FCTUC FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Projeto 2

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
Loader	✓	✓
SOAP Web Service (Researchers)	✓	✖
SOAP Web Service (Publications)	✓	✖
REST Web Service (Institutions)	✖	✓
Client application	✓	✓
Web front-end	✓	✓

d.

António Lopes	Lucas Ferreira
95%	80%

e.

António Lopes	Lucas Ferreira
37h	29h

JPAs e Classes

Para a parte A) do projeto, nomeadamente ProjetoJPA, foi necessário criar as classes **Researcher, Publication, Skills** e **Institution,** tanto como o estabelecimento de ligações entre elas. De modo a que sejam criadas as tabelas de ligações entre as classes é necessário indicar o tipo de relações entre elas. No caso do **Researcher**, é necessário indicar as relações **@ManyToOne** com as instituições, **@ManytoMany** com as ***Skills*** e ainda **@ManytoMany** com as publicações.

```java
import java.io.Serializable;

@Entity
public class Researcher implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Column(name = "name", nullable = false)
    private String name;
    @Column(name = "num_publications")
    private Long num_publications;
    @Column(name = "num_reads")
    private Long num_reads;
    @Column(name = "citations")
    private Long citations;

    @ManyToMany(mappedBy = "researchers", fetch = FetchType.LAZY)
    private List<Skill> skills;
    @ManyToOne()
    @JoinColumn(name="institution_name", referencedColumnName = "name")
    private Institution institution;
    @ManyToMany(mappedBy = "author_names", fetch = FetchType.EAGER)
    private List<Publication> publications;


    public Researcher() {}

    public Researcher(String name, Long num_publications, Long num_reads, Long citations) {
        this.name = name;
        this.num_publications = num_publications;
        this.num_reads = num_reads;
        this.citations = citations;
    }

    public String getName() {
        return this.name;
```

Figura 1 – Exemplo da implementação da classe Researcher

Para iniciarmos o preenchimento das Tabelas na Base de Dados, teremos de inicializar a ligação com o **EntityManager** para tal é necessário introduzir o nome da **Persistence**, que irá aceder ao **persistence.xml** e receberá as informações de onde se encontra a base de dados e permissões para o seu preenchimento.

```java
    {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("TestPersistence");
        List<Institution> mylistI = new ArrayList<Institution>();
        List<Researcher> mylistR = new ArrayList<Researcher>();
        List<Publication> mylistP = new ArrayList<Publication>();
        List<Skill> mylistS =  new ArrayList<Skill>();

        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();

        Scanner sc = new Scanner(System.in);
        int flag = 1;

        while(flag == 1) {
            System.out.println("SELECT OPTION :");
            System.out.println("1 - ADD_INSTITUIONS :");
            System.out.println("2 - ADD_RESEARCHER :");
            System.out.println("3 - ADD_PUBLICATIONS :");
            System.out.println("4 - ADD_SKILLS :");
            System.out.println("5 - ADD_INSTITUTION TO RESEARCHER :");
            System.out.println("6 - ADD SKILLS TO RESEARCHER :");
            System.out.println("7 - ADD PUBLICATIONS TO RESEARCHER :");
            System.out.println("0 - LEAVE :");
            System.out.println("SELECT OPTION :");
            int x = Integer.parseInt(sc.nextLine());
            switch(x) {
                case 0:
                    flag = 0;
                    break;
                case 1:
                    mylistI = GetInstitution.get();
                    break;
                case 2:
                    mylistR = GetResearchers.get();
                    break;
                case 3:
                    mylistP = GetPublication.get();
                    break;
                case 4:
                    mylistS = GetSkills.get();
                    break;
                case 5:
                    for(int i = 0 ; i < mylistR.size();i++) {
                        System.out.println(i+" "+mylistR.get(i).getName());
                    }
                    System.out.println("Choose Person:");
                    int a = Integer.parseInt(sc.nextLine());
                    for(int i = 0 ; i < mylistI.size();i++) {
                        System.out.println(i+" "+mylistI.get(i).getName());
                    }
                    System.out.println("Choose Department:");
                    int b = Integer.parseInt(sc.nextLine());
```

Figura 2 – Início para escrita na base de dados com hibernate

```xml
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
    <persistence-unit name="TestPersistence" transaction-type="RESOURCE_LOCAL" >

    <!-- use transaction-type="JTA" -->
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <!-- With a standard configuration and hibernate, we don't need to identify the Entity classes here -->
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
            <!-- table generation policies: validate, update, create, create-drop -->
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver" />
            <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://localhost:5432/postgres" />
            <property name="javax.persistence.jdbc.user" value="postgres" />
            <property name="javax.persistence.jdbc.password" value="postgres" />
        </properties>
    </persistence-unit>
</persistence>
```

Figura 3 – Persistence

Após preenchimentos as **Arraylists** com as informações necessários e estabelecermos as ligações utilizamos **begin()** e damos persist a cada elemento das **Arraylists** , de modo a preencher as tabelas. No final usamos **commit()** para gravar as mudanças na base de Dados.

EJBs

Na parte B) do projeto foram criadas dos **EJBs** necessários para cada **WebService**. Em cada EJB é implementada **EJBRemote** de modo a que as suas funções sejam acedidas pelo **WebService**.

Cada **EJB** é responsável por comunicar com a base de dados para ir buscar a informação para preencher as **ArrayLists** das classes que serão chamadas pelo **WebService**, para tal é estabelece-se ligação com o **Entity Manager** que vai buscar as permissões e os dados da Base de Dados ao ficheiro **persistence.xml**. Após chamarmos o **EntityManager** com a anotação @PersistenceContext é necessário inicializar as querys que vão buscar a informação. De modo a irmos buscar todas as Relações do **Researcher**, é necessária utilização do LEFT JOIN FETCH que vai buscar as restantes tabelas as informações correspondentes do **Researcher**. Como o **Hibernate** só permite um LEFT JOIN FETCH para cada ligação do tipo LAZY ou EAGER, é necessário fazer uma segunda query para ir buscar as **skills** do user. O EJB possui métodos necessários para devolver as informações das **ArrayLists** ao **WebService**.

```
        tx.begin();
        //persists all arraylists to tables
        for (Institution st : mylistI)
            em.persist(st);



        //persists all arraylists to tables
        for (Researcher st : mylistR)
            em.persist(st);



        //persists all arraylists to tables
        for (Publication st : mylistP)
            em.persist(st);

        //persists all arraylists to tables
        for (Skill st : mylistS)
            em.persist(st);

        tx.commit();

        // after commit we have ids:
        //for (Researcher st : mylistR)
            //System.out.println(st);

        System.out.println("Done");
        // Close an application-managed entity manager.
        em.close();
        //Close the factory, releasing any resources that it holds.
        emf.close();
        sc.close();
    }
}
```

Figura 4 – Entity manager em persistence

```java
import java.util.ArrayList;

/**
 * Session Bean implementation class EJBResearcher
 */
@Stateless
@LocalBean
public class EJBResearcher implements EJBResearcherRemote {

    @PersistenceContext(name = "Persistence")
    private EntityManager em;
    /**
     * Default constructor.
     */
    public EJBResearcher() {
        // TODO Auto-generated constructor stub
    }

    private List<Researcher> researchers;
    private List<Skill> skills;

    public void initial() {

        String jpqlR = "SELECT DISTINCT r FROM Researcher r LEFT JOIN FETCH r.institution i LEFT JOIN FETCH r.publications p";
        String jpqlS = "SELECT DISTINCT s FROM Skill s";

        TypedQuery<Researcher> typedQueryR = em.createQuery(jpqlR, Researcher.class);
        List<Researcher> mylistR = typedQueryR.getResultList();

        jpqlR = "SELECT DISTINCT r FROM Researcher r JOIN FETCH r.skills s";
        typedQueryR = em.createQuery(jpqlR, Researcher.class);
        mylistR = typedQueryR.getResultList();

        TypedQuery<Skill> typedQueryS = em.createQuery(jpqlS, Skill.class);
        List<Skill> mylistS = typedQueryS.getResultList();


        /*for (Researcher st : mylistR)
            System.out.println(st);
        */

        skills = mylistS;
        researchers = mylistR;

```

Figura 5 – EJB do Researcher

EJBPublication.java | persistence.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xs
    <persistence-unit name="Persistence" transaction-type="JTA" >

    <!-- use transaction-type="JTA" -->
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <jta-data-source>java:jboss/datasources/postgresDS</jta-data-source>
    <!-- With a standard configuration and hibernate, we don't need to identify the Entity classes here -->
        <class>common.Researcher</class>
        <class>common.Publication</class>
        <class>common.Institution</class>
        <class>common.Skill</class>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
            <property name="                    l.auto" value="update" />
            <property name="                    action.jta.platform"
            value="org.hiber                    .platform.internal.JBossAppServerJtaPlatform" />
        </properties>
    </persistence-unit>
</persistence>
```

Attribute : name
Data Type : string

Figura 6 – Persistence.xml do EJB

SOAP Web Services

De modo a criarmos um **Web Service** é necessário adicionar anotação **@WebService** em cima da classe e consequentemente a anotação **@WebMethod** em cima de cada operação que o **Web Service** terá. De modo a estabelecermos ligação entre o **Web Service** e o EJB é necessário definir as propriedade **jndi** para conseguirmos utilizar a **context.lookup()** para estabelecermos ligação. Uma vez criada a ligação, O **Web Service** chamada o método **initial**() do EJB de modo a que este vá buscar as informações às tabelas e de seguida pede para devolver as **ArrayLists** com a informação necessária. Cabe depois ao **Web Service** construir uma string que será enviada ao **Client** com a informação que este último pretende receber.

```
@Stateless
@WebService
public class Soapresearcher {


    public Soapresearcher() {

    }

    @WebMethod
    public String getSkills() {
        Context context;
        Properties jndiProperties = new Properties();
        jndiProperties.setProperty("java.naming.factory.initial", "org.jboss.naming.remote.client.InitialContextFactory");
        jndiProperties.setProperty("java.naming.provider.url","http-remoting://localhost:8080");
        jndiProperties.setProperty("jboss.naming.client.ejb.context","true");
        try {
            context = new InitialContext(jndiProperties);
            //java:jboss/exported/projetoEAR/Projeto-EJB/EJBResearcher!ejb.EJBResearcherRemote
            EJBResearcherRemote myejb = (EJBResearcherRemote)context.lookup("ProjetoEAR/Projeto-EJB/EJBResearcher!ejb.EJBResearcherRemote");
            myejb.initial();
            System.out.println("chamando MyEJB...");
            List<Skill> lista = myejb.getSkillsInfo();
            String x= "";
            for(int i = 0 ; i < lista.size(); i++) {
                x = x + lista.get(i).getName();
                x = x + "\n";
            }
            return x;

            } catch (NamingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            return "Nada";
        }
    }
```

Figura 7 – Arranque do Web Service do Researcher

Rest Web Services

Para dar início à parte do REST, é chamado o EJB da **Institution** onde são carregadas as listas com a informação de cada uma das entidades e devolvida em formato de string a informação pretendida pelo utilizador a cada um dos pedidos. Já do lado do cliente é onde são criadas as sessões do cliente, target para localizar o URL, e devida resposta para tal. É também são feitos os pedidos **GET e POST** para cada operação efetuada pelo cliente. Para exemplificar, serão mostrados em seguida algumas figuras com exemplos de execução do REST.

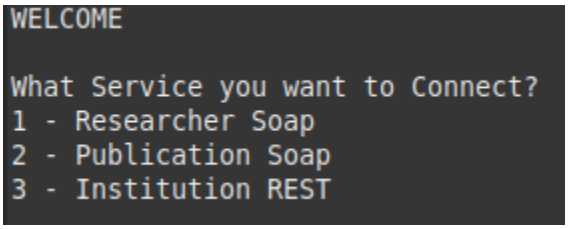

Figura 8 – Menu de boas-vindas e respetivas opções

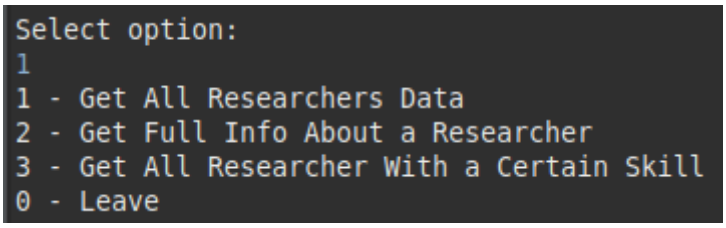

Figura 9 – Menu do Researcher SOAP

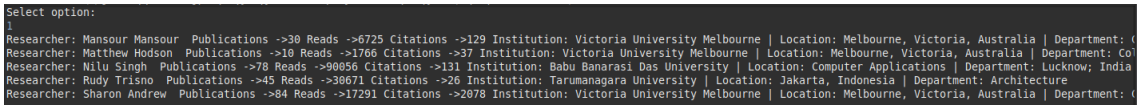

Figura 10 – Resposta da opção Get All Researchers Data

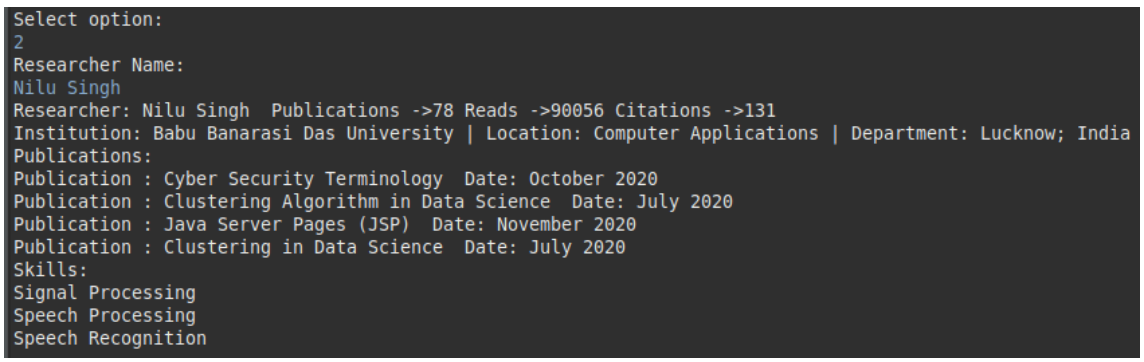

Figura 11 – Pedido e resposta da opção Get Full Info About a Researcher

```
Select option:
3
Skills

Patient
Nursing
Lung Diseases
Housing
Healthcare
Energy Efficient in Building
Spirometry
Architecture
Speech Processing
Signal Processing
Speech Recognition
Mixed Methods

Skill:
Spirometry
Skill => Spirometry
Researcher: Matthew Hodson  Publications ->10 Reads ->1766 Citations ->37
```

Figura 12 – Pedido e resposta da opção Get All Researcher With a Certain Skill

```
Select option:
2
1 - Get All Publications Data
2 - Get Full Info About a Publication
3 - Get All Publication Create By Researcher
0 - Leave
```

Figura 13 – Menu do Publication SOAP

```
Select option:
1
Publication : Research prioritees of respiratory nursing  Date: April 2018
Publication : Hove Based Therapy reduces re-admission rate for Chronic Obstructive Pulmonary Disease  Date: September 2012
Publication : Pre-registration nursing students understanding of patient safety and peer reporting  Date: April 2012
Publication : Clustering Algorithm in Data Science  Date: July 2020
Publication : Kajian Pusat Spiritual  Date: January 2020
Publication : Methodological and Ethical Challenges in investigating of Medication Admininstration  Date: April 2015
Publication : Emergency Shelter Design For Disaster Preparation  Date: July 2020
Publication : Factor Analysis of nursing students perception of patient safety education  Date: May 2014
```

Figura 14 – Resposta da opção Get All Publications Data

```
Select option:
2
Publication Name:
Mixed Method Research
Publication : Mixed Method Research  Date: January 2015
Authors:
Researcher: Sharon Andrew  Publications ->84 Reads ->17291 Citations ->2078
```

Figura 15 – Pedido e resposta da opção Get Full Info About a Publication

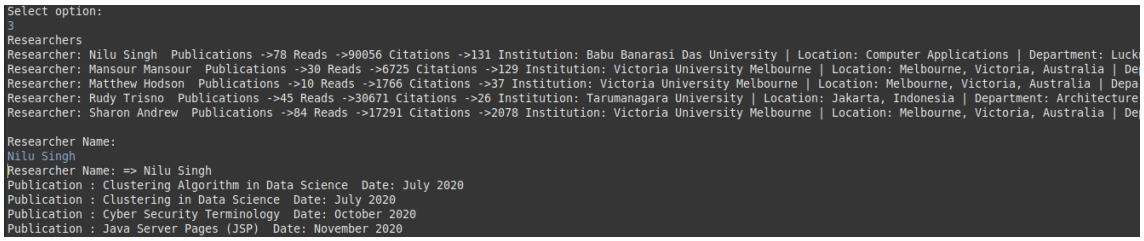

Figura 16 – Pedido e resposta da opção Get All Publications Created by a Researcher

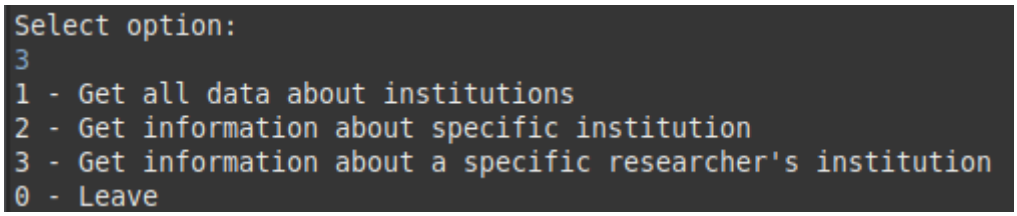

Figura 17 – Menu do Institution REST

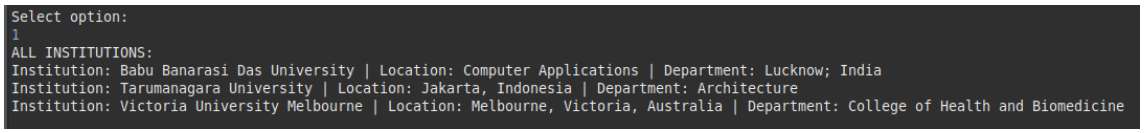

Figura 18 – Resposta da opção Get all data about institutions

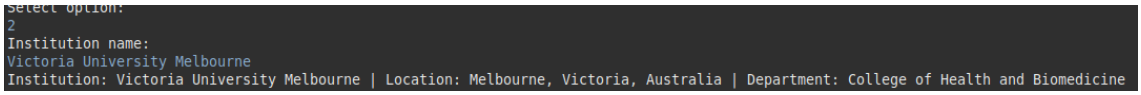

Figura 19 – Pedido e resposta da opção Get information about specific institution

Figura 20 – Pedido e resposta da opção Get information about specific researcher's institution

FrontEnd WebPage

Para esta parte do trabalho foi criado um web servlet, que chama um método do EJB, o qual é inicalizado com anotação @EJB, na qual lhe devolve uma ArrayList com as informações das publicações. De seguida, damos setAtribute da lista de modo a que esta seja acedida do info.jsp, e enviamos o request e response para o info.jsp.

O info.jsp torna se então responśavel por dar print aos items da lista chamando o método toString() de cada item, de modo a demonstrar no ecrã as informações necessárias

```java
/**
 * Servlet implementation class PublicationsInfo
 */
@WebServlet("/PublicationsInfo")
public class PublicationsInfo extends HttpServlet {
    private static final long serialVersionUID = 1L;


    @EJB
    EJBPublicationRemote ejb;
    /**
     * @see HttpServlet#HttpServlet()
     */
    public PublicationsInfo() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        //response.getWriter().append("Served at: ").append(request.getContextPath());
        response.getWriter().println("INFORMAÇÕES DAS PUBLICACOES");
        ejb.initial();
        List<Publication> pubs = ejb.getPublicationsInfo();
        request.setAttribute("lista", pubs);
        request.getRequestDispatcher("/info.jsp").forward(request, response);

    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }

}
```

Figura 21 – Web servlet da Publicação

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Insert title here</title>
    </head>
    <body>
        <strong>Publications Info!</strong>
        <div>Greetings This is the Info About the Publications!</div>
        <br />
        <c:forEach var="item" items="${lista}">
            <div>Publication is ${item.toString()}</div>
        </c:forEach>
    </body>
</html>
```

Figura 22 – info.jsp