivamente, como é mostrado nas figuras abaixo.

```
public static void main(String[] args)throws NamingException
{
    Client receiver = new Client();
    try(JMSContext context = receiver.connectionFactory.createContext("john","!1secret");)
    {
        while(receiver.denied) {
            Client sender = new Client();
            System.out.println("Welcome\n1-Register\n2-Login\n-1 Exit\n\nSelect Option:");
            Scanner sc = new Scanner(System.in);
            int option = Integer.parseInt(sc.nextLine());
            if(option == 1) {
                System.out.println("User:");
                String user = sc.nextLine();
                System.out.println("Password:");
                String password = sc.nextLine();
                String x = "REGISTER "+user+" "+password;
                if(nome == null || !nome.equals(user)) {
                    nome = user;
                    context.setClientID(nome);
                    String property = "user='";
                    property = property + nome+"'";
                    JMSConsumer consumer = context.createDurableConsumer((Topic)receiver.destinationReceiver,nome, property, true);
                    consumer.setMessageListener(receiver);
                }
                sender.send(x);
            }
```

Figura 1 - Exemplo da criação do registo

```
User:
Lucas
Password:
teste
Received in Temporary Queue: Register Done Waiting for Admin Approval
Welcome
1-Register
2-Login
-1 Exit

Select Option:
Got message: USER ACCEPTED BY ADMIN
```

Figura 2 - Registo de uma conta e posterior mensagem de aprovação, dada pelo administrador

```java
public Client() throws NamingException
{
    this.connectionFactory = InitialContext.doLookup("jms/RemoteConnectionFactory");
    this.destination = InitialContext.doLookup("jms/queue/playQueue");
    this.destinationReceiver = InitialContext.doLookup("jms/topic/playTopic");
    this.denied = true;
}

private void send(String text)
{
    try(JMSContext context = connectionFactory.createContext("john","!1secret");)
    {
        JMSProducer messageprod = context.createProducer();
        TextMessage msg = context.createTextMessage();
        Destination tmp = context.createTemporaryQueue();
        msg.setJMSReplyTo(tmp);
        msg.setText(text);
        messageprod.send(destination, msg);

        JMSConsumer cons = context.createConsumer(tmp);
        String receive = cons.receiveBody(String.class);
        System.out.println("Received in Temporary Queue: "+receive);
        String[] parser = receive.split(" ");
        if(parser[0].equals("LOGIN")) {
            status = Boolean.parseBoolean(parser[2]);
        }
    }catch(Exception re) {
        re.printStackTrace();
    }
}
```

Figura 3 - Inicialização do cliente e método de envio de mensagens através da queue temporária

No método **send** mostrado acima é onde são feitas as comunicações em tempo-real e onde são recebidas as respostas de acordo com as permissões do cliente. Para tal, sempre que é feito um pedido pela parte do cliente, é feita a verificação do status do mesmo, que poderá ser revogado por um administrador a qualquer momento, rejeitando qualquer tipo de pedido feito pelo cliente a partir do momento em que este status é alterado. A comunicação é feita entre o **JMS** e o **MDB** através de protocolos de mensagens, que a título de exemplo, novamente da figura anterior, esta mensagem é separada por um *parser* que nos permite identificar se o cliente tem as ditas permissões para executar pedido, sendo necessário o campo do status estar a *true* (p.e. **LOGIN Lucas true**).

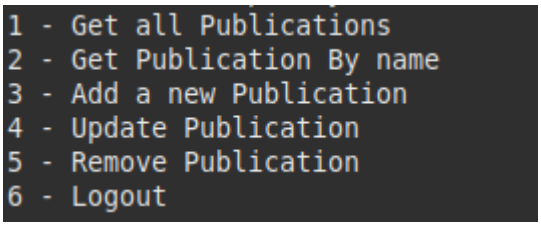


Figura 4 - Menu apresentado ao cliente após fazer login

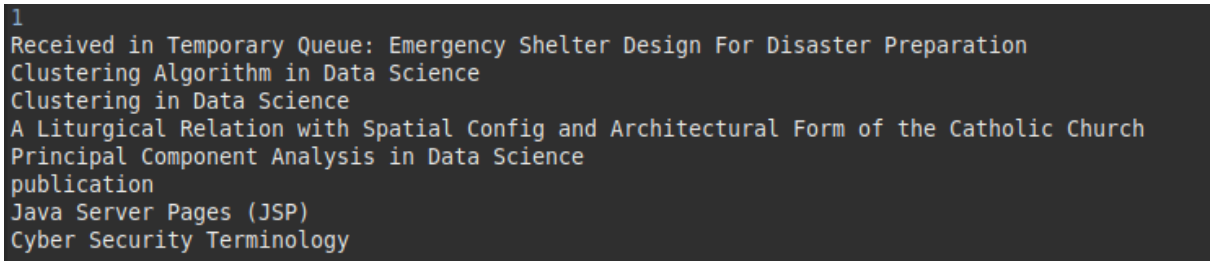


Figura 5 - Resultado da opção **Get all Publications**

Após fazer login, são apresentadas todas as opções permitidas ao cliente após ter sido aprovado pelo administrador. De modo a fazer uma pequena demo, sem explorar todas as funcionalidades extensivamente, será mostrado nas figuras abaixo o seguimento de adição de uma publicação e posterior aprovação pela parte do administrador.

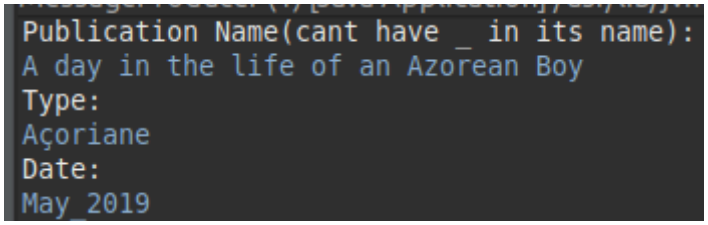


Figura 6 - Exemplo de adição de uma nova publicação por parte do cliente

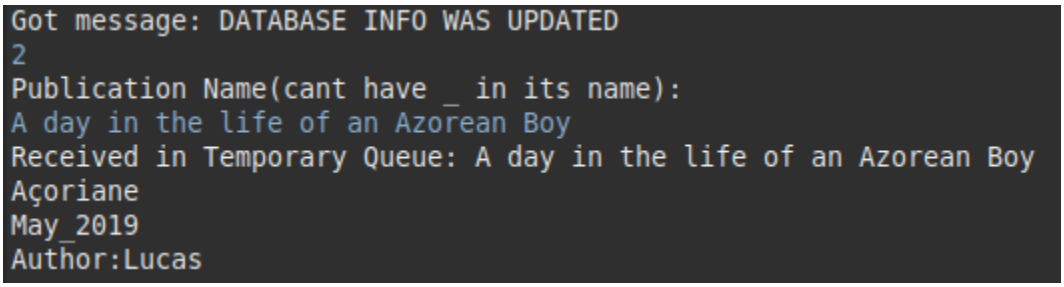


Figura 7 - Mensagem a informar que a base de dados foi atualizada e pesquisa da publicação criada

Administrator

Do lado do administrador é onde se dá a comunicação direta entre este, a base de dados e o respetivo bean. Para tal, este é o único que tem permissões para aprovar ou rejeitar quaisquer pedidos feitos por um cliente, através de uma consola. Ao fazer isto, este seleciona uma das opções do menu apresentado para executar operações, e a título de exemplo, segue abaixo a aprovação do registo de um cliente.

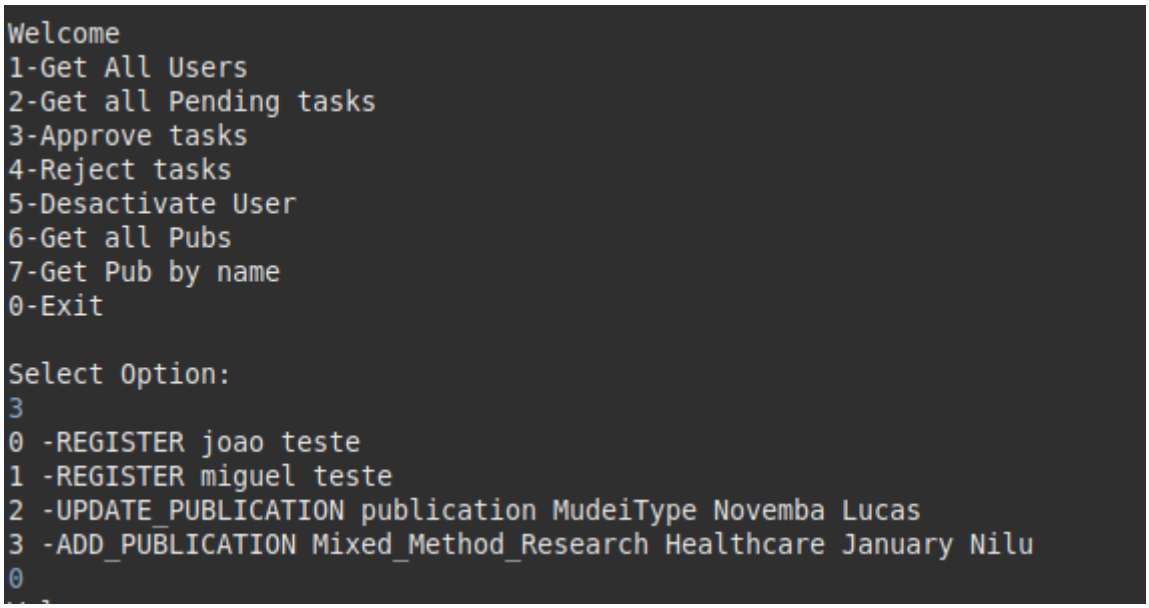


Figura 8 - Menu do administrador e respetiva aprovação de registo de um cliente

Após aprovar este registo, o bean trata de fazer as alterações na base de dados com o auxílio do JPA, persistindo o seu registo, e é mandada através da queue uma mensagem ao cliente a informar que este já foi aprovado, como é mostrado em figuras anteriores.

```java
public void ApproveMessages(String name) {
    String jpqlP = "SELECT DISTINCT p FROM AdminMessages p WHERE p.message = :message";

    TypedQuery<AdminMessages> typedQueryP = em.createQuery(jpqlP, AdminMessages.class);
    typedQueryP.setParameter("message", name);
    AdminMessages mylistP = typedQueryP.getSingleResult();

    List<Users> lista;

    String jpqlU = "SELECT DISTINCT r FROM Users r LEFT JOIN FETCH r.publications p";

    TypedQuery<Users> typedQueryU = em.createQuery(jpqlU, Users.class);
    List<Users> mylistU = typedQueryU.getResultList();

    lista = mylistU;

    String[] parser = mylistP.getMessage().split(" ");
    if(parser[0].equals("REGISTER")) {
        RegisterUser(parser[1],parser[2]);
        //send message to async queue
        try {
            sendMessages("USER ACCEPTED BY ADMIN",parser[1]);
        } catch (JMSException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    else {
```

Figura 9 - Comunicação do bean com o JPA e para aprovação do cliente

No excerto de código mostrado acima, é apresentado o método **ApproveMessages**, onde são feitas as queries para selecionar e mostrar quais as tarefas pendentes por aprovar ao administrador.

```java
private void sendMessages(String message, String user) throws JMSException {
    JMSProducer prod = context.createProducer();
    TextMessage reply =context.createTextMessage();
    reply.setText(message);

    reply.setStringProperty("user", user);

    prod.send(destination, reply);
    System.out.println("Sent reply to: "+destination + " "+reply.getStringProperty("user"));
}
```

Figura 10 - Método que cria o producer e trata de remeter a mensagem para o cliente em questão, identificado pelo username.

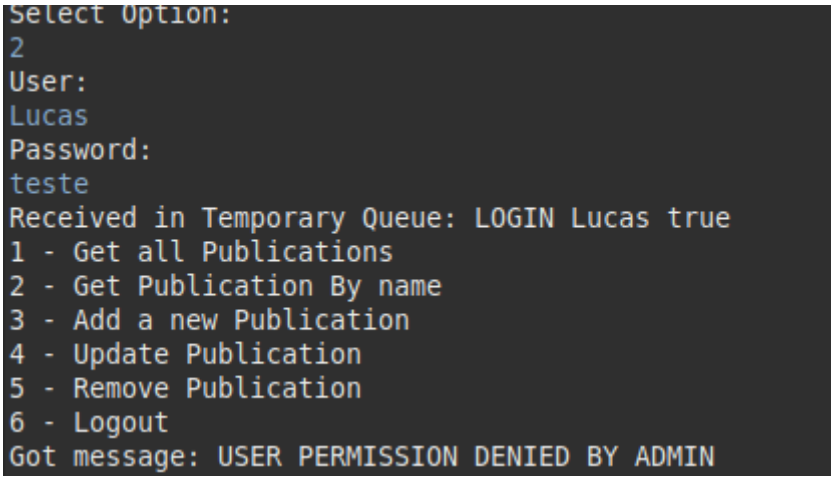


Figura 11 - Mensagem apresentada ao cliente após o administrador revogar permissões

Aquando da revogação das permissões da parte do administrador, o cliente recebe a mensagem descrita na figura acima, e após isto, a sessão em que se encontra é fechada entre ele a fila de mensagens, não executando mais nenhuma ação após isto.