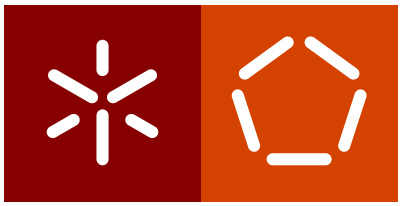
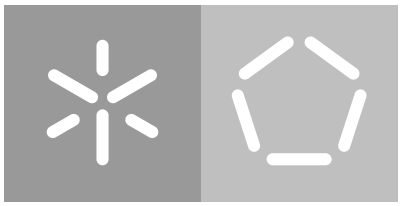


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Frederico Mendes

Geração de aplicações multi-plataforma a partir de modelos

November 2017



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Frederico Mendes

Geração de aplicações multi-plataforma a partir de modelos

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho realizado sob orientação de
António Nestor Ribeiro

November 2017

Anexo 3

DECLARAÇÃO

Nome

Frederico Jorge Falcão Torres de Castro Mendes

Endereço electrónico: frederico.mendes10@gmail.com Telefone: 915081198 / _____

Número do Bilhete de Identidade: 14409312

Título dissertação □/tese □

Geração de aplicações multi-plataforma a partir de modelos

Orientador(es):

António Nestor Ribeiro

Ano de conclusão: 2017

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado Integrado em Engenharia Informática

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, 24 / 10 / 2017

Assinatura: Frederico Mendes

AGRADECIMENTOS

Ao professor António Nestor Ribeiro pela orientação na dissertação, por toda a disponibilidade e por toda a ajuda dada.

Aos meus pais e irmã por me proporcionarem a possibilidade de concluir este grau académico e por todo o apoio e paciência dados.

Aos meus amigos e família que sempre me incentivaram a terminar a dissertação das mais diversas formas.

ABSTRACT

In software engineering, modeling systems using diagrams, allows a system to be represented in a standardized way, in order to ease the understanding of its logical structure, specification and documentation.

Nowadays at the business world, the use of diagrams using the proper tools, aims communication between teams and is introduced at an early stage of the project as modeling it. However, the construction of applications using the low code techniques or even zero code is an increasing reality.

The natural evolution of this concept will result in the automatic code generation through a visual language, such as diagrams, thus facilitating the code's production, and in the meantime, time can be saved by having work done at an earlier stage of the project. Having that said, the use of models, more or less standard as a way to specify and prototyping applications, are and will be a well established reality and with considerable success, also allowing to manage more effectively multiplatform's issues, since code generation is not exclusive to any paradigm or specific programming language.

With this dissertation is intended to use Unified Modeling Language (UML) models as the only mechanism to specify applications, automating code's construction process and technological aspects of its deployment and installation, providing a tool that enables the process of creating Web applications and android from UML models.

Therefore, an application was created, through user interaction, which receives class diagrams exported in eXtensible Markup Language (XML) format that are interpreted and thereby generates *web* and android *applications*. These applications take under Create Read Update and Delete (CRUD) operations for each entity represented on the class diagram.

RESUMO

Na área de Engenharia de *Software*, a modelação de sistemas com recurso a diagramas, permite representar um sistema de forma padronizada, com o intuito de facilitar a compreensão da especificação, estrutura lógica, e documentação dos mesmos.

Hoje em dia, no mundo empresarial, a utilização de diagramas através de ferramentas próprias para o efeito tem como objetivo a comunicação entre equipas, inserindo-se na fase de modelação dos projetos. No entanto, a construção de aplicações com recurso a técnicas de *low code*, ou mesmo *zero code*, é uma realidade cada vez mais atual.

A evolução natural deste conceito resultará na geração automática de código através de uma linguagem visual, como os diagramas, facilitando, assim, a produção de código, e ao mesmo tempo, conseguir-se-á uma poupança de tempo aproveitando o trabalho realizado numa fase mais precoce do projeto. Posto isto, a utilização de modelos, mais ou menos *standard*, como forma de especificar e prototipar aplicações é e será, cada vez mais, uma realidade bem fundada e com sucesso assinalável, permitindo também gerir de forma mais eficaz questões de multi-plataforma, visto que a geração de código não é exclusiva a nenhum paradigma nem linguagem de programação específica.

Com esta dissertação pretende-se, então, utilizar modelos UML como mecanismo único de especificação de aplicações, automatizando o processo de construção do respetivo código e os aspetos tecnológicos relativos ao seu deployment e instalação, disponibilizando uma ferramenta que possibilite o processo de criação de aplicações *web* e *android* a partir de diagramas UML.

Assim, foi criada uma aplicação que, através da interação do utilizador, recebe diagramas de classe exportados em formato XML interpretando-os e gerando aplicações *android* e aplicações *web*. Estas aplicações realizam as operações CRUD para cada entidade representada no diagrama de classe.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE ACRÓNIMOS

API	Application programming interface.
CIM	Computation Independent Model.
CRUD	Create Read Update and Delete.
DSL	Domain Specific Languages.
EMF	Eclipse Modeling Framework.
HTML	HyperText Markup Language.
HTTP	Hypertext Transfer Protocol.
IDE	Integrated development environment.
JET	Java Emitter Templates.
JSF	JavaServer Faces.
JSON	JavaScript Object Notation.
JSP	JavaServer Pages.
JVM	Java Virtual Machine.
M2C	Model-to-Code.
M2T	Model-to-Text.
MDA	Model-Driven Architecture.
MDE	Model Driven Engineering.
MEI	Mestrado em Engenharia Informática.
MOF	MetaObject Facility.
MOFM2T	MOF Model To Text Transformation Language.
MTE	Model Transformation Environment.
MVC	Model View Controller.
NPM	Node Package Manager.

OAW	OpenArchitectureWare.
OMG	Object Management Group.
PHP	Hypertext Preprocessor.
PIM	Platform Independent Model.
PSM	Platform Specific Model.
REST	Representational state transfer.
SBT	Simple Build Tool.
SDK	Android Software Development Kit.
SE	Java Standard Edition.
SPA	Single-Page Application.
SQL	Structured Query Language.
SWOT	strengths weaknesses opportunities and threats.
UI	User Interface.
UJECTOR	UML to Java Executable Code generaTOR.
UM	Universidade do Minho.
UML	Unified Modeling Language.
URL	Uniform Resource Locator.
UWE	UML-based Web Engineering.
WebML	Web Modeling Language.
XMI	XML Metadata Interchange.
XML	eXtensible Markup Language.

1

INTRODUÇÃO

1.1 ENQUADRAMENTO

Estão a surgir cada vez mais soluções relativamente à construção de aplicações que impliquem menos escrita de código. Este crescimento é sustentado com o aparecimento das linguagens de modelação.

As linguagens de modelação são linguagens que nos permitem expressar informação, conhecimento ou sistemas de uma estrutura que é definida por um conjunto consistente de regras. Esta representação pode ser feita, por exemplo, usando UML, que é uma linguagem de modelação que nos permite fazer a representação anteriormente abordada, através de diagramas. Para além disto, esta linguagem consegue gerar código a partir das representações elaboradas na mesma, isto porque são linguagens que têm uma sintaxe e uma semântica bem definidas levando assim ao objetivo previamente referido: a menor escrita de código.

Os diagramas representam um sistema podendo dar origem a código, sendo exportados num formato denominado de XML Metadata Interchange (XMI), que consiste num padrão da Object Management Group (OMG) para troca de informações baseado em XML, cujo principal objetivo é facilitar a troca de meta dados entre as ferramentas de modelação e os programadores [Beno4].

Com o intuito de alcançar o objetivo baseado nos conceitos de *low code*, o uso de um Integrated development environment (IDE) pode ser importante, pois estas aplicações, para além de facilitarem o processo de desenvolvimento de *software*, possuem como vantagens a incorporação na sua estrutura, normalmente, de um *debugger*, um compilador e um interpretador. Estas aplicações de *software* conseguem, também, facultar a possibilidade de integração de *plugins* para acrescentar funcionalidades a si mesmas, ou permitir ao programador que construa o seu próprio *plugin* ou programa. Com isto, um programador que utilize um IDE, como por exemplo, o *Eclipse*, consegue acrescentar um *plugin* ao mesmo, ou construir uma ferramenta que, facilmente processe o XMI ou XML gerado pelos diagramas representados no UML e os converta em alguma linguagem de programação, como por exemplo, *Java*, facilitando bastante a atividade de escrita de código.

1.2 MOTIVAÇÃO

A facilidade e a rapidez com que se constroem aplicações robustas são qualidades apreciadas por qualquer engenheiro de *software*, ou qualquer empresa. Quanto menos tempo se demorar a construir algo suficientemente bom para ser lançado no mercado e consequentemente obter lucros do mesmo, maior benefícios promoverá, quer para os consumidores finais que terão o produto desejado com maior celeridade do que o habitual, quer para a entidade que produziu o mesmo, visto que obtém rendimentos mais prematuramente.

Para além disto, se a esta celeridade no processo de desenvolvimento de *software* se conseguir alicerçar componentes de desenvolvimento de um produto sem que a qualidade do mesmo seja comprometida, fornecer-se-á ainda mais confiança ao programador para adotar esta metodologia. Procura-se, então, a solução que torne ainda mais próximas duas etapas inerentes ao desenvolvimento de produtos de *software*, nomeadamente as fases de *design* e de implementação, usando modelos para especificar problemas e, a partir dos mesmos, gerar aplicações multi-plataforma, tentando poupar no tempo que as fases anteriormente mencionadas exigem, contribuindo assim, para um tempo de produção menos extenso.

Posto isto, é neste problema que reside a génese desta dissertação, isto é, há necessidade de elaborar uma ferramenta que faça esta aproximação das duas fases de desenvolvimento de *software* anteriormente referidas, levando a que a poupança de tempo e o tempo de lançamento de novos produtos para o mercado sejam encurtados, mantendo a qualidade e robustez das aplicações que se guiam por esta abordagem. Pretende-se, então, que o tema da presente tese consista em validar estas hipóteses para um determinado tipo de aplicações (orientadas à introdução de dados), com uma lógica de negócio pouco complexa e que partilham um estilo de navegação semelhante.

1.3 OBJETIVOS

Os objetivos deste trabalho são:

1. Efetuar o levantamento e estudo das várias ferramentas e *plugins* que existem para geração de código.
2. Estudar os formatos XMI e XML a ser exportado pelos diagramas.
3. Exportar diagramas de classe, sequência e estado em formatos XMI e XML.
4. Construir mecanismos de transformação de diagramas em código, para um determinado tipo de aplicações.
5. Definir um determinado tipo de famílias de aplicações para o qual o trabalho vai incidir.

6. Elaborar uma ferramenta ou um *plugin* que através dos XMI ou XML tratado, crie aplicações *web* e *android*.

Em suma, pretende-se com este projeto elaborar uma ferramenta que consiga através do tratamento de diagramas UML, gerar aplicações *android* e *web*.

1.4 ESTRUTURA DO DOCUMENTO

O presente documento encontra-se dividido em 9 capítulos, pelo que o primeiro é a introdução 1.

No capítulo 2 serão revelados alguns conceitos considerados importantes, que estão relacionados com o âmbito da dissertação. De seguida, o estado da arte 3, onde irão ser explorados todos os trabalhos realizados alusivos ao tema desta dissertação.

O capítulo 4 irá relatar a abordagem que será seguida para que o objetivo deste trabalho seja atingido. Posteriormente, nas alterações modelo para texto 5, será apresentada a solução encontrada para as transformações dos diagramas em código, bem como a estrutura que os diagramas de classe devem seguir.

Tanto no sexto 6, como no sétimo capítulo 7, far-se-á uma análise ao meio encontrado para que a geração de aplicações *android* e *web*, respetivamente, seja possível começando-se por fazer uma contextualização das ferramentas utilizadas, apresentando-se, posteriormente, os requisitos necessários para a sua integração, finalizando com a apresentação de imagens das aplicações geradas. A aplicação gerador de aplicações CRUD será apresentada no capítulo 8, fazendo-se uma contextualização da mesma, assim como uma análise à forma como esta foi desenvolvida e estruturada, acabando-se o capítulo da mesma forma que finalizarão os dois que lhe antecedem, ou seja, exibindo-se imagens relativas à aplicação.

Por fim, no último capítulo 9, serão apresentadas as conclusões relativas ao trabalho realizado e as novas abordagens que possam ser adotadas, relacionadas com o tema inerente a este trabalho.

2

CONCEITOS INTRODUTÓRIOS

Com o objetivo de uma maior compreensão do domínio do problema abordado nesta dissertação, foi criado este tópico para proporcionar um conhecimento aprofundado em relação a conceitos essenciais que se encontram associados ao âmbito da mesma.

2.1 MODELO

O modelo de um sistema é a descrição que especifica o próprio sistema e o seu ambiente para um certo objetivo. Um modelo é apresentado como uma combinação de textos e desenhos. O texto pode estar em linguagem de modelação ou em linguagem natural [BCD+03].

2.2 DIAGRAMAS QUE INTEGRAM A LINGUAGEM UML

O UML especifica 13 tipos diferentes de diagramas. Estes diagramas estão organizados em 3 categorias:

- Diagramas Estruturais: *Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram and Deployment Diagram.*
- Diagramas de Comportamento: *Use Case Diagram, Activity Diagram and State Machine Diagram.*
- Diagramas de interação: *Sequence Diagram, Communication Diagram, Timing Diagram and Interaction Overview Diagram.*

De entre estes diagramas, irão ser definidos, nas próximas subsecções, os que serão mais utilizados/importantes ao longo desta dissertação.

DIAGRAMA DE CLASSE Um diagrama de classe demonstra um conjunto de classes, interfaces e os seus relacionamentos[JBR+99]. Uma classe é vista como algo que contém objetos que devem ter as mesmas operações, atributos e associações dessa classe, mas com valores de atributo diferentes. Os diagramas de classe são usados em qualquer processo de desenvolvimento que usa UML como uma notação de modelação. Estes são úteis no processo de

desenvolvimento desde o início, onde podem ser usados para identificar os requisitos do sistema e as suas entidades.

DIAGRAMA DE ESTADO Um diagrama de estado exibe uma máquina de estados com ênfase no fluxo de controlo entre estados [S+04].Os mesmos são muito utilizados na modelação do comportamento dinâmico do sistema, visando representar o comportamento de uma única entidade ou objeto dentro do sistema. Estados, eventos e transições são os principais elementos que podem formar qualquer diagrama deste tipo.

DIAGRAMA DE SEQUÊNCIA Os diagramas de sequência são usados para capturar o cenário específico de um sistema, ao mostrar como os objetos envolvidos no mesmo, colaboram através da troca de mensagens para execução de um comportamento concreto. [AP08].

2.3 GERAÇÃO DE CÓDIGO

Geração de código é um nome alternativo para conceitos como as transformações Model-to-Text (M2T) ou Model-to-Code (M2C) onde o *input* é um dado modelo ou uma dada especificação e o *output* é o código gerado a partir das transformações aplicadas. Geração de código não é mais que escrever um programa que escreve outro programa, ou código que escreve outro código, e é conhecida como meta programação [SVC06].

As principais vantagens associadas à geração de código são:

- auxílio a duplicação de código;
- aumento de produtividade, pois diminui o tempo de codificação;
- redução o tempo e esforço de trabalho dos programadores.

2.4 ENGENHARIA ORIENTADA AO MODELO (MDE)

Model Driven Engineering (MDE) é uma metodologia de desenvolvimento de *software* que tem como base o uso de modelos. O principal objetivo da mesma é que os modelos sejam o elo de ligação entre todas as atividades que o desenvolvimento de *software* implica, desde o *design* do sistema, implementação de código e *deployment*. Esta abordagem fornece algumas vantagens como melhorias de qualidade e aumento da produtividade no desenvolvimento de *software*, bem como, melhor comunicação entre especialistas do domínio e programadores [BBG+05].

2.5 MODEL-DRIVEN ARCHITECTURE (MDA)

Model-Driven Architecture (MDA) é uma abordagem ao MDE proposta pelo OMG [G+00] para o desenvolvimento de sistemas de *software*. Fornece um conjunto de diretrizes e princípios para especificar um sistema com base em modelos.

Esta abordagem tem um ciclo de vida característico para o processo de desenvolvimento, que começa com a definição do sistema de uma forma abstrata, e independente da linguagem a que o mesmo se destina, utilizando para isso o Platform Independent Model (PIM). A segunda etapa consiste em refinar o PIM para dar origem ao Platform Specific Model (PSM) que especifica as possíveis plataformas para implementar o sistema de *software*. O passo final consiste em transformar as especificações em código, que irá ser executado na linguagem destino [MM+03]. Na Figura 1 podemos ver uma representação da organização da abordagem proposta.

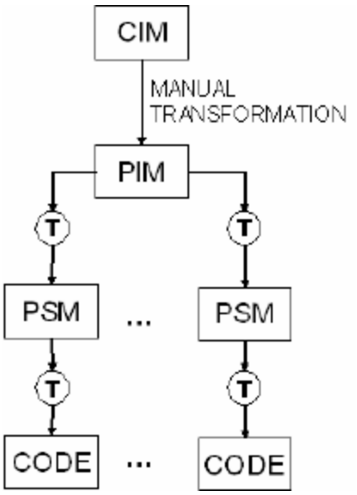


Figura 1: Exemplo da Abordagem Model-Driven Architecture (MDA) [MTSP05].

2.5.1 *Computation Independent Model (CIM)*

O Computation Independent Model (CIM), muitas vezes referido como modelo de domínio, descreve o sistema não considerando detalhes da estrutura do mesmo. O objetivo do CIM é preencher a lacuna entre os especialistas de domínio que especificam os requisitos do sistema e os especialistas em *design* e tecnologia. O CIM não considera como o sistema vai ser implementado.

2.5.2 *Platform Independent Model (PIM)*

O PIM visualiza o sistema de uma perspetiva independente relativamente à plataforma em que o mesmo vai ser inserido, com o objetivo de torná-lo adequado a múltiplas e diferentes plataformas.Posto isto, o PIM é uma abstração de detalhes tecnológicos [SVEH12].

2.5.3 *Platform Specific Model (PSM)*

O PSM implementa e aperfeiçoa o PIM e é alcançado através de técnicas de transformação de modelos. Resulta da combinação das especificações independentes da plataforma com os detalhes da plataforma destino, com o intuito de permitir a geração de código e os ficheiros de configuração que tornam o *software* executável. Um PSM usa os conceitos de uma linguagem específica para descrever um sistema [SVEH12].

2.6 META MODELAÇÃO

É um dos principais problemas no processo de desenvolvimento de *software* orientado ao modelo. A meta-modelação é utilizada nos seguintes problemas [SVEH12]:

- Construção de *Domain Specific Languages (DSL)*: a sintaxe abstrata da linguagem é definida pelo meta-modelo.
- Validação de modelos: validação de modelos contra restrições definidas no meta-modelo.
- Transformações de modelo para modelo: definição de regras de mapeamento entre meta-modelos.
- Geração de código: Os modelos para geração de código referem-se ao meta-modelo da DSL.
- Integração de ferramentas: Um meta-modelo é usado para a adaptação de ferramentas de modelagem para um certo domínio.

Um meta-modelo é usado para a adaptação de ferramentas de modelação para um determinado domínio.

Um meta-modelo define a estrutura dos modelos. Em resumo, define a forma de construir uma linguagem de modelação e as suas relações. Assim, um meta-modelo define a sintaxe abstrata e a semântica estática de uma linguagem de modelação [SVEH12].

O uso de técnicas de meta-modelação durante o tempo de execução fornece uma resposta eficaz para sistemas de alta variabilidade, onde o grande desfasamento semântico entre

elementos de especificação e implementação em sistemas tradicionais podem ser reduzidos pelo uso de modelos, meta modelos e meta dados em geral [RFBLO01].

Os modelos podem ser descritos por qualquer linguagem de modelação, mas o domínio deve ser crucial para a escolha da linguagem. [SVEH12]

A Figura 2 representa as 4 meta-camadas definidas pela OMG, MetaObject Facility (MOF) [SVEH12]:

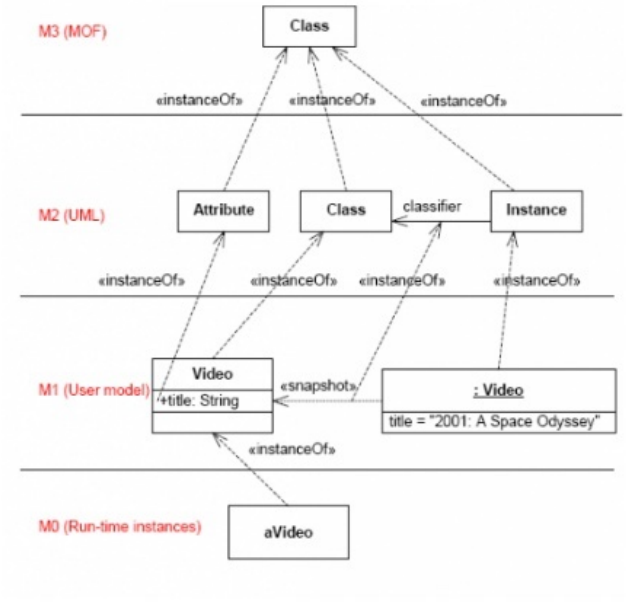


Figura 2: As 4 meta-camadas definidas pela OMG. [Cia].

- M0: A camada M0 é responsável pela construção de instâncias das classes definidas na camada M1. Representa a atribuição de valores aos atributos da classe.
- M1: Nesta camada ocorre a declaração de classes.
- M2: A camada M2 representa o meta-modelo. Serve para a definição de construções que podem ser utilizadas na camada subjacente, M1. Assim, os elementos de M1 são instâncias da camada M2.
- M3: Define as chamadas classes MOF. O MOF é usado para definir linguagens de modelação para a camada M2, como por exemplo o UML.

2.7 TRANSFORMAÇÕES M2T

Uma transformação M2T consiste num processo que recebe um modelo como *input* e em seguida, produz texto como *output*. Uma vez que a MDE pretende gerar um sistema de *software* em execução a partir de modelos, o texto gerado pelas transformações M2T normalmente corresponde ao código fonte.

Assim, a transformação referida também pode ser conhecida como geração de código [CH03].

2.8 ANDROID

Android é uma plataforma *mobile* desenvolvida pela *Google*, que representa uma solução *open-source* com ferramentas de desenvolvimento, fornecendo um grande suporte para diferentes tipos de dispositivos e sistemas operativos. As aplicações podem ser desenvolvidas usando o Android Software Development Kit (SDK). Atualmente a utilização deste sistema operativo atinge uma percentagem de quase 90% [Tra16]. Na figura 3 podemos ver representada a composição deste sistema operativo.

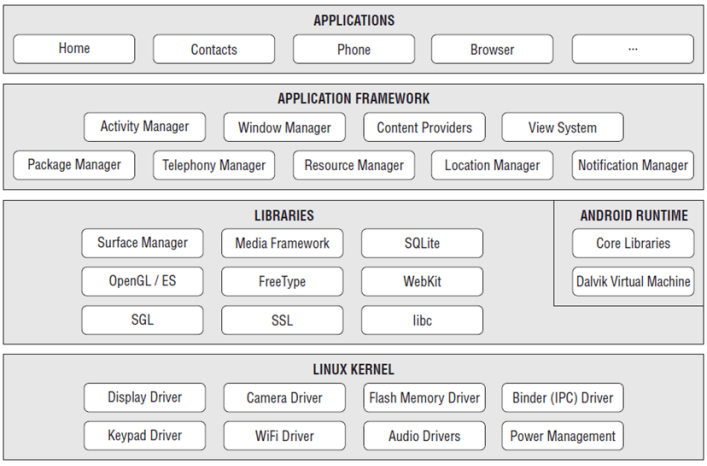


Figura 3: Representação da arquitetura do sistema operativo *android* [Jai].

2.8.1 *Principais Componentes*

Entre os componentes essenciais de uma aplicação *android*, a *activity* e o *service* encapsulam a maior parte do código de comportamento das aplicações. As aplicações *android* devem obedecer a protocolos de ciclo de vida rigorosos. Uma *activity* é controlada por 7 métodos da classe *android.app.Activity*.Estes representam o comportamento que a *activity* irá adotar

nos diferentes estados que pode tomar. A Figura 4 apresenta os 7 métodos de ciclo de vida da mesma. O método *onCreate* é chamado quando a *activity* é criada pela primeira vez. O método *onStart* é chamado quando uma *activity* está na iminência de se tornar visível para o utilizador, enquanto o método *onResume* é chamado quando uma *activity* começa a interagir com o utilizador e o método *onPause* é chamado quando uma *activity* não é visível para o utilizador. O método *onStop* é chamado quando a *activity* não é mais visível para o utilizador.

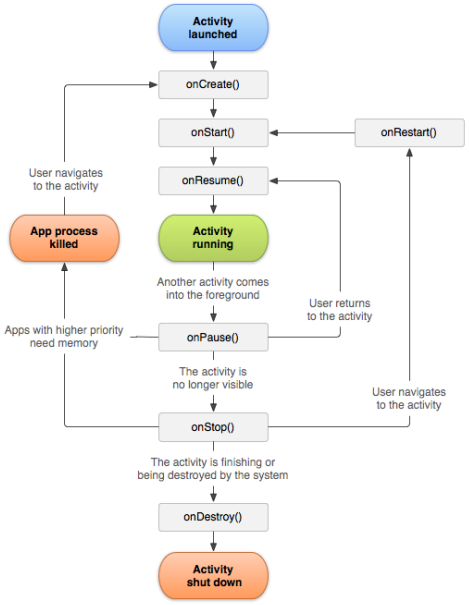


Figura 4: Representação do ciclo de vida de uma *activity* [Lif].

2.8.2 *Android Manifest*

Para que o sistema *android* funcione corretamente, este precisa de obter uma descrição detalhada da aplicação e como ela é composta antes de poder executá-la. O *AndroidManifest.xml*, um ficheiro necessário em todas as aplicações *android*, exibe informações importantes para o sistema *android*. Possui algumas responsabilidades, sendo as mais importantes:

- descrever todos os componentes incluídos na aplicação: todas as *Activities*, *Broadcast Receivers*, *Services* e *Content Providers*;
- declarar as permissões que a aplicação deve ter para interagir com outras aplicações e aceder a partes protegidas da API das mesmas;
- declarar bibliotecas necessárias e um nível mínimo da API do *android* que a aplicação requer.

3

ESTADO DA ARTE

Neste capítulo estão expostos alguns dos trabalhos que partilham o âmbito desta dissertação. São exploradas ferramentas, dissertações ou *plugins* que foram desenvolvidos e que possam ser similares em alguma das fases/objetivos pelos quais este projeto irá passar. Sendo assim, ir-se-á analisar desde ferramentas de modelação, ferramentas de transformações M2T e ferramentas de geração de código, até aos projetos que conseguem gerar aplicações quer *android*, quer *web*.

3.1 FERRAMENTAS DE MODELAÇÃO

A primeira fase desta dissertação, passará pela escolha de uma ferramenta que faculte a possibilidade de desenhar diagramas UML e, posteriormente, permita exportá-los no formato desejado.

3.1.1 *Visual Paradigm*

Visual Paradigm [Par13] é uma ferramenta UML utilizada para o desenvolvimento de aplicações de larga escala, seguindo uma abordagem *agile* e orientada aos objetos. Pode ser integrada noutros ambientes de desenvolvimento, por exemplo, com IDE, como *Eclipse*, *Netbeans*, etc. Esta ferramenta é bastante conhecida e muito utilizada.

3.1.2 *Papyrus*

Papyrus é uma ferramenta que fornece ao IDE *Eclipse* um editor de diagramas UML totalmente robusto, suportando a maior parte dos diagramas que a linguagem de modelação anteriormente referida faculta [LTE+09].

3.1.3 *ArgoUML*

ArgoUML [Arg05] é uma aplicação *open-source* que usa UML para modelar diagramas relativos a *software*. A aplicação corre na maior parte das plataformas uma vez que é implementada em *Java*. Providencia suporte para quase todos os tipos de diagrama da UML padrão e inclui suporte cognitivo.

3.2 TRANSFORMAÇÕES MODELO PARA TEXTO (M2T)

Após estar efetuado o desenho dos diagramas, é necessário haver algo que interprete os mesmos, passando-os de linguagem visual para textual, para que depois possam ser processados e interpretados. Para isto foram analisadas algumas ferramentas.

3.2.1 *Xpand*

Xpand foi inicialmente desenvolvida por OpenArchitectureWare (OAW) como uma linguagem de transformação M2T. Entretanto, tornou-se parte das linguagens de transformação que integram a plataforma *Eclipse*. *Xpand* é uma linguagem de transformação baseada em modelos que controla o processo de geração de código [EFH+08].

3.2.2 *XSLT*

XSLT [C+99] é uma linguagem de transformação declarativa em que se especifica um conjunto de nós XML, chamados *templates*. Cada *template*, irá corresponder a um nodo dos elementos XML do documento dado como *input*. O *output* gerado a partir das transformações, não tem necessariamente de ser um ficheiro XML, sendo possível gerar qualquer formato textual, como HyperText Markup Language (HTML), por exemplo.

3.2.3 *MOF linguagem de transformação de modelo para texto (MOFM2T)*

MOF Model To Text Transformation Language (MOFM2T) [MOF08] é uma abordagem MDA elaborada pela OMG e visa permitir a transformação de modelos gráficos baseados em MOF para texto. É usada para transformações entre PSM e código fonte. Como outras abordagens, MOFM2T é baseada em transformações de modelos segundo *templates*, originando texto como *output*.

3.3 GERAÇÃO DE CÓDIGO

Já existem muitas ferramentas que incorporam em si a capacidade de fazerem transformações M2T e, com isso, gerar código, proporcionando assim aos utilizadores uma maior facilidade no desenvolvimento do mesmo. De salientar que as ferramentas analisadas no capítulo 3.1 têm esta característica referida, porém, no decorrer da investigação inerente à presente dissertação, deparámo-nos com mais algumas que nos parecem interessantes e que acrescentam mais funcionalidades às abordadas anteriormente.

3.3.1 *JCode*

JCode gera automaticamente código *java* através das especificações de diagramas de classe e estado elaboradas em UML. A ferramenta utiliza máquinas de estados de objetos e especificações estruturais que são dadas pelo diagrama de classe do sistema e gera código para toda a aplicação. Gera código quer para os objetos, quer para o seu comportamento e para as suas ações [N+05].

3.3.2 *Rhapsody*

Rhapsody é uma ferramenta que permite a criação de modelos UML para uma aplicação e que pode gerar código *C, C++* ou *Java*. Esta ferramenta gera código a partir de diagramas de classe e de diagramas de estado [HK04].

3.3.3 *Java Emitter Templates (JET)*

Java Emitter Templates (JET) [C+10] é uma ferramenta que gera código fonte através do uso de *templates*. Faz parte do projeto Eclipse Modeling Framework (EMF) [SBP09] e usa conjuntos da sintaxe de JavaServer Pages (JSP) para construir os *templates* para geração de código. Consegue gerar código Structured Query Language (SQL), XML e *Java*.

3.3.4 *UJECTOR - UML to Java Executable Code generaTOR*

Esta ferramenta gera código a partir de diagramas de classe, sequência e de atividade. Os diagramas de classe são usados para gerar o esqueleto do código, os de sequência adicionam comportamento ao esqueleto gerado, enquanto que os de atividade fornecem interação de utilizador ao código e completam-no [UN09]. UML to Java Executable Code generaTOR (UJECTOR) é constituída por 3 componentes:

- XMIParser - Recebe os diagramas no formato XMI como *input* e gera meta-modelos dos diagramas recebidos.
- Code Generator - Recebe como *input* os meta-modelos criados pela primeira componente desta ferramenta, gerando depois, código *Java* isolado para cada diagrama que mais tarde será agregado.
- Code Merger - Trabalha o código isolado gerado pela componente *Code Generator* e fica responsável por gerar código *Java* completo para as classes e interfaces constituintes da aplicação.

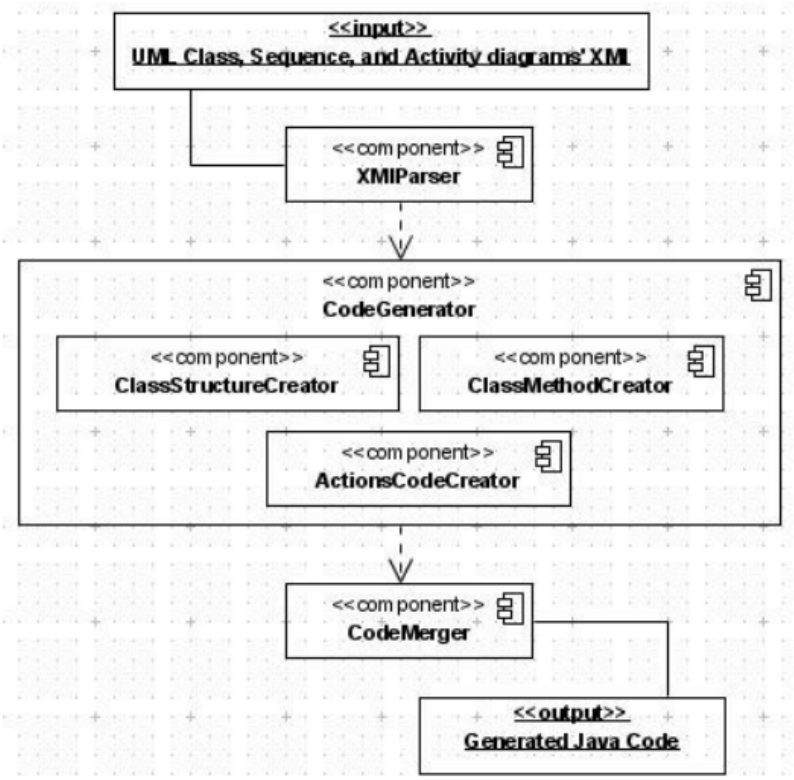


Figura 5: Arquitetura da ferramenta *Ujector* [UNo9].

3.4 GERAÇÃO DE APLICAÇÕES ANDROID

Nesta secção são apresentados alguns dos trabalhos, dissertações, ferramentas ou artigos realizados e que têm como propósito a geração de aplicações *android* robustas e funcionais.

3.4.1 *Desenvolvimento de aplicações móveis usando uma abordagem MDE*

Em [RdS14] é proposto o uso de uma abordagem MDE para o desenvolvimento de aplicações móveis. Esta abordagem, chamada *XIS-Mobile*, usa uma DSL (definida como um perfil UML) e a sua *framework* é baseada na metodologia MDE. Propõem a definição de PIM para descrever aplicações móveis e a partir dele gerar automaticamente o código-fonte da aplicação para múltiplas plataformas.

O resultado obtido após a realização deste projeto foi a definição de uma DSL sob a forma de um perfil UML focado no domínio das aplicações móveis, a linguagem *XIS-Mobile* usa conceitos específicos das aplicações móveis, como é o caso de *gestures*, *providers* e *widgets*, fornece uma DSL que especifica uma aplicação móvel, aplica um conjunto de padrões User Interface (UI) para gerar e modelar aplicações móveis, e por fim, gera código para múltiplas plataformas através do uso de PIM definidos na linguagem *XIS-Mobile* que são transformadas em código usando o *Acceleo*.

3.4.2 *Desenvolvimento de aplicações android com Arctis*

Arctis é uma ferramenta para modelar aplicações com UML, que podem ser implementadas nas plataformas *android*. Uma biblioteca *Arctis* contém blocos de construção que acedem aos serviços do *android* ou executam outras tarefas que são usadas no desenvolvimento de projetos de sistema para aplicações *android*.

O objetivo exposto em [Hau09] é o de criar aplicações *android* através de blocos desenvolvidos na ferramenta *Arctis*. O principal foco do trabalho seria encontrar um método para desenvolver aplicações *android* através da ferramenta referida, ajustando o gerador de código Java Standard Edition (SE) existente com o intuito de trabalhar com *android* e fornecer os blocos de construção para o uso dos serviços do mesmo. Foi também proposta uma arquitetura de como deveriam ser desenvolvidas as aplicações usando esta ferramenta.

O resultado final desta dissertação consistiu no facto de o editor de *Arctis* ficar equipado com uma opção de *deployment* de aplicações *android* chamada *Java SE for Android*. Adicionalmente, foi criada uma biblioteca constituída por blocos de construção para o desenvolvimento de aplicações *android*.

3.4.3 *Desenvolvimento de aplicações android com Scala e SBT*

A meta estabelecida em [Fon14] foi a criação de uma plataforma completa e estável, de forma a promover o desenvolvimento e prototipagem rápida de aplicações *android* com recurso a *Scala*. *Scala* é uma linguagem de programação multi-paradigma orientada aos

objetos que compila o mesmo *byte-code* que *Java* e executa em qualquer Java Virtual Machine (JVM) [OAC+04].

A ferramenta foi implementada como um *plugin* do SBT que é uma *framework* de compilação para projetos *Scala* e *Java*. De salientar que a aplicação implementada consegue ser facilmente importada por um IDE como o *Eclipse* ou *Android Studio*. A interação com esta aplicação é feita através da linha de comandos em que, ao inserirmos uma série de instruções, são gerados os ficheiros necessários para que consigamos obter uma aplicação *android* totalmente funcional.

Este projeto fornece uma *framework* completa e utilizável para o desenvolvimento *android* com *Scala*.

3.4.4 *Geração de aplicações android usando Acceleo*

No seguimento do trabalho apresentado em [BEAM16] é exposta uma abordagem MDA para o desenvolvimento de aplicações *android*. Esta abordagem inclui a modelação UML e a geração automática de código usando o *Acceleo* com o objetivo de acelerar e facilitar o desenvolvimento de aplicações *android*. *Acceleo* [Acc] é um gerador de código *android open-source*. A geração de código do *android* é baseada em diagramas de classes, que são usados pelo *Acceleo* para gerar a estrutura da aplicação. A relação entre interfaces ou classes, como a associação ou herança, é respeitada durante a geração do código.

3.4.5 *PhoneGap*

PhoneGap é uma *framework* gratuita, *open-source*, que permite a criação de aplicações móveis utilizando tecnologias *web (HTML5, CSS3 e javascript)*. Para além da facilidade em aceder a recursos nativos do dispositivo, uma das maiores vantagens do *Phonegap* é permitir compilar o mesmo projeto para as principais plataformas, como: *android, IOS e Windows Phone* [GP12].

3.5 Geração de aplicações web

Como um dos objetivos primordiais desta dissertação é também a geração de aplicações *web*, foi, então, realizado um estudo em relação ao trabalho já realizado neste âmbito.

3.5.1 *Desenvolvimento de aplicações web usando AJAX*

O foco em [GMVD08] é criar uma aplicação *AJAX* através de um conjunto de ferramentas MDA *open-source*, *AndroMDA*.

AndroMDA [BB+07] usa os modelos UML como *input* utilizando uma *framework* para gerar código *AJAX*. É uma *framework* MDA *open-source* que aceita qualquer número de modelos (geralmente modelos UML) combinado com qualquer número de *plugins AndroMDA* e produz qualquer número de componentes personalizados. Pode-se gerar componentes para qualquer linguagem de programação como: *Java*, *.Net*, HTML, Hypertext Preprocessor (PHP), etc.

O resultado do trabalho de implementação desta dissertação é a adição de um componente *AJAX* para *AndroMDA* que pode ser usado para gerar aplicações *single-page AJAX*, de acordo com um conjunto de requisitos relacionados com a linguagem.

3.5.2 *Linguagem de modelação web (WebML)*

A Web Modeling Language (WebML) [CFB00] é uma notação visual para especificar a estrutura e navegação de aplicações *web*. A notação é bastante similar aos diagramas de classe e diagramas de entidade-relação do UML.

3.5.3 *Engenharia web baseada em UML (UWE)*

UML-based Web Engineering (UWE) é uma abordagem de engenharia de *software* para o desenvolvimento de aplicações *web* que é continuamente desenvolvida desde 1999 [BKM99]. A UWE sustenta o desenvolvimento de aplicações *web* com foco especial na sistematização [KK02]. O seu processo é orientado a objetos, iterativo e incremental, tentando abranger todo o ciclo de vida de uma aplicação *web*, tendo como principal foco a geração automática de aplicações [KK03].

3.5.4 *Geração de aplicações web com JavaServer Faces*

O objetivo em [OVD08] é a geração automatizada do *layout* e da estrutura de páginas *web*, utilizando o padrão Model View Controller (MVC). Para além disso, é gerado um ficheiro XML que serve como *input* para o *Controller* que gere o *pageflow*. O resultado final deste trabalho culmina com a geração de um protótipo *web* com base numa abordagem MDA. Este protótipo é uma aplicação *web* JavaServer Faces (JSF) para o servidor *web Apache Tomcat*.

Concluído o projeto, conseguiu-se introduzir uma forma de modelação e geração de aplicações *web* com base numa abordagem MDA. Definiu-se um meta-modelo que especi-

fica a DSL para modelos de aplicações *web*. Para além disso, etapas de transformações e *templates* foram criados para gerar aplicações *web* modeladas baseadas na *framework* JSF. O *deployment* da aplicação gerada pode ser feito num servidor *web Apache Tomcat*.

3.5.5 *ArgoUWE*

É apresentada uma ferramenta *ArgoUWE* em [KKMZ03] que descreve os conceitos UWE subjacentes, as funcionalidades fornecidas pela mesma aos utilizadores e a sua arquitetura. *ArgoUWE* é baseado em *ArgoUML* e faz uso da interface gráfica de utilizador do *ArgoUML*. Basicamente, é uma ferramenta aliada à *framework ArgoUML*, ou seja, é um *plugin* que se adiciona à ferramenta anteriormente referida, para desenvolver aplicações *web*.

Neste trabalho é então apresentada a ideia de como *ArgoUWE* pode conseguir gerar uma aplicação*web* baseada numa abordagem *mde*. Adicionalmente é apresentado um exemplo executável de como a ferramenta lida com o *design* de 3 principais componentes dos modelos UWE: modelo conceptual, modelo de navegação e modelo de apresentação.

3.5.6 *Geração de aplicações web através de UML*

O trabalho apresentado [CLH+] revela uma estrutura generativa visando simplificar o desenvolvimento de aplicações *web*, com base na construção e transformação de modelos, utilizando um perfil UML e uma DSL construída para esse fim.

O projeto é parte de um programa denominado Model Transformation Environment (MTE) que pretende aumentar a qualidade e produtividade do desenvolvimento de *software*, com foco na construção e transformação de modelos, com o objetivo de algumas atividades que são feitas manualmente, poderem ser automatizadas.

Este trabalho descreveu os elementos utilizados no processo de transformação de um modelo de domínio UML numa aplicação *web* implementável. Esses elementos incluem uma extensão de UML por meio de um perfil de aplicação *web* chamado *WebApp*; Uma especificação de uma DSL representada num meta modelo chamado *WAMM*; A definição de uma M2T para transformar um modelo de domínio, ao qual o perfil *WebApp* foi aplicado, numa instância do meta modelo *WAMM*; E finalmente a definição de uma transformação M2T, de modo a que o código executável possa ser gerado a partir de uma instância do *WAMM*.

3.5.7 *OpenXava*

OpenXava [Mel], é uma *framework* que visa a geração de aplicações *web Enterprise*. É usada para o rápido desenvolvimento de aplicações *Java*. Ao mesmo tempo, o *OpenXava* é ex-

tensível, personalizável e o código da aplicação é estruturado de uma forma orientada a objetos, permitindo desenvolver aplicações complexas. A abordagem *openXava* para o desenvolvimento rápido de aplicações *web*, difere daquelas que usam ambientes visuais como o PHP. Em vez disso, *OpenXava* usa uma abordagem MDE onde o núcleo da sua solução se baseia em classes *Java* que modelam o problema mantendo um alto nível de encapsulamento [Pan11].

3.6 CONCLUSÃO

Apesar dos diversos projetos apresentados neste capítulo 3, o trabalho que integra a presente dissertação apenas integrará o projeto apresentado em 3.4.3, que terá um papel fulcral na geração de aplicações *android*. Os restantes temas que serão explorados neste projeto, como por exempl,o a geração de aplicações *web* e as transformações modelo para texto, seguirão abordagens diferentes das que foram apresentadas no presente capítulo, não sendo integrado nenhum trabalho exposto como no caso da geração de aplicações *android*.

4

ABORDAGEM PROPOSTA

Depois de analisados todos os componentes necessários, bem como os trabalhos relacionados com o âmbito da presente dissertação, é necessário tomar decisões para que haja um guia de trabalho que ajude a uma implementação eficaz e organizada do que é pretendido com este projeto. Posto isto, algumas escolhas foram tomadas em relação a alguns aspetos fulcrais desta tese e serão, então, expostas ao longo deste capítulo.

Um aspeto importante foi definir-se, desde logo, a família de aplicações que serão geradas. Pretende-se, então, que sejam geradas aplicações que dado um diagrama de classes, em que cada classe constituinte do mesmo representa uma entidade, serão geradas as operações CRUD para a mesma. Isto implica que haja a necessidade da existência de uma interface para o utilizador em que este possa realizar as operações referidas, (pelo menos) uma tabela na Base de Dados para cada modelo, bem como as suas relações com as outras entidades constituintes do diagrama.

Para se realizar a modelação será escolhida a ferramenta *Visual Paradigm*, pois esta é bastante utilizada, não obriga a utilização de nenhum IDE específico e tem todas as funcionalidades necessárias e que serão úteis para a realização deste projeto, sendo exemplo a exportação de diagramas em formato XML.

Quanto às transformações M2T, depois de analisadas algumas técnicas para o efeito, optar-se-á por utilizar bibliotecas que permitam fazer a interpretação dos modelos após a sua exportação em formato XML, sem que seja necessário o recurso a outro tipo de ferramentas, ficando a cargo da aplicação geradora de aplicações CRUD esta tarefa.

Falando das aplicações *android*, a abordagem que irá ser seguida será idêntica à que foi apresentada na secção 3.4.3. O objetivo será tentar integrar com o código da dissertação (que se encontra disponível), a possibilidade de, através de um *script*, gerar a aplicação CRUD para todos os elementos presentes no diagrama de classes passado à aplicação, ou seja, seria feito o aproveitamento do trabalho realizado na dissertação já falada 3.4.3, sendo que seria adicionada a opção de em vez de haver interação utilizador-programa através da linha de comandos, a mesma seria feita através de diagrama de classes *UML*, evitando assim a inserção de comandos por parte do utilizador, assim como acrescentar as funcionalidades necessárias para que se gerem, efetivamente, aplicações CRUD.

No que toca à geração das aplicações *web*, sabendo-se de antemão que há inúmeras formas de se construir as mesmas devido à existência de várias linguagens e *frameworks* para o efeito, decidimos optar pela geração de um servidor *NodeJS* que permite a disponibilização de uma API REST para cada modelo, gerindo uma base de dados que contém uma tabela referente ao mesmo, bem como as relações que o modelo tem com as restantes entidades. Para que esta API REST seja facultada ao utilizador de uma forma amigável, é também gerada, para o lado do cliente, uma Single-Page Application (SPA). Esta tem como principal intuito fornecer através de uma interface gráfica não só todas as operações CRUD para cada modelo ao utilizador final, mas também a possibilidade de uma definição simples de relacionamentos entre entidades. Para este efeito a escolha recaiu por utilizar uma *framework* de *javascript*, *AngularJS*.

Por fim, a aplicação que irá ser geradora de aplicações CRUD quer para o âmbito do *android*, quer no âmbito da *web*, irá ser construída em *JavaFX*. Isto porque é uma linguagem de *desktop* que inclui a possibilidade de elaboração de um ambiente gráfico, evitando assim que o utilizador final da aplicação tenha de usar a linha de comandos. Esta aplicação terá como objetivo final a criação de aplicações robustas, tendo como principais os seguintes passos: receber como *input* os diagramas UML exportados em formato XML, tratá-los e, posteriormente, gerar aplicações *android* e/ou *web*, conforme a escolha do utilizador, que efetuem com sucesso as operações CRUD referentes a cada modelo.

5

TRANSFORMAÇÕES MODELO PARA TEXTO

5.1 CONTEXTUALIZAÇÃO

Como se havia abordado no capítulo anterior, apesar das diversas formas e ferramentas estudadas que visam o tratamento e análise dos diagramas exportados em formato XML, optou-se por apenas utilizar-se uma biblioteca *Java* (org.w3c.dom) que fizesse o pretendido, integrando-a com a aplicação final desenvolvida em *JavaFX*. Esta decisão prendeu-se com o facto de, para o que era necessário obter dos ficheiros XML, com o objetivo de que a geração de aplicações CRUD fosse possível, não ser necessária a integração de uma ferramenta ou de algo muito complexo com a aplicação, sendo então suficiente a solução adotada para o efeito.

5.2 ESTRUTURA DOS DIAGRAMAS DE CLASSE

Visto que a ferramenta de modelação *Visual Paradigm* permite inúmeras formas de se construirem diagramas de classe, com diversos elementos que permitem especificá-los e completá-los ao mais ínfimo detalhe, surgiu a necessidade de se estabelecerem regras para que a análise e obtenção da informação provida pelo ficheiro XML fosse garantida com sucesso.

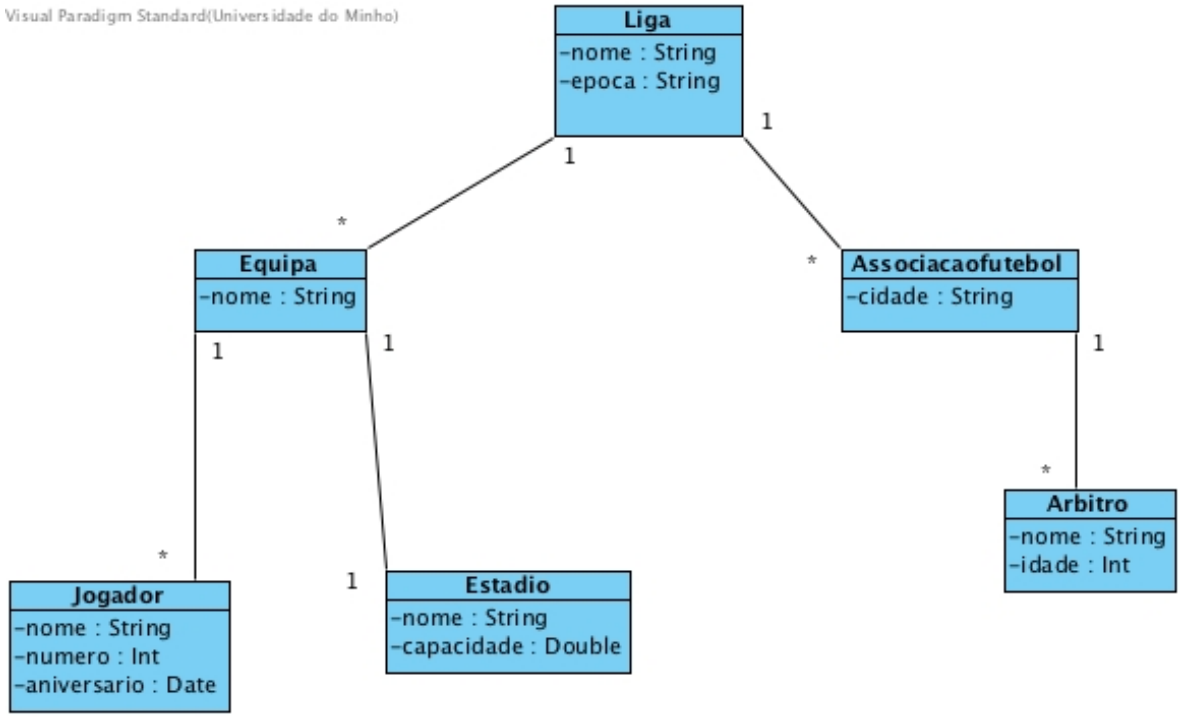


Figura 6: Estrutura do diagrama de classe a ser exportado.

Analisando a figura 6, é importante tirar algumas ilações para que a forma de construção do diagrama de classe seja bem compreendida e realizada de forma correta. Começando pelas entidades, cada classe deve ter o seu nome escrito com a primeira letra maiúscula e as restantes minúsculas. Para adicionar um atributo a uma classe deve-se ter em atenção que a aplicação suporta os seguintes tipos de dados: *Int*, *Float*, *Double*, *Boolean*, *String*, *Array* e *Date*. Para além disto, o nome dos atributos deve estar escrito todo em minúsculas. Para estabelecer a ligação entre duas entidades, ter-se-á de usar o elemento *association*. Terminado este processo, ficará apenas a faltar um passo: definir as multiplicidades entre os modelos, isto é, dizer se um determinado modelo pode ter 0, 1 ou N entidades de outro. É importante referir que no que toca às multiplicidades de relações entre modelos, foram definidas as seguintes normas:

- Não é possível haver relações entre modelos com multiplicidade 0 para 0.
- Se houver relações de 0 para 1 ou para N, o modelo que fica com o atributo que identifica a relação entre as entidades é o modelo que tem a multiplicidade de valor 0.
- Se houver relações de 1 para N, o modelo que fica com o atributo que fica com o atributo que identifica a relação entre as entidades é o modelo que tem a multiplicidade de valor 1.

- Se houver relações de 1 para 1 ou de N para N, o modelo que fica com o atributo que fica com o atributo que identifica a relação entre as entidades é o modelo onde o elemento do tipo *association* previamente falado tem como origem.

Depois de estarem definidas as classes, atributos e relações entre as mesmas, ter-se-á apenas de exportar o diagrama de classes em formato XML como está representado na figura 7:

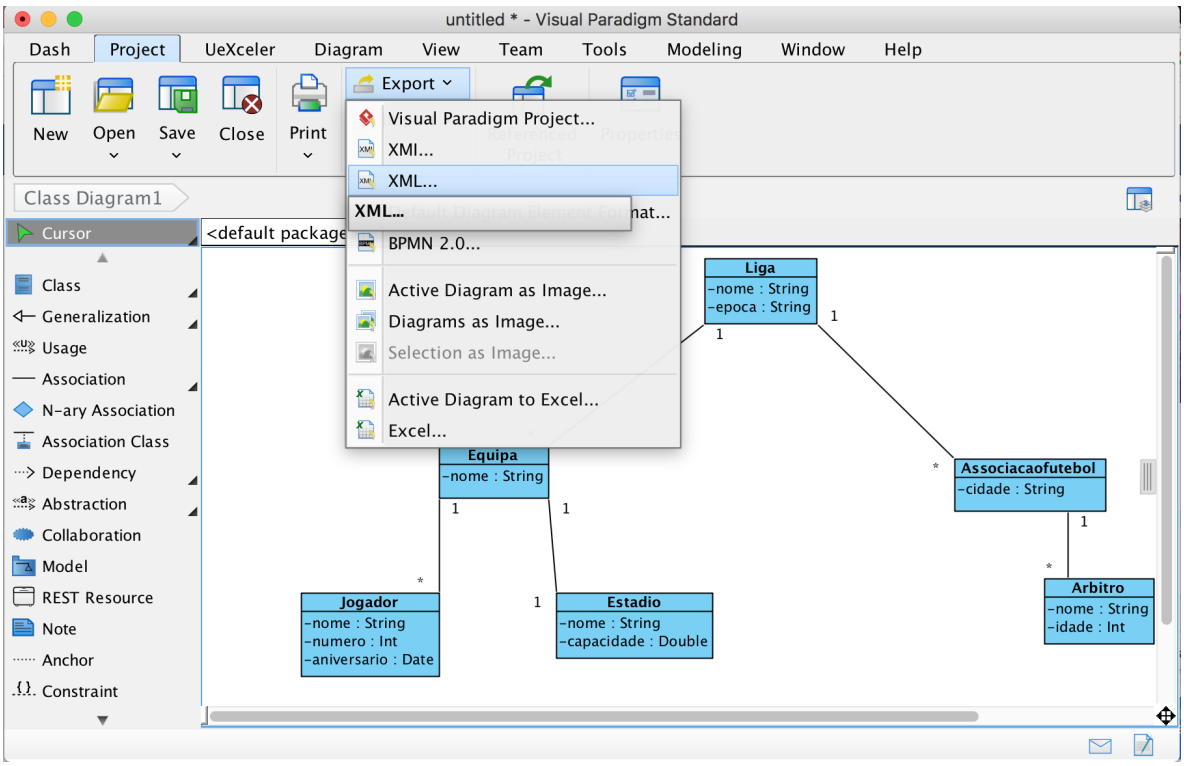


Figura 7: Exportação de um diagrama de classes em formato *XML*

Após a exportação do diagrama de classes, obter-se-á então um ficheiro XML que contém todas as informações relevantes para que a geração de aplicações CRUD seja possível, sendo este ficheiro que deverá ser submetido na aplicação geradora para que o objetivo previamente referido seja alcançado.

6

GERAÇÃO DE APLICAÇÕES ANDROID

6.1 AGILE-SCALA-ANDROID

Como já referido em 3.4.3, esta ferramenta foi elaborada com o intuito de gerar aplicações *android* de forma rápida e relativamente simples. Posto isto o objetivo é, então, a integração da mesma com o trabalho elaborado na presente dissertação. De salientar que toda a interação com esta aplicação é realizada via linha de comandos. Para uma melhor perceção de como funciona a aplicação *agile-scala-android*, analisemos a figura 8,

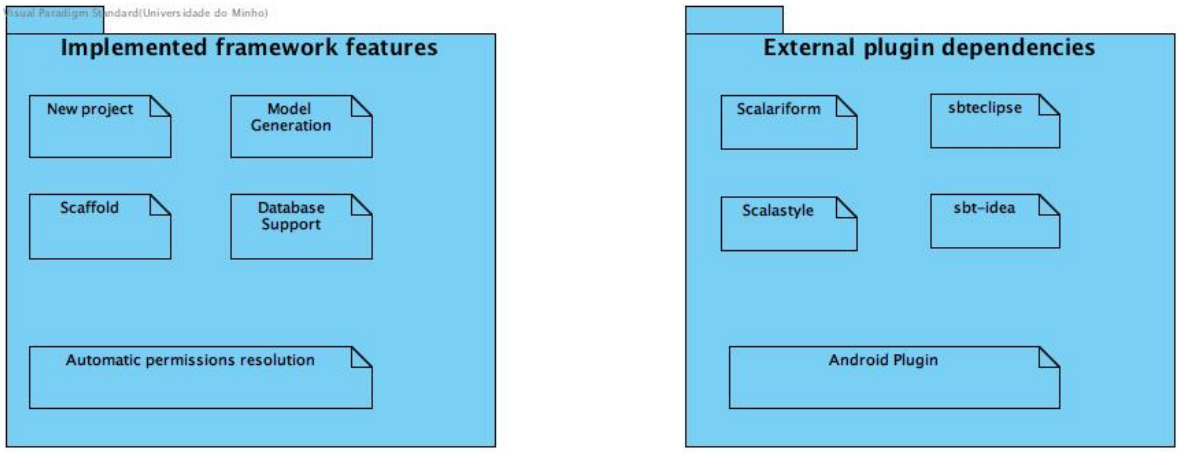


Figura 8: Estrutura da aplicação agile-scala-android.

Começando pelas funcionalidades que foram elaboradas no projeto:

- New Project - Capacidade de criar um novo projeto *SBT plugin*. Comando para gerar um novo projeto:

```
$ sbt
> npa NOME_PRJETO VERSAO_ANDROID
```

O comando *npa* aceita apenas dois argumentos que correspondem ao nome do *package* onde a aplicação estará inserida e a versão da *android* API, respetivamente.

- Model Generation - Capacidade de gerar classes através de modelos bem como as suas variáveis de instância.

```
$ sbt
> generate MODELO NOME_DO_ATRIBUTO:TIPO_DO_ATRIBUTO
```

- Scaffold - Gera as *views* e *controllers* inerentes aos modelos para a concretização das aplicações CRUD.

```
$ sbt
> scaffold MODELO
```

- Database Support - Ligação com a base de dados para que a(s) tabela(s) inerentes às classes sejam geradas. É usado *Slick* como motor de *queries* da base de dados.

```
$ sbt
> migrateDatabase
```

- Automatic permissions resolutions - Resolução de dependências necessárias para o bom funcionamento das aplicações *android*, utilizando o *PSCout* que é uma ferramenta que faz o mapeamento da API do *android* com as permissões.

```
$ sbt
> checkPermissions
```

Do lado direito da Figura 8 temos os *plugins* que foram integrados na ferramenta para que esta funcionasse de forma correta:

- sbteclipse - *Plugin* que facilita a integração da ferramenta com o IDE *Eclipse*.
- sbt-idea - *Plugin* que facilita a integração da ferramenta com o IDE *IntelliJ*.
- Android Plugin - Adiciona vários recursos específicos para a criação de aplicações *android*.

- Scalariform - Reformata o código para uma forma específica.
- Scalastyle - É uma ferramenta de análise de código que verifica possíveis problemas no mesmo, principalmente questões de formatação.

6.1.1 *Requisitos necessários*

O primeiro passo para integrar a ferramenta de geração de aplicações foi através do *Homebrew*, instalar o SBT. Para isto, é apenas necessário correr o seguinte comando:

```
brew install sbt
```

Será também preciso instalar o SDK do *android* para ficarmos aptos a criar um novo projeto e instalar o *plugin*. A instalação do supracitado SDK pode ocorrer de 3 formas:

- Se o download foi feito a partir do próprio *website*, então o seu *path* será:

```
/Downloads/ADT/sdk
```

- Se se instalar o SDK a partir do *Homebrew*, então o seu *path* será:

```
/usr/local/Cellar/android-sdk/{NUMERO_VERSAO_SDK}
```

- Se foi instalado automaticamente a partir do *Android Studio*, então o seu *path* será:

```
/Users/{NOME_DE_UTILIZADOR}/Library/Android/sdk
```

6.1.2 *Criação de um novo projeto e configuração do plugin SBT*

Para a criação de um novo projeto *android* e da configuração do SBT, será necessário alterar alguns ficheiros e criar uma diretoria para que lá fique alojado o projeto (no exemplo será no *Desktop*):

- Comando que cria a diretoria */.sbt/0.13/plugins* se ainda não existir e cria um ficheiro *agile-scala-android.sbt*.

```
echo 'addSbtPlugin("pt.pimentelfonseca" % "agile-scala-android" %
    "0.6")' > ~/.sbt/0.13/plugins/android.sbt
```

- Comando que adiciona as configurações *agileAndroidNewProjectTask* globalmente, o que irá adicionar apenas uma única tarefa: npa, e cria um ficheiro chamado *npa.sbt* na diretoria */.sbt/0.13/*.

```
echo 'seq(AgileAndroidKeys.agileAndroidNewProjectTask: _*)' > ~/.sbt
    /0.13/npa.sbt
```

- Comandos de criação da diretoria "test"no *Desktop*:

```
cd Desktop
mkdir test
cd test
```

Estando dentro da pasta "test"ter-se-á de correr o seguinte comando

```
$ sbt
> npa test.pt 20
```

Aquando da introdução do mesmo, caso não esteja definida a variável de ambiente *ANDROID_HOME*, ocorrerá o seguinte erro: set ANDROID_HOME or run 'android update project -p android studio'. Ter-se-á então de definir a variável de ambiente num ficheiro chamado *.bash_profile*. Para mitigar este erro, basta então adicionar as seguintes linhas ao ficheiro previamente referido:

```
export ANDROID_HOME=.../Android/sdk
export PATH=$ANDROID_HOME/platform-tools:$PATH
export PATH=$ANDROID_HOME/tools:$PATH
```

A segunda e a terceira linhas são necessárias para o caso de estar a ser utilizada uma versão do SDK do *android* igual ao superior à 20, isto porque caso contrário, irá ocorrer o seguinte erro:

```
Cannot run program ".../Library/Android/sdk/tools/zipalign": error=2,
    No such file or directory
```

O *Zipalign* é uma ferramenta usada para realinhar os dados da aplicação que não estão comprimidos, providenciando, assim, uma importante otimização para a mesma [KG]. Após a definição das variáveis de ambiente, ter-se-á também de copiar o ficheiro *zipalign* da pasta *.../Library/Android/sdk/build-tools* para a pasta *.../Library/Android/sdk/tools*, isto porque nas versões SDK anteriores, o *zipalign* encontrava-se na pasta *build-tools*, tendo depois sofrido esta alteração.

6.1.3 *Integração da ferramenta Agile-Scala-Android com Android Studio*

Apesar da aplicação referida ser de fácil integração tanto com o IDE *Eclipse* como com o IDE *IntelliJ*, a verdade é que, no que toca ao desenvolvimento de aplicações *android*, estes IDE caíram em desuso e portanto não é muito vantajosa a integração da ferramenta com os mesmos. Posto isto, surgiu a necessidade de fazer a integração do *Agile-Scala-Android* com o IDE *Android Studio*. Para isto, seguiu-se o tutorial *"Scala on Android"* (que se encontra em: http://scala-on-android.taig.io/editor/android-studio/).

Após a realização do mesmo, foi necessária a alteração da forma como é feito o *deploy* da aplicação *android*, sendo então necessário escolher o menu *"file"*, posteriormente escolher a opção *"Project Structure"*e por fim selecionar *"Packaging"*e escolher como diretoria de *deployment* a pasta *android-bin* do projeto, como está explícito na figura 9

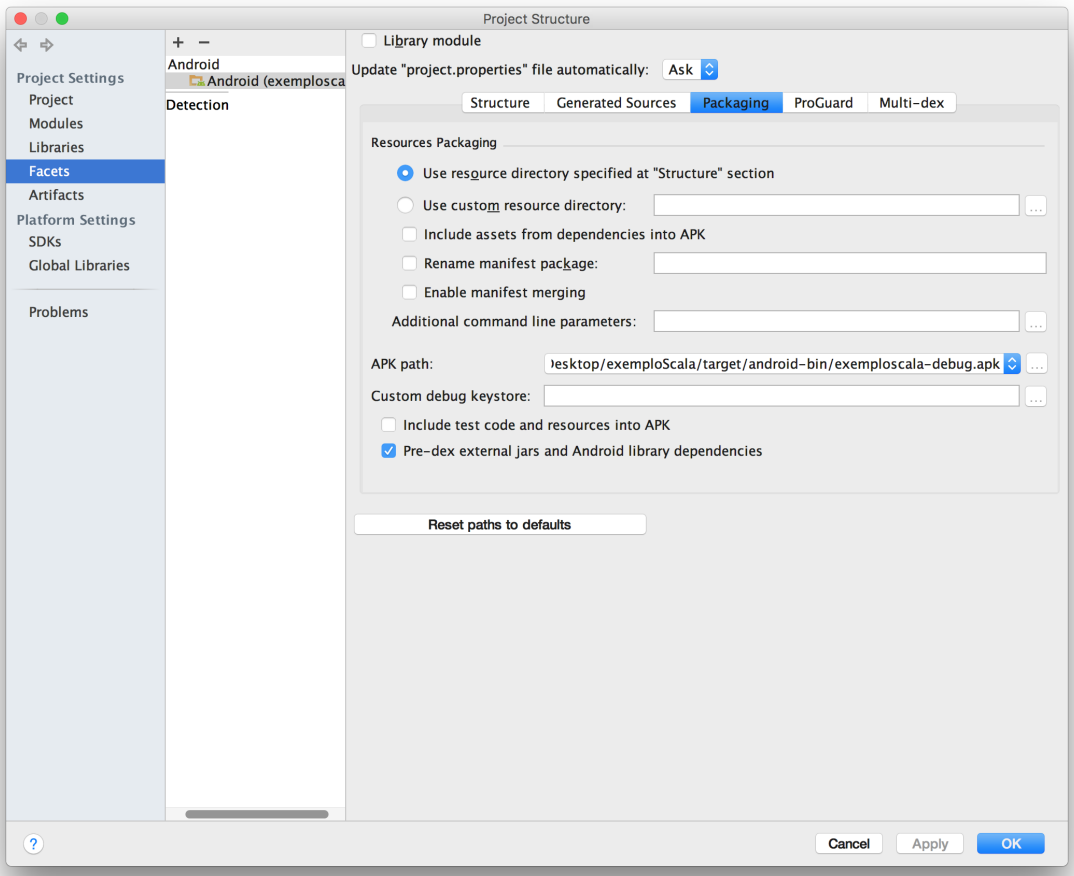


Figura 9: Deployment da aplicação *android* gerada através da aplicação *Agile-Scala-Android*

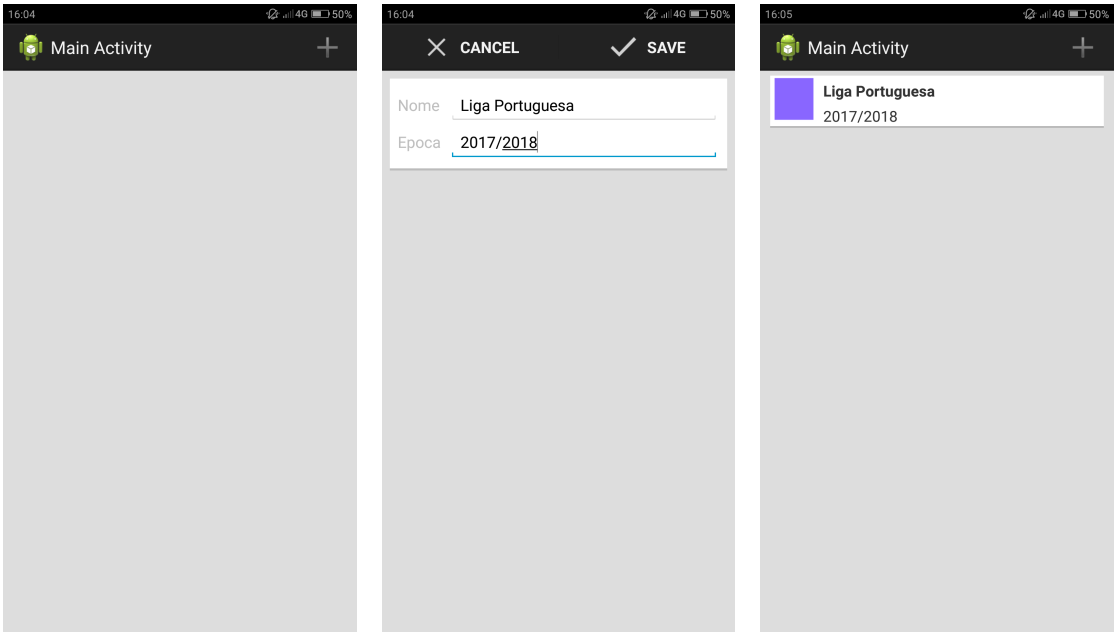
Para que seja possível fazer *debug* à aplicação gerada pela ferramenta previamente referida, será necessário adicionar a seguinte linha à tag *"application"*do ficheiro *AndroidManifest.xml*:

```
android:debuggable="true"
```

6.2 APLICAÇÕES *android* GERADAS

Após a correção de alguns erros, bem como a substituição de algumas ferramentas e bibliotecas que tinham caído em desuso, a integração com a ferramenta *Agile-Scala Android* foi conseguida com sucesso. No entanto, as operações CRUD relativas a cada modelo

não se encontravam implementadas, apesar de estar tudo bem estruturado para que tal fosse possível, pelo que foi necessário assegurar que as operações fossem garantidas, sendo que o maior desafio foi compreender o funcionamento da aplicação referida em 3.4.3, e como se conseguiria interagir com a mesma, para que o objetivo já referido fosse atingido. Após este obstáculo estar ultrapassado, a meta da geração de aplicações CRUD para *android* foi alcançada. Tomando como exemplo apresentado em 5.2, podemos observar como será o resultado tipicamente obtido após a aplicação do trabalho desenvolvido na presente dissertação, estando o mesmo representado nas seguintes imagens:

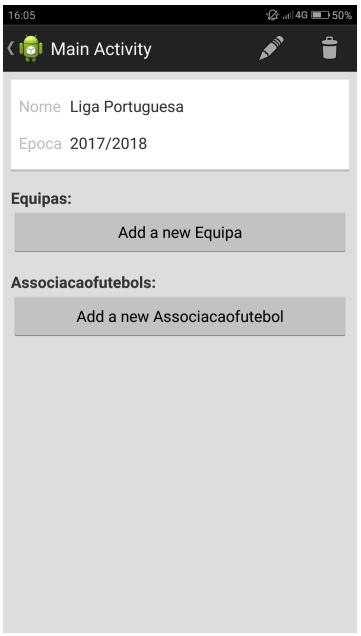


(a) Vista da aplicação sem dados inseridos

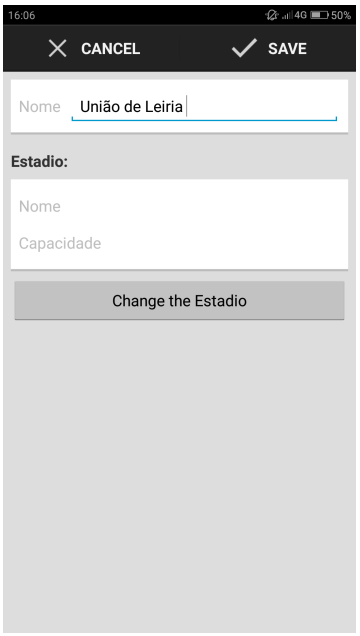
(b) Vista de inserção de uma Liga

(c) Vista da Liga após a sua inserção

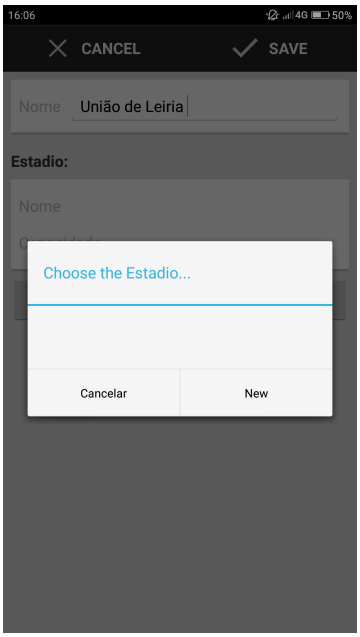
Figura 10: Estrutura das aplicações *android* geradas.



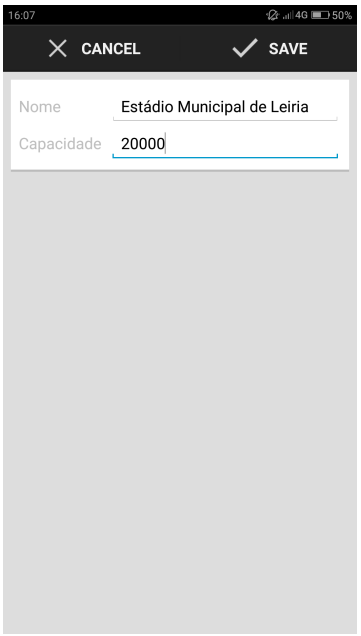
(d) Vista de apresentação de uma Liga, com possibilidade de editar e eliminar a mesma.



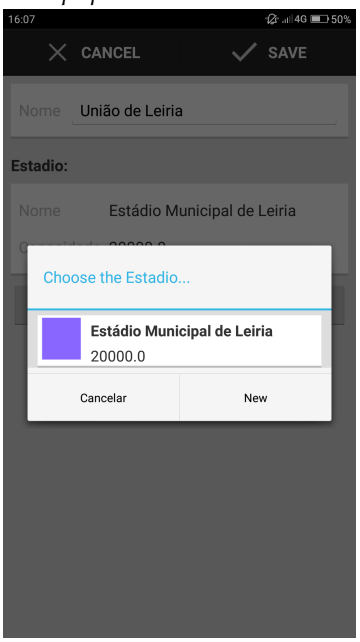
(e) Vista de inserção de uma Equipa, após clicar no botão *"Add a new Equipa"*em 10c.



(f) Escolher um estádio para associar a uma equipa depois de clicar no botão *"Change the Estadio"*em 10e



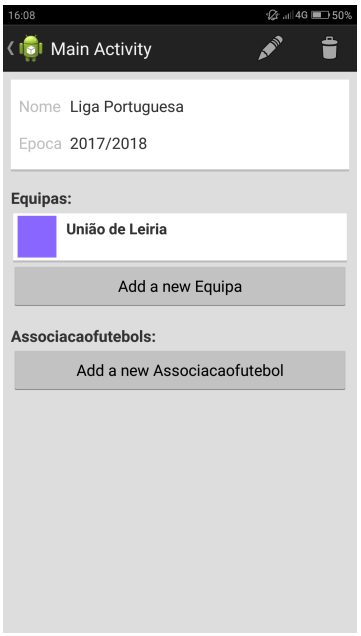
(g) Vista de inserção de um estádio, após clicar no botão *new* em 10f.



(h) Associar um Estádio a uma Equipa, após clicar no botão *save* em 10g.

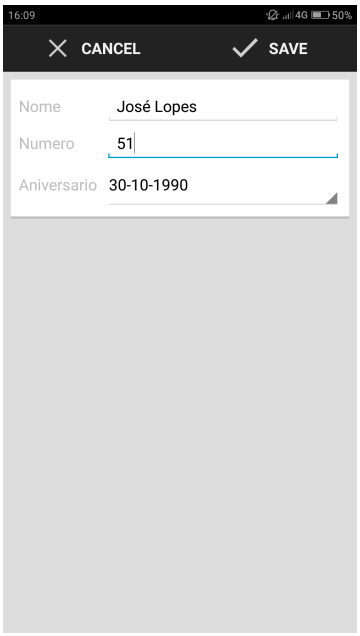
(i) Vista para a criação de uma equipa, após associar-se o Estádio à mesma em 10h.

Figura 10: Estrutura das aplicações *android* geradas.

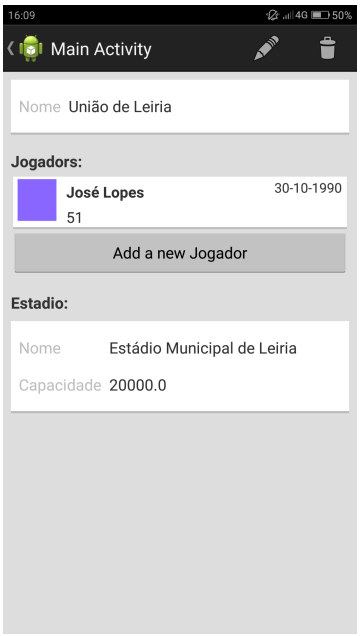


(j) Vista de apresentação de uma Liga após a inserção de uma Equipa, ao clicar no botão *save* em 10i

(k) Vista de apresentação de uma Equipa, após clicar no botão roxo *União de Leiria* em 10j.



(l) Vista inerente à inserção de um Jogador após clicar no botão *Add a new Jogador* em 10k.

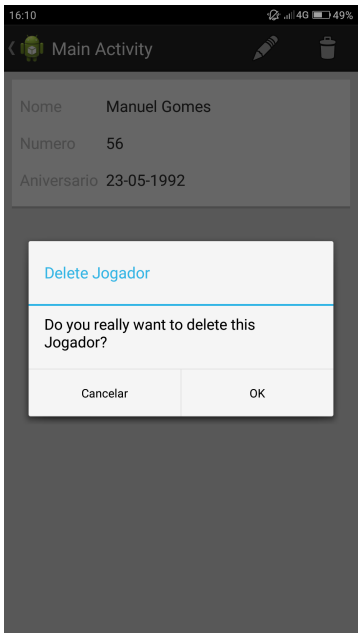


(m) Vista de apresentação de uma Equipa após a inserção de um Jogador, ao clicar no botão *save* em 10l.

(n) Vista da edição de um Jogador, alterando os campos relativos ao mesmo, atualizando a instância apresentada em 10m.

(o) Vista de apresentação de uma Equipa após a edição de uma instância de um Jogador, ao clicar no botão *save* em 10n.

Figura 10: Estrutura das aplicações *android* geradas.



(p) Vista da apresentação de um Jogador, fornecendo a possibilidade de eliminar a instância apresentada em 10m.

(q) Vista de apresentação de uma Equipa, após clicar no botão "*ok*"em 10p da instância apresentada em 10o.

Figura 10: Estrutura das aplicações *android* geradas.

Consegue-se, então, com as imagens exibidas anteriormente, ter uma melhor perceção do funcionamento da aplicação e como são feitas as operações CRUD relativas a cada modelo, ficando-se também com uma noção do *workflow* que as aplicações geradas tendem a seguir. Não foram apresentadas todas as operações associadas a todas as entidades apresentadas em 6, pois a sua lógica e apresentação são, em tudo, semelhantes às demais apresentadas previamente.

7

GERAÇÃO DE APLICAÇÕES *WEB*

7.1 SERVIDOR

Apesar de os projetos estudados no âmbito do estado-da-arte da geração de aplicações *web* não se enquadrarem nesta solução, aquando da geração destas aplicações, tomou-se a decisão de seguir um padrão que hoje em dia está muito em voga: uma aplicação cliente e outra servidor. Posto isto, optou-se por usar *NodeJS* como linguagem principal para o servidor. O *NodeJS* é uma ferramenta construída sobre o motor de *javascript V8* do *Chrome*, que torna possível desenvolver aplicações para o lado do servidor utilizando esta linguagem. Neste caso, a aplicação servidor disponibiliza uma API REST, que não é mais que um conjunto de princípios *standard* de interoperabilidade entre sistemas via *web* [Mas11], que pode ser consumida por diversas aplicações cliente, podendo elas ser dos mais diversos tipos (browser, smartphones, IoT) obtendo-se sempre uma resposta eficaz do mesmo. A Figura 11 retrata o que foi descrito anteriormente, dando a possibilidade de se compreender melhor o funcionamento desta arquitetura:

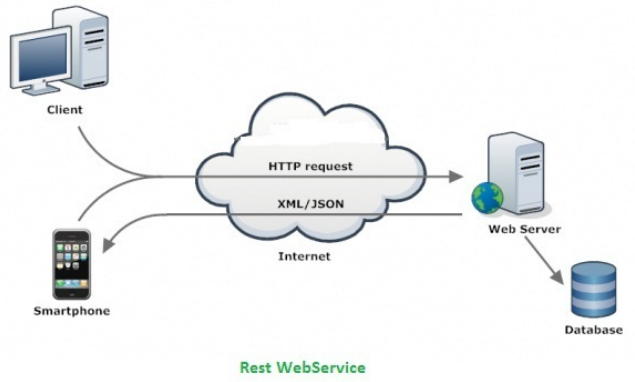


Figura 11: Exemplo de comunicação entre servidor e clientes numa API REST

Esta linguagem força a adoção de um estilo de programação orientada ao evento - ou programação assíncrona - o que significa que em vez de se usar o resultado de uma

função, os programadores definem funções que são invocadas através de um *event handler* ou de uma *callback* quando um determinado evento ocorre. Define-se por ser um estilo de programação onde o fluxo de programação é determinado pelos eventos, garantindo que os processos de *input/output* nunca bloqueiam, permitindo que sejam atendidos vários pedidos em simultâneo, sem que a integridade das aplicações seja comprometida [Tei12].

Para a criação rápida de uma API REST dinâmica com pequeno ou nenhum esforço de programação, foi utilizada a *framework Loopback* de *NodeJS*, sendo que para a gestão da base de dados foi utilizado *MySQL*.

7.1.1 *Requisitos necessários*

Para que se consiga ter a geração do servidor *NodeJS* completamente funcional, será necessário ter instalado na máquina alguns componentes essenciais. O primeiro será, obviamente, o *NodeJS*, sendo que para isso há duas formas de o fazer, ou via terminal [1] ou descarregando do próprio site [2] os pacotes para que a instalação seja feita corretamente na máquina. De notar que é crucial garantir que o comando Node Package Manager (NPM) fique também ele instalado.

Será também necessária a instalação da *framework* referida no início do presente capítulo 7.1, *loopback* para que a a geração do servidor seja lograda, e para isto basta correr-se os seguintes comandos:

```
npm install -g strongloop
```

```
npm install -g loopback-cli
```

O primeiro comando serve para instalar a ferramenta para linha de comandos *slc*. O segundo tem propósito de facultar ao utilizador a possibilidade de criar rapidamente uma API REST.

Por fim, a instalação do *MySQL* é também estritamente necessária para o bom funcionamento da aplicação. Para a mesma há, igualmente, duas maneiras de a instalação ser efetuada, descarregando os pacotes de do *MySQL* do site [3], sendo a alternativa, mais uma vez, recorrer ao uso da linha de comandos, inserindo o seguinte comando:

```
brew install mysql
```

1 https://changelog.com/posts/install-node-js-with-homebrew-on-os-x
2 https://nodejs.org/en/download/
3 https://www.mysql.com/downloads/

E instalar-se no *Loopback* o conetor *MySQL*, sendo que isto é feito através do seguinte comando:

```
install loopback-connector-mysql --save
```

Após a instalação destas ferramentas, estão reunidas as condições para que se possa configurar e prosseguir à geração do servidor que é parte fulcral das aplicações *web*. Os comandos expostos neste capítulo são direcionados para o sistema operativos *OSX* pelo que para os outros, ter-se-á de adaptar os comandos para que a instalação dos componentes seja igualmente conseguida.

7.1.2 *Criação e configuração do servidor*

Com o objetivo de obter um servidor completamente funcional, o utilizador terá de seguir os seguintes passos antes de proceder à geração do mesmo:

- Criar uma diretoria para o projeto:

```
mkdir NOME_DA_DIRETORIA
cd NOME_DA_DIRETORIA
```

- Criação de um servidor através do Loopback:

```
slc loopback
```

Após a inserção deste comando, irá surgir o seguinte menu:

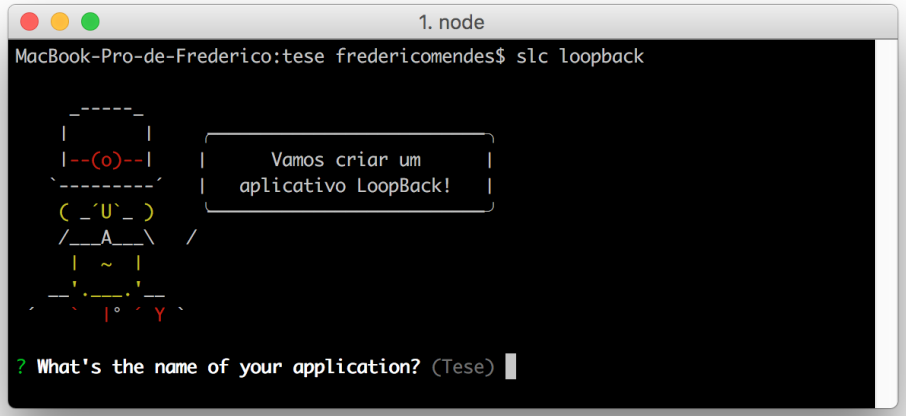


Figura 12: Primeiro passo para a geração do servidor *NodeJS*.

Sendo que o utilizador só terá de premir a tecla *Enter* para que surja, depois, o seguinte:

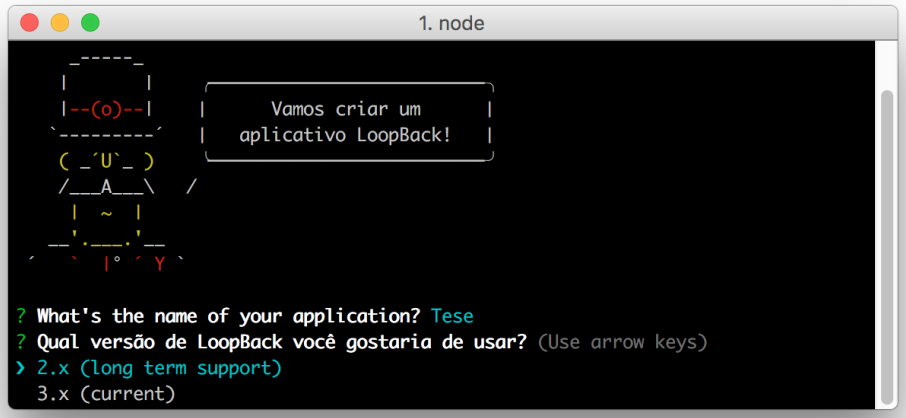


Figura 13: Segundo passo para a geração do servidor *NodeJS*

O utilizador deverá selecionar a opção *"2.x (long term suport)"*e prosseguir para o último menu:

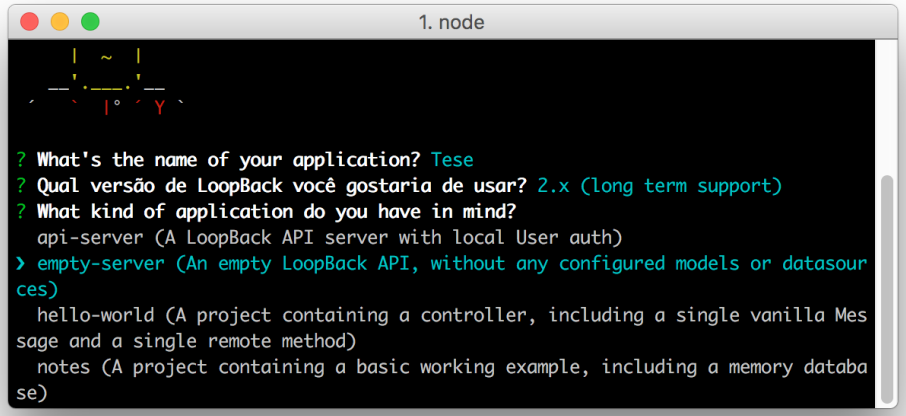


Figura 14: Terceiro e último passo para a geração do servidor *NodeJS*

Por fim, deverá escolher-se a opção selecionada "*empty-server*", para que esta etapa da criação do servidor esteja concluída.

- Criação de um schema para a base de dados - O utilizador deverá criar um *schema* para a base de dados. Para esta tarefa poderá utilizar, por exemplo, a ferramenta *MySQL Workbench*, sendo que a criação do mesmo é bastante fácil como se pode verificar pela seguinte imagem:

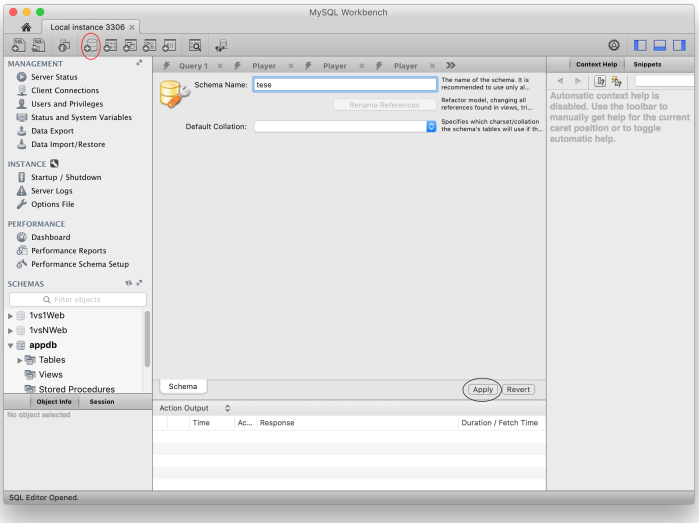


Figura 15: Criação de um *schema* para a base de dados do servidor *NodeJS*

Depois de abrir o programa para a criação do *schema*, ter-se-á de clicar no botão que está rodeado na imagem a vermelho para proceder à criação do mesmo, preenhcer o campo *Schema Name* e carregar no botão circundado a preto "*Apply*", como se pode verificar pela Figura 15. A partir daqui, estão dados todos os passos necessários para que a geração do servidor *NodeJS*, que é parte essencial das aplicações *web*, seja conseguida com sucesso. Neste ponto, o código do servidor criado terá a seguinte organização:

7.1.3 *Servidor NodeJS gerado*

Depois de se seguir os passos enunciados na secção anterior, o servidor criado terá a seguinte organização de diretorias:

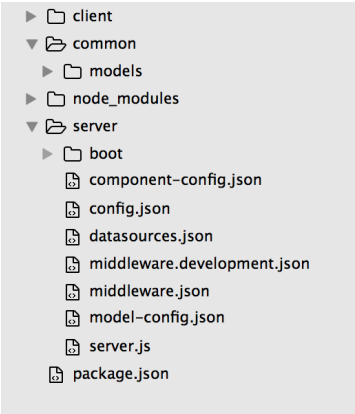


Figura 16: Organização do servidor após a sua criação

Analisando a imagem, há 4 diretorias principais, a *common* onde se podem encontrar os ficheiros relativos a cada modelo constituinte do diagrama de classes, a *node_modules* onde se encontram as bibliotecas e dependências relativas ao *NodeJS*, a *server* que contém os ficheiros de configuração do servidor e por fim, a diretoria *client* onde se inserem os ficheiros *HTML*, *JavaScript*, etc. É importante referir que o servidor neste momento não está apto a fornecer uma API REST pronta a ser consumida pelas aplicações cliente, pelo que é necessário usar a aplicação criada no âmbito desta dissertação, para que tal seja possível. Posto isto, para que pudéssemos atingir, efetivamente, o objetivo revelado anteriormente, foi utilizada uma biblioteca *Java* (com.google.gson) que nos permite interpretar e escrever documentos JavaScript Object Notation (JSON) de uma forma trivial.

Foi, também, necessária a alteração de alguns ficheiros revelados na imagem 16, realçando-se os seguintes:

- *middleware.json* - onde definimos a localização em disco dos ficheiros estáticos a serem servidos pelo servidor (neste caso, a aplicação cliente).
- *datasource.json* - onde se definem os detalhes de ligação à base de dados e respetivas credenciais.
- *package.json* - onde se adicionam as dependências necessárias para que a integração com o *MySQL* seja possível.

Para além dos ficheiros mencionados, dentro da diretoria *common/models*, para cada modelo, são sempre criados dois ficheiros, nomeadamente o *NOME_DO_MODELO.json* e o *NOME_DO_MODELO.js* onde são definidos os nomes e tipos dos atributos pertencentes às entidades, bem como a relação entre os modelos constituintes da aplicação que se pretende que seja gerada.

O resultado depois de alterar o servidor através da nossa ferramenta é retratado na Figura 17, onde é exposta a API REST correspondente ao exemplo apresentado em 6:

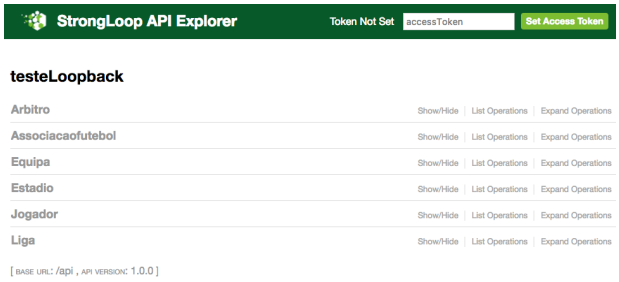


Figura 17: Entidades disponibilizadas pelo Servidor REST - vista da ferramenta *Swagger*

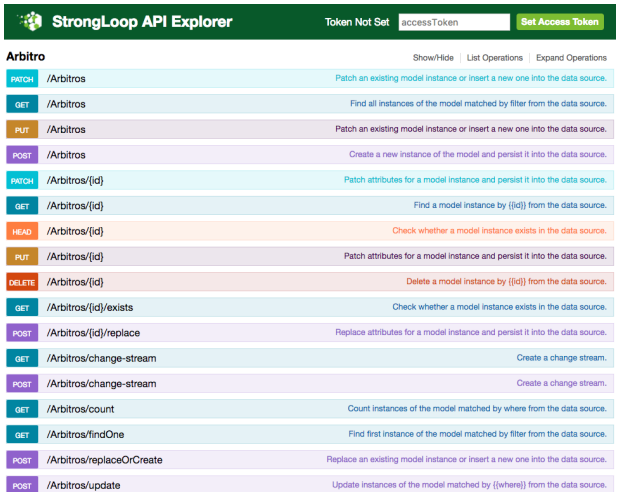


Figura 18: Exemplo dos pedidos HTTP possíveis para cada modelo.

7.2 CLIENTE

O *AngularJS* é uma *framework* de *javascript* que tem como objetivo a construção de aplicações *web* dinâmicas para o lado do cliente baseando-se no padrão MVC. Esta apresenta a possibilidade de acrescentar à sintaxe do HTML atributos, tornando as aplicações mais reativas às ações do utilizador, evitando que o utilizador replique muito do código através de conceitos como o *Data Binding*, que consiste em complementar as partes de interface gráfica da aplicação com os componentes de *javascript* (ocorre uma sincronização automática entre modelos e *views*), e como a *Dependency Injection* que, ao invés de criar dependências nas classes, injeta nas mesmas o que estas necessitam durante a execução da aplicação [GS13]. Para além disto, o *AngularJS* é uma linguagem compilável em todos os *browsers* atuais, garantindo assim flexibilidade neste aspeto.

Com a conjugação do servidor REST em *NodeJS* e o lado cliente em *AngularJS* conseguimos obter, de forma simples e fácil de manter, uma SPA. Esta consiste numa página que durante o seu uso não é mais recarregada totalmente pelo *browser*, proporcionando ao utilizador uma experiência de utilização mais fluída e estável. Ao mesmo tempo, esta arquitetura liberta o servidor de tarefas de renderização de HTML [MP13].

7.2.1 *Requisitos necessários*

Após a instalação do NPM ter sido realizada, como recomendado em 7.1.1, o único comando que ter-se-á de executar para que a instalação do *AngularJS* na máquina seja realizada com sucesso, é o seguinte:

```
npm install angular
```

Assim, não será necessária a instalação adicional de qualquer outro componente para que a geração de aplicações *web* seja conseguida.

7.2.2 *Visão geral das aplicações cliente*

A *framework* utilizada para a geração de aplicações *web* tem algumas características que nos permitem criar aplicações bem estruturadas e organizadas, sendo que foi com este intuito que algumas delas foram aplicadas neste trabalho. Para se perceber como ficam organizadas as aplicações geradas é necessário, primeiramente, perceber como funcionam e quais são os conceitos inerentes ao *AngularJS* implementados, destancando-se os seguintes:

- *Modules* - permitem-nos separar a camada lógica, de serviços, etc, mantendo o código limpo, claro e organizado.
- *Sevices* - São métodos *javascript* que são responsáveis por responder a uma tarefa específica, apenas, contribuindo para a modulação e organização da aplicação.
- *Custom Directives* - São diretivas de *AngularJS* que nos permitem controlar a renderização de elemetos *HTML* tornando o código mais modular, e possível de injetar em qualquer parte da aplicação, sem grandes dificuldades.

7.2.3 *Aplicações web geradas*

Inicialmente, aquando da criação e posterior geração do servidor falada em 7.1, a diretoria *client* encontra-se vazia. Porém, após a geração das aplicações *web*, a mesma terá a seguinte arquitetura:

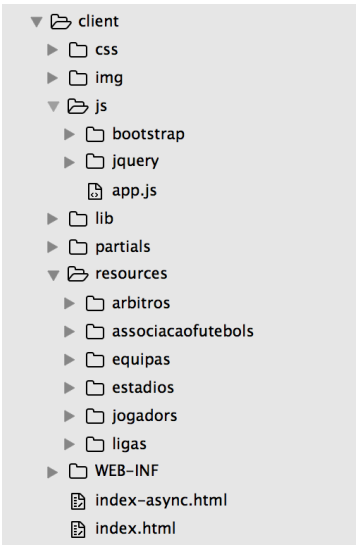


Figura 19: Estrutura da diretoria *client* após a geração de aplicações *web*.

Na aplicação gerada, destacam-se dois ficheiros *app.js*, que se encontra dentro da diretoria *js*, e *index.html*, onde, no primeiro, se declaram todos os módulos e se mapeiam as rotas da aplicação com os componentes HTML e os seus respetivos *controllers*. O segundo representa a página onde toda a aplicação será executada, dando corpo a todo o conceito de SPA. Este ficheiro, para além da referência ao *app.js*, inclui também todos os ficheiros de *scripts* auxiliares referidos em 7.2.2.

Para cada modelo constituinte do diagrama de classes, será criada, dentro da diretoria *resources*, uma pasta com o nome do mesmo, que no seu interior contemplará três ficheiros intitulados:

- *services.js* onde se mapeiam os *endpoints* relativos à API REST com os métodos da camada lógica, ficando ao encargo deste a interação entre o servidor *NodeJS* e a aplicação cliente[4].
- *directives.js* onde são definidas as *Custom Directives* da entidade.
- *controller.js* onde toda a lógica de negócio relativa à entidade está presente.

Além dos três ficheiros, a diretoria anteriormente mencionada, conterá também, uma pasta *partials* onde estarão presentes as *views* relativas ao modelo, que servem de interface gráfica e que são o ponto de interação para que o utilizador possa realizar as operações CRUD intrínsecas à entidade.

Para que se consiga interagir com as aplicações *web* via *browser*, ter-se-á que, através da linha de comandos e estando na diretoria do projeto, executar os seguintes comandos:

```
mysql.server start
```

Que faz com que o servidor *MySQL* arranque, sendo que o próximo e último comando, põe em execução o servidor: *NodeJS*:

```
node .
```

O resultado final será o apresentado na figura 20:

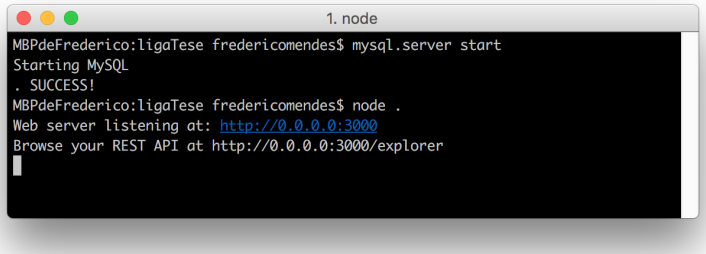
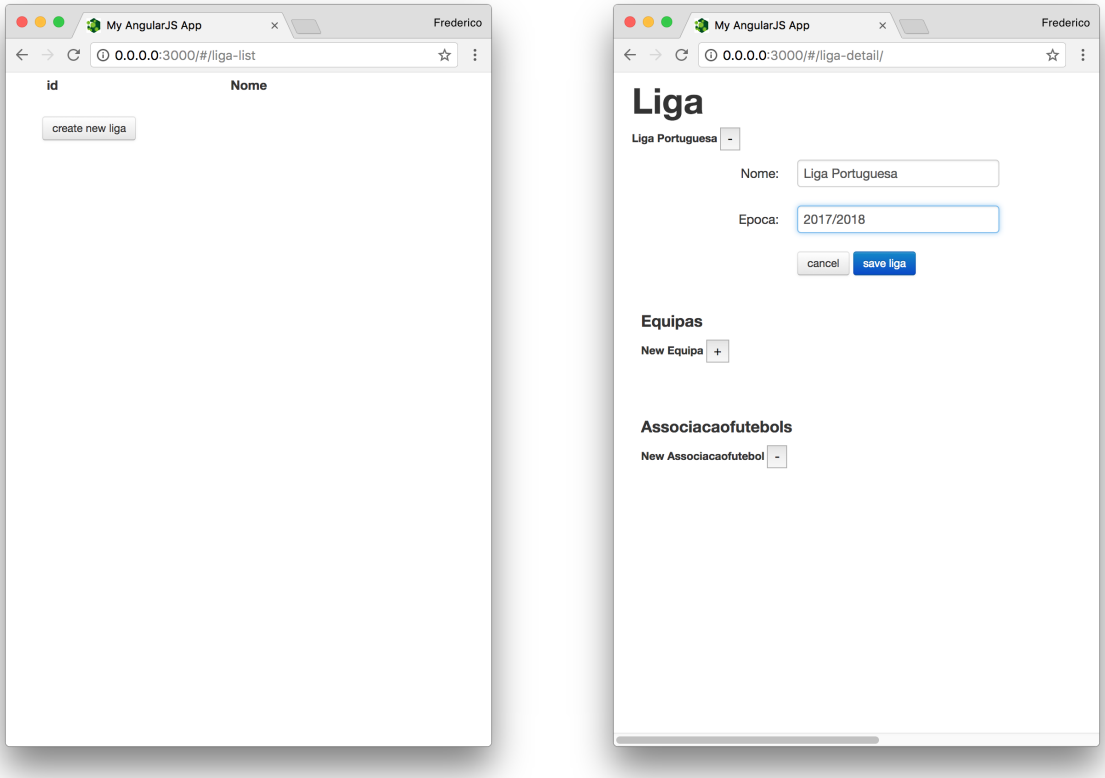


Figura 20: Servidor a executar

4 Esta interação utiliza um mecanismo baseado em *Promises* para lidar com as longas latências envolvidas.

Sendo que, temos então, o servidor e uma aplicação *web* aptos a responderem a pedidos consequentes da interação feita pelo utilizador.

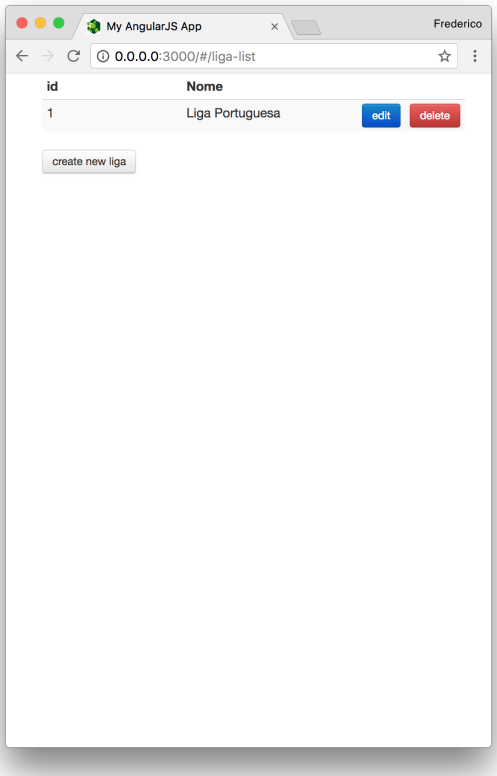
As figuras que serão apresentadas em seguida demonstram o aspeto típico que as aplicações *web* geradas terão. Mais uma vez, as ilustrações referem-se ao diagrama de classes apresentado na secção 5.2



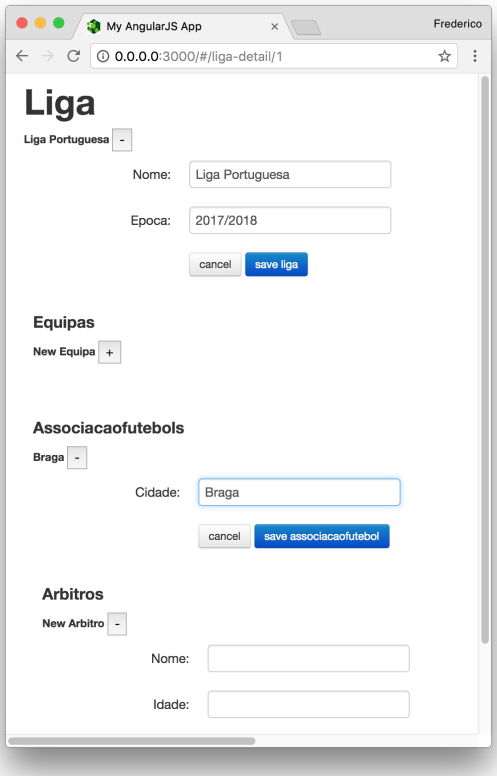
(a) Vista da aplicação sem dados inseridos

(b) Vista de inserção de uma Liga

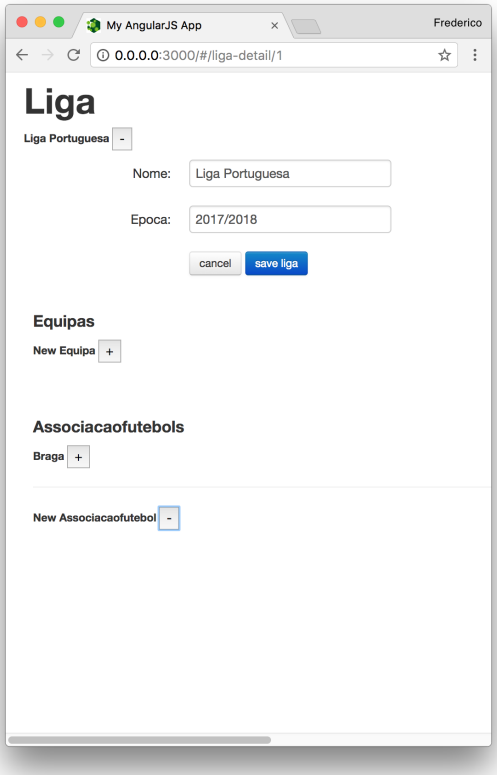
Figura 21: Estrutura das aplicações *web* geradas.



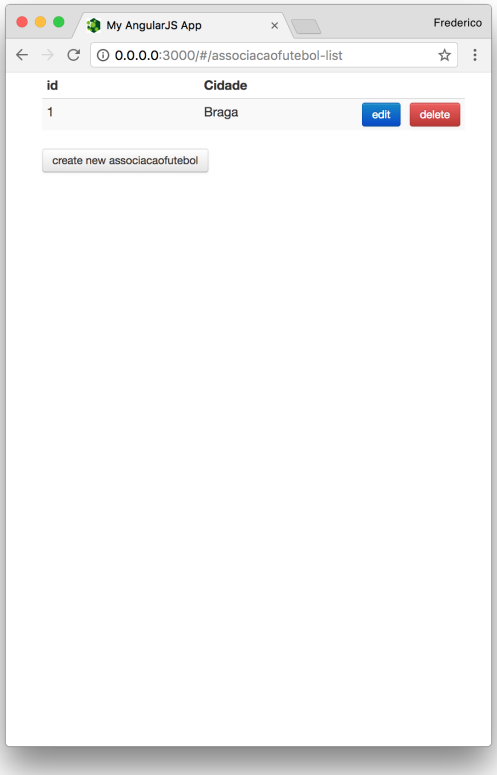
(c) Vista de apresentação de uma Liga, com possibilidade de editar e eliminar a mesma.



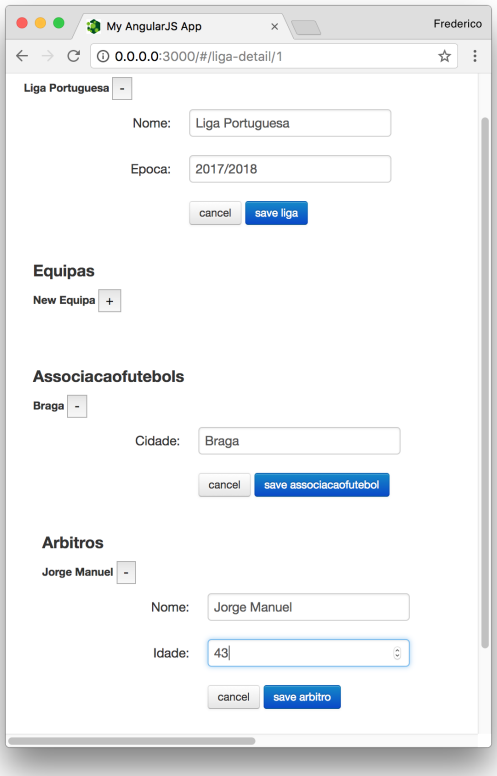
(d) Vista de inserção de uma associação de futebol, associando-a a uma liga.



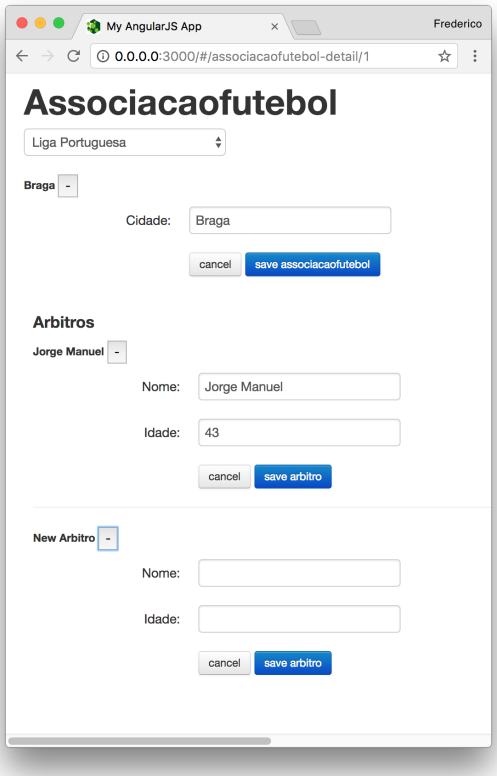
(e) Vista da apresentação de uma liga com a associação de futebol criada em 21d, associada a si mesma.



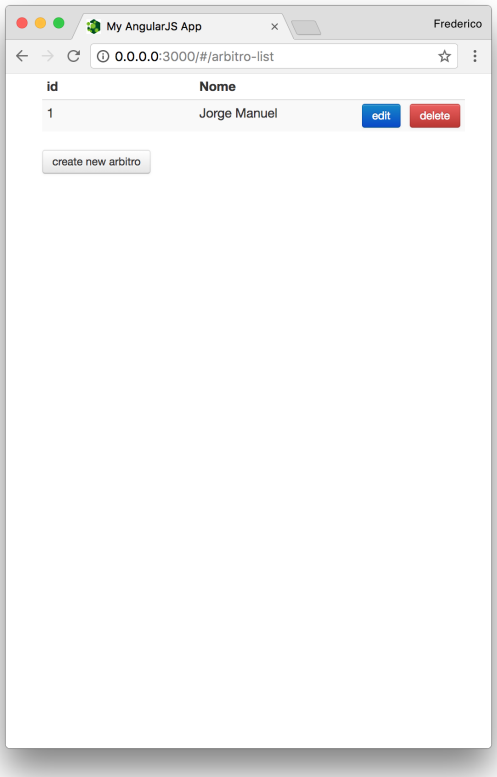
(f) Vista de apresentação da associação de futebol criada em 21d, com possibilidade de editar e eliminar a mesma.



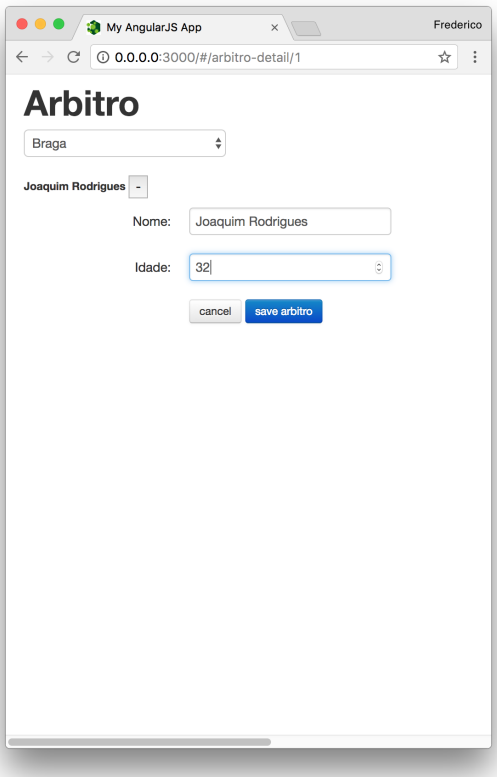
(g) Vista de inserção de um árbitro, associando-a a uma liga.



(h) Vista de apresentação da associação de futebol criada em 21d, com o árbitro criado em 21g associado a si mesma.



(i) Vista de apresentação do árbitro criado em 21g, com possibilidade de editar e eliminar a mesma.



(j) Vista de apresentação de um árbitro após a sua edição.

Novamente, como a lógica e apresentação relativa aos modelos são idênticas, não foram ilustradas todas as possibilidades de operações para o conjunto de modelos representados no diagrama de classes em 6. No entanto, é percetível e fica patente como funciona e como dever ser feita a interação do utilizador com as aplicações *web* geradas.

8

APLICAÇÃO GERADORA DE APLICAÇÕES CRUD

8.1 CONTEXTUALIZAÇÃO

JavaFX é uma linguagem baseada em *Java* que nos permite conjugar a construção da camada lógica referente à aplicação, com a parte gráfica que muitas vezes as mesmas necessitam. Devido à sua simplicidade e aliando à sua utilização ferrramentas como o *SceneBuilder*, que contém componentes de interface gráficas pré-definidos sendo necessário apenas clicar e arrastar os mesmos para se criar o cenário desejado, permite-nos construir rapidamente aplicações *desktop* com ambiente gráfico para uma melhor interação com o utilizador final das mesmas [CCB09]. Esta linguagem é multiplataforma, ou seja, pode executar em diversos dispositivos, quer via *desktop*, quer via *web*, sendo que é vista como uma substituta do conhecido *Java Swing*.

8.2 REQUISITOS NECESSÁRIOS

Com vista ao bom funcionamento da aplicação que gera aplicações CRUD, é necessária a instalação do próprio *Java* sendo que se pode, por exemplo, recorrer à instalação descarregando do próprio site[1] o pacote, prosseguindo posteriormente, à sua instalação. Para além da instalação do *Java*, será também necessárias a instalação e descarga do SDK do *JavaFX*, podendo esta ser efetuada, mais uma vez, através do site[2]. Por fim, poderá, de igual forma ser essencial, dependendo do sistema operativo que o utilizador esteja a manipular, a instalação de alguns comandos *bash* como é o caso de, por exemplo, o comando *sed* que é bastante utilizado na geração das aplicações CRUD.

1 https://java.com/pt$_B$R/*download*/
2 http://www.oracle.com/technetwork/java/javase/downloads/index.html

8.3 VISÃO GERAL DA APLICAÇÃO GERADORA

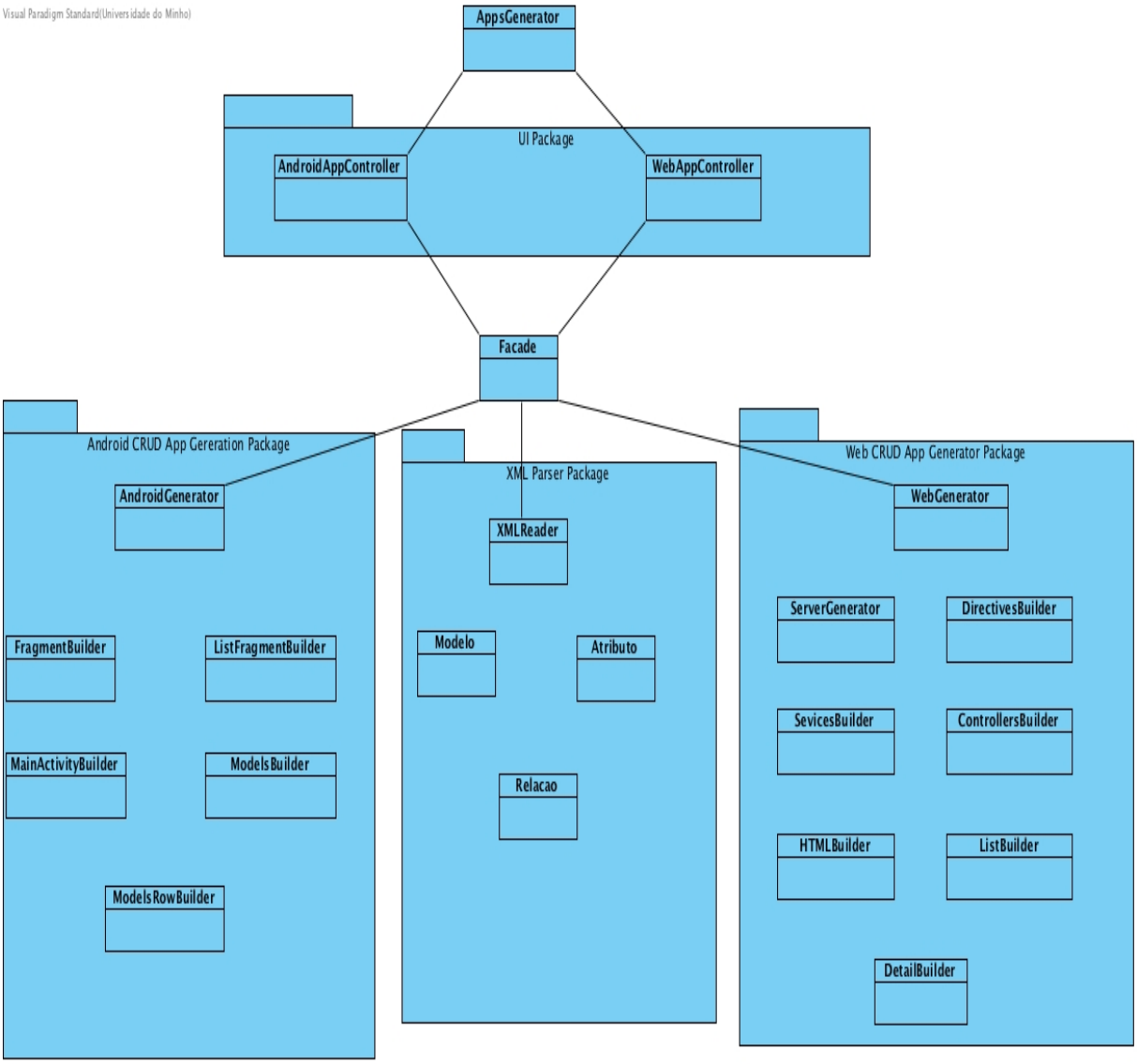


Figura 22: Visão global da aplicação geradora de aplicações CRUD

Como se pode verificar pela imagem anterior, a aplicação, construída no trabalho relativo à presente dissertação, é constituída por diversas classes estando as mesmas distribuídas por *packages*.

A classe do topo da hierarquia, *AppsGenerator*, é a classe principal da aplicação uma vez que contém a função *main* que tem como tarefa possibilitar o arranque da mesma. Posteriormente, surge o primeiro *package*, *UIPackage*, que se refere à interface gráfica, isto é, contém os *controllers* e as *views* que fazem com que a interação que o utilizador promove seja

respondida de forma eficiente. Para que a lógica de negócio do programa seja encapsulada e se reduza a sua complexidade, foi utilizado o *desgin pattern facade*. A sua existência deve se ao facto de providenciar uma interface mais global para um subsistema que se revela um pouco mais complexo, tornando a compreensão e uso da aplicação, a este nível, mais trivial. Entrando agora no subsistema previamente falado, este pode dividir-se em 3 *packages* principais. O *XMLParserPackage*, como o próprio nome indica, tem como função o tratamento do ficheiro XML dado como *input*, tratá-lo e daí popular classes que criam as relações entre os modelos constituintes do diagrama de classes. As funções destinadas ao *AndroidCRUDGeneratorPackage*, como ao *WebCRUDGeneratorPackage* são semelhantes, uma vez que estes têm como propósito a criação de aplicações CRUD editando, para isso, os ficheiros necessários para que este objetivo seja alcançado. Dentro do *package* referente às aplicações *web*, está inserida, também, a classe responsável pela construção do servidor *NodeJS*.

De notar que ao longo da elaboração da aplicação *JavaFX* foi feito um esforço para que as regras de boa programação fossem mantidas, zelando sempre pela construção de uma aplicação que facilitasse quer a sua leitura, quer a sua manutenção.

8.4 APLICAÇÕES GERADORA

Nesta secção, irá ser exposta a aplicação que foi concebida, no âmbito deste trabalho, para que o objetivo principal do mesmo fosse atingido. Posto isto, o utilizador, quer para a geração de aplicações *android*, quer para as aplicações *crud web*, terá apenas que preencher os campos inerentes à geração das aplicações, apresentados nas figuras 23 e 25, e a partir daí obtém o que é pretendido, aplicações robustas e funcionais.

8.4.1 *Geração de aplicações CRUD - android*

Como se pode visualizar na figura 23, o utilizador terá que que preencher 4 campos para que seja gerada uma aplicação *crud android*. Primeiramente, terá de escolher o ficheiro XML que resulta da exportação do diagrama de classes como explicado em 5.2, tendo depois de escolher as diretorias onde quer guardar o projeto e onde se encontra o *android* SDK na sua máquina. Por fim, o utilizador terá de dar um nome ao projeto que irá ser gerado.

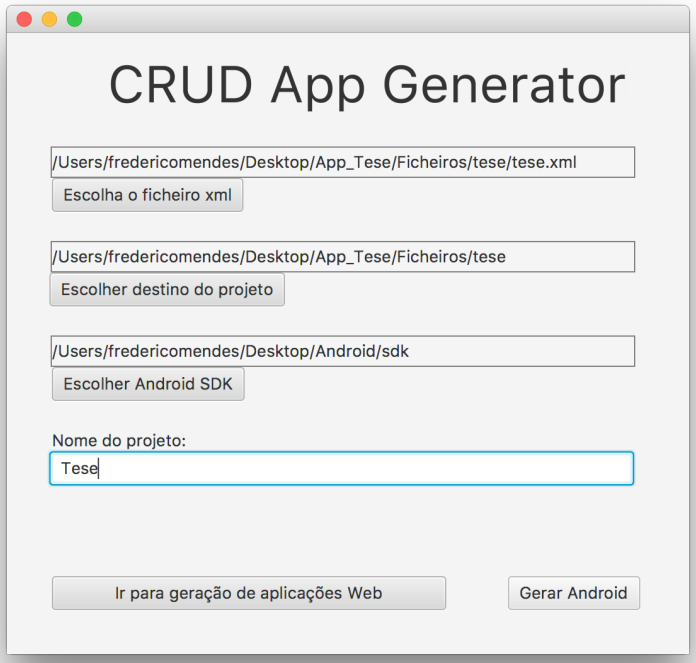


Figura 23: Aplicação geradora - *android*

Após os passos enunciados anteriormente, surgirá a mensagem que está representada na figura 24:

Figura 24: Confirmação da aplicação gerada - *android*

O resultado final, será uma aplicação CRUD *android* funcional e que se poderá encontrar na diretoria homónima ao nome escolhido no quarto passo referido anteriormente.

8.4.2 *Geração de aplicações CRUD - web*

No que toca às aplicações *web*, os dois primeiros passos são análogos aos previamente referidos em 8.4.1. No entanto, e como se pode verificar na figura 25, ter-se-á também de preencher o nome que se quer dar à base de dados, bem como o *usernome* e *password* que se utiliza para a conexão ao servidor *MySQL*. Relativamente ao nome da base dados, o mesmo tem que ser impreterivelmente igual ao nome dado no *schema* criado na secção 7.1.2, para que a geração de aplicações *web* seja bem sucedida.

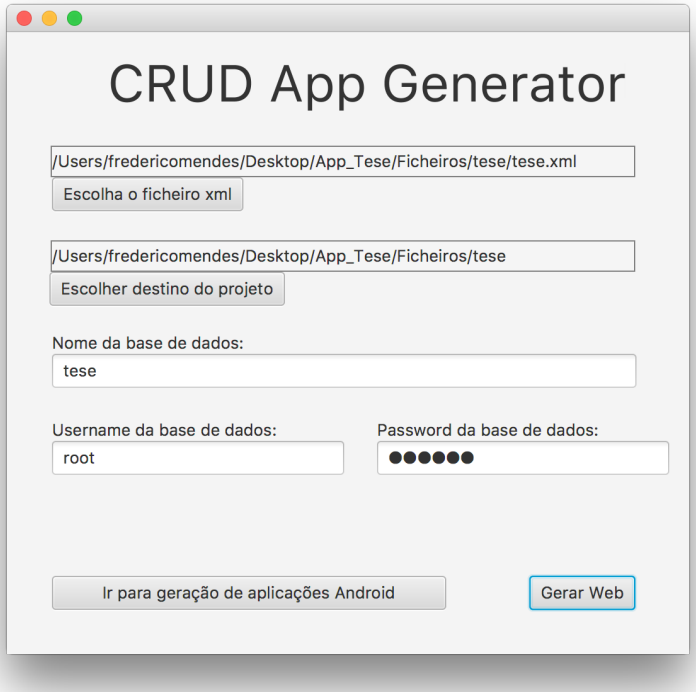


Figura 25: Aplicação geradora - *web*

Após os passos enunciados anteriormente, surgirá a mensagem que está representada na figura 26:

Figura 26: Confirmação da aplicação gerada - *web*

O resultado final será uma aplicação CRUD *web* funcional e que se poderá encontrar na diretoria que se escolheu como destino do projeto.

9

CONCLUSÃO E TRABALHO FUTURO

9.1 CONCLUSÃO

O intuito desta dissertação era facultar uma maneira rápida e eficaz de criação de aplicações, envolvendo o mínimo de escrita de código possível por parte dos programadores, aproximando assim duas fases importantes do desenvolvimento de *software*, a fase de *design* e a fase de implementação.

Foi, então, criada uma aplicação capaz de interpretar e processar os diagramas de classe exportados em XML, tendo esta a capacidade de analisar e construir as relações entre modelos, assim como os atributos dos mesmos. A interpretação dos diagramas é feita através da utilização de uma biblioteca de *Java* que facilitou bastante a interpretação do XML. No entanto, há outros formatos que poderiam ter sido considerados para o processamento dos diagramas como é o caso do XMI, que apesar de ser mais utilizado e aconselhado no que toca a geração de código a partir de UML, acarreta a desvantagem de ter muito mais conteúdo a ser analisado, e que, muito do mesmo, não é necessário para que o objetivo do presente trabalho seja comprometido. O facto de, também, não se ter utilizado as transformações enunciadas em 3.2, prende-se com questões de simplicidade, visto que este projeto não é de grandes dimensões e não exige um grande processamento de informação e dados, optou-se pela solução previamente referida.

Para as aplicações *android*, foi tomada a decisão de se adaptar e integrar um trabalho elaborado também no âmbito de uma dissertação de mestrado Desenvolvimento de aplicações *android* com *Scala* e *SBT*. Esta escolha permitiu que muito do trabalho que era preciso ser realizado ficasse adiantado, como por exemplo, a construção de tabelas para cada elemento constituinte do diagrama de classes, bem como as interfaces para que o utilizador conseguisse interagir de forma simples com a aplicação. Posto isto, ficou a cargo da presente dissertação a implementação dos métodos necessários para que as operações CRUD fossem concretizadas. Porém, a integração desta ferramenta fez com que a flexibilidade e a capacidade de decisão em determinados aspetos fosse bastante reduzida, como são exemplos a seleção do motor de *queries* à base de dados ou a escolha da linguagem de programação para as aplicações *android*.

Todavia, no que toca à geração de aplicações *web*, nenhum trabalho ou aplicação existente foi integrado, tendo estas sido feitas de raíz. Com isto, pôde-se utilizar as tecnologias e arquiteturas que foram consideradas as mais apropriadas para que o principal objetivo fosse atingindo, obtendo mais controlo sobre como as coisas foram implementadas e estruturadas. Foram então criadas aplicações que têm um servidor que implementa uma API REST para que este comunique com as aplicações cliente. Esta solução acarreta algumas consequências pois este tipo de arquitetura e escolha de linguagens pode não ir de encontro à solução pretendida pelo utilizador.

A aplicação geradora de aplicações CRUD foi desenvolvida apenas num contexto de *desktop*, o que facilitou bastante a celeridade com que a mesma foi desenvolvida, mas que por si só, é um entrave no que toca à disponibilização da mesma à comunidade. Se fosse elaborada e alojada como uma ferramenta *online*, esta desvantagem não se verificaria, podendo também, facilitar o melhoramento do *design* gráfico da aplicação.

No final, pode-se concluir que a meta essencial deste trabalho, no cômputo geral, foi atingida com sucesso, tendo sido desenvolvida uma aplicação com uma interface intuitiva e de fácil utilização que permite ao utilizador, após modelar o seu problema, exportar os diagramas de classe em formato XML e obter aplicações quer *web*, quer *android*, que fornecem operações CRUD para cada entidade constituinte do referido diagrama.

9.2 TRABALHO FUTURO

Uma vez que esta é uma área onde há ainda muito por explorar, há inúmeras vertentes que poderão ser trabalhadas.

Desde logo, no trabalho realizado na presente dissertação, o passo mais lógico a ser tomado, de seguida, seria implementar as operações CRUD para modelos que possuam relações N para N, uma vez que, no caso de duas entidades apresentarem a relação referida, os requisitos inerentes à mesma só estão garantidos, única e exclusivamente ao nível da base de dados, não acontecendo o mesmo para a camada lógica.

Tendo sido apenas explorados diagramas de classe neste projeto, outra possibilidade seria a integração de outro tipo de diagramas que o UML faculta, sendo exemplo os diagramas de sequência, podendo-se gerar métodos especificados através dos mesmos, ou diagramas de estado, que ajudam a definir concretamente a sequência de execução das aplicações geradas.

Visto que foi criado um servidor REST para as aplicações *web*, e como nas aplicações *android* os dados das aplicações geradas são guardados numa base de dados local, a integração das aplicações *android* com o servidor previamente falado é vista como um possível acréscimo ao trabalho realizado.

Como abordado nos capítulos 3 e 4 existem diversas formas quer para se realizarem as transformações M2T, quer para a implementação das aplicações *web* e *android* concretizadas neste trabalho. Posto isto, as linguagens e arquiteturas utilizadas nas aplicações geradas e na interpretação dos modelos exportados, podem ser alargadas contribuindo para que a aplicação seja mais universal e satisfaça mais requisitos e necessidades que os utilizadores tenham.

Todos os pontos anteriormente enumerados são vistos como mais valias e complementos ao trabalho realizado na presente dissertação.

BIBLIOGRAFIA

[Acc] MDA Acceleo. Generator home.

[AP08] AA Abdullatif and Rob Pooley. A computer assisted state marking method for extracting performance models from design models. *International journal of simulation*, 8(3):36–46, 2008.

[Arg05] L Argoum. Argouml tool. 2005.

[BB+07] Matthias Bohlen, C Brandon, et al. Andromda. *Archived from the original on*, 27, 2007.

[BBG+05] Sami Beydeda, Matthias Book, Volker Gruhn, et al. *Model-driven software development*, volume 15. Springer, 2005.

[BCD+03] Mariano Belaunde, Cory Casanave, Desmond DSouza, Keith Duddy, William El Kaim, Alan Kennedy, William Frank, David Frankel, Randall Hauch, Stan Hendryx, et al. Mda guide version 1.0. 2003.

[BEAM16] Hanane BENOUDA, Redouane ESSBAI, Mostafa AZIZI, and Mimoun MOUSSAOUI. Modeling and code generation of android applications using acceleo. *International Journal of Software Engineering and Its Applications*, 10(3):83–94, 2016.

[Ben04] M Benoit. Working xml: Uml, xmi, and code generation, part 2. *World Wide Web electronic publication: http://www. ibm. com/developerworks/xml/library/xwxxm24*, 2004.

[BKM99] Hubert Baumeister, Nora Koch, and Luis Mandel. Towards a uml extension for hypermedia design. In *International Conference on the Unified Modeling Language*, pages 614–629. Springer, 1999.

[C+99] James Clark et al. Xsl transformations (xslt). *World Wide Web Consortium (W3C). URL http://www. w3. org/TR/xslt*, page 103, 1999.

[C+10] Eclipse Consortium et al. Java emitter templates (jet), 2010.

[CCB09] Jim Clarke, Jim Connors, and Eric J Bruno. *JavaFX: developing rich Internet applications*. Pearson Education, 2009.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo Bongio. Web modeling language (webml): a modeling language for designing web sites. *Computer Networks*, 33(1):137–157, 2000.

[CH03] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17. USA, 2003.

[Cia] Paolo Ciancarini. Metamodelling and the model driven architecture. Disponível em https://www.slideshare.net/kronat/5-architetture-software-metamodelling-and-the-model-driven-architecture, year = 2011.

[CLH+] Juan José Cadavid, David Lopez, Jesús Andrés Hincapié, Carlos Andrés, and Juan Bernardo Quintero Ospina. Generating web applications from uml domain models: A model-driven approach using profiles and metamodels.

[EFH+08] Sven Efftinge, Peter Friese, Arno Haase, Clemens Kadura, Bernd Kolb, Dieter Moroff, Karsten Thoms, and Markus Völter. openarchitectureware user guide. version 4.3. *openArchitectureWare.¡ http://www. openarchitectureware. org/pub/documentation/4.3*, 1, 2008.

[Fon14] Luís Miguel Guimarães Pimentel Fonseca. Exploring rapid application development for android with scala and sbt. 2014.

[G+00] OBJECT MANAGEMENT GROUP et al. Object management group home page. *Disponível por WWW em: http://www. omg. org/(7 mar. 2000)*, 2000.

[GMVD08] Vahid Gharavi, Ali Mesbah, and Arie Van Deursen. Modelling and generating ajax applications: A model-driven approach. Technical report, Delft University of Technology, Software Engineering Research Group, 2008.

[GP12] Rohit Ghatol and Yogesh Patel. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Apress, 2012.

[GS13] Brad Green and Shyam Seshadri. *AngularJS*. "O'Reilly Media, Inc.", 2013.

[Hau09] Stephan Haugsrud. Developing android applications with arctis. 2009.

[HK04] David Harel and Hillel Kugler. The rhapsody semantics of statecharts (or, on the executable core of the uml). In *Integration of Software Specification Techniques for Applications in Engineering*, pages 325–354. Springer, 2004.

[Jai] Shubham Jain. Androidarchitecture. Disponível em http://shubhamjain.space/android/android-architecture/, year = 2016.

[JBR+99] Ivar Jacobson, Grady Booch, James Rumbaugh, James Rumbaugh, and Grady Booch. *The unified software development process*, volume 1. Addison-wesley Reading, 1999.

[KG] Mr Sharad R Kurhade and Mr Nayan D Gite. Androidanti-malware analysis.

[KK02] Nora Koch and Andreas Kraus. The expressive power of uml-based web engineering. In *Second International Workshop on Web-oriented Software Technology (IWWOST02)*, volume 16. CYTED, 2002.

[KK03] Nora Koch and Andreas Kraus. Towards a common metamodel for the development of web applications. In *International Conference on Web Engineering*, pages 497–506. Springer, 2003.

[KKMZ03] Alexander Knapp, Nora Koch, Flavia Moser, and Gefei Zhang. Argouwe: A case tool for web applications. In *EMSISE Workshop*, 2003.

[Lif] Android Activity Lifecycle. Android activity lifecycle.

[LTE+09] Agnes Lanusse, Yann Tanguy, Huascar Espinoza, Chokri Mraidha, Sebastien Gerard, Patrick Tessier, Remi Schnekenburger, Hubert Dubois, and François Terrier. Papyrus uml: an open source toolset for mda. In *Proc. of the Fifth European Conference on Model-Driven Architecture Foundations and Applications (ECMDA-FA 2009)*, pages 1–4. Citeseer, 2009.

[Mas11] Mark Masse. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. "O'Reilly Media, Inc.", 2011.

[Mel] Mauro Miguel Melo. Openxava manual.

[MM+03] Joaquin Miller, Jishnu Mukerji, et al. Mda guide version 1.0. 1, 2003.

[MOF08] OMG MOFM2T. Omg mof model to text transformation language (omg mofm2t) version 1.0. *Object Management Group. http://www. omg. org/spec/-MOFM2T/1.0*, 2008.

[MP13] Michael S Mikowski and Josh C Powell. Single page web applications. *B and W*, 2013.

[MTSP05] Jose-Norberto Mazón, Juan Trujillo, Manuel Serrano, and Mario Piattini. Applying mda to the development of data warehouses, 01 2005.

[N+05] Iftikhar Azim Niaz et al. Automatic code generation from uml class and statechart diagrams. *Graduate School of Systems and Information Engineering., University of Tsukuba, Ph. D. Thesis*, 2005.

[OAC+04] Martin Odersky, Philippe Altherr, Vincent Cremet, Burak Emir, Sebastian Maneth, Stéphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, and Matthias Zenger. An overview of the scala programming language. Technical report, 2004.

[OVD08] Ernst Oberortner, Martin Vasko, and Schahram Dustdar. Towards modeling role-based pageflow definitions within web applications. In *Proc. of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008)*, volume 389, pages 1–15, 2008.

[Pan11] Javier Paniza. Learn openxava by example. 2011.

[Par13] Visual Paradigm. Visual paradigm for uml. *Visual Paradigm for UML-UML tool for software application development*, 2013.

[RdS14] André Ribeiro and Alberto Rodrigues da Silva. Development of mobile applications using a model-driven software development approach. 2014.

[RFBLO01] Dirk Riehle, Steven Fraleigh, Dirk Bucka-Lassen, and Nosa Omorogbe. The architecture of a uml virtual machine. *ACM SIGPLAN Notices*, 36(11):327–341, 2001.

[S+04] Kendall Scott et al. *Fast track UML 2.0*, volume 11. Springer, 2004.

[SBP09] David Steinberg, Frank Budinsky, and Marcelo Paternostro. Merks, emf: Eclipse modeling framework 2.0, 2009.

[SVC06] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2006.

[SVEH12] Thomas Stahl, Markus Völter, Sven Efftinge, and Arno Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt. verlag, 2012.

[Tei12] Pedro Teixeira. *Professional Node. js: Building Javascript based scalable software*. John Wiley & Sons, 2012.

[Tra16] Worldwide Quarterly Mobile Phone Tracker. Smartphone vendor market share, 2016 q3. *International Data Corporation*, 2016.

[UN09] Muhammad Usman and Aamer Nadeem. Automatic generation of java code from uml diagrams using ujector. *International Journal of Software Engineering and Its Applications*, 3(2):21–37, 2009.