INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Framework de Agilização do Processo de criação de Aplicações Multiplataforma através da Geração Automática de Código

ANDRÉ FILIPE MOREIRA DA SILVA
Outubro de 2016

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

Ambiente de Integração de Serviços Web

Framework de Agilização do Processo de criação de Aplicações Multiplataforma através da Geração Automática de Código

André Filipe Moreira da Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática na Área de Especialização em Sistemas Gráficos e Multimédia

Orientador: Paulo Alexandre Maio

Porto, Outubro 2016

Dedicatória

A todos aqueles que não se conformam com a realidade e procuram um futuro melhor. Àqueles que tem a vontade de mudar o mundo porque de facto são esses que o farão.

Resumo

Este documento apresenta uma proposta de solução para agilização do processo de criação de software multiplataforma. Num mercado de constante atualização e em crescimento, há cada vez maior foco na experiência do utilizador final e em entregar software de qualidade rapidamente. Tempo é um recuso limitado que, quando poupado, permite também poupar outros recursos semelhantes como orçamento e mão de obra. Então como criar software multiplataforma rapidamente?

A solução descrita consiste na criação de um ambiente de desenvolvimento de aplicações baseado nos princípios e padrões de engenharia capaz de gerar código fonte automaticamente utilizando linguagens e bibliotecas pré-configuradas em modelos geradores.

Após uma análise de estado de arte do mercado tecnológico atual foram apuradas algumas tecnologias que procuram responder à mesma pergunta e tecnologias que de alguma forma permitem uma agilização de parte do processo de desenvolvimento. Os resultados desta análise serviram assim como base inspiracional para a criação desta solução.

Baseando as suas ações em criar atores, criar entidades de negócio e criar casos de uso, a solução desenvolvida faz uso dos conceitos mais simples da gestão de negócio de um sistema informático e potencia as funcionalidades definidas nas linguagens configuradas através da geração de código *frontend*, *backend*, base de dados e de integração.

Desta forma procura-se adicionar valor a empresas de produção de software, programadores e a clientes finais utilizadores do software produzido, que, devido à célere produção de código, poderão ter um produto em mãos mais rapidamente e ainda assim sólido e estável segundo padrões de engenharia.

A implementação foi efetuada segundo algumas das tecnologias mais recentes como AngularJS 2.0 e Node.js, procurando assim preparação para o futuro. Após testada e validada como produto final procurar-se-á investimento externo para continuação de desenvolvimento e possível inclusão de teses MEI ISEP para desenvolvimento de módulos integrantes.

Palavras-chave: velocidade, aceleração, desenvolvimento, multiplataforma, geração automática de código, lógica de negócio

Abstract

This document presents a proposed solution for streamlining the creation of cross-platform software. In a constantly updated and growing market, there is increasing focus on the end user experience and deliver quality software quickly. Time is a limited resource that when saved, allows also to save other similar resources as budget and manpower. So how to create cross-platform software quickly?

The described solution consists in creating an application development environment based on the principles and engineering standards, capable of generating source code automaticallyusing pre-configured languages and libraries in models for code generation.

After a state analysis of the current technology market there were pointed some technologies that seek to answer the same question and technologies that somehow allow for a streamlining of the development process. The results of this analysis served as the inspirational basis for creating this solution.

Basing its actions in creating actors, creating business entities and creating use cases, the developed solution makes use of the simplest concepts of business management of a computer system and powers functionalities in defined coding languages by generating frontend, backend, databases and integration code.

In this way we seek to add value to software production companies, developers and end users of the software produced, that due to the rapid code production, may have a product in hand faster and still solid and stable by engineering standards.

The implementation was carried out according to some of the latest technology like AngularJS 2.0 and Node.js, looking for preparation to the future. After tested and validated as final product, external investment will be searched and MEI ISEP thesis may be included to develop new modules.

Keywords: speed, acceleration, development, cross-platform, automatic code generation, business logic

Agradecimentos

À minha namorada, Glória Reis, pela sua fé na minha pessoa, pelo companheirismo, pela paixão e carinho, pelo seu apoio e compreensão e pela sua ajuda nos momentos mais difíceis e de maior cansaço.

Ao meu orientador, professor Paulo Alexandre Maio, que durante estes seis anos esteve sempre a meu lado providenciando a melhor orientação que um aluno pode ter. Toda a sua compreensão e fé no meu trabalho fizeram de mim o profissional que sou hoje.

À minha família, pela fé, pelo carinho e pela compreensão. Por todo o apoio e orgulho que sempre tiveram por mim.

A Leandro Gomes, Stefan Gomes e Jorge Correia, meus colegas de curso, por todos os momentos, por todo o trabalho de equipa, por todas as vitórias e derrotas que fizeram de nós os profissionais que hoje somos.

E por fim, mas não em último, a todos os colaboradores da Deloitte Digital Portugal, pela fé na minha pessoa desde do primeiro dia, pela oportunidade e por tudo aquilo que me providenciaram até hoje que me permitiu o crescimento profissional que diariamente atinjo.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

RAD	*Rapid Application Development*
SQL	*Structured Query Language*
API	*Application Programming Interface*
HTML	*HyperText Markup Language*
CSS	*Cascading Style Sheets*
CRM	*Customer Relationship Management*
ERP	*Enterprise Resource Planning*
IDE	*Integrated Development Environment*
MVC	*Model-View-Controller*
CRUD	*Create, Read, Update and Delete*
FURPS	*Functionality*, *Usability*, *Reliability*, *Performance*, *Supportability*
RUP	*Rational Unified Process*

1 Introdução

O presente capítulo tem como objetivo apresentar o assunto desta tese, explicando o seu contexto, o problema que a motiva e a abordagem para a resolução do mesmo.

1.1 Contexto

O presente documento insere-se no contexto da Tese do Mestrado em Engenharia Informática do Instituto Superior de Engenharia do Porto, no ramo de Sistemas Gráficos e Multimédia. Esta tese é assim elaborada por André Filipe Moreira da Silva, licenciado em Engenharia Informática pela mesma escola e com o número mecanográfico 1100540. O objetivo deste documento é apresentar uma solução informática para a resolução de um problema atual recorrendo a toda a experiência profissional e aos conteúdos lecionados durante o curso.

1.2 Problema

Desenvolver software, tal como qualquer outra atividade de produção, necessita de matérias-primas e recursos para criar os seus produtos finais. Produzir software é um processo de investigação, definição, implementação, teste e manutenção, e todas estas fases tem em comum as matérias primas, tempo e pessoas.

Tipicamente quando uma necessidade de software surge é realizada uma análise daquilo que é necessário fazer e uma definição do que será feito, ou seja, modelação de negócio, especificação de requisitos e design. Normalmente essa definição é posteriormente traduzida numa determinada linguagem de código ou tecnologia e testada criando assim um produto de

software que será entregue ao cliente final. Este processo é definido como o Processo RUP – "Rational Unified Process" e é ilustrado na Figura 1.

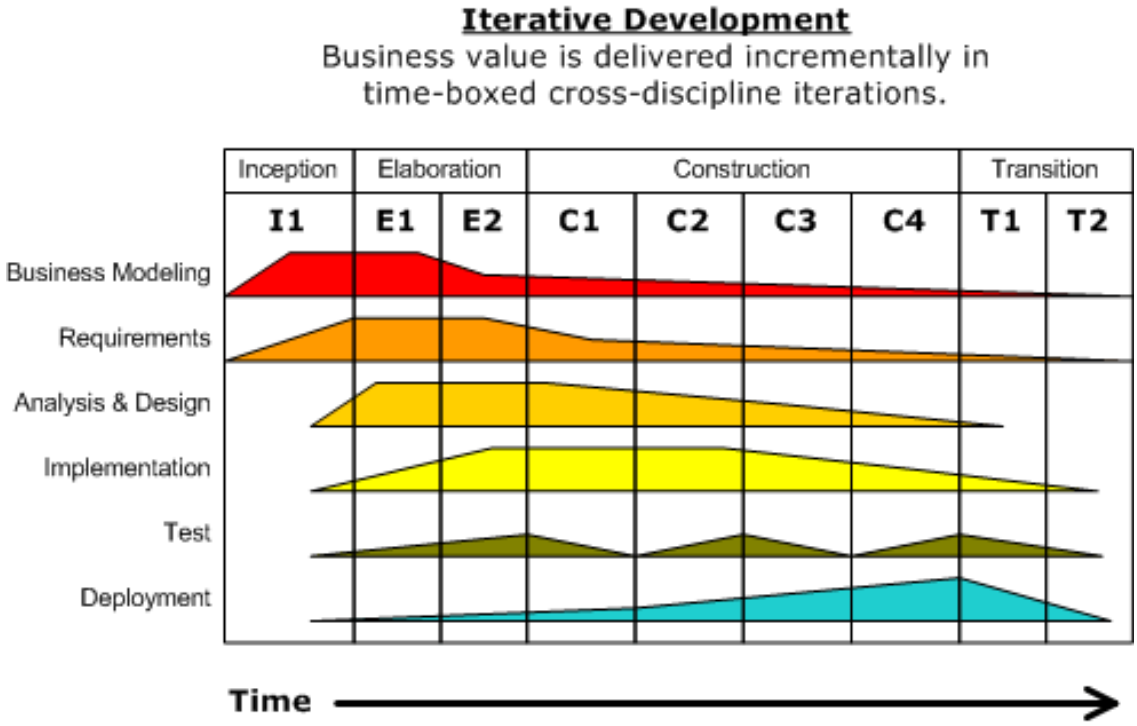


Figura 1 - Fases e Disciplinas do "RUP - Rational Unified Process"

A partir deste momento o produto de software criado irá crescer, modificar-se, sofrer alterações de âmbito, de requisitos, integrar-se com outros softwares, outras tecnologias, incluir outras interfaces gráficas ou mesmo ser abandonado por desinteresse ou invalidade. Perante estes acontecimentos há lugar a um reinicio do processo, com revalidação da definição e novas implementações. Normalmente estes processos ocorrem durante longos períodos de tempo e por múltiplas e diferentes pessoas.

É exemplo disto o processo "Scrum", considerado como metodologia ágil e demonstrado na Figura 2. Após a análise de negócio e especificação de requisitos e design são criadas listas de tarefas a executar. Essas tarefas vão sendo revistas para dar resposta a novas necessidades ou alterações de âmbito do negócio. A cada interação de implementação (designada por "Sprint") o software vai sendo alterado pela execução das tarefas planeadas para essa iteração. O objetivo é desenvolver software mais próximo das reais necessidades do cliente, mais rapidamente e com menos custos posteriores de alteração.

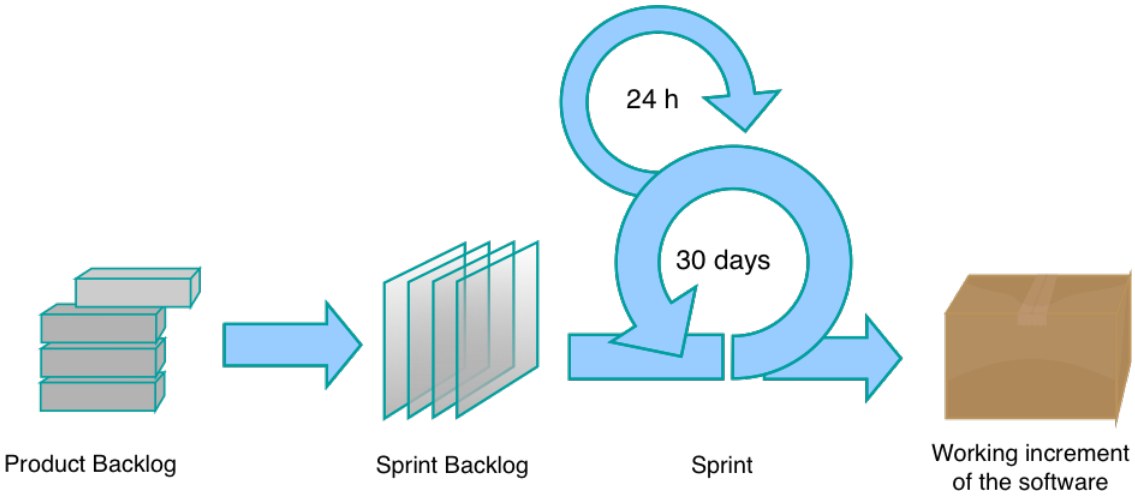


Figura 2 - Fases do Processo Scrum

Mas estes factos suscitam duas importantes questões: (i) a tecnologia evolui rapidamente e, portanto, um software fica facilmente desatualizado, podendo isto acontecer mesmo antes de estar terminado; (ii) diferentes pessoas tem diferentes abordagens de desenvolvimento o que poderá comprometer a qualidade do código e consequentemente a qualidade do software como peça.

Na conjetura atual onde o utilizador final é cada vez mais exigente, esta situação adquire uma complexidade maior. Desde do aparecimento do computador pessoal até aos mais recentes *smartphones, smartwatches*, eletrodomésticos e cockpits de automóveis digitais que o utilizador final sabe cada vez melhor aquilo que procura num software. A empresa InVision[1] descreve no seu documentário "*Design Disrruptors*"[2] a importância de criar experiências digitais, sendo mesmo essa a chave do sucesso para serviços como o Netflix, AirBnB, Uber, etc. À medida que a tecnologia vai estando cada vez mais perto do utilizador, há também um desejo desse utilizador em que a tecnologia esteja a seu lado em todos os momentos, seja no autocarro com um telefone ou no computador do escritório ou em casa na *smart tv*. Há assim uma necessidade de criar software multiplataforma, software como um serviço disponível em qualquer lugar da vida do utilizador quotidiano, focado na sua qualidade de vida e aceleração do seu ritmo de trabalho. E se o software representa uma ligação cada vez maior a todos os elementos do utilizador final a consequência natural será que um software se integre e comunique com

[1] https://www.invisionapp.com/
[2] Trailer em: https://www.youtube.com/watch?v=W4AViRgrgkU

outros softwares de forma a cobrir todas as dimensões de funcionalidades necessárias à criação da experiência de utilização que cada vez mais se espera entregar e usufruir.

Em suma, há cada vez maior necessidade de desenvolvimento de software, criação de mais e melhores experiências de utilização, integração de informação entre vários outros softwares e tudo isto num ritmo de mercado que evolui com uma rapidez difícil de acompanhar. As metodologias ágeis procuram resolver o problema de acompanhamento do ritmo de mercado mas revelam-se insuficientes perante as necessidades de acompanhamento da evolução tecnológica, desenvolvimento de software com melhores experiências de utilização e facilidade de integração e ainda manutenção da qualidade de código escrito. A responsabilidade das operações envolvidas nestes âmbitos em falta pode ser atribuída às ferramentas para desenvolvimento. As ferramentas existentes atualmente parecem não ser eficazes na resolução deste problema visto os seus objetivos não estarem focados nestas áreas.

A visão deste problema surge com base em três anos de experiência profissional, em que, participando em atividades de desenvolvimento em equipas com dimensões entre os 5 e os 40 programadores, em empresas com pequena e grande dimensão, apesar de se utilizarem metodologias ágeis, os problemas mantiveram-se. E assim, o problema que esta tese pretende dar reposta é: Como criar software multiplataforma com foco na experiência, de forma fácil e rápida, sem comprometer qualidade?

1.3 Abordagem

O problema destacado já existe há relativamente algum tempo e por esse facto já foram criadas algumas possíveis soluções para o mesmo. Um exemplo de tais soluções são as ferramentas de desenvolvimento RAD – *Rapid Application Development*. Estas aplicações consistem em ambientes de desenvolvido integrados capazes de gerar interfaces gráficas, serviços de *backend* e base de dados através de configurações ao nível da lógica de negócio. Caracterizam-se por serem soluções testadas, com uma arquitetura já definida e que desta forma permitem prototipagem rápida. Em contrapartida, normalmente oferecem graus de personalização muito baixos, uma vez que a maior parte destas não permite acesso ao código fonte, baseando apenas em modelos pré-configurados.

Numa abordagem um pouco menos complexa que a anterior é possível falar em (i) bibliotecas de software que aceleram o desenvolvimento e (ii) ambientes de desenvolvimento integrado

com aceleradores. As bibliotecas de software, recorrendo ao uso de padrões de desenvolvimento, geram ações e comportamentos de forma automática por configuração. Normalmente oferecem acesso ao código fonte e a sua geração de código consiste na replicação de modelos pré-configurados. Os ambientes de desenvolvimento integrados permitem a implementação de código tal e qual como esperado mas possuem extensões com a possibilidade de criação de partes do código de forma automática, seja pela geração de classes de software ou inserção de *snippets*. Esta abordagem apresenta altos níveis de personalização uma vez que oferece acesso ao código fonte, mas com a liberdade de desenvolvimento o risco de existirem erros ou inconformidades na arquitetura são maiores, necessitando assim de mais testes e validação.

A solução proposta por esta tese pretende assim integrar as melhores partes de cada uma das abordagens anteriormente apresentadas e remover os compromissos apresentados por ambas. Para isso tentou-se entender qual a base lógica que ambas as soluções oferecem. A conclusão foi que tanto ambientes RAD como ambientes aceleradores baseiam a sua atividade na definição da lógica de negócio por engenharia de software. Nos seus fundamentos, uma aplicação de software contém atores que desempenham casos de uso sobre entidades de negócio. Assim a proposta desta tese consiste num ambiente de desenvolvimento integrado de geração de código que permite o desenvolvimento de aplicações por indicação dos atores, entidades de negócio e casos de uso, assim como de todos os serviços e atividades necessárias ao desempenho do sistema. Este ambiente através da especificação de negócio ambiciona permitir a geração de um *frontend* multiplataforma, de um *backend* como serviço de interface aplicacional para agilizar integração com outras interfaces, de uma base de dados e ainda, se necessário, inclusão de *message queues* para integração com outros sistemas.

Nesta aplicação destinada a *developers* será possível criar novos projetos de software definindo quais as camadas necessárias e qual a tecnologia para cada camada. Cada opção tecnológica para uma camada consiste num conjunto de modelos pré-configurados para gerar código que serão invocados no momento da criação de qualquer elemento de negócio. Desta forma, ainda que atuando como uma plataforma RAD, esta solução oferece total liberdade tecnológica de tal forma que, caso o programador pretenda, poderá abandonar o uso do ambiente a qualquer momento e continuar a desenvolver o código criado sem apoio.

Para garantir que esta solução apresenta a melhor performance e os mais recentes avanços tecnológicos por omissão serão incluídos como modelos pré-configurados "Angular 2.0"[3] para *frontend*, "hapi"[4] e "sails"[5] para *backend*, base de dados não relacional em "MongoDB"[6] e "RabbitMQ"[7] para *message queues* de integração entre aplicações.

O foco principal desta abordagem é a velocidade. Para tal, esta plataforma apresenta uma interface multi-tarefa capaz de gerar código em *background* enquanto o programador vai configurando e programando o seu sistema através dos processadores de texto incluídos. Inclui também um conjunto de outras ferramentas na sua interface como por exemplo comandos GIT para gestão de versões, sistema de gestão de base de dados e evocador de interfaces de programação aplicacional.

Sendo um dos objetivos desenvolver software com uma experiência de utilização multiplataforma, os modelos pré-configurados para a geração de *frontend* que sejam baseados em tecnologia Web poderão ser executados no navegador, em tecnologia Electron[8] para execução nativa como aplicação para ambiente (Windows, macOS, Linux, etc.) e/ou em tecnologia Apache Cordova[9] para execução como aplicação nativa para sistemas operativos móveis (Android, iOS, Windows Phone, etc). Com isto as aplicações desenvolvidas terão acesso a todas as interfaces de programação de ambas as tecnologias, como por exemplo notificações, sensores, informações de sistema, câmaras, etc.

Depois de se procurar acelerar a produção de código de maior granularidade (como por exemplo: classes de software, controladores, serviços, etc.) surge a necessidade de também acelerar o desenvolvimento de código de menor granularidade (métodos, vistas, estilos, etc.). O processo tradicional envolve a escrita de código com posterior compilação e execução. Para acelerar a pré-visualização da aplicação/resultado final, a abordagem desta solução, sempre que possível, será ter uma atitude reativa às alterações efetuadas. Através da observação de alterações a ficheiros será possível despertar alterações à execução em andamento. Em *frontend* serão incluídos sockets através de "Socket.IO"[10] para conexão entre a aplicação criada e o ambiente de desenvolvimento de tal forma que alterações a vistas HTML ou código CSS se

[3] https://angular.io/
[4] http://hapijs.com/
[5] http://sailsjs.org/
[6] https://www.mongodb.com/
[7] https://www.rabbitmq.com/
[8] http://electron.atom.io/
[9] https://cordova.apache.org/
[10] http://socket.io/

reflitam de forma imediata sem qualquer outra intervenção do programador para além da escrita desse mesmo código. Nas restantes camadas quaisquer alterações iram despertar reinicio dos servidores se tal for necessário. De forma a exemplificar a potencialidade do sistema, o próprio ambiente gerador a desenvolver será implementado em tecnologias web e potenciado por Electron para poder disponibilizar aplicações desktop nativas em diferentes sistemas operativos. Uma outra razão por esta opção de tecnologias web reside na existência de uma visão futura deste software em que pudesse ser interessante usar o mesmo a partir de um navegador ou num sistema operativo móvel.

Em resumo, a abordagem consiste em utilizar a engenharia de software como base de criação de um sistema informático que será criado de forma acelerada pelo motor de geração de código incorporado, pela interface completa de utensílios auxiliares necessários e com as possibilidades de execução em multiplataforma e integração com outros sistemas.

1.4 Objetivos

A abordagem descrita neste documento pretende apresentar uma resposta ao problema cumprindo os seguintes objetivos:

- Acelerar o processo de desenvolvimento de aplicações, auxiliando e orientando os programadores na sua atividade de implementação através de uma ferramenta geradora de código;

- Permitir a inclusão de modelos para geração de código que mais tarde possam ser personalizados em cada projeto;

- Permitir a alteração de tecnologia e do modelo gerador de código em uso num projeto, gerando automaticamente toda a lógica de negócio definida na nova tecnologia e com o novo modelo gerador de código em uso;

- Disponibilizar um ambiente reativo a alterações que possa potenciar e acelerar o desenvolvimento de aplicações multiplataforma através da disponibilização de uma pré-visualização do resultado final, atualizada automaticamente a cada modificação de código;

- Providenciar comandos para tarefas auxiliares de implementação como gestão de versões e controlo de execução;
- Permitir a execução da aplicação final em formato web, aplicação desktop nativa e aplicação móvel nativa;
- Permitir desenvolver software como estruturas simples, escaláveis e extensíveis, que possam acompanhar o processo de desenvolvimento desde da sua versão protótipo até à entrega final;
- Permitir ao programador que possa alterar o código gerado de forma a ser possível estender as aplicações criadas sem qualquer restrição imposta pela plataforma;
- Facilitar a integração de informação por parte de outros sistemas com o sistema desenvolvido através do ambiente proposto;
- Garantir bons princípios de engenharia de software através da implementação por configuração do modelo de negócio.

1.5 Estrutura do Documento

A presente secção tem como objetivo apresentar a estrutura deste documento, elucidando o leitor sobre a lógica da mesma.

O capítulo 1 procura introduzir e contextualizar esta tese, apresentando o problema encontrado. É descrita a abordagem adotada e os objetivos a atingir visando a resolução desse mesmo problema.

Para criar uma abordagem eficaz torna-se necessário analisar o foi feito anteriormente para a resolução do problema. Desta forma no capítulo 2, são analisadas e comparadas as soluções já existentes para o problema abordado, concebendo um estado da arte do mercado tecnológico de ferramentas de desenvolvimento de software. São ainda apresentadas as motivações que despertaram e possibilitaram a execução desta tese.

De seguida no capítulo 3, de forma a justificar o trabalho executado, é explorado o valor de negócio que a solução apresentada poderá trazer para os interessados na mesma. É também

apresentado um modelo canvas para uma análise de como irá funcionar o negócio baseado na abordagem adotada.

O capítulo 4 apresenta os requisitos de software para a conceção da solução definindo o que é necessário executar. É realizado um enquadramento da proposta, apresentando a missão a cumprir, os atores e sistemas internos que irão definir os utilizadores e qual a prioridade de requisitos entre si. É também apresentado neste capítulo o diagrama de casos de uso gerados com base nos requisitos funcionais apresentados e nos objetivos a atingir definidos pelos requisitos não funcionais.

Por sua vez o capítulo 5 apresenta o design gráfico e funcional da solução. Assim, é apresentado o modelo de domínio que define a solução e os mockups gráficos dos ecrãs. São também incluídos os diagramas de sequência de funcionamento do gerador de código concebido e do funcionamento de um modelo gerador de código.

No capítulo 6 são abordados os detalhes da implementação dos requisitos definidos nos capítulos anteriores. São descritas quais as tecnologias utilizadas e são apresentadas evidências da implementação do design traçado.

De forma a comprovar a eficácia desta solução são apresentadas as experiências e a avaliação a executar à plataforma criada no capítulo 7. É abordada a metodologia que será seguida e apresentadas as experiências a executar.

O capítulo 8 apresenta as conclusões da elaboração desta tese, refletindo sobre todo o processo descrito neste documento e discutindo os resultados obtidos e analisando qual o possível trabalho futuro a concretizar.

2 Estado da arte

No presente capitulo é feita referência às soluções já existentes no mercado que de alguma forma procuram resolver o problema abordado nesta tese. Trata-se de uma atividade critica e reflexiva com o objetivo de auxiliar e melhorar a solução a desenvolver, entendendo o que já existe, o que não existe e, por consequência, aquilo que é possível fazer e deverá ser feito.

Após efetuar uma investigação do estado atual do mercado tecnológico perante o problema destacado foi possível apurar que existem algumas diferentes abordagens sobre a premissa de resolução do mesmo. Cada uma destas abordagens difere no formato com que projeta o problema e nos seus objetivos. São diferentes nos públicos-alvo na medida em que seguem um possível caminho tomando os compromissos do mesmo. Algumas das seguintes abordagens procuram resolver o problema apontado em toda a sua plenitude, enquanto outros resolvem apenas partes do mesmo, mas todos com o objetivo de promover o aceleramento do processo de desenvolvimento.

2.1 Rapid Application Development

As abordagens que mais se aproximam da visão do problema que motivou esta tese são as plataformas *Rapid Application Development*. Como já referido no capítulo anterior, estas aplicações consistem em ambientes de desenvolvido integrados capazes de gerar interfaces gráficas, serviços de *backend* e base de dados através de configurações ao nível da lógica de negócio. São soluções que permitem prototipagem rápida devido ao nível de geração de código que possuem. Este nível caracteriza-se por incluir um ambiente gráfico com a possibilidade de arrastamento de componentes para o ecrã que representará a aplicação a ser criada, interfaces

gráficas para gestão e consulta de bases de dados e fluxos de trabalho para representação de lógica de *backend*.

Uma outra característica destas plataformas é não dar acesso direto e simplificado ao código gerado. Para alterar código o utilizador deve aceder a controlos específicos na interface e apenas poderá efetuar modificações restritas em alguns componentes. Por este facto, admitindo que a cada versão da plataforma é devidamente testada antes de ser colocada em produção, é possível afirmar que estas plataformas diminuem o espaço possível para cometer erros ao implementar software, criando aplicações mais robustas e estáveis. Em contrapartida todo este controlo no processo de implementação implica menor liberdade criativa e de customização: (i) é mais complexo criar interfaces gráficas com muito detalhe gráfico e criativo; (ii) não é possível modificar arquitetura das diferentes camadas.

Visto que estas ferramentas RAD apenas permitem programação por construção de diagramas de fluxo de trabalho, funcionalidades como a gestão de versões, *debug* e *logging* são diferentes dos processos mais típicos. Assim tem em si integradas extensões próprias para concretizar estas atividades. Para além destas extensões, tipicamente necessitam também de processos de *deploy* em servidores muito específicos, o que leva às criadoras destas plataformas a disponibilizarem serviços de *cloud computing* para desenvolvimento e para colocação das aplicações criadas em produção. O *deploy* e configuração *on premises* também é possível quando devidamente acordado/negociado com o gestor da ferramenta.

As plataformas desta abordagem têm foco na modelação do negócio para gerar os seus componentes. Recorrem à definição de atores para representar papéis de utilizadores registados no sistema que operam sobre entidades do modelo de dados cujos dados são visualizados e manipulados em vistas. Esta modelação por engenharia de software e montagem de fluxos irá resultar na geração de código numa linguagem especifica.

Com estas funcionalidades torna-se possível desenvolver software de forma rápida e com um baixo nível de complexidade, sendo ferramentas muito usadas por empresas cujos recursos humanos envolvidos não têm tanta experiência em desenvolvimento de software ou cujo cliente pretende uma solução rápida para colocação em produção.

Dois exemplos bastante representativos desta abordagem são as plataformas "OutSystems"[11] e "WaveMaker"[12]. Ambas são descritas nas próximas sub-secções.

2.2 OutSystems

"OutSystems" é uma plataforma construída por uma equipa sediada em Braga, Portugal e é considerado pelo website de análise de plataformas "G2 Crowd"[13] (website com mais de 102200 análises de software de negócio) como uma das plataformas com maior satisfação por parte dos utilizadores sendo considerada uma plataforma de alta performance[14].

Esta plataforma é capaz de criar aplicações que poderão ser executada sobre tecnologia Microsoft.NET[15] ou JAVA[16] e ligar-se a base de dados Microsoft SQL Server[17], Oracle[18] entre outras após criação do respetivo adaptador para tal através da ferramenta de integração que disponibiliza. A plataforma em si, esta apenas disponível para ambiente Microsoft Windows e está ilustrada na Figura 3.

Tem como objetivo a criação de aplicações web para navegadores de desktop com capacidade mínima de adaptação a ambientes móveis. Oferece também a possibilidade de acrescentar um novo projeto de *frontend* com especificação para ambientes móveis, perante o mesmo *backend*. Estas últimas poderão ser executadas em modo nativo numa versão da tecnologia Apache Cordova personalizada pela própria OutSystems. Possui um ambiente de desenvolvimento integrado que consiste num construtor gráfico de interfaces. Todo o processo de desenvolvimento é gerido através da representação por fluxos de informação entre entidades e o programador tem qualquer tipo de acesso ao código gerado (excepto código CSS e JavaScript).

Devido às características inerentes de plataformas RAD neste ambiente de desenvolvimento, a empresa OutSystems disponibiliza, por um preço, um serviço em *cloud computing* para desenvolvimento e *deploy* para produção de aplicações. Em qualquer momento o programador poderá solicitar o abandono deste sistema em *cloud* e permanecer com o código fonte gerado

[11] https://www.outsystems.com/
[12] http://www.wavemaker.com/
[13] https://www.g2crowd.com/
[14] https://www.g2crowd.com/categories/rapid-application-development-rad
[15] https://www.microsoft.com/net
[16] http://www.java.com/
[17] https://www.microsoft.com/pt-pt/server-cloud/products/sql-server/overview.aspx
[18] https://www.oracle.com/database/index.html

da aplicação por um custo a negociar com a empresa criadora desta plataforma. Existem também a possibilidade de instalar um servidor OutSystems numa maquina pessoal de forma a melhorar o desempenho da plataforma. Esta plataforma é uma possível resolução parcial para o problema destacado nesta tese e é uma base inspiracional para a solução desenvolvida devido às suas capacidades de geração de código e gestão de base de dados.

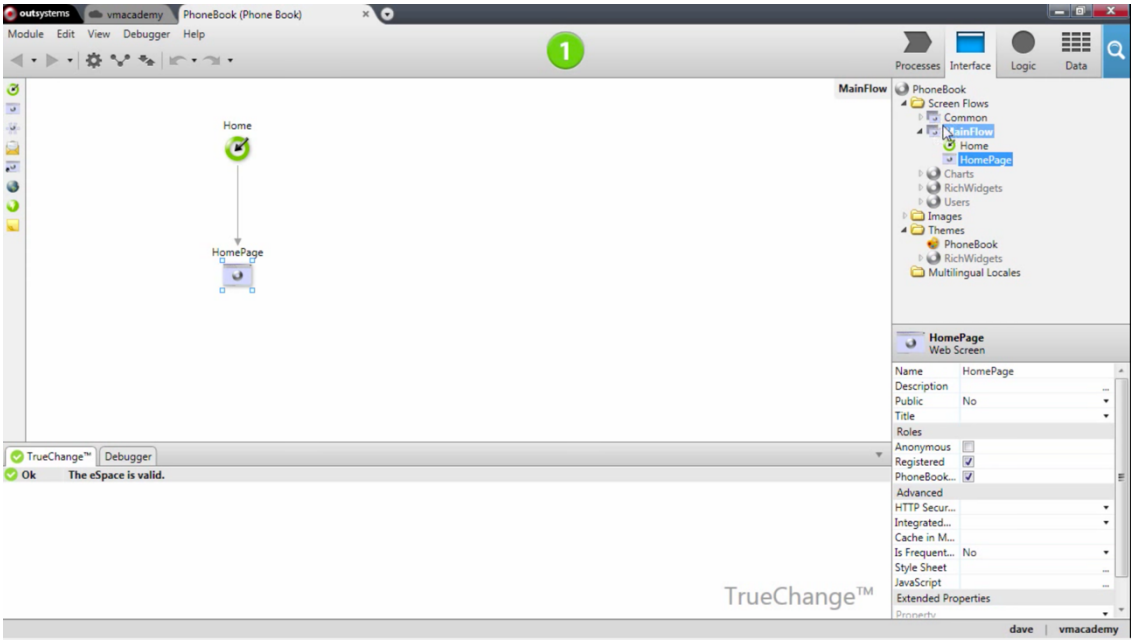


Figura 3 - Ecrã de Edição de Fluxo de Interfaces em OutSystems

2.3 WaveMaker

A "WaveMaker"[19] é uma plataforma RAD produzida pela empresa americana WaveMaker, Inc iniciada em 2003. Esta plataforma foi premiada internacionalmente pelas entidades "American Business Awards"[20], "Thinkstrategies"[21] e "Developer Week"[22] e afirma na sua página LinkedIn[23] que é capaz de criar aplicações de nível empresarial 67% mais rápido do que os métodos de desenvolvimento tradicionais.

Baseada em tecnologias abertas, esta plataforma é executada em ambiente web (sendo assim multiplataforma) conforme ilustrado na Figura 4, e permite a criação de aplicações web com

[19] http://www.wavemaker.com/
[20] http://stevieawards.com/aba
[21] http://thinkstrategies.com/
[22] http://www.developerweek.com/
[23] https://www.linkedin.com/company/wavemaker

design responsivo e aplicações móveis nativas (web-hibridas) utilizando AngularJS[24], JAVA, Bootstrap[25] e Apache Cordova.

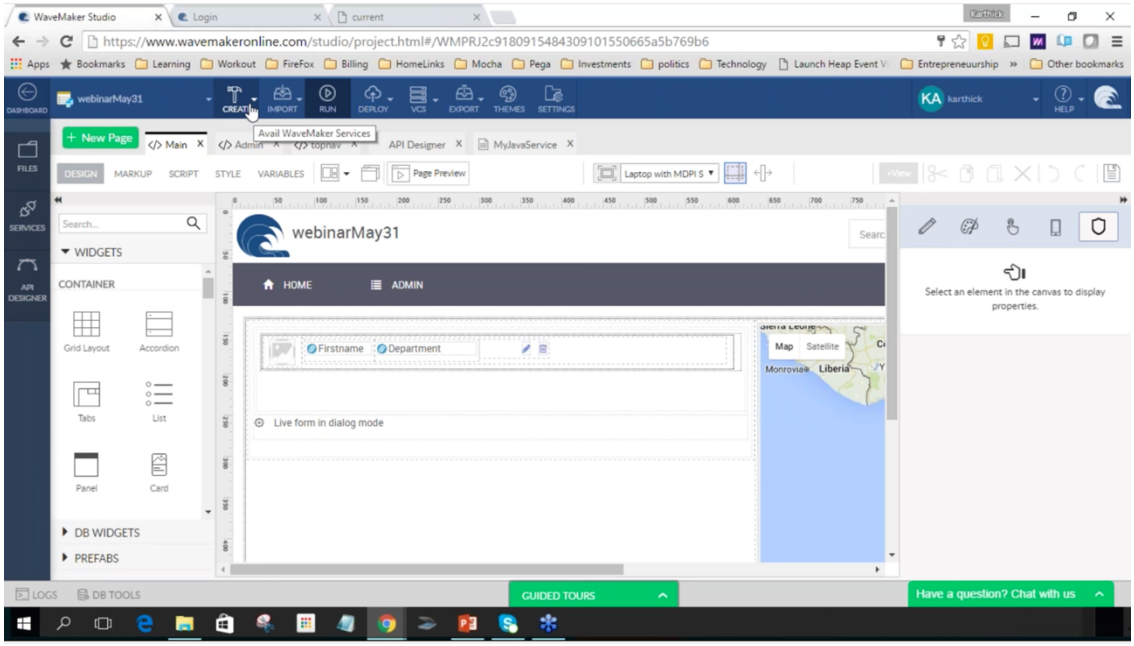


Figura 4 - Ecrã de Edição de Interface Gráfica em WaveMaker

Possui uma interface gráfica para construção de ecrãs por arrastamento de elementos, interface gráficas para consulta e manipulação de dados em bases de dados e ferramentas auxiliares para gestão de código em sistemas de gestão de versões e comandos para *deploy* em servidores AWS[26] ou WaveMaker.

Como plataforma RAD, não permite acesso a maior parte do código fonte e disponibiliza, por um preço, um serviço em *cloud* para *deploy*. Todo o ambiente é web e está dependente de licenciamento para ser possível executar. Esta plataforma é uma possível resolução parcial para o problema destacado nesta tese e é uma base inspiracional para a solução desenvolvida devido à sua natureza web e interface gráfica.

2.4 Gestão de Conteúdos

As plataformas de *Content Manager*, ou em português, Gestão de Conteúdos, são compostas por um conjunto de processos e tecnologias que suportam uma coleção de itens, gerindo e

[24] https://angularjs.org/
[25] http://getbootstrap.com/
[26] https://aws.amazon.com/

publicando informação de qualquer forma e por qualquer meio. Quando guardada e acedida via computadores, esta informação refere-se a conteúdos digitais. Estes conteúdos digitais podem ter o formato de texto, áudio, vídeo, ou qualquer outro tipo que cumpra os requisitos da plataforma. A grande mais valia destas tecnologias no âmbito desta tese é a sua capacidade de guardar informação sobre o formato de páginas web, permitindo ao programador adicionar novas paginas à aplicação sem ter necessidade de as codificar. O esforço inicial reside na criação de *Templates*, páginas web que irão definir o modelo para cada nova página criada. Este processo resulta na possibilidade de as páginas poderem ser criadas por um utilizador sem conhecimentos de programação.

Dois exemplos bastante representativos desta abordagem são as plataformas "WordPress" e "Sitecore". Ambas são descritas nas próximas sub-secções.

2.5 WordPress

O WordPress[27] é um sistema de gestão de conteúdo que pode ser usado para criar e manter o conteúdo de um site, de um blog ou de uma aplicação. Combina estética, standards da web e usabilidade e é um serviço gratuito. É uma solução *open-source* desenvolvido por centenas de voluntários de uma comunidade mundial. Possui *plugins* e temas criados pela comunidade na internet que potenciam o desenvolvimento nesta plataforma.

Esta plataforma web (ilustrada na Figura 5) permite ao utilizador mesmo sem conhecimentos de programação criar um website de forma apoiada e acompanhada, na medida em que todas as paginas criadas seguem Templates definidos no tema selecionado no momento da configuração inicial. Apesar de não resolver o problema desta tese, é um caso de estudo no que toca à velocidade de desenvolvimento e apoio da comunidade na internet.

[27] https://wordpress.com/create/

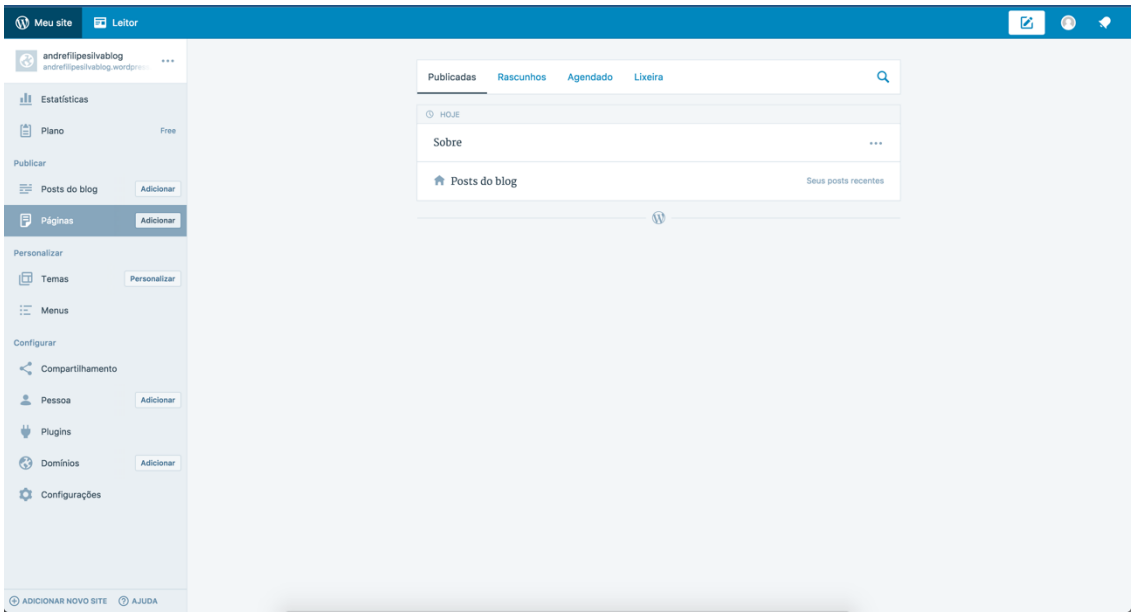


Figura 5 – Menu de Edição de Site no WordPress

2.6 Sitecore

A plataforma Sitecore[28] é construída em ASP.NET[29] e foi considerada Líder no Quadrante Mágico[30] da agência Gartner[31] para *Web Content Manager* em 2015. Permite a editores de conteúdos web e a gestores de marketing ter controlo completo dos aspetos do seu website desde da integração com as redes sociais e publicações de blog até às opções mais avançadas como personalização e *e-commerce*. O Sitecore foi lançado em 2001, e contém uma interface web executada localmente, ilustrada na Figura 6, que permite tarefas como edição de conteúdos, gestão de utilizadores, monotorização de serviços e configuração de fluxos de trabalho.

Esta plataforma permite a criação de *Templates* que poderão ser mais tarde utilizados para gerar páginas para o website. Possui uma interface para construção por arrastamento de elementos para a área de desenho, funcionalidades de desenho de *wireframes* e design preparado para interfaces móveis. Permite a integração com CRM, ERP, APIs e bases de dados, integra serviços internos de *analytics, insights* e automação de processos.

[28] http://www.sitecore.net/en
[29] http://www.asp.net/
[30] https://www.gartner.com/doc/3101917/magic-quadrant-web-content-management
[31] http://www.gartner.com/technology/home.jsp

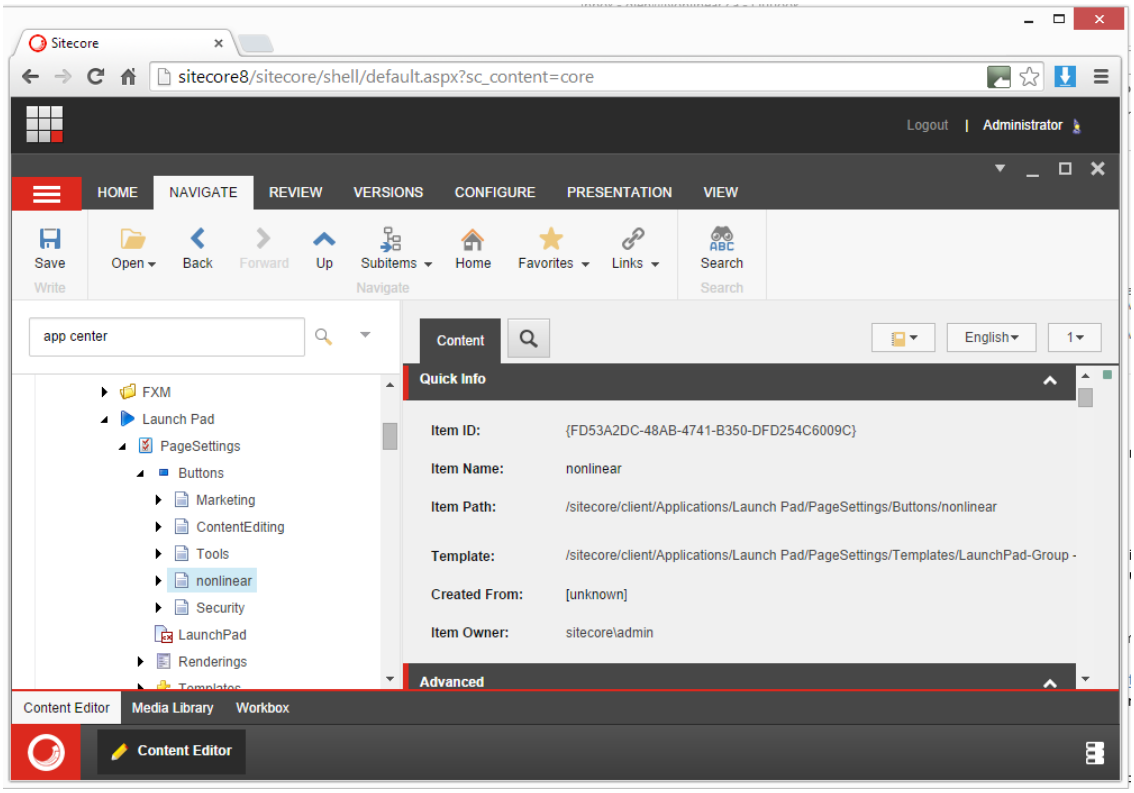


Figura 6 - Ecrã de Trabalho da Plataforma Sitecore

O Sitecore responde ao problema apresentado na medida em que permite gerar páginas para o website com base nos *Templates* previamente criados, efetuando geração de código para acelerar o desenvolvimento. Esta solução não resolve o problema apresentado, mas é um caso de estudo no que toca à capacidade de geração de código.

2.7 Ambientes de Desenvolvimento Integrado

Os ambientes de desenvolvimento integrados (também conhecidos como IDEs - *Integrated Development Environments*) são aplicações concebidas para permitir redigir código fonte. São a abordagem mais tradicional da programação e consistem num processador de texto com um compilador e a capacidade de executar o código redigido. Alguns destes ambientes incluem ferramentas capazes de gerar código-fonte, acelerando a tarefa de codificação do programador, respondendo desta forma ao problema abordado. Estas ferramentas não resolvem diretamente o problema desta tese mas possuem funcionalidades que interligação o desenvolvimento e a geração automática de código.

2.8 Microsoft Visual Studio

O ambiente Visual Studio[32] é produzido pela empresa Microsoft[33] e é possivelmente um dos mais conhecidos ambientes de programação. Para além das tradicionais funcionalidades de adição de *snippets* e inclusão de bibliotecas de código, inclui também funcionalidades de *Scaffolding*[34]. Estas funcionalidades consistem na montagem de componentes e aplicam-se no processo de criação de código de maior granularidade.

A titulo de exemplo, no padrão MVC, esta funcionalidade (demonstrada na Figura 7) permite que o programador possa definir o nome do controlador e associar uma entidade do modelo de negócio, resultando isso em geração automática de código, criando o controlador para a entidade de negócio definida e vistas para listar, ver, editar e remover instâncias dessa mesma entidade. Após isto apenas resta ao programador adicionar validações ou alterações que considere necessárias ao código gerado.

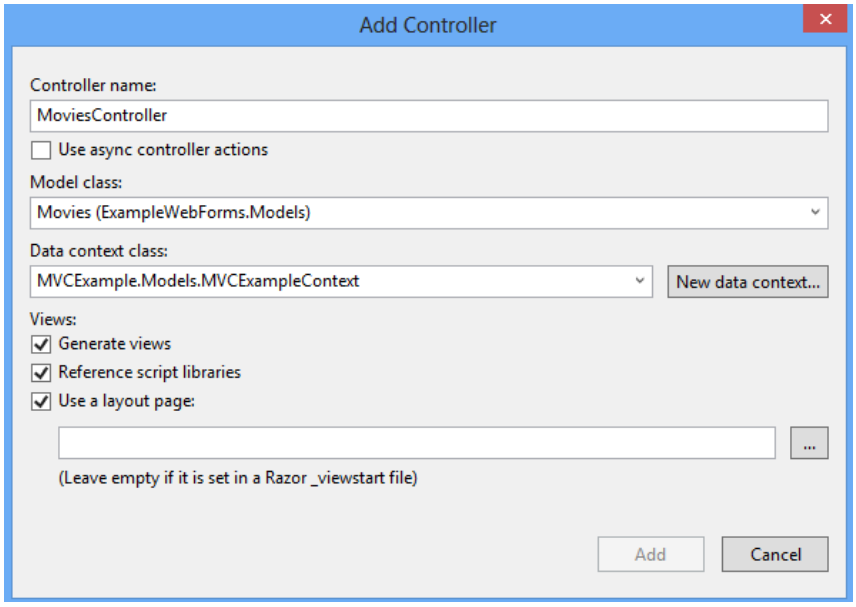


Figura 7 - Geração de Código para um Controlador no Microsoft Visual Studio 2013

Esta funcionalidade, apesar de suscetível a erros por ausência de contexto de negócio, é muito próxima daquilo que é a abordagem desenhada para o problema destacado. Esta geração de

[32] https://www.visualstudio.com/
[33] https://www.microsoft.com
[34] http://www.asp.net/visual-studio/overview/2013/aspnet-scaffolding-overview

código concilia uma entidade de modelo de negócio e gera toda a funcionalidade base CRUD para a gestão da mesma, acelerando o desenvolvimento de código.

2.9 Netbeans

O IDE Netbeans[35] é produzido pela Netbeans Inc. com o patrocínio da Oracle. Consiste num ambiente para produção de aplicações desktop, aplicações web e móveis com JAVA, JavaScript, HTML5, PHP, C/C++ entre outras linguagens. O Netbeans é um projeto gratuito e de código aberto com uma comunidade mundial de utilizadores e programadores.

Este IDE com o *plugin* Visual Paradigm[36] permite ao programador criar um diagrama de classes e gerar o código correspondente numa linguagem definida previamente para o projeto.

Esta funcionalidade, ilustrada na Figura 8, permite ao programador desenhar todo o modelo de negócio em linguagem UML e converter o mesmo sem esforço adicional para o código fonte correspondente, acelerando assim o desenvolvimento.

Figura 8 - Operação de Adição de um Método a uma Interface em Netbeans

2.10 Comparação

Após identificar algumas tecnologias em mercado com potencialidades semelhantes à abordagem pretendida e objetivos de resolução do mesmo problema, surge a necessidade de procurar sumarizar a informação de forma a obter um estado de arte sobre aquilo que já foi concretizado e aquilo que poderá ainda ser concretizado.

[35] https://netbeans.org/
[36] https://www.visual-paradigm.com/tutorials/netbeans-java-to-uml.jsp

A Tabela 1 tem assim como objetivo comparar diretamente soluções RAD, de Gestão de Conteúdos e Ambientes de Desenvolvimento integrado de forma a obter um estado visual das expectativas a poder alcançar.

Tabela 1 - Comparação de Funcionalidades entre as Tecnologias Abordadas

Funcionalidade	**RAD**	**Gestão de Conteúdos**	**Ambientes de Desenvolvimento Integrado**
Aplicação desktop multiplataforma	Não (apenas OutSystems em Windows)	Não	Sim
Geração de código com base no modelo de negócio	Sim	Sim	Não
Geração de código com base em UML	Não	Não	Sim
Criação de aplicações móveis nativas	Sim	Não	Sim
Construção de Aplicações por arrastamento de componentes	Sim	Sim	Sim
Acesso ao código	Sim (mas com acesso limitado)	Sim (apenas código web)	Sim
Possível utilização por não programadores	Sim	Sim	Não
Permite modificação das *Blueprints*	Sim (mas apenas interface)	Sim	Não
Liberdade de escolha de linguagem de código/tecnologias a usar na geração automática de código	Não	Não	Sim

2.11 Tecnologias Usadas

Na conceção da abordagem ao problema apontado nesta tese foram tidas em conta algumas tecnologias que representam funcionalidades uteis/interessantes para a resolução do mesmo. São tecnologias recentes com propriedades que combinadas podem potenciar uma solução completa com um conjunto de funcionalidades que farão a diferença perante outras abordagens já existentes. Funcionaram como inspiração para a implementação descrita neste documento.

2.11.1 Electron

A *framework* Electron[51], ilustrada na Figura 9, é escrita em JavaScript e é executada sobre Node.js. A sua principal funcionalidade permitir criar aplicações desktop multiplataforma com base em código web. Funciona como um *iframe* que permite acesso às interfaces aplicacionais de cada sistema operativo, encapsulando essa lógica pelo utilizador.

Esta plataforma combina o browser Chromium[52] e Node.js e permite criar aplicações utilizando tecnologias web: HTML, CSS e JavaScript. É um projeto de código aberto mantido no GitHub[53] compatível com todas as plataformas (devido à sua base Node.js).

A sua utilidade perante a abordagem a concretizar é a sua capacidade de gerar aplicações desktop nativas para sistemas operativos Windows, macOS e Linux através de um código web. Admitindo que todo o código do ambiente a criar como o código gerado serão web, esta plataforma irá permitir acesso a funcionalidades como atualizações automáticas da própria aplicação, menos nativos e notificações, gestão de energia e pastas do sistema. Esta é a tecnologia que potencia aplicações como Slack[54], Visual Studio Code[55] e Atom[56].

[51] http://electron.atom.io/
[52] https://www.chromium.org/
[53] https://github.com/
[54] https://slack.com/
[55] https://code.visualstudio.com/
[56] https://atom.io/

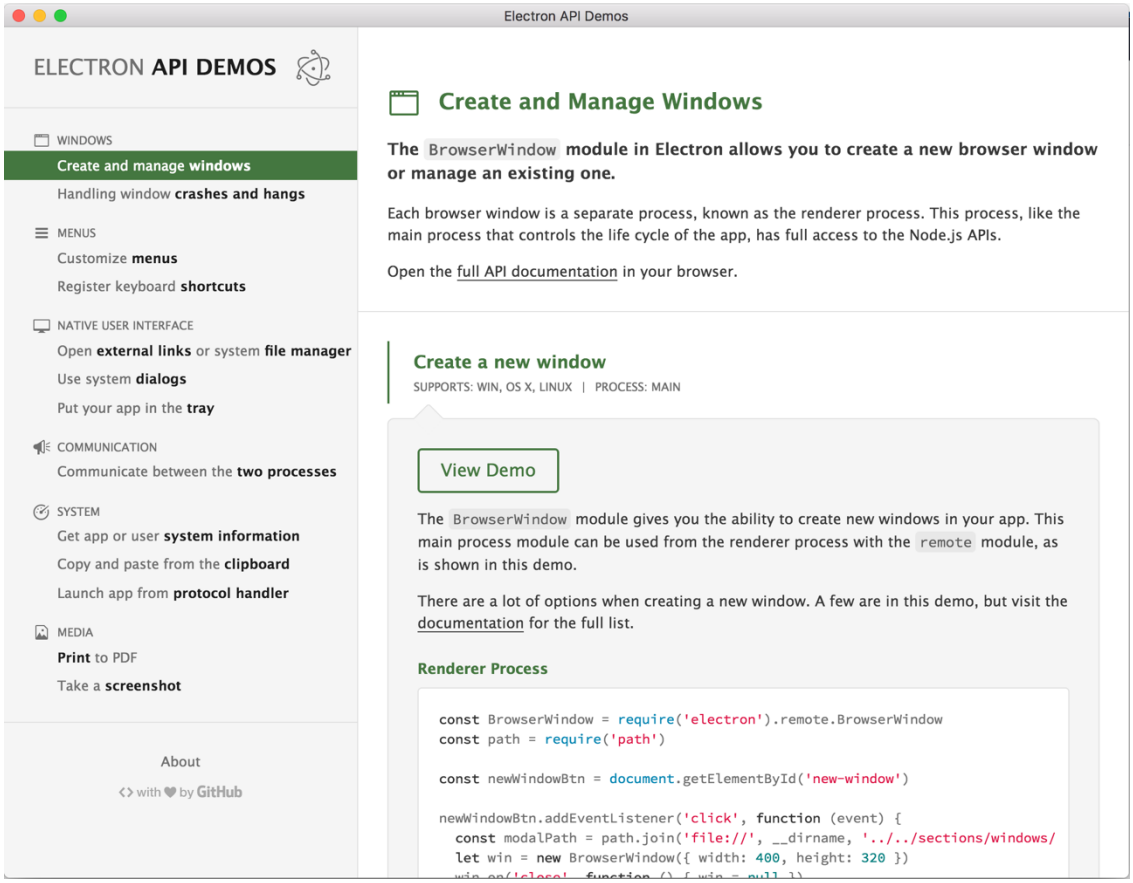


Figura 9 - Janela de Demonstração de Demos Electron

2.11.2 Apache Cordova

O projeto Apache Cordova é licenciado pela "The Apache Software Foundation" e consiste numa exportação de um *iframe* para os diferentes sistemas operativos móveis, capaz de executar aplicações web e oferecendo acesso às interfaces de programação de aplicações de cada um desses sistemas de forma abstraída. Esta plataforma é gratuita e de código aberto e tem como objetivo permitir a escrita de um código base para múltiplas plataformas móveis através unicamente de HTML, CSS e JavaScript.

Esta tecnologia disponibiliza um conjunto de *plugins* para dar acesso às diversas funcionalidades de cada sistema operativo como por exemplo: câmara, sensores, leitores de impressão digital, vibração, informações de sistema, etc. Possui também uma interface de testes durante o desenvolvimento, ilustrada na Figura 10.

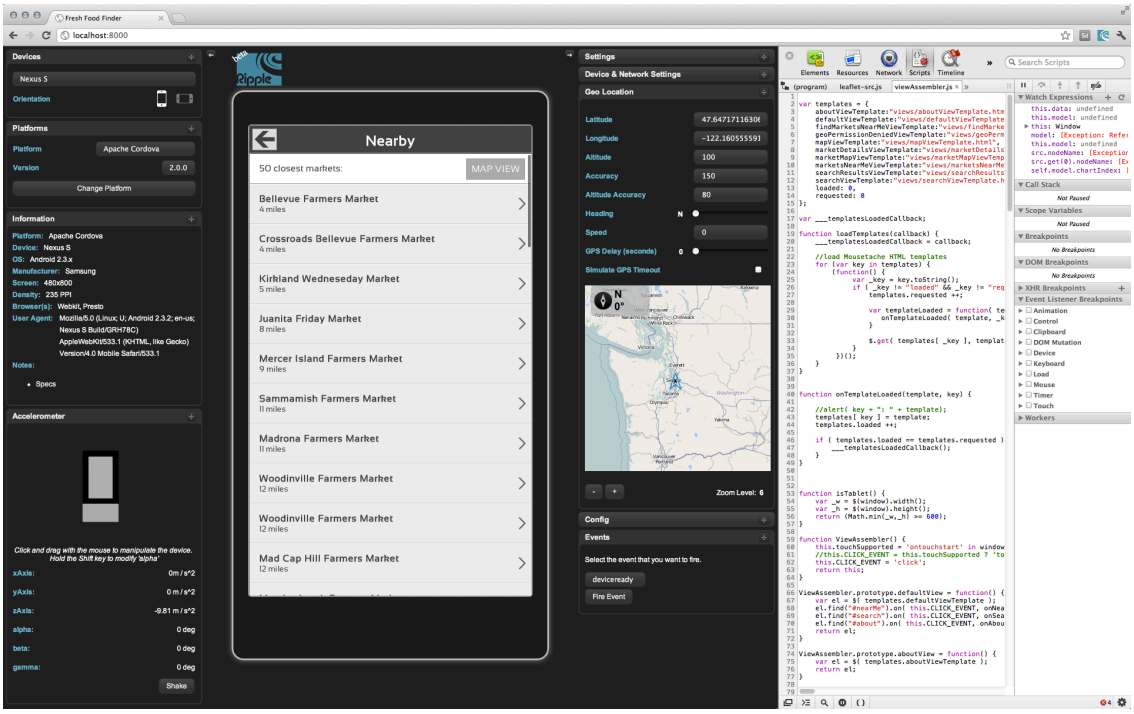


Figura 10 - Exemplo de Debug de uma Aplicação Apache Cordova

Esta tecnologia motivou a abordagem traçada na medida em que, admitindo a criação de um só código web, é possível criar aplicações nativas para sistemas operativos móveis sem esforço adicional.

2.11.3 Angular 2.0

O Angular 2.0[57] é a mais recente iteração da *Framework* de desenvolvimento de aplicações *client-side* apoiada pela Google[58]. Desenvolvida em JavaScript permite a criação de aplicações web por HTML, CSS e TypeScript. Esta nova edição tem foco em performance e velocidade, disponibiliza uma nova sintaxe com vasto numero de ferramentas e tem uma vasta comunidade mundial de apoiantes. Abandonou o padrão MVC e passou a executar a sua estrutura sobre o formato de componentes que podem ser utilizados dentro de outros componentes e em que um determinado componente corresponde à aplicação em si. Inclui também como novidade a possibilidade de criar aplicações *server-side*, funcionalidade útil para necessidades de *frontend* com menor exigência a nível de casos de uso.

[57] https://angular.io/
[58] https://www.google.com/about/company/

A principais motivações encontradas nesta tecnologia para a abordagem tomada são: (i) a possibilidade de criar aplicações web de forma estruturada e com recurso a TypeScript; (ii) Performance, desempenho e apoio da comunidade; (iii) Angular CLI[59].

O Angular CLI consiste na possibilidade de gerar e executar código Angular a partir de comandos numa linha de comandos. A partir de determinados *blueprints* (também conhecidos *templates*) podem ser gerados os diversos elementos de código necessários. Isto representa uma estratégia muito semelhante à estratégia planeada e poderá ser incorporado pelo próprio ambiente a desenvolver na medida em que definir lógica de negócio poderá utilizar este gerador para criar o código necessário correspondente para essa funcionalidade ser possível.

2.11.4 Sails

A plataforma Sails[60] tem como objetivo a construção de aplicações Node.js de nível empresarial. É desenhada para emular o padrão MVC mas com suporte às necessidades das aplicações modernas: *data-driven* APIs escaláveis e uma arquitetura orientada a serviços. É uma *Framework* escrita totalmente em JavaScript, compatível com várias bases de dados através da abstração conferida por um conjunto de adaptadores (oriundos da *framework* Waterline[61]) e é executada sobre Node.js, Express[62] e Socket.io[63].

Mas o principal interesse que serviu de motivação a este projeto é a funcionalidade de geração de código para APIs. O Sails possui um conjunto de *blueprints* que definem a estrutura de uma API para uma determinada entidade de negócio. O comando "sails generate api dentista" permite gerar uma REST API para a entidade do modelo de negócio "dentista" com as operações CRUD. Esta funcionalidade acelera a produção de *backend* na medida em que o utilizador não tem de replicar e alterar o código manualmente.

[59] https://cli.angular.io/
[60] http://sailsjs.org/
[61] https://github.com/balderdashy/waterline
[62] http://expressjs.com/
[63] http://socket.io/

2.11.5 adminMongo

O adminMongo[64] é um projeto de código aberto que consiste numa interface gráfica em tecnologias web para gerir conexões e bases de dados MongoDB.

Este projeto foi uma motivação para a abordagem adotada na medida em que torna possível combinar dentro da própria aplicação um sistema de gestão de base de dados, centralizando todas as tarefas num só local acredita-se poder acelerar o desenvolvimento por diminuir o tempo e a necessidade de troca de aplicações.

[64] https://github.com/mrvautin/adminMongo

3 Análise de Negócio

No presente capítulo é apresentada uma análise ao modelo de negócio a ser implementado na solução desenvolvida. Acredita-se que, mais do que desenvolver uma solução, é necessário entender se essa solução acrescenta algum valor à realidade existente. Assim é neste momento contextualizado qual o valor inerente e qual o valor que esta solução enquanto negócio poderá gerar.

3.1 Valor

Numa era onde existem cada vez mais dispositivos digitais ligados ao nosso dia-a-dia a Engenharia de Software é apontada como o futuro. Um artigo da revista Visão[65] de 13 de Junho de 2016 afirma que "a procura de especialistas nas áreas de engenharia de software e de telecomunicações quadruplicou nas bases de dados de empresas de recrutamento como a Hays[66]". Afirma ainda que "as funções nas engenharias e telecomunicações vão manter a dinâmica devido à constante inovação tecnológica. Informação e tecnologia tem 100% de emprego à saída da universidade". Com este aumento de procura pode deduzir-se que existem hoje, talvez mais do que anteriormente, necessidades vastas de produção de software.

A propósito da necessidade de engenheiros informáticos, um artigo de 2008 do Dr. Lyle N. Long da "The Pennsylvania State University", com o título "The Critical Need for Software Engineering

[65] http://visao.sapo.pt/actualidade/sociedade/2016-06-13-Profissoes-com-futuro
[66] https://www.hays.pt/

Education"[67] realça que o software afeta quase todos os aspetos da nossa vida diária (indústria, bancos, viagens, comunicações, defesa, medicina, investigação, governação, educação, entretenimento, lei, etc). Numa era digital onde cada pessoa está conectada ao mundo através de um computador no seu bolso surgem novos serviços digitais, novas expectativas, partilhas de informação e necessidades de integração. Tudo isto resulta em necessidades de desenvolvimento de software, facto que poderá estar associado a existir cada vez mais emprego para engenheiros informáticos. Mais software implica mais empresas de desenvolvimento, mais produtos e mais competição em mercado.

Apesar de não podermos confirmar se a velocidade com que se dão os negócios está ou não a aumentar, como afirma o artigo[68] "The creed of speed - Is the pace of business really getting quicker?" do website "The Economist"[69], podemos olhar à nossa volta e ter uma noção da velocidade atual de produção de software. Um telefone Android tem atualizações para as suas aplicações numa base quase diária. Em 1913, Henry Ford reinventou o fabrico em linha de montagem e cortou o tempo de construção de um automóvel de 12 horas para 90 minutos, mais carros em menos tempo. Em 1965, Gordon Moore, cofundador da Intel previa num artigo que o número de transístores num circuito integrado iria duplicar a cada dois anos, facto que praticamente se manteve real até 2011, maior capacidade de processamento em chips do mesmo tamanho. Esta previsão ficou conhecida como a "Lei de Moore" e levou ao nível computacional do nosso quotidiano. Velocidade aparenta ser um fator critico para o sucesso dos negócios.

O Dr. John Sullivan, especialista em recursos humanos de Silicon Valley, escreveu no seu artigo de 2014 "The Need For Speed And Why It Is Critical For Business Success"[70] que século 21 pode ser apelidado de "o século onde a velocidade dominou". Este artigo afirma que independentemente de onde trabalhamos no mundo, quase toda a gente pode sentir o facto que todos os aspetos dos negócios globais parecem mais significativamente mais rápidos que há 10 anos atrás. É uma época tão rápida que empresas como a Apple e a Google não foram rápidas o suficiente, permitindo o aparecimento e domínio de mercado por empresas como o Facebook ou o Twitter. Empresas que não se adaptem rápido podem desaparecer como foi o caso, por exemplo, da Kodak, da Xerox e da Blockbuster.

[67] http://www.personal.psu.edu/lnl/papers/CrossTalk_2008.pdf
[68] http://www.economist.com/news/briefing/21679448-pace-business-really-getting-quicker-creed-speed
[69] http://www.economist.com/
[70] https://www.eremedia.com/tlnt/the-need-for-speed-and-why-it-is-critical-for-business-success/

Um estudo[71] de 2014 da empresa CA Technologies afirma que modernização de software pode dar às empresas vantagens competitivas ajudando-as a acelerar os seus ciclos de produção e *time-to-market*, a reduzir custos de manutenção e aumentar a performance geral de negócio e lucros. Um estudo[72] da agência Gartner refere que é esta mesma modernização que está por detrás do crescimento das empresas: As empresas estão num estado de transição para novos formatos de consumo, abandonando as suas estruturas tradicionais e passando para o formato de soluções como serviços. A Gartner refere também que em 2020, 75% do processo de compras para suporte ao processo digital será "construído" e não "comprado". Ou seja, haverá uma preferência pela construção com base em módulos já testados do que em soluções out-of-the-box, isto implica desenvolvimento de software à medida, mais desenvolvimento, mais engenheiros necessários.

Em resumo, as empresas precisam de mais velocidade, e para a terem necessitam de mais e melhor software, desenvolvido de forma ágil e com produção rápida para que as empresas possam então acompanhar o ritmo que o mercado parece apresentar, podendo assim adaptarem-se e serem competitivas.

Mas quão rápido pode ser demais? Poderá esta necessidade de velocidade conduzir os programadores a um ponto de rotura? Mais requisitos em cada vez menos tempo. Segundo um artigo[73] de Matt Calkins, CEO da Appian, publicado no Wired em 2015, existem duas maneiras de lidar com esta constante necessidade de velocidade: (i) desmistificar os preconceitos de como o trabalho de desenvolvimento de software é feito ou (ii) usar novas plataformas para acelerar a entrega de soluções. O artigo sugere considerar o uso destas formas mesmo por membros da equipa que não sejam de áreas tecnológicas, especialmente quando estas não necessitam de conhecimentos de programação, deixando os programadores para os momentos mais necessários e agilizando assim a alocação de recursos. Esta é uma das premissas defendidas pelas plataformas RAD – não são necessários conhecimentos avançados de engenharia informática para se ser capaz de usar uma plataforma RAD (devido à sua lógica de operação).

Mais necessidades de software num mercado cada vez mais rápido levam à necessidade de mais engenheiros. Isto traduz-se em produzir software mais rapidamente e isso é algo que as

[71] http://www.ca.com/us/company/newsroom/press-releases/2014/modernizing-enterprise-software-increases-business-performance-speeds.html
[72] http://www.gartner.com/newsroom/id/3119717
[73] https://www.wired.com/insights/2015/01/need-for-speed-developers-breaking-point/

plataformas RAD podem solucionar. Mas poderão estas plataformas ser uma solução completa? Perante projetos em prototipagem estas plataformas são claramente uma vantagem pois permitem em poucos passos criar um software demonstrável. Mas o baixo nível de personalização que oferecem por não permitir acesso ao código fonte é uma desvantagem ao comprometer variáveis como performance e design. Ainda, as plataformas RAD possuem uma arquitetura tecnológica predefinida que normalmente poderá ser pouco ou nada alterada.

Ter software criado rapidamente é necessário, mas segundo a empresa Invision[74] no seu documentário de 2016 "DESIGN DISRUPTORS"[75], a experiência de utilização é essencial e aqui as plataformas RAD poderão não ser a solução. Após iniciar trabalhos de implementação com estas plataformas o produto final poderá ser uma solução completa criada rapidamente, mas irá oferecer a performance e o detalhe de performance necessário? Utilizando estas plataformas não será possível aceder a código nativo de cada um dos sistemas operativos em que se irá operar e este facto poderá resultar num trabalho extra de reconcepção do software numa nova tecnologia sem o apoio concedido pela RAD.

Então é neste momento que começa a surgir o valor da abordagem apresentada nesta tese. O valor de negócio consiste em potenciar o programador, disponibilizando um ambiente gerador de código, código este que será gerado segundo o modelo pré-configurado em uso. Por sua vez estes modelos consistem num conjunto de ficheiros de código numa determinada tecnologia ou linguagem. Mas na prática, esta geração de código já existe implementada nos tradicionais ambientes de desenvolvimento. Tipicamente quando se cria um novo projeto, uma nova classe, os ambientes de desenvolvimento geram o código base pelo programador. Mas é aqui que a abordagem apresentada vai mais longe. O valor está presente na geração de código com contexto de negócio. Ao utilizar a plataforma que se pretende implementar, o programador poderá gerar código com base num caso de uso de negócio, não só gerando código para uma determinada pasta ou camada, mas sim código em todas as camadas de forma transversal, tornando assim possível acelerar a produção de código e ainda assim obter o máximo nível de personalização possível, tendo total acesso ao código. Acresce ainda a possibilidade ao programador de personalizar o próprio modelo gerador de código em uso em cada projeto de forma singular, aumentando assim ainda mais o nível de personalização possível.

[74] https://www.invisionapp.com/
[75] https://www.youtube.com/watch?v=W4AViRgrgkU

Definindo atores, entidades e casos de uso, o programador torna-se capaz de gerar código no *frontend*, *backend*, base de dados e em camadas de integração. Mais tarde, se pretender, poderá reutilizar a definição de negócio para novamente gerar código com um outro modelo pré-configurado, acelerando um inicio de um novo projeto. O valor de negócio da solução apresentada reside no aceleramento do processo de desenvolvimento de código sem compromissos futuros e com base nos princípios de engenharia de software.

3.2 Modelo Canvas

De forma a descrever o modelo de negócio inerente à abordagem apresentada nesta tese é apesentado um modelo Canvas. Um modelo de Canvas descreve como é criado valor de mercado, quem é o cliente, qual o seu problema, qual é o produto que resolve esse problema e quais os benefícios desta atividade. Este modelo é composto por nove blocos, nomeadamente: (i) Segmentos de clientes; (ii) Propostas de valor; (iii) Canais; (iv) Relacionamento com clientes; (v) Fontes de receita; (vi) Recursos-chave; (vii) Atividades-chave; (viii) Parcerias-chave e (ix) Estrutura de custos.

O Segmentos de clientes representa grupos de pessoas ou organizações com necessidades ou comportamentos comuns que se pretende servir. São o fundamento do modelo de negócio, a sua razão de ser. E na abordagem apresentada existem 3 segmentos a considerar: (i) os programadores – na medida em que vêm o seu trabalho diário facilitado e agilizado; (ii) as empresas de software que empregam programadores – que obtém uma produção mais rápida e menos limitada por código gerado ao qual não tem acesso ou controlo e ainda conseguem garantir melhor a qualidade e consistência desse código; e (iii) os clientes dessas mesmas empresas de software – que recebem o software que encomendam num menor espaço de tempo e assim possivelmente com menores custos.

Proposta de valor – produtos e serviços que geram valor para os segmentos de clientes. São a forma como se responde às necessidades dos clientes. Neste sentido é apresentado: (i) aceleração do desenvolvimento de software; (ii) desenvolvimento de software com base no modelo de negócio; (iii) Personalização do gerador de código; e (iv) possibilidade de editar o código gerado, eliminando limitações de melhoria.

Canais – quais os canais pelos quais existem comunicação e entrega de valor ao cliente. Correspondem à interface entre a empresa fornecedora do serviço e o utilizador. Neste caso esta interface consiste em: (i) um website onde a solução será convenientemente apresentada segundo o tipo de visitante (segmento de cliente) e com uma loja online de exposição de modelos geradores de código para incluir na própria plataforma; (ii) Participação ativa nas redes sociais para divulgação e publicidade da solução; e (iii) Participação em feiras e convenções de forma a divulgar a solução perante possíveis utilizadores.

Relacionamento com clientes – tipos de relacionamentos estabelecidos entre a empresa e os segmentos de clientes. Estes relacionamentos serão: (i) Serviço de Assistência dedicada onde por um custo os utilizadores poderão pedir apoio para utilização da plataforma e desenvolvimento dos seus próprios projetos; (ii) A comunidade web onde todos os utilizadores da plataforma e equipa de desenvolvimento poderão participar deixando ideias, sugestões e criticas; e (iii) Academias de formação onde irá ser lecionado aos participantes como utilizar a solução concebida, podendo os mesmos obter certificações sobre a frequência e resultados da participação na mesma.

Fontes de receita – possibilidades de geração de valor financeiro que a empresa poderá obter com cada um dos segmentos de clientes. Corresponde ao valor que o cliente está disposto a pagar pelo valor para ele gerado. A geração de valor está pensada para existir nos seguintes meios: (i) comercialização do software para uso profissional; (ii) vendas de modelos geradores de código; (iii) disponibilização de serviços de *cloud computing* para desenvolvimento e produção; e (iv) venda de serviços de assistência dedicada.

Recursos-chave – ativos fundamentais para fazer o negócio funcionar. São exemplos os ativos físicos, intelectuais, recursos humanos e financeiros. Os principais ativos serão (i) a comunidade de programadores na internet que venha a apoiar a solução; (ii) possíveis investidores nesta solução; (iii) clientes finais que procuram um software feito à medida mas ainda assim rapidamente elaborado; e (iv) a equipa de desenvolvimento da solução apresentada.

Atividades-chave – atividades fundamentais para que o modelo de negócio opere como esperado, nomeadamente produção de bens, resolução de problemas, gestão de plataformas, entre outras. Aqui enumera-se: (i) manutenção e melhoria da arquitetura do ambiente gerador de código; e (ii) criação e melhoramento dos modelos de geração de código.

Parcerias-chave – fornecedores e parceiros que garantem o funcionamento do modelo de negócio como alianças estratégicas, redes de cooperação, parcerias de exclusividade, entre outras. A solução proposta tem dependências nas tecnologias que usa, sendo que as mesmas são as principais parcerias que elevam a produção a um nível de unicidade importante, nomeadamente: (i) Node.js – tecnologia sobre a qual assenta o ambiente implementado; (ii) Angular 2.0 – tecnologia sobre a qual é implementada a interface do ambiente; (iii) Cordova tecnologia a partir do qual é possível conceber aplicações móveis nativas a partir de aplicações web; (iv) Electron – tecnologia sobre a qual assenta a aplicação desktop do ambiente; (v) hardware para alojamento de soluções; e (vi) investidores que poderão comparticipar no projeto investindo recursos para o crescimento da solução enquanto plataforma.

Estrutura de custos – principais custos na operação do modelo de negócio, como por exemplo custos fixos, custos varáveis, economias de escala, comissões, entre outros. Referem-se (i) custos com Marketing; (ii) custos com a equipa de desenvolvimento desta solução e (iii) custo de aluguer/compra de hardware para alojamento de soluções.

Desta forma, resumindo a visão de negócio resulta a Figura 11 onde é apresentado o modelo de negócio proposto pela abordagem descrita nesta tese.

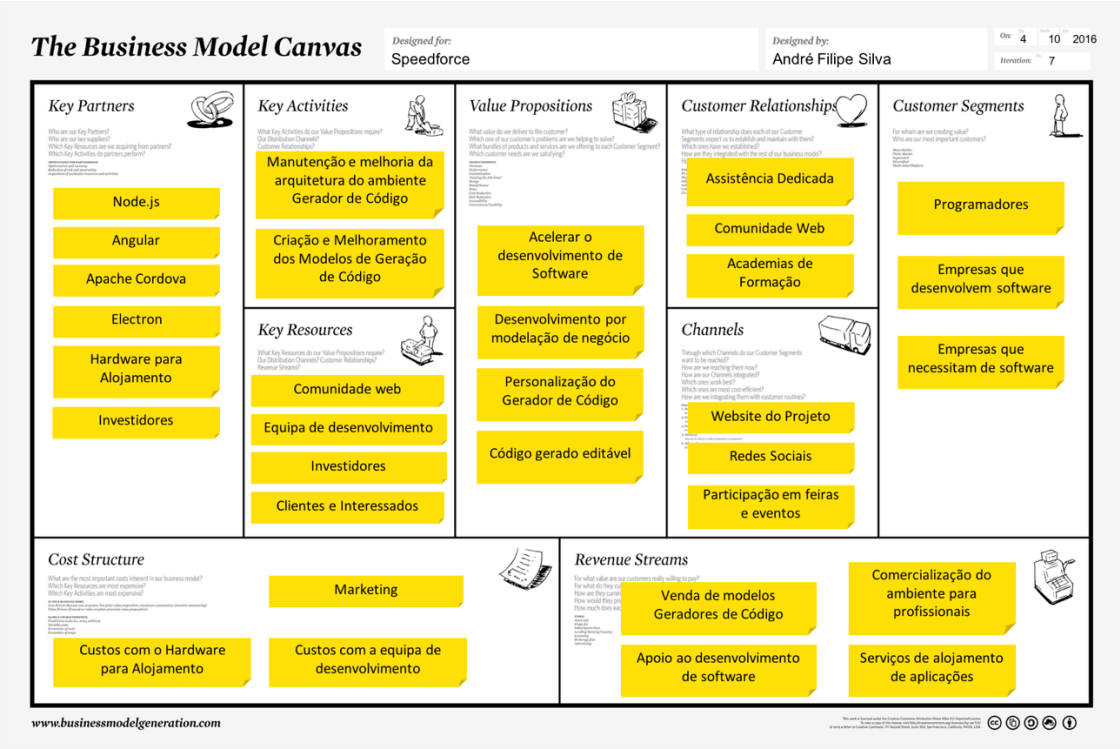


Figura 11 - Modelo de Negócio Canvas

4 Requisitos

Neste capítulo serão apresentados os requisitos para a solução a desenvolver, procurando especificar qual o seu âmbito de atuação, quais as suas funcionalidades, atores e casos de uso.

4.1 Enquadramento

Antes de avançar para apresentação de requisitos torna-se necessário explicar as razões para a criação dos mesmos, as suas prioridades e quais os atores intervenientes nos mesmos.

4.1.1 Missão

A missão da solução apresentada na abordagem proposta nesta tese consiste na aceleração do processo de desenvolvimento de software, procurando aumentar a velocidade com que um programador consegue desenvolver casos de uso de um modelo de negócio, não descuidando fatores como a performance, validade, segurança e qualidade do software produzido, e a capacidade de edição do código do mesmo. Esta aceleração é assim potenciada por um gerador de código com bases no modelo de negócio em uso, gerador este com independência da linguagem de código.

4.1.2 Atores e Sistemas externos

Este sistema é composto por apenas um único ator: o programador. O programador é o engenheiro de informático ou semelhante que é responsável pela implementação de um ou mais casos de uso de negócio funcional em código de software cujo o resultado é uma aplicação.

Independentemente do seu nível de conhecimento e experiência profissional, o programador será aquele que irá interagir em primeiro lugar com o sistema de forma a construir todos os casos de uso. Nas descrições dos casos de uso que se seguem o ator programador também é denominado de utilizador.

4.1.3 Prioridade de Requisitos

Para estabelecer a prioridade dos requisitos, foram adotadas as denominações: essencial, importante e desejável. A Tabela 2 apresenta a descrição de significado de cada uma dessas denominações.

Tabela 2 - Prioridade de Requisitos

Essencial	É o requisito sem o qual o sistema não entra em funcionamento. Requisitos essenciais são requisitos imprescindíveis que têm que ser implementados impreterivelmente.
Importante	É o requisito sem o qual o sistema entra em funcionamento, mas de forma não satisfatória. Requisitos importantes devem ser implementados, mas, se não forem, o sistema poderá ser implementado e utilizado ainda assim.
Desejável	É o requisito que não compromete as funcionalidades básicas do sistema, isto é, o sistema pode funcionar de forma satisfatória sem ele. Requisitos desejáveis são requisitos que podem ser deixados para versões posteriores do sistema, caso não haja tempo útil para implementá-los na corrente versão.

4.2 Funcionais

A Figura 12 apresenta o mapeamento dos requisitos funcionais para casos de uso ligados ao ator do sistema a implementar.

No mesmo é possível avistar 4 grupos de casos de uso: (i) Geração de Código, (ii) Edição e Execução, (iii) Gestão de Versões e (iv) Ferramentas.

O grupo de Geração de Código consiste num conjunto de casos de uso associados a todas as atividades que irão culminar na geração de código ou alteração de código já existente.

Correspondem à definição do negócio do projeto corrente e são a base de operação representada pelos requisitos funcionais essenciais.

O grupo de Edição e Execução é composto por um conjunto de casos de uso de edição e validação de código, podendo a mesma resultar na execução, *debug* e *deploy* do código. São na sua maioria casos de uso de prioridade importante.

O grupo de Gestão de versões corresponde a casos de uso associados ao uso de um sistema de gestão de versões dentro do ambiente, evitando assim que o ator tenha que sair do ambiente e utilizar outras aplicações para efetuar essa mesma gestão.

O grupo de Ferramentas consiste em casos de uso para gestão e execução de outras funcionalidades que não estão diretamente ligadas a nenhum dos três primeiros grupos referidos. São na sua maioria casos de uso de prioridade desejável.

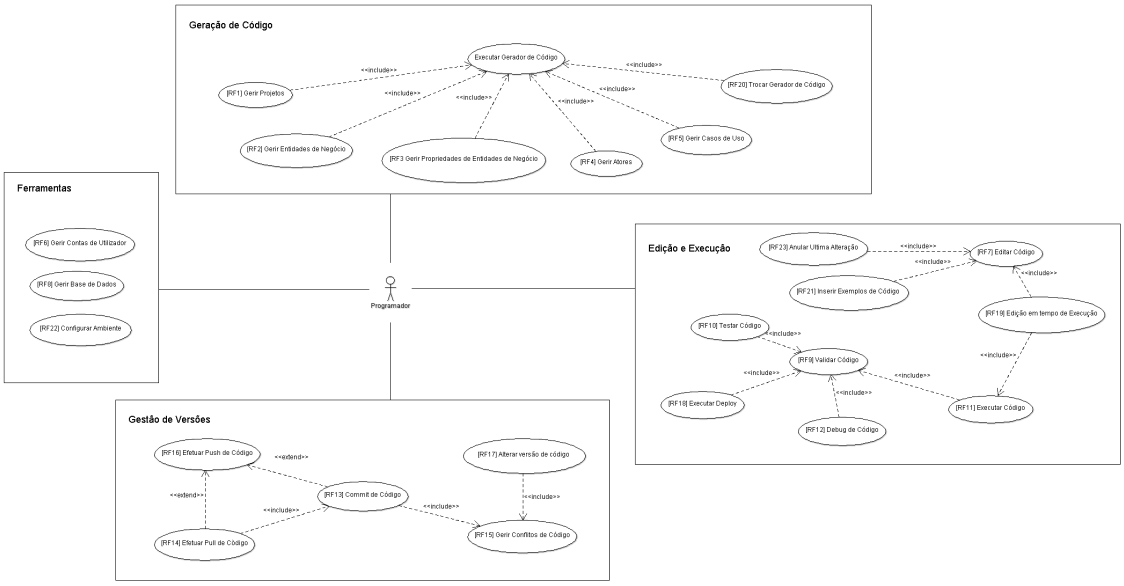


Figura 12 - Diagrama de Casos de Uso

4.2.1 [RF1] Gerir Projetos

Deverá ser possível criar, editar, listar, eliminar e importar projetos de software, pastas onde irão residir todos os ficheiros de código gerado e editado assim como os ficheiros do gerador em uso.

Prioridade: Essencial.

Entradas e pré-condições: A criação é feita através de um *wizard* com vários passos que variam consoante as informações introduzidas em passos anteriores. Na criação o utilizador deverá, num primeiro passo, indicar qual o nome do projeto, qual a descrição do mesmo, a localização de uma imagem para referência e a desejada localização no sistema para os ficheiros a criar. Num segundo passo de criação o utilizador deverá indicar quais os modelos geradores de código que pretende utilizar para cada camada do sistema a implementar. Por fim, num terceiro passo, serão apresentadas opções e funcionalidades especificas de cada um dos modelos geradores escolhidos. Para edição e eliminação o utilizador apenas terá de indicar qual o projeto que pretende trabalhar. Para a importação o utilizador deverá indicar a localização do projeto a importar.

Saídas e pós-condições: Perante a criação e edição de projetos o programador deverá ser levado ao ecrã de edição dos conteúdos do projeto. Perante a eliminação o sistema deverá mostrar ao utilizador uma janela de confirmação de eliminação e após essa confirmação deverá ser levado ao ecrã de listagem. Perante a importação deverá ser apresentada a interface de listagem, agora atualizado com a inclusão do projeto importado.

4.2.2 [RF2] Gerir Entidades de Negócio

Deverá ser possível criar, editar, listar, eliminar entidades de negócio.

Prioridade: Essencial.

Entradas e pré-condições: Na criação o utilizador deverá indicar qual o nome singular e plural da entidade de negócio, qual a descrição da mesma e as propriedades que compõem essa entidade.

Saídas e pós-condições: Perante a criação e edição de entidades será executada a geração de código correspondente, dando possivelmente origem a novas classes de software e entradas na base de dados. Perante a eliminação o sistema deverá mostrar ao utilizador uma janela de confirmação de eliminação e após essa confirmação todo o código gerado deverá ser removido. Todo o código gerado que dependia desta entidade de negócio será mantido. Caberá ao programador efetuar a correção de código necessária, sendo que a plataforma irá indicar uma lista de localizações das referências existentes para essa entidade.

4.2.3 [RF3] Gerir Propriedades de uma Entidade de Negócio

Deverá ser possível criar, editar, listar, eliminar propriedades de uma entidade de negócio.

Prioridade: Essencial.

Entradas e pré-condições: Na criação o utilizador deverá indicar o nome da propriedade, o seu tipo (de uma lista de tipos tradicionais pré-definida), se a mesma tem preenchimento obrigatório ou não e qual o valor por pré-definido.

Saídas e pós-condições: Perante a criação e edição de novas propriedades será executada a geração de código correspondente, dando possivelmente origem a alterações em classes de software e entradas na base de dados. Perante a eliminação o sistema deverá mostrar ao utilizador uma janela de confirmação de eliminação e após essa confirmação todo o código gerado deverá ser removido.

4.2.4 [RF4] Gerir Atores

Deverá ser possível criar, editar, listar, eliminar atores.

Prioridade: Essencial.

Entradas e pré-condições: Na criação o utilizador deverá indicar qual o nome do ator e uma descrição do mesmo.

Saídas e pós-condições: Perante a criação e edição de atores será executada a geração de código correspondente, dando possivelmente origem a novas classes de software. Perante a eliminação o sistema deverá mostrar ao utilizador uma janela de confirmação de eliminação e após essa confirmação todo o código gerado deverá ser removido.

4.2.5 [RF5] Gerir Casos de Uso

Deverá ser possível criar, editar, listar, eliminar casos de uso.

Prioridade: Essencial.

Entradas e pré-condições: Na criação o utilizador deverá indicar quais os atores que podem executar a ação, o tipo de caso de uso (Listagem, Criação, Edição, Remoção, Visualização ou

Personalizado), qual a principal entidade associada, uma descrição e se o caso de uso deverá ou não estar visível no menu da aplicação criada.

Saídas e pós-condições: Perante a criação e edição de casos de uso atores será executada a geração de código correspondente, dando possivelmente origem a novas classes de software para utilização do caso de uso e testes ao mesmo. Caso seja utilizado um ator que não o ator já existente por omissão "Anónimo" (representativo de todos os utilizadores não autenticados na aplicação criada) será executada a geração de um sistema de autenticação de utilizadores. Perante a eliminação o sistema deverá mostrar ao utilizador uma janela de confirmação de eliminação e após essa confirmação todo o código gerado deverá ser removido.

4.2.6 [RF6] Gerir Contas de Acesso para Utilizadores

Deverá ser possível criar, editar, listar, eliminar contas de acesso para utilizadores através da plataforma.

Prioridade: Importante.

Entradas e pré-condições: Na criação o utilizador deverá indicar qual o nome de utilizador para a conta, qual a password desejada e quais os papeis de ator que a conta terá acesso.

Saídas e pós-condições: Perante a criação e edição de casos de uso atores será executada a geração de código correspondente, dando possivelmente origem a novas entradas na base de dados. Perante a eliminação o sistema deverá mostrar ao utilizador uma janela de confirmação de eliminação e após essa confirmação as entradas em base de dados correspondentes deverão deixar de existir.

4.2.7 [RF7] Editar Código

Deverá ser possível editar o código gerado para *frontend*, *backend*, base de dados e camada de integração, assim como os próprios modelos de geração de código em uso no projeto, dentro do ambiente.

Prioridade: Importante.

Entradas e pré-condições: O utilizador deverá indicar qual o ficheiro gerado que pretende editar.

Saídas e pós-condições: O Ficheiro selecionado será aberto num editor de texto com realce de sintaxe para a linguagem de código em uso, caso aplicável. Caso haja uma posterior nova geração de código, o gerador não irá substituir o ficheiro criado, limitando-se apenas a acrescentar e eliminar as propriedades e métodos necessários.

4.2.8 [RF8] Gerir Base de dados

Deverá ser possível executar tarefas de gestão de base de dados como consultas, gestão de tabelas ou coleções, entidades e utilizadores.

Prioridade: Importante.

Entradas e pré-condições: O utilizador deverá selecionar a opção desejada na área de opções para base de dados.

Saídas e pós-condições: O Ficheiro selecionado será aberto num editor de texto com realce de sintaxe para a linguagem de código em uso, caso aplicável.

4.2.9 [RF9] Validar Código

Deverá ser possível validar o código escrito de forma a verificar se o mesmo está correto sintaticamente. O sistema irá solicitar ao modelo gerador que execute a compilação do código.

Prioridade: Importante.

Entradas e pré-condições: O utilizador deverá selecionar a opção de validar código ou aguardar que o sistema a execute automaticamente durante a edição de um ficheiro ou após geração de código.

Saídas e pós-condições: O resultado da validação deverá ser mostrado num painel auxiliar.

4.2.10 [RF10] Testar Código

Deverá ser possível executar os testes de código gerados, criados e editados de forma a validar se a aplicação cumpre os casos de uso implementados conforme esperado. O sistema irá solicitado ao modelo gerador que execute os testes.

Prioridade: Importante.

Entradas e pré-condições: O utilizador deverá selecionar a opção de executar testes, selecionar que camadas pretende testar e, para cada camada, quais os casos de uso a testar, ou, selecionar quais os casos de uso a testar, e para cada caso de uso, selecionar as camadas a testar.

Saídas e pós-condições: O resultado dos testes deverá ser mostrado num painel auxiliar.

4.2.11 [RF11] Executar Código

Deverá ser possível executar as diferentes camadas da aplicação. O sistema irá solicitar ao modelo gerador que execute as diferentes camadas na plataforma selecionada.

Prioridade: Importante.

Entradas e pré-condições: O utilizador deverá selecionar a opção de executar aplicação e selecionar que camadas pretende executar e em que plataforma pretende executar. As plataformas disponíveis serão aquelas indicadas pelos modelos geradores de código em uso.

Saídas e pós-condições: O resultado do pedido de execução deverá ser mostrado num painel auxiliar.

4.2.12 [RF12] Debug de Código

Deverá ser possível executar a aplicação em modo *debug*. O sistema irá solicitado ao modelo gerador que execute a aplicação em modo *debug*.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá selecionar a opção de executar aplicação em modo *debug* e selecionar que camadas pretende executar.

Saídas e pós-condições: O resultado do pedido de execução em modo *debug* deverá ser mostrado num painel auxiliar. A partir deste momento o ambiente ficará à escuta parar parar perante pontos de paragem no código ou possíveis erros. O painel auxiliar ficará também responsável por mostrar valores de variáveis em uso, assim como valores de uso de memória, consumo de bateria, etc.

4.2.13 [RF13] Integração para Commit de Código

Deverá ser possível executar *commit* de código para o repositório local, dentro do ambiente, executando a instrução perante o sistema de versões em uso no projeto corrente.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá selecionar a opção de *commit* de código e selecionar o código a guardar.

Saídas e pós-condições: O resultado do pedido de *commit* deverá ser mostrado num painel auxiliar. Caso o processo seja bem-sucedido e não existirem outros *commit* necessários deverá ser questionado ao utilizador se pretende efetuar *push* do código para o repositório remoto.

4.2.14 [RF14] Integração para Efetuar Pull de Código

Deverá ser possível obter o código presente no repositório remoto do sistema de controlo de versões em uso.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá selecionar a opção de *pull* de código, indicando o endereço do repositório remoto.

Saídas e pós-condições: O resultado do pedido de *pull* deverá ser mostrado num painel auxiliar. Caso existam conflitos o utilizador deverá ser levado ao ecrã de Gestão de Conflitos. Caso o processo seja bem-sucedido deverá ser questionado ao utilizador se pretende efetuar *push* do código para o repositório remoto.

4.2.15 [RF15] Integração para Gerir Conflitos de Código

Deverá ser possível gerir possíveis conflitos entre o código atual e o código obtido do comando de *pull* de código.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá resolver todos os conflitos existentes selecionando partes do código dos ficheiros locais ou dos ficheiros remotos ou então editando os ficheiros novamente.

Saídas e pós-condições: O resultado desta gestão deverá ser mostrado num painel auxiliar. Caso o processo seja bem-sucedido deverá ser questionado ao utilizador se pretende efetuar *push* do código para o repositório remoto.

4.2.16 [RF16] Integração para Efetuar Push de Código

Deverá ser possível enviar o código presente no repositório local para o repositório remoto do sistema de controlo de versões em uso.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá selecionar a opção de *push* de código, indicando o endereço do repositório remoto.

Saídas e pós-condições: O resultado do pedido de *push* deverá ser mostrado num painel auxiliar. Caso existam conflitos o utilizador deverá ser levado ao ecrã de *pull* de código e iniciar o processo novamente.

4.2.17 [RF17] Integração para Alterar versão de código

Deverá ser possível alterar a versão do código atual para uma das versões existentes no repositório local ou no repositório remoto do sistema de gestão de versões em uso.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá selecionar a opção *de alterar versão* de código e indicar qual a versão para a qual transitar. O utilizador poderá optar por uma troca completa ou efetuar uma fusão entre a versão atual e a versão pretendida.

Saídas e pós-condições: O resultado do pedido de alteração deverá ser mostrado num painel auxiliar. Caso existam conflitos o utilizador deverá ser levado ao ecrã de resolução de conflitos de código.

4.2.18 [RF18] Executar *Deploy*

Deverá ser possível efetuar *deployment* do código atual em edição para um dispositivo e/ou servidor. O sistema irá solicitar ao gerador de código a execução de *deploy* do código.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá selecionar a opção de *deploy*. Poderá selecionar quais as camadas, e, para cada camada, qual a plataforma para o qual deverá o processo ser executado. O utilizador terá também a hipótese de gerar executáveis para a aplicação para cada sistema operativo e publicação da aplicação em lojas de aplicações.

Saídas e pós-condições: O resultado deste processo deverá ser mostrado num painel auxiliar.

4.2.19 [RF19] Edição em Tempo de Execução

Deverá ser possível editar o código e visualizar o resultado das alterações durante a edição. O sistema deverá solicitar ao gerador de código a atualização da execução durante a edição dos ficheiros de código ou após geração de código.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá editar o código ou efetuar alguma atividade de geração de código enquanto a aplicação está em execução.

Saídas e pós-condições: O resultado deste processo deverá ser visível nas plataformas em que a aplicação está a ser executada.

4.2.20 [RF20] Trocar de Gerador de código

Deverá ser possível alterar o gerador de código em uso de uma determinada camada da aplicação. Uma vez que a lógica de negócio se encontra guardada no processo, o sistema irá solicitar a execução da geração de todo o código necessário no novo gerador em uso, com perda de possíveis edições efetuadas ao código anteriormente gerado.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá selecionar a opção de troca de gerador de código, selecionar qual a camada pretendida e qual o novo gerador a adotar.

Saídas e pós-condições: Novo código será gerado com o novo gerador em uso a partir do modelo de negócio configurado no projeto.

4.2.21 [RF21] Inserir Exemplos de código

Deverá ser possível inserir exemplos de código com o objetivo de execução de determinada funcionalidade.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá, durante a edição de um ficheiro de código, selecionar a um exemplo no painel de exemplos.

Saídas e pós-condições: O código do exemplo selecionado será inserido no local onde o cursor está presente no ficheiro de código em edição.

4.2.22 [RF22] Configurar Ambiente

Deverá ser possível configurar fatores na interface gráfica como o tema visual, as cores, os tamanhos e os tipos das fontes e espacejamentos e teclas de atalho.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá aceder à pagina de edição de configurações, efetuar as suas alterações e clicar no botão de guardar alterações.

Saídas e pós-condições: As alterações efetuadas deverão ser persistidas e as novas configurações devem tomar efeito imediato.

4.2.23 [RF23] Anular Última Alteração

Deverá ser possível anular a última alteração efetuada numa janela do editor de ficheiros da solução.

Prioridade: Desejável.

Entradas e pré-condições: O utilizador deverá clicar no botão de anulação ou utilizar o atalho de teclado predefinido para anulação.

Saídas e pós-condições: O conteúdo do ficheiro em edição deverá voltar para o estado anterior à última edição efetuada.

4.3 Não Funcionais

De forma a descrever os requisitos não funcionais da abordagem apresentada segue uma enumeração dos atributos de qualidade sobre a norma de qualidade de sofware ISO/IEC 9126[76].

4.3.1 Funcionalidade

[RNF1] [Essencial] O Sistema deve disponibilizar conjunto de ferramentas que permita ao programador de software executar a sua atividade de desenvolvimento de código;

[RNF2] [Essencial] O Sistema deverá gerar unicamente o código necessário para a tarefa solicitada pelo utilizador e nunca mais do que isso;

[RNF3] [Essencial] O Sistema deverá ter uma integração simples com as plataformas nas quais o código gerado poderá ser executado permitindo comunicação entre ambas;

[RNF4] [Importante] O Sistema deverá proteger a informação sobre projetos criados de outras aplicações em execução na mesma máquina em que o ambiente será executado;

[RNF5] [Desejável] Todas as atividades de geração de código deverão seguir o mesmo procedimento garantindo padronização.

4.3.2 Confiabilidade

[RNF6] [Importante] O sistema deverá ter uma rotina auxiliar de verificação de resposta da rotina principal de forma a garantir que em caso de falha a mesma ocorre de forma graciosa, informando o utilizador sobre a mesma;

[RNF7] [Importante] O sistema deverá continuar a operar mesmo em caso de falha da funcionalidade selecionada pelo utilizador que deu origem a essa mesma falha;

[RNF8] [Desejável] Em caso de falha que obrigue à paragem de execução o sistema deverá ser capaz de ao reiniciar recuperar o estado em que estava antes da falha.

[76] https://pt.wikipedia.org/wiki/ISO/IEC_9126

4.3.3 Usabilidade

[RNF9] [Importante] O sistema deverá orientar o utilizador de tal forma que o mesmo entender quais são as funcionalidades disponíveis;

[RNF10] [Importante] O sistema deverá estar organizado de forma a que o utilizador deverá ser capaz de criar um projeto e colocar o mesmo em execução sem apoio de qualquer outro utilizador;

[RNF11] [Importante] O sistema deverá ser tolerante com os erros do utilizador, corrigindo os mesmos automaticamente sempre que possível;

[RNF12] [Importante] O sistema deverá apresentar uma interface segundo os princípios de design mais recentes e atualizados, oferecendo assim uma experiência gráfica atrativa;

[RNF13] [Importante] O sistema deverá ter um tema com alto contraste para garantir acessibilidade por utilizadores com dificuldades.

[RNF14] [Desejável] O sistema deverá disponibilizar um botão de anulação da anterior ação de forma a evitar danos colaterais decorrentes de modificações erradamente introduzidas;

4.3.4 Desempenho

[RNF15] [Desejável] O sistema deverá responder a todas as ações do utilizador num prazo máximo de 500 milissegundos;

[RNF16] [Desejável] O sistema não deverá em momento algum ocupar mais de 200 Mb de memória RAM.

4.3.5 Manutenibilidade

[RNF17] [Desejável] Em caso de erro o sistema deverá apresentar um erro e um registo dos últimos métodos executados antes do erro;

[RNF18] [Importante] O sistema deverá apresentar uma interface para configuração de opções, permitindo modificar funcionalidades como o tema em uso, o volume dos sons emitidos e o envio de notificações;

[RNF19] [Importante] O sistema deverá estar implementado sobre módulos, de tal forma que seja possível testar o funcionamento de cada parte da funcionalidade de forma independente;

4.3.6 Portabilidade

[RNF20] [Essencial] O sistema deverá ser possível executar em sistemas operativos Windows, macOS e Linux de forma a se adaptar ao sistema operativo preferido pelo utilizador;

[RNF21] [Essencial] O sistema deverá possuir um assistente de instalação para facilitar o processo de configuração num novo sistema operativo;

[RNF22] [Desejável] O sistema não deverá depender de outros sistemas informáticos mas deverá permitir que certas atividades possam ser executadas noutros softwares se assim o utilizador o permitir;

5 Análise e Design

No presente capítulo apresenta-se a análise efetuada para o desenvolvimento da solução, assim como o design para concretizar os requisitos e necessidades encontradas. É assim apresentado o modelo de domínio, a arquitetura dos principais componentes e o design gráfico da solução.

5.1 Modelo de Domínio

No presente ponto apresenta-se o modelo de domínio para o problema em mãos. O principal objetivo deste é permitir a criação de projetos de software, configurando em cada um, atores, entidades de negócio e casos de uso, ilustrando assim todo um modelo de negócio de forma representativa.

A Figura 13 apresenta as principais entidades em uso na solução implementada. O sistema (representado por "System") contém uma biblioteca ("Library").

Esta biblioteca agrupa projetos ("Project"), entidades estas que representam as aplicações que são criadas com esta solução. Cada projeto define quais as camadas de software ("Software Layer") que o constituem, qual a localização do projeto e a referência para a imagem de capa representativa do projeto. Cada projeto tem ainda como funcionalidade sustentar a informação sobre o modelo de negócio do projeto, associando a si um conjunto de instâncias das entidades: (i) ator ("Actor"), (ii) entidade de negócio ("Entity") e (iii) caso de uso ("UseCase").

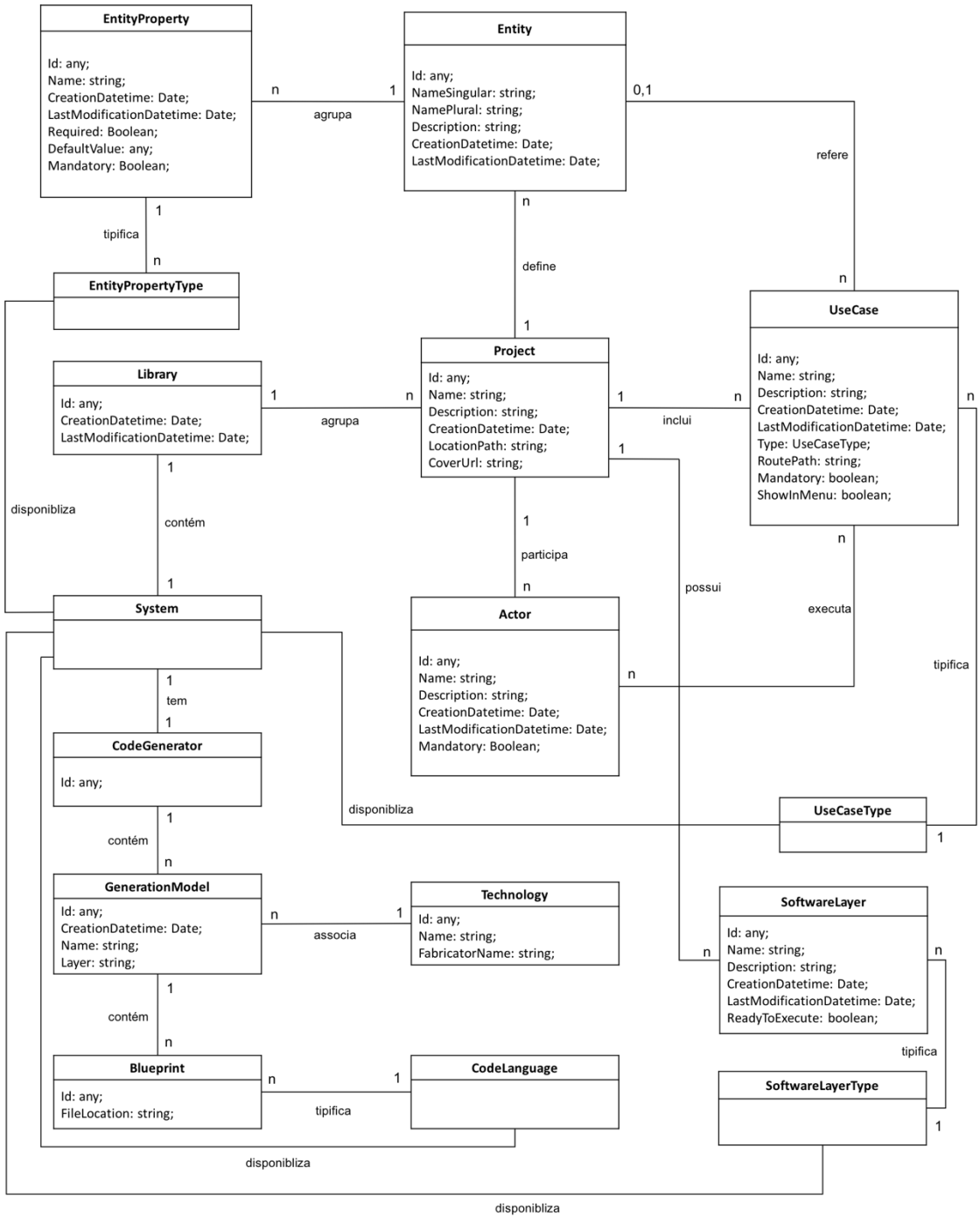


Figura 13 - Modelo de Domínio da Solução

A entidade Ator (“Actor”) representa um ator no projeto em criação e possui propriedades descritivas do mesmo. Possui também uma propriedade para identificar se é ou não um ator obrigatório perante o sistema (“Mandatory”). Esta definição tem como objetivo permitir configurar instâncias que não são possíveis eliminar devido à sua natureza de importância no

sistema. É o caso dos atores “Anónimos” e “Registados”, sendo que o primeiro representa todos os utilizadores não autenticados perante a aplicação e a segunda todos os utilizadores autenticados perante a aplicação.

A Entidade de negócio (“Entity”) representa entidades do sistema a desenvolver. Estas entidades estão ligadas ao negócio que irá ser implementado em cada novo projeto de tal forma que o definem. Possuem um conjunto de propriedades (definidas como instâncias de “EntityProperty”).

As propriedades de uma entidade (“EntityProperty”) tem como objetivo definir a entidade de negócio, enumerando quais os campos que pertencem à mesma, quais os seus tipos, valores por defeito e se são ou não obrigatórias definir para criar instâncias da entidade que representam. De forma a definir um tipo de uma propriedade existe uma ligação a um enumerado de tipos de propriedades de entidades (“EntityPropertyType”).

O enumerado de tipos de propriedades poderá ter os valores:

- “EntityIdentifier” (identificador de objeto);
- “Text” (texto);
- “Integer” (número inteiro);
- “LongInteger” (número inteiro de maior dimensão);
- “Decimal” (número decimal);
- “Boolean” (valor booleano) ;
- “DateTime” (data e hora);
- “Date” (data);
- “Time” (hora);
- “PhoneNumber” (número de telefone);
- “Email” (endereço de email);
- “BinaryData” (informação binária);
- “Currency” (moeda);
- “FolderLocation” (localização de uma pasta);
- “FileLocation” (localização de um ficheiro);
- “List” (lista de objetos);
- “Password“ (palavra-passe).

Por último na definição do modelo de negócio nos ficheiros de projeto existe a representação de casos de uso ("UseCase"). Estas entidades referenciam um conjunto de atores em associação a uma ou nenhuma entidade do modelo de negócio. Os casos de uso definem um tipo representado numa enumeração de tipos de casos de uso ("UseCaseType").

A solução suporta os seguintes tipos de casos de uso:

- "List" (Listagem de instâncias);
- "Create" (Criação de uma nova instância);
- "Read" (Leitura/listagem de detalhes de uma instância);
- "Update" (Atualização/edição de uma instância);
- "Delete" (Remoção de uma instância);
- "Custom" (Qualquer outro tipo de ação que não corresponda aos anteriores);

As camadas de software de um projeto dizem respeito às unidades que compõem o mesmo. Durante o processo de criação do projeto o utilizador terá a liberdade de selecionar quais as camadas que pretende incluir, assim como o modelo gerador de código que irá gerar cada uma dessas camadas. Cada camada de software é tipificada pelo enumerado de tipos de camadas de software ("SoftwareLayerType"). Estas entidades contêm também uma propriedade para confirmação se o processo de configuração para execução está terminado. Isto permite que, para geradores com processos de geração inicial de código mais demorados, o utilizador possa editar o código enquanto as necessidades para execução do projeto são satisfeitas.

Os tipos de camadas de software ("SoftwareLayerType") presentes nesta versão da solução são:

- *frontend* (interfaces gráficas);
- *backend* (servidores/interfaces de programação de aplicações);
- base de dados;
- integração com outros sistemas.

A definição de camadas de software e a sua tipificação estão ligadas à existência e seleção de modelos geradores de código. O sistema possui então um gerador de código ("CodeGenerator"). Este gerador de código contém instâncias da entidade modelo gerador de código ("GenerationModel").

Cada modelo de gerador de código é tipificado por um tipo de camada de software de forma a identificar qual o tipo de camada que poderá gerar. Uma vez que o modelo gerador de código é replicado para o projeto em edição de forma a permitir personalização do editor perante esse mesmo projeto, cada modelo é replicado e associado à camada de software para a qual irá gerar conteúdos. Esta entidade tem a si associada uma tecnologia ("Technology"), como por exemplo: "AngularJS", "ReactJS", "Microsoft ASP.NET", etc. Um modelo gerador de código contém um conjunto de entidades modelo de ficheiro ("Blueprint").

Um modelo de ficheiro consiste numa estrutura e/ou ficheiro que tem o objetivo de ser replicado e adaptado a funcionalidade em causa. É exemplo um ficheiro de texto com uma classe de software para gerar um controlador para uma entidade, designada de "EntityController". Para criar uma classe de software controladora para entidade "Person", o gerador de código irá usar esta instância e gerar a classe de software "PersonController". Cada modelo é tipificado por uma linguagem de código ("CodeLanguage").

A solução suporta várias linguagens de código. Uma lista completa das mesmas está presente no anexo 10.1. O suporte de linguagens é o correspondente ao suporte do editor que foi integrado na solução.

O modelo de domínio apresentado corresponde à base necessária para suportar a geração de código e manutenção de projetos sobre a alçada dos modelos geradores.

5.2 Arquitetura da Solução

Sendo esta aplicação um ambiente com o objetivo de ser executado como uma aplicação *desktop*, a mesma apresenta uma arquitetura simples e minimalista, com foco na compatibilidade multiplataforma e performance. Analisa-se assim então com algum detalhe a arquitetura da solução.

5.2.1 Vista de Instalação

O diagrama de instalação em UML da Figura 14 tem o objetivo de mapear os artefactos desta solução que compõem a mesma.

Figura 14 - Diagrama de Instalação da Solução

A apresenta a visão de instalação da solução concretizada, mostrando que a aplicação será instalada e executada a partir do computador do utilizador final. De forma a que a aplicação possa ser executada sobre plataformas Linux, macOS e Windows, o ambiente de execução será a tecnologia Electron que emula um navegador de web nativo de cada sistema numa aplicação desktop. A aplicação desenvolvida em Angular 2.0 é executada dentro do ambiente de execução selecionado.

5.2.2 Vista Lógica

O âmbito da vista lógica é apresentar a funcionalidade que o sistema disponibiliza aos seus utilizadores. Para isso, na presente subsecção são apresentados diagramas UML como os diagramas de sequência.

5.2.2.1 Gerador de Código

O gerador de código é a estrutura mais importante da solução a criar. É através desta estrutura que a definição do modelo de negócio por parte do utilizador irá resultar na geração de código de software, promovendo assim aceleração do processo de desenvolvimento.

Para que este gerador possa ser escalável e personalizável e não apenas obter o mesmo funcionamento que uma plataforma RAD, em vez de possuir um conjunto de linguagens e tecnologias já pré-definidas e não alteráveis, este gerador faz uso de modelos de geração de código. O utilizador poderá então selecionar esses modelos entre os incluídos na solução. O objetivo é tornar a criação e importação de modelos de geração de código algo simples de forma a promover o desenvolvimento dos mesmos pela comunidade que apoia e usa esta plataforma, aumentando assim o catálogo de modelos disponíveis.

Para que o gerador se possa adaptar a modelos que possivelmente serão completamente diferentes sem que hajam alterações do código da plataforma, existe uma interface de programação de aplicações, interface essa que todos os modelos terão de implementar. Esta interface consiste numa implementação aproximada do padrão de software *Adapter*[77], padrão estrutural GoF[78]. Define-se esta implementação como "aproximada" uma vez que não consiste na tradicional implementação de classes de software mas sim na presença de respostas a um conjunto de pedidos definidos. Em vez da tradicional especificação de todos os métodos declarados na classe de interface, os modelos geradores de código incluem uma serie de scripts, definidos.

Definindo, o gerador de código irá evocar scripts específicos para tarefas especificas como por exemplo: (i) criar ator (requisito [RF4] Gerir Atores), (ii) criar entidade de negócio (requisito [RF2] Gerir Entidades de Negócio), (iii) criar caso de uso (requisito [RF5] Gerir Casos de Uso), etc. Estes scripts estão presentes dentro de cada modelo gerador de código, e a sua localização especifica deverá estar definida num ficheiro de configurações de cada um desses modelos. O gerador de código faz a leitura do ficheiro de configurações do modelo, e, durante a utilização, evoca o script correto para a tarefa. Os scripts necessários estarão definidos na definição da interface de programação de aplicações para modelos de geração de código.

Estes scripts estão definidos em JavaScript, sobre a tecnologia Node.js visto que esta tecnologia é a aquela que potencia todo o ambiente, mantendo assim conformidade com toda a plataforma. Cada script irá receber um conjunto de parâmetros definidos na sua evocação e será o responsável por gerar e alterar todo o código necessário para dar vida à instrução, seja por cópia de ficheiros de exemplo, seja por escrita integral do mesmo. Esta abordagem irá fazer

[77] https://pt.wikipedia.org/wiki/Adapter
[78] https://en.wikipedia.org/wiki/Design_Patterns

com que exista toda a liberdade possível para criação. Com isto espera-se poder ver modelos geradores cada vez com melhor performance e melhor eficiência.

De forma a acelerar o processo entre a criação de uma nova entrada de modelo de negócio e a continuação do desenvolvimento, todo o processo é executado de forma assíncrona com posterior notificação ao utilizador do termino do mesmo. Todo o processo é descrito pela Figura 15, no exemplo de criação de um novo ator. Após a ação do utilizador para criação de um novo ator o gerador verifica quais as camadas de software existentes no projeto em edição. Para cada camada existente é evocado o script "CreateActor.js" para geração de código para a ação desejada. De imediato após a evocação dos scripts o utilizador é informado que o ator foi criado com sucesso e que a geração de código teve inicio. Após todos os scripts terminarem a sua execução, o ambiente irá informar o utilizador do novo estado.

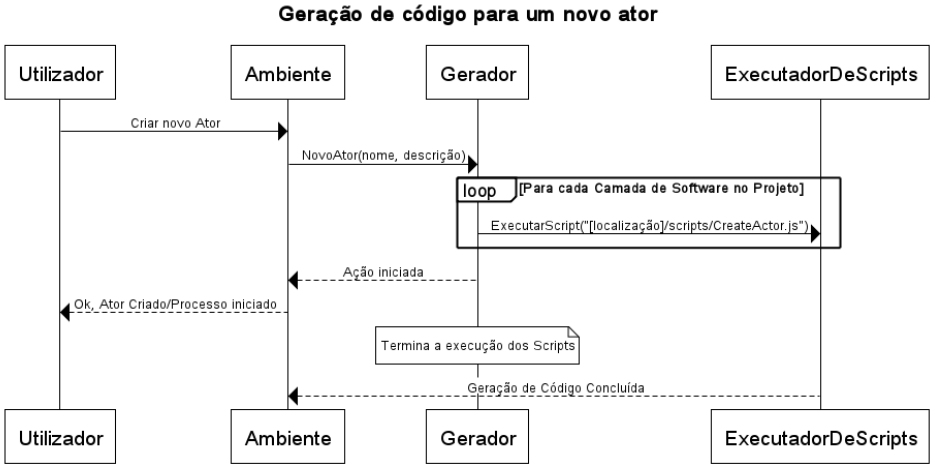


Figura 15 - Diagrama de Sequência para Criação de um Novo Ator

5.2.2.2 Modelo de Geração de Código

Os modelos de geração são um conceito criado com um objetivo principal: permitir o desenvolvimento e inclusão de novos modelos em tempo de execução. Pretende-se dar grande enfase à geração de código, criar cada vez mais e melhores geradores, com melhor performance e eficiência. Pretende-se criar geradores para várias tecnologias, por exemplo, desde interfaces web em HTML e JavaScript até aplicações nativas Android em JAVA, que permitam criar camadas de *frontend* ou *backend* ou base de dados ou integração com outros softwares. Será possível à comunidade criar esses mesmos modelos de forma a que se possa acelerar o desenvolvimento e aumentar qualidade destes.

Para que isto seja possível, cada modelo deverá implementar uma estrutura definida pelo gerador de código, definindo (i) um conjunto de recursos próprios consoante as suas necessidades, (ii) um conjunto de scripts para responder à interface mencionada e (iii) um ficheiro de configurações que irá representar o modelo perante o gerador.

Conforme indicado na Figura 16 é incluída uma pasta com o nome "resources". Esta pasta destina-se a conter tudo aquilo que for entendido pelo criador do modelo em termos de ficheiros de texto e media necessários para o correto funcionamento e operação do modelo. O expectável será encontrar nesta pasta ficheiros de texto modelos para criação, ou seja, *blueprints* para a geração de código e todos os restantes ficheiros necessários para criar um projeto funcional. Caberá ao modelo de geração decidir e operar se, ao executar geração de código, irá manter ou descartar as alterações efetuadas pelo utilizador ao código do gerado do projeto.

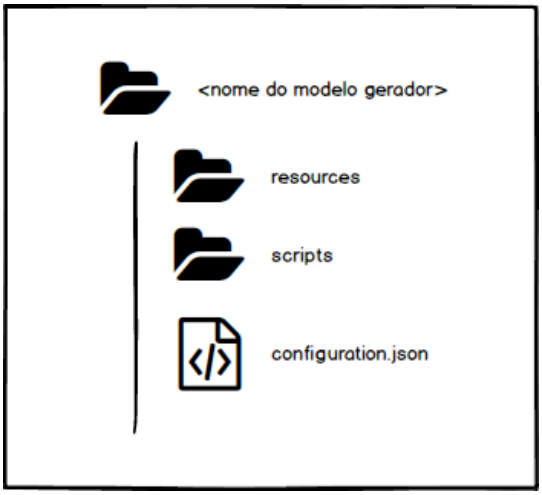


Figura 16 - Estrutura de Pastas de um Modelo Gerador de Código

Para que o gerador de código possa funcionar conforme esperado, cada modelo de geração deverá implementar um conjunto de scripts conforme o descrito na Tabela 3. Estes scripts irão residir na pasta scripts e deverão na sua execução retornar 0 caso a sua execução seja bem-sucedida, ou em caso de erro, um código numérico de erro.

Tabela 3 - Descrição da Interface para Modelos Geradores de Código - Versão 1.0

Nome do Script	**Parâmetros de Entrada**	**Funcionalidade a Concretizar**
CreateProject.js	[nome do projeto, localização disponibilizada pelo gerador, modo]	Criar ou substituir um projeto do modo indicado, com o nome indicado e na localização disponibilizada. Este script é evocado na criação do projeto.
PrepareProject.js		Executar todas e quaisquer atividades de preparação do projeto de forma a que o mesmo possa ser executado. Neste âmbito encaixam-se todas as ações que necessitam de um tempo de execução maior que assim sendo não podem estar contidas no "CreateProject.js". Este script é evocado no momento em que o utilizador entra no editor. Quando terminar, informa o utilizador que já poderá executar o código.
CreateBusinessEntity.js	[nome singular, nome plural, descrição]	Cria ou substitui todo o código necessário para sustentar e testar uma entidade de negócio perante o sistema. Evocado quando criada/modificada uma entidade de negócio.
CreateBusinessEntityProperty.js	[nome, tipo, obrigatória, valor por defeito]	Cria ou substitui todo o código necessário para sustentar e testar uma propriedade de uma entidade de negócio perante o sistema. Evocado quando criada/modificada

		uma propriedade de uma entidade de negócio.
CreateActor.js	[nome, descrição]	Cria ou substitui todo o código necessário para sustentar e testar um ator perante o sistema. Evocado quando criado/modificado um ator.
CreateUseCase.js	[lista de atores (identificados pelo nome), tipo, entidade de negócio associada, descrição, visibilidade em menu]	Cria ou substitui todo o código necessário para sustentar e testar uma entidade de negócio perante o sistema. Evocado quando criado/modificado um caso de uso.
CreateAccessAccount.js	[lista de atores (identificados pelo nome), nome de utilizador, password]	Cria ou substitui todo o código necessário para sustentar e testar uma conta de acesso ao sistema. Evocado quando criada/modificada uma conta de acesso.
ValidateCode.js		Executar tarefas de validação e/ou compilação de código. Evocado quando pedido pelo utilizador.
TestCode.js		Executar testes unitários ao código. Evocado quando pedido pelo utilizador.
ExecuteCode.js		Executar o código. Evocado quando pedido pelo utilizador.
DebugCode.js		Executar o código em modo *debug*. Evocado quando pedido pelo utilizador.

ExecuteDeploy.js	[localização do *deploy*]	Executar *deploy* do código para uma localização identificada. Evocado quando pedido pelo utilizador.
InsertSnippet.js	[código identificador do exemplo de código a inserir, localização do ficheiro, numero da linha onde inserir]	Executar a inserção de um exemplo de código no ficheiro indicado, na linha indicada. Evocado quando pedido pelo utilizador.

Todos os códigos de erro possíveis devem estar declarados também no ficheiro de configurações para que seja possível ao gerador mostrar alguma informação/explicação ao utilizador do ambiente.

O ficheiro de configurações do modelo gerador, ilustrado na Figura 17, é o seu representativo perante o gerador de código. É neste ficheiro que estão definidos os caminhos para os scripts de resposta aos pedidos de geração, dados específicos do modelo e ainda os caminhos para os recursos usados para a geração de código, de tal forma que o ambiente possa indicar ao utilizador que ficheiros poderá alterar para personalizar o modelo. A apresenta as principais chaves e tipos de valores esperados.

```
{
  "Name": "Model1", //Nome do Modelo
  "TechInUse": "C#", //Linguagem/Tecnologia em uso neste modelo
  "Description": "This is the Model1 code generation model based on the C#", //Descrição do modelo
  "Image": "images/cover.png", //Caminho para uma imagem representativa do modelo
  "Type": "frontend", //Tipo de modelo. Tipos existentes: "frontend", "backend", "database" e "integration"
  "Creator": "Model1 Inc.", //Nome da entidade criadora deste modelo
  "License": "Open-source", //Licença de operação com este modelo
```

```
  "Version": "1.0.0", //Versão do modelo
 "GenerationBlueprintsMap": {
   "CreateActor": ["actor/blueprint.model.cs", "actor/blueprint.service.cs"], //Diretórios de
localização dos ficheiros para geração de código
   (...) //Restantes mapeamentos para os restantes scripts
 },
 "ErrorCodes": { //Lista de códigos de erro possiveis na execução de scripts
   "1": "Could no complete task due to File Missing. Please Reinstall Model",
   "2": "Task Timeout",
   "Default": "Could not figure it out what happenned. Please call support"
 }
}
```

Figura 17 - Estrutura do Ficheiro de Configurações do Modelo Gerador de Código

Um dos fatores que distingue este ambiente de todos aqueles que foram referidos no estado da arte deste documento é a possibilidade de edição do gerador de código em uso. Tal é possível na medida em que, ao criar um projeto, é efetuada uma cópia do gerador selecionado para dentro da pasta de projeto. A partir desse momento todas as atividades que resultem em geração de código irão utilizador a cópia do gerador presente dentro do projeto. Isto permite ao programador alterar o gerador consoante as necessidades.

Para melhor entendimento de como funcionam os scripts de um modelo gerador, apresenta-se na Figura 18 o processo exemplificado. Este processo consiste num caso simples de cópia de uma *blueprint* com inserção do nome e descrição do ator recebidos por parâmetro por substituição com retorno do código de sucesso em caso do mesmo. Diferentes geradores poderão ter execuções completamente diferentes. O presente exemplo apenas demonstra uma possibilidade.

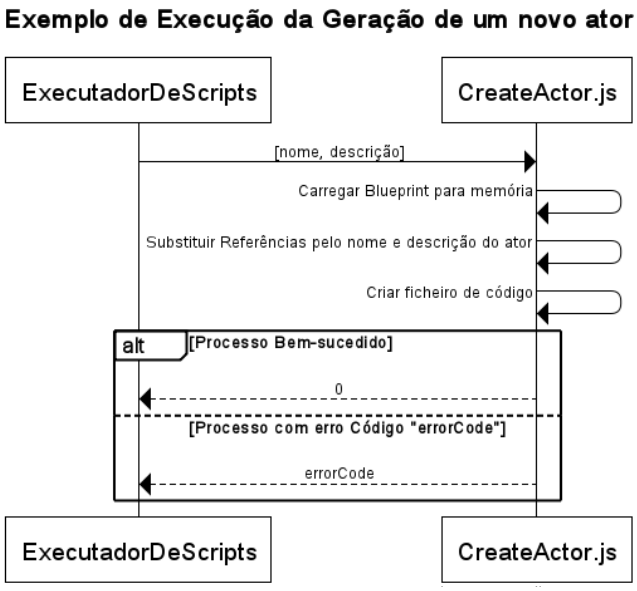


Figura 18 - Diagrama de Sequência de Exemplo de Execução da Geração de um Novo Ator

5.3 Interface Gráfica

A interface gráfica é uma componente essencial da abordagem apresentada. Para além de ser onde o ator deste sistema terá acesso a todas as funcionalidades, este ambiente será onde o programador passará mais tempo por dia. Assim, há a preocupação de oferecer uma interface onde o utilizador se sinta confortável e em controlo de todas as atividades. De forma a alcançar estas necessidades traçaram-se os seguintes objetivos: (i) personalização da interface e (ii) assistência ao desenvolvimento.

A personalização da interface é a concretização do requisito [RF22] Configurar Ambiente. Consiste em criar uma página em que o utilizador possa configurar variáveis como o tema visual, as cores, os tamanhos e os tipos das fontes, de forma a preparar o ambiente a seu gosto procurando conforto visual.

A assistência ao desenvolvimento consiste num conjunto de funcionalidades que apoiem o utilizador nas suas tarefas de desenvolvimento. É a concretização dos requisitos:

- [RF7] Editar Código
- [RF8] Gerir Base de dados
- [RF9] Validar Código
- [RF10] Testar Código

- [RF11] Executar Código
- [RF12] Debug de Código
- [RF13] Integração para Commit de Código
- [RF14] Integração para Efetuar Pull de Código
- [RF15] Integração para Gerir Conflitos de Código
- [RF16] Integração para Efetuar Push de Código
- [RF17] Integração para Alterar versão de código
- [RF19] Edição em Tempo de Execução
- [RF21] Inserir Exemplos de código
- [RF23] Anular Última Alteração

5.3.1 Ecrã inicial

O ecrã inicial da aplicação é composto por atalhos e ecrãs para acesso à gestão de projetos, gestão de modelos geradores de código e gestão de definições do ambiente. Baseada num sistema de separadores esta interface gráfica permite desenvolver várias atividades em simultâneo.

A Figura 19 - Ecrã Inicial da Aplicação mostra três áreas fundamentais: (i) painel de opções (área 1), (ii) barra de separadores (área 2) e (iii) listagem de projetos (área 3).

Na figura, à esquerda, destaca-se o painel de opções (área 1) com as principais funcionalidades para gestão de projetos e acesso às definições e menus de ajuda. Um clique num dos itens deste painel irá abrir um novo separador (exceto situações como os botões de importação que irão abrir o navegador de ficheiros do sistema). Este processo de abertura de um novo separador ocorre também com todas as ações que envolvam *wizards*.

A barra de separadores (área 2) no topo do ecrã é uma adição muito semelhante àquilo que é possível encontrar hoje em navegadores de internet e alguns editores de texto mais recentes. O objetivo é permitir ao utilizador executar várias operações em simultâneo e facilitar a transição entre diferentes conteúdos, precavendo assim possíveis necessidades e diminuindo o numero de cliques necessários.

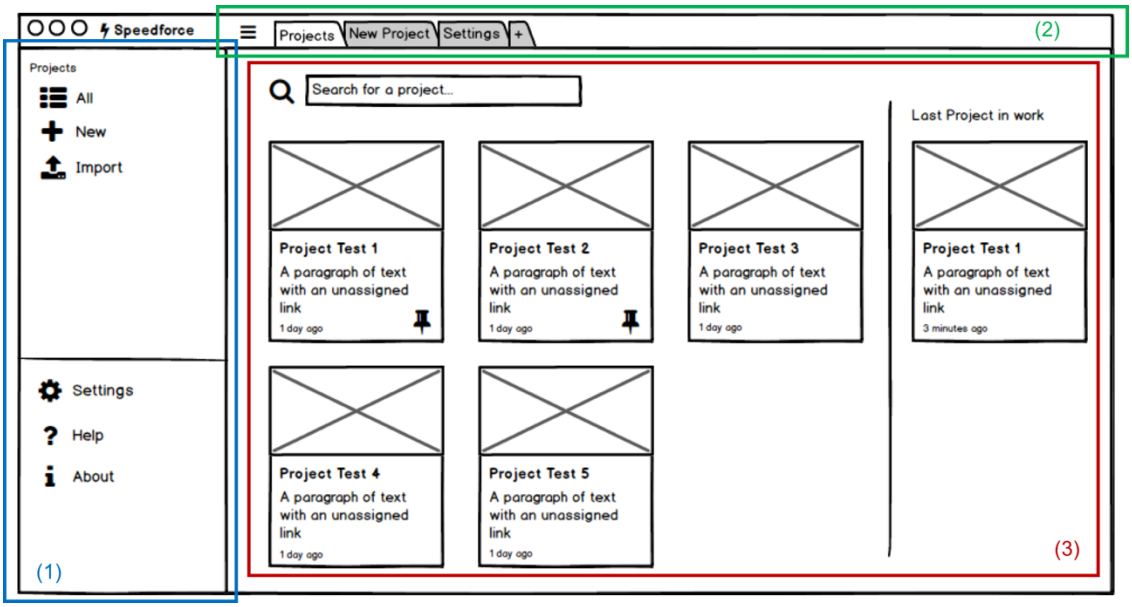


Figura 19 - Ecrã Inicial da Aplicação

A listagem de projetos (área 3) apresentada consiste numa apresentação dos itens mais recentemente modificados. Cada projeto criado ou aberto na plataforma irá ser incluído nesta listagem para acesso rápido. Através de uma caixa de pesquisa será possível filtrar projetos por nome ou descrição. Existe uma opção de afixamento de um determinado projeto ao inicio do ecrã, facilitando assim o acesso a projetos usados mais recorrentemente. Por fim, mais à direita, é apresentada uma listagem inteligente de forma a mostrar os projetos mais recentemente modificados e importados, permitindo que o utilizador possa de forma rápida recuperar o estado anterior ao fecho da aplicação.

5.3.2 Editor

O editor de código, representado na Figura 20, é o ecrã principal na medida em que representa o espaço onde o utilizador irá executar a maior parte dos casos de uso definidos. Foram tidas em conta preocupações como o design simplificado com possibilidade de arrumação de painéis, execução de várias atividades em simultâneo e disposição de ferramentas para que o utilizador se sinta sempre em controlo.

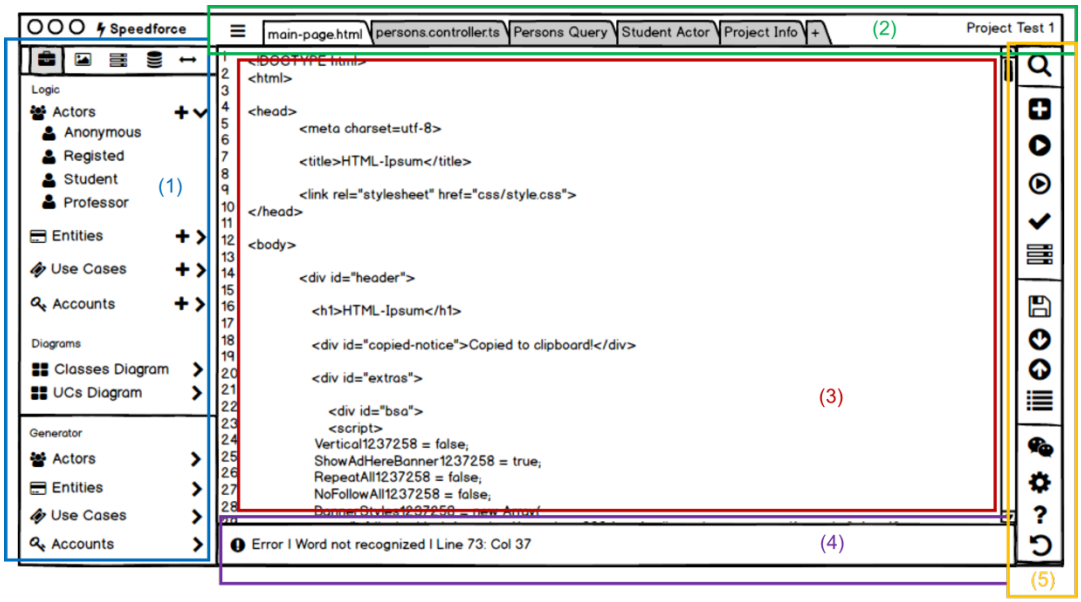


Figura 20 - Vista Principal do Editor de Código

A figura apresenta aquela que será a vista do utilizador ao executar a abertura de um projeto, ou seja, executar a edição do mesmo. Este ecrã é composto por cinco secções principais: (i) árvore de itens (área 1), (ii) barra de separadores (área 2), (iii) zona de edição (área 3), (iv) zona de validação (área 4) e (v) painel de instrumentos (área 5).

A árvore de itens é a área mais à esquerda onde serão enumeradas todas as entidades editáveis na plataforma. Por norma um clique num item desta área irá resultar na abertura desse mesmo item como um separador existente de forma temporária. Este separador é fechado assim que exista um clique numa outra área que não a área de edição. Um duplo clique irá abrir um separador para edição do item em causa de forma permanente que apenas será fechado após clique no botão de fecho do próprio separador.

Um pormenor a destacar é o seletor de camada a trabalhar, demonstrado na Figura 21. Esta área é composta pelos itens (da esquerda para a direita): (i) Gestão de Negócio, (ii) *frontend*, (iii) *backend*, (iv) base de dados e (v) camada de integração. Cada um destes botões irá transformar a árvore de itens mostrando os itens relacionados com essa mesma área.

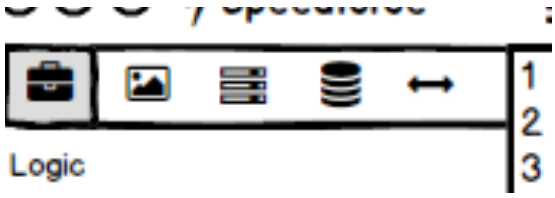


Figura 21 - Detalhe do Seletor de Camada

Neste ecrã é apresentado o nome da aplicação à esquerda e imediatamente a seguir um botão para mostrar/ocultar a árvore itens, alternando entre a vista na Figura 20 e a vista na Figura 22. Após os separadores é possível encontrar um botão de adicionar novo separador. Este último tem como funcionalidade abrir um novo separador com atalhos para acesso rápido aos itens mais usados recentemente, assim como reabrir separadores recentemente fechados. No canto mais à direita é apresentado o nome do projeto atual em edição, elemento este que funciona também como um botão para voltar ao ecrã inicial da aplicação.

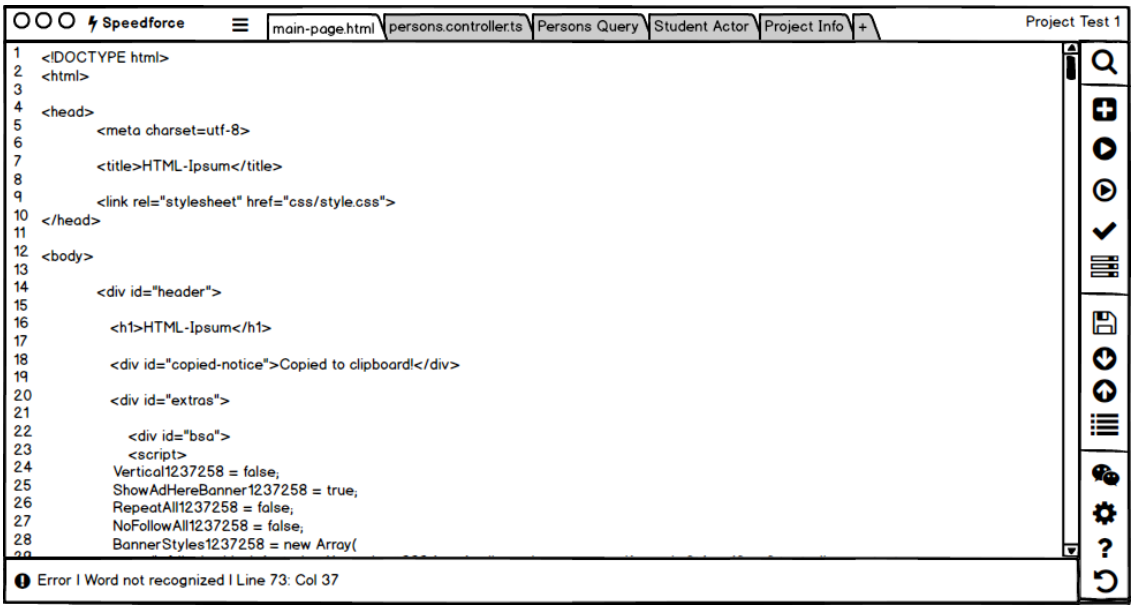


Figura 22 - Editor com Árvore de Itens Ocultada

A zona de edição é um editor de texto para edição de código da aplicação. Este editor possuiu funcionalidades como completar e sugerir automaticamente código/termos. O utilizador poderá alterar o design gráfico deste editor através das configurações da plataforma.

A zona de validação tem como objetivo reportar ao utilizador erros no código. Esta área é atualizada sempre que código é editado ou gerado. Na existência de erros os mesmos são mostrados indicando a mensagem de erro, o ficheiro em que ocorrem, a linha e a coluna desse

ficheiro. Ao clicar nesta informação é aberto um separador com esse ficheiro e o cursor posicionado na linha e coluna indicadas no erro, caso não seja o ficheiro aberto atualmente. Neste caso apenas o cursor será posicionado na linha e colunas referentes.

Por último surge a zona de painel de instrumentos. Nesta zona estão disponíveis botões para acesso às funcionalidades de validação (requisito [RF9] Validar Código) e execução de código (requisito [RF11] Executar Código), assim como às funcionalidades do sistema de gestão de versões (todos os requisitos desde [RF13] Integração para Commit de Código até [RF17] Integração para Alterar versão de código).

A Figura 23 ilustra um painel auxiliar que é aberto perante a funcionalidade de alguns dos botões do painel de instrumentos, com o objetivo de apresentar estados e resultados de procedimentos. Nesta figura, este painel localiza-se junto ao painel de instrumentos e apresenta o estado de execução do código atual nas diversas camadas que compõem o projeto em desenvolvimento.

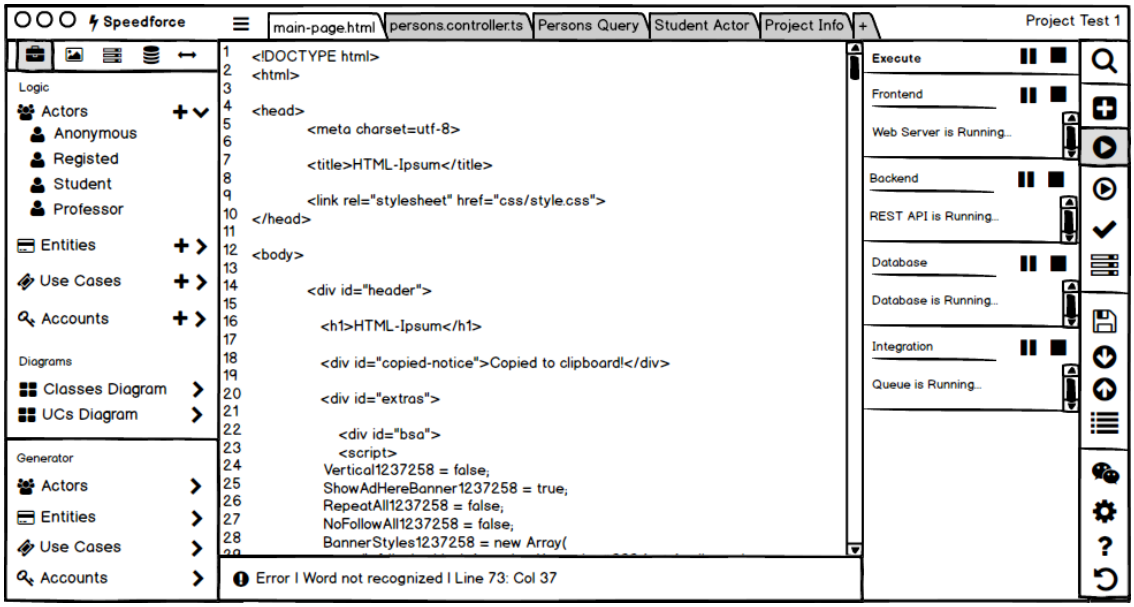


Figura 23 - Editor com Painel Auxiliar do Painel de Instrumentos Aberto

.

6 Implementação e Testes

Neste capítulo faz-se uma abordagem no domínio da implementação da solução abordada. Especifica-se aqui que tecnologias foram utilizadas, como foi estruturada a aplicação e que testes serão executados para garantir o cumprimento dos requisitos anteriormente descritos.

6.1 Tecnologia em Uso

De forma a cumprir os objetivos a que esta tese se propõe existiu um esforço de reunião das tecnologias mais recentes de mercado, procurando combinar tendências e motivações, criando assim uma solução que consiga reter o melhor dessas mesmas tecnologias.

6.1.1 Ambiente de Desenvolvimento

De forma a conseguir criar um ambiente de desenvolvimento multiplataforma fez-se uso da tecnologia Electron. Assim tornou-se possível com apenas um só código (web) criar aplicações nativas para sistemas operativos Linux, macOS e Windows.

O código do ambiente foi desenvolvido em tecnologia web de forma a ser possível integrar o mesmo com a tecnologia Electron. Este código foi construído sobre a *framework* Angular 2.0 escrito com TypeScript, escolha feita no seguimento da vontade de utilizar tecnologias em tendência.

Para os ecrãs de edição de ficheiros de código são baseados na biblioteca "Ace"[83], uma tecnologia para criação de editores de código em ambientes web. A escolha desta biblioteca foi realizada com base na análise comparativa[84] efetuada na Wikipédia onde, com alguma claridade, é possível verificar que se trata de uma das mais utilizadas, sendo open-source e suportada pela comunidade GitHub.

Uma especificidade é interface de gestão da camada de base de dados. Esta camada fornecesse acesso a bases de dados, permitindo tarefas como consulta, edição e remoção de dados, através de uma interface gráfica. A implementação desta funcionalidade é baseada no conceito criado para a biblioteca "adminMongo" e efetua a implementação de um padrão *Adapter* de forma a procurar garantir que torna possível conectar com várias bases de dados de uma forma devidamente encapsulada em termos de engenharia de software.

Toda a interface gráfica será implementada em termos de estilização através da *framework* Bootstrap 4. Esta opção foi tomada com base na estrutura oferecida por esta tecnologia, permitindo o desenvolvimento de interfaces gráficas de forma rápida e simples por apresentar um conjunto de definições de CSS e componentes complexos já incluídas.

6.1.2 Modelos Base Geradores de Código

De forma a exemplificar e dotar a aplicação com a capacidade de gerar outras aplicações são incluído alguns modelos geradores de código.

Para criação de base de dados será incluído um modelo baseado na tecnologia MongoDB, uma base de dados não relacional.

Para a criação de *backend* são adotados dois modelos: um baseado em Hapi.js [85] e outro em Sails[86].

Para a criação de *frontend* será adotado Angular 2.0 com TypeScript.

As escolhas destas tecnologias foram baseadas nas preferências no uso de tecnologias *trending* e com elevada performance.

[83] https://ace.c9.io/#nav=about
[84] https://en.wikipedia.org/wiki/Comparison_of_JavaScript-based_source_code_editors
[85] http://hapijs.com/
[86] http://sailsjs.org/

6.2 Implementação

Na presente secção apresentam-se algumas evidências de implementação do design descrito no capitulo 5.

6.2.1 Arquitetura

Esta subsecção tem como objetivo mostrar evidências de como a solução está instalada e funcional no computador do cliente, do funcionamento do gerador de código e do funcionamento de um modelo de geração de código.

6.2.1.1 Vista de Instalação

Como mencionado anteriormente a solução consiste numa aplicação Angular 2.0 em execução sobre Electron. A Figura 24 ilustra a utilização a inicialização da aplicação desktop e a instrução de carregamento da página inicial da aplicação Angular 2.0 desenvolvida (linha 21).

```
// initialize variables
const electron = require('electron')
const app = electron.app;
const BrowserWindow = electron.BrowserWindow;

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let mainWindow

function createWindow() {
  // Create the browser window.
  mainWindow = new BrowserWindow({
    width: 1500,
    height: 600,
    icon: "app/assets/img/windowicon.png",
    autoHideMenuBar: true,
    titleBarStyle: 'hidden-inset'
  });

  // and load the index.html of the Angular 2.0 app.
  mainWindow.loadURL(`file://${__dirname}/index.html`)

  // Open the DevTools (if in Development mode)
  if(ConfigurationService.InDevMode()){
      mainWindow.webContents.openDevTools();
  }

  // Emitted when the window is closed.
  mainWindow.on('closed', function() {
    // Dereference the window object, usually you would store windows
    // in an array if your app supports multi windows, this is the time
    // when you should delete the corresponding element.
    mainWindow = null;
  })
}
```

Figura 24 - Inicialização da Aplicação com Electron

6.2.1.2 Vista Lógica

Perante a vista lógica demonstra-se a implementação efetuada para tornar possíveis as funcionalidades de geração de código. Para exemplifica-se demonstra-se a implementação do [RF4] Gerir Atores, neste caso, com foco na criação.

Toda a implementação teve inicio no desenvolvimento da classe de software correspondente à entidade “Project”. Esta é a unidade desta solução onde serão geridas todas as informações de negócio que irão permitir a geração de código. É ilustrada na Figura 25.

```typescript
import { Actor } from '../actor/actor.model';
import { Entity } from '../entity/entity.model';
import { UseCase } from '../use-case/use-case.model';
import { SoftwareLayer } from '../software-layer/software-layer.model';

export class Project {
    public Id: any;
    public Name: string;
    public Description: string;
    public CreationDatetime: Date;
    public LocationPath: string;
    public CoverUrl: string;

    public Entities: Array<Entity>;
    public Actors: Array<Actor>;
    public UseCases: Array<UseCase>;

    public SoftwareLayers: Array<SoftwareLayer>;

    constructor() { }
}
```

Figura 25 - Classe de Software para Representação de um Projeto

A Figura 26 ilustra o serviço Angular 2.0 criado com o objetivo de executar as instruções de geração de código. Este serviço representa o gerador de código e tem um método para cada funcionalidade de geração disponível, fazendo uso da interface Node.js “exec” para executar os scripts do modelo gerador de código em uso. O objetivo final é criar condições para que a aplicação criada reconheça um novo ator.

```typescript
export class CodeGeneratorService {

    public NewActor(name: string, description: string): Promise<Array<Array<any>>> {
        //init array of Promises to run
        var PromisesToRun = new Array();

        //for each Software Layer in Project
        for (var i = 0; i < this.ProjectEditorService.CurrentProject.SoftwareLayers.length; i++) {
            //create a new promise
            PromisesToRun.push(new Promise((resolve, reject) => {
                //define script path
                var scriptPath = this.ProjectEditorService.CurrentProject.LocationPath + "/code-generator/" +
                    this.ProjectEditorService.CurrentProject.SoftwareLayers[i].Type + "/" +
                    this.ProjectEditorService.CurrentProject.SoftwareLayers[i].GenerationModel.Name + "/scripts/CreateActor.js";

                //execute Script
                const child = NodeAPIs.exec(scriptPath + " " + name + " " + description,
                    (error, stdout, stderr) => {
                        if (error !== null) {
                            reject([this.ProjectEditorService.CurrentProject.SoftwareLayers[i].Type, error]);
                        }
                    });
                child.on('close', (code) => {
                    resolve([this.ProjectEditorService.CurrentProject.SoftwareLayers[i].Type, code]);
                });
            }));
        }

        //return the promise of all promises
        return Promise.all(PromisesToRun);
    }
```

Figura 26 - Implementação da Criação de um Novo Ator no Gerador de Código

A geração de código foi implementada conforme o design descrito na Figura 15 da secção 5.2.2.1.

A Figura 27 demonstra a implementação do script “CreateActor.js”. O script neste exemplo pertence a um gerador de código de *backend* em TypeScript e utiliza uma *blueprint* apresentada na Figura 28. Exemplificando o código gerado no final do processo. Com o nome “Professor” e a descrição “Professor Actor to represent all the Professors of the University”, o resultado final é ilustrado na Figura 29.

```
const fs = require('fs');

try {

  // load parameters
  var actor = {
    Name: process.argv[1],
    Description: process.argv[2]
  }

  // define blueprint path
  var pathToRead = "../resources/blueprints/actor.js";

  // load blueprint
  fs.readFile(pathToRead, (err, data) => {
      try {
        if (err) {
          throw e;
        } else {
          // replace references
          data = str.replace("{Actor}", actor.Name, "g");
          data = str.replace("{Description}", actor.Description, "g");

          // define result path
          var pathToWrite = "../../../backend/actors/" + actor.Name + ".actor.js";

          // save new file
          fs.writeFile(pathToWrite, data, (err) => {
              if (err) {
                throw e;
              } else {
                //  everything is fine, return 0;
                return 0;
              });
          }
        } catch (e) {
          throw e;
        }
      });

  });
} catch (err) {
  // something went wrong, return error code
  return -1;
}
```

Figura 27 - Script de Criação de Atores

```
/*
 * {Description}
 */
export class {Actor}Actor  {

  public Id: any;
  public Name: string;
  public UseCases: Array<UseCase>;

  constructor(private name: string){

  }
}
```

Figura 28 - *Blueprint* para a Geração de Atores em *Backend* Hapi.js/Node.js

```
/*
 * Professor Actor to represent all the Professors of the University
 */
export class ProfessorActor  {

  public Id: any;
  public Name: string;
  public UseCases: Array<UseCase>;

  constructor(private name: string){

  }
}

```

Figura 29 - Classe de Software Ator Automaticamente Criada

6.3 Testes

Na presente secção são apresentados e descritos os diferentes tipos de testes de software que foram efetuados à solução com o objetivo de preparar a mesma para as experimentações a realizar. Acredita-se que a verdadeira validação dos objetivos propostos será creditada através da execução da experimentação. Assim, definem-se neste momento, apenas de forma breve, quais os testes a executar com vista a atingir um nível de qualidade suficiente para poder experimentar a solução.

6.3.1 Configuração

Para testar se o software funciona no hardware a ser instalado deverão ser efetuadas instalações bem-sucedidas, ou seja, ser possível executar o software nas plataformas mínimas suportadas segundo o requisito não funcional número 20 (descrito na secção 4.3.6). Assim, o suporte da solução desenvolvida irá ser testado em máquinas:

- macOS 10.9 64bits, ou superior;
- Windows 7 32 ou 64 bits, ou superior;
- Ubuntu 12.04 ou superior;
- Fedora 21
- Debian 8

6.3.2 Instalação

Deverá ser testado se o processo de instalação decorre conforme o planeado, em diferente hardware e em diferentes condições como por exemplo pouco espaço de memória, interrupções de rede, interrupções de instalação. Assim, perante:

- Espaço insuficiente em disco:
    - O software deverá informar o utilizador do facto, cancelando a instalação
- Interrupção de rede
    - O software não deverá reagir perante este facto visto que a sua instalação não necessita de uma conexão à rede depois de feita transferência do instalador
- Interrupções de instalação
    - O software deverá reagir de forma a terminar o processo, removendo todos os dados até ao momento instalados e informando o utilizador do mesmo
- Incompatibilidade da maquina
    - O software deverá informar que a maquina em uso é incompatível com a plataforma a instalar

6.3.3 Integridade

De forma a testar a robustez do software implementado serão efetuados testes aos requisitos funcionais com valores não adequados aos esperados assim como tentativas de instalação e execução do software em plataformas incompatíveis.

6.3.4 Segurança

A plataforma deve assegurar que todas as aplicações criadas através da mesma implementam níveis adequados de segurança. Para isto serão efetuados testes de segurança aos modelos geradores de código incluídos de forma a garantir que todos os dados são acedidos de maneira segura e apenas pelo autor das ações.

6.3.5 Funcionais

De forma a testar a vertente funcional da solução todos os casos de uso serão testados inserindo os dados indicados no formato esperado. Uma comparação entre o resultado esperado e o resultado obtido indicará sucesso no teste em caso de igualdade.

6.3.6 Unidade

Cada modelo de geração de código deverá ser testado de forma individual garantindo assim o seu funcionamento. O teste deverá consistir na criação e geração de um ator, uma entidade de negócio e um caso de uso para cada modelo e combinação de modelos.

6.3.7 Integração

Todos os modelos de geração de código deverão em combinações garantindo assim os seus funcionamentos e o funcionamento do modelo geração de código. Os testes deverão consistir na criação e geração de um ator, uma entidade de negócio e um caso de uso para cada modelo e combinação de modelos.

6.3.8 Performance

Os presentes testes têm como objetivo medir e aplicar objetivos a atingir em termos de performance da solução perante situações variáveis, nomeadamente:

1. Testes de Carga

Logo após o inicio da aplicação os comandos devem ser executados num espaço inferior a 50 milissegundos após o inicio da ação requisitada.

2. Testes de Stress

A aplicação deverá ter um tempo de resposta inferior a 100 milissegundos a todas as ações, em situações em que tenha mais de 5 instâncias abertas com um mesmo projeto.

3. Testes de Estabilidade

O sistema deverá ser capaz de funcionar conforme o esperado em termos de qualidade mesmo após a execução do mesmo de forma interrupta durante mais de 80 horas (cerca de 10 dias de trabalho com 8 horas cada).

7 Experiências e avaliação

Em 1990 o professor universitário Paul R. Cohen realizou um questionário sobre 150 artigos científicos e concluiu, entre várias coisas, que: (i) 80% das teses elaboradas não faziam qualquer tentativa de explicar performance das soluções, de explicar porque é que são boas ou más e em que condições são melhores ou piores; (ii) 16% das teses não oferecem nada que possa ser interpretado como uma questão ou uma hipótese. Surge assim uma clara falta de sinergia entre a teoria apresentada e os resultados finais reais.

Assim torna-se essencial a presença deste capítulo de forma a suportar se a abordagem apresentada resolve ou não o problema inicial através da avaliação de resultados de um processo de experimentação.

7.1 Abordagem

De forma a poder efetuar uma avaliação sobre a solução torna-se necessário definir uma hipótese. Em vista do problema abordado neste trabalho a aceleração do processo de desenvolvimento de software torna-se num ponto fundamental a ser avaliado. Exploram-se então duas hipóteses: (i) Será está solução realmente eficaz na resolução do problema? (ii) É melhor ou pior do que as soluções já existentes atualmente? Assim, após ter sido executada a devida analise através da elaboração do estado da arte, são formuladas duas afirmações: (i) “A solução abordada acelera o processo de desenvolvimento de software”; (ii) “A solução abordada é melhor que as alternativas existentes”.

Assim serão efetuados dois processos de experimentação, um para cada hipótese, respetivamente nas secções 7.2 e 7.3.

7.2 Tempo de Desenvolvimento

A primeira experimentação irá ser executada através de grupo de controlo de forma a medir a grandeza tempo, testando a primeira hipótese. O objetivo é perceber se a abordagem proposta acelera ou não o desenvolvimento de software.

Assim um grupo de programadores será submetido ao processo de criação de uma aplicação, duas vezes, uma vez sem o apoio de qualquer gerador de código e uma outra vez utilizando a solução desenvolvida. Acredita-se que a melhor forma de entender se a solução acelera de facto o desenvolvimento ou não é comparar o processo com o equivalente sem qualquer tipo de geração automática.

O grupo de programadores será constituído por um mínimo de 5 pessoas, e um máximo de 10 pessoas por sessão, com idades compreendidas entre os 20 e os 40 anos, de ambos os sexos. Cada programador deverá ter no mínimo 1 ano de experiência em desenvolvimento de software. Irá tentar-se reunir participantes com diferentes níveis de experiência e a maior diversidade de conhecimento de tecnologias possível.

A experiência irá consistir na criação de uma aplicação composta por interface gráfica com o objetivo de listar, criar, editar e eliminar instâncias de uma classe de software “Contacto”, definida na Figura 30.

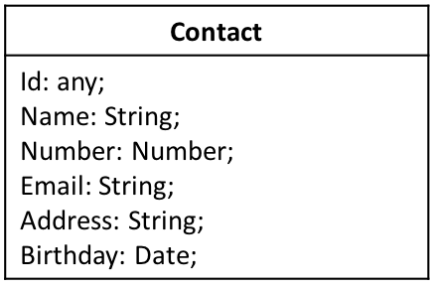


Figura 30 - Modelo de Dados para a Experiência ao Tempo de Desenvolvimento

Esta interface gráfica deverá depender de uma interface aplicacional para gestão, validação e persistência dos dados numa base de dados, também a definir. A aplicação a criar tem apenas

uma entidade de modelo de negócio de forma a que, após os resultados, seja possível efetuar projeções sobre o tempo de implementação de aplicações com mais entidades de negócio.

Para apuração dos resultados será calculada uma média dos tempos de desenvolvimento em ambas as abordagens na expectativa de obter uma visão da diferença de tempos de desenvolvimento entre ambas as plataformas.

A execução da experiência será numa sala de escritório em ambiente controlado por um orientador/moderador, em que os programadores estarão todos em simultâneo a executar a implementação das aplicações. A experiência inicia-se com o desenvolvimento sem gerador de código. Assim que cada programador terminar é-lhe permitido que inicie o segundo desenvolvimento, desta vez utilizando a solução criada.

7.3 Satisfação do Utilizador

A segunda experimentação irá ser executada através de um inquérito de satisfação de forma a medir a grandeza satisfação do utilizador, testando a segunda hipótese. O objetivo é perceber se a abordagem proposta é ou não melhor que um dos "concorrentes" mais aceites atualmente na perspetiva do programador.

Assim será distribuído um inquérito com cerca de 12 perguntas sobre a experiência de utilização de plataformas RAD e a solução desenvolvida com 10 respostas fechadas e 2 questões de exploração de opinião. Este inquérito terá como alvos programadores que tenham utilizado a solução criada e uma plataforma RAD e ainda que tenham diferentes níveis de experiência profissional. No inicio do inquérito o participante deverá preencher qual a plataforma RAD que experimentou e sobre a qual pretende efetuar a comparação. No caso de existir mais do que uma plataforma RAD o participante deverá selecionar aquela que mais o satisfez na sua experiência de utilização, escolha essa que será o alvo para comparação.

O inquérito exposto na Tabela 4 é baseado no "Guia prático para elaboração de inquéritos por questionário"[87] escrito pelos professores Rui Mendes, João Fernandes e Manuel Correia do Instituto Superior Técnico de Lisboa. Tem o objetivo de identificar caraterísticas entre plataformas RAD e a solução criada, com a intenção de confirmar ou verificar se a solução criada é ou não melhor que as "concorrentes". Ao mesmo tempo pretende descrever a população de

[87] https://fenix.tecnico.ulisboa.pt/downloadFile/3779580654133/Guia%20Pratico.pdf

programadores no que se relaciona com as suas expectativas em relação à geração automática de código.

Tabela 4 - Inquérito de Satisfação de Utilização

Número	**Tipo de questão**	**Designação da questão**	**Tipo de resposta**	**Opções de resposta fechada**
1	Preferência	Prefere utilizar uma RAD ou desenvolver código diretamente?	Fechada	RAD; Desenvolver diretamente
2	Preferência	Prefere desenvolver uma solução gerida centralmente em *cloud computing* ou presente localmente na sua máquina?	Fechada	*Cloud Computing*; Local na minha máquina
3	Factos	Quantas horas demora em média a construir uma aplicação com um caso de uso numa Aplicação RAD?	Fechada	1 a 4 horas; 5 a 8 horas; Mais de 8 horas;
4	Factos	Já necessitou de customizar/estender uma aplicação criada por uma plataforma RAD e sentiu-se limitado pela ausência de possibilidade de edição de código?	Fechada	Sim, sempre; Sim, às vezes; Não, nunca; Não, mas podia ser melhor;
5	Atitudes	Em que medida concorda ou discorda com a não possibilidade de edição de código das plataformas RAD?	Fechada	Discordo totalmente; Discordo em parte; Não concordo nem discordo; Concordo em parte; Concordo totalmente;
6	Atitudes	Em que medida concorda ou discorda com a geração do	Fechada	Discordo totalmente;

		código com base no modelo de negócio da aplicação a criar?		Discordo em parte; Não concordo nem discordo; Concordo em parte; Concordo totalmente;
7	Satisfação	Em que medida está satisfeito com a utilização da solução desenvolvida?	Fechada	Nada satisfeito; Pouco satisfeito; Satisfeito; Muito Satisfeito
8	Satisfação	Em que medida está satisfeito com as aplicações resultantes da utilização da solução desenvolvida?	Fechada	Nada satisfeito; Pouco satisfeito; Satisfeito; Muito Satisfeito
9	Opiniões	Qual a sua opinião em relação a aplicações RAD?	Aberta	(Não aplicável)
10	Opiniões	Qual a sua opinião em relação à solução criada RAD?	Aberta	(Não aplicável)
11	Valores	Qual é a importância que atribui à possibilidade de escolher quais as tecnologias a utilizar na geração de código?	Fechada	Nada importante; Pouco importante; Importante; Muito importante;
12	Valores	Qual é a importância que atribui à possibilidade de poder editar/personalizar o gerador de código?	Fechada	Nada importante; Pouco importante; Importante; Muito importante;

8 Conclusões e trabalho futuro

No presente capítulo são apresentadas as conclusões da execução desta tese, sintetizando todos os capítulos que compõem a mesma e discutindo sobre o trabalho efetuado. Será também feita uma análise ao trabalho futuro que poderá ser desenvolvido.

8.1 Síntese

No decorrer desta secção vai ser efetuada uma breve síntese do que foi desenvolvido durante a elaboração desta tese, analisando brevemente cada um dos capítulos existentes.

O Capítulo 1 introduz e contextualiza esta tese e apresenta o problema em causa. Desenvolver software de forma rápida pode ser um problema complexo. Existe necessidade de efetuar desenvolvimentos que sejam capazes de acompanhar a evolução do mercado e que possam ser extensíveis, capazes de ser evoluídos a um ponto de criar experiências verdadeiramente interessantes para o utilizador final. Há assim que tomar uma abordagem para a resolução deste problema.

Esta tese apresenta um modelo de geração automática de código como possível resolução, implicando estratégias como o foco no modelo de negócio do cliente e a possibilidade de seleção das tecnologias a utilizar com modelos gerados de código e de editar os mesmos.

Neste capítulo são ainda definidos os objetivos a atingir com esta proposta, visando não só a resolução do problema mas a disrupção no modelo de solução já existe e apresentado pelas soluções indicadas no capítulo 2. É ainda apresentada a estrutura deste documento.

Por sua vez o capítulo 2 apresenta um estudo das soluções já existentes para o problema apontado. São referidos vários tipos de aplicações: (i) aplicações RAD, que dão uma resposta direta ao problema, permitindo o desenvolvimento de alto nível (com foco na funcionalidade e não no código) através da geração automática de código; (ii) aplicações de gestão de conteúdos que permitem o desenvolvimento de aplicações por replicação de modelos já definidos; e ainda (iii) os ambientes de desenvolvimento integrado, que oferecem funcionalidades simples geração de código, permitindo acelerar o desenvolvimento de código.

É efetuada uma comparação entre todas as soluções e tecnologias de forma a avaliar quais as possibilidades de execução, aquilo que já foi feito, que poderá ser feito e que valerá a pena fazer.

Por último é também efetuada uma análise às tecnologias usadas que despertaram a execução desta tese, tecnologias que, nas suas medidas, mesmo não respondendo ao problema, criam condições para que a solução para o problema possa ser concebida.

De seguida o capítulo 3 procura perceber qual o valor que a solução apresentada poderá trazer para os interessados na mesma. É realizada uma exploração do valor para os programadores, para as empresas de desenvolvimento que empregam esses mesmos programadores, e ainda para os clientes dessas mesmas empresas.

É tecido um Modelo Canvas para uma análise de como irá funcionar o negócio baseado na abordagem adotada.

O capítulo 4 apresenta os requisitos de software para a conceção da solução. É realizado um enquadramento da proposta, apresentando a missão a cumprir, os atores e sistemas internos que irão definir os utilizadores e qual a prioridade de requisitos entre si.

É aqui apresentado o diagrama de casos de uso, definindo 23 requisitos funcionais considerados mínimos e essenciais para se considerar a solução pronta para exposição ao mercado. Estes requisitos comtemplam as necessidades de geração de código, as necessidades de edição e

execução desse mesmo código, a gestão de versões do trabalho em desenvolvimento e um conjunto de ferramentas adicionais variadas para facilitar o desenvolvimento.

Para além dos requisitos funcionais são definidos também requisitos não funcionais com vista a aumentar e zelar pela qualidade da solução desenvolvida, atual e futura. Estes requisitos são escritos com vista a atingir as áreas de funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

O modelo de domínio e a arquitetura da solução são apresentadas no capítulo 5. São apresentadas as vistas de instalação e funcional, detalhando o funcionamento do gerador de código e dos modelos geradores de código. Para isso é apresentado o diagrama de sequência de funcionamento do gerador de código concebido, exemplificando como funciona o mesmo. É também apresentada a estrutura de um modelo gerador de código, mostrando como é feita a sua integração no gerador de código e qual a estrutura/composição do mesmo.

Neste capitulo são ainda apresentados os mockups gráficos do ecrã inicial e do ecrã de edição. O ecrã de edição representa a primeira interface visível para com o utilizador, interface esta onde poderão ser realizadas as tarefas de gestão de projetos criados e gestão de configurações. O ecrã de edição representa o ecrã mais importante para a solução, ecrã este onde será efetuada a edição do projeto em desenvolvimento com esta solução.

O capítulo 6 aborda os detalhes da implementação feita, nomeadamente, descreve quais as tecnologias em uso para a conceção do ambiente de desenvolvimento e para os modelos de geração de código incluídos no ambiente por defeito. São apresentas evidências da implementação do Design descrito no capítulo 5.

São apresentadas as experiências e a avaliação a executar à plataforma criada no capítulo 7. Este é um capítulo de extrema importância pois acredita-se que será a experimentação a executar que irá comprovar se a solução proposta resolve ou não o problema apresentado e em que medida. É feita uma abordagem à metodologia que será seguida e apresentadas as duas experiências a executar, nomeadamente o tempo de execução e a satisfação do utilizador. A primeira será avaliada propondo aos participantes da experiência criar uma aplicação utilizando a solução apresentada e outro meio mais tradicional, comparando assim as diferenças de tempo em ambas as abordagens. A segunda será avaliada através de um inquérito onde os

participantes poderão através de um conjunto de perguntas de reposta fechada e aberta exprimir as suas experiências entre a utilização da solução criada e a utilização de uma plataforma RAD.

O presente capítulo apresenta as conclusões da elaboração desta tese, refletindo sobre todo o processo descrito neste documento e discutindo os resultados obtidos e analisando qual o possível trabalho futuro a concretizar.

8.2 Discussão

Nesta secção é efetuada uma análise critica sobre os aspetos mais relevantes do trabalho desenvolvido durante a presente dissertação.

8.2.1 Evolução do Objetivo Inicial

O objetivo inicial que despertou o desenvolvimento desta tese consistiu na criação e desenvolvimento de um ambiente de integração de aplicações. Fruto da experiência profissional surgiu a vontade de conceber uma solução que permitisse criar software de forma simples e rápida com facilidade em integrar-se com outros softwares. O plano consistia na criação de uma estrutura que permitisse modernizar aplicações já existentes, rescrevendo as mesmas sobre o formato de uma interface de programação de aplicações REST com um *frontend* alimentado pela mesma. Esta conceção iria permitir interligar aplicações como serviços web, modernizando as suas estruturas e promovendo alta coesão entre diferentes sistemas. Com a exploração da ideia entendeu-se que criar um conceito/padrão de estruturação de software não seria suficiente. Era necessária uma ferramenta que de facto agiliza-se o processo de desenvolvimento de software, orientando o programador a alcançar a estrutura de engenharia pretendida para promover a integração, e este foi o resultado atingido.

8.2.2 Desenvolvimento Ágil como Metodologia

O processo lógico foi então tentar conceber uma solução/metodologia que permitisse ao programador conceber de forma rápida e estruturada o seu código. Rapidamente se entendeu na elaboração do estado da arte que já existiam soluções para o problema: as plataformas RAD. Mas estas plataformas apresentam limitações consequentes de toda a aceleração de

desenvolvimento que proporcionam. Ao permitirem o desenvolvimento de software numa visão de alto nível, sem necessitar de conhecimentos de programação, concebem aplicações com interfaces simples, limitadas a uma tecnologia não alterável posteriormente e sem acesso ao código. As plataformas RAD permitem assim o desenvolvimento de software por utilizadores não programadores, aumentando o numero de recursos possíveis de alocar para a tarefa a executar.

Mas permaneceu a questão se poderia tudo isto ser algo positivo ou não. Conceber software sem conhecimentos para tal pode comprometer a estrutura desenvolvida a longo prazo visto que não programadores podem não entender quais as consequências das suas ações perante a programação de um novo sistema. Ainda, o não acesso ao código representa uma limitação na medida em que inibe o programador de desenvolver livremente, ficando tipicamente dependente da plataforma. A acrescentar a este facto surge a questão das tecnologias em uso. Estas plataformas concebem código numa só tecnologia, nomeadamente em alguns casos, tecnologia web, e exportam as suas criações para os formatos desktop e móvel. Com isto uma possível mudança de tecnologia no futuro poderá ficar comprometida por um reinicio de toda a implementação. E assim mais uma vez entende-se que o caminho ainda não era o correto.

8.2.3 Geração Automática de Código

A base da aceleração do desenvolvimento de software procurada estava na geração automática de código e por isso havia que conceber uma solução que pudesse providenciar isso aos utilizadores, com as mesmas vantagens das plataformas RAD mas sem as suas limitações. Esta abordagem já existe em vários ambientes de desenvolvimento integrado, mas toma apenas proporções mínimas criando apenas pequenas partes do software. Assim a presente tese resultou na criação de uma *framework* de agilização do processo de criação de aplicações multiplataforma através da geração automática de código, realçando e orientado o processo através dos padrões de engenharia de software e promovendo a integração de aplicações como serviços web.

A abordagem apresentada pelas plataformas RAD para a conceção de um sistema consiste na criação de estruturas baseadas em requisitos de negócios como páginas a criar, entidades do sistema e papeis de utilizadores. Entendeu-se que, no fundo, toda a estrutura consistia em princípios de engenharia de software como entidades de modelo de negócio, atores e casos de

uso. Então esta foi a abordagem escolhida, revisitando as raízes do desenvolvimento de software, procurando descomplicar o mesmo e aproximar o programador dos padrões de software. Destaca-se que, esta solução não propõe ao utilizador a implementação de um determinado padrão conforme o mesmo existe, mas sim procurar orientar o utilizador a criar novas estruturas que no seu interior implementam esses mesmos padrões.

8.2.4 A Necessidade de Múltiplas Tecnologias

Limitar um gerador automático de código a uma só linguagem de código é algo que se compreendeu como não necessário. Entendeu-se no desenvolvimento desta tese que esta poderia ser uma desvantagem facilmente ultrapassa se aplicados os padrões de software adequados. Se a solução pudesse ter vários modelos de geração de código em vez de apenas um, ou, num mesmo ambiente de desenvolvimento, fosse, por exemplo, possível criar interfaces web, interfaces nativas Android, bases de dados não relacionais e bases de dados relacionais, seria possível otimizar e acelerar o desenvolvimento. E a solução apresentada inclui esta proposta, sendo uma das funcionalidades que mais se orgulha uma vez que representa uma disrupção no modelo atualmente conhecido em mercado para este tipo de plataformas.

8.2.5 A Necessidade de Edição de Código Gerado

Durante a análise de mercado entendeu-se que o futuro do desenvolvimento está na criação de software como experiências de utilização ricas para o utilizador. Mais do que conceber software funcional, bem testado e de alta performance, torna-se necessário atualmente criar software que promova uma experiência emocional no utilizador, fazendo com que o mesmo sinta satisfação na utilização da aplicação. E este facto resulta na necessidade de alta personalização. Há assim que remover a limitação de não possibilidade de edição de código gerado e confiar o mesmo ao programador. Desta forma o programador terá o seu trabalho acelerado, não ficando dependente da plataforma em momento algum.

8.2.6 Motivações

As conjugações de algumas tecnologias em mercado representaram um forte impulso na realização e conceção desta tese. Olhando para um mercado onde existem cada vez mais opções para implementação de software, foi a presença de tecnologias como Angular 2.0,

Apache Cordova, Electron e o Ace Editor que tornaram possível o ideal de desenvolver um ambiente capaz de gerar aplicações que, com apenas um só código, fossem executáveis em formato web, como aplicações desktop nativas e como aplicações móveis nativas. Este facto foi também motivo para a idealização da criação de vários modelos de geração de código, permitindo assim uma liberdade de escolha ao programador, facto que se considera ideal e disruptivo perante outras soluções semelhantes.

8.2.7 Trabalho Realizado

A criação de um ambiente gerador de código capaz de incluir diferentes modelos de geração é um resultado que motiva orgulho. Foi possível conceber um produto que procura oferecer uma alternativa com inovações em mercado, procurando preencher e definir o seu âmbito em pequenos nichos de negócio que outras plataformas não vieram a completar.

A evolução desde do objetivo inicial desta tese até este ponto presente mostra a dedicação e paixão que este produto gerou em volta da motivação em criar uma solução que permitisse integração. E assim, não só é possível integrar aplicações como integrar modelação de negócio com o desenvolvimento de software. Este ambiente representa um esforço em consolidar os princípios de engenharia de software, tornando-os mais simples de entender e mostrando que os mesmos são normalmente a base fundamental de qualquer sistema.

Infelizmente, por motivos de insuficiência de tempo, não foi possível implementar todas as funcionalidades consideradas essenciais ou desejáveis. Contudo, já é possível executar a criação de projetos e a geração automática de código.

Após a finalização da implementação será possivel levar o software a experimentação e provar o conceito de geração automática de código a partir do modelo de negócio e toda a funcionalidade que o mesmo poderá implicar. Tentar-se-á provar que através da utilização de modelos geradores de código personalizáveis é possível conceber diversas abordagens que darão total liberdade ao programador, oferecendo-lhe um desenvolvimento acelerado de software.

8.2.8 Expectativas

Espera-se que este trabalho possa de facto permitir acelerar a vida do programador no seu dia-a-dia de desenvolvimento, oferecendo a oportunidade de escrever o seu código de forma rápida e simples, sem necessidade de testes exaustivos.

O desenvolvimento de um mercado de modelos de geração de código é também uma expectativa desta tese. Este mercado irá ser o modelo potenciador do crescimento da plataforma e representa um núcleo fundamental na medida em que permitirá um aumento do leque de possibilidades para criação de novas aplicações.

Espera-se ainda poder encontrar interesse da comunidade e interesse em investidores externos de forma a poder conceber ainda mais trabalho, aumentando as funcionalidades desta solução em trabalho futuro. Um objetivo a atingir seria poder vir a disponibilizar oportunidades de desenvolvimento desta solução aos alunos do MEI ISEP, procurando desenvolver as suas carreiras em engenharia informática e usufruir da sua qualidade para aumentar a qualidade da solução.

8.3 Trabalho Futuro

Serão agora referidas algumas ideias sobre o que poderá ser efetuado como trabalho futuro para a solução criada com o objetivo de melhorar a mesma.

8.3.1 Objetivo

Após o trabalho realizado para esta tese entendeu-se que a solução desenvolvida tem potencial para crescer, incluindo novas funcionalidades. O objetivo é tornar o produto criado cada vez mais poderoso e mais completo, disponibilizando assim de forma constante as melhores inovações possíveis, procurando manter e aumentar a base de utilizadores.

O primeiro passo a executar será terminar a implementação da solução podendo assim disponibilizar a solução como produto completo, capaz de ser levado à experimentação para atestar e comprovar esta tese junto do público-alvo.

8.3.2 Futuro do Mercado

No que está relacionado com prospeção futura acredita-se existir muito a fazer. A visão desta tese construiu uma noção que o futuro do mercado passa pela mobilidade, pela utilização de uma solução a partir de qualquer plataforma, pela preferência por aplicações/serviços que oferecem de facto uma experiência de utilização de alguma forma inovadora, disruptiva, cativante e útil. Assim, traçam-se algumas ideias para possíveis novas funcionalidades.

8.3.3 Importar Geradores de Código

Visto que esta solução foi desenhada para permitir utilizar múltiplos modelos de geração e código e que os mesmos poderão ser modificados em cada novo projeto, seria interessante considerar a possibilidade de importar gerador de código, tanto um modelo transferido da internet como um modelo utilizado num projeto já existente.

O objetivo seria criar um mercado de geradores de código no website da plataforma, permitindo aos utilizadores transferirem os mesmos e integra-los nos seus ambientes. Por outro lado, poderiam também importar outros modelos oriundos de outras fontes que não o mercado da plataforma. Seria efetuada a questão/aviso que utilização de modelos criados por outras entidades poderá ser inseguro para o sistema, visto que seriam executados scripts não validados pelos criadores do ambiente.

A importação de modelos de geração de código a partir de modelos editados em projetos poderá representar mais um fator de aceleração a criar. Acredita-se que no presente caso um modelo não serve para todas as necessidades de todos os utilizadores. Assim é dada a possibilidade a cada utilizador de efetuar as pretendidas alterações aos modelos geradores de código que escolhe para o seu projeto. Ao permitir a importação dos mesmos noutros projetos o utilizador disfruta de uma base de arranque que já reconhece e aprova pela utilização que fez anteriormente. Como trabalho ainda mais futuro é possível imaginar a exportação direta de módulos editados para o mercado da plataforma, potenciando assim o crescimento em numero de modelos disponíveis para transferência.

8.3.4 Gerir Geradores de código

Admitindo que seria possível importar modelos de geração de código os utilizadores da plataforma iriam atingir um momento em que teriam a necessidade de gerir esses modelos, adicionando, removendo e editando modelos importados/incluídos no ambiente.

A ideia consiste então em adicionar um caso de uso à plataforma onde seria possível executar esta gestão permitindo ao programador mais uma funcionalidade de personalização da plataforma.

8.3.5 Gerar Diagramas

Visto que a aplicação define em cada projeto a lógica de negócio do mesmo torna-se possível imaginar uma funcionalidade em que o ambiente poderia usar essa informação para gerar os tradicionais diagramas como o diagrama de casos de uso, diagramas de classes e o modelo de domínio.

Aplicando algumas *frameworks* de desenho seria possível concessionar uma funcionalidade onde o utilizador poderia visualizar a informação do negócio de forma gráfica, talvez mesmo, especificar o negócio através da construção de diagramas. Será algo a implementar no futuro.

8.3.6 Tradução de Casos de Uso para Código

Pretende-se algures no futuro dotar esta solução de uma funcionalidade onde o programador possa apenas definir o caso de uso e ver o mesmo automaticamente mapeado para a criação de um ator, uma entidade e um caso de uso.

A ideia passa por ser capaz de receber a introdução do utilizador “Professor Cria Pauta”, e criar o ator “Professor”, a entidade “Pauta” e o caso de uso “Criar Pauta”. Esta funcionalidade poderá aumentar a aceleração do processo de desenvolvimento na medida em que os requisitos funcionais poderão quase que literalmente ser transformados em código automática gerado.

Após esta funcionalidade ser implementada, se forem adicionadas funcionalidades de reconhecimento de voz, torna possível idealizar um caso de uso em que o programador poderá falar com o sistema, indicando-lhe os detalhes do negócio e em tempo-real ver a aplicação em edição a construir-se de forma a implementar esses mesmos detalhes.

8.3.7 Definição de Tarefas

E se, a solução criada pudesse suportar a criação de atividades do Sprint tal como criação de novas funcionalidades ou correção de bugs? O gestor de projeto poderia planear todas as atividades, atribuir atividades a programadores e cada programador poderia rapidamente visualizar o seu trabalho atual e guardar o seu código em associação a uma atividade.

Esta funcionalidade a desenvolver obrigará à alteração da lógica de negócio apresenta, passando a ter mais um ator, o gestor de projeto e ainda entidades para gestão de atividades. Acredita-se que esta funcionalidade poderia ainda mais acelerar o desenvolvimento na medida em que toda a informação de gestão e estado do projeto estaria centralizada num só lugar e facilmente poderia ser acedida.

8.3.8 Chat e Colaboração em Tempo-real

Uma funcionalidade interessante a incluir será a possibilidade de troca de mensagens de texto, ou conversação por voz entre programadores de forma a esclarecerem duvidas e terem apoio dos seus colegas.

Em cada linha do editor seria possível identificar quem foi o programador responsável pela mesma, e, com um clique num local a especificar, iniciar um contacto com esse mesmo programador. Esta funcionalidade promove a mobilidade de programadores visto que poderão não estar no escritório num mesmo momento e ainda assim colaborar, assim como promove a aceleração do desenvolvimento e diminuição de erros por obtenção de apoio.

8.3.9 Assistente Virtual

Num momento onde cada vez mais se fala em assistentes virtuais e inteligência artificial, acredita-se que a inclusão de um BOT poderá ser uma adição positiva para a solução. Um assistente com quem o utilizador poderá conversar, agendar tarefas, resolver questões solicitando ajuda e lembretes.

Por exemplo, o programador poderá ensinar o BOT que resolveu determinada situação no seu código executando uma atividade em especifico. Mais tarde, perante a mesma situação, o BOT poderá recordar o programador da solução, mesmo que este ultimo já não se recorde da mesma. Será uma funcionalidade a implementar.

8.3.10 Desenvolvimento em *Cloud*

Pensando em mobilidade acredita-se ser importante para o programador poder ter acesso ao seu código em qualquer lugar. Mas isto pode revelar-se uma tarefa complexa. Perante os meios mais tradicionais de programação, é necessária uma maquina para ter o código, executar o mesmo e suportar a aplicação em execução.

Existem atualmente vários sistemas que permitem a execução dó código em maquinas virtuais em *cloud.* O objetivo deste trabalho futuro será permitir a um utilizador da solução codificar a solução em que está a trabalhar, mas num repositório alojado na *cloud* podendo assim ler, editar e ver a execução do seu código em qualquer maquina, sem necessidade de elevadas especificações técnicas. Ou seja, computadores, tablets e telemóveis são possíveis dispositivos onde se poderá programar. Acredita-se que, apesar de não aparentar ser o ambiente ideal para tal tarefa, se bem implementada poderá ser uma adição interessante para a resolução de pequenos problemas de momento mesmo sem ter a maquina de desenvolvimento presente.

9 Referências bibliográficas

Alpaydm, E. *Combined 5 × 2 Cv F Test for Comparing Supervised Classification Learning Algorithms*. Neural Computation, 1999.

Apache Cordova. "Get Started Fast," January 3, 2016. https://cordova.apache.org/#getstarted.

Azevedo, Paulo J., and Alípio Mário Jorge. *Ensembles of Jittered Association Rule Classifiers*. Data Mining and Knowledge Discovery 21.1, 2010.

Barrows, M. E., D. E. Gregory, L. Gao, A. L. Rosenberg, and P. R. Cohen. *An Empirical Study of Dynamic Scheduling on Rings of Processors*. Parallel Computing, 1999.

Beck, Kent. *Extreme Programming Explained*. Addison Wesley, n.d. http://ksg.meraka.org.za/~agerber/publications.html.

Begel, Andrew, and Nachiappan Nagappan. *Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study, Microsoft Research*, 2008. http://research.microsoft.com/pubs/56015/AgileDevatMS-ESEM07.pdf.

Bellucci, E., and J. Zeleznikow. *A Comparative Study of Negotiation Decision Support Systems*. Vol. System Sciences. Proceedings of the Thirty-First Hawaii International Conference, 1998.

Boehm, Barry. *A Spiral Model of Software Development*. IEEE Computer, 2014. http://www.dimap.ufrn.br/~jair/ES/artigos/SpiralModelBoehm.pdf.

Bootstrap team, @mdo, and @fat. “Bootstrap 4 Alpha,” August 19, 2015. http://blog.getbootstrap.com/2015/08/19/bootstrap-4-alpha/.

Brooks, Fred, and H.J. Kugler. *No Silver Bullet Essence and Accidents of Software Engineering*. Information Processing ’86. North-Holland: Elsevier Science Publishers B.V, 2014. http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s10/cis20.2/papers/brooks-no-silver-bullet.pdf.

Carnevale, P.J., and Pruitt, D.G. *Negotiation and Mediation*. Annual Review of Psychology, 1992.

Chen, H. *A Research Based on Fuzzy AHP for Multicriteria Supplier Selection in Supply Chain*. Department of Industrial Management, University of Science and Technology, 2004.

Chen, M.-F., and G. H. Tzeng. “Fuzzy MCDM Approach For Evaluation Of Expatriate Assignments.” *International Journal of Information Technology & Decision Making*, 2005.

Coffin, M., and M. J. Saltzman. *Statistical Analysis of Computational Tests of Algorithms and Heuristics*. INFORMS Journal on Computing, 2000.

Cohen, P.R. *A Survey of the Eighth National Conference on Articial Intelligence: Pulling Together or Pulling Apart?* AI Magazine, 1991.

Cohen, P. R. *Empirical Methods for Artificial Intelligence*. IEEE Intelligent Systems, 1996.

De Moor, A., and H. Weigand. *Business Negotiation Support: Theory and Practice*. International Negotiation, 2004.

Demsar, J. *Statistical Comparisons of Classifiers over Multiple Data Sets*. Journal of Machine Learning Research, 2006.

Deng, H. *Multicriteria Analysis with Fuzzy Pairwise Comparison*. International Journal of Approximate Reasoning, 1999.

Drucker, Peter. *Post-Capitalist Society*. Harper Collins e-books, 2009.

Electron. “Get Started with Electron.” Accessed January 3, 2016. http://electron.atom.io/#get-started.

Facebook. “Getting Started,” January 7, 2016. https://facebook.github.io/react/docs/getting-started.html.

Filzmoser, M., and R. Vetschera. *A Classification of Bargaining Steps and Their Impact on Negotiation Outcomes*. Group Decision and Negotiation 17, 2008.

Gama, João, A. Carvalho, K. Faceli, A. Lorena, and M. Oliveira. *Extração de Conhecimento de Dados - Data Mining*. Edições Silabo, n.d.

Gerber, Aurona, Alta Van der Merwe, and Ronell Alberts. *Implications of Rapid Development Methodologies*. CSITEd 2007. Mauritius, 2007. http://ksg.meraka.org.za/~agerber/publications.html.

Glock, Jonathan. "Node.js Framework Comparison: Express vs. Koa vs. Hapi," January 20, 2016. https://www.airpair.com/node.js/posts/nodejs-framework-comparison-express-koa-hapi#5-1-2-the-bad.

Haber, Itamar. "Why Redis Beats Memcached for Caching," January 25, 2016. http://www.infoworld.com/article/2825890/application-development/why-redis-beats-memcached-for-caching.html.

hapi. "Hapi API," December 10, 2015. http://hapijs.com/api.

Hotle, Matt. *The Disintegration of AD: Putting It Back Together Again*. Enterprise Integration Summit. Sao Palo, Brazil: Gartner Group, 2014. http://www.gartner.com.br/tecnologias_empresariais/pdfs/brl37l_a3.pdf.

Japkowicz, Nathalie. "Performance Evaluation for Learning Algorithms." Accessed February 2, 2016. http://www.icmla-conference.org/icmla11/PE_Tutorial.pdf.

Japkowicz, Nathalie, and Mohak Shah. *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

Kerr, James M.;, and Richard Hunter. *Inside RAD: How to Build a Fully Functional System in 90 Days or Less*. McGraw-Hill, n.d.

Martin, James. *Rapid Application Development*. Macmillan, 1991.

Maximilian, E. M., and L. Williams. *Assessing Test-Driven Development at IBM*. Proceedings of International Conference of Software Engineering. Portland, OR, USA, 2003.

Memcached. "About Memcached," January 21, 2016. http://memcached.org/about.

"Message Queue Evaluation Notes." *Second Life*, January 13, 2016. http://wiki.secondlife.com/wiki/Message_Queue_Evaluation_Notes.

MongoDB. "The MongoDB 3.2 Manual." Accessed December 3, 2015. https://docs.mongodb.org/manual/.

mongoosejs. "Guide," December 17, 2015. http://mongoosejs.com/docs/guide.html.

Nicola, S., and E. Pinto Ferreira. "Value Model For Supporting Negotiation In Collaborative Networks," 2010.

Nicola, Susana, Eduarda Pinto Ferreira, and J. J. Pinto Ferreira. "A Novel Framework For Modeling Value For The Customer, An Essay On Negotiation." *International Journal of Information Technology & Decision Making*, 2012.

Node.js. "About Node.js®." Accessed December 5, 2015. https://nodejs.org/en/about/.

OAuth. "About OAuth 2.0." January 4, 2016. http://oauth.net/2/.

Osterwalder, A., and Y. Pigneur. *Business Model Generation: A Handbook for Visioneries, Game Changers, and Challengers, John Wiley & Sons*, 2010.

OutSystems. "Rapid Application Delivery Solution for the Enterprise," January 12, 2016. http://www.outsystems.com/platform/.

Passport.js. "Overview," January 6, 2016. http://passportjs.org/docs.

Redis. "Documentation," January 20, 2016. http://redis.io/documentation.

Robinson, W. N., and V. Volkov. *Supporting the Negotiation Life Cycle*. Communications of the ACM, 1998.

Sails. "Getting Started," December 14, 2015. http://sailsjs.org/get-started.

Socket.io. "Get Started," December 13, 2015. http://socket.io/get-started/.

Stephens, M., and D. Rosenberg. *Extreme Programming Re Factored: The Case Against XP*. Apress, 2003.

Swagger. "What Is Swagger?," December 20, 2015. http://swagger.io/getting-started/.

The Angular Core Team. “Moving the Web Forward,” January 10, 2016. https://angular.io/about/.

Tomislav, Capan. “Why The Hell Would I Use Node.js? A Case-by-Case Tutorial.” Accessed April 1, 2016. http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js.

TypeScript. “TypeScript Handbook,” November 2, 2015. http://www.typescriptlang.org/Handbook.

wavemaker. “Benefits,” January 12, 2016. http://www.wavemaker.com/benefits/.

Weigand, H., and M. Schoop. *B2B Negotiation Support: The Need for a Communication Perspective*. Group Decision and Negotiation, 2003.

Woodall, T. *Conceptualising Value for the Customer: An Attributional, Structural and Dispositional Analysis*. Academy of Marketing Science Review: 12, 2003.

Zaiontz, Charles. “Real Statistics Using Excel,” February 1, 2016. http://www.real-statistics.com/.

Box Uk Team. “What is Sitecore: a developer's view of the CMS”, June 20, 2013. https://www.boxuk.com/insight/blog-posts/what-is-sitecore-developer-view-cms

Mendes, Rui; Fernandes, João; Correia, Manuel; “Guia Prático para a Elaboração de Inquéritos por Questionário”

https://fenix.tecnico.ulisboa.pt/downloadFile/3779580654133/Guia%20Pratico.pdf

10 Anexos

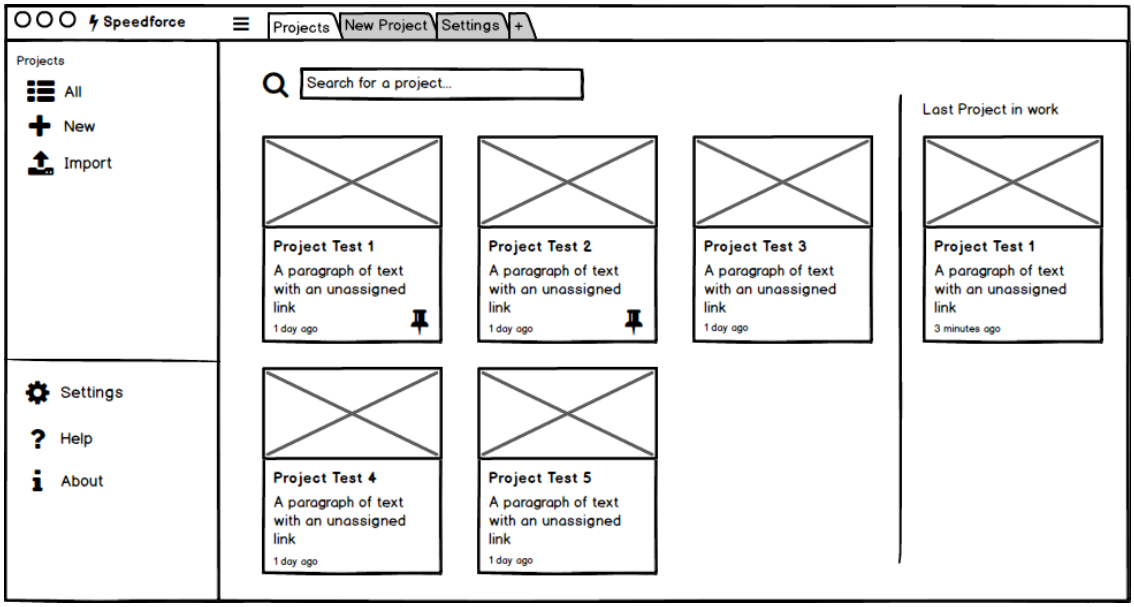
Na presente secção estão presentes todos os conteúdos desta tese que devido ao seu contexto ou dimensão são considerados como anexos a este documento.

10.1 Lista de Linguagens de Código Suportadas pelo Editor da Solução

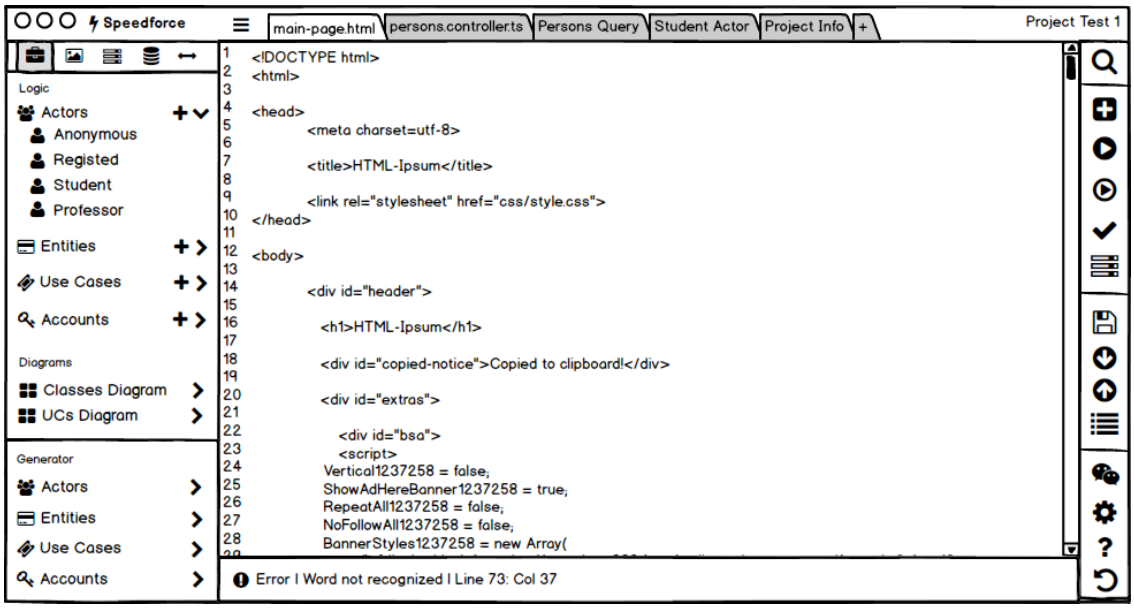
ABAP	Gherkin	Lucene	SCSS
ABC	Gitignore	Makefile	SH
ActionScript	Glsl	Markdown	SJS
ADA	Gobstones	Mask	Smarty
Apache Conf	Go	MATLAB	snippets
AsciiDoc	Groovy	Maze	Soy Template
Assembly x86	HAML	MEL	Space
AutoHotKey	Handlebars	MUSHCode	SQL
BatchFile	Haskell	MySQL	SQLServer
C and C++	haXe	Nix	Stylus
C9Search	HTML	NSIS	SVG
Cirru	HTML (Elixir)	Objective-C	Swift
Clojure	HTML (Ruby)	OCaml	Tcl
Cobol	INI	Pascal	Tex
CoffeeScript	Io	Perl	Text
ColdFusion	Jack	pgSQL	Textile
C#	Jade	PHP	Toml
CSS	Java	Powershell	Twig
Curly	JavaScript	Praat	Typescript
D	JSON	Prolog	Vala
Dart	JSONiq	Properties	VBScript
Diff	JSP	Protobuf	Velocity
Dockerfile	JSX	Python	Verilog
Dot	Julia	R	VHDL
Dummy	LaTeX	Razor	Wollok
DummySyntax	Lean	RDoc	XML
Eiffel	LESS	RHTML	XQuery
EJS	Liquid	RST	YAML
Elixir	Lisp	Ruby	Django
Elm	LiveScript	Rust	
Erlang	LogiQL	SASS	
Forth	LSL	SCAD	
FreeMarker	Lua	Scala	
Gcode	LuaPage	Scheme	

10.2 Mockups da Interface Gráfica

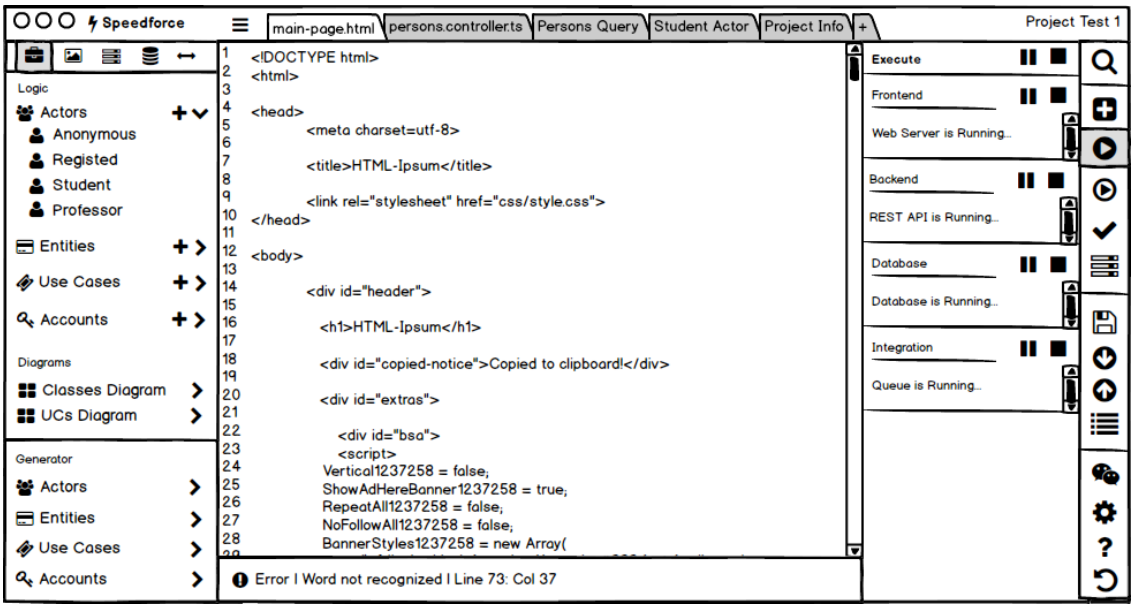
10.2.1 Ecrã Inicial da Aplicação para Gestão de Projetos



10.2.2 Ecrã Principal de Edição de Projeto



10.2.3 Ecrã Principal de Edição de Projeto com Painel Auxiliar Aberto



10.2.4 Ecrã Principal de Edição de Projeto com Painel de Árvore de Itens Fechado