Universidade Autónoma de Lisboa

Departamento de Ciências e Tecnologias

Gestão de Instaladores e/ou Implementadores

(Back-End)

Afonso Casimiro Nº 20150333

Bruno Mileu Nº 20151115

Luís Marques Nº 20150153

Relatório de projeto realizado no âmbito da cadeira de Laboratório de Projeto para obtenção de licenciatura no curso de Engenharia Informática.

Orientador: Prof. Daniel de Matos Silvestre

Parceria: Grupo Luis Simões

Lisboa 2018

Dedicatória

Dedicamos à nossa família, por nos terem apoiado em todo o processo e nos terem acompanhado ao longo deste percurso, de forma incansável.

Índice

Resumo

Este documento é um relatório final de um projeto que foi realizado no âmbito da cadeira Laboratório de Projeto decorrida no 3º ano da licenciatura em Eng. Informática/Eletrónica e de Telecomunicações pela UAL[1], e que se realizou entre Setembro de 2017 e Junho de 2018. Tem por objetivo dar a conhecer o trabalho realizado e as diferentes fases do seu desenvolvimento.

Fruto de uma parceria entre a UAL e o Grupo LS[2], visa o desenvolvimento de uma aplicação de Back-End que permita gerir uma lista de trabalhos a serem realizados em espaços comerciais. Estes trabalhos seriam, por exemplo, para instalar um POS[3] no site XXX, com os componentes YYY e na data DDD, para um grupo de instaladores/implementadores que não estão fisicamente na LS. Visto que a missão da LS é "garantir soluções eficientes e competitivas de Transporte, Logística e serviços auxiliares, promovendo a satisfação de clientes… sob o ponto de vista, económico, social e ambiental" [1], esta aplicação irá certamente auxiliar neste aspeto.

Palavras-chaves: Desenvolvimento de software; Java; PostgreSQL; REST.

[1] Veja a Lista de Abreviaturas e Siglas nº 1, na página 10.
[2] Veja a Lista de Abreviaturas e Siglas nº 2, na página 10.
[3] Veja a Lista de Abreviaturas e Siglas nº 3, na página 10.

Abstract

This document is a final report of a project that was carried out under the chair. Informatics / Electronics and Telecommunications by UAL, which are held in September 2017 and June 2018. It aims to make known and execute the phases of its development.

In order to get a partnership between UAL and the LS Group, the development of a back-end application that has become a list of works of companies in commercial spaces. These jobs were, for example, to install a POS on site XXX, with YYY components and DDD date, for a group of installers / implementers who are not physically in LS. At present it is a mission of LS, which is to guarantee the best and most competitive transport, logistics and ancillary services, promoting customer satisfaction ... from the economic, social and environmental point of view. [1].

Keywords: Software development; Java; PostgreSQL.

Agradecimentos

Este trabalho não ficaria completo sem agradecer a todos os que nos ajudaram a concretizá-lo.

Em primeiro lugar, queremos agradecer ao nosso orientador, o Professor Daniel de Matos Silvestre, pela sua sempre disponível orientação, paciência e simpatia.

Ao grupo Luis Simões, mas principalmente ao seu representante, o Engenheiro Rafael Victória Pereira, pela sua disponibilidade, ajuda e apoio na concretização deste projeto, sem a qual este não seria possível, para solucionar problemas encontrados ao longo da realização do mesmo.

Os nossos agradecimentos finais vão para a nossa família e para os nossos amigos e colegas de curso pelo apoio constante, pelos incentivos e pelas dicas ao longo da realização deste trabalho e, sobretudo, pela paciência.

Índice de figuras

Lista de Abreviaturas e Siglas

1.	**UAL**	*Universidade Autónoma de Lisboa*
2.	**LS**	*Luis Simões*
3.	**POS**	*Point-of-Sale (Posto de Venda)*
4.	**DER**	*Diagrama Entidade-Relacionamento*
5.	**SGBD**	*Sistema de Gerenciamento de Banco de Dados*
6.	**SQL**	*Structured Query Language, ou Linguagem de Consulta Estruturada*
7.	**IBM**	*International Business Machines*
8.	**MER**	*Modelo Entidade Relacionamento*
9.	**ER**	*Entidade Relacionamento*
10.	**SGBDR**	*Sistema de Gerenciamento de Banco de Dados Relacional*

11.	**MCC ou MVCC**	*Controle de concorrência multiversão (ou, Multiversion competition control, em inglês)*
12.	**POO**	*Programação Orientada a Objetos*
13.	**USB**	*Universal Serial Bus*
14.	**MVC**	*Model, view and controller*
15.	**DAO ou DAL**	*Data Access ou Object Access Layer*
16.	**GUI**	*Graphical User Interface*
17.	**SOAP**	*Simple Object Access Protocol*
18.	**REST**	*Representational State Transfer*
19.	**JSON**	*JavaScript Object Notation*
20.	**WS**	*Web Service*

Tabela 1: Tabela das Abreviaturas e Siglas.

1. Introdução

1.1 Propósito do documento

Este documento especifica o projeto realizado no âmbito da cadeira de final de curso, Laboratório de Projeto, que decorreu na Universidade Autónoma de Lisboa. Este trabalho permitiu a criação de um gestor de instaladores/implementadores (Back-End), que consiste numa interface com o utilizador de uma empresa de distribuições para apoio a atribuição de operações e gestão de dados.

1.2 Âmbito do Back-End

O Back-End especificado neste documento permite gerir instaladores/implementadores do Grupo Luis Simões. O mesmo permite a um utilizador gerir dados de uma base de dados de maneira a poder criar operações, requisitadas por clientes, e atribuir implementadores a essa operação.

O Back-End não tem contacto direto com o exterior, mais propriamente, os clientes. Sendo que existe outra parte, o Front-End, que faz a ligação com esses mesmos clientes.

1.3 Situação atual

A empresa em questão não dispõe de um serviço de gestão de base de dados que atribua implementadores a uma operação.

Este Back-End tem a capacidade de armazenar dados relativos a operações a efetuar e posteriormente gerir, através da implementação de uma base de dados relacional. É implementada uma interface gráfica para o utilizador poder fazer a gestão na base de dados.

2. Base de dados

Neste capítulo são apresentados os conceitos e os programas que foram utilizados na criação da base de dados a ser utilizada neste projeto. São também descritos outros programas de forma a mostrar em que aspetos se diferencia do programa utilizado.

2.1 Base de dados hierárquica

“Uma base de dados hierárquica é um tipo de sistema de gerenciamento de [base de dados] que conecta registos numa estrutura de dados em árvore através de ligações de tal modo que cada tipo de registo tenha apenas um possuidor. A base de dados se baseia em um Modelo de Entidades e Relacionamentos: cada registro é uma coleção de atributos (campos), cada um dos quais contendo somente uma informação; uma ligação é a associação entre dois registros. Por exemplo: em uma dada base de dados comercial, uma encomenda (registro) é possuída por um único cliente” [4].

2.2 Base de dados relacional

O termo base de dados está intimamente associado à noção de "uma coleção de informação". De um ponto de vista mais teórico pode-se afirmar que uma base de dados é um conjunto estruturado de informação. Uma base de dados é uma coleção de dados formalmente definida, informatizada, partilhável e sujeita a um controlo central [2].

Uma base de dados é uma coleção de dados inter-relacionados com múltiplas utilizações. Uma base de dados relacional (daqui para diante a expressão base de dados é usada como sinónimo de base de dados relacional) é um sistema de gestão de informação relativamente complexo.

Dado que a base de dados é a componente central do sistema, uma boa técnica de desenho é crucial para a eficácia do sistema.

Se a função duma base de dados fosse simplesmente a de armazenar dados, a sua organização seria relativamente simples. A complexidade estrutural das bases de dados resulta do facto de que ela deve também mostrar as relações que existem entre os dados.

Uma base de dados é composta por um conjunto de tabelas e associações entre as tabelas. A associação entre os dados é o ponto forte dos sistemas relacionais. As tabelas são formadas por linhas e colunas onde figuram os dados. Numa base de dados relacional os dados estão todos representados como valores nas colunas das tabelas.

Neste tipo de aplicação os dados e os programas estão completamente separados. Já o mesmo não se passa, por exemplo, nas folhas de cálculo em que os dados e procedimentos estão frequentemente misturados.

Uma vantagem importante da tabela resulta do facto duma tabela poder ter mais do que uma finalidade e dos seus dados poderem ser vistos com diferentes formas e formatos, ao contrário de um ficheiro.

Neste projeto usamos uma base de dados relacional porque pela sua vantagem em "prover acesso facilitado aos dados, possibilitando que os [utilizadores] utilizassem uma grande variedade de abordagens no tratamento das informações. Pois, enquanto que em [uma base de dados] hierárquica os [utilizadores] precisam definir as questões de negócios de maneira específica, iniciando pela sua raiz, [na base de dados relacionais] os usuários podem fazer perguntas relacionadas aos negócios por meio de vários pontos. A linguagem padrão [da base de dados relacionais] … é a SQL" [3].

Uma base de dados relacional segue o modelo relacional, abaixo veremos no que consiste um modelo relacional.

2.2.1 MER[4]

"Um modelo entidade relacionamento é uma maneira sistemática de descrever e definir um processo de negócio" [5]. O processo é modelado por entidades que são ligadas umas às outras por relacionamentos que expressam as dependências e exigências entre elas, tal como: um cliente pode ter várias operações, mas uma operação só pode ter um cliente. Entidades podem ter várias propriedades/atributos que os caracterizam. Diagramas criados para representar

[4] Veja a Lista de Abreviaturas e Siglas nº 8, na página 10.

graficamente essas entidades, atributos e relacionamentos são designados de diagramas entidade relacionamento.

Um modelo ER é normalmente implementado como uma base de dados. Nos casos de uma base de dados relacional, que armazena dados em tabelas, as próprias tabelas representam as entidades. Alguns dados nestas tabelas apontam para índices em outras tabelas. Tais apontamentos representam relacionamentos.

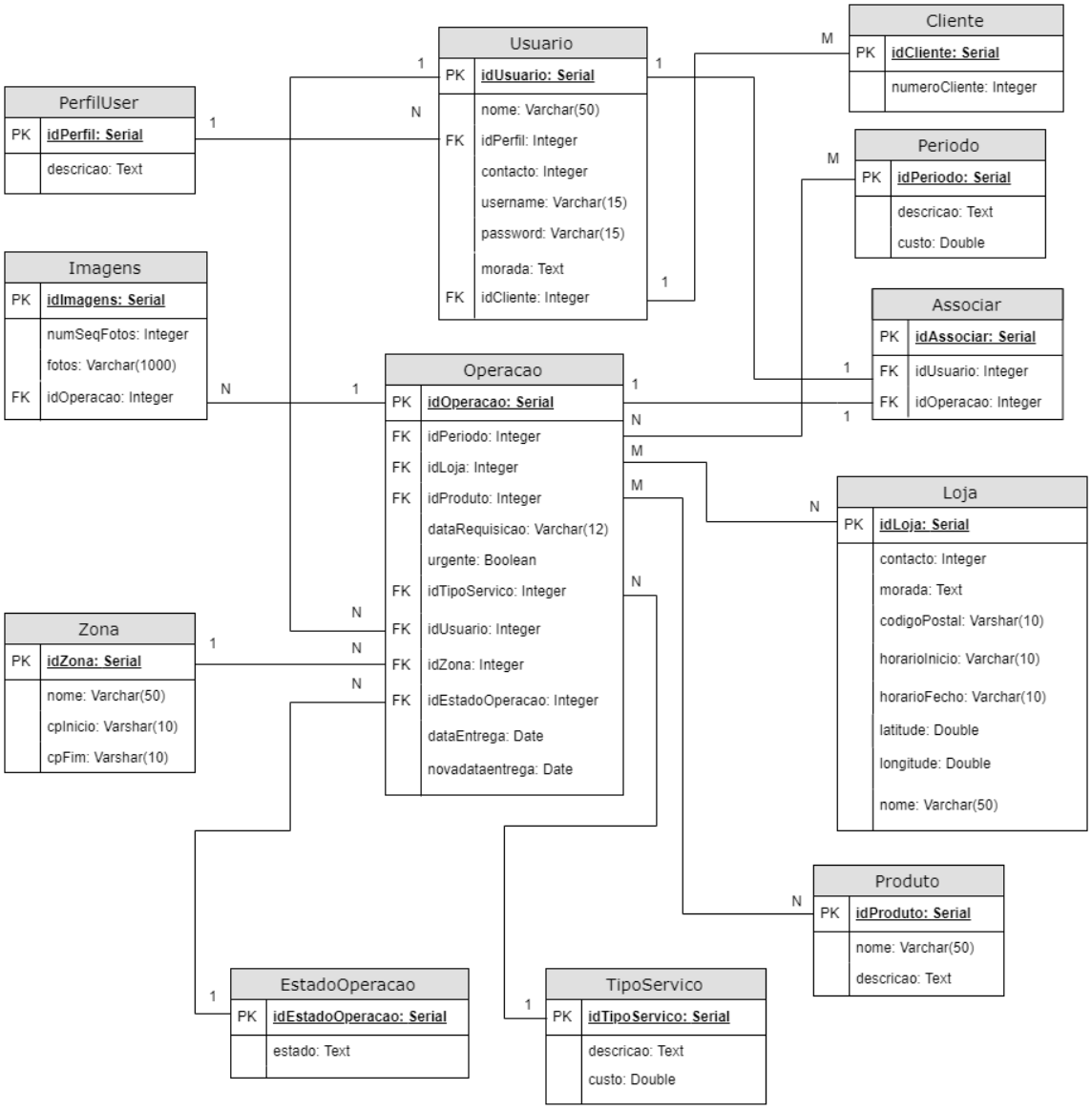

Figura 1 - DER do projeto

2.3 Constituição da Base de Dados implementada

A Base de Dados é constituída pelas seguintes tabelas:

- Usuario (tabela do utilizador);
- PerfilUser (tabela do perfil de utilizador);
- Cliente (tabela específica para o cliente);
- Operacao (tabela de operações);
- Imagens (tabela das imagens);
- Periodo (tabela de período);
- Loja (tabela das lojas);
- Produto (tabela dos produtos);
- Zona (tabela das zonas);
- EstadoOperacao (tabela do estado da operação);
- TipoServico (tabela do tipo de serviço);
- Associar (tabela de associações).

Cada uma das tabelas contém os seus respetivos atributos, que serão descritos em seguida para cada uma delas:

Usuario: como chave primária (PK/Primary Key) tem o idUsuario do tipo Serial, e como chave estrangeira (FK/Foreign Key) tem o idPerfil e o idCliente do tipo Integer. Do mesmo tipo tem também o contacto. Do tipo Varchar contém o nome, o username e a password. Do tipo Text contém a morada.

PerfilUser: como PK tem o idPerfil do tipo Serial, do tipo Text contém o atributo descricao.

Cliente: como PK tem o idCliente do tipo Serial. Do tipo Integer tem o numeroCliente.

Operacao: como PK tem o idOperacao do tipo Serial, e como FK's tem o idLoja, idPeriodo, idProduto, idTipoServico, idEstadoOperacao, idUsuario e idZona do tipo Integer. Do tipo Boolean tem o atributo urgente. Do tipo Varchar tem os atributos dataEntrega e dataRequisicao.

Imagens: como PK tem o idImagens do tipo Serial, e como FK tem o idOperacao do tipo Integer. Do tipo Integer tem o atributo numSeqFotos. O atributo fotos é do tipo Varchar.

Periodo: como PK tem o idPeriodo do tipo Serial. Do tipo Text tem a descricao e do tipo Double tem o custo.

Loja: como PK tem o idLoja do tipo Serial. Do tipo Integer tem o contacto. Do tipo Varchar tem o codigoPostal, o horarioInicio e o horarioFecho. Do tipo Text tem a morada. Do tipo Double tem a latitude e a longitude.

Produto: como PK tem o idProduto do tipo Serial. Do tipo Varchar o nome e do tipo Text a descricao.

Zona: como PK tem o idZona do tipo Serial. Do tipo Varchar o nome, o cpInicio e o cpFim.

EstadoOperacao: como PK tem o idEstadoOperacao do tipo Serial. Do tipo Text o estado.

TipoServico: como PK tem o idTipoServico do tipo Serial. Do tipo Text tem a descricao e do tipo Double o custo.

Associar: como PK tem o idAssociar do tipo Serial, e como FK's tem o idUsuario e o idOperacao, ambos do tipo Integer.

2.4 Sistemas de gestão de bases de dados

Os SGBDs ou SGBDRs[5] são aplicações informáticas complexas, mas essenciais em muitas áreas científicas onde grandes quantidades de informação necessitam de ser combinadas ou exploradas, de diversas formas nem todas fáceis de prever.

Tudo o que fazemos numa base de dados passa pelo SGBD, tornando assim o SGBD responsável por tudo. Tal como guardar os dados, manter em memória os dados mais utilizados, ligar dados e metadados, disponibilizar uma interface para que programas e utilizadores externos possam aceder à base de dados (neste caso, é utilizada a linguagem SQL), encriptar dados, controlar o acesso a informações, manter cópias dos dados para recuperação de uma possível falha, garantir transações na base de dados (veja a figura abaixo).

Figura 2 - Constituição de um SGBD.

Existem vários SGBDs, mas neste projeto será utilizado o PostgreSQL.

[5] Veja a Lista de Abreviaturas e Siglas nº 5, na página 10.

2.4.1 PostgreSQL

O PostgreSQL é um SGBD de código aberto, com mais de 15 anos de desenvolvimento, baseado no POSTGRES versão 4.21, que foi desenvolvido na Universidade da Califórnia em Berkeley Computer Science Department. Foi pioneiro em muitos dos conceitos que só se tornaram disponíveis em alguns sistemas de base de dados comerciais mais tarde. É extremamente robusto e confiável, além de ser extremamente flexível e rico em recursos. É considerado objeto-relacional por implementar, além das características de um SGBD relacional, algumas características de orientação a objetos, como herança e tipos personalizados. A equipa de desenvolvimento do PostgreSQL sempre teve uma grande preocupação em manter a compatibilidade com os padrões SQL92/SQL99.

Como um servidor de base de dados, a sua principal função é armazenar dados de forma segura, apoiando as melhores práticas, permitindo a recuperação dos mesmos a pedido de outras aplicações de software. Pode lidar com cargas de trabalho que vão desde pequenas aplicações single-machine a aplicações de grande porte voltadas para a Internet, onde será utilizada de forma simultânea por vários utilizadores.

Alguns recursos presentes na versão mais recente:

- Sub-consultas;
- Controlo de concorrência multi-versão (MCC ou MVCC);
- Integridade Referencial;
- Funções armazenadas (Stored Procedures), que podem ser escritas em várias linguagens de programação (PL/PgSQL, Perl, Python, Ruby, e outras);
- Gatilhos (Triggers);
- Tipos definidos pelo usuário;
- Esquemas (Schemas);
- Conexões SSL;
- Áreas de armazenamento (Tablespaces);
- Commit em duas fases;
- Arquivamento e restauração da base de dados a partir de logs de transação;
- Diversas ferramentas de clonagem;

- Extensões para dados geoespaciais, indexação de textos, xml, e outras mais.

2.4.2 Outros SGBDs

De seguida são apresentadas 5 alternativas ao PostgreSQL:

- **Oracle:** Originada nos anos 80, criada por Larry Ellison, a Oracle (veja a figura 3) é hoje uma das maiores empresas de tecnologia do mundo, sendo que desde 2009 é também proprietária das linhas de software Java.
  O SGBD é o seu principal produto e desde o seu lançamento no mercado tem vindo a ser o mais aperfeiçoado e desenvolvido para atender as necessidades de empresas e do mundo conectado em que vivemos. Existem diversas versões do software, sendo que cada uma delas contam com características que a tornam ideal a diferentes modelos de negócio. É um software focado para as necessidades de empresas de médio e grande porte.
  Vale lembrar que, até o momento, o SGBD da Oracle é relacional. Para manipulação e gestão do sistema, utiliza-se a linguagem PL/SQL. Linguagem desenvolvida pela Oracle, a partir do SQL (ANSI), expandindo a sua capacidade original.

Figura 3 - Logo da Oracle.

- **MySQL** (veja a figura 4): Uma das bases de dados mais utilizadas e populares, trata-se de uma tecnologia Open Source, ou seja, de utilização de código livre.
  Isto permite que o desenvolvimento do mesmo seja realizado de acordo com as necessidades de uma organização.

Além disso, a sua facilidade de uso e capacidade de rodar em diferentes sistemas operacionais garante um lugar cativo na preferência de grandes empresas por todo o mundo. É considerado o SGBD mais popular do mundo.

Figura 4 - Logo do MySQL.

- **SLQ Server** (veja a figura 5): O poderoso SGBD relacional da Microsoft, lançado em 1988 como parte do Windows NT e posteriormente comercializado como um produto separado e em constante desenvolvimento desde então.
  A grande diferença em relação às outras opções já citadas é a possibilidade do desenvolvedor utilizar linguagens de programação gerenciadas, como o C# e o Visual Basic .NET, em vez de usar declarações SQL. De qualquer das formas, pode utilizar o SQL, ou melhor, o T-SQL – extensão do SQL (ANSI) para o MS SQL Server. O MS SQL Server também possibilita consultas transparentes.
  O Microsoft SQL Server é uma opção extremamente poderosa e, apesar de ser uma solução totalmente paga, consegue destacar-se entre o top dos SGBDs.

Figura 5 - Logo do SQL Server.

- **MongoDB** (veja a figura 6): É um dos SGBDs que mais cresceu nos últimos anos. Trata-se de um sistema NoSQL que foi lançado em 2009 pela empresa que leva o mesmo nome.

  O mesmo procura unir o melhor dos sistemas relacionais e as inovações do NoSQL, mantendo muitas características do primeiro, como índices e consultas dinâmicas, mas também com o modelo de dados orientados a documentos.

  Dessa forma, os aumentos na rapidez de execução através de esquemas flexíveis e a maior facilidade na escalabilidade horizontal, são relevantes quando comparados a outras soluções. O software é Open Source, assim como, o MySQL e o PostgreSQL. O que permite o desenvolvimento do sistema conforme as necessidades do negócio.

Figura 6 - Logo do MongoDB.

- **DB2** (veja a figura 7): Base de dados da gigante IBM. Lançando ainda na década de 1980, embora o seu desenvolvimento tenha ocorrido durante os anos 70. Usufruindo da teoria das bases de dados relacionais, desenvolvida por Edgar Frank Codd, que na época trabalhava para a empresa.

  O sistema é marcado pelo pioneirismo e, além disso, pelo seu suporte aos sistemas operacionais.

  Historicamente, disputou a primeira posição do mercado com a Oracle durante muitos anos. Contudo, o surgimento e o desenvolvimento de novas funcionalidades nos outros SGBDs, contribuíram para uma significativa perda de espaço para o DB2. Ainda assim, é um sistema extremamente relevante.

Figura 7 - Logo do DB2 da IBM.

3. Implementação de POO[6]

Neste capítulo são apresentados os conceitos e os programas que foram utilizados na implementação de programação orientada a objetos neste projeto. São também descritos outros programas de forma a mostrar em que aspetos se diferencia do programa utilizado.

3.1 POO

A POO é uma forma especial de programar, mais próximo de como expressaríamos as coisas na vida real do que outros tipos de programação.

Com a POO temos que aprender a pensar nas coisas de uma maneira distinta, para escrever os nossos programas em termos de objetos, propriedades, métodos, entre outras coisas.

3.1.1 Desenvolvimento da POO

Durante anos, os programadores dedicavam-se a construir aplicações muito parecidas, que resolviam uma vez ou outra, os mesmos problemas. Para conseguir que esses esforços dos programadores possam ser utilizados por outras pessoas foi criada a POO. Esta programação tem uma série de normas a seguir para realizar as coisas de maneira a que outras pessoas possam utilizá-las e adiantar trabalho, sendo assim possível reutilizar o código. Embora possamos fazer os programas de formas distintas, nem todas elas são corretas.

A herança serve para criar objetos que incorporem propriedades e métodos de outros objetos. Assim, podemos construir uns objetos a partir de outros sem ter que reescrever tudo.

O polimorfismo serve para que não tenhamos que nos preocupar sobre o que estamos a trabalhar, e abstrairmos para definir um código que seja compatível com objetos de vários tipos.

[6] Veja a Lista de Abreviaturas e Siglas, na página 10.

3.1.2 Abstração

É utilizada para a definição de entidades do mundo real. Sendo onde são criadas as classes. Essas entidades são consideradas tudo que é real, tendo como consideração as suas características e ações, note o exemplo no quadro abaixo.

Entidade	Características	Ações
Carro, Moto	tamanho, cor, peso, altura	acelerar, parar, ligar, desligar
Elevador	tamanho, peso máximo	subir, descer, escolher andar
Conta Banco	saldo, limite, número	depositar, sacar, ver extrato

Figura 8 - Exemplo de Abstração.

3.1.3 Herança

Na Programação Orientada a Objetos o significado de herança tem o mesmo significado que no mundo real. Assim como um filho pode herdar alguma característica do pai, na Orientação a Objetos é permitido que uma classe herde atributos e métodos da outra (veja a gravura a baixo), tendo apenas uma restrição para a herança. Os modificadores de acessos das classes, métodos e atributos só podem estar com visibilidade public e protected para que sejam herdados. Sendo que public significa que o atributo pode ser visto e que pode ser utilizado de qualquer parte de uma aplicação Java, e que protected significa que o atributo pode ser visto apenas pelas subclasses de uma superclasse.

Uma das grandes vantagens de usar o recurso da herança é na reutilização do código. Este reaproveitamento é utilizado quando se identifica que o atributo ou método de uma classe será igual para as outras. Para efetuar uma herança de uma classe é utilizada a palavra reservada chamada extends.

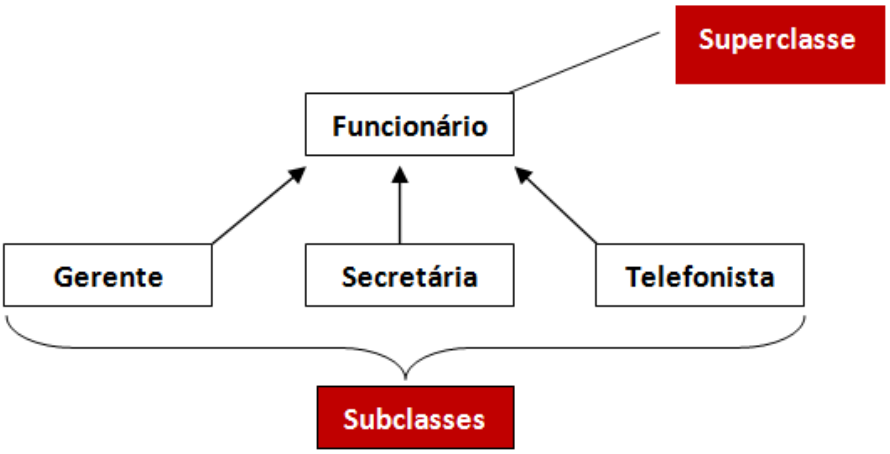

Figura 9 - Exemplo de Herança.

3.1.4 Polimorfismo

O polimorfismo permite que classes abstratas consigam receber comportamentos através de classes concretas. Por exemplo, um dispositivo USB, podemos considerar que o USB seria uma classe abstrata enquanto os dispositivos (Pen Driver, iPad, Câmaras, etc.) seriam as classes concretas, ou seja, o USB é uma especificação que pode ter várias implementações com características diferentes.

O polimorfismo é caracterizado quando duas ou mais classes distintas têm métodos de mesmo nome (veja a figura abaixo), de forma que uma função possa utilizar um objeto de qualquer uma das classes polimórficas, sem necessidade de tratar de forma diferenciada conforme a classe do objeto [10].

O principal objetivo do polimorfismo é diminuir a quantidade de código escrito, de maneira a aumentar a clareza e a facilidade de manutenção.

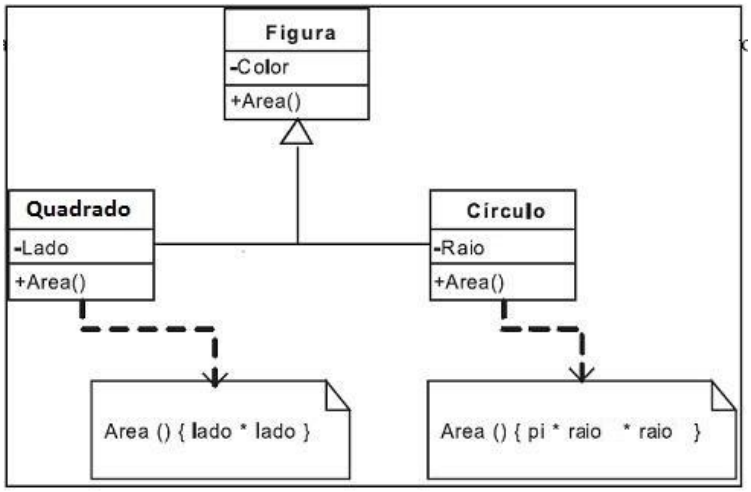

Figura 10 - Exemplo de Polimorfismo.

3.1.5 Encapsulamento

É a técnica utilizada para não expor detalhes internos para os utilizadores externos, tornando algumas partes do sistema o mais independentes possível. Por exemplo, quando um comando de televisão se estraga, apenas se troca ou arranja o mesmo e não a televisão inteira. Neste exemplo do comando, acontece a forma clássica de encapsulamento, pois quando o utilizador muda de canal não se sabe que programação acontece entre a televisão e o comando para efetuar a respetiva ação.

Num processo de encapsulamento os atributos das classes são do tipo private. Sendo que private significa que o atributo é visto por todos, mas não pode ser utilizado nem pelas subclasses.

Para ter acesso a esses tipos de modificadores, é necessário criar métodos setters e getters (veja a figura 11).

Os métodos setters servem para alterar a informação de uma propriedade, ou de um objeto, e os métodos getters para retornar o valor dessa propriedade.

Neste projeto, usamos os métodos de encapsulamento mencionados acima, permitindo então que pudéssemos alterar dados e fazer o retorno dos mesmos a partir da base de dados, sendo

que estes serviços são a essência do projeto, os métodos de encapsulamento, estão presentes em todo projeto, tanto na aplicação, como nos web services.

Métodos getters	Métodos setters

public String getNome() {
return nome;
}
 	
public double getSalario() {
return salario;
}
	public void setNome(String nome) {
	this.nome = nome;
	}
	
	public void setSalario(double salario) {
	this.salario = salario;
	}

Figura 11 - Exemplos de métodos de Encapsulamento.

3.1.6 Exception

As exceptions podem ocorrer por falhas de hardware, exaustão de recursos e erros, estas são uma ocorrência que altera o fluxo do programa.

É através do uso do método try {} e catch {} que se consegue lançar uma exception.

Alguns conceitos de programação das exceções:

- **Throwable:** É o pai de todas as exceptions;
- **Error:** Não são exceptions, mas sim, erros que jamais poderiam ter acontecido. Ex.: rebentamento da memória;
- **Exception:** As classes é que deveriam aqui lançar exceptions e não erros de programação. Ex.: tentar abrir um arquivo que não existe. Então, é lançada uma exception verificada, porque a classe de leitura de arquivos deriva de exception;
- **Runtime Exception:** São exceptions que indicam erros de programas e não de lógica. Este tipo de exception é conhecida como não verificada. Sendo assim, não é requisito declarar uma cláusula try {} e catch {}.

No presente, é possível verificar, muitas clausulas try {} e catch {} ao longo do código, isto facilita em muito na detenção de erros ou outros problemas que possam surgir.

3.1.7 Arquitetura MVC (Introdução – Model e Controller).

MVC é o acrônimo de Model-View-Controller (em português, Modelo-Visão-Controlo) e que, como o próprio nome supõe, separa as camadas de uma aplicação em diferentes níveis.

Embora existam outros padrões de arquitetura de software tais como Pipeline, Blackboard, Microkernel e Reflection, o MVC é um dos mais difundidos e utilizados pelos programadores, principalmente pela funcionalidade e objetividade.

O MVC define a divisão de uma aplicação em três componentes: Modelo, Visão e Controlo. Cada um destes componentes tem uma função específica e estão conectados entre si. O objetivo é separar a arquitetura do software para facilitar a compreensão e a manutenção.

A camada de Controlo atua como intermediária entre as regras de negócio (camada Modelo) e a Visão, realizando o processamento de dados fornecidos pelo utilizador e repassando-os para as outras camadas.

A camada de Modelo consiste na essência das regras de negócio e envolve as classes do sistema e o acesso aos dados.

Cada camada tem uma tarefa distinta dentro do software. A princípio, pode-se dizer que o sistema fica mais organizado quando está estruturado com o modelo MVC. Um novo programador que começar a trabalhar no projeto não terá grandes dificuldades em compreender a estrutura do código, nenhum dos autores de projeto é programador experiente, mas acreditamos que até os experientes se beneficiariam muito da organização que o modelo MVC proporciona, assim como nós nos podemos beneficiar do mesmo, tornou-se muito mais fácil ao longo do projeto, nos orientarmos, pois estava tudo organizado. Além disso, as exceptions são mais fáceis de serem identificadas e tratadas. Em vez de ter de verificar o código todo, atrás do erro, o modelo MVC pode indicar em qual das camadas o erro ocorreu, beneficiando em tempo e esforço. Outro ponto importante do modelo MVC, é a segurança da transição de dados, através da camada de Controlo, é possível evitar que qualquer dado inconsistente chegue à camada

Modelo para persistir na base de dados. A camada Controlo é como uma espécie de Firewall: o utilizador entra com os dados e a camada Controlo fica responsabilizada por bloquear os dados que venham a causar inconsistências na base de dados, sendo que se torna correto afirmar que essa camada é muito importante.

Embora o MVC sugira a organização do sistema em três camadas, nada impede que o programador "estenda" essas camadas para expandir a dimensão do projeto. Mas essa é uma escolha particular de cada programador. No nosso projeto, escolhemos adicionar outras camadas como a Imagem, Excepçoes, Ficheiros e RegistoActividades, para que pudéssemos organizar-nos melhor, mas sempre seguindo a logica do modelo MVC. DAO[7] ou DAL, são camadas muito simples, apenas com os objetos de conexão com a base de dados e as instruções SQL, mas que proporciona uma melhor organização do projeto.

Mas e a camada view (visão)? Abordaremos sobre ela no capítulo 4.

3.2 Outros tipos de programação

3.2.1 Programação Imperativa

Programação Imperativa, é a que a maioria dos programadores aprendem a quando do seu primeiro contato com a área de programação.

Neste tipo de programação a aplicação ou o algoritmo é desenvolvido a pensar numa série de etapas (uma sequência lógica) que o software deve cumprir para atingir o seu objetivo.

3.2.2 Programação Funcional

A programação funcional é um paradigma da programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Este tipo de programação enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa [6].

[7] Veja a Lista de Abreviaturas e Siglas, na página 10.

Uma função, neste sentido, pode ter ou não ter parâmetros e um simples valor de retorno. Os parâmetros são os valores de entrada da função, e o valor de retorno é o resultado da função. A definição de uma função descreve como a mesma será avaliada em termos de outras funções.

4. Interface Gráfica do Utilizador

Neste capítulo são apresentados os conceitos e os programas que foram utilizados na implementação da interface gráfica neste projeto. São também descritos outros programas de forma a mostrar em que aspetos se diferencia dos programas utilizados.

4.1 O que é?

Conhecida como GUI, é onde os resultados são apresentados em modo gráfico. Esta interface é formada através de componentes GUI, conhecidos por controlos ou widgets. Os respetivos componentes são objetos que fazem a interação com o utilizador por teclado, rato ou outros dispositivos de Input ou que venham a servir para entrada de dados. A GUI é visualmente mais iterativa em relação a interface de linha de comandos, esta última é uma interface textual que interpreta os comandos existentes em um sistema operativo e os envia ao núcleo imprimindo posteriormente o resultado do processamento na tela. No presente, a camada view (visão) trata dos aspetos visuais, abaixo abordaremos sobre esta camada.

4.1.1 Arquitetura MVC (View)

A arquitetura MVC é constituída por 3 camadas, duas das quais (Model e Controller) estão referenciadas no capítulo 3.2.7. Fica então a faltar a camada de View (Visão), que está relacionada ao visual da aplicação, ou seja, as janelas que serão exibidas para o utilizador. Nessa camada apenas os recursos visuais devem ser implementados, como janelas, botões e mensagens. Ela recebe dados da camada Controller do MVC e empacota estes dados para depois serem exibidos. Decidimos falar sobre esta camada aqui neste capítulo, porque achamos que é adequado sendo que estamos a abordar as interfaces gráficas.

No presente, a camada view (visão) contém todas das janelas que serão exibidas para o utilizador. Veja o exemplo da janela na imagem abaixo, referente a tela login, ali notamos que a janela contém botões, barra de título e campo para escrita, todos estes são componentes GUI.

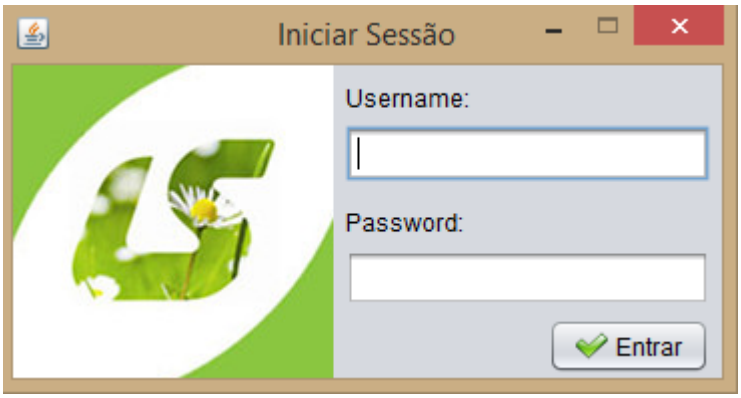

Figura 12 – Janela de inicio de secção com componentes GUI.

4.1.2 Pacotes

O programa de java, Netbeans, contém vários pacotes para a criação de interfaces gráficas. De seguida estão descritos os pacotes que foram utilizados no projeto, tais como:

- **JXL:** JXL API (também conhecida como Java Excel API) é a API mais utilizada para executar testes constituídos por dados de Selenium, que permite aos utilizadores ler, escrever, criar e modificar documentos em Excel(.xls) enquanto está em execução. Não suporta o formato .xlsx.
- **Java.sql:** Fornece ao programa a API para aceder a dados guardados numa fonte de dados (geralmente uma base de dados relacional) e processar os mesmos, usando a linguagem de programação Java. Esta API inclui uma estrutura na qual diferentes drivers podem ser instalados dinamicamente para aceder a diferentes origens de dados. Embora a API JDBC seja principalmente para transmitir instruções SQL para uma base de dados, ela fornece dados de leitura e gravação de qualquer fonte de dados em formato de tabela. O recurso leitor/editor, disponível por meio do grupo de interfaces javax.sql, pode ser personalizado para usar e atualizar dados de um documento, arquivo simples ou qualquer outra origem de dados em formato de tabela.
- **Java.io:** Fornece ao programa o input e o output (entrada e saída, respetivamente) através de fluxos de dados, serialização e sistemas de arquivos. Sem aviso contrário, passar um argumento nulo a um construtor ou um método em qualquer classe ou interface neste pacote, irá lançar uma exceção designada aNullPointerException.

- **Javax Swing:** Os componentes GUI Swing estão dentro do pacote javax.swing que são utilizados para construir as interfaces gráficas.

  Alguns componentes não são do tipo GUI Swing e sim componentes AWT.

  Abaixo são mostrados alguns dos componentes mais usados:

  - **JTextField –** Insere dados provenientes do teclado e serve também para exibição do texto editável ou não editável.
  - **JButton –** Executa uma ação quando o utilizador clicar nele com o cursor.
  - **JCheckBox –** Fornece uma opção que pode ser ou não selecionada.
  - **JComboBox –** Fornece uma lista de itens onde possibilita que o utilizador possa selecionar um item ou escrever para procurar.
  - **JList –** Lista de itens onde podem ser selecionados vários itens.
  - **JPanel –** É a área que abriga e organiza os componentes inseridos.

- **Java AWT:** Antes de existir o GUI Swing, o Java tinha componentes AWT (Abstract Windows Toolkit) que faz parte do pacote javax.awt.

  A diferença entre o GUI Swing e AWT, é na aparência e comportamento dos componentes, ou seja, quando criado por AWT, a aparência e o comportamento dos respetivos componentes são diferentes para cada plataforma e quando é feito por GUI Swing, a aparência e o comportamento funcionam da mesma forma para todas as plataformas.

  Os componentes AWT são mais pesados, pois necessitam de uma interação direta com o sistema de janela local, podendo restringir na aparência e na funcionalidade, tornando-os menos flexíveis comparativamente aos componentes do GUI Swing.

- **Java.util:** Contém a estrutura de coleções, classes de coleção, modelos de eventos, recursos de data e hora, internacionalização e diversas classes de utilitários (um tokenizer de uma string de caracteres, um gerador de números aleatórios e uma matriz de bits).

5. Web Services

Neste capítulo são apresentados os conceitos e os programas que foram utilizados na implementação dos WS[8] utilizados neste projeto. São também descritos outros programas de forma a mostrar em que aspetos se diferencia do programa utilizado.

5.1 O que é?

Um WS é um conjunto de métodos acedidos e invocados por outros programas que utilizam tecnologias Web. É utilizado para transferir dados através de protocolos de comunicação para diferentes plataformas, independentemente das linguagens de programação utilizadas nas mesmas.

Funcionam com qualquer sistema operativo, plataforma de hardware ou linguagem de programação de suporte Web. Estes transmitem apenas informação, ou seja, não são aplicações Web que suportam páginas que podem ser acedidas por utilizadores através de navegadores Web.

Os WS permitem reutilizar sistemas já existentes numa organização e acrescentar-lhes novas funcionalidades sem que seja necessário criar um sistema a partir do zero. Assim, é possível melhorar os sistemas já existentes, integrando mais informação e novas funcionalidades de forma simples e rápida.

[8] Veja a Lista de Abreviaturas e Siglas nº 20, na página 10.

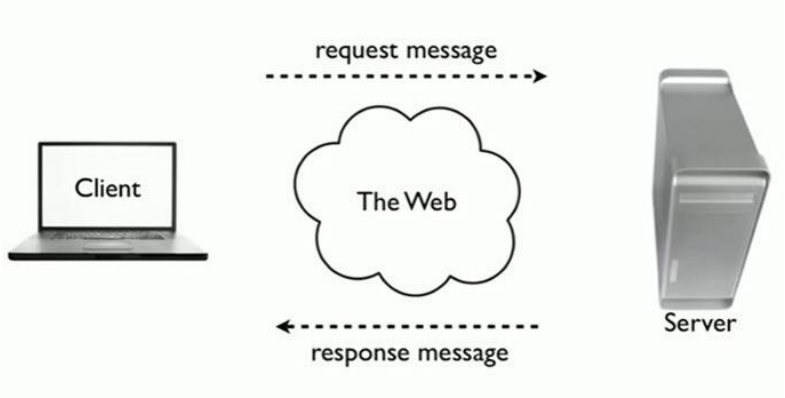

Figura 13 - Estrutura de um Web Service.

5.2 Funcionamento

Tendo em conta as operações disponíveis no WS, a aplicação solicita uma dessas operações. O WS efetua o processamento e envia os dados para a aplicação que solicitou a operação. A aplicação recebe os dados e faz a sua interpretação, convertendo-os para a sua linguagem própria. Por exemplo, a figura abaixo mostra um WS sendo executado, trata-se do WS de inicio de secção, que recebe como parâmetros o username e a password e retorna estes mesmos parâmetros, caso sejam validos, ou seja, caso existam na base de dados.

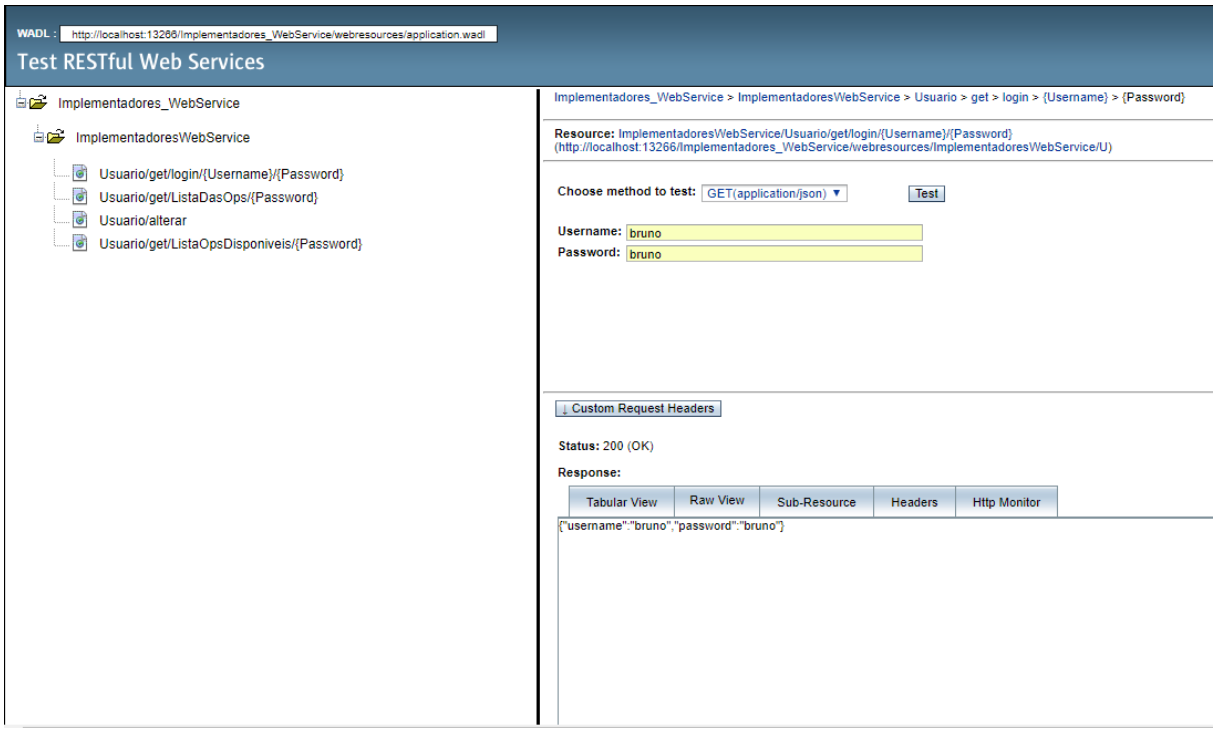

Figura 14 – Web Service de inicio de secção.

5.3 Linguagens

É necessária uma linguagem intermédia que garanta a comunicação entre a linguagem do WS e o sistema que faz o pedido ao WS. Para tal, existem protocolos de comunicação como o SOAP (Simple Object Access Protocol) e o REST (Representational State Transfer).

5.3.1 Protocolo SOAP

O protocolo SOAP utiliza XML para enviar mensagens e, geralmente, serve-se do protocolo HTTP para transportar os dados. Associado ao protocolo SOAP está o documento WSDL (Web Service Definition Language) que descreve a localização do WS e as operações que dispõe. Além disso, fornece a informação necessária para que a comunicação entre sistemas seja possível.

5.3.2 Protocolo REST

O REST é o protocolo mais recente, que surgiu com o objetivo de simplificar o acesso aos WS. Este baseia-se no protocolo HTTP e permite utilizar vários formatos para representação de dados, como JSON (um dos mais utilizados e o usado na parte dos WS deste projeto), XML, RSS, entre outros. No presente, usamos este protocolo utilizando o formato de representação JSON, porque queríamos familiarizarmo-nos com o protocolo mais recente e também por outras vantagens que este oferece como veremos abaixo, o que nos dará mais bagagem no mercado de trabalho.

Assim, uma das grandes vantagens do REST é a sua flexibilidade, já que não limita os formatos de representação de dados. O protocolo REST é também utilizado quando a performance é importante, uma vez que é um protocolo ágil e com a capacidade de transmitir dados diretamente via protocolo HTTP.

5.4 JSON

JSON (JavaScript Object Notation) é um formato leve de troca de dados. É fácil para nós humanos lermos e escrevermos e é fácil para as máquinas analisarem e gerarem informação.

O HTTP permite que protocolos diferentes sejam usados para a comunicação entre o cliente e o servidor. Não vamos explicar todos eles, exceto os quatro mais usados: GET, PUT, POST e DELETE.

GET, PUT e DELETE devem ser implementados como métodos impotentes, que podem ser implementados várias vezes sem que o valor do resultado se altere após a aplicação inicial. O POST, por outro lado, não deve ser idem potente.

5.5 Benefícios

A utilização de WS traz vários benefícios tanto a nível tecnológico, como a nível do negócio. Seguem-se os mais relevantes:

- **Integração de informação e sistemas:** uma vez que o funcionamento do Web Service necessita apenas de tecnologia XML/JSON e protocolos HTTP, a comunicação entre sistemas e aplicações é bastante simplificada. Com um Web Service é possível trocar informação entre dois sistemas, sem necessidade de recolher informação detalhada sobre o funcionamento de cada sistema. Os Web Services permitem ligar qualquer tipo de sistema, independentemente das plataformas (Windows, Linux, entre outras) e linguagens de programação (Java, Perl, Python, etc.) utilizadas.
- **Reutilização de código:** um Web Service pode ser utilizado por várias plataformas com diferentes objetivos de negócio. O código do Web Service é feito uma vez e pode ser utilizado vezes sem conta por diferentes aplicações.
- **Redução do tempo de desenvolvimento:** é mais rápido desenvolver com Web Services, porque os sistemas não são totalmente construídos a partir do zero e facilmente são incluídas novas funcionalidades. O tempo de implementação de sistemas com a utilização de Web Services é mais reduzido, sendo uma boa opção no desenvolvimento de software à medida.
- **Maior segurança:** o Web Service evita que se comunique diretamente com a base de dados, assim, a segurança do sistema que fornece os dados está salvaguardada.
- **Redução de custos:** Com a utilização de Web Services não é necessário criar aplicações à medida para a integração de dados, algo que pode ser bastante caro. Os Web Services tiram partido de protocolos e da infraestrutura Web já existente na organização, requerendo por isso pouco investimento.

5.6 Servidores de base de dados

5.6.1 Glassfish

O servidor do programa Glassfish (veja a figura abaixo) é o utilizado no projeto para estabelecer a ligação entre a base de dados e o Web Service. É um servidor de aplicação open source liderado pela Sun Microsystems para a plataforma Java EE. Glassfish é um software livre, tendo duas licenças que comprovam o mesmo, que são: Common Development and Distribution License (CDDL) e GNU General Public License (GPL).

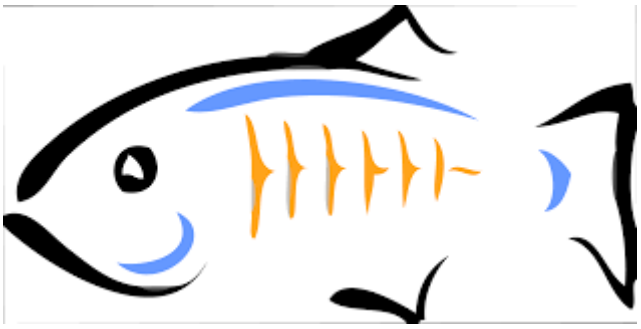

Figura 15 - Logo do Glassfish.

O GlassFish suporta todas as especificações da API Java EE, tais como JDBC, RMI, JavaMail, JMS, JMX etc. e define como coordená-las. Suporta também algumas especificações para componentes Java EE, como Enterprise JavaBeans, conectores, servlets, portlets, JSF e diversas tecnologias de WS. Isto permite que os programadores criem aplicações corporativas portáveis, escaláveis e fáceis de integrar com código legítimo.

A base de Glassfish é o código-fonte fornecido pela Sun e o sistema de persistência TopLink da Oracle. Este código utiliza uma variante do Apache Tomcat como container de servlets, com um componente adicional chamado Grizzly que utiliza para maior escalabilidade e eficiência.

No presente, utilizou-se este servidor aplicacional pela sua popularidade, pela facilidade em maneja-lo e por ser muito completo, além disso, é rico em informação na internet.

5.6.2 Tomcat

O Tomcat (veja a figura abaixo) é um servidor web Java, mais especificamente, um container de servlets. O Tomcat implementa, dentre outras de menor relevância, as tecnologias Java Servlet e JavaServer Pages (JSP) e não é um container Enterprise JavaBeans (EJB).

Desenvolvido pela Apache Software Foundation, é distribuído como software livre. Hoje é um projeto independente, foi criado dentro do Apache Jakarta e posteriormente separado, uma vez que o Jakarta foi concluído.

Figura 16 - Logo do Tomcat.

Na época em que foi criado, foi oficialmente declarado pela Sun como a implementação de referência para as tecnologias Java Servlet e JavaServer Pages. Hoje, no entanto, a implementação de referência é o GlassFish, referida acima.

Este software cobre parte da especificação Java EE com tecnologias como servlet e JSP, e tecnologias de apoio relacionadas como Realms e segurança, JNDI Resources e JDBC DataSources. Tem a capacidade de atuar também como servidor web, ou pode funcionar de maneira integrada a um servidor web dedicado como o Apache ou o IIS. Como servidor web, prevê um servidor web HTTP puramente em Java.

O servidor inclui ferramentas para configuração e gestão, o que também pode ser feito editando-se manualmente arquivos de configuração formatados em XML.

5.6.3 Diferenças

As principais diferenças entre o Tomcat e o Glassfish é que o Tomcat é simplesmente um servidor HTTP e é constituído por servlets Java enquanto que o Glassfish é um servidor de aplicativos Java EE completo, que inclui EJB e todos os outros recursos dessa pilha.

O Tomcat tem uma pegada de memória mais leve, comparado ao Glassfish. O Tomcat tem uma memória de 60 a 70 MB, enquanto que os servidores Java EE pesam em centenas de Megas.

O Tomcat é muito popular para aplicativos da Web simples, em comparação com o Glassfish. Por comparação, a administração do servidor Tomcat é mais fácil do que a administração do Glassfish, já que há menos partes móveis no Tomcat.

Tanto o Tomcat quanto o Glassfish são open source e gratuitos, mas possuem licenças diferentes. Glassfish é licenciado com o dobro, enquanto o Tomcat tem uma licença única. O Tomcat usa a licença Apache enquanto a Glassfish foi licenciada sob CDDL e GPL, referido anteriormente.

5.7 Conexão com a base de dados

Uma pool de conexão, também conhecida como Datasouce, é um recurso que pode ser utilizado no ambiente de desenvolvimento Java EE, onde toda a gestão das transações da base de dados (JTA) será atribuída ao servidor, neste caso o Glassfish.

Em engenharia de software, uma pool de conexões é um conjunto de conexões da base de dados, mantida de forma a que as conexões possam ser reutilizadas quando existirem requisições futuras à base de dados. Pools de conexões são usadas para garantir o desempenho da execução de comandos sobre uma base de dados.

Abrir e manter uma conexão com a base de dados para cada utilizador, especialmente requisições feitas a uma aplicação web dinâmica baseada em bases de dados, é dispendioso e desperdiça recursos.

A pool também reduz a quantidade de tempo que um utilizador deve esperar para estabelecer uma conexão com a base de dados. A sua aplicação prática está na utilização da criação de

bases de dados com a especificação JPA2, que no momento da configuração apontará no respetivo arquivo XML, onde o servidor GlassFish passará a controlar as transações, como já foi referido anteriormente.

6. Conclusão

6.1 Dificuldades encontradas

As dificuldades sentidas ao longo deste projeto foram as normais associadas a um projeto de final de curso. No entanto, creio que, a estas, há a adicionar o facto de se estar a trabalhar em conjunto com outro grupo (Front-End), e de ser um projeto para uma empresa externa e uma empresa de uma dimensão apreciável. Verificou-se uma maior demora de processos, tais como o de análise de recursos e procedimentos a tomar, ter de analisar e optar pelo mais favorável.

Relativamente ao projeto, as dificuldades que existiram não foram significativas. O maior problema que surgiu esteve relacionado com a ligação do servidor com a base de dados e posteriormente a criação dos WS, houve dificuldade em encontrar material integro na internet, além disso, para muitos dos autores os WS eram uma matéria completamente nova.

6.2 Trabalho Futuro

Com o trabalho desenvolvido até este ponto, seria interessante estender este projeto, de modo a que tenha mais utilidade e autonomia a nível de atribuição de tarefas.

Assim poder-se-ia implementar um sistema autónomo em que iria reduzir os custos e aumentaria a velocidade de funcionamento, ficando menos dependente da mão humana.

7. Conclusões Finais

Este projeto foi bastante útil visto que nos proporcionou o contacto com uma realidade diferente, realidade essa que se situa no mundo empresarial. Tornando este projeto como que uma primeira abordagem ao nosso futuro.

Permitiu-nos aplicar conhecimentos adquiridos ao longo do ensino superior e ao mesmo tempo crescer em termos de programação relacionada com bases de dados e com WS. Permitiu também crescer a níveis pessoais, em termos de responsabilidade, lidar com pressão, etc, em parte pelo peso da empresa com a qual a UAL fez parceria neste projeto, o representante da mesma ajudou-nos bastante com o seu conhecimento e experiencia, apreciamos bastante o apoio que tivemos tanto dele como do nosso orientador.

8. Referências

[1] Luis Simões - Missão | Visão | Política | Valores [Em linha]. [Consultado em Set. de 2018]. Disponível em: < http://www.luis-simoes.pt/page/missao-visao-politica-valores-LS :>.

[2] Escola Profissional da APRODAZ - Relatório PAP Escrito - Curso Técnico de Sistemas de Informação Geográfica [Em linha]. (2015), páginas 56. [Consultado em Set. de 2018]. Disponível em www:<https://pt.slideshare.net/zuzu1993/relatrio-pap-escrito-curso-tcnico-de-sistemas-de-informao-geogrfica :>.

[3] Wikipédia - Banco de dados relacional [Em linha]. [consultado em Set. de 2018]. Disponível em www:< https://pt.wikipedia.org/wiki/Banco_de_dados_relacional >.

[4] Wikipédia - Modelo hierárquico [Em linha]. [consultado em Set. de 2018]. Disponível em www:< https://pt.wikipedia.org/wiki/Modelo_hier%C3%A1rquico >.

[5] Wikipédia - Modelo entidade relacionamento [Em linha]. [consultado em Set. de 2018]. Disponível em www:< https://pt.wikipedia.org/wiki/Modelo_entidade_relacionamento >.

[6] Wikipédia - Programação funcional [Em linha]. [consultado em Set. de 2018]. Disponível em www:< https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_funcional#cite_note-1 >.

[7] Paul Deitel and Harvey Deitel, Java: como programar; tradução Edson Furmankiewicz; revisão técnica Fábio Lucchini, São Paulo: Pearson Education do Brasil, 2017. [Consultado em Set. de 2018]. Disponível na internet em:< https://www.wook.pt/livro/java-como-programar-harvey-deitel/11352691 :>. ISBN 9788576055631.

[8] Vinícius Carvalho, PostgreSQL Banco de dados para aplicações Web modernas, São Paulo: Casa de Código, 2015. [Consultado em Set. de 2018]. Disponível em: < https://www.casadocodigo.com.br/products/livro-postgresql:>. ISBN: 978-85-5519-255-5.

[9] Rafael Santos, Introdução à Programação Orientada a Objetos Usando Java – 2ª Ed. Campus, 2003. [Consultado em Set. 2018]. Disponível em:< https://www.fnac.pt/Introducao-a-programacao-orientada-Rafael-Santos/a634574 :>. ISBN 9788535212068.

[10] Carlos Pampulim Caldeira, PostgreSQL - Guia Fundamental, 1ª Edição. Lisboa: Manuel Robalo, 2015. [Consultado em Set. de 2018]. Disponível em:< http://www.silabo.pt/livros.asp?num=533 :>. ISBN 978-972-618-795-0.

[11] Steven F. Lott, A Programmer's Introduction to Python, Building Skills in Python, Part III. Data + Processing = Objects, Chapter 22. Advanced Class Definition, Polymorphism.