Universidade de Coimbra

Faculdade de Ciências e Tecnologia

Departamento de Engenharia Informática

Mestrado em Engenharia Informática

Integração de Sistemas

Projeto 2, 2020/2021, 1º Semestre

Docente

Pedro Nuno San-Bento Furtado
(PNF@DEI.UC.PT)

Grupo: 23

David Silva de Paiva – nº 2020178529 - PL2

Ricardo David Da Silva Briceño – nº 2020173503 – PL2

Coimbra, 2 de dezembro de 2020

Registo de Trabalhos

Lista de funcionalidades/objetivos implementados

Funcionalidades/Objetivos	Implementado/Não implementado	Responsável	Esforço (horas)
Loader	**Implementado**	David	9,5
Desenhar modelo de dados	**Implementado**	David	1/2
Classe Institution	**Implementado**	David	1/4
Classe Publication	**Implementado**	David	1/4
Classe Researcher	**Implementado**	David	1/4
Classe Skill	**Implementado**	David	1/4
Classe JPAWriteResearchs	**Implementado**	David	4
Ficheiro SQL	**Implementado**	Ricardo	4
SOAP Web Service – Researchers	**Implementado**	David	43
Configuração dos projetos	**Implementado**	David	10
Session Bean que pesquisa na base de dados e devolve uma lista de Researchers	**Implementado**	David	8
Criação de um simples WebService SOAP	**Implementado**	David	20
Completar Web Service Soap que invoca o Session Bean e devolve o byte array com o conjunto de Researchers	**Implementado**	David	5
SOAP Web Service – Publications	**Implementado**	Ricardo	10
Configuração dos Projetos	**Implementado**	Ricardo	2
Session Bean que pesquisa na base de dados e devolve uma lista de Publications	**Implementado**	Ricardo	2
Criação de um simples WebService SOAP	**Implementado**	Ricardo	3
Completar Web Service Soap que invoca o Session Bean e devolve o byte array com o conjunto de Publications	**Implementado**	David	3
REST Web Service - Institutions	**Implementado**	Ricardo	24
Configuração dos Projetos	**Implementado**	Ricardo	12

Desenvolvimento de um Session Bean para coletar dados à Base de Dados	**Implementado**	Ricardo	5
Desenvolvimento de uma classe para receber pedidos HTTP	**Implementado**	Ricardo	2
Clonagem das Entidades para simples classes Java	**Implementado**	Ricardo	5
Web Front-end	**Implementado**	Ricardo	5,5
Configuração do Projetos	**Implementado**	Ricardo	1/2
Desenvolvimento de um Session Bean para coletar dados à Base de Dados	**Implementado**	Ricardo	1/2
Desenvolvimento de um Servlet para processar um pedido HTTP	**Implementado**	Ricardo	1/2
Desenvolvimento de um ficheiro JSP	**Implementado**	Ricardo	4
Cliente: consola de texto	**Implementado**	David	7
Configuração do projeto	**Implementado**	David	1
Ligação ao SOAP Web Service - Researchers	**Implementado**	David	2
Ligação ao SOAP Web Service - Publications	**Implementado**	David	1
Ligação ao REST Web Service - Institutions	**Implementado**	David	1
Pesquisa filtrada	**Implementado**	David	1
Apresentação da informação aos utilizadores	**Implementado**	David	1
Relatório	**Implementado**	Ambos	14

Tabela 1 - Funcionalidades e Objetivos implementados no Projeto 2

Autoavaliação individual e global do projeto

O aluno, David Paiva, autoavalia o desempenho do grupo com uma nota de dezanove valores - numa escala de zero a vinte. Relativamente a uma autoavaliação individual, avalia o seu desempenho e o do colega, Ricardo Briceño, com dezanove valores para ambos.

O aluno, Ricardo Briceño, autoavalia o desempenho do grupo com uma nota de dezanove valores - numa escala de zero a vinte. Relativamente a uma autoavaliação

individual, avalia o seu desempenho e o do colega, David Paiva, com dezanove valores para ambos.

Índice de Figuras

Índice de Tabelas

Índice

Esta página foi deixada propositadamente em branco.

1. Enquadramento do projeto

Nesta secção são explicados, de uma forma concisa, os objetivos chave do projeto. Para além disso, é feito um breve enquadramento das funcionalidades da aplicação desenvolvida.

1.1 Objetivo geral

O segundo projeto, da unidade curricular de Integração de Sistemas, consiste na implementação de uma aplicação empresarial organizada em camadas utilizando o modelo Java Enterprise Edition (Java EE). Com isso em mente, foi proposto o desenvolvimento de algumas aplicações baseadas em Enterprise JavaBeans, o desenvolvimento de dois web services SOAP e um web service REST, o uso de Persistence Engine e, também, foi proposto o desenvolvimento de um web front-end.

1.2 Enquadramento

O segundo projeto é composto por um conjunto de aplicações independentes que trocam informação entre si, como é possível observar, na arquitetura apresentada, na Figura 1.

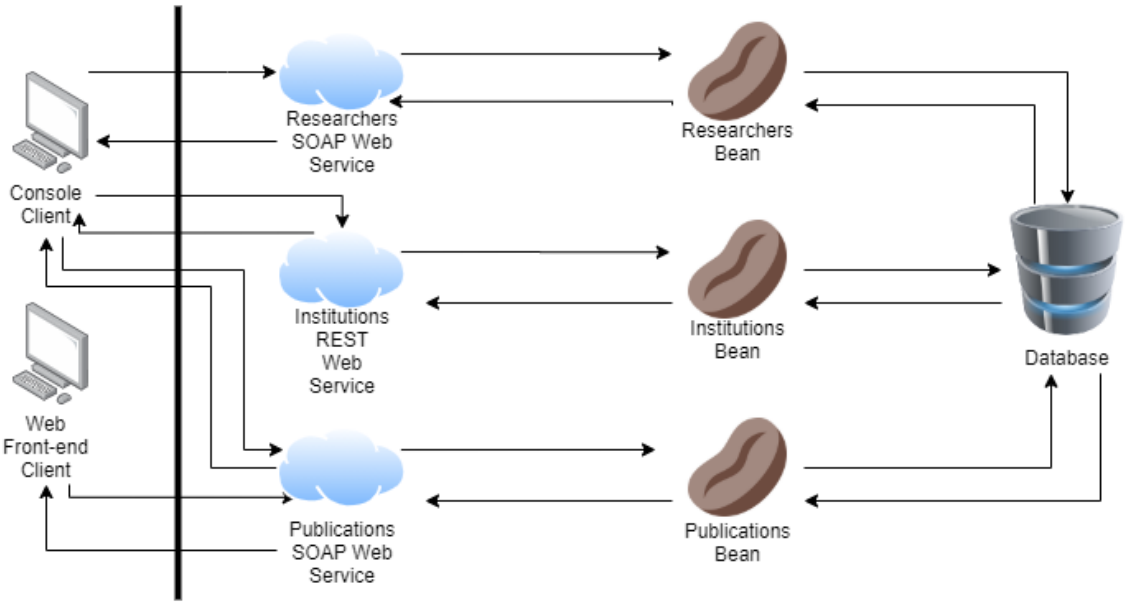

Figura 1 - Arquitetura do segundo projeto

O projeto contém uma base de dados que será carregada com informação, relativa a investigadores, publicações e instituições, através de uma aplicação Loader. De forma a visualizar esta informação foram desenvolvidas duas interfaces, uma gráfica que apresenta a informação relativa a todas as publicações que se encontram na base de dados

e outra interface em formato consola de texto onde o utilizador pode fazer pedidos para ver as informações relativas aos investigadores, publicações e instituições.

As interfaces solicitam os dados aos web services e estes invocam um Enterprise JavaBean (EJB) que vai à base de dados pesquisar os dados solicitados. Posto isto, o EJB devolve a informação para o web service e, consequente, este devolve-a para o cliente. Desse modo, foram desenvolvidos os seguintes web services e respetivos EJB:

- Um SOAP web service que invoca métodos de um EJB que, por sua vez, vai à base de dados pesquisar informação relativa a investigadores.
- Um SOAP web service que incova métodos de um EJB que, por sua vez, vai à base de dados pesquisar informação relativa a publicações.
- Um REST web service que invoca métodos de um EJB que, por sua vez, vai à base de dados pesquisar informação relativa a instituições.

1.3 Estrutura do documento

Para além deste enquadramento, este documento é composto pelos seguintes capítulos:

- Capítulo 2 – Loader
- Capítulo 3 – SOAP Web Service - Researchers
- Capítulo 4 – SOAP Web Service – Publications
- Capítulo 5 – REST Web Service – Institutions
- Capítulo 6 – Projeto Client Application
- Capítulo 7 – Web Front-end
- Capítulo 5 – Considerações Finais, onde é feito um balanço dos objetivos alcançados, assim como as competências adquiridas com a realização do projeto.

2. Loader

Nesta secção, é apresentado a aplicação Loader, implementada no projeto, assim como os aspetos da sua implementação. Esta aplicação tem como objetivo carregar uma base de dados com dados relativos a investigadores, publicações e instituições.

2.1 Modelo de dados

De modo a guardar os dados descritos acima foram criadas quatro classes, que são mapeadas para uma base de dados relacional - Figura 2 – com recurso à tecnologia *Java Persistence API/Hibernate*.

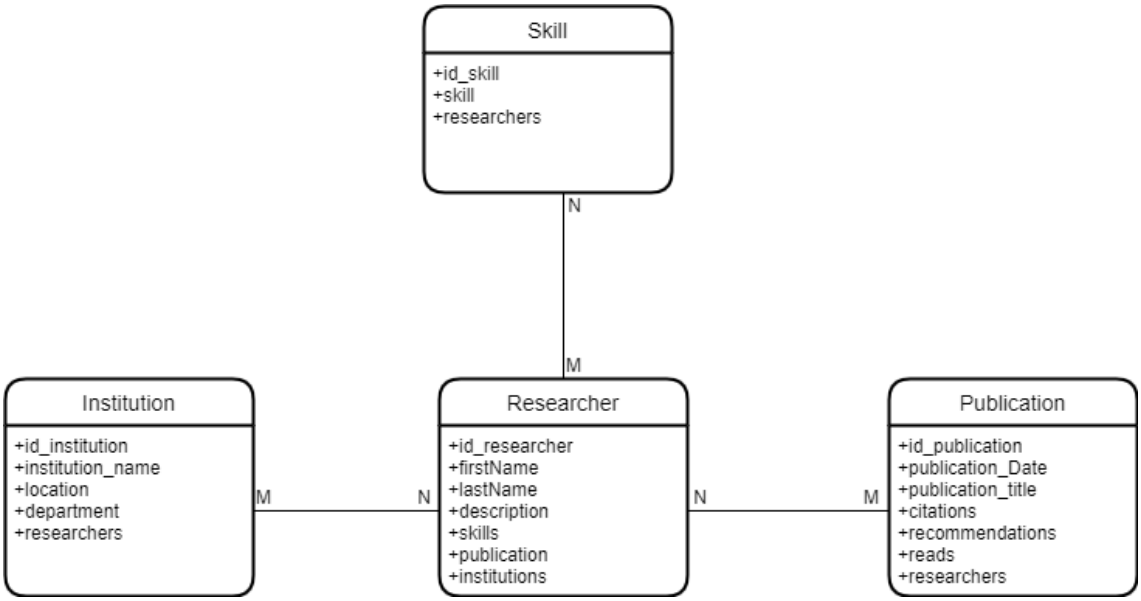

Figura 2 - Modelo de dados utilizado no projeto

A classe Researcher é responsável por armazenar os dados relativos a um investigador. Nomeadamente, o primeiro e último nome, uma pequena descrição, as competências, as publicações das quais é autor e, por fim, as instituições a que está afiliado. Para além disso, possuí um campo *id_researcher* que corresponde a um identificador único que é gerado automaticamente através de duas anotações - Figura 3Figura 3 - Anotações de geração automática de ID.

```
@Id
@GeneratedValue(strategy GenerationType.IDENTITY)
private Long id_researcher;
```

Figura 3 - Anotações de geração automática de ID

A classe Publication é responsável por armazenar os dados relativos a publicações científicas. Nomeadamente, a data de publicação, o título da publicação, os números de citações, recomendações e leituras e o conjunto de investigadores que são autores da publicação. Por fim, possuí um campo *id_publication* que corresponde a um identificador único que é gerado automaticamente.

A classe Institution é responsável por armazenar os dados relativos a instituições académicas ou centros de investigação. Nomeadamente, o nome da instituição, a localização, o nome do departamento e o conjunto e investigadores que fazem parte desta instituição. Por fim, possuí um campo *id_institution* que corresponde a um identificador único que é gerado automaticamente.

A classe Skill é responsável por armazenar os dados relativos a uma competência. Nomeadamente, o nome da competência e o conjunto de investigadores que possuem essa competência. Por fim, possuí um campo *id_skill* que corresponde a um identificador único que é gerado automaticamente.

A relação entre as entidades é de muitas para muitas em todas. Com isto em mente, um investigador pode pertencer a várias instituições e as instituições podem ter vários investigadores a si associados. Um investigador pode ter vários trabalhos publicados e cada publicação pode ter vários investigadores como autores. Por fim, vários investigadores podem ter várias competências.

2.2 Aspetos de Implementação

Como referido acima, o mapeamento entre as classes de dados e a base de dados relacional foi feita com recurso à tecnologia *Java Persistence API/Hibernate*. Com isso em mente, foi criada uma classe – JPAWriteResearchs – que cria os objetos descritos na secção 2.1, com dados retirados do portal *ResearchGate*, e os insere na base de dados. Para isso, é criada uma transação onde todos os dados são persistidos na base de dados e, por fim, é feito um *commit* com os novos dados. Assim a base de dados é carregada com informação relativa aos investigadores, instituições, publicações e competências.

2.3 Ficheiro SQL

Como solicitado no enunciado do projeto, foi criado um ficheiro com algumas *queries sql*. Estas *queries* inserem alguns investigadores e respetivas instituições, publicações e competências à base de dados.

3. SOAP Web Service – Researchers

Nesta secção, é apresentado o SOAP web service que é responsável por fazer a ponte entre um cliente – interface texto – e um EJB que irá consultar a informação relativa a investigadores na base de dados.

3.1 Session Bean: SessionBeanResearcher

De modo a efetuar pesquisas, relativas a investigadores, na base de dados foi criado um EJB remoto com três métodos. Estes estabelecem uma ligação à base de dados e devolvem uma lista de investigadores que lá estão guardados e respetiva informação, com base na consulta realizada. Um dos métodos devolve uma lista com todos os investigadores, e respetiva informação, que se encontram guardados na base de dados, outro devolve uma lista de investigadores com base numa pesquisa pelo primeiro e último nome e, por fim, um terceiro método devolve uma lista de investigadores com base numa pesquisa por uma competência. Na Figura 4 é possível ver a interface remota do EJB. Para efetuar estas pesquisas foi usado JPQL.

```
@Remote
public interface SessionBeanResearcherRemote {
	public List<Researcher> getResearcherByName(String firstName,
	String lastName);
	public List<Researcher> getAllResearchrsInformation();
	public List<Researcher> getResearchrsBySkill(String skill);
};
```

Figura 4 - Interface Remota do EJB: SessionBeanResearcher

Devido às relações entre classes/entidades serem todas de muitos para muitos, na hora de efetuar as pesquisas à base de dados com JPQL tornou-se complicado de conseguir extrair toda a informação associada aos investigadores. Nomeadamente, quando eram acedidas as instituições, publicações e competências a que os investigadores estavam associados. Após várias pesquisas, descobriu-se que uma possível solução passava por fazer várias pesquisas consecutivas, adicionando um *left join fetch* para cada relacionamento entre classes. Para além disso, utilizou-se a *keyword: distinct* para excluir resultados repetidos. Na Figura 5 é possível observar as três queries utilizadas na pesquisa de toda a informação relativa a todos os investigadores presentes na base de dados.

```
String jpql = "SELECT distinct s FROM Researcher s LEFT JOIN FETCH s.institutions";
String jpql2 = "SELECT distinct s FROM Researcher s LEFT JOIN FETCH s.skills";
String jpql3 = "SELECT distinct s FROM Researcher s LEFT JOIN FETCH s.publications";
```

Figura 5 - Queries utilizada para pesquisar todos os invesigadores na base de dados

3.2 SOAP Web Service: ServiceResearchSoap

O SOAP web service relativo aos investigadores possuí três métodos, sendo que cada um deles é responsável por invocar um método do EJB e devolver esse resultado para o cliente que invoca o web service.

À semelhança do EJB descrito na secção 3.1, um dos métodos devolve uma lista com todos os investigadores, e respetiva informação, outro devolve uma lista de investigadores com base numa pesquisa pelo primeiro e último nome e, por fim, um terceiro método devolve uma lista de investigadores com base numa pesquisa por uma competência. No entanto esta lista é enviada para o cliente não como um *ArrayList*, mas como um *array* de *bytes*. Foi tomada esta decisão após vários problemas na criação do web service, visto que não era possível criar um web service com métodos que devolviam *ArrayLists* ou *arrays* de objetos (isto é, *Objetct[]*). Deste modo, decidiu-se transforma o *ArrayList* de investigadores num *array* de *bytes*. É possível observar a implementação dessa transformação na Figura 6.

```
ByteArrayOutputStream out = new ByteArrayOutputStream();
ObjectOutputStream os = new ObjectOutputStream(out);
os.writeObject(ej.getAllResearchrsInformation());
return  out.toByteArray();
```

Figura 6 - Transformação de ArrayList num byte array

De forma a permitir que o web service acedesse aos métodos do EJB, foi feita uma chamada remota sobre o EJB. Para isso, foram definidas as propriedades *jndi* em *runtime* – Figura 7 – e foi feito o *lookup* da interface remota do EJB. Posto isto, o web service conseguia invocar os métodos do EJB apresentado na secção 3.1.

```
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");
```

Figura 7 - Definição das jndi properties em runtime

3.3 Dificuldades sentidas durante a implementação

Numa fase inicial foi um pouco complicado configurar os projetos, devido a problemas de versões do JDK e de outras bibliotecas.

Durante a implementação do EJB foram surgindo alguns problemas, nomeadamente no resultado que era devolvido nas consultas à base de dados. A lista de investigadores devolvida tinha os investigadores corretos, no entanto quando se acedia a outras entidades dentro dos objetos Researcher – as entidades Institution e Skill – era lançada uma exceção. Após muita pesquisa e muito tempo investido, descobriu-se que isto acontecia devido às relações de muitos para muitos entre as entidades. Isto foi resolvido alterando um pouco as queries de pesquisa – como é explicado na secção 3.1.

Foram também sentidas dificuldades na criação do web service, nomeadamente na forma de devolver as listas de investigadores nos métodos do serviço. Para além disso, houve uma grande dificuldade em aceder aos EJB no web services. Numa fase inicial, tentou-se injetar o EJB, no web service, através das anotações *@EJB* e *@Inject*, mas isto não era possível. Estes problemas foram ultrapassados, como foi explicado na secção 3.2.

Esta página foi deixada propositadamente em branco.

4. Projeto SOAP Web Service – Publications

Nesta secção, é apresentado o SOAP web service que é responsável por fazer a ponte entre os clientes – interface texto e web front-end – e um EJB que irá consultar a informação relativa a publicações na base de dados.

4.1 Session Bean: SessionBeanPublications

De forma idêntica, e agora com experiência do SOAP – Researchers, desenvolveu-se um EJB remoto com três métodos. Tal como no SOAP Web Service – Researchers estes métodos estabelecem uma ligação à base de dados e devolvem os resultados pretendidos. O primeiro método devolve uma lista de todas as publicações existentes na base de dados. O segundo método devolve uma lista filtrada de publicações à qual lhe é passada o nome da publicação. O último método devolve, também, uma listra filtrada de publicações, o qual lhe é passado o primeiro e último nome de um investigador, ou seja, o método devolve todas as publicações de um investigador. Na Figura 8 é possível ver a interface remota do EJB. Para efetuar estas pesquisas foi usado JPQL.

```
@Remote
public interface SessionBeanPublicationsRemote {
	public List<Publication> getAllPublications();
	public List<Publication> getPublicationsByTitle(String title);
	public List<Publication> getPublicationsByResearcher(String
firstName, String lastName);
```

Figura 8 - Interface Remota do EJB: SessionBeanPublications

Foi necessário adaptar a implementação do método que devolve a lista de publicações quando lhe é passado o primeiro e último nome de um investigador, pois numa primeira abordagem só era associado o investigador que era passado como argumento e os restantes autores da publicação não eram devolvidos na pesquisa. Desta forma, decidiu-se usar o resultado da primeira pesquisa e voltar a realizar uma pesquisa de modo a guardar todos os autores da publicação. A Figura 9 apresenta a implementação das duas pesquisas que devolvem a lista de publicações quando é recebido o primeiro e último nome de um investigador.

```
@Override
public List<Publication> getPublicationsByResearcher(String firstName, String lastName){
          String jpql = "SELECT distinct p FROM Publication p JOIN p.researchers r WHERE r.firstName='" +
          firstName + "' AND r.lastName='" + lastName+"'";
          TypedQuery<Publication> typedQuery = em.createQuery(jpql,
          Publication.class).setHint(QueryHints.HINT_PASS_DISTINCT_THROUGH, false);
          List<Publication> publicationsList = typedQuery.getResultList();
          List<Publication> pubFinal = new ArrayList<>();
          for (Publication p : publicationsList) {
                    String jpqlR = "SELECT distinct p FROM Publication p JOIN FETCH p.researchers r WHERE
                    p.publication_title='" + p.getPublication_title() +"'";
                    List<Publication> pubAux = em.createQuery(jpqlR,
Publication.class).setHint(QueryHints.HINT_PASS_DISTINCT_THROUGH, false).getResultList();
                    pubFinal.addAll(pubAux);
          }
          return pubFinal;}

}
```

Figura 9 - Implementaçãoo do método getPublicationsByResearcher

4.2 SOAP Web Service: PublicationsSoap

Seguindo a mesma lógica do SOAP – Researchers foram também desenvolvidos três métodos que invocam os métodos do Session Bean e que devolvem o resultado para o cliente num array de bytes, como explicado na secção 3.2.

De maneira ao web service conseguir aceder aos métodos do EJB, foi feita uma chamada remota sobre o EJB. Para isso, foram definidas as propriedades *jndi* em *runtime* e foi feito o *lookup* da interface remota do EJB. Posto isto, o web service conseguia invocar os métodos do EJB tal como feito no SOAP anterior.

4.3 Dificuldades sentidas durante a implementação

Após ter-se realizado o SOAP anterior, o desenvolvimento deste componente não revelou complicações. Houve um cuidado de seguir os mesmos passos de implementação para evitar possíveis problemas e reduzir o esforço investido no SOAP – Researchers. No entanto, surgiram alguns problemas nas queries JPQL que foram mais tarde resolvidos.

5. Projeto REST Web Service – Institutions

A aplicação REST Web Service – Institutions, implementada no projeto, foi desenvolvida com apoio aos tutoriais realizados nas aulas teóricas[1]. Esta aplicação tem como objetivo devolver informação relativa a instituições, em formato JSON, através de pedidos HTTP, passando-lhe URIs especificas.

5.1 Configuração da aplicação: ApplicationConfig

A aplicação REST desenvolvida usa uma configuração simples que oferece um URI base, de modo a ser acedido por todos os seus clientes. Com o recurso da anotação *@ApplicationPath("/request")* definiu-se que *Projeto-WebProject-REST/request/* será a URI base da aplicação.

5.2 Session Bean: RESTBean

Foi criado um EJB responsável por realizar toda a parte lógica do serviço REST, isto é, aceder a base de dados e obter os dados solicitados. Com isso em mente, foram desenvolvidos três métodos que devolvem resultados de pesquisas, por instituições, na base de dados. O primeiro método devolve todas as instituições armazenadas na base de dados, assim como toda a informação associada às mesmas, como os investigadores que nela trabalham. O segundo método devolve a informação de uma instituição caso lhe seja passado por argumento o nome da instituição que se pretende obter a informação. O terceiro método devolve todas as instituições de um dado investigador, sendo passado por argumento o primeiro e último nome do investigador.

5.3 REST Web Service: HTTPRequestCalls

Criou-se uma classe Java responsável por processar os pedidos HTTP do cliente. Esta classe acede ao Session Bean através da anotação *@EJB* para que se possam utilizar os métodos desenvolvidos. Para isto ser possível teve, também, que se adicionar a anotação *@RequestScoped* no cabeçalho da classe *HTTPRequestCalls*. Os métodos são invocados seguindo uma lógica de serviço, através de URIs - Tabela 2.

[1] Using JPA in a WildFly managed environment - Enterprise Applications Tutorial, Web Applications Tutorial - REST Web Services

URI	Método
Projeto-WebProject-REST/request/path/getallinstitution	getAllInsitutions()
Projeto-WebProject-REST/request/path/getinstitutionbyname?name="nomeDaInstituição"	getInstitutionByName(nomeDaInstituição)
Projeto-WebProject-REST/request/path/getinstitutionbyresearchername?firstname=”primeiroNome”&lastname=”ultimoNome	getInstitutionByResearchName(primeiroNome, ultimoNome)

Tabela 2 - URI dos métodos do web service REST

Para que os parâmetros do cliente sejam interpretados, é utilizada a anotação *@QueryParam* o que permite que o valor associado ao parâmetro ‘name’ (?name=) possa ser manipulado em Java. Desta forma, é passado o parâmetro da URI por argumento no método correspondente, como é possível ver no código da Figura 10.

```
@GET
@Path("/getinstitutionbyname”
@Produces(MediaType.APPLICATION_JSON )
public List<InstitutionRest> getInstitutionByName(@QueryParam("name")
String value) {
            List<InstitutionRest> json;
            json = rb.getInstitutionByName(value);
            System.out.println("Requesting intitution by name to the bean
from Bean...");
            return json;
      }
```

Figura 10 - Método getInstituitionByName do web service REST

Dado que os métodos desenvolvidos são meramente informativos e, só servem para a representação dos dados da base de dados, utilizou-se nos 3 métodos a anotação *@GET*. Para representação dos dados, foi soliciado o formato JSON, daí se utilizar a anotação *@Produces(MediaType.APPLICATION_JSON)* o que implica que, quando um cliente realizar um pedido HTTP, o seu pedido será processado e produzido no formato JSON.

5.4 Dificuldades sentidas durante a implementação

Ocorreram muitos problemas com este componente em especial. Primeiro, havia problemas com versões do JDK, optando-se por reinstalar o kit e realizar as devidas configurações. Isto, foi uma boa decisão pois, resolveu vários problemas causados pelas versões Java.

Houve alguns problemas com o CDI quando se associavam os projetos EAR, EJB e Web, no entanto foram resolvidos com pequenas configurações no BuildPath, visto que

quando eram passados os projetos para outros computadores, estes não suportavam as versões do computador e era necessário reconfigurar as bibliotecas do JRE e do Wildfly.

No desenvolvimento propriamente dito, problemas maiores surgiram com as entidades no momento de produzir o JSON no Web Service, nomeadamente problemas com a serialização devido as associações muitos para muitos entre as entidades

As queries foram adaptadas, com recurso ao *keyword join fectch*, permitindo associar as relações e pesquisar todos os dados na base de dados. No entanto, na construção da resposta no formato JSON estes erros acabaram por quebrar o fluxo da nossa abordagem. De forma a resolver estes problemas, tentou-se evitar ao máximo uma abordagem que, concordámos ser pouco correta, optando sempre por seguir as orientações do docente, recursos online como exemplos e as boas práticas de desenvolvimento. Contudo, após inúmeras tentativas não surgiu nenhuma solução e o grupo decidiu avançar com a ideia de criar classes auxiliares - clones das entidades, mas sem anotações. Os dados são clonados no EJB, como mostra o exemplo da Figura 11.

```
public InstitutionRest(Institution ins) {
	this.department = ins.getDepartment();
	this.id_institution = ins.getId_institution();
	this.institution_name = ins.getInstitution_name();
	this.location = ins.getLocation();
}
```

Figura 11 - Clonagem de informação relativa às instituições

Estas são as classes usadas para produzir o JSON e não a entidade propriamente dita. Como se pode observar na figura anterior, a package *entities* tem as classes auxiliares usadas para a produção dos resultados em formato JSON e a package *entitiesEJB* tem as entidades.

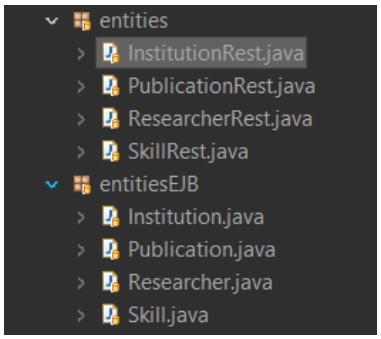

Figura 12 - Classes clones vs entidades

Esta página foi deixada propositadamente em branco.

6. Projeto Client Application

Nesta secção é apresentada a interface de texto que permite invocar os métodos dos web service desenvolvidos e apresenta a informação devolvida pelos mesmos.

6.1 Aspetos de Implementação

A aplicação cliente foi relativamente fácil de desenvolver, visto que o IDE gera automaticamente os *proxys* para os SOAP web service, recorrendo ao ficheiro *wsdl* do web service. Já no caso do REST web service, também foi simples, visto que foi apenas necessário criar um cliente e invocar os métodos do web service através das URIs dos métodos.

Nos web services do tipo SOAP a informação chega ao cliente no formato de um *array* de *bytes*. Deste modo, é necessário transformar este *array* de *bytes* numa lista de objetos – Researcher ou Institution – como mostra a Figura 13.

```
// Deserialize the List of Publications
if (arrPublications != null) {
        ByteArrayInputStream in = new ByteArrayInputStream(arrPublications);
        ObjectInputStream is;
        try {
                is = new ObjectInputStream(in);
                try {
                        publications = (List<Publication>) is.readObject();
                } catch (ClassNotFoundException e) {
                        // e.printStackTrace();
                        System.out.println("Error in the Desearialize of the publication list
- Class Not Found Exeception");
                }
        } catch (IOException e) {
                // e.printStackTrace();
                System.out.println("Error in the Desearialize of the publication list - IO
Exception");
        }
}
```

Figura 13 - Transformação de um array de bytes numa lista de objetos

No web service do tipo REST quando a informação chega ao cliente não era possível percorrer a lista de publicações. Após algum tempo de pesquisa descobriu-se uma solução que é apresentada na Figura 14. Esta solução foi partilhada no fórum de dúvidas da unidade curricular, visto que é um problema que pode afetar todos os alunos.

```
GenericType genericType = new GenericType<ArrayList<InstitutionRest>>(){};
…
institutions =  (ArrayList<InstitutionRest>) response.readEntity(genericType);
```

Figura 14 - Receção da lista de publicações vinda do REST Web Service

6.2 Exemplo de Execução

O utilizador possuí um menu – Figura 15 – com um conjunto de opções, onde pode:

- Listar a informação de todos os investigadores;
- Listar a informação de todas as publicações;
- Listar a informação de todas as instituições;
- Pesquisar investigadores pelo nome;
- Pesquisar investigadores por competências;
- Pesquisar publicações pelo título;
- Pesquisar publicações por investigadores
- Pesquisar instituições pelo nome;
- Pesquisar instituições por investigadores;

```
================================================================================
1 - Get all the Researchers Information
2 - Get all the Publications Information
3 - Get all the Institutions Information
4 - Search Research by Name
5 - Search Research by Skill
6 - Search Publication by Title
7 - Search Publication by Researcher Name
8 - Search Institution by Name
9 - Search Institution by Researcher Name
0 - Exit
```

Figura 15 - Menu da cliente com interface texto

As figuras - Figura 16, Figura 17, Figura 18 - mostram exemplos da execução de algumas das operações possíveis.

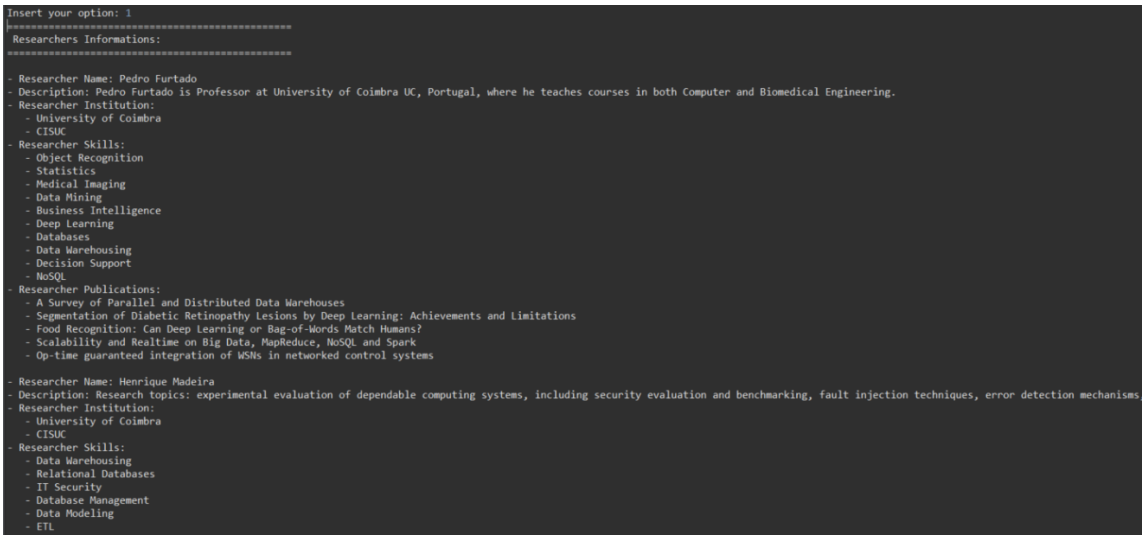

Figura 16 - Listagem de todos os investigadores

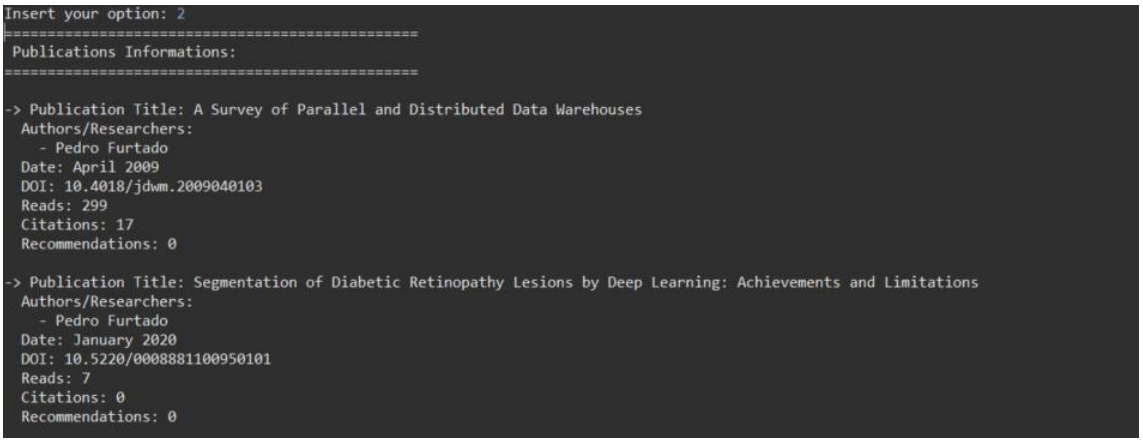

Figura 17 - Listagem de todas as publicações

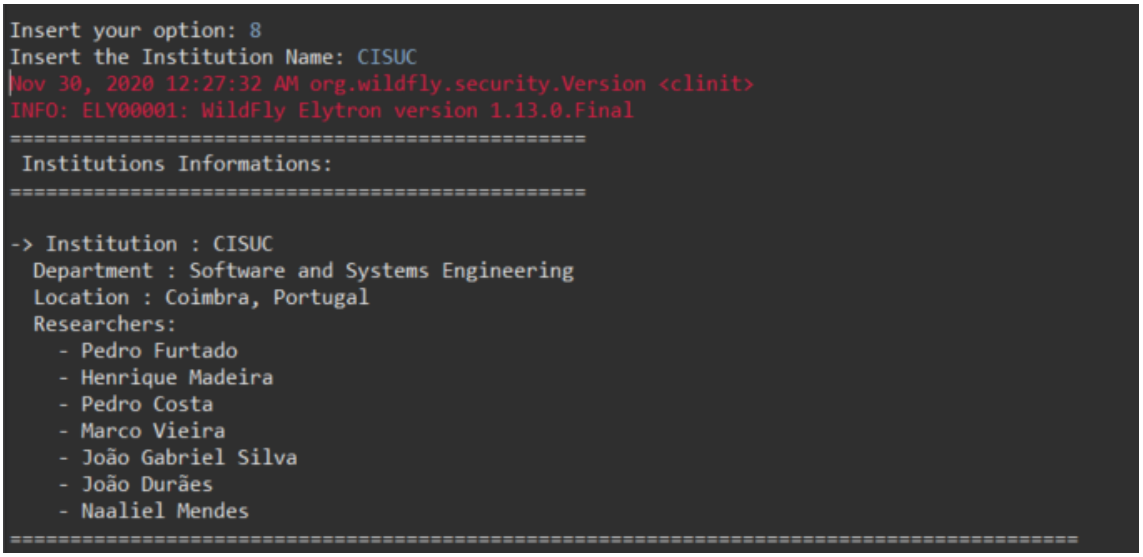

Figura 18 - Listagem das instituições com o nome "CISUC"

6.3 Dificuldades sentidas durante a implementação

Na ligação do cliente com o REST web service ocorreram alguns problemas devido a não existirem algumas bibliotecas, mas depois de alguma pesquisa as mesmas foram adicionadas uma a uma ao projeto e os problemas foram resolvidos.

7. Projeto Web Front-end

Nesta secação é apresentada a aplicação Web Front-end, desenvolvida com apoio aos tutoriais realizados nas aulas teóricas[2]. Esta aplicação tem como objetivo apresentar informação relativos a publicações num browser de forma remota.

7.1 Consumo do SOAP Web Service – Publications: Servlet

O *proxy* do SOAP web service - Publications foi gerado automaticamente pelo IDE recorrendo ao ficheiro *wsdl* do web service. Desta forma, instanciou-se um novo *proxy* e *service* de modo a fazer a chamada do web service SOAP. Após o Servlet consumir o SOAP, o resultado - todas as publicações da base de dados - é guardado num *array* de *bytes*. De seguida, este *array* é deserializado e guardado numa variável que irá ser passada por argumento para um ficheiro JSP.

7.2 Desenvolvimento do ficheiro JSP

Foi desenvolvido um ficheiro no formato JSP de modo a poder mostrar os dados obtidos no consumo do web service SOAP – Publications. Primeiro, começou-se por definir os JSTL, ou seja, as bibliotecas que nos permitem fazer loops, processar dados, aplicar condições, entre outros. Neste caso em específico, utilizaram-se dois JSTL – apresentados na Figura 19, um para percorrer dados e outro para fazer encriptação de dados.

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/functions" prefix = "fn" %>
```

Figura 19 - Definição de bibliotecas no ficheiro display.jsp

Toda a parte de desenvolvimento do JSP foi desenvolvida em HTML com CSS embutido, na qual são apresentados um número dinâmico de Cards conforme o número de publicações, como mostra a Figura 20. Cada publicação tem informação relevante associada, como os seus investigadores (autores), título da publicação, e uma secção, para os mais curiosos, que dá informação mais detalhada da publicação, como a descrição de cada um dos autores, citações, leituras, recomendações e data de publicação como se pode observar na Figura 21.

[2] Web Applications Tutorial - Web applications

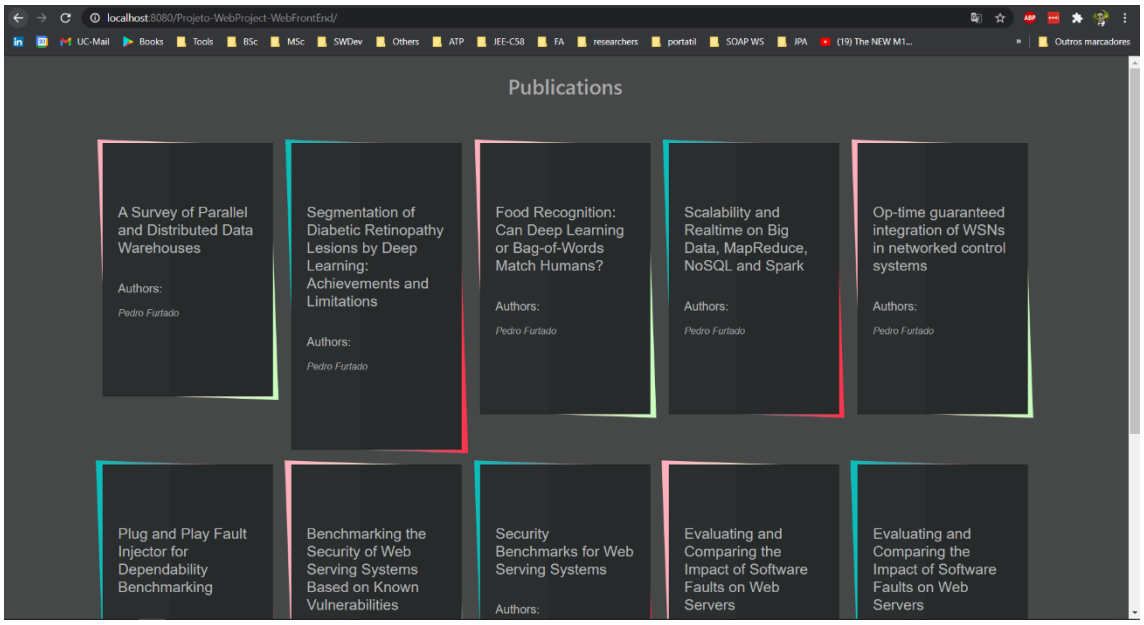

Figura 20 - Página Front-end com a lista de publicações

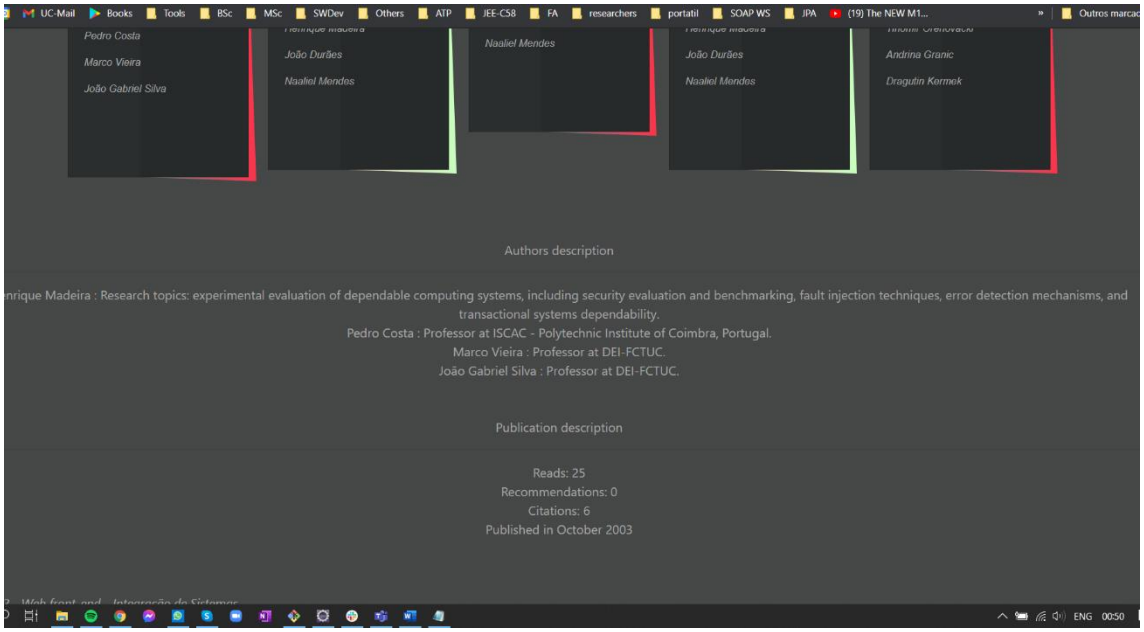

Figura 21 - Informação detalhada de uma publicação

De modo a fazer o collapse da informação detalhada das publicações foi necessário realizar uma codificação do DOI como mostra a Figura 22, recorrendo à função split. A justificação para fazer esta codificação prende-se com a necessidade de aceder a secção detalhada de cada publicação. Para fazer o collapse da informação é necessário ter uma referência a cada descrição. Segundo as regras de CSS, não se podem usar referências com caracteres especiais como o DOI tem, daí não se poder usar este atributo como referência e se ter de criar uma codificação própria para cada publicação.

```
<c:forEach var="pub" items="${publications}">
        <c:set var = "code1" value = "${fn:split(pub.publication_title, ' ')}" />
        <c:set var = "code2" value = "${fn:split(pub.publication_Date, ' ')}" />
        <c:set var = "code3" value = "${pub.reads}" />
```

Figura 22 – Ciclo for que percorre a lista de publicações e as guarda em variáveis para serem usadas no HTML

7.3 Dificuldades sentidas durante a implementação

Neste componente não houve grandes problemas durante o desenvolvimento. O tutorial disponibilizado pelo docente e as aulas práticas, foram suficientes para ter este componente completamente funcional.

Perdeu-se algum tempo a perceber como utilizar os atributos geridos no Servlet e a aceder à informação associada às publicações, como por exemplo a lista de investigadores. Após algumas tentativas, e com a experiência do primeiro trabalho prático, tentou fazer-se uma abordagem semelhante – Figura 23 - e desta forma ultrapassaram-se as dificuldades.

```
<div class="card-body" style="color: darkgrey">
        Authors description <hr>
        <c:forEach var="res" items="${pub.researchers}">
                ${res.firstName} ${res.lastName} : ${res.description}
                <br>
        </c:forEach>

        <br><br>Publication description <hr>
        Reads: ${pub.reads}<br>
        Recommendations: ${pub.recommendations}<br>
        Citations: ${pub.citations}<br>
                Published in ${pub.publication_Date}
        </div>
```

Figura 23 - Código que mostra a informação das publicações na página WEB

Esta página foi deixada propositadamente em branco.

8. Considerações finais

Dado por terminado o projeto é necessário fazer um balanço dos objetivos alcançados, assim como as competências adquiridas.

De uma forma geral, todas as funcionalidades solicitadas no enunciado foram implementadas com sucesso. Com isto em mente, é importante realçar que os objetivos de aprendizagem foram alcançados. Nomeadamente, foram apreendidas um conjunto de técnicas e ferramentas no que toca ao desenvolvimento de web services SOAP e REST.

O desenvolvimento do trabalho prático, as aulas práticas e tutoriais, no seu conjunto consolidaram as nossas competências nestas tecnologias que acabam por ser complexas. Assim sendo, no futuro caso estaremos preparados para enfrentar desafios no âmbito do desenvolvimento deste tipo de aplicações.

Consideramos, ainda, que as nossas soluções podem não estar da forma mais eficiente possível, pois existem inúmeras alternativas de desenvolvimento recorrendo às tecnologias EJB, CDI e JPA. Contudo, tentamos procurar sempre soluções ajustadas aquilo que nos é exigido e esperado no âmbito da disciplina de Integração de Sistemas.