INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Articulação de várias linguagens de programação/tecnologias no desenvolvimento aplicacional

ANTONIO ARISTIDES ROMUALDO CARVALHO
Julho de 2015

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Articulação de várias linguagens de programação/tecnologias no desenvolvimento aplicacional

António Aristides Romualdo Carvalho

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Tecnologias do Conhecimento e Decisão

Orientador: Professor Doutor Luiz Felipe Rocha de Faria

Júri:

Presidente:

Vogais:

Professor Doutor Luiz Felipe Rocha de Faria

Porto, Julho 2015

Dedicatória

Aos meus pais e noiva Vânia Fernandes pela companhia, alento e força que me proporcionaram para a conclusão do mestrado.

Resumo

O desenvolvimento aplicacional é uma área em grande expansão no mercado das tecnologias de informação e como tal, é uma área que evolui rápido. Os impulsionadores para esta característica são as comunicações e os equipamentos informáticos, pois detêm características mais robustas e são cada vez mais rápidos. A função das aplicações é acompanhar esta evolução, possuindo arquiteturas mais complexas/completas visando suportar todos os pedidos dos clientes, através da produção de respostas em tempos aceitáveis.

Esta dissertação aborda várias arquiteturas aplicacionais possíveis de implementar, mediante o contexto que esteja inserida, como por exemplo, um cenário com poucos ou muitos clientes, pouco ou muito capital para investir em servidores, etc. É fornecido um nivelamento acerca dos conceitos subjacentes ao desenvolvimento aplicacional. Posteriormente é analisado o estado de arte das linguagens de programação *web* e orientadas a objetos, bases de dados, *frameworks* em JavaScript, arquiteturas aplicacionais e, por fim, as abordagens para definir objetivos mensuráveis no desenvolvimento aplicacional. Foram implementados dois protótipos. Um deles, numa arquitetura multicamada com várias linguagens de programação e tecnologias. O segundo, numa única camada (monolítica) com uma única linguagem de programação. Os dois protótipos foram testados e comparados com o intuito de escolher uma das arquiteturas, num determinado cenário de utilização.

Palavras-chave: arquitetura multicamada, arquitetura monolítica, *framework*, *front-end*, *middle-tier, back-end*

Abstract

The application development is an area in great expansion in the information technologies' market and therefore is a fast evolving field. The boosters for this feature are the communications and computer equipment, as they have more robust features and are getting faster. The applications' function is to track these developments, owning more complex / complete architectures in order to support all customer orders, by producing responses in acceptable time.

This dissertation covers several applicational architectures possible to implement through the context which are inserted, for example, a scenario with few or many customers, or very little capital to invest in servers, etc. A leveling about the concepts underlying the application development is provided. Later is analyzed the state of the art of web programming languages and object-oriented databases, JavaScript's frameworks, applicational architectures and finally approaches to set measurable goals in application development. Two prototypes were implemented. One of them is a multilayer architecture with multiple programming languages and technologies. The second is a single layer (monolithic) which has a single programming language. The two prototypes were tested and compared in order to select one of the architectures in a given usage scenario.

Keywords: *multilayer architecture, monolithic architecture, framework, front-end, middle-tier, back-end*

Agradecimentos

Gostaria de agradecer ao meu orientador, Professor Doutor Luíz Faria, pelo seu empenho, disponibilidade e orientações que foram fundamentais para a conclusão desta dissertação, bem como ao ISEP pela excelente estrutura curricular que o curso possui e pelos excelentes profissionais que tem, sempre dispostos a ajudar e a esclarecer dúvidas. Queria manifestar também o meu agradecimento às empresas CGI e OLR (bem como aos colegas das mesmas) que me acompanharam e apoiaram na elaboração da presente dissertação. Agradeço também aos meus pais e à minha noiva Vânia Fernandes pela companhia, alento e força. Finalmente, agradeço a colegas e amigos pelo apoio e incentivo constantes durante a realização desta dissertação.

Índice

Lista de Figuras

Lista de Tabelas

Índice de Código

Acrónimos e Símbolos

Lista de Acrónimos

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASP	Active Server Pages
ASPX	Active Server Page eXtended
BD	Base(s) de Dados
CGI	Common Gateway Interface
CRUD	Create Read Update Delete
DDR	Double Data Rate
DLL	Dynamic-Link Library
FTP	File Transfer Protocol
GB	Gigabyte
Ghz	Gigahertz
GQM	Goal Question Metrics
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
IIS	Internet Information Services
JDBC	Java Database Connectivity
JDK	Java Development Kit
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
Mb/s	Megabits per second
Mhz	Megahertz

ms	Milissegundos
MVC	Model-View-Controller
NAT	Network Address Translation
NT	New Technology
ODBC	Open Database Connectivity
POP	Post Office Protocol
RAM	Random Access Memory
RMI	Remote Method Invocation
RPC	Remote Procedure Calls
SCSI	Small Computer System Interface
SDRAM	Synchronous Dynamic Random-Access Memory
SGBD	Sistema(s) de Gestão de Base(s) de Dados
SGBDOR	Sistema de Gestão de Base de Dados Objeto Relacional
SGBDR	Sistema de Gestão de Base de Dados Relacionais
SMART	Specific Mensurable Attainable Relevant Timely
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SP	Service Pack
SQL	Structured Query Language
SSD	Solid-State Drive
SSH	Secure Shell
TCP/IP	Transmission Control Protocol / Internet Protocol
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

WSDL	Web Services Description Language
WWW	World Wide Web
XML	eXtensible Markup Language
XSD	XML Schema Definition

1 Introdução

Neste capítulo irá ser feita a apresentação do projeto, a nível de interesse e motivação. Serão traçados os objetivos do estudo, bem como os resultados esperados e a estrutura da presente dissertação. Irão ser apresentados conceitos a nível das linguagens de programação web e orientadas a objetos, bem como sistemas de gestão de base de dados, servidores aplicacionais e processo de avaliação do desempenho das aplicações. Por fim, será feita uma síntese do que foi abordado no presente capítulo.

1.1 Contribuição e motivação

Atualmente a informação contida nas páginas de Internet é vasta e permite-nos expandir/aprofundar os nossos horizontes, permitindo-nos ser pessoas detentoras de um conhecimento mais amplo. A informação e o conhecimento extraídos nessas páginas são armas poderosas que são acedidas de qualquer ponto do mundo (desde que se possua internet e um dispositivo que suporte um *browser*).

A evolução tecnológica incrementou e melhorou funcionalidades disponíveis, por exemplo, os computadores e/ou dispositivos móveis tornaram-se uma necessidade nos dias que correm, pois permitem o acesso a serviços, educação, email, etc. Esta evolução leva também a uma maior complexidade no processo de desenvolvimento de software. Para ultrapassar esta dificuldade surgem as *frameworks*. Estas *frameworks* visam facilitar o processo de desenvolvimento de *software*.

O interesse desta dissertação é comparar diversas arquiteturas aplicacionais para, seguidamente se compararem algumas *frameworks* de uma linguagem de programação utilizada no desenvolvimento *web* (para *front-end*), com o *middle-tier* e *back-end* em linguagens de programação orientadas a objetos. Este processo é ilustrado na Figura 1.

A forma de comparação será feita através:

- Dos tempos de resposta para fazer operações à base de dados, de forma a validar que arquitetura poderá ser mais vantajosa (mediante o contexto inserido).
- O custo/benefício das diferentes arquiteturas, quer a nível de esforço quer a nível de custo monetário.

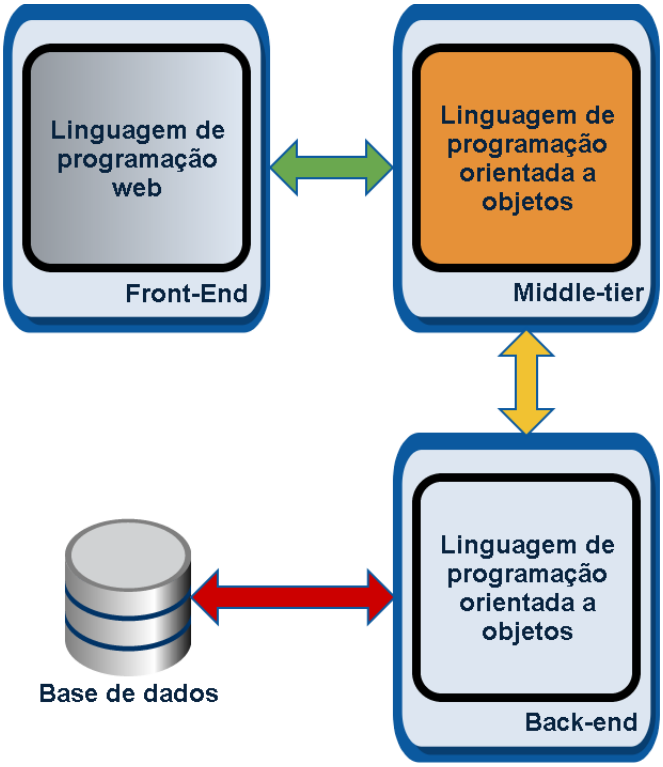

Figura 1 - Processo de interligação das diversas linguagens de programação

1.2 Objetivos

Nesta dissertação, serão estudadas arquiteturas de *software* para analisar como se podem articular linguagens de programação *web* e linguagens orientadas a objetos, bem como Sistemas de Gestão de Bases de Dados (SGBD) existentes, apresentando detalhes, vantagens e desvantagens dos mesmos. Será feita uma verificação dos requisitos apresentados pelas linguagens de programação *web* do lado do cliente e servidor.

Os objetivos principais desta dissertação são:

- Validar a possível integração entre as linguagens de programação *web* (para *front-end*) com as orientadas a objetos (para *middle-tier* e *back-end*), bem como as suas vantagens e desvantagens
- Apresentar conclusões acerca da escolha da linguagem de programação *web*, linguagem orientada a objetos e SGBD escolhido
- Realizar uma análise a algumas *frameworks* para a linguagem de programação *web* escolhida, enumerando as vantagens e desvantagens das mesmas

- Implementar um protótipo que permita demonstrar as potencialidades e benefícios da integração de linguagens de programação na arquitetura de aplicações *web*

Os objetivos enunciados serão alcançados através da implementação das seguintes tarefas:

- Implementar protótipos baseados em diferentes arquiteturas, para se poder ter uma estimativa do esforço computacional em *run-time* dos mesmos, bem como tempo despendido na implementação dos protótipos e para poder verificar qual das arquiteturas é mais vantajosa
- Analisar as vantagens de usar JDBC em alternativa a usar serviços ao SGBD em *C++*.

1.3 Resultados esperados

Com o desenvolvimento desta dissertação pretende-se fazer uma análise de desempenho de diferentes soluções (uma com várias linguagens de programação e outra somente com uma linguagem de programação), de forma a concluir qual das soluções possui maior performance, robustez, requer menor esforço de desenvolvimento e melhores tempos de resposta.

1.4 Estrutura da dissertação

Esta dissertação está organizada em quatro capítulos:

- **Capítulo 1** – este capítulo possui uma introdução ao tema da dissertação, bem como definição de objetivos, resultados esperados, o estado da arte e um resumo
- **Capítulo 2** – este capítulo apresenta um estudo acerca das várias linguagens de programação web e orientadas a objetos, bem como sistemas de gestão de base de dados, *frameworks* em JavaScript, arquiteturas aplicacionais e metodologias/boas práticas
- **Capítulo 3** – é apresentado neste capítulo o caso de estudo desta dissertação, ou seja, apresentação dos casos de uso permitidos nos protótipos, os padrões de *software* utilizados para a sua elaboração, as duas implementações efetuadas para se efetuar a análise comparativa de desempenho, bem como as suas arquiteturas, modelos de domínio, diagramas (de sequência e de classes), tabelas da base de dados, exemplos de código utilizado e *screenshots*. É feito posteriormente um desenho da experiência (onde se refere como irão ser feitos os testes aos protótipos). Finalmente é feita a análise de resultados
- **Capítulo 4** – este capítulo apresenta as conclusões do estudo efetuado na presente dissertação.

1.5 Conceitos subjacentes ao desenvolvimento aplicacional

Nesta secção serão fornecidos conceitos genéricos no âmbito das linguagens de programação *web*, orientadas a objetos, sistemas de gestão de base de dados, servidores aplicacionais e as metodologias/boas práticas no processo de avaliação do desempenho de aplicações.

1.5.1 Linguagens de programação *web*

Neste tópico será apresentada uma introdução a nível geral de conceitos das linguagens de programação *web*.

O que é a internet? [1]

A Internet é um conjunto vasto de redes informáticas com milhões de computadores, permitindo o acesso a transferência de dados e a informações. Possui uma grande variedade de recursos e serviços, desde documentos interligados, partilha de arquivos, etc. A Figura 2 ilustra uma representação do que foi descrito no presente parágrafo.

Alguns dos serviços disponíveis na Internet são *feeds*, acesso remoto a outras máquinas, e-mail, transferência de arquivos, etc.

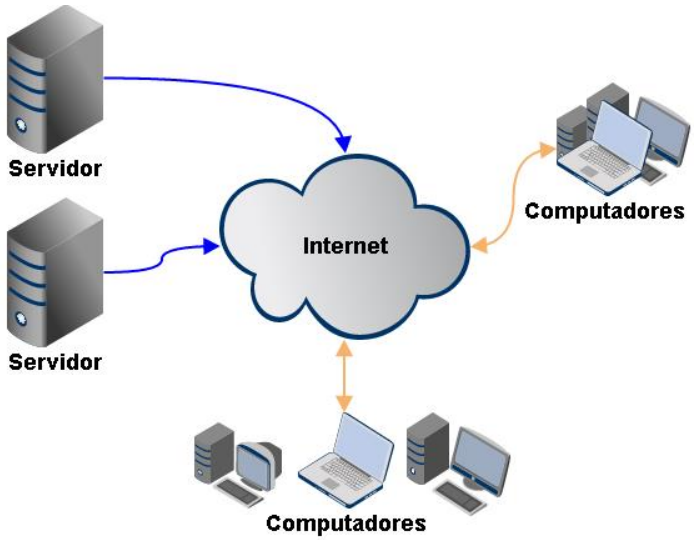

Figura 2 - Representação da Internet

Principais Modelos e Tecnologias [1]

Os principais modelos e tecnologias são:

1. Modelo Cliente-Servidor
2. Protocolo HTTP
3. Estrutura URI

O Modelo Cliente-Servidor (ilustrado na Figura 3):

- **É a estrutura mais comum nas aplicações da Internet** - Inclui transferência de ficheiros, *e-mail* , etc
- **Aplicações "distribuídas"** - consiste num conjunto de processos de aplicação que interagem por intermédio de mensagens
- **Programa Cliente** - programa que funciona num equipamento terminal que solicita e recebe um/a serviço/resposta
- **Programa Servidor** - disponibiliza serviços aos clientes (servidores *Web*, bases de dados, e-mail, etc)

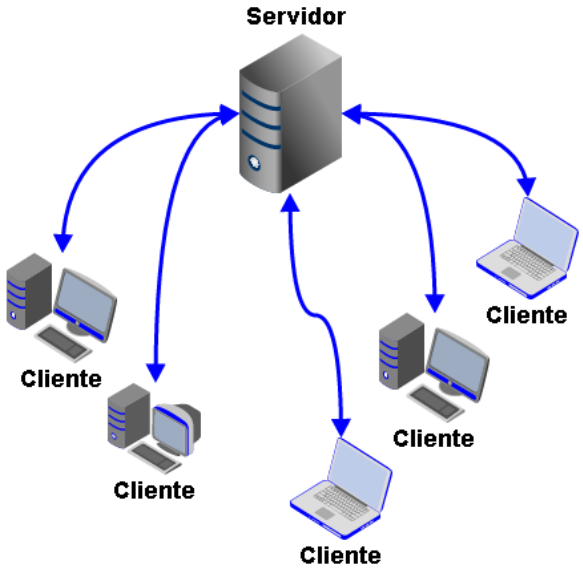

Figura 3 - Demonstração ilustrativa do Modelo Cliente-Servidor

O Protocolo HTTP (ilustrado na Figura 4):

- É o protocolo utilizado por servidores e clientes para transferência de dados tais como texto, sons e imagens na internet
- Define a estrutura e troca de mensagens entre servidor e cliente
- Atualmente, vai na versão HTTP/1.1 incluindo um conjunto de implementações adicionais à versão anterior, como por exemplo, o uso de conexões persistentes, o uso de servidores *proxy* que permitem uma melhor organização da cache, novos métodos de requisições, entre outros

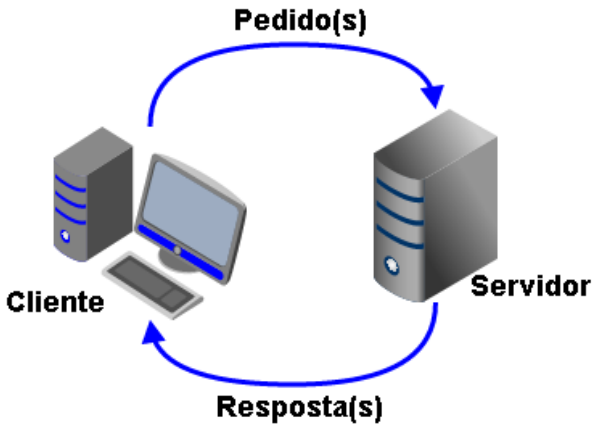

Figura 4 - Exemplo ilustrativo do Protocolo HTTP

A estrutura URI:

É composta por páginas *web*, estando acessíveis através dos seus URL's. O URL é o endereço de um recurso disponível numa rede seja Internet ou Intranet[1] e tem a estrutura demonstrada na Tabela 1.

Tabela 1 - Exemplo de URI com protocolo HTTP

http://www4.dei.isep.ipp.pt/gestmei/index.php	
http:	Protocolo de comunicação
//	Raiz dos endereços de Internet
www4.dei.isep.ipp.pt/	Domínio ou IP onde está alojado o recurso
gestmei/index.php	Caminho local do recurso

Um *Website* é um espaço virtual de uma organização ou pessoa. Tecnicamente é o conjunto de documentos escritos geralmente em linguagem *web*, pertencentes a um mesmo endereço denominado URL, disponível na Internet (*World Wide Web*) que é o conjunto de redes de computadores que se comunicam por meio dos protocolos TCP/IP em todo o mundo. Para desenvolver um *website* é necessário:

- Planeamento prévio acerca do que vai ser desenvolvido e como vai ser desenvolvido
- Definir (antes do desenvolvimento) objetivos, requisitos, estrutura, *design*
- Identificar e desenvolver as fases Análise dos Dados, Modelação da Base de Dados, Arquitetura, Programação do lado Cliente, Programação do lado Servidor, Testes e Avaliação

[1] *Intranet* é um recurso *web* disponível numa rede empresarial.

1.5.2 Linguagens de programação orientada a objetos

A programação orientada a objetos é um paradigma de programação que usa "objetos" que modelam atributos e comportamentos, juntamente com as suas interações.

A programação orientada a objetos consiste em representar o mundo real como uma coleção de objetos que incorporam uma estrutura de dados e um conjunto de operações que manipulam estes dados e trocam mensagens entre si. Os componentes da programação orientada a objetos estão presentes na Tabela 2 [2].

Tabela 2 - Componentes da programação orientada a objetos

Componente	Descrição
Classe	Estrutura que abstrai um conjunto de objetos com características similares. É a partir dela que se criam os objetos utilizados no programa.
Método	Sub-rotina que é executada por um objeto ao receber uma mensagem. É responsável por determinar o comportamento dos objetos de uma classe.
Herança	Mecanismo que permite que características comuns a diversas classes sejam herdadas duma classe base ou superclasse.
Encapsulamento	Princípio pelo qual cada componente de um programa deve agregar toda a informação relevante para a sua manipulação.
Abstração	Processo de extrair as características essenciais de um objeto real.
Polimorfismo	Permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam. Desta forma, um mesmo método pode apresentar várias formas, de acordo com o seu contexto.
Interface	Conjunto de métodos que um objeto pode suportar, mas contendo apenas a especificação da funcionalidade que uma classe deve conter, sem determinar como essa funcionalidade deve ser implementada.
Acoplamento	Grau de dependência entre classes.

Como vantagens deste tipo de programação, podem-se destacar as seguintes:

- **Facilidade na descrição do mundo real através de objetos**
- **O encapsulamento facilita a manutenção do código**
- **Maior facilidade em reutilização do código**

Como desvantagens, podem-se enumerar:

- **Complexidade para quem está a aprender** (comparando com a programação procedimental)
- **Conceitos mais difíceis de aprender**

1.5.3 Sistemas de Gestão de Bases de Dados

Um Sistema de Gestão de Bases de Dados (SGBD) são programas ou conjunto de programas que possibilitam a manipulação de uma base de dados (a nível da inserção, eliminação,

alteração e consulta dos dados). O seu objetivo é gravar e manter a informação que for considerada necessária a quem gere o sistema, disponibilizando-a automaticamente para os mais diversos fins. Através de um SGBD, é possível realizar:

- Operações de registos (Inserção, alteração e eliminação)
- Critérios de visualização de registos (através de v*iews*, por exemplo)
- Operações estatísticas sobre os dados (
- Automatização de funções (*Triggers*)

Uma base de dados possui vantagens na sua utilização, como por exemplo:

- Diminuição de espaço físico ocupado
- Maior integridade dos dados
- Redundância menor
- Maior facilidade na partilha de dados e manutenção

Os SGBD possuem uma arquitetura com 3 níveis, representada na Tabela 3 [3].

Tabela 3 - Níveis de arquitetura dos SGBDs

Nível de arquitetura	Descrição
Físico	Os ficheiros são guardados em suportes de armazenamento informático e depois manipulados pelo SGBD em execução no computador.
Concetual	Organização da informação em tabelas e relacionamentos.
Visualização	Forma como são apresentados os dados aos utilizadores finais, através de interfaces gráficos proporcionados pelo SGBD.

1.5.4 Servidores aplicacionais

"Um Servidor de Aplicações é um servidor que disponibiliza um ambiente para a instalação e execução de certas aplicações, centralizando e dispensando a instalação nos computadores clientes. Os servidores de aplicação também são conhecidos por *middleware*" [4].

O objetivo do servidor de aplicações é disponibilizar uma plataforma que reduza a complexidade de um sistema computacional. No desenvolvimento de aplicações comerciais, por exemplo, o foco dos programadores deve ser a resolução de problemas relacionados com o negócio da empresa e não questões de infraestrutura da aplicação. O servidor de aplicações responde a algumas questões comuns a todas as aplicações, como segurança, garantia de disponibilidade, balanceamento de carga e tratamento de exceções.

1.5.5 Metodologias/Boas práticas e processo de avaliação do desempenho aplicacional

Uma metodologia, em termos da engenharia de *software*, é um conjunto de boas práticas que pode ser seguido e repetido durante todo o processo de criação do *software*, para avaliar o seu desempenho de forma constante e assídua.

O processo de avaliação do desempenho de aplicações recorre a aspetos como métricas, ferramentas e metodologias.

É importante medir o desempenho para saber se a solução proposta é mais vantajosa que outra(s). Para isso, utilizam-se os indicadores de desempenho e as métricas. As componentes deste processo irão respeitar a Figura 5 [5].

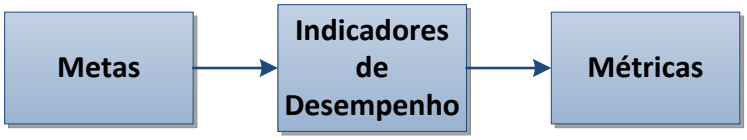

Figura 5 - Componentes das métricas

O item "Metas" permite estabelecer um sistema de medição de desempenho do *software*, baseado em objetivos [5].

Os "Indicadores de Desempenho" são medidas quantitativas que se usa para medir o desempenho face a uma Meta estabelecida. Permite estabelecer referenciais, avaliar o desempenho e melhorar continuadamente [5].

As "Métricas" são medidas quantitativas que, neste caso, ajudam a medir a capacidade e potencialidade do *software*/protótipo.

1.6 Resumo

Neste capítulo foram abordadas temáticas, no âmbito do nivelamento das linguagens web (o que é a internet, modelo cliente-servidor e tecnologias), linguagens de programação orientadas a objetos (componentes, vantagens e desvantagens), sistemas de gestão de bases de dados (operações, vantagens, objetos a atingir com o uso das mesmas e arquiteturas), servidores aplicacionais e metodologias/boas práticas no desempenho aplicacional.

No próximo capítulo será feito um estudo de algumas das linguagens *web* e orientadas a objetos, bem como sistemas de gestão de bases de dados, *frameworks* em JavaScript, arquiteturas possíveis de utilizar para se efetuar os protótipos para a dissertação. Por fim, será feito um estudo ao nível das metodologias/boas práticas no desenvolvimento aplicacional.

2 Tecnologias de Desenvolvimento e Análise de Desempenho

Neste capítulo serão abordadas várias tecnologias de programação tais como linguagens de programação *web* e orientadas a objetos. Seguidamente serão abordados os sistemas de gestão de bases de dados e *frameworks* em JavaScript. Por fim, serão referidas arquiteturas aplicacionais e indicadores de desempenho e métricas.

2.1 Linguagens de programação *web*

As linguagens de programação *web* têm um papel fundamental no desenvolvimento de aplicações *web*, pois trazem benefícios e novos desafios para os programadores. A escolha de uma linguagem de programação *web* pode significar sucesso (caso tenha sido bem escolhida) ou insucesso (caso tenha sido mal escolhida). Deve-se ter a atenção voltada para, por exemplo, se queremos uma linguagem de programação que seja *server-side* ou *client-side*, pois cria impacto no tráfego na rede, bem como do lado dos servidores.

Para que esta escolha seja o mais acertada possível, serão apresentadas 3 linguagens de programação *web*, nomeadamente PHP, JavaScript e ASP.NET, bem como uma descrição acerca das mesmas e as suas vantagens e desvantagens. Destas 3 linguagens, uma delas será utilizada para a camada *front-end* de um dos protótipos.

2.1.1 PHP

O PHP é uma linguagem de programação muito utilizada para gerar conteúdo para a internet. Foi criado em 1995, por Rasmus Lerdorf, e tinha sido inicialmente um pacote CGI[2] para substituir os s*cripts* Perl. O criador do PHP disponibilizou o código fonte para que os utilizadores pudessem aperfeiçoar e melhorar o código (por exemplo: adicionar funções e corrigir eventuais problemas) [6].

A licença de uso e edição é *Open Source*, ou seja, não é possível comercializar qualquer versão modificada do PHP e qualquer modificação que seja efetuada deve continuar com o código fonte aberto para que os utilizadores o possam explorar e modificar. Este sistema de licença não traz lucro aos programadores, pois estes disponibilizam tudo para o público

[2] CGI é um "acordo" entre os servidores HTTP e as aplicações *Web*. Em *background*, o servidor *Web* vai passar uma série de parâmetros para o seu programa e esse programa deve dar uma resposta para o servidor *Web*.

gratuitamente e o público, por sua vez, ajuda ao reportar erros e a modificar o código fonte [7].

Como principais vantagens, o PHP possui [8]:

- **O facto de ser grátis** - não possui encargos para se utilizar
- **Linguagem de programação de fácil aprendizagem** – o PHP tem elementos do Perl, Java e C. Como a maioria dos programadores conhece pelo menos uma destas linguagens, a aprendizagem fica mais facilitada
- **Acesso a bases de dados** – o PHP permite implementar facilmente a ligação entre sistemas compatíveis com o padrão ODBC[3].
- **Multiplataforma** – o PHP funciona em qualquer plataforma onde for possível instalar um servidor *Web* (Linux, Solaris, Windows, etc).

Como desvantagens, o PHP possui:

- **Incompatibilidade entre versões** – falta alguma padronização, pois por exemplo, um comando que funciona em determinada versão do PHP pode não funcionar noutra
- **Documentação incompleta** – os recursos surgem antes de estarem documentados
- **Suporte para datas** – é possível fazer cálculos utilizando datas, mas é mais limitado quando comparado com o ASP, por exemplo.

2.1.2 JavaScript

JavaScript é uma linguagem de programação muito utilizada no desenvolvimento de aplicações para a *Web*. A aparição do JavaScript começou com a criação, pela Netscape, de uma linguagem de criação de *Scripts* executados no servidor [9].

Uma característica do código JavaScript é ser executado do lado do cliente, permitindo libertar recursos do lado do servidor. A grande diferença do JavaScript é que este permite o desenvolvimento do código dentro do código HTML. O programador ao desenvolver o *site*, basta colocar o código "`<script>`" e iniciar a programação em JavaScript (permitindo também código em HTML dentro do código de JavaScript). Para finalizar a programação em JavaScript, basta digitar "`</script>`".

Como principais vantagens, o JavaScript possui [10]:

- **Interatividade com o utilizador** – o JavaScript melhora a interatividade do utilizador em *websites* ao utilizar recursos, como por exemplo caixas *pop-up* e ferramentas de navegação
- **Suporte de imagens como *links*** – permite o uso de imagens como *links* em documentos *Web*. Pode-se utilizar esta vantagem para criar animações, por exemplo
- **Suporta validações do lado cliente** – O JavaScript pode ser validado do lado do cliente. Não é necessário enviar os dados para a validação do servidor. Enviar dados para o

[3] ODBC é um padrão para acesso a sistemas de gestão de bases de dados (SGBD).

servidor para validação, origina mais tráfego, portanto, o JavaScript pode ser usado para reduzir o tempo de validação

- **Suporta identificação do *browser*** – O JavaScript pode automaticamente identificar o tipo e versão do *browser*. É possível utilizar o código JavaScript para enviar comandos diferentes para navegadores diferentes
- **Suporta deteção de *plug-in*** - O JavaScript pode detetar automaticamente os *plug-ins* do *browser*. Pode-se utilizar o JavaScript para facilitar a instalação de *plug-ins* (caso o *browser* do utilizador não o tenha instalado). Se o JavaScript deteta um *browser* sem *plug-ins*, ele ignora os *plug-ins* utilizados no código e modifica a página *Web* para permitir que ela seja executada no *browser*

Após análise das vantagens, serão mencionadas algumas desvantagens do uso do JavaScript [11]:

- **Sobreposição de funcionalidades** – pode apresentar em relação aos estilos CSS alguma sobreposição de funcionalidade, como por exemplo *mouse over* (que pode ser obtido com outro recurso). A aplicação dos dois recursos ao mesmo objeto pode gerar conflito
- **Capacidade de operação limitada, devido a restrições de segurança** – o JavaScript não pode interferir com as configurações do sistema operativo, como por exemplo ativar programas instalados. Também não pode alterar o *layout* de uma página com origem noutro domínio.
- **Capacidade de operação limitada pelo próprio utilizador, nas opções dos *browsers*** – Muitos utilizadores desativam os *scripts* dos seus *browsers* para abrir *pop-ups*, escrita de textos na barra de estado, operar com o botão direito do rato, etc.

2.1.3 ASP.NET

O ASP.NET é uma tecnologia de desenvolvimento de aplicações *Web*, que foi lançada oficialmente em 2002 juntamente com a *framework .NET* 1.0. Na altura era conhecido como ASP+ por ser o sucessor direto do ASP 3.0 (Clássico). Foi o primeiro *framework* de desenvolvimento *web* da Microsoft.

As diferenças entre o ASP e o ASP.NET estão descritas na Tabela 4 [12].

Tabela 4 - Diferenças entre o ASP e o ASP.NET

ASP	ASP.NET
Código interpretado	Código compilado
Código misturado (HTML com VbScript)	Código separado (ASPX e CodeBehind)
Linguagem VbScript	Linguagens (Visual C#, Visual Basic)
Menos seguro e menos rápido	Mais seguro e mais rápido
	Rico em controlos que facilitam criação da página
	Possui *Master Page*

Como principais vantagens, o *ASP.NET* possui [13]:

- **Compilação antes da execução** - o que origina um ganho de performance
- **Distribuição gratuita, em conjunto com o *Windows* –** o que não implica compra de novo *software*
- **Controlos a nível de alta produtividade adicionais** - como por exemplo o "*drag-drop*"
- **Facilidade na utilização de componentes** - basta copiar .DLLs (bibliotecas dinâmicas) de componentes para utilizá-los, sem fazer grandes configurações
- **Facilidade de aprendizagem** - o programador pode escolher uma das mais de 30 linguagens que a plataforma suporta;

No entanto, também possui desvantagens, nomeadamente [13]:

- **Apenas as linguagens VB.NET e C# são nativas** - as outras são adicionadas via componentes
- **Portabilidade** - Só é executada em algumas versões do *Windows* e existe a necessidade do *.NET framework* e servidor *IIS*;
- **Licença paga e não *open source***

2.1.4 Linguagem *web* adotada

Após a análise das diversas linguagens de programação *web*, ir-se-á optar por se utilizar o JavaScript para a elaboração de um dos protótipos, pois para além de ser uma linguagem que não possui custos de utilização, suporta validações do lado cliente (o que alivia a carga na rede e no servidor) e normalmente é compatível com diversos *browsers* sem precisar de *plug-ins* adicionais.

Uma vez que as validações são executadas do lado do cliente, existe um alívio de carga na rede e no servidor. Isto será particularmente útil para os testes que se irão fazer na secção 3.4, a fim de os tornar mais credíveis e menos aleatórios, pois evita que erros de inserção de dados afectem os tempos de resposta.

2.2 Linguagens de programação orientadas a objetos

No presente tópico, será apresentada uma abordagem às linguagens de programação orientadas a objetos (Java, C++ e C#), bem como as suas vantagens e desvantagens. A linguagem de programação orientada a objetos terá todo o interesse no âmbito desta dissertação, pois será a linguagem utilizada nos 2 protótipos desenvolvidos.

Um dos protótipos terá o *middle-tier* e *back-end* implementados com uma linguagem de programação orientadas a objetos, enquanto que o outro protótipo será integralmente

desenvolvido nessa linguagem. Portanto, ao efetuar a escolha referida na secção 2.2.4 ter-se-á que ter em conta a linguagem mais vantajosa que permita concretizar ambos os cenários.

2.2.1 Java

Java é uma linguagem de programação orientada a objetos que começou a ser criada em 1991, na *Sun Microsystems* [14]. Possui características ao nível da portabilidade, pois é independente do sistema operativo. A nível de recursos de rede, possui uma extensa biblioteca que facilita a cooperação com protocolos TCP/IP (como HTTP e FTP). Por outro lado, a nível de segurança, pode executar programas via rede com restrições de execução [15].

Para além das características descritas anteriormente, possui vantagens, tais como [16] [17]:

- **Simplicidade na especificação** - tanto da linguagem como do "ambiente" de execução (JVM[4])
- **É distribuída com um vasto conjunto de bibliotecas (ou APIs)**
- **Possui suporte para a criação de programas distribuídos e multitarefa** - múltiplas linhas de execução no mesmo programa
- **A memória é libertada automaticamente** - feita pelo *garbage collector* do Java
- **O código é carregado dinamicamente** – os programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

Apesar das vantagens referidas anteriormente, o Java também possui limitações, como por exemplo [17]:

- **A pré-compilação exige tempo** - o que faz com que programas em Java demorem um tempo significativamente maior para começarem a funcionar. Não é de todo uma desvantagem para programas que são executados em servidores, mas para programas clientes, já se torna mais problemático
- **Reverse engineering** - os *bytecodes* produzidos pelos compiladores Java podem ser utilizados num processo de *reverse engineering* para a obtenção do *source code*
- **O Java não suporta herança múltipla, nem apontadores** - que são aceites em C++, o que pode ter um pequeno impacto na produtividade do programador.

2.2.2 C++

C++ é uma linguagem de programação baseada na linguagem C. O desenvolvimento da linguagem começou na década de 80, por Bjarne Stroustrup. O objetivo do desenvolvimento desta linguagem era melhorar uma versão do *kernel* Unix. Para desenvolver a linguagem,

[4] JVM é um programa que carrega e executa os aplicativos Java, convertendo os *bytecodes* (resultado da compilação do código Java para uma forma intermediária de código) em código executável de máquina.

foram acrescentados elementos de outras linguagens, na tentativa de criar uma linguagem com elementos novos, sem trazer problemas para a programação. No início do desenvolvimento, a linguagem utilizava um pré-processador, mas o criador da linguagem criou um compilador próprio, com novas características [18].

O C++ é uma linguagem criada para ser tão eficiente quanto o C, porém com novas funções e suporta múltiplos paradigmas. Não é necessário um ambiente de desenvolvimento muito elaborado para o desenvolvimento de C++.

Alguns dos programas mais conhecidos ou parte do seu código são feitos em C++, por exemplo, Adobe Photoshop, Mozilla Firefox, Internet Explorer, MySQL, Microsoft Windows, etc.

Como principais vantagens, o C++ possui:

- **Produção de código mais eficiente** – adequado para programação de alto e baixo nível
- **Adequado para grandes projetos** – devido à sua alta flexibilidade, portabilidade e consistência
- **Não está sob o domínio de uma empresa** – como acontece com o Java com a Oracle ou o Visual Basic com a Microsoft
- **Compatibilidade com o C** – como já foi dito anteriormente, o C++ é uma linguagem de programação baseada na linguagem C

No entanto, também possui desvantagens, nomeadamente:

- **Herança dos problemas de compreensão de sintaxe do C** - por ter compatibilidade com o C
- **Os compiladores atuais nem sempre produzem o código mais otimizado** - tanto em velocidade como em tamanho
- **Tempo de aprendizagem superior**
- **A biblioteca padrão não cobre áreas importantes da programação** - como *threads*, conexões TCP/IP, interface gráfica e manipulação de sistemas de ficheiros, o que implica a necessidade de criação de bibliotecas próprias para tal, que pecam por terem portabilidade limitada

2.2.3 C#

C# é uma linguagem de programação orientada a objetos, que foi desenvolvida pela Microsoft e faz parte da *framework* .NET. O C# teve por base a linguagem C++ e alguns elementos da linguagem Pascal e Java.

O C# foi criado especificamente para .NET, sendo que muitas outras linguagens têm suporte com C#. Algumas destas linguagens são VB.NET, C++ e J#. Embora a linguagem C# seja considerada muito semelhante ao Java, existem algumas diferenças, descritas na Tabela 5 [19].

Tabela 5 - Diferenças entre o Java e o C#

Diferenças	Java	C#
Função de implementar propriedades e/ou sobrecarga de operadores	Não	Sim
Função para implementar um modo não seguro (que pode ser utilizado para manipulação de apontadores e aritmética)	Não	Sim
Exceções verificadas	Sim	Não
Documentação automática	Javadoc	XML
Suporte a indexadores	Não	Sim

Como principais vantagens, o C# possui [20]:

- **Recursos do ambiente Windows** – sistema operativo mais utilizado no mundo
- **Não é preciso registar componentes** – pois já provêm do ambiente Windows
- **Validação de dados e tratamento de erros**
- **SOAP / XML** - tecnologias de interação com outras plataformas
- **Instalação de ficheiros**

No entanto, também possui desvantagens, nomeadamente [20]:

- **Os programas e componentes antigos devem ser reescritos**
- **Difícil aprendizagem** - para programadores mais rigorosos, devido à maior possibilidade de se gerar código menos robusto

2.2.4 Linguagem adotada

Após a análise das diferentes linguagens de programação orientadas a objetos, ir-se-á utilizar a linguagem de programação Java na implementação dos protótipos. As razões cruciais que levaram à escolha da linguagem foram: a sua portabilidade, as API's existentes e por ser uma linguagem que não exige a compra de *software* para efetuar o desenvolvimento.

Como acesso à base de dados, num dos protótipos, será utilizado também o C++ por ser baseado em C, permitindo acessos à base de dados e a troca de informação serem mais rápidos e fluidos.

2.3 Sistemas de Gestão de Bases de Dados

Nos dois protótipos implementados, será utilizado o mesmo sistema de gestão de base de dados, para que os resultados dos testes não sejam condicionados pelo SGBD. Apesar dos protótipos não manipularem grandes quantidades de dados, irão ser analisados sistemas de gestão de base de dados robustos, de forma a ter os melhores tempos de resposta, visando

focar mais a arquitetura aplicacional e a escolha de linguagens que propriamente o sistema de gestão de bases de dados.

No presente tópico, serão abordadas os sistemas de gestão de base de dados, bem como as vantagens e desvantagens de diferentes sistemas de gestão de bases de dados (Oracle, PostgreSQL e Microsoft SQL Server).

2.3.1 Oracle

O Oracle é um sistema de gestão de base de dados que surgiu no fim dos anos 70, quando Larry Ellison encontrou uma descrição de um protótipo funcional de uma base de dados relacional e descobriu que nenhuma empresa se tinha empenhado em comercializar essa tecnologia [21].

Como principais vantagens, o Oracle possui [22]:

- **Grande otimização de performance para grandes quantidades de dados**
- **Robustez e segurança dos dados**
- **Possibilidade do carregamento de diversos tipos de dados binários** - imagens, filmes, sons, etc
- **Sistema multiutilizador** - permitindo a consulta simultânea de dados por diversas pessoas
- **Permite intercâmbio com diversas tecnologias** - programação de interfaces em linguagens de programação como VB, C, Java, etc

No entanto, também possui desvantagens, nomeadamente [22]:

- **Exige especialização técnica para administração da base de dados** – em tarefas como *backup*/*recovery*, afinação de desempenho, gestão de utilizadores/segurança, etc
- **Alto custo das licenças e do *hardware* necessário para correr os *softwares***

2.3.2 PostgreSQL

PostgreSQL é um Sistema de Gestão de Base de Dados Objeto Relacional (SGBDOR) desenvolvido como projeto *open source*.

Como principais vantagens, o PostgreSQL possui [23]:

- **Incorpora uma vasta variedade de linguagens procedimentais**
- **Possibilidade de criação de *queries* SQL a partir de linguagem C**
- **É extensível ao SQL**
- **Suporte a *subqueries* e *JDBC* para Java e PHP**

No entanto, também possui desvantagens, nomeadamente [23]:

- **Não tem parametrizações por defeito**

- **Não tem suporte nativo para criar procedimentos baseados em *web***
- **Para obter ajuda, pode implicar consultar uma *mailing list***

2.3.3 Microsoft SQL Server

O *Microsoft SQL Server* é um Sistema de Gestão de Base de Dados Relacionais (SGBDR) que funciona unicamente sobre o sistema operativo *Windows* [24].

Foi originalmente baseado no *Sybase SQL Server X* versão 4.2. A partir da versão 6 do *SQL Server*, a Microsoft implementou modificações que visavam dotá-lo de características multitarefa do *Windows NT* [24].

Como principais vantagens, o *Microsoft SQL Server* possui [25]:

- **Excelente suporte para recuperação de dados** – o *Microsoft* S*QL Server* tem uma série de características que promovem o restauro e recuperação desses dados. Embora que tabelas individuais não possam ser copiadas ou restauradas, existem opções completas de restauro da base de dados que podem ser úteis nestes casos (exemplo: *logs*, *cache* e *backups*).
- **Software de gestão de alto nível** - o *Microsoft SQL Server* inclui *softwares* de gestão de base de dados tanto para nível profissional como empresarial. Alguns concorrentes, como o MySQL, desenvolveram *softwares* semelhantes nos últimos anos, mas o *Microsoft SQL Server* é mais fácil de usar e tem mais recursos. Os *triggers*, por exemplo, têm total suporte para produtos *Microsoft*. O *software* oferecido pela *Microsoft* também oferece integração com a *framework* .NET, o que não é o caso de produtos concorrentes.

No entanto, também possui desvantagens, nomeadamente [25]:

- **Custo** - Uma das principais desvantagens de se utilizar o *Microsoft SQL Server* em vez de um sistema de gestão de base de dados relacional alternativo, é que as opções de licenciamento são muito caras. Apesar do uso do *software* para fins académicos ou desenvolvimento serem gratuitos, qualquer tipo de uso comercial resulta numa taxa de licenciamento.
- **Compatibilidade limitada** - O *Microsoft SQL Server* só é projetado para ser executado em servidores W*indows*.

2.3.4 Base de dados adotada

Após a análise dos diversos sistemas de gestão de base de dados, optou-se por se utilizar o Oracle. A escolha baseou-se no facto de ser um produto que possui mais recursos de segurança e performance, permitindo desta forma analisar de forma isenta, que protótipo é mais vantajoso (pois o sistema de gestão de base de dados é rápido e fluido). Para além desta

vantagem, é multiplataforma, ou seja, não é dependente de um sistema operativo específico, o que permite mais liberdade de escolha.

2.4 Frameworks em JavaScript

Quando o objetivo é criar *sites* que tenham boa capacidade de resposta, manutenção fácil, entre outras funcionalidades, é muito difícil e trabalhoso de se conseguir fazer manualmente. É nessa altura que entram as *frameworks*, para facilitar o trabalho do programador.

No presente tópico, serão abordadas algumas das *frameworks* em JavaScript (ExtJS 4, jQuery e Dojo Toolkit), bem como as vantagens e desvantagens, devido à escolha referida na secção 2.1.4.

2.4.1 ExtJS 4

ExtJS é uma *framework* JavaScript com uma gama enorme de *widgets*[5] para aplicações *web*. Existem diversos temas prontos a utilizar, alguns fornecidos pelo próprio *site* da ExtJS e outros facilmente encontrados pela internet, além da flexibilidade fornecida pela *framework*, para a extensão dos *widgets* já disponíveis [26].

Esta *framework* possui algumas vantagens, nomeadamente [26]:

- **Suporte a diversos *browsers*** - os *widgets* seguem estritamente os padrões *web* e os temas fornecem um bom suporte em *browser* como Internet Explorer, Firefox, Opera, entre outros
- **A quantidade de componentes que possui** - pois é extremamente raro existir a necessidade de se implementar de raiz um novo componente
- **Excelente documentação e exemplos *offline***

No entanto, possui algumas desvantagens, como por exemplo [26]:

- **Sintaxe mais extensa** - que o jQuery, por exemplo, quando queremos utilizar ou estender os componentes
- **Ajuda da comunidade (a nível de fóruns)** - pois é possível obter as respostas que pretendemos, mas nota-se que os programadores mais experientes são um pouco impacientes com os menos experientes
- **A *framework* é um pouco complexa numa fase inicial**

2.4.2 jQuery

jQuery é uma *framework* para a linguagem JavaScript, ou seja, um produto que simplifica a vida para programar nessa linguagem, conforme referido no início da secção 2.4. Esta

[5] Widgets são componentes de interface do utilizador.

framework oferece uma infraestrutura com a qual se terá mais facilidade para a criação de aplicações complexas do lado do cliente (*client-side*) [27].

Como principais vantagens, podemos referir [28]:

- **Disponibilização de ajuda em qualquer item que se esteja a programar** - na criação de interfaces de utilizador, efeitos dinâmicos, aplicações que utilizam Ajax, etc
- **Compatibilidade em todos os *browsers*** - ao programar em JavaScript utilizando jQuery, é disponibilizada uma interface para programação que permite fazer correr em todos os *browsers*, sem problemas de compatibilidade
- **Uso gratuito da *framework*** - seja para fins comerciais ou pessoais e é possível fazer o *download* da própria página *web* e começar a utilizar

No entanto, o jQuery também apresenta algumas desvantagens [28]:

- **As aplicações precisam de um servidor para criar e gerir sessões**
- **É difícil proteger o código-fonte**
- **As aplicações precisam de outro aplicativo para fornecimento de dados** – sendo este escrito noutra linguagem

2.4.3 Dojo Toolkit

O Doko Toolkit é uma *framework* JavaScript que fornece mais produtividade para o desenvolvimento aplicacional. Os recursos que esta possui proporcionam um código mais legível e menos extenso [29].

O Dojo Toolkit está dividido em vários *packages* que irão constituir uma completa distribuição do Dojo Toolkit. Estes *packages* são [29]:

- **Dojo** – conhecido como "núcleo" é a parte principal do Dojo e é onde estão contidos os packages e módulos mais gerais. O núcleo cobre uma longa rede de funcionalidades como AJAX, manipulação DOM, programação por tipo de classe, eventos, *data stores*, *drag-drop* e bibliotecas de internacionalização;
- **Dijit** – um extenso grupo de *widgets* e o sistema subjacente para apoiá-los. É feito totalmente sobre o núcleo Dojo.
- **DojoX** – uma coleção de *packages* e módulos que fornecem uma vasta gama de funcionalidades que são construídos sobre o núcleo Dojo e o Dijit. *Packages* e módulos contidos no DojoX terão variados níveis de maturidade (e estão descritos nos ficheiros "*Readme*" de cada *package*). Alguns dos módulos são extremamente maduros e alguns outros altamente experimentais.
- **Util** – são várias ferramentas que dão suporte para o resto do *toolkit*, como sendo possível construir, testar e documentar o código.

As vantagens de se usar o Dojo Toolkit são as seguintes [30]:

- **Vários *widgets fornecidos***- para desenvolver a interface do utilizador para aplicações web
- **É uma *framework* robusta** - que pode ser utilizada para desenvolver aplicações de nível empresarial

Como desvantagens, o Dojo Toolkit possui [30]:

- **O programador está dependente do suporte para o Dojo do *browser***
- **Não há forma de esconder o código do Dojo** - esta característica faz com que a utilização desta *framework* não seja adequada para o desenvolvimento de aplicações comerciais.

2.4.4 Conclusões

Após estudo de diversas *frameworks* em JavaScript, ir-se-á utilizar a *framework* ExtJS 4, devido a:

- **Efeitos visuais**, que são obtidos através de elementos de manipulação (aparecer/desaparecer, redimensionar, mover, etc)
- **Manipulação Json**, que maximiza a manipulação de dados a serem utilizados para fornecer informações dinamicamente em *run-time* (muito útil para a implementação desenvolvida)
- **Possuir uma biblioteca *offline* e repleta de exemplos-** que poderá ser uma mais-valia na implementação do protótipo

2.5 Arquiteturas aplicacionais

É necessário que haja uma boa separação dos dados entre as diversas camadas das aplicações (desde o *front-end* até ao *back-end*). Desta forma, a flexibilidade aumenta e permite o desenvolvimento de novos serviços, favorecendo a reutilização de componentes e aumentando a facilidade de integração dos novos serviços nos sistemas informáticos.

Neste tópico irão ser abordadas 3 tipos de arquitetura aplicacional: Arquiteturas 3-Tier, SOA e Monolíticas, bem como as suas vantagens e desvantagens.

Os 2 protótipos desenvolvidos irão ter arquiteturas aplicacionais diferentes, pois o objetivo principal da dissertação é articular várias linguagens de programação.

2.5.1 Arquiteturas 3-Tier

As Arquiteturas 3-Tier são também conhecidas por *client-server*. Caracterizam-se pela divisão em 3 módulos com funcionalidades distintas: UI (*interface* do utilizador), lógica do processo

funcional ("regras de negócio") e armazenamento/acesso de dados. Estes módulos usam interfaces de comunicação entre eles o que permite serem desenvolvidos, mantidos e testados em separado [31]. Na Figura 6 é possível observar a sua separação.

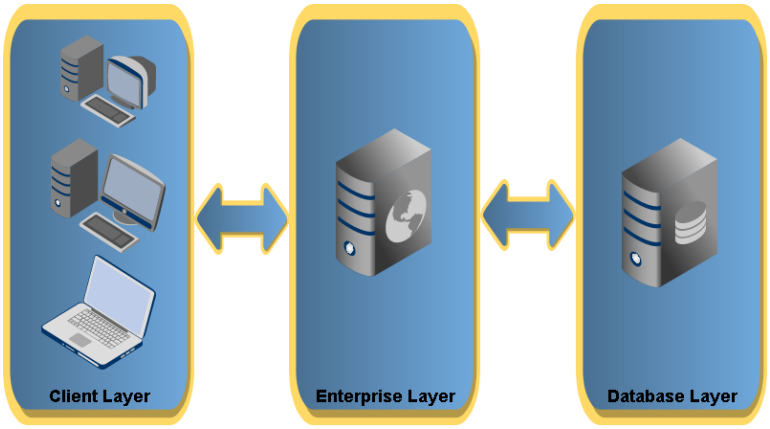

Figura 6 - Divisão típica de módulos de uma arquitetura 3-tier

2.5.2 Arquiteturas Service-Oriented

As Arquiteturas Service-Oriented (SOA) são semelhantes às referidas na secção 2.5.1, encontrando-se a diferença no facto de que cada aplicação é desenvolvida como um serviço para ser consumido por outro, podendo-se construir uma árvore de dependências entre serviços, tal como ilustrado na Figura 7. Deste modo, uma arquitetura 3-tier pode ser vista como uma SOA em que cada módulo tem no máximo uma dependência [31].

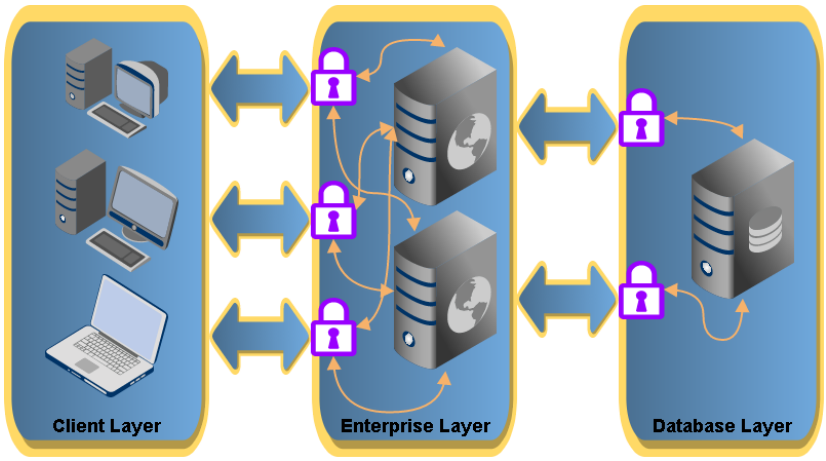

Figura 7 - Estrutura de um SOA

Normalmente, estes serviços usam uma comunicação baseada em XML, devido à estruturação e organização da linguagem, e em *XSD*, para validação de mensagens.

2.5.3 Aplicações Monolíticas

São arquiteturas de apenas uma camada. Isto quer dizer que, o código dos vários módulos (*Client Layer*, *Enterprise Layer* e *Database Layer*) estão interligados. Este tipo de arquitetura tem a vantagem de ser independente de outras aplicações e serviços e de reduzir o número de erros provenientes de aplicações ou serviços externos, pois não tem nenhuma ou quase nenhuma dependência (eventualmente a nível da base de dados pode ter alguma dependência, pois será sempre feito num *software* externo ao que a aplicação foi desenvolvido). Na Figura 8 é demonstrada a fusão entre as diversas camadas da aplicação [31].

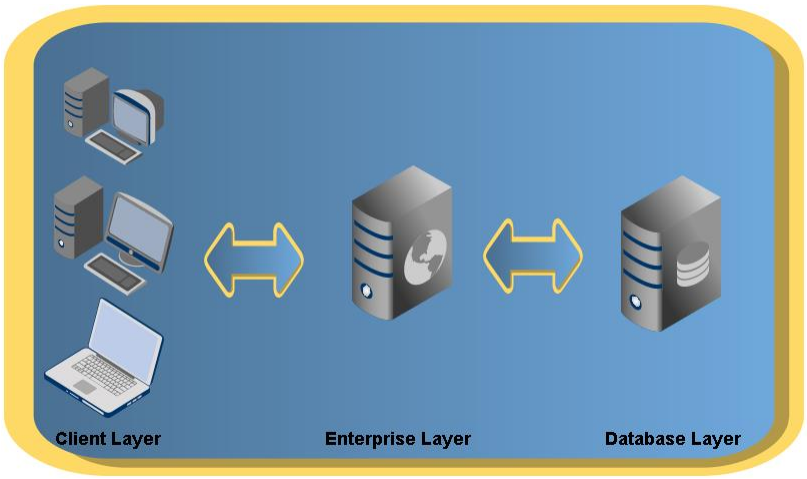

Figura 8 - Arquitetura monolítica

2.5.4 Arquiteturas adotadas

Após análise das várias arquiteturas existentes, foi decidido que a implementação de um dos protótipos seria uma arquitetura *3-Tier* e outra seria Monolítica.

A razão da escolha da arquitetura *3-Tier* foi a distribuição do protótipo por camadas o que torna fácil de efetuar manutenção devido ao baixo acoplamento entre serviços e por ser uma arquitetura que é baseada no uso de padrões de desenvolvimento aplicacional, como por exemplo, *Modal-View-Controller* (MVC).

A escolha da arquitetura deveu-se a ser uma condição fulcral para o âmbito desta dissertação, pois pretende-se analisar as vantagens do desenvolvimento aplicacional em multilinguagem de programação versus desenvolvimento aplicacional numa única linguagem de programação.

2.6 Abordagens para definir objetivos mensuráveis

A razão principal para efetuar medidas em *software* é obter dados que auxiliem o controlo a nível de planeamento, custo e qualidade do *software*. É importante ser-se capaz de contar e medir constantemente, pois a medição é necessária para verificar o cumprimento dos

requisitos funcionais (que descrevem a funcionalidade ou serviços do sistema) e não funcionais (que definem propriedades e restrições do sistema).

Um requisito funcional poderá ser "o sistema disponibilizará ao utilizador um visualizador apropriado para a leitura de documentos". Este requisito é ambíguo, pois o utilizador poderá ter uma forma de considerar um "visualizador apropriado" e o programador do sistema ter outra. Isto origina problemas e uma forma de os contornar será através da medição, por exemplo, da satisfação do utilizador aquando da utilização do visualizador do sistema.

Um exemplo de um requisito não funcional é "utilizadores experientes, devem ser capazes de utilizar todas as funcionalidades do sistema após duas horas de formação. Após a formação, o número médio de erros cometidos não pode exceder 2 erros por dia". Como foi possível constatar, introduziu-se uma medida para validar se a formação serviu para que os utilizadores, ao utilizar o sistema, não errem em média mais de duas vezes por dia.

A razão fundamental para a medição do processo de desenvolvimento aplicacional é obter dados que nos auxiliem a controlar melhor o cronograma, custo e qualidade dos produtos aplicacionais. É importante serem capazes de contar e medir, de forma consistente, entidades básicas que são diretamente mensuráveis, tais como tamanho, defeitos, esforço e tempo (cronograma).

Medições consistentes fornecem dados para:

- Expressar quantitativamente requisitos, metas e critérios de aceitação
- O acompanhamento dos progressos e antecipação de problemas
- Quantificar compensações utilizadas na alocação de recursos
- Prever os atributos do *software* para planeamento, custo e qualidade do mesmo.

Para estabelecer e manter o controlo sobre o desenvolvimento e manutenção de um produto aplicacional, é importante que os programadores meçam os problemas encontrados no *software* (sejam defeitos ou problemas) para determinar a ação corretiva a tomar, de forma a medir e melhorar o processo de desenvolvimento de *software* e, dentro do possível, prever outros defeitos.

A seguir, irão ser apresentadas *frameworks* que permitem medir os protótipos desenvolvidos nesta dissertação, nomeadamente *Goal-Question-Metrics* (GQM), *Quality Function Deployment* (QFD) e *Software Quality Metrics* (SQM).

2.6.1 Goal-Question-Metrics (GQM)

Muitos programas de métricas de *software* não foram bem-sucedidos porque possuíam objetivos pouco definidos ou até mesmo inexistentes. Para combater este problema Vic Basili e os seus colegas da Universidade de Maryland desenvolveram uma rigorosa abordagem orientada à medição. Devido à sua natureza intuitiva, a abordagem tornou-se apelativa, onde

a ideia fundamental é que os gestores procedam de acordo com as três fases apresentadas em baixo [32] :

1. Definir objetivos específicos para as necessidades em termos de propósito, perspetiva e meio ambiente
2. Refinar os objetivos em questões quantificáveis
3. Deduzir as métricas e dados a serem recolhidos (e os meios para recolhê-los) de forma a responder às questões feitas no ponto anterior

É apresentada na Figura 9 um exemplo de 3 objetivos, no qual o "Objetivo 1" possui só uma questão ("Questão A") e 1 métrica ("Métrica 1A"). Já o "Objetivo 2" possui 2 questões ("Questão B" e "Questão C") e 1 métrica para cada questão ("Métrica 2B" para a "Questão B" e "Métrica 2C" para a "Questão C"). Por fim, o "Objetivo 3" tem uma questão ("Questão D") e duas métricas para a mesma questão ("Métrica 3D1" e "Métrica 3D2").

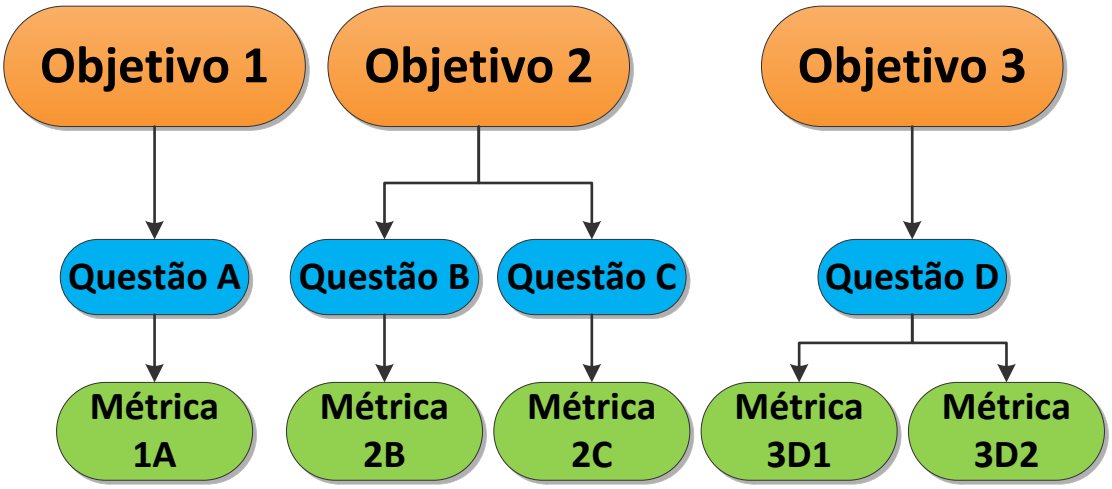

Figura 9 - Exemplo GQM

2.6.2 Quality Function Deployment (QFD)

Quality Function Deployment (QFD) é uma metodologia para a construção da "Voz do Cliente" na concepção de produtos e serviços. É uma ferramenta de equipa que capta os requisitos do cliente e os traduz em características sobre um produto ou serviço. As origens do QFD vêm do Japão. Em 1966, os japoneses começaram a formalizar o que Yoji Akao ensinava em QFD [33].

O QFD desde a sua introdução, tem ajudado a transformar a forma como as empresas [33]:

- Planeiam novos produtos
- Desenham requisitos do produto
- Determinam as características do processo
- Controlam o processo de fabrico
- Documentam as especificações de produtos existentes.

O QFD utiliza alguns princípios de Engenharia Concorrente (paralelização de tarefas) em que as equipas todas (ou praticamente todas) estão envolvidas em todas as fases de desenvolvimento do produto. Cada uma das quatro fases de um processo de QFD utiliza uma matriz para traduzir os requisitos do cliente em estados iniciais de planeamento em controlo de produção. Cada fase ou matriz representa um aspeto mais específico de requisitos do produto. Relações binárias entre elementos são avaliadas para cada fase. Apenas os aspectos mais importantes de cada fase são encaminhados para a próxima matriz.

A metodologia QFD é implementada através das seguintes etapas sequenciais [34]:

1. O "*Objective Statement*" - identificação do cliente, necessidades e objetivo da equipa QFD
2. Os "*Whats*" - identificação das características do produto e/ou serviços desejados pelo cliente
3. Os "*Hows*" - identificação das formas de alcançar os "*Whats*".

Estes passos são aplicados em todas as fases sequenciais do ciclo de desenvolvimento do produto/serviço: planeamento, *design*, processo e produção.

Para cada fase, é utilizada uma matriz (à semelhança da apresentada na Figura 10) para mapear a partir do desejado, as características ("*Whats*") para as opções a atender a essas características ("*Hows*").

Por exemplo, na fase 1 (planeamento) os "*Whats*" são os requisitos do cliente e agem como as entradas da matriz. As saídas de matriz para a fase 1 são os "*Hows*" e são as especificações do *design* do produto/serviço. Estas saídas da matriz da fase 1 (os "*Hows*"), por sua vez, tornam-se a fase 2 da entrada da matriz, os "*Whats*" (conforme apresentado na Figura 11). Esta abordagem sequencial continua, tendo como resultado as exigências de produção das saídas (ou "*Hows*") da fase 4 (produção).

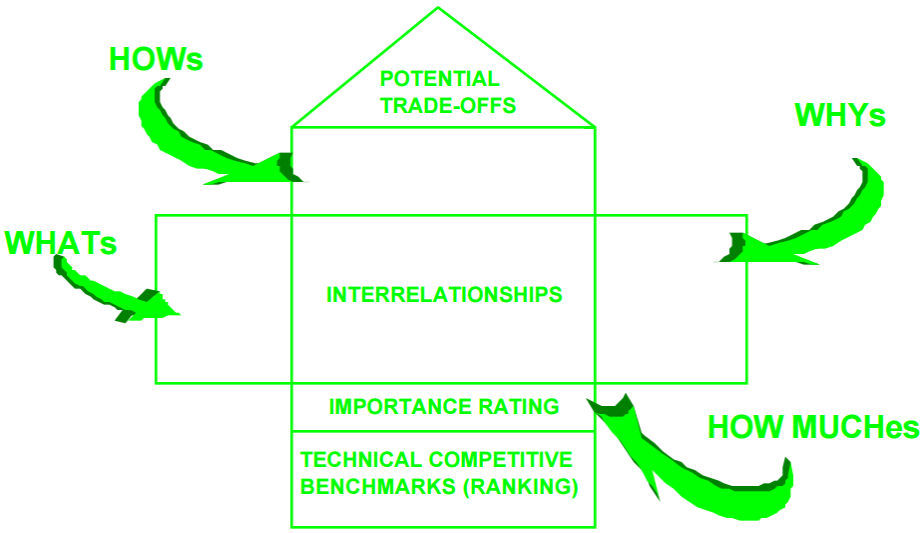

Figura 10 - Matriz QFD [34]

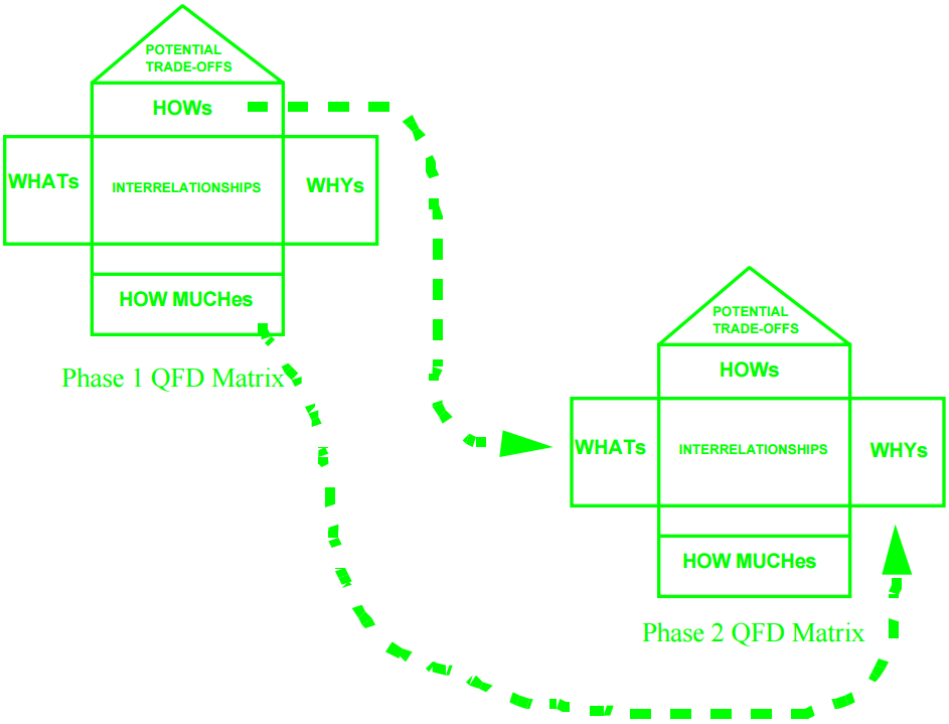

Figura 11 - *Flowdown* matriz QFD [34]

2.6.3 Software Quality Metrics (SQM)

Esta abordagem foi exemplificada por McCall e foi desenvolvida para permitir que o cliente avaliasse um produto ao ser desenvolvido por uma entidade. Neste caso, é definido um conjunto de fatores de qualidade no produto final; os factores são refinados num conjunto de critérios, que são ainda mais refinados (posteriormente) num conjunto de métricas. Essencialmente, este é um modelo para a definição de atributos de qualidade do produto externos.

Este modelo tem três grandes perspectivas para a definição e a identificação qualitativa de um produto/*software* (apresentadas na Figura 12) [35]:

- Revisão (capacidade de sofrer alterações)
- Transição (capacidade de adaptação a novos ambientes)
- Operações (as suas características de operação).

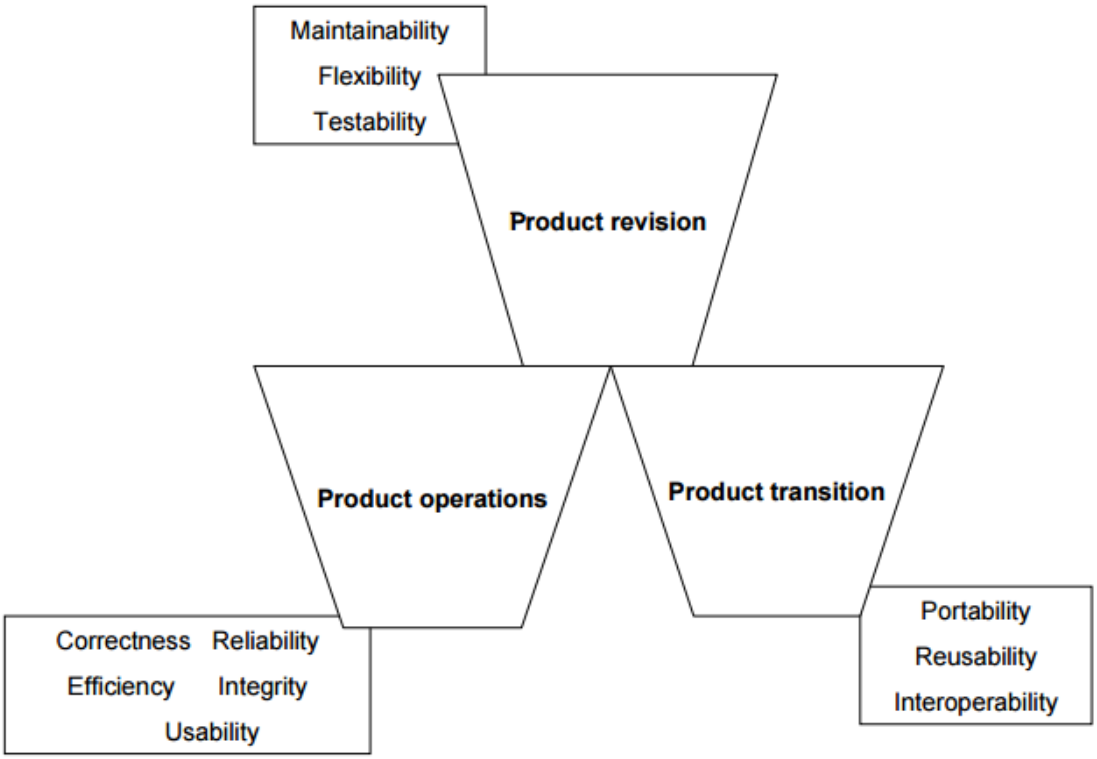

Figura 12 - Modelo qualitativo de McCall (SQM) [35]

Este modelo também detalha os três tipos de características de qualidade (grandes perspectivas) numa hierarquia de fatores, critérios e métricas, conforme apresentado na Figura 13 [35]:

- 11 Fatores (para especificar) - descrevem a visão externa do *software*, como é visto pelos utilizadores (lado esquerdo da Figura 13)
- 23 Critérios de qualidade (para construír) - descrevem a visão interna do *software*, ou seja, como é visto pelo programador (lado direito da Figura 13)
- Métricas (para controlar) - são definidas e usadas para fornecer uma escala e um método para a sua medição.

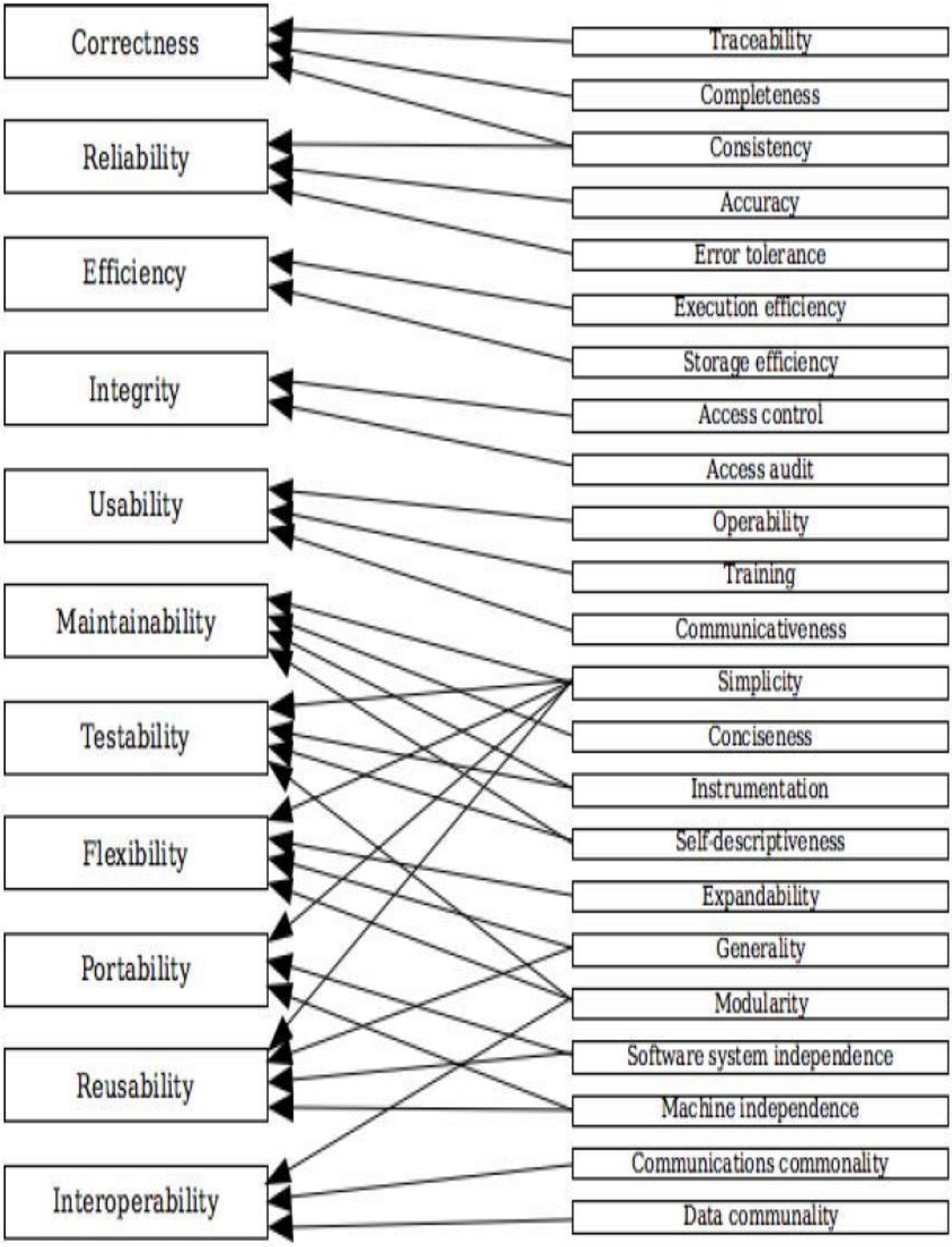

Figura 13 - McCall's Quality Model (SQM) [36]

2.6.4 Conclusões

Após a análise das abordagens apresentadas, ir-se-á utilizar a abordagem GQM, pois é mais simples e retrata perfeitamente o que se pretende analisar na presente dissertação, ao nível dos protótipos desenvolvidos.

No seguimento da abordagem selecionada, serão utilizadas os objetivos, questões e métricas, apresentados na Tabela 6, para avaliar a nível quantitativo e qualitativo, o desempenho dos protótipos.

Tabela 6 - Abordagem GQM aplicada no âmbito da dissertação

Objetivos	Questões	Métricas
Melhor desempenho entre os protótipos desenvolvidos	Qual o tempo necessário para efetuar consulta(s), inserção(ões) ou atualização(ões) de dados?	Tempo(s) de execução do(s) método(s) respetivo(s) desde o pedido do utilizador, até o retorno de dados ou confirmação da operação.
		Tempos médios e desvios-padrão (em ms).
Melhor custo para a realização de uma tarefa de programação	Quanto tempo demorou a fazer um protótipo e outro?	Tempo (em dias).
Melhor código	Quantas linhas possui cada protótipo?	Número de linhas.
	Quantas classes possui cada protótipo?	Número de classes.
Melhor custo-benefício	Qual o custo duma implementação e da outra?	Número de servidores necessários.
	Quantas pessoas irão utilizar os protótipos em ambiente empresarial?	Número de utilizadores.

3 Caso de Estudo

Como já foi referido anteriormente, o objetivo principal desta dissertação consiste em analisar se é mais vantajoso articular várias linguagens de programação ou implementar uma única linguagem de programação. Assim, surgiu a ideia de criar 2 protótipos que cumprissem os requisitos mencionados. Desta forma, neste capítulo vão ser apresentados 2 protótipos, um em versão *Web* e outro em versão de aplicação Java. Estes protótipos vão ser analisados e comparados a nível de indicadores de desempenho e comparados, usando-se para o efeito diferentes métricas.

Os protótipos concebidos têm em vista cumprir requisitos a nível de lógica de negócio e a nível da camada de dados, para que sejam mais reais e também aplicáveis em realidades empresariais.

A descrição da lógica de negócio será feita através de diagramas de classes, sequência.

A nível da camada de dados, os protótipos irão suportar 3 operações a nível dos dados, ilustrada na Figura 14.

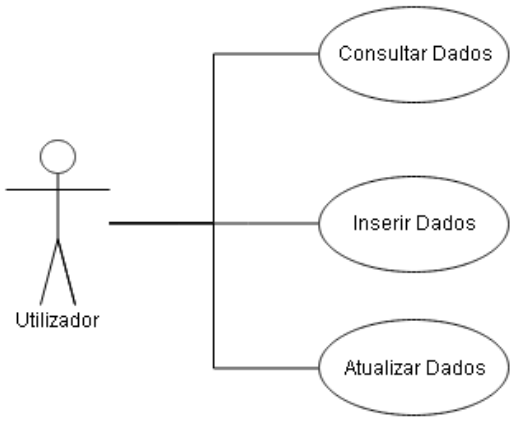

Figura 14 – Casos de uso suportados em ambos protótipos.

A operação de consulta dos dados permite ao utilizador obter dados já existentes ou dados que já tinha inserido/atualizado anteriormente. Os protótipos desenvolvidos têm apresentações dos dados diferentes, pois o protótipo descrito na secção 3.1 apresentará os dados em duas *grids* e o descrito na secção 3.2 apresentará os resultados em *combobox* e caixas de texto.

A inserção de dados permite ao utilizador inserir dados na base de dados. Os protótipos permitem a inserção de dados de forma diferenciada, ou seja, no protótipo descrito na secção 3.1 a inserção é feita através das *grids* e no protótipo da secção 3.2 é feita através de caixas de texto.

A atualização de dados implica que tenha existido um registo prévio, que pode ter sido inserido pelo utilizador ou que já existia na base de dados. O processo da atualização de dados é semelhante à inserção dos dados, ou seja, no protótipo descrito na secção 3.1 a operação é realizada através de *grids* e no protótipo da secção 3.2 é realizada através de caixas de texto.

A operação de apagar dados não será contemplada (apesar de estar disponível nos protótipos), pois não será relevante para a presente dissertação.

O padrão de *software* utilizado em ambos os protótipos foi o padrão MVC (*Model-View-Controller*). É uma abordagem que facilita o desenvolvimento e estruturação da aplicação, uma vez que há uma separação clara entre a interface e a camada lógica. Temos portanto um conjunto de classes que assumem o papel de *controller*, sendo que outras classes assumem o papel de *views* e temos ainda os *models*, entre outras. Os *controllers* tratam pedidos que provenham do utilizador, a solicitar por exemplo, dados da base de dados, e encaminham os dados solicitados para que o utilizador os consiga visualizar. As *views*, por sua vez, são as classes que fazem parte da camada da interface e portanto são responsáveis pela geração das páginas da aplicação. Quanto aos *models*, estes encapsulam informações sobre os vários objetos do sistema e são utilizados tanto pelos *controllers* como pelas *views*, de forma a serem um meio de troca de informação. Será apresentado um diagrama de classes, para cada protótipo, com a especificação das classes responsáveis por executar cada um dos papéis acima descritos. No protótipo 2, uma vez que é usada somente uma tabela que não está dependente de nenhuma outra, não se irá usar uma classe *model*, sendo os parâmetros passados diretamente da *view* para o *controller* e do *controller* para o *back-end*.

Como já foi dito anteriormente, para a presente dissertação, foram desenvolvidos 2 protótipos:

- Um protótipo com um *front-end* em ExtJS 4, com um *middle-tier* em Java e um *back-end* em C++ e uma base de dados Oracle;
- Um protótipo monolítico desenvolvido somente em Java, recorrendo a *JDBC* para conexão à base de dados Oracle.

3.1 Implementação 1

Na presente secção irão ser apresentadas as questões mais técnicas relativas ao protótipo 1, nomeadamente a nível de arquitetura aplicacional utilizada para o desenvolver (que será uma arquitetura 3-*Tier*, conforme mencionado na secção 2.5.4).

Será apresentado o modelo de domínio da implementação, bem como diagramas de classes e sequência que permitirão visualizar o fluxo da informação desde o *front-end* até ao *back-end*.

Por fim, são apresentados exemplos de código utilizado no presente protótipo, bem como *screenshots* da aplicação em funcionamento.

3.1.1 Arquitetura aplicacional

A arquitetura da aplicação do protótipo 1 é apresentada na Figura 15. O protótipo foi desenvolvido através das seguintes tecnologias:

- *Front-end* (ou *View*) em ExtJS 4
- *Middle-tier* (ou *Controller*) em Java, com chamadas Ajax e transmissão de dados *JSon String* entre o *front-end* ExtJS 4 e este *middle-tier*
- *Back-end* de acesso aos dados em C++
- Base de dados Oracle

Neste caso, recorreu-se a serviços em C++ em vez de *JDBC*, pois a nível transacional o C++ garante a atomicidade nas transações, ou seja, como os serviços têm *commit* ou *rollback*, está garantida a integridade dos dados, não dependendo esta, nem do Java nem do *front-end*. Se fosse usado *JDBC* teria que se realizar, caso fosse necessário, um trabalho para se garantir isto mesmo, ou seja, criar classes que conteriam as transações completas e que se encarregariam de fazer o *commit* ou o *rollback* de cada transação. Para além destas razões, em termos de velocidade, espera-se que o C++ supere um *JDBC*, pois será algo que também se pretende avaliar na presente dissertação.

Os utilizadores do protótipo utilizarão *browsers* como forma de aceder ao *front-end*, permitindo-lhes disfrutar duma experiência única em HTML5. Seguidamente farão pedidos Ajax (recorrendo a uma *JSon String*) que serão encaminhados através do *middle-tier* (utilizando RMI[6] e XML Base 64) para o servidor C++. A camada *back-end* está toda na mesma máquina física, no entanto, o servidor C++, para garantir a atomicidade dos dados, faz operações a nível transacional, interagindo com a base de dados para realizar os pedidos feitos pelos utilizadores.

O retorno dos dados, do servidor C++ para o *middle-tier* é feito por um *array*-bidimensional (caso o serviço solicitado tenha sido um os métodos de acesso - *getter*, caso seja outro serviço, o utilizador só é informado, através de uma mensagem, se o serviço que requisitou foi concluído com sucesso ou insucesso) que posteriormente é convertido para uma *JSon String* e enviado para o *front-end*, preenchendo os campos respetivos com os dados retornados.

3.1.2 Modelo de domínio

Na primeira implementação, visando o que foi referido na secção 3.1.1, foi adotado o modelo de domínio referido na Figura 16.

A Grid1 irá ter linhas com 5 campos de dados visíveis e 1 invisível. Ao clicar numa das linhas, poderá ou não ser preenchida a GridTab1 com 3 campos visíveis e 2 invisíveis (caso tenha dados para o efeito), pois conforme o diagrama da Figura 16, a relação é de 0 ou muitos.

[6] RMI é um sistema Java que permite a comunicação remota entre programas escritos em Java, utilizando para isso RPC..

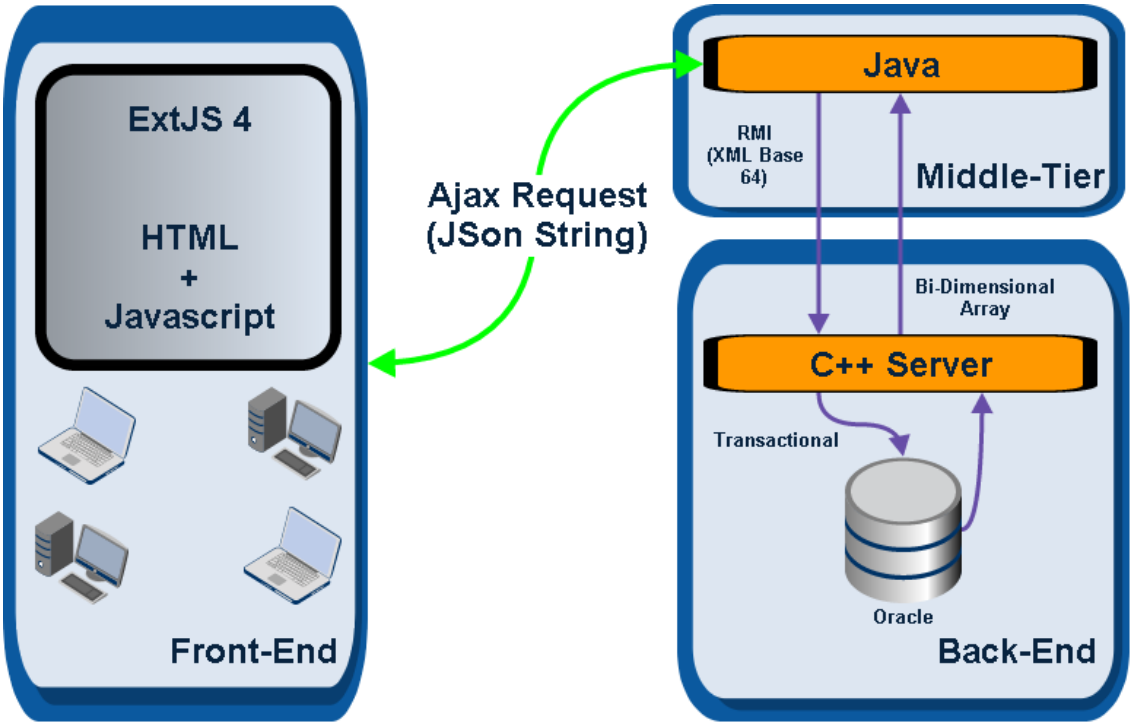

Figura 15 - Arquitetura do protótipo 1.

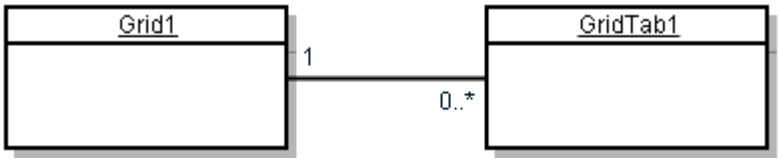

Figura 16 - Modelo de domínio da Implementação 1

3.1.3 Diagrama de classes

Front-end

Para o *front-end* foram utilizadas as classes apresentadas na Figura 17. O protótipo será composto por duas *grids*: Grid1 e GridTab1.

Inicialmente foi criada a classe UI_View, onde irão ser colocadas as *grids*. Esta classe estende uma outra classe fornecida com a *framework* do ExtJS4 denominada Ext.form.Panel. De seguida, foi criada a Grid1 que é composta por Grid1Model (estende a classe da *framework* Ext.data.Model), Grid1Store (estende a classe da *framework* Ext.data.Store) e Grid1View (estende a classe da *framework* Ext.grid) e colocada sobre a UI_View.

Para a criação da GridTab1, em primeiro lugar, tem que ser adicionado um painel que suporte *tabs*, portanto constatou-se a necessidade de criar uma classe GridTabPanel que irá estender uma classe fornecida na *framework* ExtJS4 denominada Ext.tab.Panel. Posteriormente, adicionou-se a GridTab1, composta à semelhança da Grid1, por GridTab1Model (estende a classe da *framework* Ext.data.Model), GridTab1Store (estende a classe da *framework*

Ext.data.Store) e GridTab1View (estende a classe da *framework* Ext.grid) e colocada sobre a GridTabPanel.

Em paralelo foi também criada a classe UI_Controller que irá estender a classe Ext.app.Controller da *framework* para encaminhar pedidos feitos pelo utilizador da UI_View para a camada *middle-tier*.

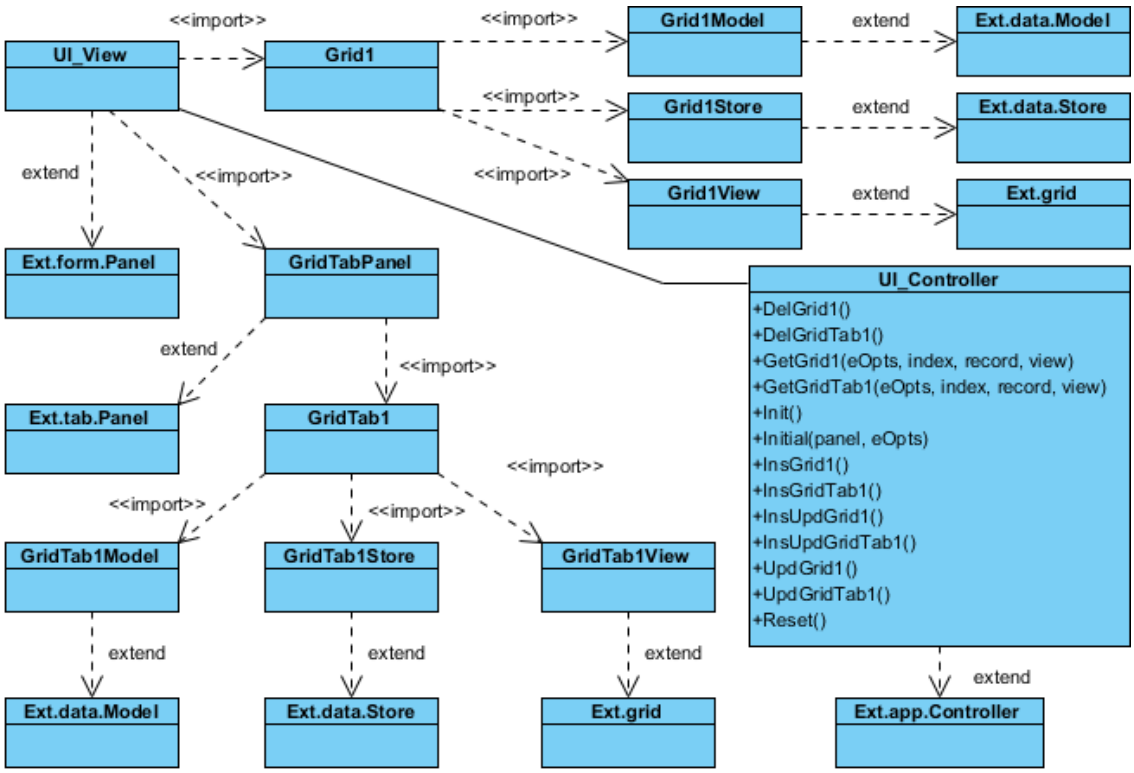

Figura 17 - *Front-end* do protótipo 1

Middle-Tier

Para o presente protótipo foi desenvolvida uma camada de *middle-tier* apresentada na Figura 18.

Foi criada a classe que responde aos pedidos gerados pelo *front-end*, denominada Controller. Esta classe contém as operações possíveis de efetuar em cada uma das *grids* (operações CRUD, à semelhança das bases de dados relacionais). Para acionar estas operações, basta que o utilizador pressione o botão correspondente na interface do utilizador (ver secção 3.1.7 afim de visualizar a interface do utilizador e como operar com a mesma). Esta classe utiliza dois modelos de dados, ou seja, um para cada *grid* (DataModelGrid1 para a **Grid1** e DataModelGridTab1 para a **GridTab1**). Estes modelos de dados contêm os campos referidos na secção 3.1.5 com os respetivos os métodos de acesso - *getter* e *setter*.

Para o encaminhamento de pedidos do *middle-tier* para o *back-end* foi adicionada a classe Bus que terá como função preparar os dados e comunicar com o Webservice para enviar os dados para o *back-end*. Os seus métodos (exceto o construtor e os métodos de acesso - *getter*) têm

retorno booleano para validar se as operações com o *back-end* tiveram sucesso ou insucesso. Os métodos de acesso - *getter* como simplesmente vão buscar dados, devolvem um *hashmap* com os dados retornados do *back-end* e tratados/moldados em *middle-tier*.

Por fim, a classe Webservice funciona como um *WSDL* e recebe nos seus métodos o *output* do servidor (se tiver sucesso, recebe o código 0, caso contrário recebe um código diferente de 0).

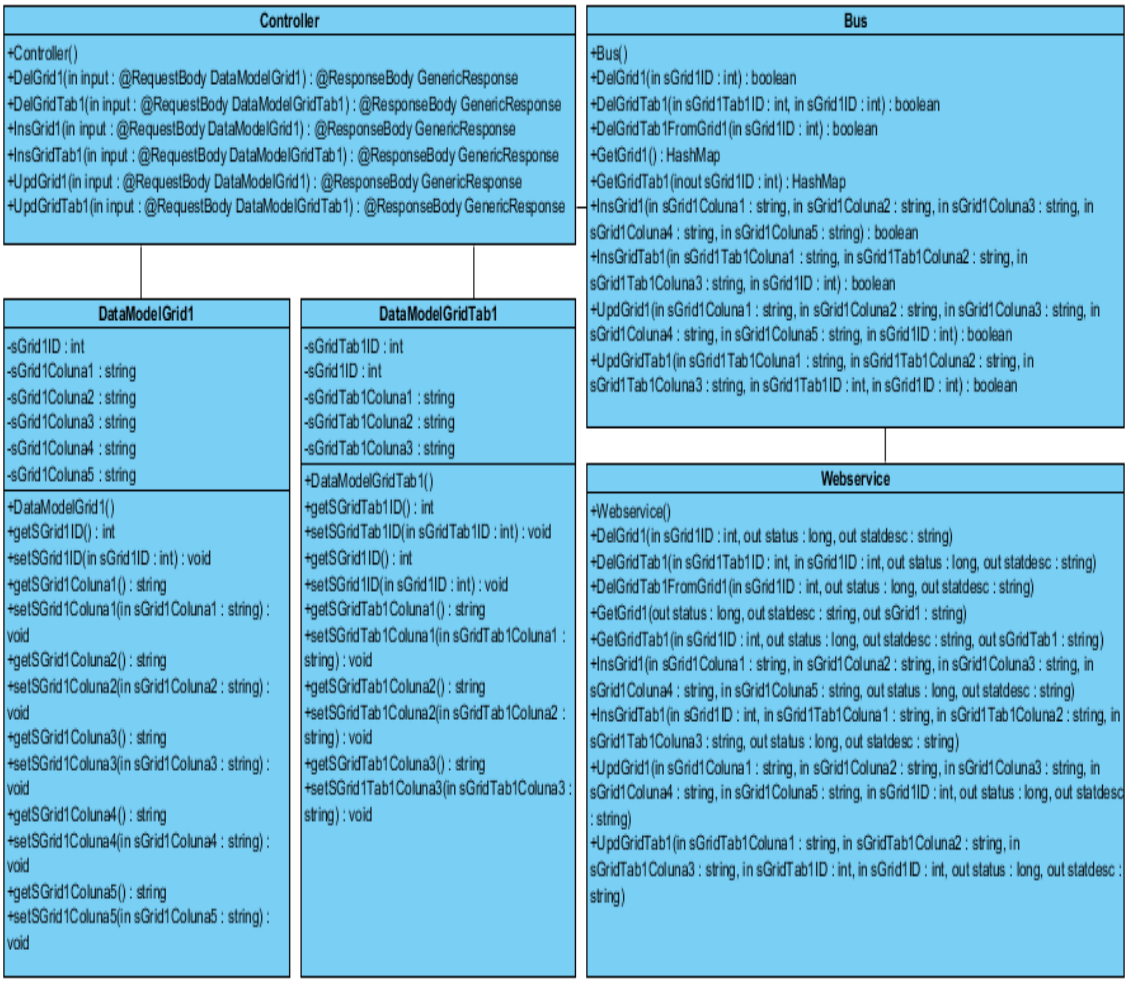

Figura 18 - *Middle-tier* do protótipo 1

Back-end

Por fim, para o *back-end* foram criadas as classes do lado do servidor, apresentadas na Figura 19.

Como foi dito ao longo desta secção, o protótipo irá ter duas *grids*, como tal, foi necessário criar 2 classes .CPP (db_Grid1.cpp e db_GridTab1.cpp), com os respetivos .h (db_Grid1.h e db_GridTab1.h), sendo estes detentores da estrutura dos campos da base de dados, referidos na secção 3.1.5, alusivos ao seu âmbito. Estas classes são necessárias para que possa haver meio de processar a informação no servidor de C++. Será utilizada uma classe denominada db_util.h que permitirá aceder a operações sobre os *arrays* bidirecionais (GP).

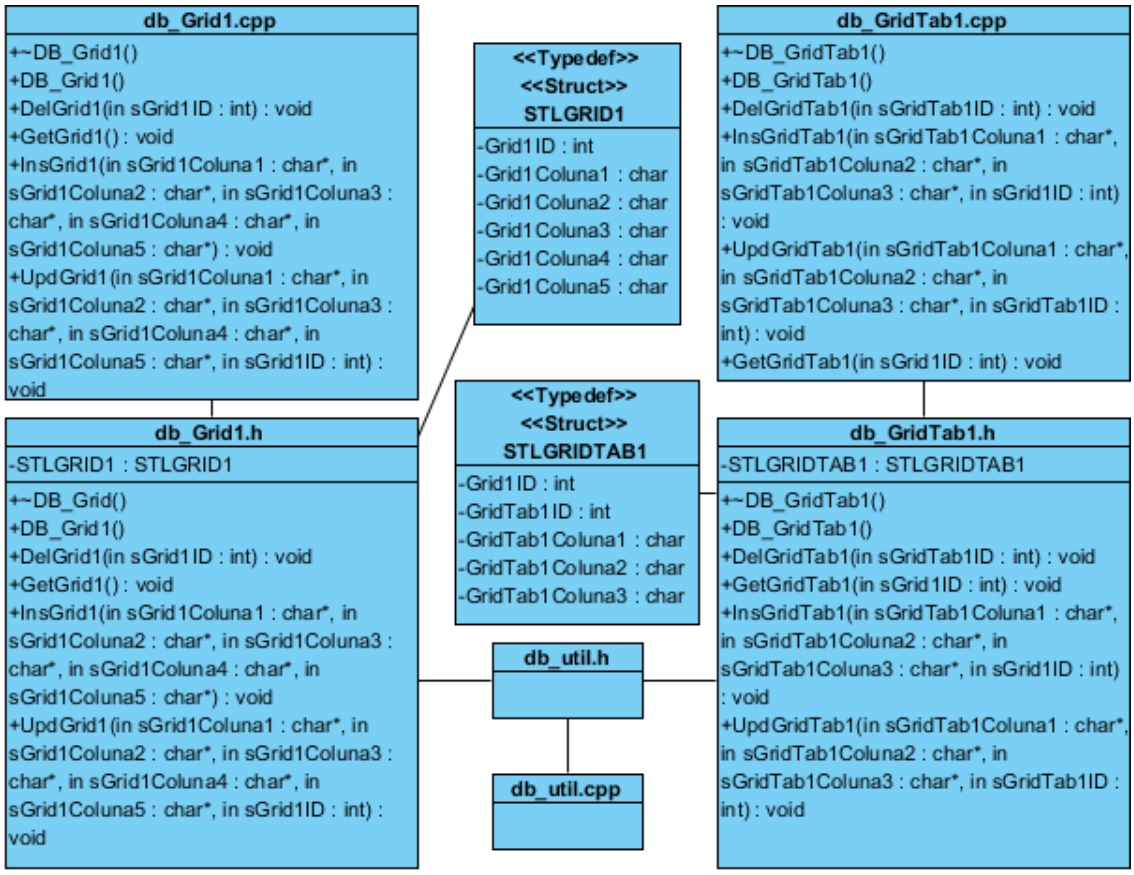

Figura 19 - *Back-end* do protótipo 1

3.1.4 Diagramas de sequência

Neste tópico serão apresentados diagramas de sequência alusivos ao protótipo 1, a nível de obtenção, inserção e edição de dados na Grid1, pois as operações para a GridTab1 são similares à Grid1 (somente muda o facto de ao obter/criar/editar dados, a mesma ter 2 campos escondidos, sendo um deles o campo Grid1ID e outro é o GridTab1ID).

Na Figura 20 é apresentado o diagrama de sequência para a obtenção de dados para preencher a Grid1. Após o utilizador abrir a aplicação na interface UI_View, é acionado o método *GetGrid1* presente no UI_Controller que, por sua vez, irá invocar o Controller que se encontra no *middle-tier*. O *middle-tier* está desenvolvido em Java *Spring* [37], logo é necessário usar, na assinatura do método *GetGrid1* do Controller, o cabeçalho: `@RequestMapping(value = "/GetGrid1.co", method = RequestMethod.POST)` para que os pedidos via *front-end* sejam encaminhados para as classes e/ou métodos do Controller.

Seguidamente, o Controller instancia o método *GetGrid1* do Bus. Por conseguinte, a classe Bus instancia um *HashMap* (para o retorno) e executa o Webservice, chamando o método *GetGrid1*.

Por fim, o Webservice devolve as variáveis *status, statdesc* e *sGrid1* e caso a variável de retorno status for 0, a classe Bus irá atribuir ao *HashMap* de retorno (*map*), os dados presentes em *sGrid1*, invocando para isso os métodos *SET* do modelo de dados. É devolvido o *HashMap* ao Controller, que posteriormente converte o *HashMap* numa *JSon String* e envia para o interface do utilizador (onde é preenchida a *Grid1* com os dados da *JSon String*, pois a classe Grid1Model possui os mesmos campos que o modelo de dados de Java, o que permite ao *front-end* saber onde colocar cada campo da *JSon String* na Grid1). Caso a variável *status* do Webservice for diferente de 0, é devolvido *null* ao Controller, que por sua vez, converte o retorno numa *JSon String* e envia para o interface do utilizador (onde é mostrada uma mensagem de erro a informar o insucesso da operação).

Na Figura 21 é apresentado o diagrama de sequência para a inserção de dados através da *Grid1*. Após o utilizador preencher a *Grid1* com dados, na interface UI_View e pressionar o botão "Gravar" da respetiva *Grid*, é acionado o método *InsUpdGrid1* que irá validar se é uma inserção ou uma atualização de dados. Como neste caso é uma inserção, é acionado o método *InsGrid1* presente no UI_Controller que, por sua vez, irá invocar o Controller que se encontra no *middle-tier*. É necessário usar, na assinatura do método *InsGrid1* do Controller, o cabeçalho:
`@RequestMapping(value = "/InsGrid1.co", method = RequestMethod.POST)`
para que os pedidos via *front-end* sejam encaminhados para as classes e/ou métodos do Controller.

Seguidamente, o Controller recebe uma instância do modelo de dados DataModelGrid1 (afim de poder fazer os métodos de acesso - *getters* das variáveis presentes no mesmo) e instancia o método *InsGrid1* do Bus. Por conseguinte, a classe Bus executa o Webservice, chamando o método *InsGrid1*, passando-lhe como parâmetros os valores das 5 colunas (independentemente de terem ou não terem sido preenchidos em *front-end*).

Por fim, o Webservice devolve as variáveis *status, statdesc* e caso a variável de retorno *status* for 0, a classe Bus irá retornar *true* ao Controller, que posteriormente converte o resultado numa *JSon String* e envia para o interface do utilizador (onde é apresentada uma mensagem a confirmar o sucesso da operação). Caso a variável *status* do Webservice for diferente de 0, é devolvido *false* ao Controller, que por sua vez, converte o retorno numa *JSon String* e envia para o interface do utilizador (onde é mostrada uma mensagem de erro a informar o insucesso da operação).

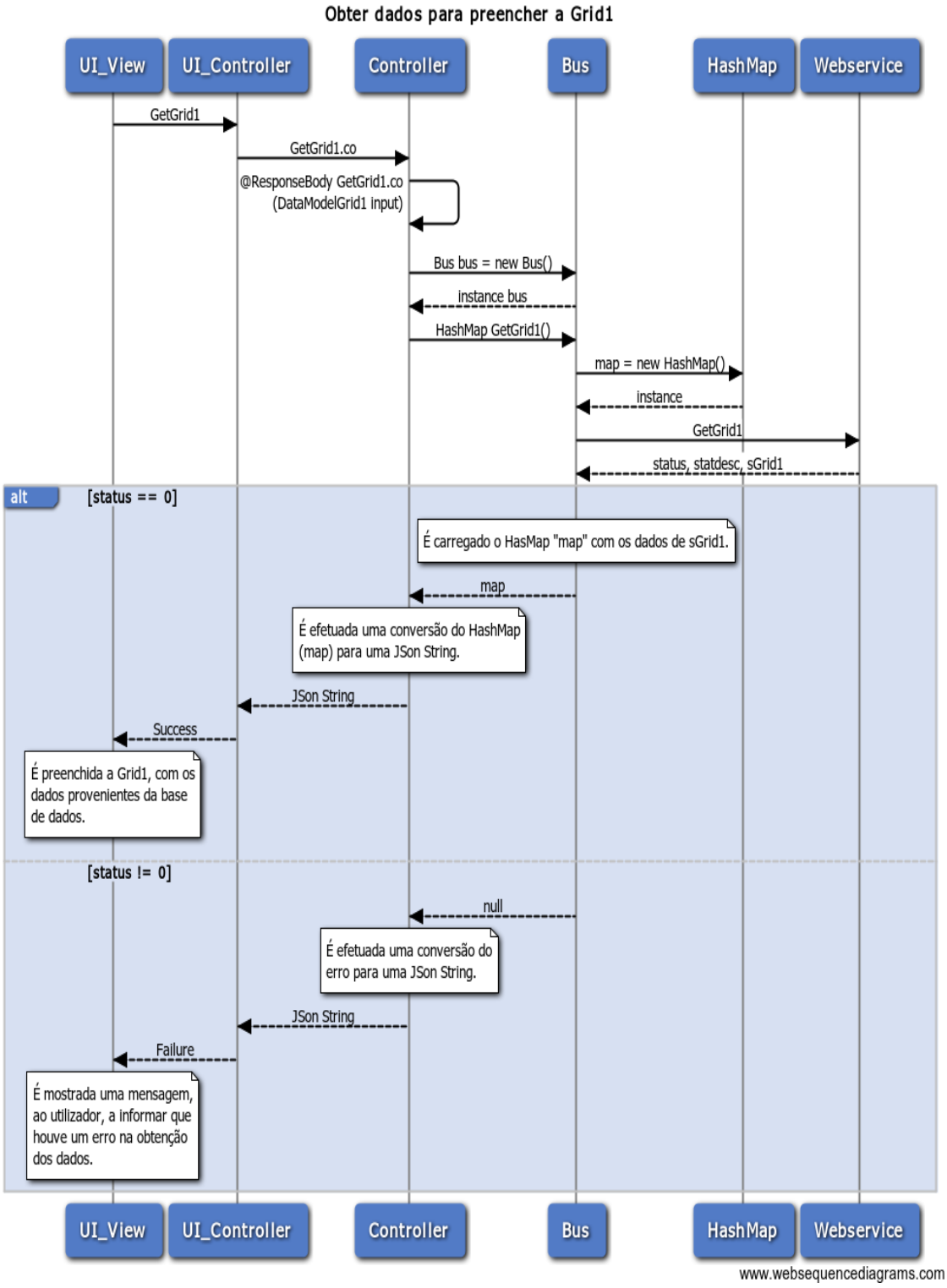

Figura 20 - Obtenção de dados para o preenchimento da Grid1

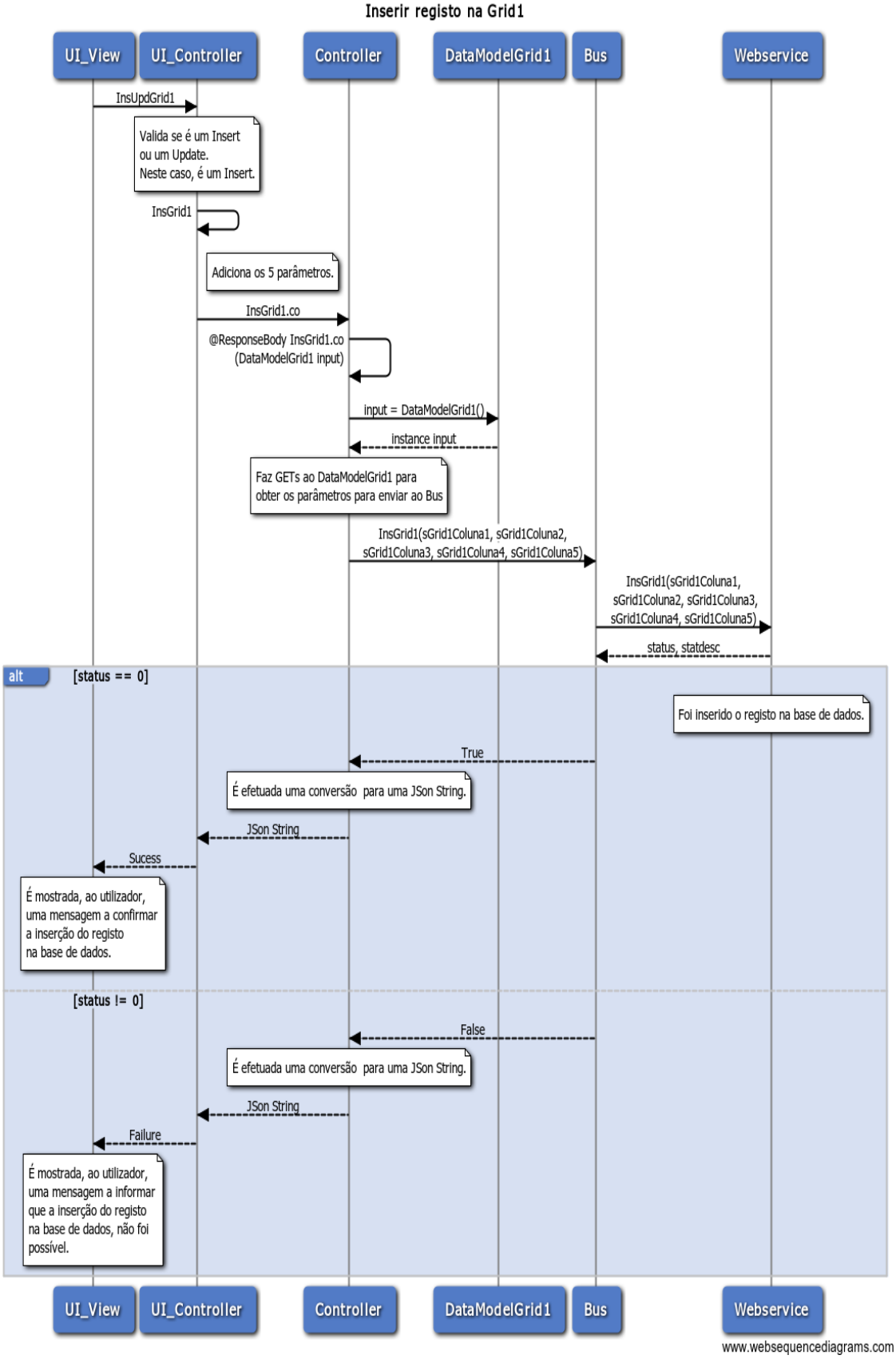

Figura 21 - Inserção de dados para o preenchimento da Grid1

Na Figura 22 é apresentado o diagrama de sequência para a atualização de dados através da Grid1. Após o utilizador preencher a *Grid1* com dados na interface UI_View e pressionar o botão "Gravar" da respetiva *Grid*, é acionado o método *InsUpdGrid1* que irá validar se é uma inserção ou uma atualização de dados. Como neste caso é uma atualização, é acionado o método *UpdGrid1* presente no UI_Controller que, por sua vez, irá invocar o Controller que se encontra no *middle-tier*. É necessário usar, na assinatura do método *InsGrid1* do Controller, o cabeçalho:

```
@RequestMapping(value = "/UpdGrid1.co", method = RequestMethod.POST)
```

para que os pedidos via *front-end* sejam encaminhados para as classes e/ou métodos do Controller.

Seguidamente, o Controller recebe uma instância do modelo de dados DataModelGrid1 (afim de poder fazer os métodos de acesso - *getters* das variáveis presentes no mesmo) e instancia o método *UpdGrid1* do Bus. Por conseguinte, a classe Bus executa o Webservice, chamando o método *UpdGrid1*, passando-lhe como parâmetros os valores das 6 colunas (independentemente de terem ou não terem sido preenchidos em *front-end*).

Por fim, o Webservice devolve as variáveis *status, statdesc* e caso a variável de retorno status for 0, a classe Bus irá retornar *true* ao Controller, que posteriormente converte o resultado numa *JSon String* e envia para o interface do utilizador (onde é apresentada uma mensagem a confirmar o sucesso da operação). Caso a variável *status* do Webservice for diferente de 0, é devolvido *false* ao Controller, que por sua vez, converte o retorno numa *JSon String* e envia para o interface do utilizador (onde é mostrada uma mensagem de erro a informar o insucesso da operação).

Na Figura 23 é apresentado o diagrama de sequência para a obtenção de dados para preencher a *GridTab1*. Após o utilizador selecionar uma linha da *Grid1* na interface UI_View, é acionado o método *GetGridTab1* (passando-lhe como parâmetro o *sGrid1ID*) presente no UI_Controller que, por sua vez, irá invocar o Controller que se encontra no *middle-tier*. É necessário usar, na assinatura do método *GetGridTab1* do Controller, o cabeçalho:

```
@RequestMapping(value = "/GetGridTab1.co", method = RequestMethod.POST)
```

para que os pedidos via *front-end* sejam encaminhados para as classes e/ou métodos do Controller.

Seguidamente, o Controller instancia o método *GetGridTab1* do Bus. Por conseguinte, a classe Bus instancia um *HashMap* (para o retorno) e executa o Webservice, chamando o método *GetGridTab1*.

Por fim, o Webservice devolve as variáveis *status, statdesc* e *sGridTab1* e caso a variável de retorno status for 0, a classe Bus irá atribuir ao *HashMap* de retorno, os dados presentes em *sGridTab1*, invocando para isso os métodos *SET* do modelo de dados. É devolvido o *HashMap* ao Controller, que posteriormente converte o *HashMap* numa *JSon String* e envia para o interface do utilizador (onde é preenchida a *GridTab1* com os dados da *JSon String*). Caso a variável *status* do Webservice for diferente de 0, é devolvido *null* ao Controller, que por sua

vez, converte o retorno numa *JSon String* e envia para o interface do utilizador (onde é mostrada uma mensagem de erro a informar o insucesso da operação).

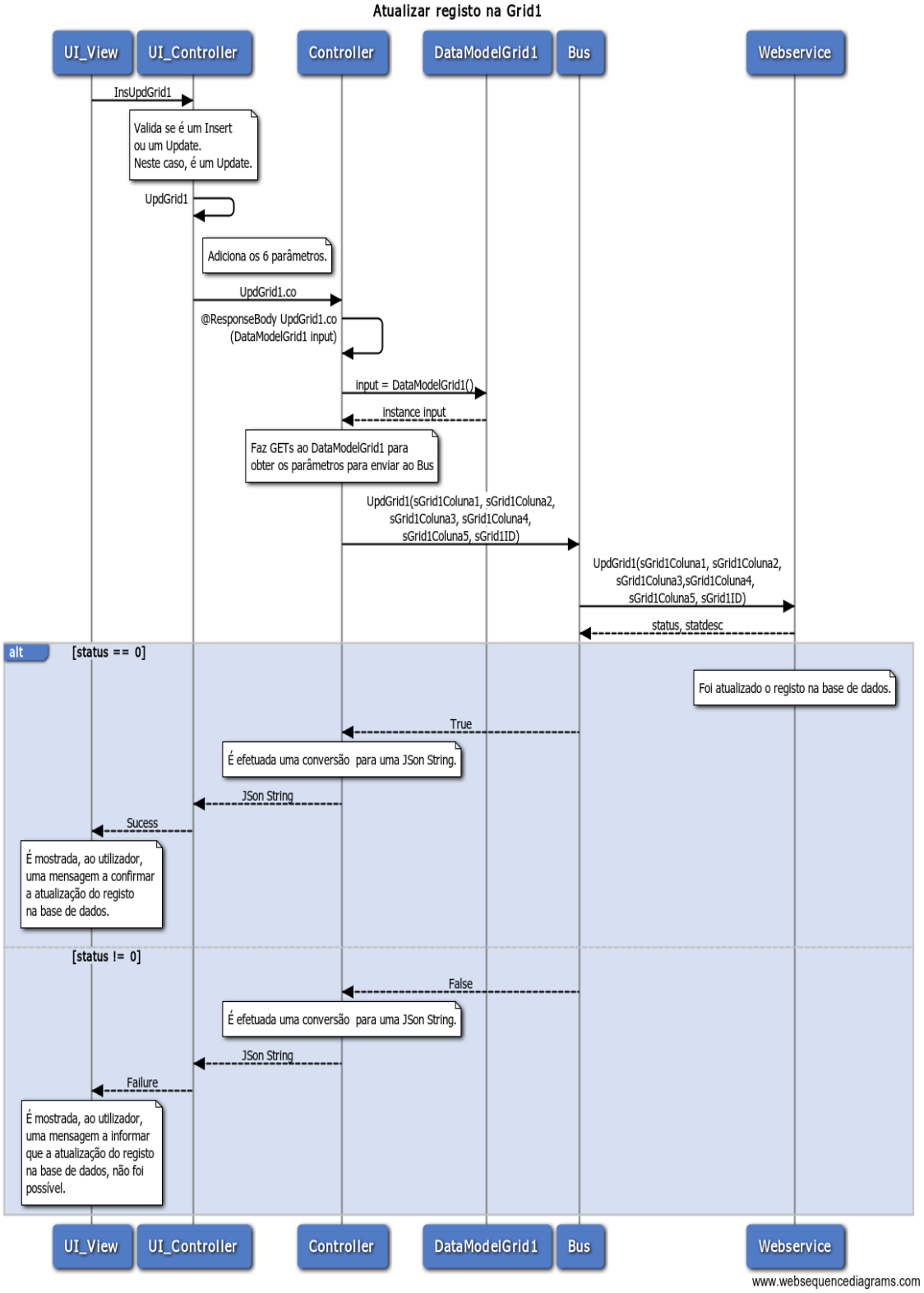

Figura 22 - Atualização de dados para o preenchimento da Grid1

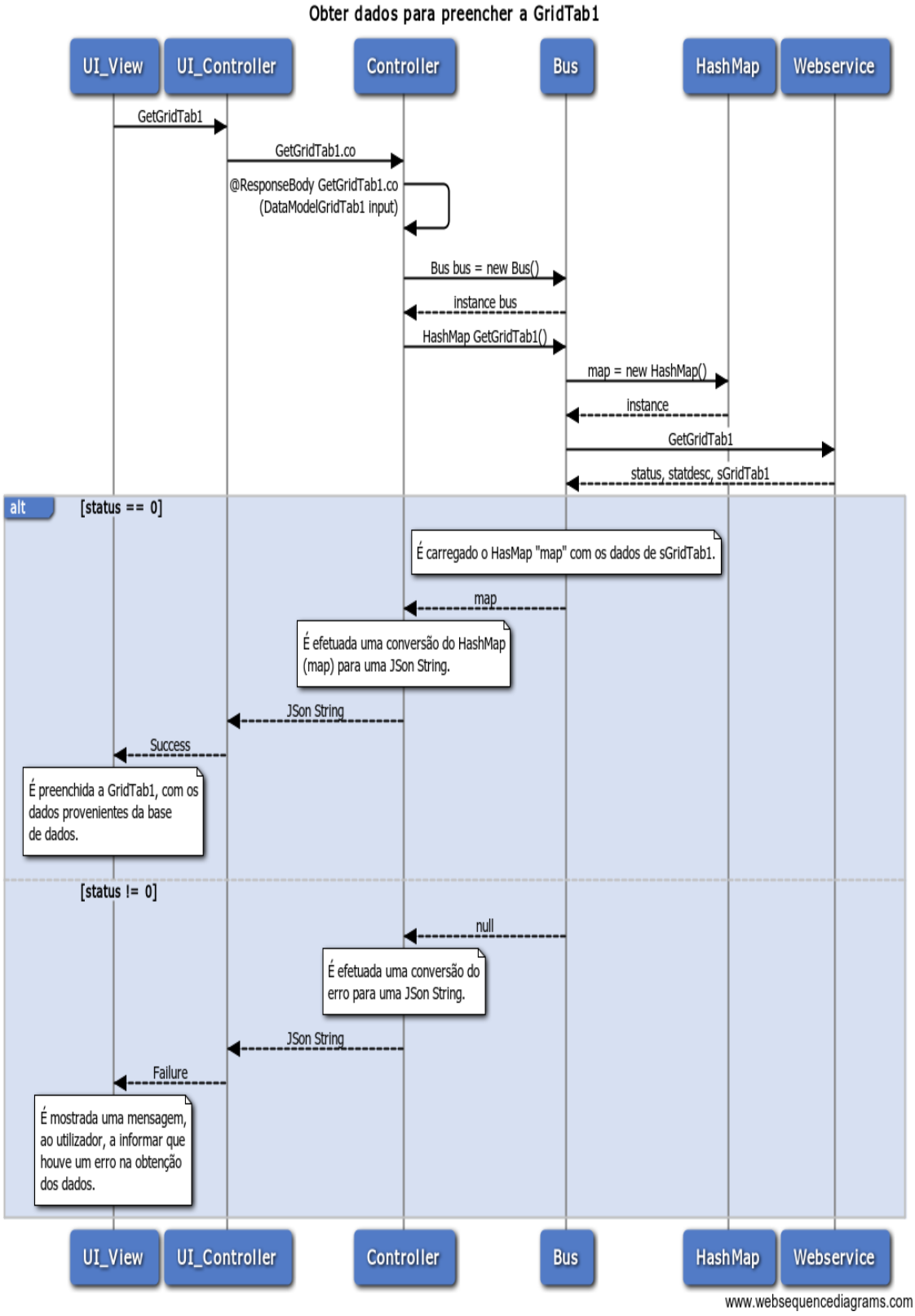

Figura 23 - Obtenção de dados para o preenchimento da GridTab1

Na Figura 24 é apresentado o diagrama de sequência para a inserção de dados através da *GridTab1*. Após o utilizador preencher a *GridTab1* com dados na interface UI_View e pressionar o botão "Gravar" da respetiva *Grid*, é acionado o método *InsUpdGridTab1* que irá validar se é uma inserção ou uma atualização de dados. Como neste caso é uma inserção, é acionado o método *InsGridTab1* presente no UI_Controller que, por sua vez, irá invocar o Controller que se encontra no *middle-tier*. É necessário usar, na assinatura do método *InsGridTab1* do Controller, o cabeçalho: `@RequestMapping(value = "/InsGridTab1.co", method = RequestMethod.POST)` para que os pedidos via *front-end* sejam encaminhados para as classes e/ou métodos do Controller.

Seguidamente, o Controller recebe uma instância do modelo de dados DataModelGridTab1 (afim de poder fazer os métodos de acesso - *getters* das variáveis presentes no mesmo) e instancia o método *InsGridTab1* do Bus. Por conseguinte, a classe Bus executa o Webservice, chamando o método *InsGridTab1*, passando-lhe como parâmetros os valores das 4 colunas (independentemente de terem ou não terem sido preenchidos em *front-end*).

Por fim, o Webservice devolve as variáveis *status, statdesc* e caso a variável de retorno *status* for 0, a classe Bus irá retornar *true* ao Controller, que posteriormente converte o resultado numa *JSon String* e envia para o interface do utilizador (onde é apresentada uma mensagem a confirmar o sucesso da operação). Caso a variável *status* do Webservice for diferente de 0, é devolvido *false* ao Controller, que por sua vez, converte o retorno numa *JSon String* e envia para o interface do utilizador (onde é mostrada uma mensagem de erro a informar o insucesso da operação).

Na Figura 25 é apresentado o diagrama de sequência para a atualização de dados através da *GridTab1*. Após o utilizador preencher a *GridTab1* com dados na interface UI_View e pressionar o botão "Gravar" da respetiva *Grid*, é acionado o método *InsUpdGridTab1* que irá validar se é uma inserção ou uma atualização de dados. Como neste caso é uma atualização, é acionado o método *UpdGridTab1* presente no UI_Controller que, por sua vez, irá invocar o Controller que se encontra no *middle-tier*. É necessário usar, na assinatura do método *InsGridTab1* do Controller, o cabeçalho:

`@RequestMapping(value = "/UpdGridTab1.co", method = RequestMethod.POST)` para que os pedidos via *front-end* sejam encaminhados para as classes e/ou métodos do Controller.

Seguidamente, o Controller recebe uma instância do modelo de dados DataModelGridTab1 (afim de poder fazer os métodos de acesso - *getters* das variáveis presentes no mesmo) e instancia o método *UpdGridTab1* do Bus. Por conseguinte, a classe Bus executa o Webservice, chamando o método *UpdGridTab1*, passando-lhe como parâmetros os valores das 4 colunas (independentemente de terem ou não terem sido preenchidos em *front-end*).

Por fim, o Webservice devolve as variáveis *status, statdesc* e caso a variável de retorno status for 0, a classe Bus irá retornar *true* ao Controller, que posteriormente converte o resultado numa *JSon String* e envia para o interface do utilizador (onde é apresentada uma mensagem a

confirmar o sucesso da operação). Caso a variável *status* do Webservice for diferente de 0, é devolvido *false* ao Controller, que por sua vez, converte o retorno numa *JSon String* e envia para o interface do utilizador (onde é mostrada uma mensagem de erro a informar o insucesso da operação).

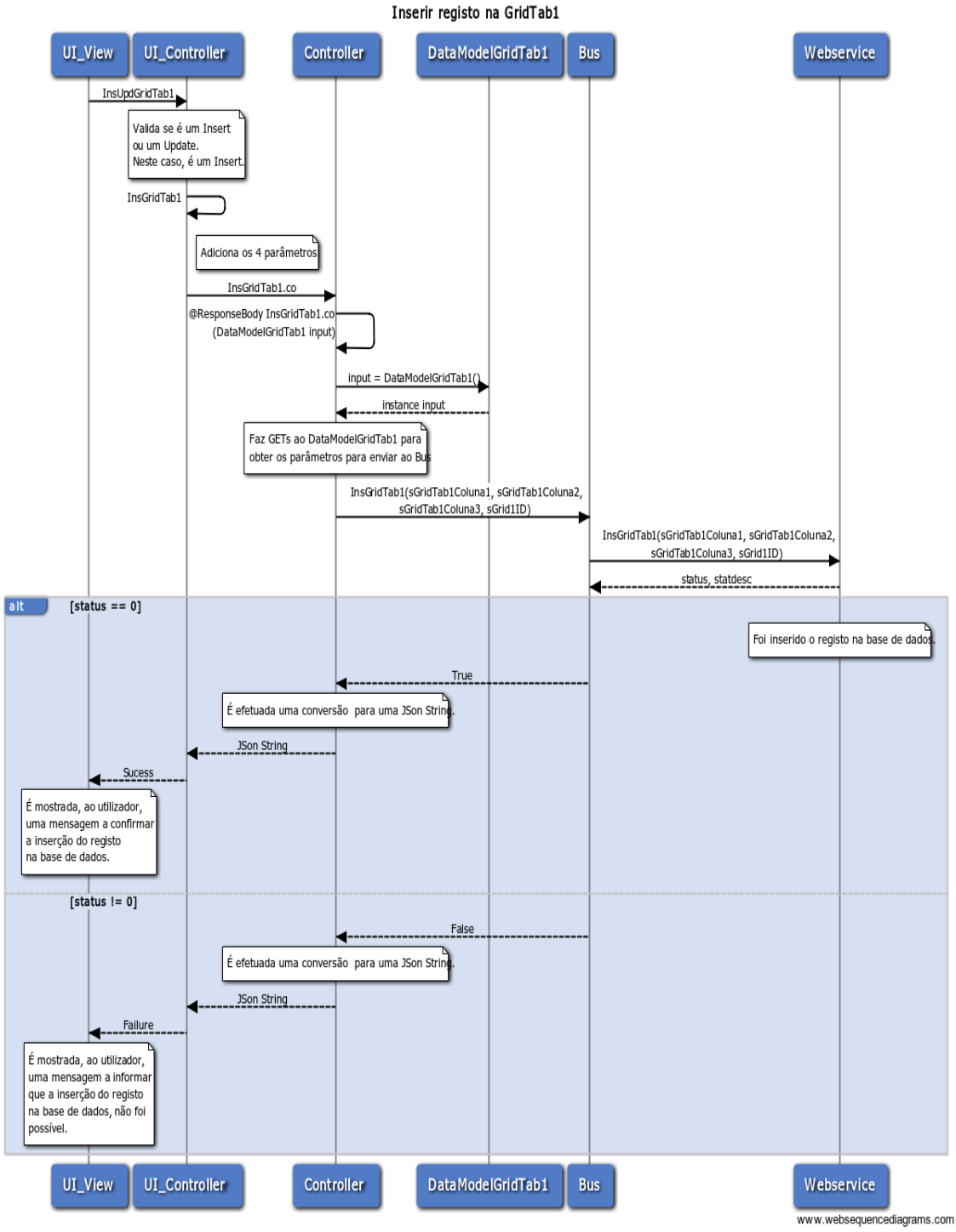

Figura 24 - Inserção de dados para o preenchimento da GridTab1

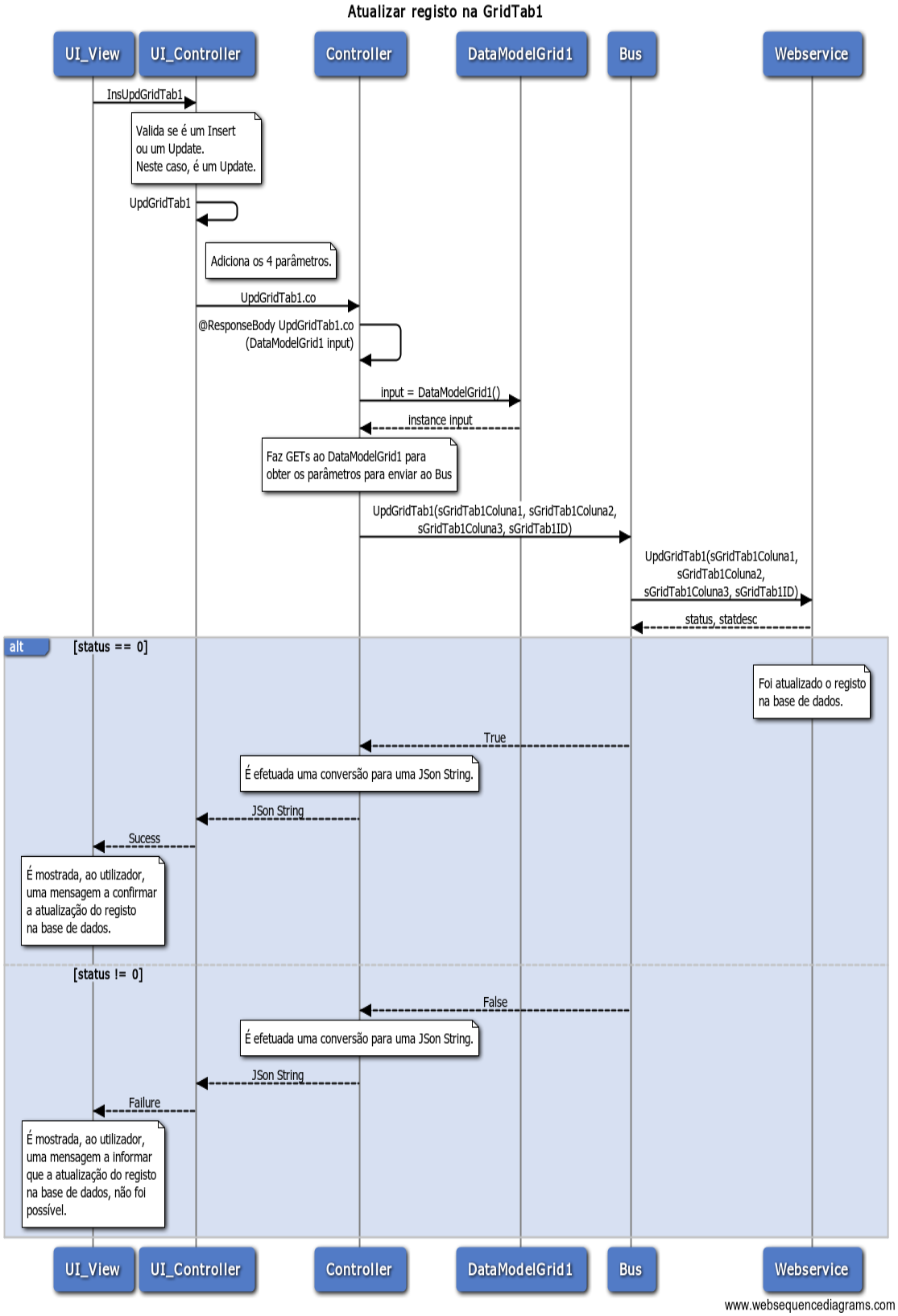

Figura 25 - Atualização de dados para o preenchimento da GridTab1

3.1.5 Tabelas da base de dados

Na presente implementação foram criadas duas tabelas, conforme a Figura 26.

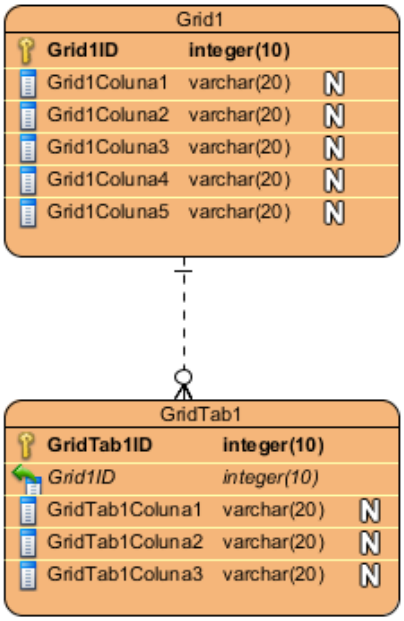

Figura 26 - Tabelas utilizadas na base de dados do protótipo 1

Estas tabelas irão servir para preenchimento das *Grids* já referidas nas seções 3.1.2 e 3.1.3. A relação entre as tabelas é de 1 (Grid1) para 0 ou muitos (GridTab1), pois só é possível obter dados da tabela GridTab1 após selecionarmos um registo presente na tabela Grid1.

Por motivos de simplificação da manipulação de dados, optou-se por colocar todos os campos da base de dados (excetuando as chaves primárias e estrangeira) como *VARCHAR*, pois os campos serem diferentes de *VARCHAR*, não iriam criar grande impacto na secção da Análise de Resultados (secção 3.4).

Na secção 3.1.7 serão apresentados *screenshots* da aplicação, onde se poderá verificar o processo de operação para efetuar o preenchimento de ambas as *Grids*.

3.1.6 Exemplos do código utilizado no protótipo

Neste tópico serão apresentados exemplos do código utilizado para a criação do presente protótipo. Serão apresentados exemplos a nível de *front-end*, *middle-tier* e *back-end* e só será apresentado o código para a Grid1, pois a GridTab1 tem o código semelhante e uma vez que, foram apresentados os diversos diagramas de sequência da mesma, poder-se-á verificar que a sequência de ações e codificação são semelhantes.

Front-end

É apresentado no Código 1 o código necessário para efetuar o pedido, em *front-end*, para a obtenção dos dados para o preenchimento da *Grid1*. Este método chamará o método da classe Controlller denominado *GetGrid1.co* (localizado na *middle-tier*). Possui duas cláusulas, nomeadamente *success* e *failure*, que serão executas em conformidade com a resposta obtida pela classe Controller (caso retorne erro executa o *failure*, caso contrário o *success*).

```
GetGrid1: function (eOpts, index, record, view) {
  this.reset();         // método para limpar as stores das Grids
  this.unmaskSearch(); // método para fazer o unmask do body do painel
  var param = this;
  Ext.Ajax.request({
      url: "Controller/GetGrid1.co",
      jsonData: param.pagent.getJson(),
      success: function (response) {
          var json = Ext.decode(response.responseText);
          if (json.success) {
           param.data = json.data;
           param.getGrid1().getStore().loadData(json.data.sGrid1);
           return;
          }
          else {
           Ext.Msg.show(translatePropertiesMessage(json.message));
           param.reset();
           return;
          }
      },
      failure: function (response) {
          param.reset();
          Ext.Msg.show(msg.eMessages.get(Ext.decode(
                  response.responseText).message));
      }
  });
}
```

Código 1 - Método GetGrid1 da classe UI_Controller

No Código 2 é apresentado o código para validar se é uma inserção ou uma atualização de dados. Caso a variável *sGrid1ID* for *null*, é invocado o método InsGrid1, caso contrário, é invocado o UpdGrid1.

```
InsUpdGrid1: function () {
        var sGrid1ID =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1ID;
        if (sGrid1ID === null) {
            this.InsGrid1();
        } else {
            this.UpdGrid1();
        }
}
```

Código 2 - Método InsUpdGrid1 da classe UI_Controller

É apresentado no Código 3 o código necessário para efetuar o pedido, em *front-end*, para a inserção dos dados provenientes da *Grid1*. Este método fará a captura dos valores localizados

na Grid1 (através de sucessivos métodos de acesso - *getters* à *Grid1*), adicionará os parâmetros e chamará o método da classe Controlller denominado *InsGrid1.co* (localizado na *middle-tier*). Possui duas cláusulas, nomeadamente *success* e *failure*, que serão executas em conformidade com a resposta obtida pela classe Controller (caso retorne erro executa o *failure*, caso contrário o *success*).

```
InsGrid1: function () {
  var param = this;
  // GET dos itens inseridos na Grid1
  var sGrid1Coluna1 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna1;
  var sGrid1Coluna2 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna2;
  var sGrid1Coluna3 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna3;
  var sGrid1Coluna4 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna4;
  var sGrid1Coluna5 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna5;
  // Processo para certificar que nenhuma das variáveis é passada a NULL
  if (sGrid1Coluna1 == null) sGrid1Coluna1 = "";
  if (sGrid1Coluna2 == null) sGrid1Coluna2 = "";
  if (sGrid1Coluna3 == null) sGrid1Coluna3 = "";
  if (sGrid1Coluna4 == null) sGrid1Coluna4 = "";
  if (sGrid1Coluna5 == null) sGrid1Coluna5 = "";
  // Atribuição dos parâmetros
  param.pagent.add('sGrid1Coluna1', sGrid1Coluna1);
  param.pagent.add('sGrid1Coluna2', sGrid1Coluna2);
  param.pagent.add('sGrid1Coluna3', sGrid1Coluna3);
  param.pagent.add('sGrid1Coluna4', sGrid1Coluna4);
  param.pagent.add('sGrid1Coluna5', sGrid1Coluna5);
  Ext.Ajax.request({
       url: service_path + "Controller/InsGrid1.co",
       jsonData: param.pagent.getJson(),
       success: function (response) {
           var json = Ext.decode(response.responseText);
           if (json.success) {
               param.data = json.data;
               alert("Foi inserido o registo na Grid1 com sucesso!");
               return;
           }
           else {
               Ext.Msg.show(translatePropertiesMessage(json.message));
               param.reset();
               return;
           }
       },
       failure: function (response) {
           param.reset();
           Ext.Msg.show(msg.eMessages.get(Ext.decode(
                   response.responseText).message));
       }
  });
}
```

Código 3 - Método InsGrid1 da classe UI_Controller

É apresentado no Código 4 o código necessário para efetuar o pedido, em *front-end*, para a atualização dos dados provenientes da *Grid1*. Este método fará a captura dos valores localizados na Grid1 (através de sucessivos métodos de acesso - *getters* à *Grid1*), adicionará os parâmetros e chamará o método da classe Controlller denominado *UpdGrid1.co* (localizado na *middle-tier*). Possui duas cláusulas, nomeadamente *success* e *failure*, que serão executas em conformidade com a resposta obtida pela classe Controller (caso retorne erro executa o *failure*, caso contrário o *success*).

```
UpdGrid1: function () {
  var param = this;
  // GET dos itens inseridos na Grid1
  var sGrid1Coluna1 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna1;
  var sGrid1Coluna2 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna2;
  var sGrid1Coluna3 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna3;
  var sGrid1Coluna4 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna4;
  var sGrid1Coluna5 =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1Coluna5;
  var sGrid1ID =
this.getGrid1().getSelectionModel().getSelection()[0].data.sGrid1ID;
  // Processo para certificar que nenhuma das variáveis é passada a NULL
  if (sGrid1Coluna1 == null) sGrid1Coluna1 = "";
  if (sGrid1Coluna2 == null) sGrid1Coluna2 = "";
  if (sGrid1Coluna3 == null) sGrid1Coluna3 = "";
  if (sGrid1Coluna4 == null) sGrid1Coluna4 = "";
  if (sGrid1Coluna5 == null) sGrid1Coluna5 = "";
  // Atribuição dos parâmetros
  param.pagent.add('sGrid1Coluna1', sGrid1Coluna1);
  param.pagent.add('sGrid1Coluna2', sGrid1Coluna2);
  param.pagent.add('sGrid1Coluna3', sGrid1Coluna3);
  param.pagent.add('sGrid1Coluna4', sGrid1Coluna4);
  param.pagent.add('sGrid1Coluna5', sGrid1Coluna5);
  param.pagent.add('sGrid1ID', sGrid1ID);
  Ext.Ajax.request({
       url: service_path + "Controller/UpdGrid1.co",
       jsonData: param.pagent.getJson(),
       success: function (response) {
           var json = Ext.decode(response.responseText);
           if (json.success) {
               param.data = json.data;
               alert("Foi atualizado o registo na Grid1 com sucesso!");
               return;
           }
           else {
               Ext.Msg.show(translatePropertiesMessage(json.message));
               param.reset();
               return;
           }
       },
       failure: function (response) {
           param.reset();
           Ext.Msg.show(msg.eMessages.get(Ext.decode(
                   response.responseText).message));
       }
```

```
  });
}
```

Código 4 - Método UpdGrid1 da classe UI_Controller

Middle-tier

É apresentado no Código 5, o código que permite a comunicação com a classe que executa o Webservice (Bus). Após execução do serviço, este método converte a informação obtida para *JSon String* sendo esta depois, passada para o *front-end*.

```
RequestMapping(value = "/GetGrid1.co", method = RequestMethod.POST)
    public @ResponseBody
    GenericResponse GetGrid1(@RequestBody DataModelGrid1 input) {
        // DataModelGrid1 contém os 6 campos da Grid1
        Bus bus = new Bus();
        GenericResponse genericResponse = null;
        genericResponse = generateGenericResponse(bus.GetGrid1(), bus);
        return genericResponse;
    }
```

Código 5 - Método GetGrid1 da classe Controller

É apresentado no Código 6, o código que permite a comunicação com a classe que executa o Webservice (Bus). Após execução do serviço, este método converte a informação obtida para *JSon String* sendo esta depois, passada para o *front-end*.

```
RequestMapping(value = "/InsGrid1.co", method = RequestMethod.POST)
    public @ResponseBody
    GenericResponse InsGrid1(@RequestBody DataModelGrid1 input) {
        // DataModelGrid1 contém os 6 campos da Grid1
        Bus bus = new Bus();
        GenericResponse genericResponse = null;
        String sGrid1Coluna1 = input.getSGrid1Coluna1();
        String sGrid1Coluna2 = input.getSGrid1Coluna2();
        String sGrid1Coluna3 = input.getSGrid1Coluna3();
        String sGrid1Coluna4 = input.getSGrid1Coluna4();
        String sGrid1Coluna5 = input.getSGrid1Coluna5();
        genericResponse =
generateGenericResponse(bus.InsGrid1(sGrid1Coluna1, sGrid1Coluna2,
sGrid1Coluna3, sGrid1Coluna4, sGrid1Coluna5), bus);
        return genericResponse;
    }
```

Código 6 - Método InsGrid1 da classe Controller

É apresentado no Código 7 o código que permite a comunicação com a classe que executa o Webservice (Bus). Após execução do serviço, este método converte a informação obtida para *JSon String* sendo esta depois, passada para o *front-end*.

```
RequestMapping(value = "/UpdGrid1.co", method = RequestMethod.POST)
    public @ResponseBody
    GenericResponse UpdGrid1(@RequestBody DataModelGrid1 input) {
        // DataModelGrid1 contém os 6 campos da Grid1
        Bus bus = new Bus();
        GenericResponse genericResponse = null;
        String sGrid1Coluna1 = input.getSGrid1Coluna1();
        String sGrid1Coluna2 = input.getSGrid1Coluna2();
        String sGrid1Coluna3 = input.getSGrid1Coluna3();
        String sGrid1Coluna4 = input.getSGrid1Coluna4();
        String sGrid1Coluna5 = input.getSGrid1Coluna5();
        int sGrid1ID = input.getSGrid1ID();
        genericResponse =
generateGenericResponse(bus.UpdGrid1(sGrid1Coluna1, sGrid1Coluna2,
sGrid1Coluna3, sGrid1Coluna4, sGrid1Coluna5, sGrid1ID), bus);
        return genericResponse;
    }
```

Código 7 - Método UpdGrid1 da classe Controller

É apresentado no Código 8, o código necessário para dar início à execução do Webservice que irá obter os dados na base de dados. Após executar o Webservice, caso o retorno seja igual a 0 (corresponde à parte do código `if ( status == 0 )`) o *HashMap* é preenchido e devolvido, caso contrário, é devolvido *null*.

```
public HashMap GetGrid1() {
        HashMap map = new HashMap();
        // EGP é a classe que permite receber os dados que provêm dos
           serviços à BD
        EGP gpGrid1List = new EGP();
        ArrayList<DataModelGrid1> grid1Arraylist = new
ArrayList<DataModelGrid1>();
        DataModelGrid1 grid1;
        try {
            // ermi é uma instância (protegida) da classe RMI do Java que
               irá fazer a conversão para XML Base 64
            ermi.exec("GetGrid1");
            // A variável status indicará se a execução do serviço deu erro
            int status = ermi.getInt("status").intValue();
            // A variável statdesc será um descritivo da variável status
            String statdesc = ermi.getStr("statdesc");
            if (status == 0) {
                gpGrid1List.load(ermi.getStr("sGrid1"));
                for (int i = 0; i < gpGrid1List.getRowCount(); i++) {
                    grid1 = new DataModelGrid1();
                    //SETs para cada elemento do "grid1Arraylist" que
                      provém do GP
                    grid1.setSGrid1ID(
                        Integer.parseInt(gpGrid1List.getStr(i, 0)));
                    grid1.setSGrid1Coluna1(gpGrid1List.getStr(i, 1));
                    grid1.setSGrid1Coluna2(gpGrid1List.getStr(i, 2));
                    grid1.setSGrid1Coluna3(gpGrid1List.getStr(i, 3));
                    grid1.setSGrid1Coluna4(gpGrid1List.getStr(i, 4));
                    grid1.setSGrid1Coluna5(gpGrid1List.getStr(i, 5));
                    grid1Arraylist.add(grid1);
                }
                map.put("sGrid1", grid1Arraylist);
            } else {
```

```
                return null;
            }
        } catch (Exception e) {
            setCatch(e);
        }
        return map;
    }
```

Código 8 - Método GetGrid1 da classe Bus

É apresentado no Código 9 o código necessário para dar início à execução do Webservice que irá inserir os dados na base de dados. Após executar o Webservice, caso o retorno seja igual a 0 (corresponde à parte do código `if ( status == 0 )`) retorna *true*, caso contrário, é devolvido *false*.

```
public Boolean InsGrid1(String sGrid1Coluna1, String sGrid1Coluna2, String
sGrid1Coluna3, String sGrid1Coluna4, String sGrid1Coluna5) {
        try {
            ermi.set("sGrid1Coluna1", sGrid1Coluna1);
            ermi.set("sGrid1Coluna2", sGrid1Coluna2);
            ermi.set("sGrid1Coluna3", sGrid1Coluna3);
            ermi.set("sGrid1Coluna4", sGrid1Coluna4);
            ermi.set("sGrid1Coluna5", sGrid1Coluna5);
            ermi.exec("InsGrid1");
            int status = ermi.getInt("status").intValue();
            String statdesc = ermi.getStr("statdesc");
            if (status == 0) {
                return true;
            } else {
                return false;
            }
        } catch (Exception e) {
            setCatch(e);
            return false;
        }
        return false;
    }
```

Código 9 - Método InsGrid1 da classe Bus

É apresentado no Código 10 o código necessário para dar início à execução do Webservice que irá atualizar os dados na base de dados. Após executar o Webservice, caso o retorno seja igual a 0 (corresponde à parte do código `if ( status == 0 )`) retorna *true*, caso contrário, é devolvido *false*.

```
public Boolean UpdGrid1(String sGrid1Coluna1, String sGrid1Coluna2, String
sGrid1Coluna3, String sGrid1Coluna4, String sGrid1Coluna5, int sGrid1ID) {
        try {
            ermi.set("sGrid1Coluna1", sGrid1Coluna1);
            ermi.set("sGrid1Coluna2", sGrid1Coluna2);
            ermi.set("sGrid1Coluna3", sGrid1Coluna3);
            ermi.set("sGrid1Coluna4", sGrid1Coluna4);
            ermi.set("sGrid1Coluna5", sGrid1Coluna5);
            ermi.set("sGrid1ID", sGrid1ID);
            ermi.exec("UpdGrid1");
            int status = ermi.getInt("status").intValue();
```

```
            String statdesc = ermi.getStr("statdesc");
            if (status == 0) {
                return true;
            } else {
                return false;
            }
        } catch (Exception e) {
            setCatch(e);
            return false;
        }
        return false;
    }
```

Código 10 - Método UpdGrid1 da classe Bus

Back-end

No Código 11 é apresentado o modelo de dados que o Java utiliza para mapear os campos da *Grid1*. Apesar do campo *SGrid1ID* da *Grid1* estar oculto, o mesmo tem que ser contemplado para os métodos de atualização e eliminação de dados (apesar desta última não ser abordada).

```
public class DataModelGrid1 {
        private int SGrid1ID;
        private String SGrid1Coluna1;
        private String SGrid1Coluna2;
        private String SGrid1Coluna3;
        private String SGrid1Coluna4;
        private String SGrid1Coluna5;
        /* GETs e SETs */
}
```

Código 11 - DataModelGrid1

No Código 12 está apresentado o código que o Webservice terá para comunicar com o servidor C++ para obter dados. Os parâmetros de saída serão:

- *status*, para se saber se teve sucesso (caso seja 0) ou insucesso (caso seja diferente de 0)
- *statdesc*, para ter um descritivo do resultado da operação
- *sGrid1*, com os registos todos da base de dados retornado pela *query* passada

```
GetGrid1=[OUT,LONG]status [OUT,STRING]statdesc [OUT,STRING]sGrid1
```

Código 12 – Webservice para o método GetGrid1

No Código 13 está apresentado o código que o Webservice terá para comunicar com o servidor C++ para inserir dados. Os parâmetros de saída serão:

- *status*, para se saber se teve sucesso (caso seja 0) ou insucesso (caso seja diferente de 0)
- *statdesc*, para ter um descritivo do resultado da operação

```
InsGrid1=[IN,STRING]sGrid1Coluna1 [IN,STRING]sGrid1Coluna2
[IN,STRING]sGrid1Coluna3 [IN,STRING]sGrid1Coluna4 [IN,STRING]sGrid1Coluna5
[OUT,LONG]status [OUT,STRING]statdesc
```

Código 13 - Webservice para o método InsGrid1

No Código 14 está apresentado o código que o Webservice terá para comunicar com o servidor C++ para atualizar dados. Os parâmetros de saída serão:

- *status*, para se saber se teve sucesso (caso seja 0) ou insucesso (caso seja diferente de 0)
- *statdesc*, para ter um descritivo do resultado da operação

```
UpdGrid1=[IN,STRING]sGrid1Coluna1 [IN,STRING]sGrid1Coluna2
[IN,STRING]sGrid1Coluna3 [IN,STRING]sGrid1Coluna4 [IN,STRING]sGrid1Coluna5
[IN,INT]sGrid1ID [OUT,LONG]status [OUT,STRING]statdesc
```

Código 14 - Webservice para o método UpdGrid1

É apresentado no Código 15 o código referente ao método *GetGrid1*, onde irá executar uma *query* à base de dados e irá preencher um *array* bidimensional (referenciado como GP), devolvendo-o através do método `gp.Print()`.

```
void db_Grid1::GetGrid1(GP &gp) {
    int nRows, iReg, iRegTot;
    char Servico[1024];
    // Definir o comprimento (1000) pelo número de colunas da tabela (6)
    gp.Create(1000, 6);
    CRecordSet crs(*db, 100);
    sprintf(Servico, "SELECT Grid1ID, Grid1Coluna1, Grid1Coluna2,
Grid1Coluna3, Grid1Coluna4, Grid1Coluna5 FROM Grid1 order by Grid1ID");
    iRegTot = 0;
    while ((nRows = crs.Retrieve(ADHOC, Servico)) > 0) {
        for (iReg = 0; iReg < nRows; iReg++) {
            crs.GetItem(iReg, "GRID1ID", stlGrid1.Grid1ID);
            crs.GetItem(iReg, "GRID1COLUNA1", stlGrid1.Grid1Coluna1);
            crs.GetItem(iReg, "GRID1COLUNA2", stlGrid1.Grid1Coluna2);
            crs.GetItem(iReg, "GRID1COLUNA3", stlGrid1.Grid1Coluna3);
            crs.GetItem(iReg, "GRID1COLUNA4", stlGrid1.Grid1Coluna4);
            crs.GetItem(iReg, "GRID1COLUNA5", stlGrid1.Grid1Coluna5);
            gp.Insert(iRegTot, "%d%s%s%s%s%s", stlGrid1.Grid1ID,
stlGrid1.Grid1Coluna1, stlGrid1.Grid1Coluna2, stlGrid1.Grid1Coluna3,
stlGrid1.Grid1Coluna4, stlGrid1.Grid1Coluna5);
            gp.Print();
            iRegTot++;
        }
    }
    if (nRows < 0) {
        cerror.Put(DBERROR, crs.GetError(), __FILE__, __LINE__,
crs.GetMsgError());
        throw ( cerror);
    }
}
```

Código 15 – Método GetGrid1 do db_Grid1.cpp

É apresentado no Código 16 o código referente ao método *InsGrid1*, onde irá executar uma *query* à base de dados para inserção do registo na tabela *Grid1*. No lado do servidor C++ estão implementados os *COMMIT* e *ROLLBACK* para caso haja sucesso ou erro na operação. Os caracteres que estão "SOH" correspondem ao caracter plica (') utilizadas para atribuir os valores passados no método à *query*, pois o servidor C++ não reconhece a plica, somente este tipo de caracter. Optou-se por colocar uma imagem no Código 16 para se notarem estes caracteres.

```
void db_Gridl::InsGridl( char* sGridlColunal, char *sGridlColuna2, char *
sGridlColuna3, char *sGridlColuna4, char *sGridlColuna5 ) {
  char servico[1024];
  sprintf( servico, "INSERT INTO Gridl ( sGridlColunal, sGridlColuna2,
  sGridlColuna3, sGridlColuna4, sGridlColuna5 ) VALUES (
  SOH%sSOH,SOH%sSOH,SOH%sSOH,SOH%sSOH,SOH%sSOH )", sGridlColunal,
  sGridlColuna2, sGridlColuna3, sGridlColuna4, sGridlColuna5 );
  if ( db->Execute( ADHOC, servico) < 0) {
    cerror.Put( DBERROR, db->GetError(), __FILE__, __LINE__, db->GetMsgError
    ());
    throw( cerror);
  }
}
```

Código 16 - Método InsGrid1 do db_Grid1.cpp

É apresentado no Código 17 o código referente ao método *UpdGrid1*, onde irá executar uma *query* à base de dados para atualização do registo na tabela *Grid1*. No lado do servidor C++ estão implementados os *COMMIT* e *ROLLBACK* para caso haja sucesso ou erro na operação. Os caracteres que estão "SOH" correspondem ao caracter plica (') utilizadas para atribuir os valores passados no método à *query*. À semelhança do método anterior, optou-se por colocar uma imagem neste código para se notarem estes caracteres.

```
void db_Gridl::UpdGridl( char* sGridlColunal, char *sGridlColuna2, char *
sGridlColuna3, char *sGridlColuna4, char *sGridlColuna5, int sGridlID ) {
  char servico[1024];
  sprintf( servico, "UPDATE Gridl SET sGridlColunal=SOH%sSOH,
  sGridlColuna2=SOH%sSOH, sGridlColuna3=SOH%sSOH, sGridlColuna4=SOH%sSOH,
  sGridlColuna5=SOH%sSOH WHERE sGridlID=SOH%dSOH", sGridlColunal,
  sGridlColuna2, sGridlColuna3, sGridlColuna4, sGridlColuna5, sGridlID );
  if ( db->Execute( ADHOC, servico) < 0) {
    cerror.Put( DBERROR, db->GetError(), __FILE__, __LINE__, db->GetMsgError
    ());
    throw( cerror);
  }
}
```

Código 17 - Método UpdGrid1 do db_Grid1.cpp

No Código 18 está apresentado o código para o .H que irá ser utilizado para as operações na base de dados. O *include db_util.h* possui os métodos para operar com os *arrays* bidirecionais (referenciados como GP). Possui uma estrutura com os campos da base de dados com mais um caracter de tamanho, para o *enter*.

```
#ifndef DB_GRID1_H
#define DB_GRID1_H
#include <db_util.h>

typedef struct STLGRID1 {
    int Grid1ID;
    char Grid1Coluna1 [20 + 1];
    char Grid1Coluna2 [20 + 1];
    char Grid1Coluna3 [20 + 1];
    char Grid1Coluna4 [20 + 1];
    char Grid1Coluna5 [20 + 1];
};

class db_Grid1 {
protected:
    STLGRID1 stlGrid1;
public:
    db_Grid1();
    ~db_Grid1();
    void GetGrid1(GP &gp);
    void InsGrid1(char* sGrid1Coluna1, char *sGrid1Coluna2, char
*sGrid1Coluna3, char *sGrid1Coluna4, char *sGrid1Coluna5);
    void UpdGrid1(char* sGrid1Coluna1, char *sGrid1Coluna2, char
*sGrid1Coluna3, char *sGrid1Coluna4, char *sGrid1Coluna5, int sGrid1ID);
    void DelGrid1(int sGrid1ID);
};
#endif
```

Código 18 - db_Grid1.h

3.1.7 *Screenshots* da aplicação

No presente tópico serão apresentados *screenshots* do protótipo desenvolvido. Os *screenshots* irão abordar algumas das operações mais comuns, tais como obter, adicionar, editar e gravar dados.

Na Figura 27 é mostrada uma visão geral do protótipo. Como foi referido anteriormente, o protótipo dispõe de duas *Grids*: Grid1 com 5 colunas visíveis e uma escondida (*Grid1ID*) e GridTab1 com 3 colunas visíveis e mais 2 escondidas (*GridTab1ID* e *Grid1ID*). Cada uma das *Grids* possui botões para efetuar as operações mais comuns na base de dados, ou seja, o botão "Novo" adiciona uma linha na respetiva *Grid* para inserir o novo registo (tendo sempre que pressionar o botão "Gravar" para inserir os registos na base de dados), "Gravar" (para eventuais alterações aos registos atuais na *Grid* respetiva) e Apagar. Para efeitos da obtenção de dados, na primeira *Grid* (Grid1), o carregamento é feito no início da aplicação e na segunda *Grid* (GridTab1), o carregamento é feito após selecionar uma linha da Grid1.

Figura 27 - Vista geral do protótipo 1

É apresentada na Figura 28 um exemplo de inserção de registo na Grid1 (após pressionar o botão "Novo", ficando o botão "Gravar" habilitado e o "Novo" desabilitado).

Quando é feita a inserção (sem gravar na base de dados), cada registo/coluna não gravado/a, contém uma marca vermelha no canto superior esquerdo.

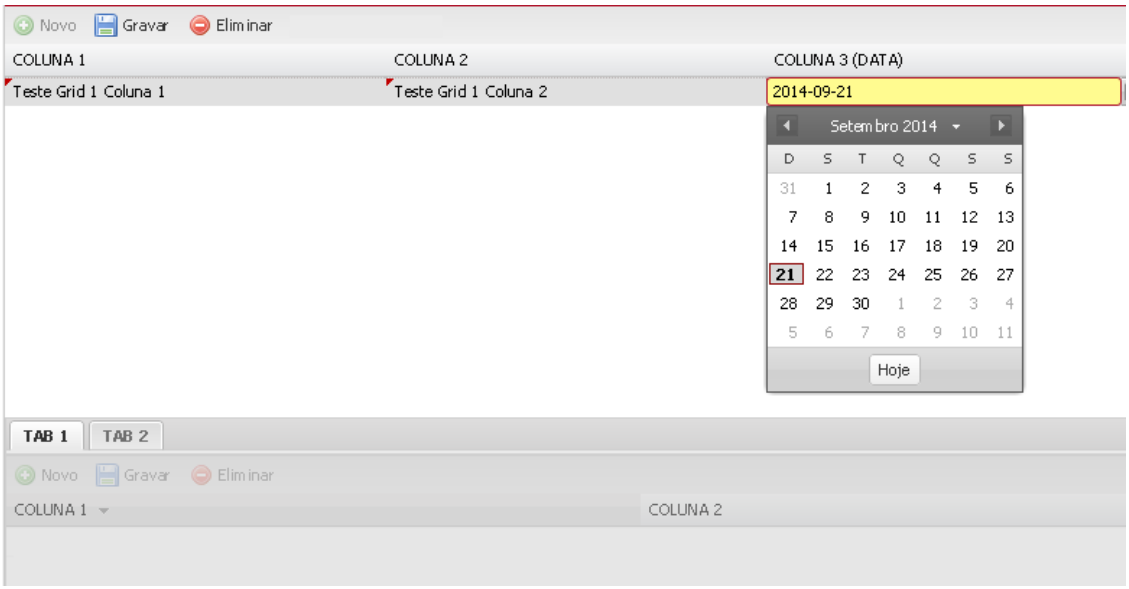

Figura 28 - Exemplo de inserção de dados na Grid1

É apresentada na Figura 29 o início da aplicação com valores na Grid1. Os botões "Novo" e "Eliminar" da respetiva *Grid* encontram-se habilitados e o "Gravar" desabilitado. Como é um registo que provém da base de dados e está selecionado, a GridTab1 está habilitada para permitir consultar /adicionar/editar/apagar itens.

Na Figura 30 é apresentado um exemplo de como inserir registos na GridTab1 (conforme foi referido anteriormente, os registos que ainda não foram guardados na base de dados, possuem uma marca vermelha no canto superior esquerdo do mesmo). Para fazer a introdução dos dados, deve-se fazer duplo clique sobre a(s) coluna(s) que se pretende alterar e após concluir, o utilizador deve pressionar o botão "Gravar". Este processo automaticamente envia o *Grid1ID* para ser guardado na base de dados.

É apresentada na Figura 31 a GridTab1 com valores provenientes da base de dados. Os botões "Novo" e "Eliminar" da respetiva *Grid* encontram-se habilitados e o "Gravar" desabilitado. Para editar os dados, basta fazer duplo clique em cima do campo da *Grid* que se pretende alterar.

Figura 29 - Exemplo do carregamento de dados na Grid1

Figura 30 - Exemplo de inserção de dados na GridTab1

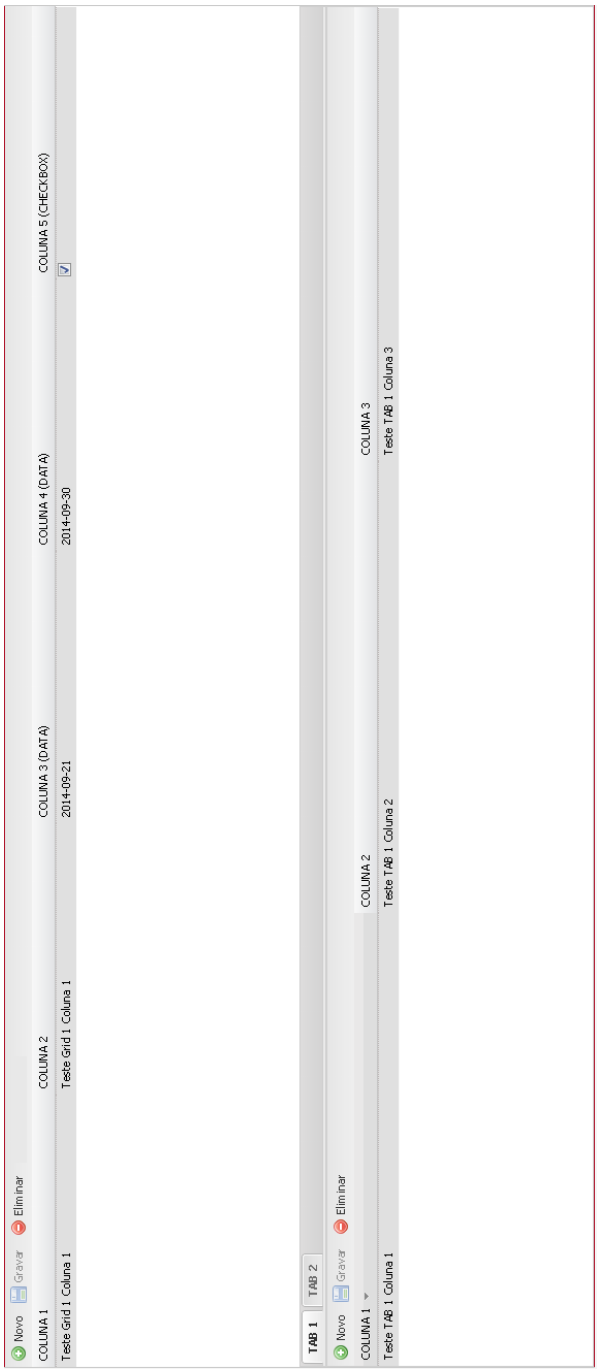

Figura 31 - Exemplo do carregamento de dados na GridTab1

3.2 Implementação 2

No presente tópico irão ser apresentadas as questões mais técnicas relativas ao protótipo 2, nomeadamente a nível de arquitetura aplicacional utilizada para o desenvolver (que será uma arquitetura monolítica, conforme referido na secção 2.5.4).

Será apresentado o modelo de domínio da implementação, bem como diagramas de classe e sequência que permitirão visualizar o fluxo da informação, desde a camada de *front-end* até ao *back-end*.

Por fim, serão apresentados exemplos do código utilizado no presente protótipo, bem como *screenshots* da aplicação em funcionamento.

3.2.1 Arquitetura aplicacional

A arquitetura da aplicação do protótipo 2 é apresentada na Figura 32. O protótipo foi desenvolvido através das seguintes tecnologias:

- Java para *front-end*, *middle-tier* e *back-end*
- *JDBC* para conexão à base de dados Oracle

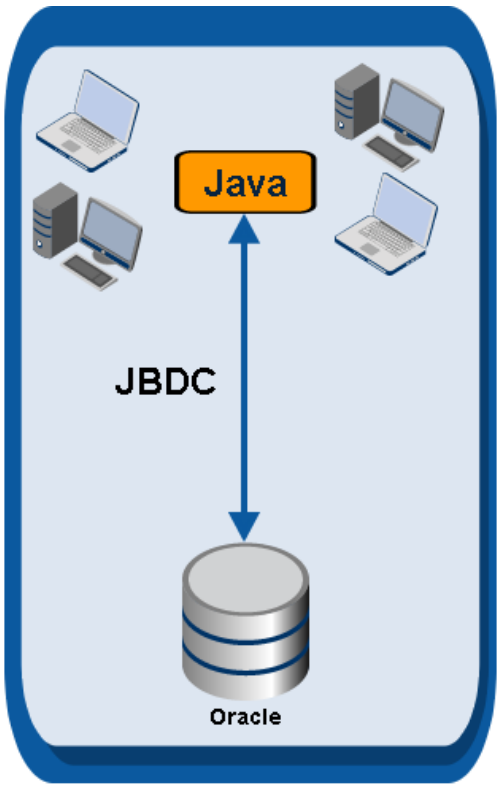

Figura 32 - Arquitetura do protótipo 2.

Uma vez que, um dos objetivos principais desta dissertação será comparar a vantagem entre articular várias linguagens de programação, em vez de uma aplicação monolítica, foram desenvolvidas aplicações que possuem o mesmo SGBD, mas com tabelas e acessos diferentes.

3.2.2 Modelo de domínio

Na segunda implementação, visando o que foi referido na secção 3.2.1, foi adotado o modelo de domínio referido na Figura 33.

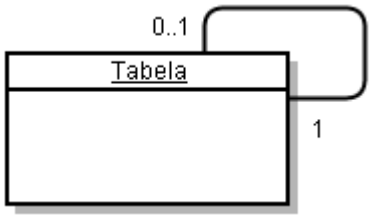

Figura 33 - Modelo de domínio da Implementação 2

A Tabela irá ter 5 colunas, das quais 3 visíveis para o utilizador e 2 campos escondidos. Poderá ter 0 ou 1 ocorrências mediante o valor do *ID* (chave primária).

3.2.3 Diagrama de classes

O diagrama de classes para a presente implementação está apresentado na Figura 34. Como se trata de uma aplicação monolítica, optou-se por juntar as camadas de *software* num único diagrama.

Para a camada de *front-end* foi criada a classe UI com os métodos que suportem as operações mais comuns a uma base de dados (Seleção, Inserção, Edição e Remoção). Uma vez que, só existe uma tabela na base de dados, serão feitas as pesquisas de duas formas, na base de dados: dos IDs (*GetId*) e por IDs (*GetById*).

Para a camada de *middle-tier* foi criada a classe Controller que suportará todas as operações solicitadas pela camada de *front-end* e encaminhá-las-á para a camada de *back-end*.

Por fim, para a camada de *back-end* foi criada a classe Database que, para além de ter todos os métodos que suportem os pedidos feitos pela camada *middle-tier*, irá utilizar um *JBDC* que permitirá a conexão (*GetConnection*) e o fecho da mesma (*CloseConnection*) a um sistema de gestão de base de dados Oracle.

A forma de transição de informação entre as diversas camadas é feita (com a exceção dos os métodos de acesso – *getter*) por variáveis booleanas, onde no caso de sucesso da operação retornam *true* e no caso de falha devolvem *false*.

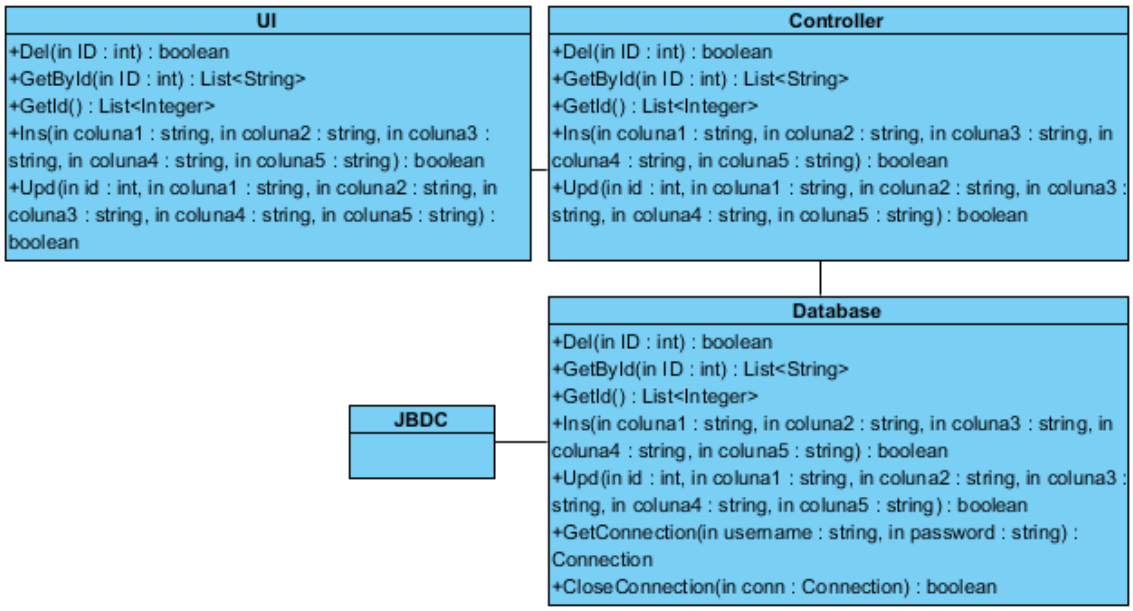

Figura 34 - Diagrama de classes para o protótipo 2

3.2.4 Diagramas de sequência

É apresentada na Figura 35 o diagrama de sequência para a obtenção de *IDs*. É feito um pedido na classe UI, onde é invocado o método *GetId* do Controller que, por sua vez, vai à classe Database executar um método com o mesmo nome.

A classe Database vai tentar obter conexão à base de dados e caso não tenha sucesso, informa o utilizador que não foi possível obter conexão à base de dados, caso contrário tenta executar uma *query* na base de dados para obter os *IDs*, devolvendo posteriormente a lista dos *IDs* e preenchendo, na UI, a *combobox* com os mesmos.

É apresentada na Figura 36 o diagrama de sequência para a obtenção de dados através de *ID*. É feito um pedido na classe UI, onde é invocado o método *GetById* do Controller (com o parâmetro *ID* proveniente da *combobox*) que, por sua vez, vai à classe Database executar um método com o mesmo nome.

A classe Database vai tentar obter conexão à base de dados e caso não tenha sucesso, informa o utilizador que não foi possível obter conexão à base de dados, caso contrário tenta executar uma *query* na base de dados, passando como parâmetro o *ID* para obter os dados, devolvendo posteriormente a lista dos dados e preenchendo, na UI, as caixas de texto com os mesmos.

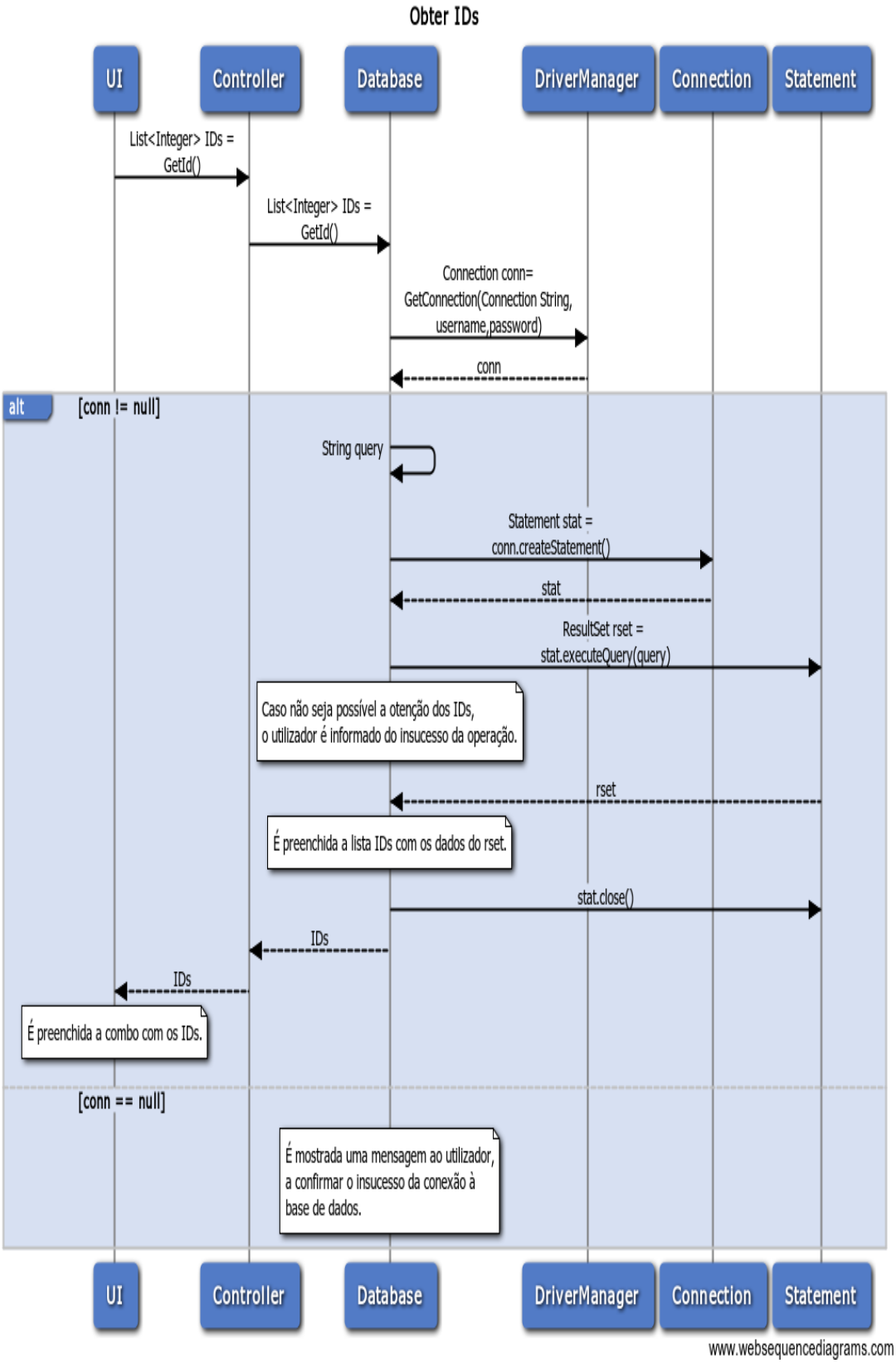

Figura 35 - Obtenção de IDs

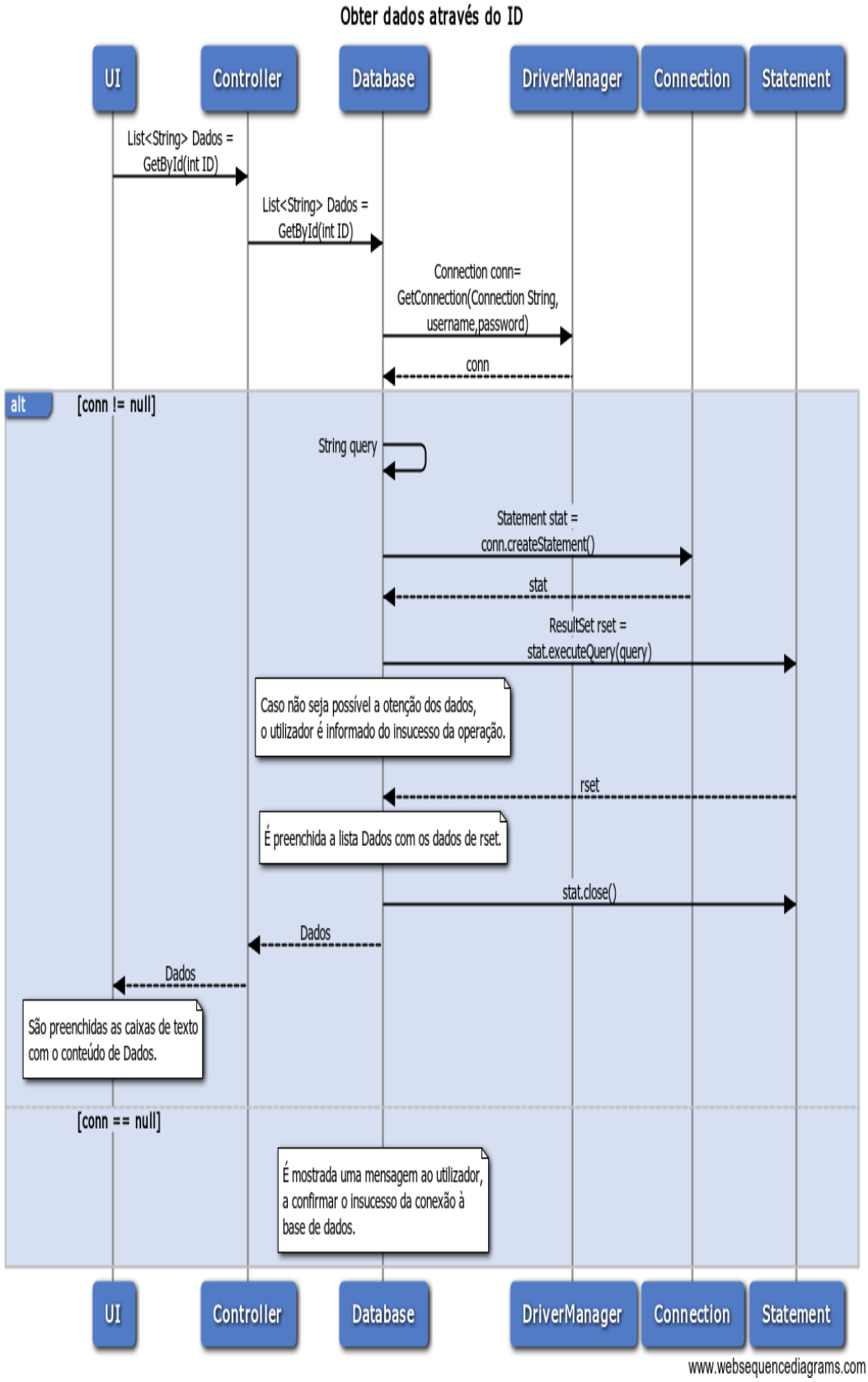

Figura 36 - Obtenção de dados através de ID

É apresentada na Figura 37 o diagrama de sequência para a inserção de dados. É feito um pedido na classe UI, onde é invocado o método *Ins* do Controller (com os parâmetros provenientes das caixas de texto) que, por sua vez, vai à classe Database executar um método com o mesmo nome. A classe Database vai tentar obter conexão à base de dados e caso não tenha sucesso, informa o utilizador que não possível obter conexão à base de dados, caso contrário tenta executar uma *query* na base de dados, passando como parâmetros os valores das caixas de texto. Caso este processo tenha sucesso (usado para isso a condição sucesso=1), o retorno é feito através de sucessivos *true* e o utilizador é informado do sucesso da operação, caso contrário, são devolvidos sucessivos *false* e é mostrada, ao utilizador, uma mensagem de erro.

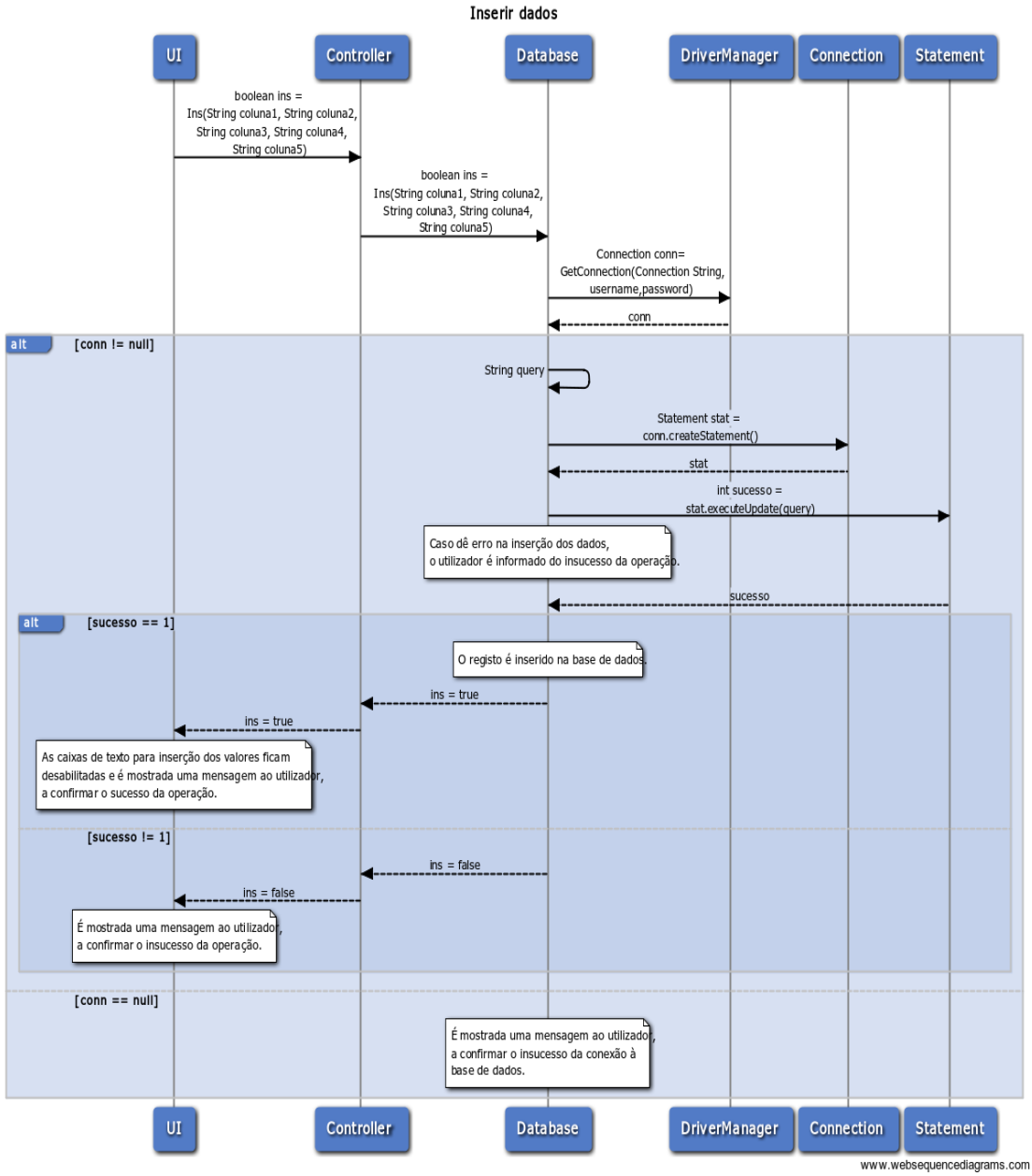

Figura 37 - Inserir dados no protótipo 2

É apresentada na Figura 38 o diagrama de sequência para a edição de dados. É feito um pedido na classe UI, onde é invocado o método *Upd* do Controller (com os parâmetros provenientes da *combobox* e das caixas de texto) que, por sua vez, vai à classe Database executar um método com o mesmo nome.

A classe Database vai tentar obter conexão à base de dados e caso não tenha sucesso, informa o utilizador que não foi possível obter conexão à base de dados, caso contrário tenta executar uma *query* na base de dados, passando como parâmetros os valores da *combobox* e das caixas de texto. Caso este processo tenha sucesso (usando para isso a condição sucesso=1), o retorno é feito através de sucessivos *true* e o utilizador é informado do sucesso da operação, caso contrário, são devolvidos sucessivos *false* e é mostrada, ao utilizador, uma mensagem de erro.

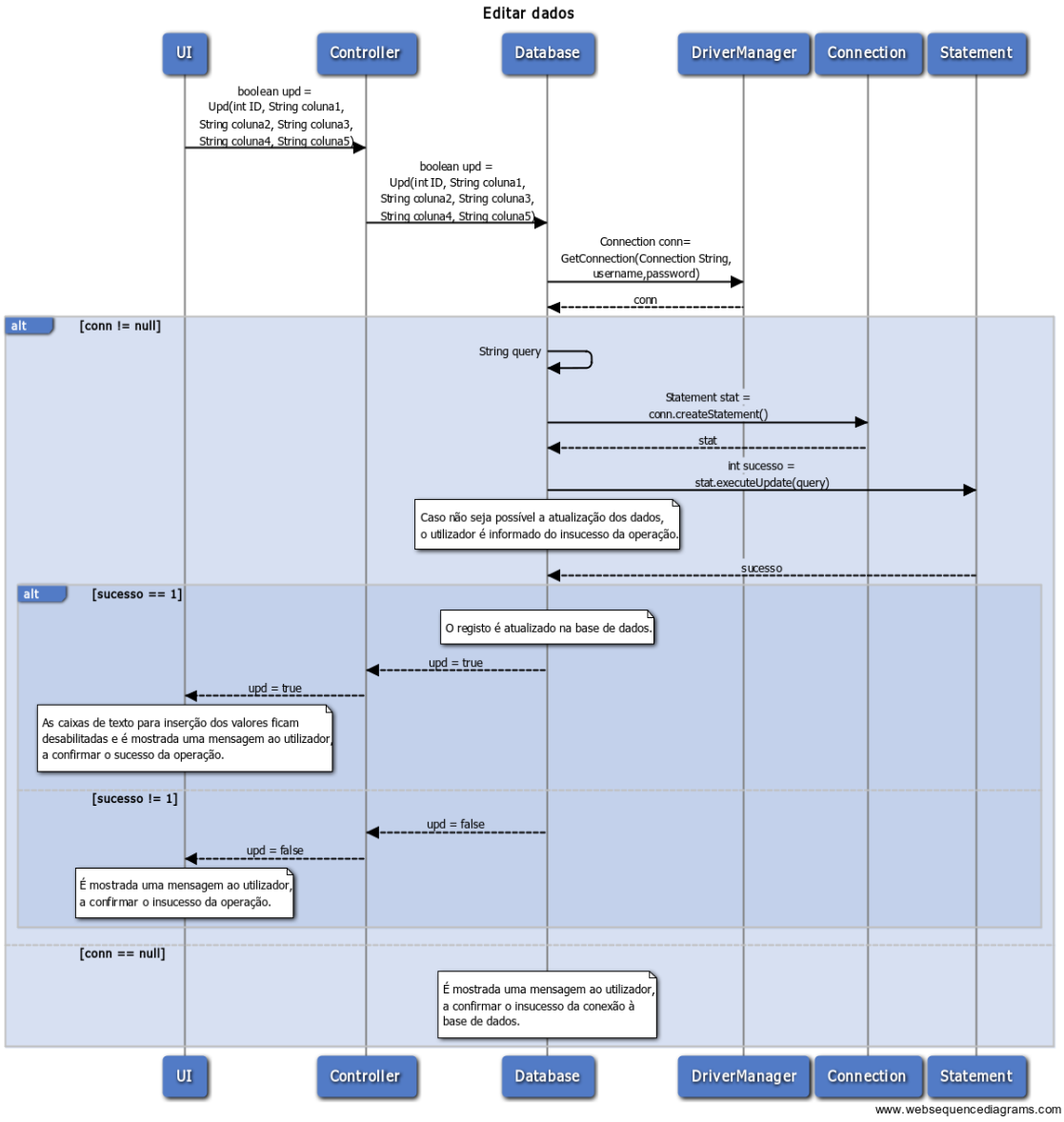

Figura 38 - Editar dados no protótipo 2

3.2.5 Tabelas da base de dados

Na presente implementação foi criada uma tabela, conforme a Figura 39.

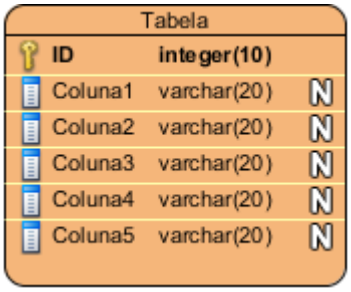

Figura 39 - Tabela utilizadas na base de dados do protótipo 2

Esta tabela irá servir para preenchimento da *combobox* e das caixas de texto, já referidas nas seções 3.2.2 e 3.2.3.

Por motivos de simplificação da manipulação de dados, optou-se por colocar todos os campos da base de dados (excetuando a chave primária) como *VARCHAR*, pois os campos serem diferentes de *VARCHAR*, não iriam criar grande impacto na secção da Análise de Resultados (secção 3.4).

Na secção 3.2.7 serão mostrados *screenshots* da aplicação, onde se poderá verificar o processo de operação para selecionar elementos da *combobox* e/ou efetuar o preenchimento das caixas de texto.

3.2.6 Exemplos do código utilizado no protótipo

Neste tópico serão apresentados exemplos do código utilizado para a criação do presente protótipo. Serão apresentados exemplos a nível de *front-end*, *middle-tier* e *back-end*.

Front-end

É apresentado no Código 19 o método que permite obter os IDs para preencher a *combobox* do *front-end*. Este método executa o método *GetId* da classe Controller, obtendo o retorno numa lista de valores inteiros. Após a obtenção dos valores, a *combobox* é limpa e preenchida com os valores provenientes da lista de inteiros. Caso este método dê erro, é mostrada uma mensagem ao utilizador a informar que houve um erro ao obter os IDs.

```
private void btnGetActionPerformed(java.awt.event.ActionEvent evt) {
        Controller controller = new Controller();
        try {
            List<Integer> IDs = controller.GetId();
            cmbId.removeAllItems();
            for (Iterator<Integer> i = IDs.iterator(); i.hasNext();) {
                int id = i.next();
                String sId = "" + id;
                cmbId.addItem(sId);
```

```
                sId = "";
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao obter os IDs!",
"Erro", JOptionPane.ERROR_MESSAGE);
        }
}
```

Código 19 - Método do front-end para obter IDs

No Código 20 é apresentado o método que permite obter os dados através de um ID, para preencher as caixas de texto do *front-end*. Este método executa o método *GetById* da classe Controller, passando como parâmetro o valor da *combobox* dos IDs e obtendo o retorno numa lista de *Strings*. Após a obtenção dos valores, as caixas de texto são preenchidas, com os valores provenientes da lista de *Strings*.

Caso este método dê erro, é mostrada uma mensagem ao utilizador a informar que houve um erro ao obter os valores do ID passado pela *combobox*.

```
private void cmbIdActionPerformed(java.awt.event.ActionEvent evt) {
        Controller controller = new Controller();
        try {
            List<String> Dados =
controller.GetById(Integer.parseInt(cmbId.getSelectedItem().toString()));
            txtColuna1.setText(Dados.get(0));
            txtColuna2.setText(Dados.get(1));
            txtColuna3.setText(Dados.get(2));
            txtColuna4.setText(Dados.get(3));
            txtColuna5.setText(Dados.get(4));
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao obter os valores
do ID!", "Erro", JOptionPane.ERROR_MESSAGE);
        }
}
```

Código 20 - Método do front-end para obtenção de dados através de ID

É apresentado no Código 21 o método que permite gravar os valores das caixas de texto do *front-end*. Este método executa o método *Ins* da classe Controller, obtendo o retorno numa variável booleana (sucesso). Caso a variável booleana esteja a *true*, o utilizador é informado do sucesso da operação, caso contrário, é informado que não foi possível concluir a operação de inserção dos dados. Caso este método dê erro, é mostrada uma mensagem ao utilizador a informar que houve um erro ao gravar os dados.

```
private void btnSaveAddActionPerformed(java.awt.event.ActionEvent evt) {
        Controller controller = new Controller();
        boolean sucesso = false;
        try {
            sucesso = controller.Ins(txtColuna1.getText(),
txtColuna2.getText(), txtColuna3.getText(), txtColuna4.getText(),
txtColuna5.getText());
            if (sucesso) {
                /* Disable das caixas de texto */
                JOptionPane.showMessageDialog(null, "Dados inseridos com
sucesso!", "Sucesso na operação", JOptionPane.INFORMATION_MESSAGE);
```

```
            } else {
                /* Enable das caixas de texto */
                JOptionPane.showMessageDialog(null, "Não foi possível
concluir a operação de inserção dos dados!", "Erro",
JOptionPane.ERROR_MESSAGE);
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao gravar!", "Erro",
JOptionPane.ERROR_MESSAGE);
        }
}
```

Código 21 - Método do front-end para inserir dados

É apresentado no Código 22 o método que permite gravar os valores editados das caixas de texto do *front-end*. Este método executa o método *Upd* da classe Controller, obtendo o retorno numa variável booleana (sucesso). Caso a variável booleana esteja a *true*, o utilizador é informado do sucesso da operação, caso contrário, é informado que não foi possível concluir a operação de atualização dos dados.

Caso este método dê erro, é mostrada uma mensagem ao utilizador a informar que houve um erro ao atualizar os dados.

```
private void btnSaveEditActionPerformed(java.awt.event.ActionEvent evt) {
        Controller controller = new Controller();
        boolean sucesso = false;
        try {
            sucesso =
controller.Upd(Integer.parseInt(cmbId.getSelectedItem().toString()),
txtColuna1.getText(), txtColuna2.getText(), txtColuna3.getText(),
txtColuna4.getText(), txtColuna5.getText());
            if (sucesso) {
                /* Disable das caixas de texto */
                JOptionPane.showMessageDialog(null, "Dados atualizados com
sucesso!", "Sucesso na operação", JOptionPane.INFORMATION_MESSAGE);
            } else {
                /* Enable das caixas de texto */
                JOptionPane.showMessageDialog(null, "Não foi possível
concluir a operação de atualização dos dados!", "Erro",
JOptionPane.ERROR_MESSAGE);
            }
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao atualizar!",
"Erro", JOptionPane.ERROR_MESSAGE);
        }
}
```

Código 22 - Método do front-end para gravar edição de dados

Middle-tier

No Código 23 é apresentado o método que permite obter os IDs para, posteriormente preencher a *combobox* do *front-end*. Este método cria uma instância do *back-end*, fazendo de seguida o retorno do método com o mesmo nome da classe Database.

```
public List<Integer> GetId() throws SQLException {
        Database database = new Database();
        return database.GetId();
}
```

Código 23 - Método do *middle-tier* para obtenção de IDs

É apresentado no Código 24 o método que permite obter os dados através do ID para, posteriormente preencher as caixas de texto do *front-end*. Este método cria uma instância do *back-end*, fazendo de seguida o retorno do método com o mesmo nome da classe Database.

```
public List<String> GetById(int ID) throws SQLException {
        Database database = new Database();
        return database.GetById(ID);
}
```

Código 24 - Método do *middle-tier* para obter dados por ID

No Código 25 é apresentado o método que permite inserir dados contidos nas caixas de texto do *front-end*. Este método cria uma instância do *back-end*, fazendo de seguida o retorno do método com o mesmo nome da classe Database.

```
public boolean Ins(String coluna1, String coluna2, String coluna3, String
coluna4, String coluna5) throws SQLException {
        Database database = new Database();
        return database.Ins(coluna1, coluna2, coluna3, coluna4, coluna5);
}
```

Código 25 - Método do *middle-tier* para inserção de dados

É apresentado no Código 26 o método que permite atualizar os dados contidos nas caixas de texto do *front-end*, passando como parâmetro o conteúdo da *combobox*. Este método cria uma instância do *back-end*, fazendo de seguida o retorno do método com o mesmo nome da classe Database.

```
public boolean Upd(int ID, String coluna1, String coluna2, String coluna3,
String coluna4, String coluna5) throws SQLException {
        Database database = new Database();
        return database.Upd(ID, coluna1, coluna2, coluna3, coluna4,
coluna5);
}
```

Código 26 - Método do *middle-tier* para atualização de dados

Back-end

É apresentado no Código 27 o método que permite, em *back-end*, obter os IDs para preencher a *combobox* em *front-end*.

Este método testa se existe conexão com a base de dados, caso exista, tenta executar uma *query* que vá buscar os IDs. Após execução da *query*, é preenchida a lista de inteiros IDs, fechado o *statement* e devolvida a lista.

```
public List<Integer> GetId() throws SQLException {
        String username = "user";
        String password = "pass";
        List<Integer> IDs = new ArrayList<Integer>();
        Connection conn = GetConnection(username, password);

        if (conn != null) {
            String query = "select ID from Tabela";
            ResultSet rset = null;
            Statement stat = null;
            try {
                stat = conn.createStatement();
                rset = stat.executeQuery(query);
                while (rset.next()) {
                    IDs.add(rset.getInt("ID"));
                }
                stat.close();
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(null, "Erro ao obter os IDs!",
"Erro", JOptionPane.ERROR_MESSAGE);
            }
        }
        return IDs;
}
```

Código 27 - Método do *back-end* para obter IDs

É apresentado no Código 28 o método que permite, em *back-end*, obter os dados para preencher as caixas de texto em *front-end*, passando como parâmetro o valor proveniente da *combobox*.

Este método testa se existe conexão com a base de dados, caso exista, tenta executar uma *query* que vá buscar os dados. Após execução da *query*, é preenchida a lista de *strings* Dados, fechado o *statement* e devolvida a lista.

```
public List<String> GetById(int ID) throws SQLException {
        String username = "user";
        String password = "pass";
        List<String> Dados = new ArrayList<String>();
        Connection conn = GetConnection(username, password);
        if (conn != null) {
            String sId = "" + ID;
            String query = "SELECT coluna1, coluna2, coluna3, coluna4,
coluna5 FROM Tabela WHERE ID=" + sId;
            ResultSet rset = null;
            Statement stat = null;
            try {
                stat = conn.createStatement();
                rset = stat.executeQuery(query);
                while (rset.next()) {
                    Dados.add(rset.getString("coluna1"));
                    Dados.add(rset.getString("coluna2"));
                    Dados.add(rset.getString("coluna3"));
                    Dados.add(rset.getString("coluna4"));
                    Dados.add(rset.getString("coluna5"));
                }
                stat.close();
            } catch (SQLException ex) {
```

```
                JOptionPane.showMessageDialog(null, "Erro ao obter os
Dados!", "Erro", JOptionPane.ERROR_MESSAGE);
            }
        }
        return Dados;
}
```

Código 28 - Método do *back-end* para obtenção de dados por ID

É apresentado no Código 29 o método que permite, em *back-end*, a inserção de dados na base de dados, provenientes das caixas de texto do *front-end*.

Este método testa se existe conexão com a base de dados, caso exista, tenta executar uma *query* que vá inserir os dados. Após execução da *query*, é testado o estado da variável sucesso (criada para validar a execução da *query*) e caso esteja a 1, o método retorna *true*, caso contrário *false*.

```
public boolean Ins(String coluna1, String coluna2, String coluna3, String
coluna4, String coluna5) throws SQLException {
        String username = "user";
        String password = "pass";
        Connection conn = GetConnection(username, password);
        if (conn != null) {
            String query = "INSERT INTO Tabela
(coluna1,coluna2,coluna3,coluna4,coluna5) VALUES ('" + coluna1 + "','" +
coluna2 + "','" + coluna3 + "','" + coluna4 + "','" + coluna5 + "')";
            Statement stat = null;
            int sucesso = 0;
            try {
                stat = conn.createStatement();
                sucesso = stat.executeUpdate(query);
                if (sucesso == 1) {
                    return true;
                } else {
                    return false;
                }
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(null, "Erro ao executar a
query para inserir os dados!", "Erro", JOptionPane.ERROR_MESSAGE);
            }
        }
        return false;
}
```

Código 29 - Método do *back-end* para inserção de dados

É apresentado no Código 30 o método que permite, em *back-end*, a atualização de dados na base de dados, provenientes das caixas de texto e *combobox* do *front-end*.

Este método testa se existe conexão com a base de dados, caso exista, tenta executar uma *query* que vá atualizar os dados. Após execução da *query*, é testado o estado da variável sucesso (criada para validar a execução da *query*) e caso esteja a 1, o método retorna *true*, caso contrário *false*.

```
public boolean Upd(int ID, String coluna1, String coluna2, String coluna3,
String coluna4, String coluna5) throws SQLException {
        String username = "user";
        String password = "pass";
        Connection conn = GetConnection(username, password);
        if (conn != null) {
            String query = "UPDATE Tabela SET coluna1='" + coluna1 +
"',coluna2='" + coluna2 + "',coluna3='" + coluna3 + "',coluna4='" + coluna4
+ "',coluna5='" + coluna5 + "' WHERE ID=" + ID;
            Statement stat = null;
            int sucesso = 0;
            try {
                stat = conn.createStatement();
                sucesso = stat.executeUpdate(query);
                if (sucesso == 1) {
                    return true;
                } else {
                    return false;
                }
            } catch (SQLException ex) {
                JOptionPane.showMessageDialog(null, "Erro ao executar a
query para atualizar dados!", "Erro", JOptionPane.ERROR_MESSAGE);
            }
        }
        return false;
}
```

Código 30 - Método do *back-end* para atualização de dados

No Código 31 é apresentado o método que permite, em *back-end*, a conexão com a base de dados através de um *JDBC* da Oracle (ojdbc7.jar, SHA1 Checksum: 7c9b5984b2c1e32e7c8cf3331df77f31e89e24c2). O equipamento onde está alojada a base de dados tem o nome **PC**, a porta de ligação é a **1521**, a base de dados é a **tmdei**.

Caso seja possível a conexão com a base de dados, o método retorna essa instância de conexão, caso contrário retorna *null*.

```
public Connection GetConnection(String username, String password) throws
SQLException {
        Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@//PC:1521/tmdei", username,
password);
        if (conn == null) {
            JOptionPane.showMessageDialog(null, "Erro a obter conexão à
base de dados!", "Erro", JOptionPane.ERROR_MESSAGE);
            return null;
        }
        return conn;
}
```

Código 31 - Método do back-end para conexão à base de dados

No Código 32 é apresentado o método que permite, em *back-end*, o fecho da conexão com a base de dados. Caso não seja possível fechar a conexão com a base de dados, o método retorna *false*, caso contrário, retorna *true*.

```
public boolean CloseConnection(Connection conn) {
        try {
            conn.close();
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(null, "Erro ao efectuar o fecho
da conexão!", "Erro", JOptionPane.ERROR_MESSAGE);
        }
        if (conn != null) {
            return false;
        }
        return true;
    }
```

Código 32 - Método do back-end para fecho da conexão à base de dados

3.2.7 *Screenshots* da aplicação

É apresentado na Figura 40 o ecrã inicial do protótipo 2. Por defeito, todos os campos se encontram no modo desabilitado, tendo para isso, o utilizador que pressionar o botão "Obter IDs" (ver Figura 35, para ver a sequência de passos que o protótipo usa ou a Figura 41, para ver o resultado final) ou o botão "Inserir" (ver Figura 43).

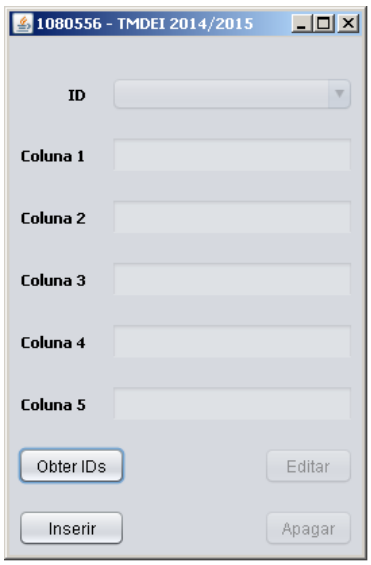

Figura 40 - Apresentação inicial do protótipo

Na Figura 41 é apresentado o ecrã após ter pressionado o botão "Obter IDs". Após o fluxo representado na Figura 35, a *combobox* apresenta todos os IDs disponíveis na base de dados.

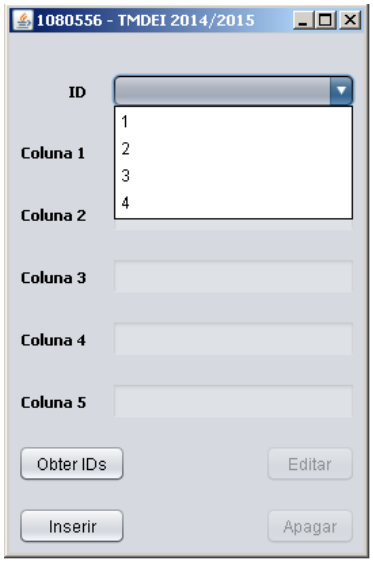

Figura 41 – Listagem dos Ids

É apresentado na Figura 42 o resultado final após a escolha de um ID da *combobox*. Ao efetuar esta escolha é percorrido o fluxo representado na Figura 36 (diagrama de sequência " Obtenção de dados através de ID ") e as caixas de texto são desabilitadas.

Figura 42 - Escolha de um ID

Na Figura 43 é apresentado o *screenshot* resultante do clique no botão "Inserir". Com este processo, as caixas de texto ficam habilitadas, os botões "Gravar" e "Cancelar" ficam ativos/visíveis, os botões "Inserir" e o "Apagar" ficam escondidos e o "Editar" fica desabilitado.

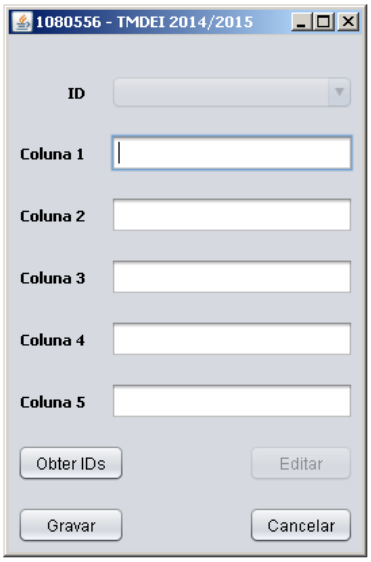

Figura 43 – Inserção de registos

É apresentado na Figura 44 o ecrã resultante da gravação de um registo (após pressionar o botão "Inserir"). Para um enquadramento na sequência das operações ver Figura 37 (diagrama de sequência " Inserir dados no protótipo 2"). As caixas de texto ficam desabilitadas, o ID é atualizado através do envio da informação por parte da base de dados. O botão "Inserir" fica desabilitado para evitar erros de utilização, sendo que para ficar habilitado, a *combobox* terá que ter o item selecionado semelhante à Figura 41, ou seja, vazio.

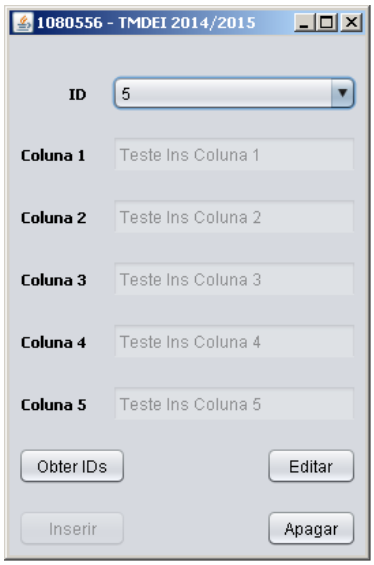

Figura 44 - Após a gravação dos dados, tendo previamente carregado no botão "Inserir"

Na Figura 45 é apresentado o ecrã resultante da ação do botão "Editar". As caixas de texto ficam habilitadas e a *combobox* fica desabilitada. Os botões "Gravar" e "Cancelar" ficam ativos/visíveis, os botões "Editar" e o "Apagar" ficam escondidos e o "Inserir" fica desabilitado.

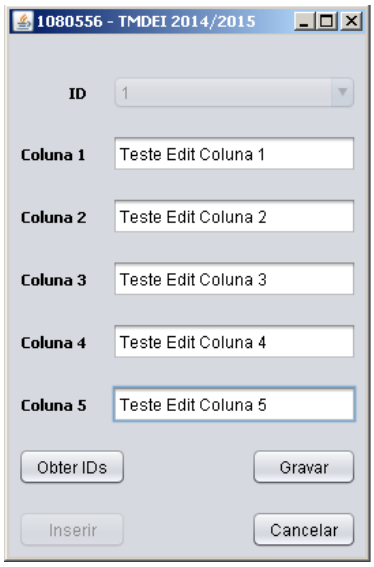

Figura 45 - Atualização de dados

É apresentado na Figura 46 o ecrã resultante da gravação de um registo (após pressionar o botão "Editar"). Para um enquadramento na sequência das operações ver Figura 38 (diagrama de sequência " Editar dados no protótipo 2"). As caixas de texto ficam desabilitadas, o botão "Inserir" fica desabilitado para evitar erros de utilização, sendo que para ficar habilitado, a *combobox* terá que ter o item selecionado semelhante à Figura 41, ou seja, vazio.

Figura 46 - Após edição de dados

3.3 Desenho da experiência

É apresentada na Figura 47 o desenho da experiência. Para efetuar os testes aos protótipos, foram utilizadas 1 máquina cliente e 1 máquina servidor. A rede informática utilizada foi rede local a 100 Mb/s, com recurso a um switch 3COM e cabos de ligação (entre o Cliente-Switch e Switch-Servidor) diretos.

As especificações técnicas da máquina cliente estão apresentadas na Tabela 7. Foi utilizado um equipamento da marca Dell, modelo Latitude E5420.

Para a máquina servidora foi utilizado um equipamento da marca Dell, modelo Latitude E7440 e as suas características técnicas estão apresentadas na Tabela 8. Sobre o sistema operativo que o equipamento servidor possui, para além do sistema de gestão de base de dados Oracle, encontra-se uma máquina virtual que permite fazer de servidor C++ e *Web*. As suas características estão apresentadas na Tabela 9.

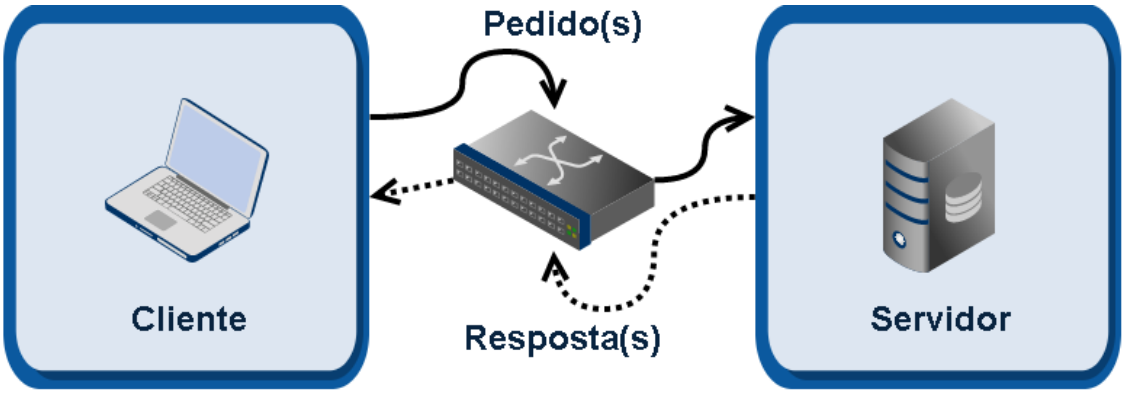

Figura 47 - Desenho da experiência

Tabela 7 - Características do equipamento cliente

Componente	Quantidade/Descrição
Processador	Intel Core i5 (2nd Gen) 2520M / 2.5 GHz
Memória RAM	4 GB DDR3 SDRAM 1333 MHz
Disco Rígido	320 GB Serial ATA-300
Sistema Operativo	Windows 7 SP1 Enterprise 64-bit
Browser	Google Chrome V41.0.2272.118 m

Tabela 8 - Características do equipamento servidor

Componente	Quantidade/Descrição
Processador	Intel Core i5-4310U 2.6 Ghz
Memória RAM	8 GB DDR3 1600 MHz PC3 12800
Disco Rígido	256 GB Samsung SSD PM851 mSATA SCSI
Sistema Operativo	Windows 7 SP1 Professional 64-bit
Browser	Google Chrome V41.0.2272.101 m
Base de Dados	Oracle Database Express Edition 11g Release 2

Tabela 9 - Características do equipamento virtual

Componente	Quantidade/Descrição
Processador	Intel Core i5-4310U 2.6 Ghz (2 Cores)
Memória RAM	4 GB
Disco Rígido	50 GB
Rede	NAT
Sistema Operativo	Solaris 10 64-bit
Browser	Firefox 31.0
Web Server	Glassfish 2.1
Java	Java SE Development Kit 8u45

Para efetuar os testes aos protótipos irão ser efetuadas 15 experiências, a cada uma das operações mais comuns sobre uma base de dados, ou seja, consulta, inserção, atualização e remoção de registos. Estas experiências irão ser realizadas através da execução de 1 *query* e 10 *queries* à base de dados. Como já foi referido anteriormente, o processo de remover dados é semelhante ao atualizar (nomeadamente a nível da passagem de parâmetros para efetuar uma operação), como tal, não irá ser contemplado nas experiências.

Para medir os tempos de resposta no protótipo 1 foi adicionado, no início de cada método, o código apresentado no Código 33 e no fim de cada método, o código apresentado no Código 34. Desta forma é possível saber a duração de tempo (em milissegundos) que o método demorou desde que o utilizador solicitou a operação, até visualizar os resultados. Uma vez que este protótipo só tem duas tabelas, foi decidido simular os tempos de resposta da consulta com base na média do somatório entre a consulta da Grid1 (método *GetGrid1*) e a consulta da GridTab1 (método *GetGridTab1*).

```
var startTime = new Date().getTime();
```

Código 33 - Captura da hora do sistema ao iniciar o método no protótipo 1

```
var endTime = new Date().getTime();
var duration = endTime - startTime;
alert(duration);
```

Código 34 - Tempo de duração e impressão do mesmo para o protótipo 1

Para medir os tempos de resposta no protótipo 2 foi adicionado, no início de cada método, o código apresentado no Código 35 e no fim de cada método, o código apresentado no Código 36. Desta forma é possível saber a duração de tempo (em milissegundos) que o método demorou desde que o utilizador solicitou a operação, até visualizar os resultados. Uma vez que este protótipo só tem uma tabela, foi decidido simular os tempos de resposta da consulta com base na média do somatório de tempos entre a consulta de ID (método *GetId*) e a consulta por ID (método *GetById*).

```
long startTime = System.nanoTime();
```

Código 35 - Captura da hora do sistema ao iniciar o método no protótipo 2

```
long endTime = System.nanoTime();
long duration = (endTime - startTime) / 1000000;
System.out.println(duration);
```

Código 36 - Tempo de duração e impressão do mesmo para o protótipo 2

Os vários testes feitos aos protótipos foram feitos em horário pós-laboral (a partir das 22:00), de maneira a terem menor impacto na experiência e poder ter resultados mais credíveis, pois a carga na rede informática é diminuta/inexistente.

Os resultados são apresentados na Tabela 10 e Tabela 11 (para o protótipo 1) e a na Tabela 12 e Tabela 13 (para o protótipo 2).

Tabela 10 - Testes de consulta, inserção e edição de registos no protótipo 1

Teste de Consulta de Registos (Grid1 + GridTab1)	Tempo de Resposta (ms)	Teste de Inserção de Registos (Grid1)	Tempo de Resposta (ms)	Teste de Edição de Registos (Grid1)	Tempo de Resposta (ms)
1	343	**1**	31	**1**	32
2	94	**2**	47	**2**	45
3	93	**3**	32	**3**	47
4	94	**4**	49	**4**	31
5	78	**5**	34	**5**	44
6	93	**6**	16	**6**	46
7	94	**7**	29	**7**	31
8	79	**8**	44	**8**	32
9	156	**9**	33	**9**	234
10	109	**10**	26	**10**	38
11	93	**11**	42	**11**	109
12	406	**12**	44	**12**	42
13	95	**13**	21	**13**	40
14	107	**14**	43	**14**	49
15	94	**15**	31	**15**	33

Tabela 11 - 10 testes de consulta, inserção e edição de registos no protótipo 1

Teste de Consulta de Registos (Grid1 + GridTab1)	Tempo de Resposta (ms)	Teste de Inserção de Registos (Grid1)	Tempo de Resposta (ms)	Teste de Edição de Registos (Grid1)	Tempo de Resposta (ms)
1	125	**1**	69	**1**	54
2	159	**2**	45	**2**	58
3	132	**3**	68	**3**	51
4	131	**4**	61	**4**	66
5	120	**5**	59	**5**	57
6	127	**6**	55	**6**	61

Teste de Consulta de Registos (Grid1 + GridTab1)	Tempo de Resposta (ms)	Teste de Inserção de Registos (Grid1)	Tempo de Resposta (ms)	Teste de Edição de Registos (Grid1)	Tempo de Resposta (ms)
7	134	**7**	63	**7**	212
8	132	**8**	47	**8**	62
9	141	**9**	67	**9**	61
10	119	**10**	52	**10**	75
11	136	**11**	68	**11**	61
12	138	**12**	60	**12**	65
13	148	**13**	63	**13**	63
14	137	**14**	47	**14**	74
15	141	**15**	57	**15**	64

Tabela 12 - Testes de consulta, inserção e edição de registos no protótipo 2

Teste de Consulta de Registos (ID + ByID)	Tempo de Resposta (ms)	Teste de Inserção de Registos	Tempo de Resposta (ms)	Teste de Edição de Registos	Tempo de Resposta (ms)
1	255	**1**	220	**1**	248
2	234	**2**	211	**2**	209
3	236	**3**	207	**3**	227
4	251	**4**	208	**4**	257
5	595	**5**	214	**5**	241
6	280	**6**	216	**6**	248
7	237	**7**	205	**7**	224
8	251	**8**	212	**8**	288
9	278	**9**	207	**9**	207
10	244	**10**	211	**10**	235
11	243	**11**	219	**11**	233
12	245	**12**	203	**12**	249
13	236	**13**	218	**13**	237
14	238	**14**	205	**14**	221
15	249	**15**	212	**15**	206

Tabela 13 - 10 testes de consulta, inserção e edição de registos no protótipo 2

Teste de Consulta de Registos (ID + ByID)	Tempo de Resposta (ms)	Teste de Inserção de Registos	Tempo de Resposta (ms)	Teste de Edição de Registos	Tempo de Resposta (ms)
1	550	**1**	315	**1**	310
2	322	**2**	332	**2**	312
3	322	**3**	329	**3**	314
4	478	**4**	319	**4**	316
5	325	**5**	324	**5**	314

Teste de Consulta de Registos (ID + ByID)	Tempo de Resposta (ms)	Teste de Inserção de Registos	Tempo de Resposta (ms)	Teste de Edição de Registos	Tempo de Resposta (ms)
6	322	**6**	309	**6**	326
7	323	**7**	308	**7**	309
8	317	**8**	313	**8**	322
9	321	**9**	328	**9**	325
10	331	**10**	316	**10**	318
11	321	**11**	305	**11**	304
12	328	**12**	307	**12**	314
13	323	**13**	617	**13**	317
14	313	**14**	317	**14**	311
15	324	**15**	307	**15**	307

3.4 Análise de Resultados

No presente tópico será apresenta a análise de resultados através da comparação de funcionalidades entre os 2 protótipos, bem como, o esforço do ponto de vista da programação para elaboração dos mesmos.

Na Figura 48 está apresentado um gráfico dos testes efetuados ao método de consulta do protótipo 1 e 2. Estes métodos executam *queries* à base de dados para retorno de dados e estas *queries* foram executadas 1 e 10 vezes seguidas, para simular testes de carga em cada protótipo. Cada vez que um teste era efetuado nos protótipos, eram seguidos os passos referidos na secção 3.3, ou seja, adicionar código ao início e ao fim de cada método. Quando o método finalizava, eram apontados os tempos de resposta retornados pelo mesmo (em ms). Foram feitos 15 testes a cada protótipo, totalizando 60 testes.

Após a conclusão dos testes e análise do gráfico, pode-se verificar que o primeiro protótipo ganha vantagem em relação ao segundo, pois os seus tempos de execução (desde que o utilizador solicita a consulta de dados até obter dados na camada *front-end*) são mais reduzidos. É possível verificar também que quando se efetuaram os testes de carga (execução de 10 *queries*), a resposta do protótipo 1 foi melhor que no protótipo 2.

Na Figura 49 está apresentado um gráfico dos testes efetuados ao método de inserção do protótipo 1 e 2. Estes métodos executam *queries* à base de dados para inserção de dados e estas *queries* foram executadas 1 e 10 vezes seguidas, para simular testes de carga em cada protótipo. Cada vez que um teste era efetuado, eram apontados os tempos de resposta (em ms). Foram feitos 15 testes a cada protótipo, totalizando 60 testes.

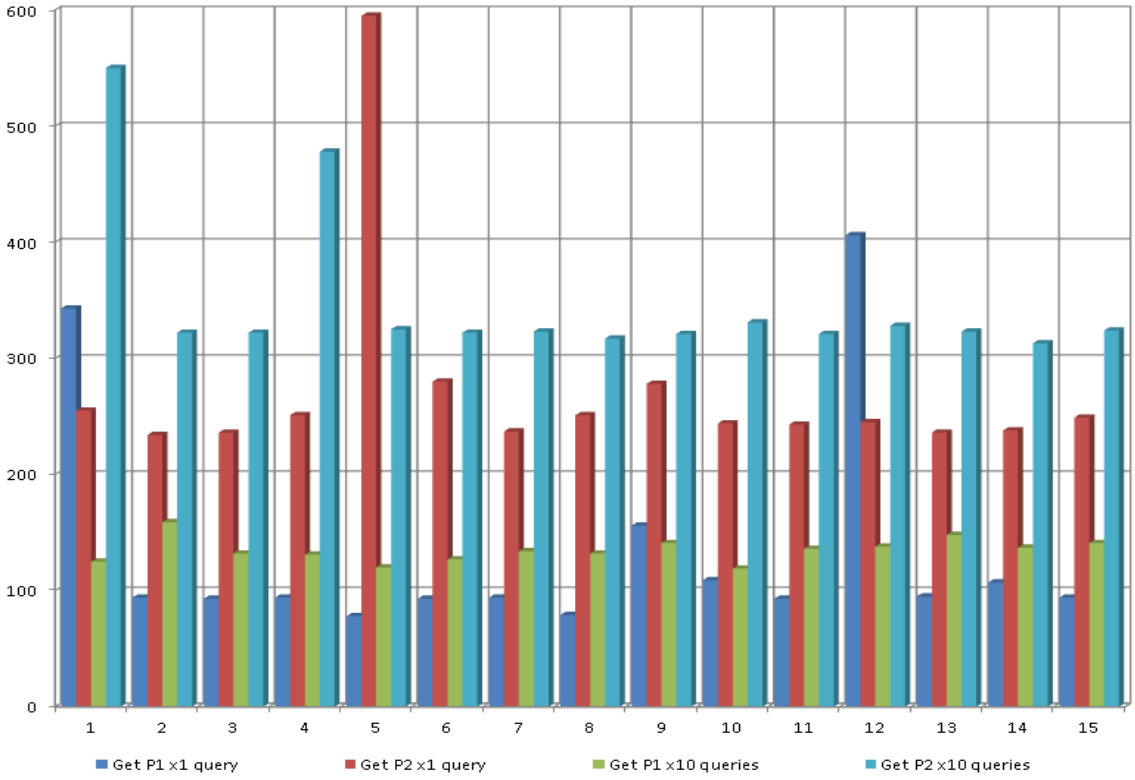

Figura 48 - Testes efetuados ao método de consulta dos 2 protótipos

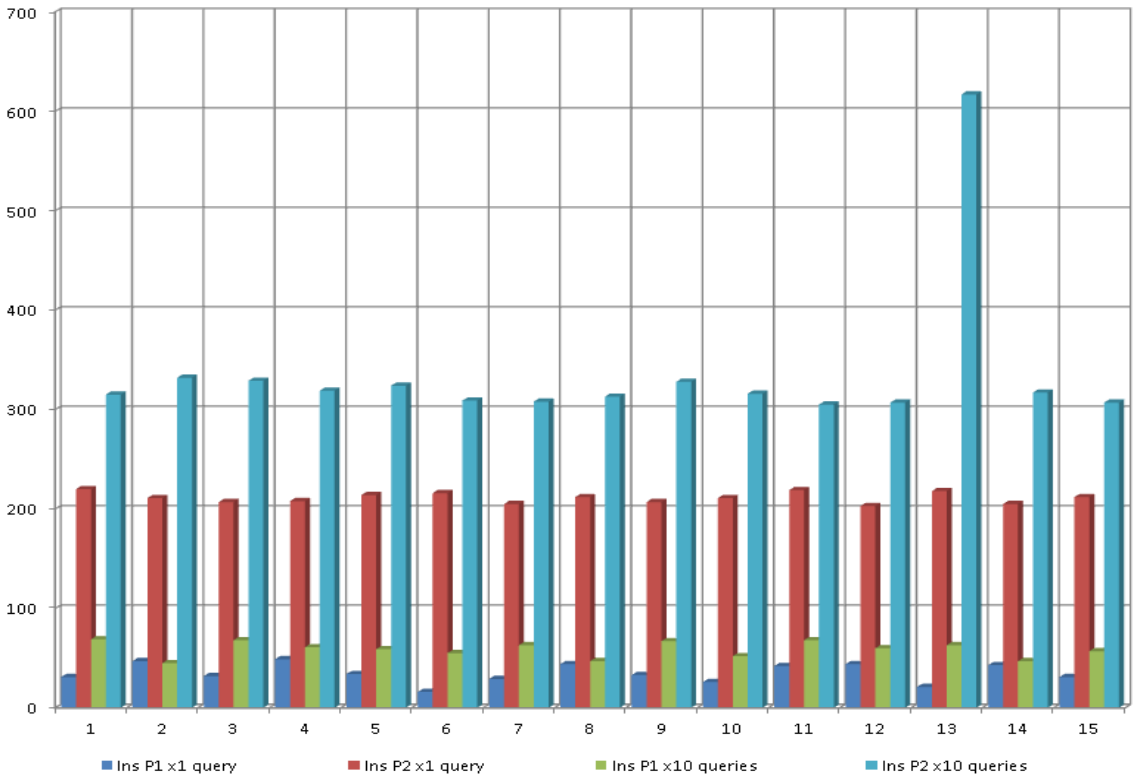

Figura 49 - Testes efetuados ao método de inserção dos 2 protótipos

Após a conclusão dos testes e análise do gráfico, pode-se verificar que o primeiro protótipo ganha vantagem em relação ao segundo, pois os seus tempos de execução (desde que o utilizador solicita a inserção de dados até conclusão do método) são mais reduzidos. É possível verificar também que quando se efetuaram os testes de carga (execução de 10 *queries*), a resposta do protótipo 1 foi melhor que no protótipo 2.

Na Figura 50 está apresentado um gráfico dos testes efetuados ao método de atualização do protótipo 1 e 2. Estes métodos executam *queries* à base de dados para atualização de dados e estas *queries* foram executadas 1 e 10 vezes seguidas, para simular testes de carga em cada protótipo. Cada vez que um teste era efetuado, eram apontados os tempos de resposta (em ms). Foram feitos 15 testes a cada protótipo, totalizando 60 testes.

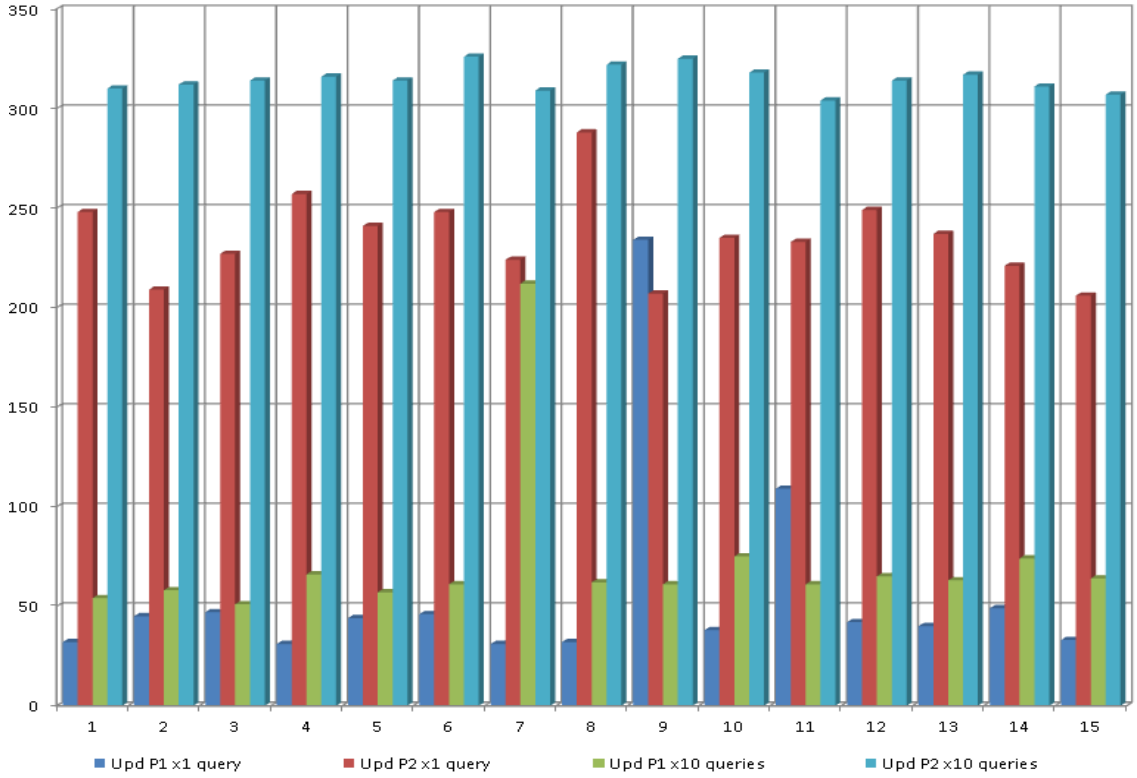

Figura 50 - Testes efetuados ao método de atualização dos 2 protótipos

Após a conclusão dos testes e análise do gráfico, pode-se verificar que o primeiro protótipo ganha vantagem em relação ao segundo, pois os seus tempos de execução (desde que o utilizador solicita a atualização de dados até conclusão do método) são mais reduzidos. É possível verificar também que quando se efetuaram os testes de carga (execução de 10 *queries*), a resposta do protótipo 1 foi melhor que no protótipo 2.

É apresentado na Tabela 14 uma tabela com os tempos médios e respetivos desvios padrão das diversas experiências realizadas. Os valores encontram-se em ms. Após efetuar os 15 testes a cada protótipo foram efetuados os cálculos para o tempo médio da operação e para o respetivo desvio-padrão.

Tabela 14 - Tempos médios e desvios-padrão das experiências

	1 *query*		10 *queries*	
	Média	**Desvio-Padrão**	**Média**	**Desvio-Padrão**
Protótipo 1				
Get	135	100	135	10
Insert	35	9,76	59	8,1
Update	57	53	72	39
Get	271	91	348	69
Insert	211	5,39	336	78
Update	235	22	315	6,3

É apresentada na Tabela 15 uma comparação entre os 2 protótipos, em termos de código. A nível de classes, o protótipo 1 tem bastantes mais classes, pois também possui o triplo das linguagens de programação, o que implica um aumento substancial de classes necessárias.

As linguagens de programação utilizadas no protótipo 1 foram:

- JavaScript
- XML Base 64
- Java
- C++
- SQL

Para o protótipo 2 foram utilizadas:

- Java
- SQL

A nível de conexão à base de dados, foi utilizado no protótipo 1 uma ligação direta (*Command Prompt*) e no protótipo 2 por JDBC. Por fim, foi utilizado o mesmo padrão de *software* para desenvolver os protótipos, ou seja, o MVC (apesar que no protótipo 2, o Modelo foi substituído por métodos de acesso - *getters* e *setters* diretamente nos componentes do protótipo, nomeadamente *combobox* e caixas de texto).

Tabela 15 - Comparativo entre os 2 protótipos

	Protótipo 1	**Protótipo 2**
Número de classes	22	3
Número de linguagens de programação	6	2
Conexão à base de dados (Oracle)	Direta/ Command Prompt	JDBC
Padrões de *software*	1	1
Linhas de código (estimadas)	1500	500

Após esta análise, pode-se verificar que é mais trabalhoso implementar o protótipo 1, devido às diversas linguagens de programação que é necessário dominar, bem como o número de classes necessárias.

O tempo de processamento, no protótipo 1, poderá ser superior nas primeiras vezes que se executa a aplicação e operações da mesma. Isto ocorre devido à memória *cache* estar a ser construída, no entanto, os testes foram feitos já a assumir que a memória *cache* estaria completamente construída. É possível verificar esta situação através da estabilidade dos tempos de resposta nas mesmas operações, por exemplo, executar 1 e 10 vezes a *query* para consultar/inserir/editar resultados.

A nível de linhas de código, o protótipo 1 tem (aproximadamente) o triplo das linhas de código que o protótipo 2, pois possui configurações de cliente e servidor, enquanto que o protótipo 2 só tem uma *connection string* para fazer o acesso e operações à base de dados.

4 Conclusões

Após o desenvolvimento do projeto conducente a esta dissertação e do estudo realizado na área do desenvolvimento aplicacional de diferentes arquiteturas e linguagens de programação, ora agregadas ora isoladas, surgem várias conclusões, algumas de caracter subjetivo, sendo que podem ter interpretações diferentes.

A área da programação *web* é uma área que está em constante evolução e como tal, a escolha da arquitetura a utilizar é fundamental para que a aplicação cumpra os requisitos que o cliente exige. Por exemplo, ao optar por uma arquitetura de *3-tier*, pode ter um custo elevado quanto às máquinas necessárias para a sua operação, no entanto pode ser considerada simples de desenvolver e manter, eficiente, reutilizável e altamente escalável. Neste tipo de arquitetura, cuja aplicação está dividida em camadas, os testes devem seguir uma estrutura que garanta o bom funcionamento entre todas as camadas. Desta maneira, a fase de testes deve ser estruturada em proporção.

No caso das arquiteturas monolíticas, a sua falta de modularidade é notória, o que faz com que esta não seja desejável em aplicações de alto nível e/ou de maior dimensão. Isto deve-se ao facto de aplicações com esta arquitetura terem potenciais erros aquando do melhoramento de funcionalidades já implementadas por poderem existir várias implementações ao longo do código com abordagens descontinuadas não detetáveis na compilação e pela falta de módulos reutilizáveis. O desenvolvimento de testes para este tipo de arquitetura deve, portanto, focar-se no desenvolvimento de casos de uso que cubram todas as alternativas de todas as funcionalidades (normalmente, dando origem a um enorme conjunto de casos de teste). Por outro lado, este tipo de arquitetura pode ser considerada adequada para aplicações de menor dimensão ou de baixo nível. Neste cenário, o número de testes, embora ainda exponencial, poderá trazer vantagens quando comparado com o número de testes numa aplicação da mesma dimensão, desenvolvida com uma arquitetura modular.

A nível de tempos de resposta, foi notória a melhoria que o protótipo 1 apresenta em relação ao protótipo 2, pois para operações idênticas, ganha clara vantagem apesar de estar repartido em várias linguagens de programação e tecnologias. Em casos de realidade empresarial, o melhor protótipo a colocar em produção seria o protótipo 1. A única forma de se colocar o protótipo 2 em produção seria por contenção de custos (pois o protótipo 1 será mais dispendioso que o protótipo 2) ou quando o número de utilizadores não é muito grande (abaixo de 30 utilizadores, pois como foi apresentado na secção da análise de resultados, em 10 pedidos seguidos, os tempos de resposta aumentaram mais que no protótipo 1).

Complementarmente, verificou-se que usar C++ em vez de JDBC é mais vantajoso, pois os tempos de resposta, a efetuar a mesma operação (com o mesmo volume de dados), são mais apelativos no protótipo 1 que no protótipo 2.

A nível de tamanho de código é esmagadoramente maior no protótipo 1 que no protótipo 2, o que implica muito mais tempo para programar o protótipo, bem como (no caso desta dissertação) dominar várias linguagens de programação.

No caso de se fazer *debug* e/ou testes nos protótipos, são consumidos mais recursos computacionais no primeiro protótipo que no segundo, pois para o primeiro caso é necessário ter um servidor aplicacional (para a parte *web*), um *browser* (para os testes em *front-end*) e um JDK. No segundo caso, só é necessário um JDK. Este consumo maior de recursos resulta em tempos de espera que poderão influenciar ou traduzir-se no tempo de desenvolvimento dos protótipos, pois para se fazer os testes, vai-se demorar mais no protótipo 1 que no 2.

Em síntese, o presente trabalho foi uma ótima experiência para desenvolver novas capacidades de análise e espirito crítico acerca de tecnologias e linguagens presentes no mercado, bem como a nível de desenvolvimento aplicacional.

5 Bibliografia

[1] “Linguagem de programação web - conceitos,linguagem e propriedades.,” [Online]. Available: http://www.ebah.pt/content/ABAAAfdpYAG/linguagem-programacao-web#. [Acedido em 07 05 2015].

[2] “Programação orientada a objetos,” [Online]. Available: http://pt.slideshare.net/cleytonferrari/programao-orientada-a-objetos-25598751. [Acedido em 30 04 2015].

[3] “Sistema de Gestão de Base de Dados (SGBD),” [Online]. Available: https://ricardo2aoc.wordpress.com/sistema-de-gestao-de-base-de-dados-sgbd/. [Acedido em 30 05 2015].

[4] “Servidores de Aplicação Web por César Rayan no Prezi,” [Online]. Available: https://prezi.com/jyg0voeacblv/servidores-de-aplicacao-web/. [Acedido em 31 05 2015].

[5] “Indicadores Desempenho e Métricas em TI v29,” [Online]. Available: http://pt.slideshare.net/RIDLO/INDICADORES-DESEMPENHO-E-MTRICAS-EM-TI-V29. [Acedido em 06 05 2015].

[6] “PHP: História do PHP - Manual,” [Online]. Available: https://php.net/manual/pt_BR/history.php.php. [Acedido em 23 11 2014].

[7] “PHP - Linguagem de Programação - InfoEscola,” [Online]. Available: http://www.infoescola.com/informatica/php/. [Acedido em 03 05 2015].

[8] “Vantagens e desvantagens do PHP | Inforlogia,” [Online]. Available: http://www.inforlogia.com/vantagens-e-desvantagens-do-php/. [Acedido em 25 11 2014].

[9] “Javascript - Informática - InfoEscola,” [Online]. Available: http://www.infoescola.com/informatica/javascript-2/. [Acedido em 05 05 2015].

[10] “Vantagens do JavaScript | Ponto do Conhecimento,” [Online]. Available: http://pontodoconhecimento.blogspot.pt/2013/06/vantagens-do-javascript.html. [Acedido em 25 11 2014].

[11] “Projetos web | Implementacao e lancamento | Tecnologias editoriais | JavaScript: Desvantagens,” [Online]. Available:

http://www.avellareduarte.com.br/projeto/producao/producao2/producao22+JavaScript+desvantagens.htm. [Acedido em 25 11 2014].

[12] "ASP.NET - Introdução ao ASP.NET - artigos TechNet - Brasil (Português) - TechNet Wiki," [Online]. Available: http://social.technet.microsoft.com/wiki/pt-br/contents/articles/15804.asp-net-introducao-ao-asp-net.aspx. [Acedido em 24 11 2014].

[13] [Online]. Available: www.inf.ufsc.br/~frank/INE5612/Seminario2012.2/ASP.pptx. [Acedido em 26 11 2014].

[14] "História do Java - Linguagem de Programação - InfoEscola," [Online]. Available: http://www.infoescola.com/informatica/historia-do-java/. [Acedido em 29 04 2015].

[15] "Estudo comparativo entre tecnologias Java: Applet e JWS.," [Online]. Available: http://www.periodicos.ulbra.br/index.php/urissane/article/download/1076/826. [Acedido em 30 04 2015].

[16] P. Coelho, Programação em Java - Curso Completo, FCA.

[17] "::: LINK-SI ::: Desenvolva-se. Integre-se. Ligue-se.: Vantagens e Desvantagens JAVA," [Online]. Available: http://link-si.blogspot.pt/2009/07/vantagens-e-desvantagens-java.html. [Acedido em 26 11 2014].

[18] "C++ - Linguagem de Programação - InfoEscola," [Online]. Available: http://www.infoescola.com/informatica/cpp/. [Acedido em 31 03 2015].

[19] "C# - C Sharp - Linguagem de Programação - InfoEscola," [Online]. Available: http://www.infoescola.com/informatica/c-sharp/. [Acedido em 01 05 2015].

[20] "Linguagem C#," [Online]. Available: http://www.geocities.ws/familiacpp/page3.html. [Acedido em 26 11 2014].

[21] "Parceiros - IOS," [Online]. Available: http://www.ios.com.br/web/ios/parceiros. [Acedido em 15 04 2015].

[22] "Oracle EXPLICAÇÕES EM PDF," [Online]. Available: http://pt.slideshare.net/sergeduardo/oracle-explicaes-em-pdf. [Acedido em 26 11 2014].

[23] "advantages and disadvantages of PostgreSQL?," [Online]. Available: http://forums.devshed.com/postgresql-help-21/advantages-disadvantages-postgresql-106805.html. [Acedido em 28 11 2014].

[24] "Introdução | Manifesto Tecnológico," [Online]. Available: https://fabadas.wordpress.com/dba-banco-de-dados/sql-server/introducao/. [Acedido

em 30 04 2015].

[25] “Vantagens e desvantagens do Microsoft SQL,” [Online]. Available: http://www.ehow.com.br/vantagens-desvantagens-microsoft-sql-lista_224636/. [Acedido em 28 11 2014].

[26] “março | 2010 | Fernando Mantoan,” [Online]. Available: http://fernandomantoan.com/2010/03/. [Acedido em 11 05 2015].

[27] “Introdução a jQuery,” [Online]. Available: http://www.criarweb.com/artigos/introducao-jquery.html. [Acedido em 15 03 2015].

[28] “jQuery,” [Online]. Available: http://pt.slideshare.net/darkdevilbr/jquery-8011096. [Acedido em 11 03 2015].

[29] “Dojo Toolkit por Allan Griebeler no Prezi,” [Online]. Available: https://prezi.com/sze5_iudubms/dojo-toolkit/#. [Acedido em 12 03 2015].

[30] “Dojo tutorial,” [Online]. Available: http://pt.slideshare.net/girishsrivastava1/dojo-tutorial. [Acedido em 20 03 2015].

[31] “Link - Gerimos Conhecimento Consigo - Internet of Things,” [Online]. Available: http://www.linkconsulting.com/upl/%7B04a6ca91-62cc-484e-8fcf-d029ec64adf4%7D.pdf. [Acedido em 11 05 2015].

[32] [Online]. Available: bibdig.poliseducacional.com.br/document/?down=184. [Acedido em 20 02 2015].

[33] “What is Quality Function Deployment (QFD)and How to Apply the Voice of the Customer,” [Online]. Available: http://qfdpro.com/qfd_explained.html. [Acedido em 28 06 2015].

[34] “Index of /~req_case/Seminarios/Prioridades,” [Online]. Available: http://www.di.ufpe.br/~req_case/Seminarios/Prioridades/qfd.pdf. [Acedido em 28 06 2015].

[35] “Blekinge Tekniska Högskola - in real life,” [Online]. Available: http://www.bth.se/com/besq.nsf/(WebFiles)/5A52350A52726F51C12570A8004CB613/$FILE/Software_quality_attributes.pdf. [Acedido em 28 06 2015].

[36] “Unit 4 : Quality Models in Software Engineering | msritse2012,” [Online]. Available: https://msritse2012.files.wordpress.com/2013/01/mccalls-11-quality-factor-hierarchy.jpg. [Acedido em 28 06 2015].

[37] [Online]. Available: http://projects.spring.io/spring-framework/. [Acedido em 21 06 2015].