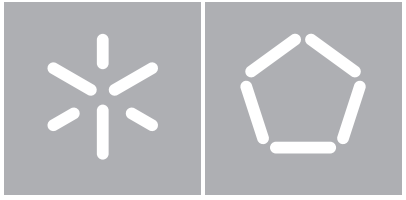
Universidade do Minho
Escola de Engenharia

Nuno Ricardo Arieira Morais

Investigação/Estudo tecnológico para introdução de HTML5 e estruturação do processo de migração

Outubro de 2014



Universidade do Minho
Escola de Engenharia
Departamento específico (facultativo)

Nuno Ricardo Arieira Morais

Investigação/Estudo tecnológico para introdução de HTML5 e estruturação do processo de migração

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor José Carlos Ramalho
Eng. Carlos Silva - Siemens

Outubro de 2014

Agradecimentos

Ao meu orientador, professor José Ramalho, que me acompanhou no desenvolvimento da tese, principalmente na correção da dissertação, um agradecimento especial pelos conselhos que me ajudaram a moldar tanto o meu rumo académico como pessoal. Ao meu supervisor na empresa Siemens S.A, Carlos Silva, pela confiança prestada durante todo o desenvolvimento da tese. Ao Rodrigo Moreira pela orientação, conselhos e ajuda prestada para a construção da tese. A toda a equipa de desenvolvimento da Siemens S.A., do setor Healthcare, Andreia Espírito Santo, Daniel Silva, Joaquim Matos, Pedro Rodrigues, Tiago Vital e ainda ao Carlos Silva e Rodrigo Moreira pela integração na equipa e ajuda disponibilizada.

Um agradecimento especial à minha namorada, Ana Filipa Silva, por toda a compreensão e apoio prestado, pois foi um pilar muito importante durante este percurso. Ainda lhe quero agradecer por todo o tempo disponibilizado na leitura e correção da dissertação.

Um agradecimento especial à minha família, pais e irmã, pelos sacrifícios que fizeram para me apoiar e permitir chegar onde cheguei. Aos meus amigos mais próximos, que sem eles seria difícil encontrar a motivação e sanidade que tanto precisei.

Resumo

Investigação/Estudo tecnológico para introdução de HTML5 e estruturação do processo de migração

O *Diagrammer* é uma aplicação criada pela Siemens S.A., direcionada para os cuidados de saúde, que utiliza a tecnologia Silverlight. O Silverlight consiste numa plataforma de desenvolvimento para escrever e correr aplicações mais complexas para *web*, *workstation* ou dispositivos móveis. Porém, após o surgimento do HTML5 e o anúncio de fim de suporte do Silverlight por parte da Microsoft, as empresas foram obrigadas a ponderar uma mudança de tecnologia para algumas das suas aplicações. A nova versão do HTML já não necessita de um *plugin* auxiliar, possuindo, ainda, outras funcionalidades, como geolocalização, *websockets* ou armazenamento *web*.

Os objetivos deste projeto consistiram em estudar a tecnologia HTML5, de forma a adquirir o máximo de conhecimento da mesma, realizar um estudo comparativo deste com a tecnologia Silverlight, fazendo corresponder algumas vantagens e desvantagens das mesmas, e, finalmente, após a avaliação do impacto da migração para HTML5, estruturar um modelo de migração que possa ser utilizado em futuros projetos.

Para cumprir estes objetivos, procedeu-se a um estudo comparativo entre estas duas tecnologias e a uma análise aprofundada do HTML5, de forma a obter toda a lista de elementos e funcionalidades relacionadas com a mesma. Além disso, ainda foram exploradas algumas das bibliotecas em *JavaScript* e mapeados os controladores em Silverlight para HTML5, terminando com o esboço de um modelo de migração e a migração propriamente dita do caso de estudo com base nesse modelo.

Apesar de ter sido necessário realizar alguns ajustes durante o processo de migração, de forma a melhorar a performance da aplicação, esta foi migrada na sua totalidade com sucesso.

Concluindo, os objetivos propostos para esta dissertação foram cumpridos. No futuro, seria importante o desenvolvimento de uma ferramenta que permitisse, de forma automatizada, realizar parcialmente a migração de aplicações entre tecnologias.

Abstract

Technological investigation/study for the introduction of HTML5 and organization of the migration process

Diagrammer is an application created by Siemens S.A., directed to the healthcare sector, that uses the Silverlight technology. Silverlight is a development tool to write and run more complex applications for web, workstation or mobile devices. However, after the HTML5 appearance and the announcement of the end of Silverlight support by Microsoft, companies were forced to consider a change in technology for some of their applications. The new HTML version doesn't need a helper plugin, and it also has some other features like geolocation, websockets or web storage.

The aims of this project were to study the HTML5 technology, in order to acquire maximum knowledge of it, to do a comparative study between this technology and Silverlight, pointing out some of their advantages and disadvantages, and, finally, after the impact evaluation of the migration to HTML5, to structure a migration model that could be used in future projects.

To fulfill these aims, a comparative study between these two technologies and a depth analysis of HTML5 were made, in order to obtain all the elements and features related to them. Besides that, some JavaScript libraries were explored and the controlers in Silverlight were mapped to HTML5, ending with the outline of a migration model and the real migration of the study case based on that model.

Although it has been necessary to make some adjustements during the migration process, in order to improve the application performance, it was totally and successfully migrated.

In conclusion, all the goals proposed to this dissertation were completed. In the future, it would be important the development of a tool that would allow, in an automated way, to do a partial migration of applications between technologies.

Conteúdo

1 Introdução **1**
- 1.1 Enquadramento . . . 1
- 1.2 Siemens, Setor Healthcare . . . 2
- 1.3 Motivação e Objetivos . . . 3
- 1.4 Estrutura da Dissertação . . . 4

2 Contextualização **5**
- 2.1 Breve História do HTML . . . 5
- 2.2 Silverlight . . . 6
  - 2.2.1 Enquadramento Histórico . . . 6
  - 2.2.2 Objetivos . . . 7
  - 2.2.3 Vantagens . . . 7
  - 2.2.4 Adoção Tecnológica . . . 7
- 2.3 HTML5 . . . 7
  - 2.3.1 Enquadramento Histórico . . . 9
  - 2.3.2 Objetivos . . . 9
  - 2.3.3 Vantagens . . . 9
  - 2.3.4 Adoção Tecnológica . . . 10
  - 2.3.5 Adoção do HTML5 pelos *Browsers* . . . 10

3 Análise **13**
- 3.1 HTML5 . . . 13
  - 3.1.1 Lista de Elementos . . . 13
  - 3.1.2 Estrutura Básica, *DOCTYPE* e *CHARSETs* . . . 19
  - 3.1.3 Funcionalidades . . . 20
    - 3.1.3.1 Novos Elementos Estruturais . . . 20
    - 3.1.3.2 Formulários . . . 23
    - 3.1.3.3 *Canvas* . . . 25
    - 3.1.3.4 Vídeo e Áudio . . . 26
    - 3.1.3.5 *Drag & Drop* . . . 28
    - 3.1.3.6 *Web Storage* . . . 30
    - 3.1.3.7 *Application Cache* . . . 31
    - 3.1.3.8 *Geolocation* . . . 33
    - 3.1.3.9 *Web Sockets* . . . 35
    - 3.1.3.10 *Web Workers* . . . 37
    - 3.1.3.11 *Server-Sent Events* . . . 39
  - 3.1.4 Bibliotecas e/ou *Frameworks* JavaScript . . . 40
    - 3.1.4.1 AngularJS . . . 41
    - 3.1.4.2 KnockoutJS . . . 43
    - 3.1.4.3 KendoUI . . . 45
    - 3.1.4.4 ExtJS . . . 46
    - 3.1.4.5 FabricJS . . . 46
    - 3.1.4.6 KineticJS . . . 47
  - 3.1.5 Mapeamento de controladores em Silverlight para HTML5 . . . 47

4 Caso de Estudo: análise da aplicação Diagrammer **53**

5 Desenvolvimento do caso de estudo **57**
5.1 *Guidelines* para JavaScript . . . 57
5.2 Modelo de Migração . . . 63
5.2.1 *Model* . . . 65
5.2.2 *View* . . . 67
5.2.2.1 *Button* . . . 67
5.2.2.2 *HyperlinkButton* . . . 68
5.2.2.3 *CheckBox*, *RadioButton*, *Slider*, *TextBox*, *PasswordBox*, *Datepicker* . . . 68
5.2.2.4 *ComboBox* e *ListBox* . . . 69
5.2.2.5 *Calendar* . . . 69
5.2.2.6 *TextBlock* . . . 69
5.2.2.7 *ProgressBar* . . . 70
5.2.2.8 *RichTextBlock* e *RichTextBox* . . . 70
5.2.2.9 *AutoCompleteBox* . . . 71
5.2.2.10 *DataGrid* . . . 71
5.2.2.11 *DataPager* . . . 72
5.2.2.12 *TreeView* . . . 72
5.2.2.13 *Image* . . . 73
5.2.2.14 *MediaElement* . . . 73
5.2.2.15 *WebBrowser* . . . 73
5.2.2.16 *Border* . . . 74
5.2.2.17 *Canvas* . . . 74
5.2.2.18 *Grid* . . . 74
5.2.2.19 *GridSplitter* . . . 75
5.2.2.20 *StackPanel* . . . 76
5.2.2.21 *ScrollBar*, *ScrollViewer* . . . 76
5.2.2.22 *TabControl* . . . 76
5.2.2.23 *DescriptionViewer*, *ValidationSummary* . . . 77
5.2.2.24 *Label* . . . 78
5.2.2.25 *ToolTip* . . . 78
5.2.2.26 *OpenFileDialog*, *SaveFileDialog* . . . 79
5.2.2.27 *ChildWindow*, *Popup* . . . 79
5.2.2.28 *RepeatButton*, *DrawingSurface*, *MultiScaleImage*, *ViewBox*, *Frame*, *Page* . . . 80
5.2.3 *ViewModel* . . . 80

6 Resultados da migração **83**
6.1 Resultados HTML5 . . . 83

7 Conclusões e Trabalho Futuro **85**

Glossário

AJAX Asynchronous Javascript and XML

API Application Programming Interface

CLR Common Language Runtime

CSS Cascading Style Sheets

DOM Document Object Model

DRM Digital Rights Management (Gestão de Direitos Digitais)

DTD Document Type Definition

GPS Global Positioning System (Sistema de Posicionamento Global)

GPU Graphics Processing Unit (Unidade de Processamento Gráfico)

HTML HyperText Markup Language (Linguagem de Marcação de Hipertexto)

HTTP HyperText Transfer Protocol

IDE Integrated Development Environment

IE Internet Explorer

JS JavaScript

MB MegaByte

MVC Model-View-Controller

MVP Model-View-Presenter

MVVM Model-View-ViewModel

NORAD North American Aerospace Defense Command

OS Operative System (Sistema Operativo)

REST Representational State Transfer

RIA Rich Internet Application

SEO Search Engine Optimization

SGML Standard Generalized Markup Language

SOAP Simple Object Access Protocol

SQL Structured Query Language

SVG Scalable Vector Graphics (Gráficos de Vetores Escaláveis)

UI User Interface (Interface de Utilizador)

WCF Windows Communication Foundation

WEB World Wide Web

WHATWG Web Hypertext Application Technology Working Group

W3C World Wide Web Consortium

WebGL Web Graphics Library

Lista de Figuras

2.1 Diagrama das APIs e tecnologias associadas ao HTML5 até junho de 2014 (adaptado) [24] 8

3.1 Possível estruturação de uma página HTML através dos novos elementos. 20
3.2 Comparação das mensagens de erro entre Google Chrome e Firefox. . . 23
3.3 Representação de um rectângulo no canvas em HTML5. 25
3.4 Representação de *drag-and-drop* em HMTL5. 29
3.5 Autorização de *geolocation* apresentada ao utilizador. 34
3.6 Apresentação das coordenadas após o utilizador aceitar partilhar a sua localização. 36

4.1 Variante *Passive View* do padrão *Model-View-Present* 53

5.1 Representação da organização em pastas de uma aplicação em HTML5. 64
5.2 Entidades do Projeto. 66

6.1 Diagrammer em Silverlight à esquerda e versão em HTML5 à direita . . 83

Lista de Tabelas

3.1 Instrução especial. . . . 13
3.2 Elemento Principal. . . . 13
3.3 Metadados do documento. . . . 14
3.4 Elementos de Scripting. . . . 14
3.5 Secções. . . . 15
3.6 Agrupamento de conteúdo num documento. . . . 15
3.7 Semântica a nível do texto. . . . 16
3.8 Elemento Principal. . . . 17
3.9 Incorporar Conteúdo. . . . 17
3.10 Tabelas em HTML5. . . . 18
3.11 Formulários. . . . 18
3.12 Elemento Principal. . . . 19
3.13 Versões dos *browsers* compatíveis com a nova semântica [140]. . . . 23
3.14 Versões dos *browsers* compatíveis com validação de formulários [141]. . . . 24
3.15 Versões dos *browsers* compatíveis com o elemento canvas [142]. . . . 26
3.16 Formatos de vídeo e áudio descodificados por cada *browser* [143], [144], [145], [146]. . . . 27
3.17 Versões dos *browsers* compatíveis com o elemento drag-and-drop [148]. 29
3.18 Versões dos *browsers* compatíveis com o elemento *web storage* [149]. . . . 31
3.19 Versões dos *browsers* compatíveis com o elemento *Application Cache* [150]. 33
3.20 Versões dos *browsers* compatíveis com o elemento *geolocation* [151] . . . 35
3.21 Versões dos *browsers* compatíveis com o elemento *web sockets* [152]. . . . 37
3.22 Versões dos *browsers* compatíveis com o elemento *web worker* [153]. . . . 38
3.23 Versões dos *browsers* compatíveis com o elemento *server-sent events* [154]. 40
3.24 Relação de controladores com funcionalidade de botão em HTML5. . . . 47
3.25 Relação de controladores com funcionalidade de seleção em HTML5. . . . 48
3.26 Relação de controladores com funcionalidade de calendário em HTML5. 48
3.27 Relação de controladores com funcionalidade de apresentar informação em HTML5. . . . 48
3.28 Relação de controladores com funcionalidade de apresentar e editar informação em HTML5. . . . 49
3.29 Relação de controladores com funcionalidade de apresentar informação em HTML5. . . . 49
3.30 Relação de controladores com funcionalidade de apresentar gráficos e imagens em HTML5. . . . 49
3.31 Relação de controladores com funcionalidade de apresentar páginas HTML externas em HTML5. . . . 50
3.32 Relação de controladores com funcionalidade de dispor e agrupar elementos em HTML5. . . . 50
3.33 Relação de controladores com funcionalidade de melhorar a interação com o utilizador em HTML5. . . . 50
3.34 Relação de controladores com funcionalidade de navegação em HTML5. 51
3.35 Relação de controladores com funcionalidade de navegação em HTML5. 51

Lista de Exemplos

3.1 Estrutura básica de uma página. 19
3.2 Cabeçalho de uma página. 21
3.3 Navegação de uma página. 21
3.4 Artigo numa página. 21
3.5 Parte lateral numa página. 22
3.6 Parte lateral numa página. 22
3.7 Formulário de email. 23
3.8 Desenho em canvas. 25
3.9 Representação de um vídeo em HTML5. 27
3.10 Representação de um exemplo *drag-and-drop* em HTML5. 28
3.11 Representação de um exemplo drag-and-drop em HTML5. 30
3.12 Representação da informação relacionada ao caminho do ficheiro manifest. 31
3.13 Representação da informação relacionada com o ficheiro manifest. . . . 32
3.14 Representação um exemplo de *geolocation* em HTML5. 33
3.15 Representação um exemplo de *web socket* em HTML5. 36
3.16 Representação de um exemplo para instanciar um *web worker* em HTML5. 38
3.17 Representação de um *web worker* em HTML5. 38
3.18 Representação de um exemplo de um *server-sent event* em HTML5. . . 39
3.19 Representação de *data binding* através de um atributo em AngularJS. . 41
3.20 Representação de *data binding* através de dupla chaveta em AngularJS. 41
3.21 Representação de validação de um formulário em AngularJS. 41
3.22 Representação de *templating* em AngularJS. 42
3.23 Representação de *data binding* em KnockoutJS. 43
3.24 Representação de um exemplo básico de validação em KnockoutJS. . . 44
3.25 Representação de um exemplo básico de *templating* em KnockoutJS. . . 44

5.1 Indentação em JavaScript. 57
5.2 Strings em JavaScript. 57
5.3 Vírgula flutuante em JavaScript. 57
5.4 Declaração de variáveis em JavaScript. 58
5.5 Espaçamento em JavaScript. 58
5.6 Espaçamento em JavaScript. 58
5.7 Declaração de objetos em JavaScript. 59
5.8 Comentários em JavaScript 59
5.9 Nomes de variáveis em JavaScript. 59
5.10 Nomes de funções em JavaScript. 60
5.11 Nome de objetos em JavaScript. 60
5.12 Nome de objetos em JavaScript. 60
5.13 *Strict Mode* em JavaScript. 60
5.14 Comparações em JavaScript. 60
5.15 *If* aninhado em JavaScript. 61
5.16 *Instruções* em JavaScript. 61
5.17 *Return* em JavaScript. 61
5.18 *If* em JavaScript. 61

5.19 Ciclo *For* em JavaScript. 62
5.20 Ciclo *while* e *do while* em JavaScript. 62
5.21 Instrução de *switch* em JavaScript. 62
5.22 Instrução de *try/catch* em JavaScript. 63
5.23 Configuração do RequireJS. 64
5.24 Model *ShapesForm* em *JavaScript*. 66
5.25 Extensão de uma classe. 67
5.26 Implementação de um botão em Silverlight. 67
5.27 Implementação de um botão em HTML5. 68
5.28 Implementação de um *HyperlinkButton* em Silverlight. 68
5.29 Implementação de um botão com hiperligação em HTML5. 68
5.30 Implementação de uma *CheckBox* em Silverlight. 68
5.31 Implementação de uma *CheckBox* em HTML5. 68
5.32 Implementação de uma *ComboBox* em Silverlight. 69
5.33 Implementação de uma *ComboBox* em HTML5. 69
5.34 Implementação de uma *Calendar* em Silverlight. 69
5.35 Implementação de um *Calendar* em HTML5. 69
5.36 Implementação de uma *TextBlock* em Silverlight. 70
5.37 Implementação de uma *TextBlock* em HTML5. 70
5.38 Implementação de uma *ProgressBar* em Silverlight. 70
5.39 Implementação de uma *ProgressBar* em HTML5. 70
5.40 Implementação de uma *RichTextBox* em Silverlight. 70
5.41 Implementação de uma *RichTextBox* em HTML5. 71
5.42 Implementação de uma *AutoCompleteBox* em Silverlight. 71
5.43 Implementação de uma *AutoCompleteBox* em HTML5. 71
5.44 Implementação de uma *DataGrid* em Silverlight. 71
5.45 Implementação de uma *DataGrid* em HTML5. 72
5.46 Implementação de uma *TreeView* em Silverlight. 72
5.47 Implementação de uma *TreeView* em HTML5. 72
5.48 Implementação de uma *Image* em Silverlight. 73
5.49 Implementação de uma *Image* em HTML5. 73
5.50 Implementação de um *MediaElement* em Silverlight. 73
5.51 Implementação de um *MediaElement* em HTML5. 73
5.52 Implementação de um *WebBrowser* em Silverlight. 73
5.53 Implementação de um *WebBrowser* em HTML5. 74
5.54 Implementação de um *Canvas* em Silverlight. 74
5.55 Implementação de um *Canvas* em HTML5. 74
5.56 Implementação de um *Grid* em Silverlight. 74
5.57 Implementação de um *Grid* em HTML5. 75
5.58 Implementação de um *GridSplitter* em Silverlight. 75
5.59 Implementação de um *GridSplitter* em HTML5. 76
5.60 Implementação de um *StackPanel* em Silverlight. 76
5.61 Implementação de um *StackPanel* em HTML5. 76
5.62 Implementação de um *TabControl* em Silverlight. 77
5.63 Implementação de um *TabControl* em HTML5. 77
5.64 Implementação de um *DescriptionViewer* em Silverlight. 77
5.65 Implementação de um *DescriptionViewer* em HTML5. 78
5.66 Implementação de um *Label* em Silverlight. 78

5.67 Implementação de um *Label* em HTML5. 78
5.68 Implementação de um *ToolTip* em Silverlight. 78
5.69 Implementação de um *ToolTip* em HTML5. 78
5.70 Implementação de um *OpenFileDialog* em Silverlight. 79
5.71 Implementação de um *OpenFileDialog* em HTML5. 79
5.72 Implementação de um *ChildWindow* em Silverlight. 79
5.73 Implementação de um *ChildWindow* em HTML5. 79
5.74 Implementação de um *ViewModel* em HTML5. 80
5.75 Declaração do *Binding* na *view* em Silverlight. 80
5.76 Implementação do *Binding* em Silverlight. 81
5.77 Declaração do *Binding* na *View* em HTML5. 81
5.78 Implementação do *Binding* no *ViewModel* em HTML5. 81

1 Introdução

O capítulo inicia com um enquadramento desta dissertação, apresentada na secção 1.1, a qual foi desenvolvida no âmbito da unidade curricular Dissertação, lecionada no curso de Mestrado em Engenharia Informática (MEI) da Universidade do Minho. Seguidamente, na secção 1.2 é apresentada a empresa Siemens, na secção 1.3 podem ser vistos os objetivos e motivação e, por fim, na secção 1.4 encontra-se presente uma visão geral da estrutura desta dissertação.

1.1 Enquadramento

Ao longo dos últimos anos, a sociedade tem sido confrontada com uma transição gradual da informação em papel, na maioria das vezes fragmentada e de difícil acesso, para centros de dados informáticos onde esta informação se pode encontrar reunida e disponível simultaneamente para vários terminais e a qualquer hora. Essa informação pode ser pública ou restrita, e estar ou não encriptada nesses centros, podendo ser acedida normalmente através de aplicações, que podem ser manipuladas por utilizadores ou ser autónomas, e que a usam para diversos fins.

As aplicações podem ser desenvolvidas de forma nativa ou mais genéricas, ou seja, híbridas, que funcionam tanto para desktop como para mobile, sendo que as nativas são desenvolvidas para cada sistema operativo (OS), enquanto que as híbridas funcionam nos demais OS com uma única implementação, através de um *browser*. Estas últimas têm vindo a crescer, principalmente com o aparecimento da nova versão do *HyperText Markup Language 5* (HTML5) e o desenvolvimento de várias frameworks em JavaScript (JS), que têm vindo a transformar o mercado, tanto em funcionalidade como em competências técnicas necessárias para criar aplicações híbridas.

Nestes últimos anos temo-nos deparado com uma enorme evolução na mentalidade da população face à utilização de aplicações, e muitas são as empresas que têm vindo a apostar e a desenvolver aplicações para web, porém em equipamentos móveis estas continuam a não ter grande adesão. Isto tem deixado as empresas reticentes quanto às tecnologias que adotam, principalmente em relação ao suporte no futuro, levando-as, muitas vezes, a proceder a migrações pelas mais variadas razões.

Com base na transição do papel para formato digital, um dos setores que mais poderá beneficiar com esta evolução tecnológica é o da Saúde, já que toda a documentação clínica relativa a um doente poderá ser facilmente consultada e atualizada por diferentes profissionais de saúde. Neste sentido, e com o objetivo de estruturar toda a informação clínica e minimizar o tempo necessário para o seu registo, a Siemens S.A. - Setor Healthcare desenvolveu uma aplicação, o *Diagrammer*, que pretende fornecer uma perspetiva cronológica e global das patologias registadas para cada doente.

O *Diagrammer* consiste numa aplicação de suporte ao profissional clínico prestador de cuidados de saúde, que recorre a diagramas representativos de uma região anatómica humana, permitindo o registo gráfico de todo um conjunto de patologias de um determinado doente, minimizando o tempo e esforço necessários para o fazer, e simplificando a comunicação entre os profissionais de saúde. Esta aplicação disponibiliza aos profissionais de saúde as ferramentas necessárias para a criação, consulta e edição de um diagrama, permitindo também a personalização, por perfil de utilizador, dos diagramas disponíveis.

O *Diagrammer* está desenvolvido sobre a tecnologia Silverlight, porém com o aparecimento do HTML5 e as suas novas e vantajosas funcionalidades, a Siemens S.A. - Setor Healthcare pretende fazer a atualização da aplicação para esta última tecnologia.

1.2 Siemens, Setor Healthcare

A Siemens está em Portugal há mais de 105 anos, sendo líder no fornecimento de soluções de engenharia nos setores de Indústria, Energia, Saúde e Infra-estruturas & Cidades. Com cerca de 2000 colaboradores, duas unidades de produção e numerosas parcerias com o meio académico, a empresa desempenha um papel ativo no desenvolvimento económico do país.

O Setor Energy é líder mundial no fornecimento de toda a gama de produtos, serviços e soluções para a produção de energia em centrais termoelétricas, aproveitamento de energias renováveis, transmissão de energia e ainda tecnologias nas áreas da extração, conversão e transporte de petróleo e gás.

O Setor Industry é um dos maiores fornecedores mundiais de produtos e soluções inovadoras e ecológicas para clientes industriais. Com uma gama completa de tecnologias de automação, soluções de software inteligente, vasta experiência industrial e serviços integrados complexos, o Setor promove a produtividade, eficiência e flexibilidade dos nossos clientes e, ao mesmo tempo, reforça a sua competitividade.

Com um portefólio que inclui soluções de mobilidade integrada, sistemas de automação e de segurança para edifícios, equipamento de distribuição de energia, aplicações de rede elétrica inteligente e produtos de baixa e média tensão, o novo Setor Infrastructure & Cities oferece tecnologias sustentáveis para centros metropolitanos e infraestruturas urbanas em todo o mundo.

O Setor Healthcare é um dos maiores fornecedores mundiais na indústria dos cuidados de saúde e líder em sistemas de imagiologia, diagnósticos laboratoriais, tecnologia de informação médica e aparelhos auditivos. É a primeira empresa a nível mundial a oferecer um portefólio integrado de tecnologia que permite responder a todas as fases do ciclo de cuidados de saúde, disponibilizando produtos e soluções para todo o tipo de cuidados ao paciente, desde a prevenção e diagnóstico precoce a situações de pós-tratamento. Ao otimizar os procedimentos clínicos associados às mais importantes

condições clínicas, permite tornar os cuidados de saúde mais rápidos, melhores e mais rentáveis.

Em Portugal, o Setor Healthcare da Siemens é um dos líderes de mercado na área da saúde, reconhecido pelas suas competências, know-how e força de inovação em diagnóstico e tecnologias terapêuticas, assim como engenharia de conhecimento, incluindo tecnologias de informação, integração de sistemas e serviços de consultoria.

Nos últimos anos, o Setor Healthcare da Siemens tem promovido uma estratégia de contacto e parceria com a Comunidade Académica e Científica em Portugal, no sentido da criação de uma rede de conhecimento e parcerias estratégicas que potenciem a inovação, a investigação e o desenvolvimento (IDI) na área da Saúde.

1.3 Motivação e Objetivos

Nos últimos meses, a Microsoft anunciou o fim do suporte do Silverlight, tendo este data marcada para 2021. Com isto, as empresas que desenvolviam em Silverlight já ponderam em migrar o seu software para outras tecnologias. Atualmente, face ao grande desenvolvimento, o HTML5 está a tornar-se muito poderoso, podendo esta ser uma das possibilidades tecnológicas para substituir o Silverlight.

Os objetivos desta dissertação assentam no propósito de aumentar o conhecimento sobre HTML5, tanto pessoal como de outrem, através da descrição do seu estado da arte. Também se pretende realizar um estudo comparativo do HTML5 com outras tecnologias, neste caso em particular o Silverlight, analisando e sistematizando vantagens e limitações associadas, tendo por base a Internet, artigos relevantes publicados na área, entre outros.

Adicionalmente, será ainda realizada uma avaliação do impacto da migração/transição de Silverlight para HTML5, onde será efetuada uma análise de um caso de estudo baseado na tecnologia Silverlight, sistematizando as necessidades e impacto do processo de migração em termos de complexidade para os principais componentes/controlos associados. Para tal, será indispensável estudar aprofundadamente estas duas ferramentas, de forma a compreender a sua ligação entre objetos gráficos, bem como estudar *frameworks* e ferramentas necessárias para todo o desenvolvimento da aplicação em HTML5.

Por fim, será estruturado um modelo para futuros projetos, de forma a conseguir um maior conhecimento para estimativas de migrações entre estas tecnologias.

1.4 Estrutura da Dissertação

Esta dissertação é composta por seis capítulos, sendo que o planeamento do trabalho para a concretização da dissertação organizou-se de acordo com as seguintes partes.

Introdução Este capítulo apresenta um breve enquadramento tecnológico sobre o tema HTML5, seguido da apresentação da empresa *Siemens* 1.2. A motivação e objetivos estão descritos na secção 1.3, procedidos da visão geral da estrutura desta dissertação na secção 1.4.

Contextualização Este capítulo apresenta uma breve história acerca do HTML 2.1 e um pouco da história de arte do Silverlight 2.2 e HTML5 2.3. Em ambas as tecnologias foi elaborado um breve enquadramento histórico, objetivos, vantagens e adoção tecnológica. Na secção do HTML5 ainda foi adicionada uma subsecção de *Adoção do HTML5 pelos browsers* 2.3.5.

Análise Este capítulo apresenta uma análise ao HTML5 3.1, abordando toda a sua lista de elementos 3.1.1, estrutura básica 3.1.2 e funcionalidades 3.1.3, sendo ainda estudadas algumas bibliotecas e *frameworks* em JavaScript 3.1.4 e elaborado um mapeamento de controladores em Silverlight para HTML5 3.1.5.

Caso de Estudo Este capítulo apresenta uma breve apresentação acerca do caso de estudo, sobre as suas funcionalidades e estrutura.

Desenvolvimento do caso de estudo Este capítulo apresenta algumas *guidelines* para escrever em JavaScript 5.1 e, seguidamente, o processo de migração 5.

Resultados Neste capítulo são abordados os resultados obtidos, as dificuldades enfrentadas e as soluções que foram utilizadas ao longo da migração do caso de estudo.

Conclusão e trabalho futuro Neste capítulo são apresentadas as conclusões desta tese, com uma visão geral sobre este processo de migração de Silverlight para HTML5.

2 Contextualização

Este capítulo permite contextualizar o leitor sobre o tema da dissertação. São abordados os conceitos principais, com uma breve história do *HyperText Markup Language* (HTML), e o surgimento tanto do Silverlight como do HTML5.

2.1 Breve História do HTML

Tim Berners-Lee, no início dos anos 90, lançou a primeira descrição pública de HTML, sendo esse um documento denominado por "HTML *Tags*" [1] [2], onde descrevia dezoito elementos simples definidos por um conjunto de regras sintáticas flexíveis.

A sigla HTML significa *HyperText Markup Language* ou, em português, Linguagem de Marcação de Hipertexto [3]. A primeira versão foi baseada na linguagem *Standard Generalized Markup Language* (SGML), que era usada para a estruturação de documentos, tendo herdado do SGML as *tags* de título (<h1> a <h6>), cabeçalho (<head>) e parágrafo (<p>). A grande diferença entre SGML e HTML era a existência da tag "<a>", que usava o atributo "href", permitindo o uso de endereços para a interligação entre documentos, que era a base de funcionamento da web [4]. Dos dezoito elementos inicialmente lançados, onze deles ainda hoje se mantêm inalterados.

Até 1993 foram lançados alguns *browsers* como o Cello, Arena, Lynx, tkWWW e Mosaic, tendo sido este último renomeado de Netscape um ano depois, tornando-se um sucesso [5]. Porém, só em 1995 se criou o primeiro conjunto oficial de *standards* do HTML, o que permitiu a sua atualização para a versão HTML 2.0 [6].

Entre 1996 e 1997 foram adicionadas novas funcionalidades importantes, surgindo então as tabelas e mapas de imagens do lado do cliente, e o HTML 3.2 [7] foi publicado sob Recomendação *World Wide Web Consortium* (W3C). Com poucas alterações surgiu, um ano mais tarde, o HTML4 [8], também com Recomendação W3C, e em 1999 foi lançada a versão 4.01, que obteve em finais de dezembro a última Recomendação W3C [9], que se mantém até à atualidade (junho de 2014).

O HTML é uma linguagem de *markup* que os web *browsers* usam para interpretar e escrever texto, imagens e outra informação visual ou sonora em páginas web [10]. Cada *web browser* começou por definir as regras para o HTML em 1997, e as páginas podiam ser melhoradas através de folhas de estilo *Cascading Style Sheets* (CSS) [11].

O propósito do *browser* é interpretar um documento HTML recebido por um servidor e apresentar a informação que este contém. Como se pode constatar, o *browser* não apresenta as *tags* HTML, mas utiliza-as para interpretar e formatar o conteúdo da página.

As *tags* foram especificadas para seguirem certas regras, de forma a que os *browsers* consigam recriar as páginas. As *tags* são compostas por palavras-chave entre os símbolos de menor '<' e maior '>', respetivamente, e normalmente encontram-se presentes em pares, sendo que à segunda é acrescentada uma barra '/' para fechar a primeira (Ex:. Isto é um texto a bold). As *tags* podem conter atributos, tal como no *EXtensible Markup Language* (XML). Por exemplo, a *tag* de hiperligação '<a>' tem o atributo *href*, que indica para onde essa hiperligação aponta (Ex:. <a href= "www.google.com»Google).

2.2 Silverlight

O Silverlight consiste numa plataforma de desenvolvimento para escrever e correr aplicações mais complexas orientadas para a *web*, *workstation*, ou dispositivos móveis [12]. É multiplataforma e necessita do *plugin* instalado para correr as aplicações, compatível com vários *browsers*, dispositivos e OSs, possuindo inúmeros recursos, incluindo suporte à *webcam*, microfone ou impressora [13].

2.2.1 Enquadramento Histórico

Em 2007 foi lançada a primeira versão do Silverlight[13], sendo então responsável pela interface do utilizador (UI), interatividade e *input*, controlos básicos de UI, gráficos e animações, reprodução de vídeo e música, gestão de direitos digitais (DRM) e integração *Document Object Model* (DOM) [14].

A segunda versão foi reestruturada, tendo sido introduzida uma *framework* .NET, implementada na mesma máquina virtual, denominada por *Common Language Runtime* (CLR), e que permitia executar programas escritos em qualquer linguagem .NET [15]. Até ao lançamento do Silverlight 4, apenas foram incrementados novos controladores e funcionalidades dos mesmos[16]. Porém, nestas duas últimas versões (4 e 5) foram implementados alguns recursos importantes, como o suporte a *webcam*, microfone e impressoras, novas funcionalidades à interação com o rato, melhorias tanto no *RichTextBox* como no *DataGrid*, possibilidade de localização, *drag-and-drop*, suporte para aceleração da Unidade de Processamento Gráfico (GPU), gráficos 3D, integração de controlo remoto, suporte para *browsers* de *64bits*, e ainda possibilidade de fazer *debug* [17].

O Silverlight é, atualmente, compatível com alguns dos *browsers* mais recentes, como o Internet Explorer (IE), Safari, Google Chrome e Mozilla Firefox, sendo que o Opera não integra esta lista [18], apenas conseguindo ser suportado com o recurso a fontes não oficiais.

Também a Microsoft tem abandonado o desenvolvimento do Silverlight, sendo que, recentemente, a empresa lançou um comunicado onde informava o fim do suporte deste

para o ano 2021 [19]. No entanto, continua a lançar constantemente novos *patchs* para as versões atuais. [19]

2.2.2 Objetivos

O Silverlight surgiu com o intuito de adicionar outras funcionalidades que o HTML, por si só, não conseguiria oferecer, como o suporte a conteúdos de vídeo e som de alta qualidade, de forma a melhorar a interação entre a aplicação e o utilizador, e ainda rentabilizar os recursos do OS e do equipamento utilizados [17].

No momento do seu lançamento, o *Siverlight* veio competir com duas tecnologias já bastante utilizadas, o *Flex* e o *Flash*, ambas desenvolvidas pela *Adobe*, tendo sido, por isso, preterido em muitas situações [17].

2.2.3 Vantagens

O Silverlight, embora seja pesado e necessite de um *plugin* para correr na máquina cliente, possui várias vantagens. Nele estão contidos muitos controladores bastante desenvolvidos e uma template *responsive*, isto é, que se adapta ao tamanho de cada equipamento [17]. Além disso, como apresenta controladores avançados, não é necessário empreender um grande esforço para o desenvolvimento de projetos de grande dimensão. Outra das vantagens do Silverlight é que é multiplataforma, porém alguns dispositivos móveis não possuem o *plugin* necessário, e este também não será desenvolvido [13].

2.2.4 Adoção Tecnológica

Com a instabilidade atual do HTML5, muitos projetos ainda adotam o Silverlight. Por exemplo, a empresa *eDreams Edusoft* decidiu apostar nesta tecnologia para o desenvolvimento da *Funtoot*, uma aplicação de ensino personalizada [20]. Também a *Kinectic* continua a apostar na tecnologia Silverlight no seu projeto *Aureus* [21], uma aplicação que permite ajudar os anunciantes a identificar o seu público alvo com base em alguns fatores, e que além de usar como recurso o *Bing Maps*, ainda utiliza outras ferramentas da Microsoft, como o *SharePoint* e o *Office*.

2.3 HTML5

O HTML5 é uma linguagem de marcação (*Markup Language*), usada para estruturar e apresentar conteúdo na *web*, e um *core* tecnológico da internet [10]. Esta é a quinta versão do *standard* HTML, e encontra-se na fase de *proposed recommendation*

da W3C [22]. Além de permitir especificar marcação, o HTML5 também especifica *Application Programming Interfaces* (*APIs*) de *scripting*, e as interfaces do DOM foram extendidas [10].

O HTML5 não funciona apenas com HTML, necessitando igualmente de JS e CSS para estilizar, dinamizar e ainda usar as APIs disponibilizadas [23]. O CSS é utilizado para especificar a aparência e a formatação de um documento HTML, sendo o seu principal benefício promover a separação entre a formatação e o conteúdo de um documento. Embora o CSS não seja uma tecnologia indispensável para o HTML5, este pode realmente apresentar as funcionalidades do HTML5 de uma maneira mais agradável ao utilizador. O JS marcou o início da era do *client-side scripting* das páginas *web*. Permite transformar o HTML em algo mais rico e com mais potencial, e é apenas com esta tecnologia que é possível que algumas das potencialidades do HTML5 possam ser exploradas. O JS permite, de forma dinâmica, editar, criar ou remover informação contida numa página HTML.

Esta versão 5 do HTML, além das novas funcionalidades, trouxe consigo mundanças importantes, nomeadamente a nível da semântica e acessibilidade, incorporando novos recursos, que anteriormente só eram possíveis recorrendo a outras tecnologias (ex.: *Flash*) [23].

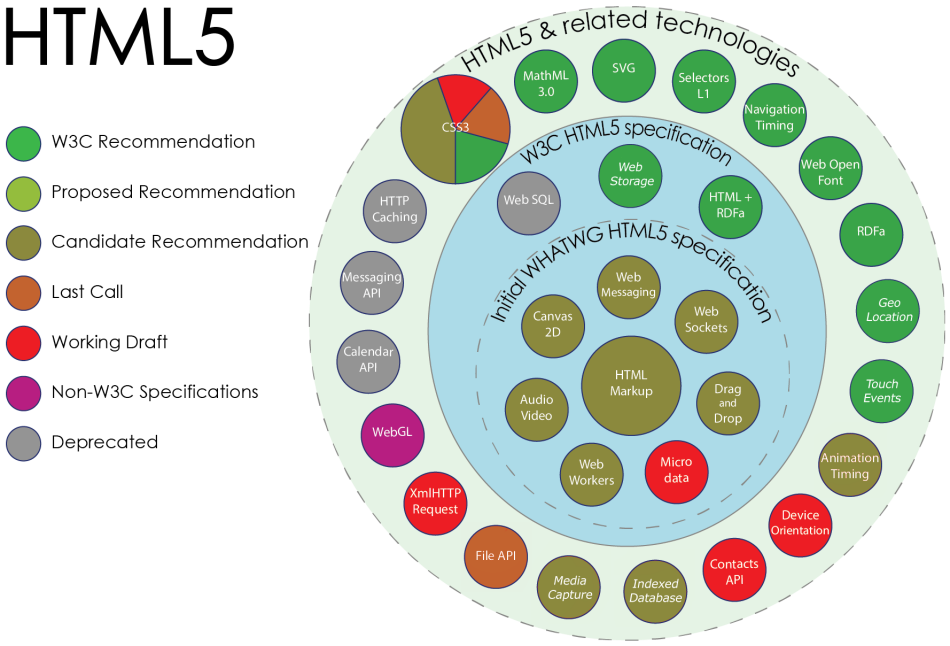


Figura 2.1: Diagrama das APIs e tecnologias associadas ao HTML5 até junho de 2014 (adaptado) [24]

Esta nova versão do HTML permite que, de forma nativa, ou seja, sem qualquer instalação de *plugins*, seja fornecido um conjunto de funcionalidades ao HTML (Figura 2.1), de entre as quais se podem realçar: o elemento *Canvas*, que permite desenhar de imediato em modo 2D; a reprodução de vídeo e música; o desenvolvimento de aplicações em modo *offline*; a edição de documentos; o *drag-and-drop*; o *Web Messaging*; o

Microdata; o *Web Storage*, que permite um armazenamento de informação superior ao dos *cookies*; a *Geolocalização*; a API para o tratamento de uploads e manipulação de ficheiros; e a API para o processamento e síntese de áudio em aplicações Web.

2.3.1 Enquadramento Histórico

A primeira versão *DRAFT* pública do HTML5 surgiu em 2008 pelo esforço de algumas empresas que formavam o *Web Hypertext Application Technology Working Group* (WHATWG), e que eram as primeiras interessadas na evolução do HTML e nas tecnologias ligadas à mesma [25].

Já em julho de 2012, a WHATWG e a W3C decidiram estabelecer um grau de separação, de forma a que a W3C continuasse a trabalhar na especificação do HTML5 e a WHATWG na *standardização*, a que deram o nome de "*Living Standard*", que tem como objetivo nunca estar completo, podendo ser sempre atualizado e melhorado, pela adição de recursos, mas sem nunca serem removidas as funcionalidades [26]. Já em setembro deste ano o HTML5 foi designado como *Proposed Candidate Recommendation* [22].

2.3.2 Objetivos

O HTML5 nasceu de necessidades notórias presentes no ecossistema dos *browsers*, e todos os objetivos das suas especificações surgiram em resposta a essas necessidades[27]. De certa forma, esta versão foi desenvolvida com o propósito de abranger algumas lacunas do HTML, que eram mascaradas por outras tecnologias e/ou *plugin*s, nomeadamente o *Flex*, o *Flash* e o Silverlight[27]. Também veio impulsionar a semântica da WEB, adicionando importantes componentes como tags, forms e *drag & drop* [27]. Outro dos objetivos do HTML5 é permitir a execução de um maior número de tarefas com um menor código, o que permite generalizar algumas das funcionalidades básicas, como, por exemplo, o texto do *placeholder* nos formulários[27].

2.3.3 Vantagens

As vantagens desta tecnologia passam também pelos objetivos que se pretendem atingir. Com o HTML5 é possível desenvolver aplicações universais, isto é, que funcionam em qualquer OS, sendo apenas necessário o tradicional *browser* com suporte ao mesmo [28]. As aplicações desenvolvidas com esta tecnologia possuem o melhor dos dois mundos, isto é, apresentam vantagens nos serviços de *Internet*, quando o equipamento está ligado à rede, mas são igualmente dotadas de características dos programas nativos, como a reprodução de som e vídeo, o armazenamento de informação no equipamento e também a interação com o próprio utilizador [29].

Trata-se de uma tecnologia bastante leve a nível de recursos usados da máquina, até porque corre sobre um *browser*, logo poderá ser interpretado por equipamentos com recursos muito limitados [29].

Outra das grandes vantagens está relacionada com o *Search Engine Optimization* (SEO) e a semântica, pois além de estruturar a aplicação, o HTML5 também possui uma relação direta com o conteúdo [30]. O HTML5, com os seus novos elementos, permite estruturar, de uma maneira muito simples, o que aparece em cada zona da página. Uma página poderá ser dividida em várias partes: cabeçalho, conteúdo e fundo. Até à versão 4.1 do HTML, para se fazer a divisão dessas três simples partes, era necessário recorrer ao elemento "div", e a Class ou ID poderia informar o tipo de divisão aplicado. Na versão atual, o HTML já possui tags específicas, como "header", "section"e "footer", entre outras, tornando a estrutura mais sucinta. Contudo, ainda é possível acrescentar microdados à estrutura, permitindo aos motores de busca compreender a informação constante na *tag* e relacioná-la.

2.3.4 Adoção Tecnológica

Algumas empresas já migraram as suas aplicações de Silverlight para HTML5, como é o caso da Microsoft, proprietária do *Bing Maps* (porém continua a ter a versão em Silverlight), ou da *Netflix*, prestadora de serviços de *stream*. No caso da *North American Aerospace Defense Command* (NORAD), esta optou por recorrer ao HTML5 para desenvolver o projeto *Tracks Santa* em 2012, que se revelou um sucesso, pois atingiu os 23 milhões de utilizadores a cada 48 horas, mantendo, em média, um total de 91.5 milhões de visualizações da página durante esse período de tempo. Face a estes resultados tão positivos, a NORAD escolheu manter a tecnologia HTML5 [31].

2.3.5 Adoção do HTML5 pelos *Browsers*

No ínicio do HTML, observou-se uma intensa 'luta' entre *browsers*, em que o principal objetivo desta 'guerra' era cada um implementar as suas funcionalidades. Assim, cada página que fosse desenvolvida por um *developer* apenas seria funcional num único *browser*, levando as pessoas a utilizá-lo. Com o decorrer dos anos, este pensamento tem evoluído, e os *developers* preocupam-se agora que as suas aplicações ou páginas funcionem independentemente do *browser* em que está a ser requisitado. Assim, este conflito tem vindo a esmorecer, e atualmente os *browsers* praticamente já só se preocupam em satisfazer ao máximo os *developers* e os utilizadores dos mesmos. O Mozilla Firefox, Internet Explorer, Google Chrome, Safari e ainda o Opera são os *browsers* mais utilizados atualmente, sendo igualmente aqueles que mais se preocupam com estas questões.

Uma das questões que se coloca quando se fala de suporte do *browser* em relação ao HTML5 é a capacidade de serem compatíveis. Esta dúvida está erroneamente a ser colocada, pois o HTML5 possui um grande conjunto de novas funcionalidades e, na realidade, nenhum dos *browsers* é compatível com a totalidade de funcionalidades do

HTML5. O mais correto é perguntar se uma determinada funcionalidade HTML5 já está implementada em determinada versão do *browser*, já que estas vão sendo implementadas faseadamente nos mesmos.

Assim, são necessárias bibliotecas para verificar a existência destas funcionalidades no *browser* que está a ser requisitado, sendo o *modernizr* uma das mais referenciadas e utilizadas pelos *developers*. Quando se verifica que uma determinada funcionalidade não está implementada, habitualmente os *developers* optam por tentar simular a funcionalidade, implementando a mesma manualmente ou importando alguma biblioteca, sendo normalmente denominado por *fallbacks*.

3 Análise

Neste capítulo são apresentadas diversas análises comparativas, nomeadamente entre a tecnologia HTML5 e a sua versão 4, de forma a obter uma visão geral e identificar quais as funcionalidades disponíveis, entre o Silverlight e o HTML5, e ainda entre *frameworks* em JavaScript, o que possibilita a escolha das mais indicadas para o desenvolvimento do caso de estudo, com base nos seus conceitos, aspetos positivos e negativos e tipo de licença.

3.1 HTML5

3.1.1 Lista de Elementos

Na nova versão do HTML, existem bastantes *tags* que podem ser utilizadas. Foram consideradas doze categorias, de forma a posicionar a informação de forma organizada, que serão representadas nas tabelas abaixo. Em cada tabela encontram-se descritas algumas informações sobre a *tag*, bem como uma pequena definição, qual o seu conteúdo e atributos permitidos.

A tabela 3.1 contém uma instrução especial que permite definir o tipo de documento, sendo que esta é a informação que primariamente deve ser definida. A partir desta instrução o *browser* será capaz de reconhecer em que versão do HTML o documento foi escrito.

Instrução	**Descrição**
<!DOCTYPE>	Definição do tipo de documento. Esta instrução foi alterada em relação à versão atual do HTML [32]

Tabela 3.1: Instrução especial.

A categoria seguinte, constante na tabela 3.2, apenas apresenta um elemento do HTML, sendo este o do topo da hierarquia de um documento. Nele podem estar presentes dois módulos: o *head* e o *body*.

Tag	**Descrição**
<HTML>	Definição do tipo de documento. [33]

Tabela 3.2: Elemento Principal.

A tabela 3.3 contém os elementos que podem representar metadados do documento, definindo assim um conjunto de informações mais relevantes sobre a página e conteúdos publicados.

Tag	Descrição
<head>	Define informação do documento. [34]
<title>	Define o título do documento. [35]
<base>	Define um endereço base para os caminhos relativos presentes no documento. [36]
<link>	Define a relação entre um documento e ficheiros externos. [37]
<meta>	Define metadados sobre o documento. [38]
<style>	Permite estilizar a informação de um documento. [39]

Tabela 3.3: Metadados do documento.

O padrão W3C suporta *scripts* do lado do cliente. Para que o *browser* os execute, estes poderão ser adicionados a partir das *tags* destacadas na tabela de elementos de *scripting* (tabela 3.4).

Tag	Descrição
<script>	Define um script do lado do cliente. [40]
<noscript>	Define conteúdo alternativo para os utilizadores que não permitem scripts do lado do cliente. [41]

Tabela 3.4: Elementos de Scripting.

As secções representadas na tabela 3.5 permitem, tal como o nome informa, definir secções ou zonas. Nesta nova versão do HTML é possível remover os blocos, normalmente representados com *divs* e com o atributo *id* ou *class*, substituindo-os por nomes bem definidos. Metade das *tags* são novas em relação à versão atual do HTML.

Tag	Descrição
<body>	Define o corpo de um documento. [42]
<section>	Define uma secção no documento. (Nova *tag*) [43]
<nav>	Define uma secção de navegação. (Nova *tag*) [44]

<article>	Define uma secção independente do documento. Pode então representar um artigo de um *blog* ou até de revista. (Nova *tag*) [45]
<aside>	Define conteúdo relacionado com o conteúdo principal do documento. (Nova *tag*) [46]
<h1>	Define título da secção. [47]
<h2>	Define título da secção. [48]
<h3>	Define título da secção. [49]
<h4>	Define título da secção. [50]
<h5>	Define título da secção. [51]
<h6>	Define título da secção. [52]
<hgroup>	Define um grupo de cabeçalhos. (Nova *tag*) [53]
<header>	Define o cabeçalho da secção. (Nova *tag*) [54]
<footer>	Define o fundo da secção. (Nova *tag*) [55]
<address>	Define uma secção para informação de contacto. [56]

Tabela 3.5: Secções.

Na tabela 3.6 encontra-se descrito um conjunto de *tags* que permitem agrupar ou organizar conteúdos num documento.

Tag	**Descrição**
<p>	Define um parágrafo. [57]
<hr>	Define uma alteração temática do conteúdo. (*Tag* alterada) [58]
 	Define uma quebra de linha [59]
<pre>	Define texto pré-formatado. [60]
<blockquote>	Define uma secção citada. [61]
	Define uma lista numerada. [62]
	Define uma lista não numerada. [63]
	Define um item da lista. [64]
<dl>	Define a descrição de uma lista. [65]
<dt>	Define um nome ou termo na descrição de uma lista. [66]
<dd>	Define uma descrição de um nome ou termo na descrição de uma lista. [67]
<figure>	Define conteúdo auto-suficiente, como ilustrações, diagramas, fotos, etc. (Nova *tag*) [68]
<figcaption>	Define uma legenda da *tag* <figure>. (Nova *tag*) [69]
	Define um bloco genérico de conteúdo. [70]

Tabela 3.6: Agrupamento de conteúdo num documento.

Tag	Descrição
<a>	Define uma hiperligação. (*Tag* alterada) [71]
	Permite realçar o texto (com estilo itálico) [72]
	Define texto importante. [73]
<small>	Define texto pequeno. (*Tag* alterada) [74]
<s>	Define texto que não está correto. (*Tag* alterada) [75]
<cite>	Define referência de uma citação. (*Tag* alterada) [76]
<q>	Define um bloco de texto citado. [77]
<dfn>	Define um termo. [78]
<abbr>	Define uma abreviação ou acrónimo. [79]
<time>	Define data e/ou hora. (Nova *tag*) [80]
<code>	Define um bloco de código. [81]
<var>	Define uma variável. [82]
<samp>	Define um exemplo de um *output*. [83]
<kbd>	Define o *input* de teclado. [84]
<sub>	Define texto inferior à linha (*subscript*). [85]
<sup>	Define texto superior à linha (*superscript*). [86]
<i>	Define um bloco de texto em itálico sem dar qualquer tipo de relevância superior ao texto envolvente. (*Tag* alterada) [87]
<b>	Define um bloco de texto em negrito sem dar qualquer tipo de relevância superior ao texto envolvente. (*Tag* alterada) [88]
<u>	Sublinha um bloco de texto sem dar qualquer tipo de relevância superior ao texto envolvente. (*Tag* alterada) [89]
<mark>	Define texto marcado ou realçado. (Nova *tag*) [90]
<ruby>	Define anotação em *ruby* (anotação da Ásia Ocidental). (Nova *tag*) [91]
<rt>	Define texto em *ruby*. (Nova *tag*) [92]
<rp>	Define parêntesis em volta do texto em *ruby*. (Nova *tag*) [93]
<bdi>	Isola uma parte do texto que pode ser formatado em diferentes direções do texto exterior. (Nova *Tag*) [94]
<bdo>	Sobrepõe a direção do texto atual.[95]
	Define uma secção no documento. [96]

Tabela 3.7: Semântica a nível do texto.

Na tabela abaixo (Tabela 3.8) encontram-se referenciadas as *tags* necessárias para destacar a edição de texto.

Tag	Descrição
<ins>	Define o texto que foi inserido no documento. [97]

<del>	Define o texto que foi eliminado no documento.[98]

Tabela 3.8: Elemento Principal.

Na tabela 3.9 constam as *tags* suportadas para incoporar conteúdo, nativo ou não, no documento.

Tag	Descrição
<img>	Representa uma imagem no documento. [99]
<iframe>	Define um novo contexto de navegação integrado no documento. [100]
<embed>	Define um *container* para uma aplicação (não HTML) externa. (Nova *tag*) [101]
<object>	Define um objeto de conteúdo externo. [102]
<param>	Define um parâmetro para um objeto. [103]
<video>	Define um vídeo ou filme. (Nova *tag*) [104]
<audio>	Define conteúdo de som. (Nova *tag*) [105]
<source>	Define múltiplos recursos para vídeo e som. (Nova *tag*) [106]
<track>	Define o caminho para legendas de vídeo ou som. (Nova *tag*) [107]
<canvas>	Usado para desenhar gráficos em tempo real. (Nova *tag*) [108]
<map>	Permite mapear uma imagem do lado do cliente. Esse mapa possui áreas clicáveis. [109]
<area>	Define uma área para um mapa de imagem. [110]

Tabela 3.9: Incorporar Conteúdo.

Na tabela 3.10 estão representadas as *tags* que permitem estruturar tabelas em HTML. Todas as *tags* foram herdadas da versão atual do HTML.

Tag	Descrição
<table>	Define uma tabela. [111]
<caption>	Define o título de uma tabela. [112]
<colgroup>	Especifica um grupo de uma ou mais colunas na tabela. [113]
<col>	Define as propriedades de uma coluna num grupo. [114]
<tbody>	Agrupa o conteúdo do corpo numa tabela. [115]
<thead>	Agrupa o conteúdo do cabeçalho numa tabela. [116]
<tfoot>	Agrupa o conteúdo do fundo numa tabela. [117]

<tr>	Define uma linha na tabela. [118]
<td>	Define uma célula na tabela. [119]
<th>	Define o cabeçalho da célula na tabela. [120]

Tabela 3.10: Tabelas em HTML5.

Os formulários permitem aos utilizadores adicionarem e enviarem dados para o servidor. Foram acrescentados novos elementos à nova versão do HTML, de forma a melhorar a interação entre o utilizador e o formulário. Na tabela 3.11 estão representadas as *tags* definidas.

Tag	**Descrição**
<form>	Define um formulário em HTML para interação *input* com o utilizador. [121]
<fieldset>	Grupo de controladores relacionados num formulário. [122]
<legend>	Título ou explicação de um *fieldset*. [123]
<label>	Descrição de um controlador no formulário. [124]
<input>	Define um controlador de *input*. (*Tag* alterada) [125]
<button>	Define um botão no formulário. [126]
<select>	Define um controlador para selecionar numa lista de opções. [127]
<datalist>	Define uma lista de opções para controladores de *input*. (Nova *tag*) [128]
<optgroup>	Define um grupo de opções relacionadas. [129]
<option>	Define uma opção. [130]
<textarea>	Define um controlador de *input* de múltiplas linhas. [131]
<keygen>	Define um controlador que gera um par de chaves pública-privada e que envia a chave pública desse par. (Nova *tag*) [132]
<output>	Define o resultado de um cálculo. (Nova *tag*) [133]
<progress>	Define o progresso de uma tarefa. (Nova *tag*) [134]
<meter>	Define uma medida escalar dentro de um *range* conhecido. (Nova *tag*) [135]

Tabela 3.11: Formulários.

À nova versão do HTML foram adicionados elementos interativos, que se encontram representados na tabela seguinte (Tabela 3.12):

Tag	Descrição
<details>	Define detalhes adicionais que o utilizador poderá expandir ou esconder. (Nova *tag*) [136]
<summary>	Define a parte visível da *tag details* (Nova *tag*) [137]
<command>	Define um comando. (Nova *tag*) [138]
<menu>	Define uma lista de comandos. (Nova *tag*) [139]

Tabela 3.12: Elemento Principal.

3.1.2 Estrutura Básica, *DOCTYPE* e *CHARSETs*

A estrutura base do HTML5 continua a ser a mesma das versões anteriores, tendo sofrido apenas algumas melhorias. No exemplo 3.1 está representada a estrutura básica de uma página em HTML, com algumas das alterações efetuadas da versão 4 para a 5. Como é possível verificar, os excertos que se encontram a vermelho foram removidos, tendo sido substituídos pela parte a verde:

Exemplo 3.1: Estrutura básica de uma página.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD␣HTML␣4.01//EN" "http://www
   .w3.org/TR/html4/strict.dtd">
<HTML lang="en">
    <HEAD>
        <meta http-equiv="Content-Type" content="text/html;␣
           charset=utf-8" charset="utf-8">
        <TITLE>My first HTML document</TITLE>
    </HEAD>
    <BODY>
        <P>Hello world!</P>
        <SCRIPT type="text/javascript">
            //...
        </SCRIPT>
    </BODY>
</HTML>
```

Inicialmente, deverá ser definido o tipo de documento HTML (*DOCTYPE*), sendo esta certamente a alteração mais básica e útil para os programadores. Na versão 4 do HTML, o *DOCTYPE* refere-se a um *Document Type Definition* (DTD), que especifica as regras para a linguagem de marcação. Na nova versão do HTML, este DTD já não é apresentado, sendo então da responsabilidade do *browser* referenciá-lo.

Dentro do elemento *HEAD* é representada informação sobre a página e conteúdo nela publicado através de metadados (*tag meta*). A codificação da página é representada pelo atributo *charset*, permitindo ao *browser*, dessa forma, conseguir decifrar

o tipo de caracteres para a página em questão. A definição do *charset* também foi compactada em relação à versão 4 do HTML.

A *tag* de referenciação de JavaScript também foi reduzida, tendo sido removido o atributo *type*, tornando-a mais simples.

Estas foram as principais alterações globais que foram realizadas ao nível dos elementos mais gerais. Além de tornar o código menos verboso, também é possível, desta forma, reduzir o peso de uma página, o que se torna extremamente relevante na vertente *web*, principalmente para quem tem ligações muito limitadas.

3.1.3 Funcionalidades

Ao HTML5 encontram-se associadas um conjunto muito grande de funcionalidades. Estas serão explicadas nesta subsecção, bem como as suas características e compatibilidades nos *browsers* mais conhecidos.

3.1.3.1 Novos Elementos Estruturais

O processo de construir páginas *web* foi simplificado com a sintaxe introduzida. Para estruturar blocos ou zonas foi deixada de parte a *tag div*, sendo então substituída pelo nome próprio. A imagem 3.1 ilustra a designação das secções que a *tag* deve conter.

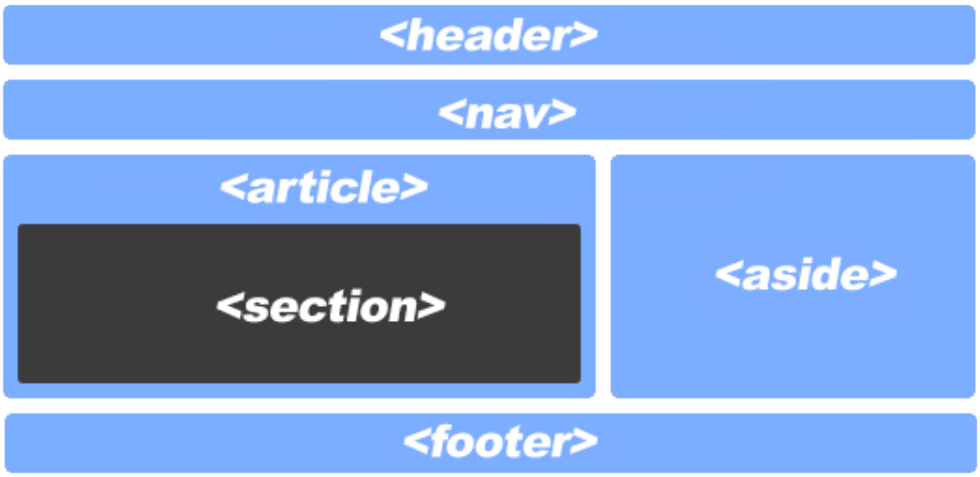


Figura 3.1: Possível estruturação de uma página HTML através dos novos elementos.

A estrutura apresentada é apenas um exemplo simples, claro e bem estruturado, não sendo então um modelo para toda e qualquer página *web*. De seguida serão de-

monstrados alguns exemplos que permitirão compreender o conjunto de elementos referenciados.

Na *tag header* normalmente constam o título da página, logótipo, bem como outros elementos referentes ao tipo de página em questão. O exemplo 3.2 demonstra um possível cabeçalho para uma página.

Exemplo 3.2: Cabeçalho de uma página.

```
<header>
    <h1>Titulo da pagina</h1>
    <a href="/"><img src=logo.png alt="Logo da Pagina X"></a>
</header>
```

O elemento *nav*, associado à navegação para páginas existentes na aplicação, poderá ser representado dentro de algum elemento estrutural (por exemplo, o *header*). Na imagem 3.1 está representado fora de qualquer outro elemento. O exemplo 3.3 demonstra uma possível barra de navegação para uma página.

Exemplo 3.3: Navegação de uma página.

```
<nav>
    <ul>
        <li><a href="/inicio">Inicio</a></li>
        <li><a href="/turismo">Turismo</a></li>
        <li><a href="/videos">Videos</a></li>
    </ul>
</nav>
```

O elemento *article* pode estar associado a um artigo de *blog* ou notícia, entre outras informações, e ainda poderá ser dividido em secções como *header*, *footer* e *section*. No *header* de um artigo poderá então ficar o título de uma notícia, na *section* poderá constar a notícia propriamente dita e no *footer* ligações externas, informação sobre o autor ou outras informações. O elemento *article* poderá estar integrado numa secção e representar uma lista de artigos de uma página.

Exemplo 3.4: Artigo numa página.

```
<article>
    <header>
        <h1>Barack Obama and UN call for immediate ceasefire<
           /h1>
        <h2>Bla bla bla bla</h2>
    </header>
    <section>
        <p>US President Barack Obama has called for an "
           immediate ceasefire" between Israel and Hamas as
           the death toll among Palestinians in the Gaza
           Strip reached 508...</p>
    </section>
    <footer>
```

```
        <span>By</span>: <a href="http://www.theguardian.com/
           world/2014/jul/21/gaza-crisis-obama-ceasefire-
           fighting-goes-on">The Guardian</a>
        <span>Keywords:</span>: <a href="#">Crisis</a>
    </footer>
</article>
```

O elemento *aside* permite incluir informação adicional ao conteúdo principal da página, tal como publicidade ou informação relacionada. No exemplo 3.5 está representada a *tag aside*, que se refere a informação sobre tecnologias *web*.

Exemplo 3.5: Parte lateral numa página.

```
<aside>
    <dl>
        <dt>CSS</dt>
        <dd>A set of standards for styling documents
           presented on the World Wide Web.</dd>
        <dt>PHP</dt>
        <dd>A server-side scripting language suited to
           dynamic HTML document generation for the web.</dd>
        <dt>JavaScript</dt>
        <dd>A client-side scripting language used for
           manipulating HTML document within a browser.</dd>
    </dl>
</aside>
```

Por fim, o elemento *footer* é importante para definir o final de uma determinada secção. No exemplo 3.4 encontra-se uma demonstração do *footer* dentro de um artigo. No caso de uma página ou aplicação *web*, normalmente são apresentadas informações sobre a secção, tais como o autor, direitos de autor, *links* para os termos de uso, contactos, etc.

Exemplo 3.6: Parte lateral numa página.

```
<footer>
  <p>Posted by: Nuno Morais</p>
  <p>Contact information: <a href="mailto:pg22806[at]alunos.
     uminho.pt">
  pg22806[at]alunos.uminho.pt</a></p>
</footer>
```

Os novos elementos semânticos, dos quais fazem parte estes que foram abordados nesta subsecção, já se encontram disponíveis a partir das versões representadas na tabela 3.13.

Browser	Versão
IE	9+

Firefox	4+
Chrome	6+
Safari	5+
Opera	11.1+
iOS Safari	4.1+
Opera Mini	7 (parcial)
Android Browser	2.2+
Opera Mobile	11.5+
Blackberry Browser	7+
Chrome for Android	35+
Firefox for Android	30+
IE Mobile	10+

Tabela 3.13: Versões dos *browsers* compatíveis com a nova semântica [140].

3.1.3.2 Formulários

Em HTML5, a implementação de formulários, comparativamente à versão anterior, tornou-se mais simples e uniformizada. Logo à partida surgem um conjunto de validações pré-definidas que, de certa forma, permitem ultrapassar algumas diferenças entre as implementações de cada *developer* e, também, poupar algum tempo na implementação em si, já que a maioria das validações é realizada apenas com a definição do tipo de *input* que se pretende.

A especificação HTML5 não define a forma como os *browsers* devem representar as mensagens de erro lançadas, nem tão pouco a interface para o utilizador.

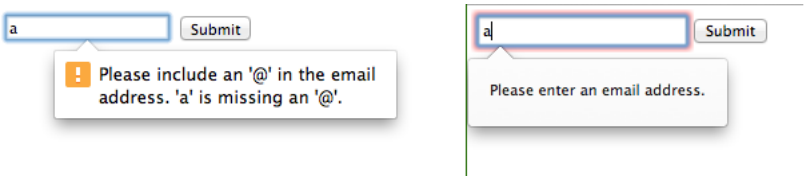


Figura 3.2: Comparação das mensagens de erro entre Google Chrome e Firefox.

Na imagem 3.2 encontra-se representada, do lado esquerdo, a mensagem de erro no Google Chrome, e à direita a do Firefox, demonstrando o que foi dito anteriormente. Nela consta uma *input box*, do tipo *email*, sendo que apenas pode ser validada se for inserido um endereço correto. No entanto, estes avisos podem ser alterados com o JavaScript, acedendo ao DOM, mas toda a simplicidade de fazer formulários será quebrada, pois todo o processo passará para o lado do *developer*.

Exemplo 3.7: Formulário de email.

```
<form>
        <input type="email" name="email" placeholder="Your␣
           email␣address"/>
        <input type="submit" value="Submit">
</form>
```

No exemplo 3.7 encontra-se um breve formulário de uma caixa para introduzir um email e um botão para enviar o valor. No entanto, existem outros tipos aceites em HTML5, nomeadamente *button*, *checkbox*, *color*, *date*, *datetime*, *datetime-local*, *file*, *hidden*, *image*, *month*, *number*, *password*, *radio*, *range*, *reset*, *search*, *tel*, *text*, *time*, *url* e *week*, sendo que cada um dos tipos possui um propósito.

Além disso, o atributo *placeholder* é uma mais valia nos formulários, pois permite adicionar informação no controlo para o utilizador, o que só era possível fazer com a ajuda do JavaScript nas versões anteriores.

É igualmente possível especificar padrões através do atributo *pattern*, com recurso a expressões regulares nos seguintes tipos de *input*: *text*, *search*, *url*, *tel*, *email* e *password*.

Estas especificações ainda não se encontram completamente disponíveis em nenhum *browser*, porém, tanto o *Google Chrome* como o *Opera* estão mais avançados, faltando-lhes apenas o tipo *datetime*. No que concerne à validação, a grande maioria dos *browsers* já oferece suporte, conforme se pode observar na tabela 3.14.

Browser	**Versão**
IE	10+
Firefox	4+
Chrome	10+
Safari	5+ (parcial)
Opera	10.1+
iOS Safari	Sem Suporte
Opera Mini	Sem Suporte
Android Browser	4.4.3+
Opera Mobile	10+
Blackberry Browser	10+
Chrome for Android	35+
Firefox for Android	30+
IE Mobile	10+ (parcial)

Tabela 3.14: Versões dos *browsers* compatíveis com validação de formulários [141].

3.1.3.3 *Canvas*

O *canvas* é o componente que permite desenhar elementos gráficos numa área da página, sendo definido pela *tag canvas*. Os elementos presentes podem ser utilizados em imagens, jogos, interação com o utilizador, entre outros, sob a forma de linhas, rectângulos, elipses, texto ou imagens. Porém, poucos são os *browsers* que já suportam todo esse conjunto de elementos, sendo que a maioria ainda só suporta os mais primitivos.

Exemplo 3.8: Desenho em canvas.

```
<!DOCTYPE HTML>

<html lang="pt">
  <head>
    <meta charset="utf-8"/>
    <title>Canvas TEST</title>
    <style type="text/css">canvas { border: 1px solid #000; }
       </style>
  </head>
  <body>
    <canvas id="myCanvas">Your browser does not support the
       HTML5 canvas tag.</canvas>
    <script>
      var c=document.getElementById('myCanvas'),
          ctx=c.getContext('2d');
          ctx.fillStyle='#FF0000';
          ctx.fillRect(0,0,80,80);
    </script>
 </body>
</html>
```

No exemplo 3.8 encontra-se representado um retângulo no *canvas*. O elemento *canvas* é declarado no HTML e depois é manipulado a partir do JavaScript, sendo necessário ir buscá-lo a partir do seu identificador, neste caso "myCanvas", e obter o seu *context* em 2D. Por fim, altera-se a cor de fundo (*fillStyle*) e desenha-se um objeto (*fillRect*). Neste exemplo, o resultado final é o apresentado na imagem 3.3.

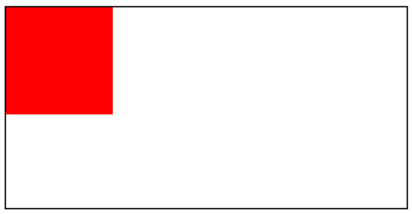


Figura 3.3: Representação de um rectângulo no canvas em HTML5.

Com esta API é possível definir qualquer efeito que o programador deseje, desde cores ou gradientes, até animações, bem como adicionar imagens e definir texto, adici-

onar linhas com curvas perfeitas, recorrendo ao método *arc*, ou construir círculos com curvas mais elaboradas, com base nas curvas de *Bézier*.

Para a elaboração de gráficos 3D, recomenda-se a utilização de bibliotecas JavaScript, que permitem agilizar a criação de animações, camadas, eventos, entre outras funcionalidades, sendo exemplo a biblioteca *WebGL*, que se baseia em *OpenGL ES 2.0*.

Browser	**Versão**
IE	9+
Firefox	2+
Chrome	4+
Safari	3.1+ (parcial)
Opera	9.6+
iOS Safari	3.2+
Opera Mini	7 (parcial)
Android Browser	3+
Opera Mobile	10+
Blackberry Browser	7+
Chrome for Android	35+
Firefox for Android	30+
IE Mobile	10+

Tabela 3.15: Versões dos *browsers* compatíveis com o elemento canvas [142].

O *canvas* já pode ser explorado nas versões representadas na tabela 3.15, no entanto, os *browsers* ainda se podem encontrar bastante limitados no que toca à API.

3.1.3.4 Vídeo e Áudio

Até esta versão não era possível reproduzir vídeo nem áudio sem o auxílio de outros *plugins*, como o *Flash*, mas agora esse problema já possui uma solução. No entanto, os *browsers* podem não suportar os mesmos *codecs*, como pode ser observado na tabela 3.16, o que os obriga a ter várias fontes para o mesmo conteúdo, sendo este um dos grandes desafios do HTML5.

	Vídeo	**Áudio**
IE	MP4/H.264	MP3
Firefox	MP4/H.264, WebM, Ogg/Theora	MP3, Wav, Ogg
Chrome	MP4/H.264, WebM, Ogg/Theora	MP3, Wav, Ogg
Safari	MP4/H.264	MP3, Wav
Opera	WebM, Ogg/Theora	Wav, Ogg
iOS Safari	MP4/H.264	MP3
Opera Mini		Ogg
Android Browser	MP4/H.264, WebM*	MP3, Ogg
Opera Mobile	MP4/H.264, WebM*	MP3, Ogg
Blackberry Browser	MP4/H.264	MP3, Ogg
Chrome for Android	MP4/H.264, WebM	MP3, Ogg
Firefox for Android	MP4/H.264*, WebM, Ogg/Theora	MP3, Ogg
IE Mobile	MP4/H.264	MP3

* Suporte parcial

Tabela 3.16: Formatos de vídeo e áudio descodificados por cada *browser* [143], [144], [145], [146].

Do lado do *developer*, o desafio é conseguir que vídeo e áudio sejam reproduzidos, independentemente do *browser* e da sua versão. Para tal, é necessário ter a noção do tipo de formato que é descodificado por cada *browser* e apresentar as fontes necessárias para a descodificação.

Exemplo 3.9: Representação de um vídeo em HTML5.

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogg" type="video/ogg">
Your browser does not support the video tag.
</video>
```

No exemplo 3.9 está representado um elemento vídeo com duas fontes para o mesmo vídeo, onde o *browser* descodifica o primeiro formato da lista que suportar. Caso não detete qualquer um dos formatos, a mensagem "*Your browser does not support the video tag*"surgirá ao utilizador.

Além das *tags audio* e *video*, existem outros atributos que permitem adicionar comportamentos ao *player*, como os *controls* (*start, pause, mute, sound, expand, time, ...*), o *preload* (permite ao *browser* descarregar o vídeo logo após o carregamento total da página) e o *autoplay* (permite que o vídeo seja logo reproduzido após o seu carregamento, sem o consentimento do utilizador) [104].

Os *browsers* para dispositivos móveis também interpretam estas *tags*, no entanto o *developer* deve ter em conta a resolução dos vídeos, de forma a adequar-se a este tipo de equipamentos. Para isso, é utilizado o atributo *media*, que funciona como uma *media query* em CSS3, podendo assim definir o tamanho mínimo ou máximo de píxeis que o dispositivo necessita de ter para poder reproduzir o mesmo [147].

3.1.3.5 *Drag & Drop*

O elemento *drag-and-drop* é a funcionalidade que permite clicar num objeto e arrastá-lo para outra zona, tendo esta, até ao momento, que ser requisitada por algumas implementações definidas em JavaScript. A versão 5 do HTML já a possui incorporada, simplificando a tarefa aos *developers*, tendo estes que usar um conjunto de instruções, também em JavaScript.

No exemplo 3.10 são apresentadas duas *divs* que permitem fazer *drop* de objetos. Neste caso, temos dois elementos muito importantes, o *ondrop* e o *ondragover*, sendo que o primeiro é chamado quando o objeto é largado e o segundo especifica onde o objeto pode ser largado. A imagem poderá então saltar para cada umas das duas caixas através dos eventos *drop*, *drag* e *allowDrop*, que se encontram devidamente configurados pela API disponibilizada, sendo que o *dataTransfer* possui dois métodos importantes, o *getData* e o *setData*, que permitem obter e alterar os dados do objeto a ser apresentado.

Exemplo 3.10: Representação de um exemplo *drag-and-drop* em HTML5.

```
<!DOCTYPE HTML>
<html lang="en">
  <head>
    <style>
      #drag1, #div1, #div2 { position:relative; float: left;
         border: 1px solid #000; width: 100px; height: 100px;
          }
    </style>
    <script>
      function allowDrop(ev) {
        ev.preventDefault();
      }

      function drag(ev) {
        ev.dataTransfer.setData("KeyImage", ev.target.id);
      }

      function drop(ev) {
        ev.preventDefault();
        var data = ev.dataTransfer.getData("KeyImage");
        ev.target.appendChild(document.getElementById(data));
      }
</script>
```

```
  </head>
  <body>
    <div id="div1" ondrop="drop(event)" ondragover="allowDrop
       (event)">
      <img id="drag1" src="html5_logo.png" draggable="true"
        ondragstart="drag(event)" width="100" height="100">
    </div>
    <div id="div2" ondrop="drop(event)" ondragover="allowDrop
       (event)"></div>
  </body>
</html>
```

Com base no exemplo 3.10 é possível obter o resultado mostrado na figura 3.4. A imagem poderá então saltar entre o quadrado da esquerda e da direita, bastando arrastar e largar a imagem entre eles.

Figura 3.4: Representação de *drag-and-drop* em HMTL5.

Esta funcionalidade encontra-se disponível nas versões dos *browsers* descritas na tabela 3.17.

Browser	**Versão**
IE	8+
Firefox	3.5+
Chrome	4+
Safari	3.1+ (parcial)
Opera	12+
iOS Safari	Sem suporte
Opera Mini	Sem suporte
Android Browser	Sem suporte
Opera Mobile	12.1
Blackberry Browser	Sem suporte
Chrome for Android	Sem suporte
Firefox for Android	Sem suporte
IE Mobile	10

Tabela 3.17: Versões dos *browsers* compatíveis com o elemento drag-and-drop [148].

3.1.3.6 *Web Storage*

O *web storage* permite, tal com o nome indica, armazenar dados localmente no *browser*. Este veio tirar lugar aos antigos *cookies*, que permitiam guardar informação da aplicação, sendo mais seguro, rápido e com um limite mais alargado, não estando sujeito a perda de performance da aplicação.

A informação é guardada num dicionário, em pares chave-valor, e cada página só pode guardar e aceder dados que lhe pertençam.

Embora possua inúmeras vantagens, esta estratégia não deverá ser utilizada para armazenar a totalidade dos dados a serem disponibilizados, removendo todos os pedidos ao servidor, e para dados críticos.

O *web storage* divide-se em duas categorias, *localstorage* e *sessionstorage*, sendo que a primeira permite armazenar dados que podem ser acedidos em sessões diferentes, enquanto que a segunda apenas guarda a informação até a *tab* ser fechada.

Exemplo 3.11: Representação de um exemplo drag-and-drop em HTML5.

```
<!DOCTYPE HTML>
<html lang="en">
  <head></head>
  <body>
    <p id="message">Welcome back, <span id="result"></span></
       p>
    <script>
      if (typeof(Storage) != "undefined") { // Check browser
         support
        localStorage.setItem("name", "Nuno"); // Store: Key,
           Value
        document.getElementById("result").innerHTML =
           localStorage.getItem("name"); // Retrieve by Key
      } else {
        document.getElementById("message").innerHTML = "Sorry
           ,␣your␣browser␣does␣not␣support␣Web␣Storage...";
      }
    </script>
  </body>
</html>
```

No exemplo 3.11 é possível verificar a simplicidade da API desta funcionalidade. Basicamente, esta apenas possui dois métodos, o *getItem* e o *setItem*, que permitem ir buscar e alterar o valor, respetivamente. Para isso, é necessário saber se o *web storage*

se encontra definido no *browser* para o poder utilizar, o qual se pode verificar na tabela 3.18.

Browser	**Versão**
IE	8+
Firefox	2+
Chrome	4+
Safari	4+
Opera	10.5+
iOS Safari	3.2+
Opera Mini	Sem suporte
Android Browser	2.1+
Opera Mobile	11.5+
Blackberry Browser	7
Chrome for Android	36
Firefox for Android	31
IE Mobile	10

Tabela 3.18: Versões dos *browsers* compatíveis com o elemento *web storage* [149].

3.1.3.7 *Application Cache*

A funcionalidade de *cache* permite ao utilizador continuar a usar uma aplicação, numa versão mais limitada, no caso de não poder manter uma ligação à internet. Para além de se conseguir armazenar informação para modo *offline*, também se garante uma maior rapidez no carregamento de recursos, visto que estes já podem estar lá armazenados, reduzindo a carga do servidor, pois os *browsers* apenas irão descarregar os recursos caso os mesmos sejam atualizados do servidor.

Em relação à informação que será apresentada ao utilizador, assim como quaisquer alterações feitas pelo mesmo durante o período *offline*, é da responsabilidade do *developer* implementar soluções para preservar os dados e os sincronizar no ou nos próximos períodos *online*.

Quanto aos ficheiros que irão ficar disponíveis, quando não existe ligação à internet, têm de ser especificados num ficheiro *manifest* que se deve encontrar do lado do servidor, contendo os nomes dos recursos que o *browser* deverá guardar localmente, e está referenciado com o atributo *manifest* na *tag* HTML, como mostra o exemplo 3.12.

Exemplo 3.12: Representação da informação relacionada ao caminho do ficheiro manifest.

```
<!DOCTYPE HTML>
<html manifest="http://ex_domain.com/manifest.appcache" lang=
   "en">
```

```
//...
</html>
```

O ficheiro *manifest* está dividido em três partes: *Cache Manifest*, *Network* e *Fallback*. A primeira, *Cache Manifest*, deve estar obrigatoriamente no ficheiro, e os nomes que o seguem consistem nos ficheiros que ficarão disponíveis após serem descarregados a primeira vez e a ligação à internet ser perdida. Depois, caso seja relevante, deverá vir o *Network*, que contém os nomes dos ficheiros que nunca ficarão em *cache*, pelo que é sempre necessário fazer um pedido ao servidor. Por fim, virá o *Fallback*, que representa a página para onde o utilizador é redirecionado caso um recurso não se encontre disponível, contendo então dois parâmetros: o primeiro permite informar o caminho que não está disponível, e o segundo para onde é redirecionado.

Exemplo 3.13: Representação da informação relacionada com o ficheiro manifest.

```
CACHE MANIFEST
# 2014-08-01 v1.0.0
/theme.css
/logo.gif
/main.js

NETWORK:
login.php

FALLBACK:
/ /offline.html
```

O exemplo 3.13 mostra a estruturação e especificação dos ficheiros que devem estar em *cache*, conforme descrito anteriormente. Depois do *Cache Manifest*, encontra-se representada uma linha iniciada com #, seguida da data e uma versão, sendo esta uma breve abordagem para posteriormente o *browser* saber se o *manifest* foi ou não atualizado.

A tabela 3.19 mostra as versões dos *browsers* que suportam esta funcionalidade.

Browser	**Versão**
IE	10+
Firefox	3.5+
Chrome	4+
Safari	4+
Opera	10.6+
iOS Safari	3.2+
Opera Mini	Sem suporte
Android Browser	2.1+
Opera Mobile	11.5+
Blackberry Browser	7
Chrome for Android	36

Firefox for Android	31
IE Mobile	10

Tabela 3.19: Versões dos *browsers* compatíveis com o elemento *Application Cache* [150].

3.1.3.8 *Geolocation*

Esta funcionalidade permite obter a posição atual do utilizador com relativa precisão. Pelo facto da geolocalização poder comprometer a privacidade do utilizador, é-lhe então enviado um pedido para que este aceite ou não a partilha da sua localização. Isto poderá ser usado para selecionar a informação mais relevante a apresentar ao utilizador, como eventos ou restaurantes perto da sua localização.

Existem vários tipos de deteção da localização, sendo os seguintes os mais utilizados:

GPS Este é o método mais preciso, pois utiliza o sistema GPS para obter a localização, no entanto este só está disponível em sistemas móveis, por isso poderá ser menos utilizado.

A-GPS (*Assisted GPS*) Este método utiliza a triangulação entre as torres da rede móvel para determinar a localização, não sendo tão preciso como o GPS.

Wi-Fi Este método recorre ao Wi-fi para obter a localização do utilizador, sendo bastante preciso.

Endereço de IP Este método necessita de uma base de dados externa, de forma a conseguir reconhecer com precisão a cidade onde está ligado.

Esta funcionalidade é muito importante para os dispositivos móveis, pois estes normalmente estão em constante movimento, e o utilizador poderá obter informação mais relevante em "tempo real". As redes sociais podem igualmente tirar partido deste tipo de informação de forma a conseguirem obter a localização de fotografias ou de *posts*, e com isso criar um historial mais enriquecido do perfil do utilizador.

O exemplo 3.14 demonstra como obter as informações das coordenadas da posição atual do utilizador. A partir das mesmas poderá ainda utilizar-se outros recursos, como os mapas do *Google*, adicionar alguma interatividade ou tratar a informação recebida como for entendido.

Exemplo 3.14: Representação um exemplo de *geolocation* em HTML5.

```
<!DOCTYPE HTML>
<html lang="en">
```

```
  //...
  <body>
    <p id="message">Welcome back Nuno, click on the button "
       try␣it" to know your coordinates. <span id="demo"></
       span></p>
    <button onclick="getLocation()">Try It</button>

    <script>
      var x = document.getElementById("demo");
      function showPosition(position) {
        x.innerHTML="<br␣/>␣-␣Latitude:␣" + position.coords.
           latitude + "<br␣/>␣-␣Longitude:␣" + position.
           coords.longitude;
      }
      function getLocation() {
        if (navigator.geolocation) {
          navigator.geolocation.getCurrentPosition(
             showPosition);
        } else {
          x.innerHTML = "Geolocation␣is␣not␣supported␣by␣this
             ␣browser.";
        }
      }
    </script>
  </body>
</html>
```

O resultado está presente na figura 3.5, onde após se clicar em "*try it*", o utilizador será questionado se pretende facultar a sua localização, e se sim, irá obter como resultado final a versão presente na figura 3.6.

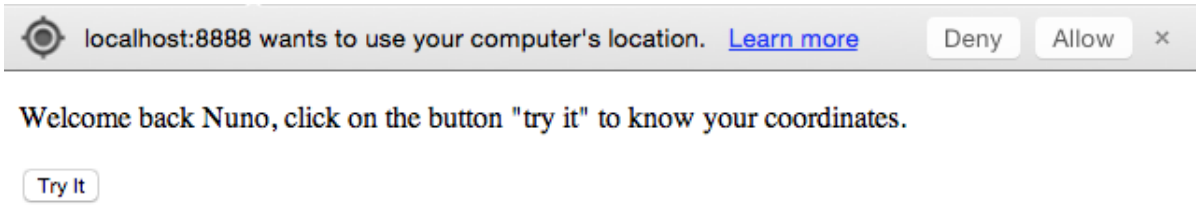


Figura 3.5: Autorização de *geolocation* apresentada ao utilizador.

Esta API pode ser usada nas versões apresentadas na tabela 3.20.

Browser	**Versão**
IE	9+
Firefox	3.5+
Chrome	5+
Safari	5+

Opera	10.6+
iOS Safari	3.2+
Opera Mini	Sem suporte
Android Browser	2.1+
Opera Mobile	11.5+
Blackberry Browser	7
Chrome for Android	36
Firefox for Android	31
IE Mobile	10

Tabela 3.20: Versões dos *browsers* compatíveis com o elemento *geolocation* [151]

3.1.3.9 *Web Sockets*

Até ao momento, toda a comunicação entre cliente e servidor era feita através de um pedido do lado do cliente e consequente resposta do servidor, sendo que a ligação era encerrada logo após o envio de toda a informação necessária.

Com a introdução do AJAX, houve então uma maior dinâmica para fazer pedidos do cliente para o servidor, mas nunca o contrário, isto é, mesmo que o servidor necessitasse de enviar mais informação auxiliar para o cliente, isto não era possível. Assim, era imperativo criar uma ligação persistente entre o servidor e o cliente, de forma a que estes pudessem comunicar bidirecionalmente. Numa fase inicial, tentou recorrer-se às tecnologias ou estratégias denominadas por *push* ou *comet*, que de certa forma prolongavam ou virtualizavam a ligação *http* para que esta não fosse desligada, mas ambas partilhavam alguns problemas, daí a necessidade de trazer os *sockets* para a vertente *web*.

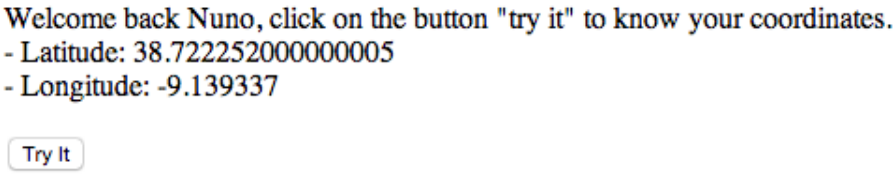


Figura 3.6: Apresentação das coordenadas após o utilizador aceitar partilhar a sua localização.

Os padrões da API *web socket* estão a ser definidos pela W3C, enquanto a criação dos protocolos está a ser auxiliada pela Internet Engineering Task Force (IETF). É possível estabelecer a ligação recorrendo ou não à ligação segura, necessitando apenas de mudar o prefixo de *ws* para *wss*, e poderá ainda utilizar vários subprotocolos especificados por argumento.

Exemplo 3.15: Representação um exemplo de *web socket* em HTML5.

```
<script>
function WebSocketTest()
{
  if ("WebSocket" in window)
  {
     console.log("WebSocket is supported by your Browser!");
     // Let us open a web socket
     var ws = new WebSocket("ws://localhost:9998/echo");
     ws.onopen = function() {
        // Web Socket is connected, send data using send()
        ws.send("Message to send");
        console.log("Message is sent...");
     };
     ws.onmessage = function (evt) {
        var received_msg = evt.data;
        console.log("Message is received...");
     };
     ws.onclose = function() {
        // websocket is closed.
        console.log("Connection is closed...");
     };
  }
  else
  {
     // The browser doesn't support WebSocket
     console.log("WebSocket NOT supported by your Browser!");
  }
}
</script>
```

O exemplo 3.15 apresenta uma implementação muito básica de *web sockets* em HTML5, que permite ligar a um servidor, sendo, neste caso, o *localhost* na porta 9998. Automaticamente este envia uma mensagem, ficando à escuta e à espera que o *socket* seja fechado. Todas as mensagens são imprimidas na consola de desenvolvimento do *browser*.

Os *web sockets* estão disponíveis a partir das versões dos *browsers* apresentados na tabela 3.21.

Browser	**Versão**
IE	10+
Firefox	6+
Chrome	14+
Safari	6+
Opera	12.1+
iOS Safari	6.1+
Opera Mini	Sem suporte
Android Browser	4.4+
Opera Mobile	12.1+
Blackberry Browser	7
Chrome for Android	36
Firefox for Android	31
IE Mobile	10

Tabela 3.21: Versões dos *browsers* compatíveis com o elemento *web sockets* [152].

3.1.3.10 *Web Workers*

Um *web worker* permite correr um *script* ou bloco de código em segundo plano. Esta funcionalidade é muito importante, pois o JavaScript é executado em *single-threaded* e, desta forma, caso haja um bloco de código contendo funções mais complexas ou a serem executadas durante bastante tempo, não ocorrerá interrupção da interação com o utilizador.

Existem dois tipos de *web workers*: *dedicated workers* e *shared workers*. Os *dedicated workers* permitem manter-se ligados apenas ao seu "pai", ou seja, o *script* que criou o seu *worker*, enquanto que os *shared workers* podem interagir com outros *scripts*, desde que estes se encontram no mesmo domínio.

Visto que os *web workers* são executados numa sequência isolada, e que inclusive o código a ser executado está num ficheiro à parte, é necessário criar um objeto para manipular o *web worker* e também receber o resultado do mesmo. O *web worker* deve igualmente ser iniciado, sendo ainda possível adicionar eventos e terminá-lo.

Exemplo 3.16: Representação de um exemplo para instanciar um *web worker* em HTML5.

```
<script>
  var worker = new Worker('sendMessage.js');

  worker.addEventListener('message', function(e) {
    console.log('Worker said: ', e.data);
  }, false);

  worker.postMessage('Hello, World'); // Send data to our
     worker and start
</script>
```

No exemplo 3.16 é possível verificar que é instanciado um ficheiro com o nome de *sendMessage.js*, e que se não existir, é dado um erro silencioso e consequentemente não arranca. Posteriormente, é adicionado um evento que fica à espera de informação e que, por fim, arranca através do método *postMessage*, enviando o parâmetro "Hello, World". Quando este arranca, o *worker*, como mostra no exemplo 3.17, espera um parâmetro e, posteriormente, com o mesmo método de arranque, *postMessage*, responde para o seu "pai"e termina.

Exemplo 3.17: Representação de um *web worker* em HTML5.

```
<script>
  self.addEventListener('message', function(e) {
    self.postMessage(e.data);
  }, false);
</script>
```

Os *web workers* estão disponíveis a partir das versões dos *browsers* especificadas na tabela 3.22.

Browser	**Versão**
IE	10+
Firefox	3.5+
Chrome	4+
Safari	4+
Opera	10.6+
iOS Safari	5.1+
Opera Mini	Sem suporte
Android Browser	4.4+
Opera Mobile	11.5+
Blackberry Browser	7
Chrome for Android	36
Firefox for Android	31
IE Mobile	10

Tabela 3.22: Versões dos *browsers* compatíveis com o elemento *web worker* [153].

3.1.3.11 *Server-Sent Events*

Um *server-sent event* permite à página receber atualizações automáticas vindas de um servidor. Com este tipo de tecnologia, o cliente já não necessita de enviar pedidos constantes ao servidor a perguntar se há atualizações. Em redes sociais, como o Facebook ou o Twitter, o número de pedidos ao servidor reduziu circunstancialmente apenas com a implementação deste mecanismo.

A grande diferença entre SSE e *WebSockets* é que o primeiro não necessita de um protocolo especial ou implementação do lado do servidor. Estes são muito úteis quando o cliente necessita de receber apenas atualizações.

Exemplo 3.18: Representação de um exemplo de um *server-sent event* em HTML5.

```
<script>
if(typeof(EventSource) !== "undefined") {
    var source = new EventSource("demo_test.php");
    source.onmessage = function(event) {
        document.getElementById("result").innerHTML += event.
           data + "<br>";
    };
} else {
    document.getElementById("result").innerHTML = "Sorry,␣
       your␣browser␣does␣not␣support␣server-sent␣events...";
}
</script>
```

O exemplo 3.18 demonstra que o *server-sent event* apenas vai adicionando ao elemento *result* a informação que for atualizada no ficheiro *demo_ test.php*. Esta funcionalidade encontra-se presente nas versões dos *browsers* representados na tabela 3.23.

Browser	**Versão**
IE	Sem suporte
Firefox	6+
Chrome	6+
Safari	5+
Opera	11+
iOS Safari	4.1+
Opera Mini	Sem suporte
Android Browser	4.4+
Opera Mobile	11.5+
Blackberry Browser	7
Chrome for Android	36
Firefox for Android	31

IE Mobile	Sem suporte

Tabela 3.23: Versões dos *browsers* compatíveis com o elemento *server-sent events* [154].

3.1.4 Bibliotecas e/ou *Frameworks* JavaScript

Nos tempos que correm, a tecnologia JavaScript tornou-se essencial para qualquer aplicação *web*, e a cada dia que corre o seu uso tem vindo a crescer, em grande parte devido à quantidade de bibliotecas que vão surgindo. Embora sejam inúmeras, é necessário perceber que cada uma faz algo muito específico. Por exemplo, o jQuery praticamente só permite manipular o DOM, enquanto a UnderscoreJS apenas possibilita ter algumas funções auxiliares para tratamento de objetos. Assim, têm vindo a surgir algumas *frameworks*, com o objetivo principal de aumentar o leque de funcionalidades do lado do *developer* sem este ter que se preocupar com algum tipo de compatibilidade entre as bibliotecas.

Para este trabalho foi necessário realizar um breve estudo de algumas *frameworks* e bibliotecas, de forma a facilitar o desenvolvimento da migração posteriormente. Antes de serem escolhidas as bibliotecas e *frameworks* a serem usadas, foi feito um levantamento de algumas delas, de forma a reconhecer algumas vantagens e desvantagens das mesmas. Para isso, foi feita uma filtragem inicial com base no padrão escolhido para migrar a nova aplicação, o MVVM, tendo ainda em atenção a necessidade de ter uma boa comunidade, a certeza da continuação da evolução e a capacidade de resposta a alguns requisitos do caso de estudo.

jQuery Esta talvez seja a biblioteca mais utilizada em JavaScript, normalmente é uma das dependência de outras bibliotecas ou *frameworks*. Permite fazer a manipulação do DOM, eventos e animações.

FabricJS Uma biblioteca em constante desenvolvimento para manipulação do canvas. Tem ainda a capacidade de emular resultados semelhantes em *browsers* antigos.

KineticJS Esta biblioteca permite auxiliar na manipulação do canvas.

KnockoutJS Trata-se de uma biblioteca JavaScript que utiliza o padrão MVVM. Permite fazer uma separação clara do domínio, componentes e dados a mostrar, conseguindo manter a UI atualizada de forma dinâmica.

AngularJS Permite fazer a separação entre MV* (*Model-view-Whatever*) da aplicação.

KendoUI Contém um conjunto de *widgets* e funcionalidades em HTML5.

ExtJS É uma *framework* com arquitetura MVC e contém um conjunto de controladores.

RequireJS Permite carregar todo o conjunto de bibliotecas e módulos em JavaScript.

UnderscoreJS Oferece um amplo conjunto de funções úteis.

De entre as bibliotecas anteriormente descritas, apenas serão analisadas a FabricJS, KineticJS, KnockoutJS, AngularJS, KendoUI e ExtJS, de forma a falar sobre as suas principais vantagens e desvantagens, comparando-as entre si. As restantes bibliotecas serão descartadas, uma vez que são apenas auxiliares para manipulação do DOM e para métodos úteis no desenvolvimento de qualquer aplicação.

3.1.4.1 AngularJS

A *framework* AngularJS é desenvolvida pela Google, o que transmite confiança na evolução e continuação da mesma. Esta é relativamente recente, encontrando-se num momento de transição e numa versão *Release Candidate* no momento em que foi testada. De seguida vai ser abordado como é feita a implementação de ***data binding***, **validação** e ***templating*** pelo *developer*.

Data Binding O AngularJS tem duas formas de fazer *data binding*: através do atributo *ng-model* (exemplo 3.19) ou através da introdução de dupla chaveta dentro da *tag* especificada (exemplo 3.20).

Exemplo 3.19: Representação de *data binding* através de um atributo em AngularJS.

```
<input type="text" ng-model="name" />
```

Exemplo 3.20: Representação de *data binding* através de dupla chaveta em AngularJS.

```
<p>My name is {{name}}</p>
```

Como é possível verificar no exemplo 3.20, a representação por dupla chaveta é uma mais valia pois permite ao *developer* escrever menos e manter um código mais limpo, o que não é possível com a representação através do atributo, como demonstra o exemplo 3.19. A atualização do *data binding* é feita quando é premida uma tecla, isto é, a informação fica disponível mal haja alguma alteração. Cada vez que há algum erro na declaração do *data binding*, o mesmo é omitido, sendo que se tornará complicado fazer *debug* nessas situações.

As validações em AngularJS recorrem aos atributos especificados no HTML5, no entanto é necessário adicionar o atributo *novalidate* para desativar as validações predefinidas pelo *browser*. É ainda possível adicionar ou especificar facilmente padrões mais complexos e mostrar as respetivas mensagens de erro em tempo real.

Exemplo 3.21: Representação de validação de um formulário em AngularJS.

```
<head>
```

```
    // ..
    <style type="text/css">
      .css-form input.ng-invalid.ng-dirty { background-color:
          #FA787E; }
      .css-form input.ng-valid.ng-dirty { background-color:
         #78FA89; }
    </style>
  </head>
  <body ng-app="formExample">
    <div ng-controller="ExampleController">
    <form name="form" class="css-form" novalidate>
      E-mail:
        <input type="email" ng-model="user.email" name="
           uEmail" required/><br />
      <div ng-show="form.uEmail.$dirty␣&&␣form.uEmail.
         $invalid">Invalid:
        <span ng-show="form.uEmail.$error.required">Tell us
           your email.</span>
        <span ng-show="form.uEmail.$error.email">This is not
           a valid email.</span>
      </div>
    </form>
  </div>
  <script>
      angular.module('formExample', [])
    .controller('ExampleController', ['$scope', function(
       $scope) {
      //...
    }]);
  </script>
```

O exemplo 3.21 mostra a validação de um endereço de email, de forma praticamente automática, em que uma mensagem de erro surgirá e o campo onde é escrito o endereço permanecerá a vermelho enquanto este não se encontrar dentro das regras definidas.

O sistema de *templating* em AngularJS permite estilizar blocos de código, que podem ser usados noutros contextos ou até substituir zonas. O exemplo 3.22 mostra uma simples maneira de fazer *templating*.

Exemplo 3.22: Representação de *templating* em AngularJS.

```
  <body ng-controller="NameList">
  <ul>
    <li ng-repeat="name␣in␣names">
      {{name.name}}
    </li>
  </ul>
<script>
  var namesApp = angular.module('namesApp', []);
```

```
  namesApp.controller('NameList', function ($scope) {
    $scope.names = [
      {'name': 'Nuno'},
      {'name': 'Filipa'},
      {'name': 'Carlos'}
    ];
  });
</script>
</body>
```

AngularJS permite criar *templates* de forma muito simplista, como mostra o exemplo 3.22, mas quando se trata de algo mais complexo é necessário adicionar muito mais código, tornando-o mais verboso, principalmente quando é necessário adicionar algum tipo de comportamento. É importante referenciar que é possível separar as *templates* do AngularJS em ficheiros, e importar para a zona onde são necessárias.

3.1.4.2 KnockoutJS

A biblioteca KnockoutJS é uma das bibliotecas mais conceituadas para *data binding* em padrões MVVM. Começou a ser desenvolvida por Steve Sanderson num projeto *open source*, e no seu portal de apresentação é possível treinar e seguir tutoriais de forma interativa, tornando a aprendizagem muito mais facilitada.

Neste caso, o *data binding* apenas pode ser feito de uma maneira, através do atributo *data-bind*, sendo que dentro do mesmo podem ser adicionados vários tipos de *binding*, por exemplo para controlo de texto, aparência ou fluxo, manipulação de formulários, etc.

Exemplo 3.23: Representação de *data binding* em KnockoutJS.

```
<input type="text" data-bind="value: name" />
```

O exemplo 3.23 demonstra a forma como o *data binding* é feito no KnockoutJS. Como se pode verificar, é bastante intuitivo, no entanto, em casos mais complexos, poderá tornar-se excessivamente verboso e complicado de ler.

Todo e qualquer erro que ocorra durante a implementação do *data binding* despoleta uma mensagem de erro, o que se torna uma mais valia para detetar qualquer equívoco do *developer*. Todas as atualizações são feitas depois de todas a alterações, no entanto isto pode ser personalizado através do tipo *data binding* "valueUpdate", podendo conter mais três estados possíveis: *keyup*, *keypress* e *afterkeydown*.

A validação em KnockoutJS só é possível após a importação de uma biblioteca auxiliar, Knockout-Validation, que está disponível no GitHub. Esta biblioteca é muito versátil, sendo possível personalizar a maneira como se pretende fazer a validação. As-

sim, esta pode ser feita a partir dos atributos ou especificando no *viewModel*, definindo mensagens de erro genéricas e escrevendo regras customizadas.

Exemplo 3.24: Representação de um exemplo básico de validação em KnockoutJS.

```
<script id="customMessageTemplate" type="text/html">
  <em class="customMessage" data-bind='validationMessage:
      field'></em>
</script>
<div data-bind='validationOptions: { messageTemplate: "
   customMessageTemplate" }'>
    <label>Email: <input data-bind='value: emailAddress,
       valueUpdate: "input"' required type="email"/></
       label>
</div>
<script>

  ko.validation.configure({
    parseInputAttributes: true
  });

  var viewModel = {
      emailAddress: ko.observable().extend({
          required: { message: 'Tell us your email' }, //
              custom message (Empty field)
          pattern: {
            message: "This is not a valid email." //
               Invalid Email
          }
      })
  };

  ko.applyBindings(viewModel);

</script>
```

No exemplo 3.24 é visível a obtenção de um bom *feedback* a partir da especificação de uma configuração muito simples em KnockoutJS. De salientar que foi mudado o *valueUpdate* para atualizar quando o estado do formulário alterar, enviando a mensagem de erro (se for o caso) para o utilizador.

O KnockoutJS possui uma excelente experiência com *templating*, sendo possível especificar nomes para as *templates* e chamá-las dinamicamente. O sistema de *templating* do KnockoutJS pode ser usado com o auxílio de alguns mecanismos, como *UnderscoreJS* ou *jQuery.tmpl*.

Exemplo 3.25: Representação de um exemplo básico de *templating* em KnockoutJS.

```
<ul data-bind="foreach: names">
    <li data-bind="text: name"></li>
```

```
</ul>
<script>
    function MyViewModel() {
        this.names = [
            { name: "Nuno"},
            { name: "Filipa"},
            { name: "Carlos"}
        ]
    }
    ko.applyBindings(new MyViewModel());
</script>
```

O exemplo 3.25 mostra como é possível fazer uma lista de nomes, sendo obtido exatamente o mesmo resultado do exemplo 3.22 do AngularJS.

3.1.4.3 KendoUI

KendoUI é uma *framework* que permite criar aplicações web com HTML5 e JavaScript. Esta *framework* tem vindo a evoluir bastante, principalmente em termos de integração com as bibliotecas anteriormente referenciadas, pois embora possua o conceito de *data binding*, é recomendado o uso de uma biblioteca externa para esse trabalho.

KendoUI possui um pacote de controladores ricos para o rápido desenvolvimento em HTML5, permitindo aproximar-se dos existentes em Silverlight, contando com mais de cinquenta *widgets*.

Uma das grandes vantagens desta *framework* é que possibilita a junção com outras bibliotecas sem grande esforço, de forma a extender as funcionalidades pretendidas. O facto de ser uma *framework* com licença comercial leva-nos a que haja um maior suporte, tanto para correção de *bugs* como para esclarecimentos de dúvidas.

A Telerik mantém uma biblioteca que permite criar um conjunto de *bindings* de KnockoutJS para os *widgets* do KendoUI. Além disso, nos últimos meses também lançou uma para integrar com AngularJS.

A KendoUI possui o seguinte conjunto de *widgets* e *web frameworks*, conforme as categorias apresentadas abaixo:

Data management *Grid*, *ListView* e *PivotGrid*.

Scheduling *Calendar*, *GanttChart* e *Scheduler*.

Layout *Splitter*, *Tooltip* e *Window*.

Diagramming *Diagram*.

Geo Visualization *Map.*

Interactivity & UX *Progress Bar*, *Slider* e *Sortable.*

Editors *AutoComplete*, *Color Picker*, *ComboBox*, *Date and Time Pickers*, *DropDownList*, *Editor*, *Masked TextBox*, *MultiSelect* e *Numeric TextBox.*

Data Visualization *Barcode*,*Charts*, *Gauges*, *QR Code*, *Stock Charts* e *TreeMap.*

File Upload & Management *Upload.*

Navigation *Button*, *Menu*, *PanelBar*, *TabStrip*, *Toolbar* e *TreeView.*

Web frameworks *DataSource*, *Drag & Drop*, *Effects*, *Globalization*, *MVVM*, *Single-Page App*, *Templates* e *Validator.*

Esta *framework* apresenta ainda alguns *widgets* e *frameworks* para dispositivos móveis.

3.1.4.4 ExtJS

A *framework* ExtJS é desenvolvida pela Sencha, e atualmente é uma das *frameworks* mais completas e complexas para HTML5. Como é utilizada por um conjunto de grandes empresas, tais como a Microsoft, a Sony Ericsson, a Amazon ou a Fujitsu, o seu suporte certamente será assegurado durante mais tempo, o que pode ser uma mais-valia.

No momento em que esta *framework* foi avaliada para o desenvolvimento do caso de estudo, apenas permitia a utilização numa arquitetura MVC. No entanto, mais recentemente já é possível, igualmente, desenvolver na arquitetura MVVM.

O ponto forte desta *framework* é o conjunto complexo de controlos presentes, que correspondem a uma grande parte dos existentes em Silverlight. Por outro lado, como pontos negativos, é possível destacar o facto da mesma ser bastante fechada, isto é, não é fácil adicionar uma biblioteca auxiliar, bem como o facto de ter uma curva de aprendizagem pouco acentuada.

3.1.4.5 FabricJS

A biblioteca FabricJS é uma biblioteca *open source* que permite ajudar no manuseamento do *canvas*. O repositório do GitHub já conta com bastantes contribuidores e tem crescido bastante, no entanto, sendo este um projeto *open source*, poderá ser abandonado a qualquer momento.

Os pontos fortes desta biblioteca assentam na sua simplicidade de utilização, abstraindo da *API* do canvas em HTML5, podendo então preencher o mesmo com objetos geométricos, tais como quadrados, círculos, polígonos ou ainda objetos complexos, sendo estes compostos por centenas de *paths* ou junções dos objetos anteriormente descritos.

Suporta ainda *browsers* mais antigos, com a emulação da grande maioria das funcionalidades, isto é, sem suporte ao HTML5, incluindo as versões do Internet Explorer. É possível mover, escalar, rodar, preencher com uma cor, gradiente ou imagem, alterar as propriedades de opacidade e organização dos objetos no canvas.

3.1.4.6 KineticJS

A biblioteca KineticJS é uma biblioteca *open source* que permite auxiliar no manuseamento do *canvas*. O repositório do KineticJS já tem bastantes contribuidores e é uma das bibliotecas *open source* para suporte do canvas mais usada.

Os seus pontos fortes focam-se na forte versatilidade ou personalização daquilo que se pretende fazer, pois não há grande abstração da *API* do *canvas* disponibilizada pelo HTML5. No entanto, o facto de não abstrair da *API* implica um maior trabalho da parte do *developer* na criação de objetos mais elaborados.

3.1.5 Mapeamento de controladores em Silverlight para HTML5

Nesta subsecção serão analisados os controladores em Silverlight, nomeadamente a sua função, de forma a conseguirmos estabelecer uma comparação e relação entre os controlos nativos do HTML5 e do KendoUI e ExtJS.

Button controls Em Silverlight existem três tipos de controlos para especificar botões: *Button*, *HyperlinkButton* e *RepeatButton*.O primeiro adota o comportamento normal de um botão, disparando um evento quando é clicado, o segundo é um bloco de texto que quando clicado aponta para um destino, e o terceiro permite associar múltiplos eventos enquanto se encontra premido.

Silverlight	**HTML5 Nativo**	**KendoUI**	**ExtJS**
Button	*Button*	*Button*	*Button*
HyperlinkButton	*a*	*Button*	*Button*
RepeatButton	Não existe.	-	*Button*

Tabela 3.24: Relação de controladores com funcionalidade de botão em HTML5.

Selection controls Existem cinco tipos de controlos de seleção primitivos em Silverlight: *checkbox*, *combobox*, *listbox*, *radioButton* e *slider*. Todos eles encontram-se disponíveis em HTML5, como mostra a tabela 3.25.

Silverlight	**HTML5 Nativo**	**KendoUI**	**ExtJS**
CheckBox	Elemento *input* do tipo *Checkbox*	Elemento *input* do tipo *Checkbox*	*Checkbox*
ComboBox	*select*	*ComboBox*	*ComboBox*
ListBox	*select*	*DropDownList*	*Multiselect*
RadioButton	*input* do tipo *radio*	*input* do tipo *radio*	*Radio*
Slider	*input* do tipo *range*	*Slider*	*Slider*

Tabela 3.25: Relação de controladores com funcionalidade de seleção em HTML5.

Date display and selection Existem dois controlos para mostrar e selecionar datas: *calendar* e *date picker*.

Silverlight	**HTML5 Nativo**	**KendoUI**	**ExtJS**
Calendar	Não existe.	*Calendar*	*Date*
Datepicker	*input* do tipo *date*	*DatePicker*	*Date*

Tabela 3.26: Relação de controladores com funcionalidade de calendário em HTML5.

Information display (read-only) O Silverlight possui três controlos que permitem apresentar informação: *textBlock*, *ProgressBar* e *RichTextBlock*. Este último permite apresentar um bloco de texto que pode ser formatado, bem apresentar hiperligações, imagens e outros conteúdos complexos.

Silverlight	**HTML5 Nativo**	**KendoUI**	**ExtJS**
TextBlock	*p, span.*	-	-
ProgressBar	*progress*	*ProgressBar*	*ProgressBar*
RichTextBlock	Não existe	*Editor*	*Editor*

Tabela 3.27: Relação de controladores com funcionalidade de apresentar informação em HTML5.

Text display and editing Os controlos de edição de texto permitem apresentar e manipular a informação apresentada. Em Silverlight podem ser encontrados quatro diferentes: *AutoCompleteBox*, *PasswordBox*, *TextBox* e *RichTextBox*.

Silverlight	HTML5 Nativo	KendoUI	ExtJS
AutoCompleteBox	*input* com *datalist*	*AutoComplete*	-
PasswordBox	*input* to tipo *password.*	-	*Password*
TextBox	*input* do tipo *text.*	-	*Text*
RichTextBox	Não existe.	*Editor*	*Editor*

Tabela 3.28: Relação de controladores com funcionalidade de apresentar e editar informação em HTML5.

Data display Para apresentar informação existem os seguintes controlos: *DataGrid*, *DataPager*, *TreeView* e *PivotViewer*. Salienta-se ainda que o *DataPager* permite apenas dividir informação em páginas.

Silverlight	HTML5 Nativo	KendoUI	ExtJS
DataGrid	Não existe.	*Grid*	*Grid*
DataPager	Não existe.	-	-
TreeView	Não existe.	*TreeView*	*Tree*
PivotViewer	Não existe.	*PivotGrid*	

Tabela 3.29: Relação de controladores com funcionalidade de apresentar informação em HTML5.

Graphics and video display Para a representação de imagens e gráficos, o Silverlight apresenta os seguintes controlos: *DrawingSurface*, *Image*, *MultiScaleImage*, *MediaElement* e *InkPresenter*.

Silverlight	HTML5 Nativo	KendoUI	ExtJS
DrawingSurface	Não existe.	-	-
Image	Elemento *img.*	-	-
MultiScaleImage	Não existe.	-	-
MediaElement	Elementos *audio* e *video*	-	*audio* e *video*

Tabela 3.30: Relação de controladores com funcionalidade de apresentar gráficos e imagens em HTML5.

HTML display, out of browser O Silverlight possui o controlo *WebBrowser* que permite incorporar e mostrar conteúdo HTML armazenado noutros *hosts*.

Silverlight	HTML5 Nativo	KendoUI	ExtJS
WebBrowser	Elemento *iframe*	*window*	*window*

Tabela 3.31: Relação de controladores com funcionalidade de apresentar páginas HTML externas em HTML5.

Layout and element grouping O Silverlight disponibiliza, para disposição e agrupamento de elementos, os controlos *Border*, *Canvas*, *ContentControl*, *Grid*,*GridSplitter*, *StackPanel*, *ViewBox*, *VirtualizingStackPanel*, *ScrollBar*, *ScrollViewer* e *Tabcontrol*.

Silverlight	HTML5 Nativo	KendoUI	ExtJS
Border	Atributo *border*.	-	-
Canvas	Elemento *canvas*.	*Drawing API*	*Draw*
ContentControl	Não existe.	-	-
Grid	Elemento *table*	*Grid*	*Grid*
GridSplitter	Não existe.	*Splitter*	*Splitter*
StackPanel	Não existe.	*Panel*	*Panel*
ViewBox	Não existe.	-	-
VirtualizingStackPanel	Não há.	-	-
ScrollBar	Propriedade *overflow*.	*Scroller*	*Scroller*
ScrollViewer	Propriedade *overflow*.	*ScrollViewer*	-
Tabcontrol	Não existe.	*TabStrip*	*Tab*

Tabela 3.32: Relação de controladores com funcionalidade de dispor e agrupar elementos em HTML5.

User Help Em Silverlight existe um conjunto de controlos que permite melhorar a interação com o utilizador: *DescriptionViewer*, *Label*, *ToolTip* e *ValidationSummary*.

Silverlight	HTML5 Nativo	KendoUI	ExtJS
DescriptionViewer	Não existe.	*Validator*	-
Label	Elemento *label*.	-	-
ToolTip	Não existe.	*ToolTip*	-
ValidationSummary	Atributo *required*	*Validator*	-

Tabela 3.33: Relação de controladores com funcionalidade de melhorar a interação com o utilizador em HTML5.

Navigation A nível de navegação, o Silverlight apresenta os controlos *Frame* e *Page*.

Silverlight	**HTML5 Nativo**	**KendoUI**	**ExtJS**
Frame	Não existe.	-	-
Page	Não existe.	-	-

Tabela 3.34: Relação de controladores com funcionalidade de navegação em HTML5.

Dialog boxes and windows Por fim, o Silverlight ainda é composto por mais alguns controladores que permitem simular a ideia de janelas e diálogos: *OpenFileDialog*, *SaveFileDialog*, *ChildWindow* e *Popup*.

Silverlight	**HTML5 Nativo**	**KendoUI**	**ExtJS**
OpenFileDialog	Elemento *input* com o atributo *file*.	*Upload*	*File*
SaveFileDialog	Elemento *input* com o atributo *file*.	*Upload*	*File*
ChildWindow	Não existe.	*Window*	*Window*
Popup	Não existe.	*Window*	*Popup*

Tabela 3.35: Relação de controladores com funcionalidade de navegação em HTML5.

4 Caso de Estudo: análise da aplicação Diagrammer

O Diagrammer é uma aplicação direcionada para os prestadores de cuidados de saúde que lhes permite, a partir de diagramas que representam áreas específicas, fazer anotações das consultas.

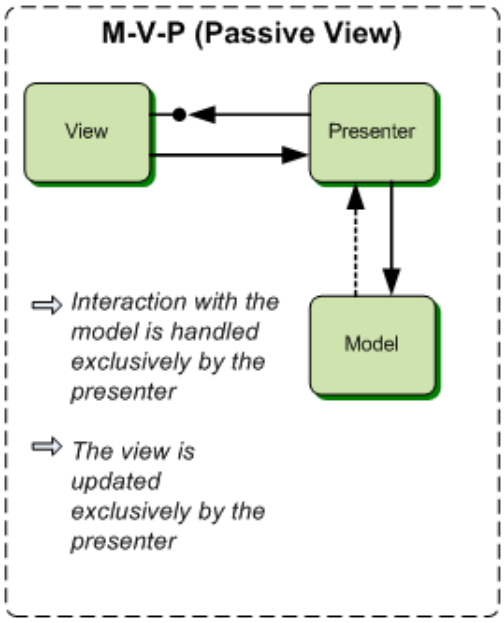


Figura 4.1: Variante *Passive View* do padrão *Model-View-Present*

Esta aplicação está subdividida em camadas lógicas, ou seja, em MVP (*Model-View-Presente*r), com a variante *Passive View*, como mostra a Figura 4.1. Esta representação é muito aproximada à MVVM, para a qual se pretende alterar. A aplicação pode ser dividida em diferentes áreas - cabeçalho, conteúdo e fundo -, sendo que em cada uma destas existem objetos que é necessário ter em conta para o seu correto funcionamento.

No cabeçalho está representado um simples logótipo e a informação referente ao utilizador, ao doente e ao estado da consulta/diagrama. Esta informação apenas necessitará de campos que representem texto, e apenas será alterada aquela que estiver relacionada com as consultas, se existirem alterações ao estado das mesmas.

No conteúdo encontram-se diferentes barras, como a barra de comandos e a barra de ferramentas, e ainda a área de representação dos diagramas com as devidas anotações. No topo encontra-se a barra de comandos, onde constam funcionalidades como "Novo", "Guardar", "Imprimir", "Ampliar", "Data e Hora", entre outros, tratando-se de simples botões. Na lateral esquerda existe a barra de ferramentas, que inclui um menu e botões simples, e que permite manipular o desenho e as anotações do diagrama. Na barra

lateral direita está representado um painel de anotações subdividido em cabeçalho, descrição e lista de anotações. Por fim, na parte central, encontra-se a representação do diagrama e das anotações, que podem ser objetos, como paralelepípedos, círculos ou elipses, linhas ou texto. Ainda terá vários eventos associados, como a seleção de um objeto, aparecimento do texto da anotação, introdução de novos textos ou alterações dos objetos, e ainda mover ou remover objetos ou anotações. No fundo apenas está representada uma barra de estado com uma caixa de texto contendo algum *feedback* criado pela aplicação, consoante a utilização.

Os botões de cada barra necessitarão também eles de eventos associados a cada uma das funcionalidades. De seguida serão mencionados todos os eventos dos botões:

- Na barra de comandos:
  - Novo: possibilidade de criar um novo diagrama;
  - Guardar e guardar todos: possibilidade de guardar ou guardar todos os diagramas que estão ativos no momento;
  - Imprimir: possibilidade de imprimir o diagrama atual;
  - Voltar ou avançar: possibilidade de retroceder ou avançar o último evento;
  - Cortar ou copiar: cortar ou copiar o diagrama atual;
  - Colar: colar o diagrama que está guardado no clipboard;
  - Ampliar ou diminuir: possibilidade de ampliar ou diminuir o diagrama e objetos representados;
  - Apagar: apagar o diagrama atual.
- Na barra de ferramentas:
  - Selecionar: possibilidade de selecionar objetos no diagrama;
  - Mover: possibilidade de mover objetos no diagrama;
  - Paralelepípedo: possibilidade de desenhar um paralelepípedo;
  - Elipse: possibilidade de desenhar uma elipse;
  - Linha: possibilidade de desenhar uma linha;
  - Linha customizada: possibilidade de desenhar uma linha customizada;
  - Menu de anotações: possibilidade de escolher um tipo de anotação no menu;

- Texto horizontal: possibilidade de escrever texto na horizontal;
- Texto vertical: possibilidade de escrever texto na vertical;
- Notas: possibilidade de adicionar notas;
- Apagar: possibilidade de apagar um dado objeto ou anotação.

Os eventos no painel de anotações resumem-se a adicionar, remover e editar anotações, sendo que o primeiro e o segundo são alimentados através de bindings, tal como na barra de estado.

Na aplicação ainda são apresentadas algumas caixas de diálogo que vão aparecendo como forma de auxílio a alguns dos eventos acionados.

O menu de anotações é preenchido por informação que terá de ser pedida ao serviço, constituída por uma imagem e o título de cada anotação. Os dados apresentados do utilizador e do doente também são requisitados ao serviço. É também possível guardar ou carregar diagramas elaborados, sendo necessário enviar ou pedir ao serviço a informação para cada um dos eventos.

5 Desenvolvimento do caso de estudo

Neste capítulo será abordado o método adotado para o procedimento da migração, abordando explicitamente a forma como deverá ser feito no futuro. O procedimento teve como base o Diagrammer, abordado no capítulo 4.

5.1 *Guidelines* para JavaScript

Antes de se proceder à migração, devem ser estipuladas um conjunto de regras para boas práticas em JavaScript. Estas permitem que a equipa trabalhe com o maior rendimento possível, porque todo o código será semelhante, permitindo ao *developer*, independentemente do ficheiro em que esteja a trabalhar, ler ou escrever o código da mesma forma, não necessitando então de reformatar ou decifrar a lógica do mesmo, e permitirá também encontrar *bugs* ou erros muito mais facilmente. [155]

Indentação Cada nível de indentação deverá ser separado por quatro espaços, substituindo os *tabs*.

Exemplo 5.1: Indentação em JavaScript.

```
if (true) {
    // Do something
}
```

Comprimento da Linha Cada linha não deverá possuir um tamanho superior aos 120 caracteres. Caso seja necessário, esta deverá ser quebrada, idealmente após uma vírgula. Caso isto aconteça, as linhas seguintes devem manter dois níveis de indentação (oito espaços).

Strings Quando declarada ou utilizada uma *string*, esta deverá ser colocada entre aspas em vez de plicas. Além disso, a barra invertida não deve ser usada para criar quebras de linha.

Exemplo 5.2: Strings em JavaScript.

```
var name = "Nuno␣Morais";
```

Vírgula flutuante Todo e qualquer número de vírgula flutuante deverá conter, pelo menos, um número à esquerda e outro à direita da vírgula.

Exemplo 5.3: Vírgula flutuante em JavaScript.

```
var abbreviationPi = 3.14;
```

Declaração de variáveis Todas as variáveis têm que ser declaradas antes de serem usadas numa função, e o uso de variáveis globais deverá ser minimizado ou até extinto. Numa função, todas as variáveis deverão ser declaradas no início, através do encadeamento das mesmas, separadas por vírgula e quebra de linha, com a mesma indentação. As mesmas deverão manter um nome sugestivo e ainda um pequeno comentário à frente sobre a função de cada uma delas.

Exemplo 5.4: Declaração de variáveis em JavaScript.

```
var seriesName, // Series name
    season,     // Number of Season
    episode,    // Number of Episode
    duration;   // Duration of episode
```

A declaração de objetos deverá ser sempre feita recorrendo ao {} em vez de *New Object()*, da mesma forma que a declaração de *arrays* deverá ser sempre com [] em vez de *New Array()*.

Espaçamento Todos os operadores com dois operandos deverão ser precedidos e seguidos por um espaço. Os operadores incluem *assignments* e operadores lógicos. Quando são utilizados parêntesis, não deverá existir um espaço imediatamente após abrir o parêntises e imediatamente antes de fechar. Deverá ser igualmente usado um espaço após o parêntesis dos argumentos da função.

Exemplo 5.5: Espaçamento em JavaScript.

```
var found = (values[i] === item);

if (values[i] === true) {
    // doSomething();
}

var defFunc = function(args) {
    // ...
}
```

Incrementação e Decrementação Não se deverá usar o duplo sinal de mais (++) ou menos (- -).

Exemplo 5.6: Espaçamento em JavaScript.

```
var inc = 0;
inc = inc + 1; // or, inc += 1;
```

Definição de Objetos Quando um objeto é declarado, existem algumas regras que deverão ser cumpridas: deve ser aberto um parêntesis logo após a sua declaração; cada par propriedade-valor deverá estar indentado um nível e a primeira propriedade deverá aparecer na linha a seguir após abrir o parêntises; cada propriedade não deverá estar entre plicas ou aspas e deverá ser seguida de vírgula após o valor; deverá ter uma linha em branco antes e depois da declaração da

função; poderão ser adicionadas quebras de linha entre propriedades, após um grupo de propriedades relacionadas ou desde que melhore a legibilidade; por fim, para fechar o parêntesis, deverá ser feita uma nova quebra de linha, seguida de ponto e vírgula.

Exemplo 5.7: Declaração de objetos em JavaScript.

```
var rectangle = {
    width: 100,
    height: 200,

    getArea: function() {
        return this.width * this.height;
    }
};
```

Comentários Podem ser escritos comentários numa única linha ou em várias. No primeiro caso, os comentários de única linha podem ser usados quando se pretende descrever a iteração que vem imediatamente a seguir ou imediatamente antes (comentando na mesma linha, logo após a iteração). Os múltiplos comentários de uma única linha são utilizados para comentar blocos de código que necessitam de mais informação, sendo iniciados sempre com /* e terminados com */, enquanto nas linhas centrais se recorre ao *.

Exemplo 5.8: Comentários em JavaScript

```
/*
 * Este metodo permite retornar a area de um
    rectangulo
 * independentemente da sua largura ou altura.
 */
 getArea: function() {
    return this.height * this.width;
 }
```

Nomes Em declaração de variáveis ou funções, os nomes estão limitados a caracteres alfanuméricos. Não se deve utilizar o dólar, $, excetuando em nomes de variáveis que recebem objetos jQuery, nem a barra invertida, \. O primeiro nome de variáveis não deverá ser um verbo para não ser confundido com funções, e caso se usem várias palavras na variável, estas deverão ser iniciadas com letra maiúscula, excetuando a primeira. Variáveis que atuam como constante devem ser formatadas em maiúsculas e poderão ser separadas com *underscore*.

Exemplo 5.9: Nomes de variáveis em JavaScript.

```
var accountNumber = "888-1";  // Var

var \$header = \$("#header"); // Var jQuery

var TOTAL_COUNT = 10; // Constante
```

As regras para a declaração de funções são semelhantes às de declaração de variáveis, excetuando o facto de a primeira ter que ser iniciada por um verbo.

Exemplo 5.10: Nomes de funções em JavaScript.

```
function getArea(width, height) {
    // doSomething()
}
```

Funções construtor são aquelas que são usadas com o operador *new* para criar novos objetos, sendo sempre iniciadas com letra maiúscula e não podendo começar com um verbo.

Exemplo 5.11: Nome de objetos em JavaScript.

```
function Rectangle() {
    // code
}
```

Variáveis e métodos privados possuem as mesmas convenções das públicas, com a exceção de que iniciam com *underscore*.

Exemplo 5.12: Nome de objetos em JavaScript.

```
var object = {
    _count: 10,

    _getCount: function() {
        return this._count;
    }

};
```

Strict Mode Deverá ser introduzido nas funções e nunca globalmente.

Exemplo 5.13: *Strict Mode* em JavaScript.

```
(function() {
    function doSomething() {
        // ...
    }
});
```

Comparações As comparações em JavaScript devem, na maioria dos casos, ser feitas através de três sinais de igual (===) em vez de dois (==). Isto permitirá verificar se os tipos de objetos são também iguais.

Exemplo 5.14: Comparações em JavaScript.

```
    var same = (a === b);
```

If aninhado O *if* aninhado deverá ser usado para devolver um de dois valores possíveis e não como um atalho.

Exemplo 5.15: *If* aninhado em JavaScript.

```
// Uso correto
var same = (a === b) ? a : b;

// Uso incorreto
var same = (a === b) ? doSomething() :
   doSomethingElse();
```

Instruções Cada linha deverá ter apenas uma instrução, terminando com ponto e vírgula.

Exemplo 5.16: *Instruções* em JavaScript.

```
var same = (a === b) ? a : b;
b += 1;
```

A instrução de retorno não deverá usar parêntesis, a menos que faça sentido para envolver outra instrução.

Exemplo 5.17: *Return* em JavaScript.

```
doSomething() {
    return 1;
}

doSomethingElse() {
    return (size > 0 ? size : 10);
}
```

A declaração do *if* poderá ser apresentada de três formas, como mostra o exemplo 5.18. As chavetas nunca poderão ser omitidas.

Exemplo 5.18: *If* em JavaScript.

```
// 1
if (condition) {
    // Statements
}

// 2
if (condition) {
    // statements
} else {
    // statements
}

// 3
```

```
if (condition) {
    // statements
} else if (condition) {
    // statements
} else {
    // statements
}
```

A declaração do ciclo *for* possui duas versões, uma para percorrer *arrays* e outra para objetos. A primeira é composta pela inicialização das variáveis, depois a condição e por fim a atualização. Se satisfizer a condição, então executará o conjunto de intruções que estiverem entre chavetas. A segunda apenas é ligada por *in*, antecedido da variável que contém os objetos e precedido pelo nome da variável que irá ficar com o objeto a cada *loop*. As chavetas nunca poderão ser omitidas.

Exemplo 5.19: Ciclo *For* em JavaScript.

```
for (i = 0; i < 10; i += 1) {
    // statements
}

for (shapes in shape) {
    // statements
}
```

As declarações dos ciclos *while* e *do while* são bastante parecidas, como mostra o exemplo 5.20. As chavetas nunca poderão ser omitidas.

Exemplo 5.20: Ciclo *while* e *do while* em JavaScript.

```
while (condition) {
    // statements
}

do {
    // statements
} while (condition);
```

A instrução de *switch* deverá manter a forma tradicional, com uma indentação em cada *case* com o *switch*, e as instruções novamente indentadas em relação ao *case*. Caso não exista valor de *default*, deverá estar indicado em comentário com *//no default*.

Exemplo 5.21: Instrução de *switch* em JavaScript.

```
switch (expression) {
    case expression:
        statements
    default:
```

```
            statements
    }
```

A declaração do *try* poderá ter apenas o *catch* ou, além deste, o *finally*, como se encontra demonstrado no exemplo 5.22.

Exemplo 5.22: Instrução de *try/catch* em JavaScript.

```
try {
    // statement
} catch (exception) {
    // statement
}

try {
    // statement
} catch (exception) {
    // statement
} finally {
    // statement
}
```

5.2 Modelo de Migração

Para o rápido desenvolvimento do caso de estudo, foram escolhidas as seguintes bibliotecas: RequireJS, KnockoutJS, KendoUI e FabricJS. No entanto, estas trazem outras como dependências, como é o caso da jQuery. A escolha das mesmas resume-se ao facto de serem bastante boas na função a que se destinam, mas também por serem muito flexíveis na introdução de outras bibliotecas para outras finalidades. Para esta escolha foi necessário prestar atenção ao crescimento das mesmas, sendo que, neste momento, apenas existem algumas estatísticas prestadas pelos seus próprios repositórios e algum feedback dado em algumas comunidades, como é o caso do *StackOverflow*.

Uma aplicação em HTML5 é composta por ficheiros HTML, CSS, JavaScript e imagens, sendo que parte deles apenas se referem à aplicação, enquanto outros podem ser partilhados nos diferentes projetos. Com base nisso, uma aplicação genérica em HTML5 deverá ficar organizada como apresentada na figura 5.1.

Com esta organização de pastas é possível separar os diferentes tipos de ficheiros, no entanto poderá tornar-se complicado fazer esta separação ao importar novas bibliotecas, pois deve-se obedecer à estrutura estipulada. Dentro da pasta CSS irão constar os ficheiros correspondentes às folhas de estilo, na JS irão ser adicionados todos os JavaScript, que por sua vez poderão ser correspondentes a bibliotecas, *models* ou *viewmodels*, por fim a pasta *views* irá conter as páginas HTML necessárias para a aplicação.

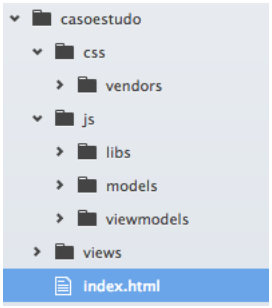


Figura 5.1: Representação da organização em pastas de uma aplicação em HTML5.

Depois da estruturação de pastas da aplicação, é então necessário importar as bibliotecas e os seus ficheiros para a pasta correspondente e, posteriormente, procerder à configuração para o RequireJS. Com este poderemos definir todo o conjunto de bibliotecas e caminhos necessários para fazer o carregamento correto de cada módulo com todas as suas dependências. Para isso, basta definir um ficheiro com a seguinte estrutura:

Exemplo 5.23: Configuração do RequireJS.

```
require.config({
    paths: { // paths
        jquery: 'libs/jquery/jquery-2.1.0.min',
        kendo: 'libs/kendoui/kendo.web',
        knockout: 'libs/knockout/knockout-3.1.0',
        kok: 'libs/kokendo/knockout-kendo.min',
        viewModels: 'viewModels',
        models: 'models',
        fabric: 'libs/fabric'
        //...
    },
    shim: { // dependencies
        "kendo" : {
            deps : ["jquery"]
        },
        "knockout" : {
            deps: ["jquery"]
        },
        "kok" : { // inform requirejs that kokendo depends on
            kendoui and knockout
            deps : ["kendo", "knockout"]
        }
        //...
    }
});

require([
    // Load our app module and pass it to our definition
       function
```

```
    'app'
    ], function (App) {
        'use strict';
        // The "app" dependency is passed in as "App"
        App.initialize();
    }
);
```

No exemplo 5.23 podemos verificar que, inicialmente, foram definidos um conjunto de *paths*, de forma a possibilitar o reconhecimento de todas as bibliotecas, *models* e *viewModels*. Posteriormente, foi passada a informação ao RequireJS de que determinadas bibliotecas dependem de outras, ou seja, só as poderá carregar após as suas dependências estarem completas. Por fim, foi definido um módulo para inicializar a aplicação.

Para facilitar o desenvolvimento do caso de estudo, procedeu-se a algumas alterações ao serviço WCF, de forma a passar *SOAP* para *RestFul* e responder no formato JSON, pois o lado do cliente tirará maior partido deste tipo de formato.

Do lado do cliente foi ainda necessário desenvolver uma pequena biblioteca que permitisse fazer de proxy para o serviço, tendo sido criados para esse efeito todos os métodos disponibilizados para o serviço. Para a comunicação recorreu-se à tecnologia *AJAX*, possibilitando, dessa forma, o envio e receção de dados.

Nas próximas sub-secções será feita a explicação sobre como se deverá proceder para migrar o *model*, *model-view* e *view*.

5.2.1 *Model*

As entidades do serviço podem ser partilhadas com o cliente Silverlight, passando estas a fazer parte do modelo do cliente, sem ser necessário reescrevê-las. No entanto, em HTML5, caso estas possuam o atributo *DataContract* na sua classe, será necessário reescrevê-las em JavaScript, uma vez que essas vão ser passadas entre o servidor e o cliente.

A figura 5.2 apresenta as entidades do serviço, e podemos verificar que na classe *ShapesForm* está lá apresentado o atributo *DataContract*. Uma vez que o JavaScript possui uma forma muito simplista de escrever, este processo de migração é muito prático de realizar.

O RequireJS e o jQuery irão auxiliar na conversão de todo o modelo. Para isso, é necessário defini-lo com todas as dependências, sendo, neste caso, a biblioteca de jQuery e o modelo *User*.

Depois é declarada uma função com o mesmo nome da classe que estamos a migrar, e chamamos um método para inicializar, ou seja, um construtor. Para que as classes

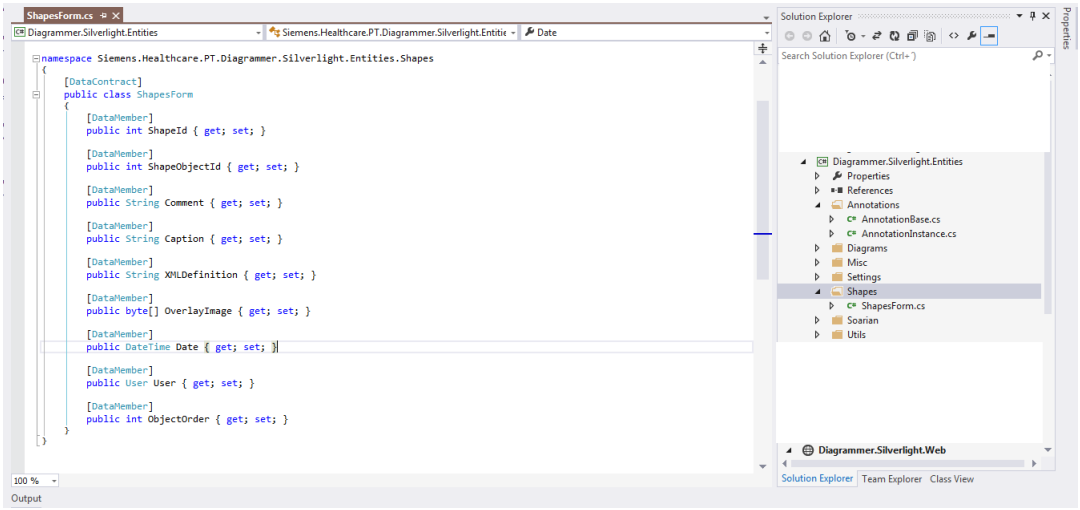


Figura 5.2: Entidades do Projeto.

sejam genéricas e possam ser extendidas, é utilizado o *extend* do jQuery, possibilitando então a declaração de todas as variáveis e métodos necessários.

Exemplo 5.24: Model *ShapesForm* em *JavaScript*.

```
define([
    'jquery',
    'models/Sorian/userModel'
], function ($, User) {

    function ShapesForm(params) {
        this.init(params);
    }

    $.extend(ShapesForm.prototype, {

        ShapeId: -1, //... Others vars

        init: function(params) { /* constructor */ },

        getShapeId: function() { return this.ShapeId; },

        setShapeId: function(shapeId) { this.ShapeId =
           shapeId; },

        // Others Gets and Sets
    });
    return ShapesForm;
});
```

Pode-se observar no exemplo 5.24 que não há informação do tipo de variável. À exceção do que foi escrito nas *Guidelines em JavaScript*, as variáveis foram iniciadas

com letra maiúscula, uma vez que é necessário que todos os nomes estejam escritos da mesma forma para que o serviço os reconheça.

O *extend* permite extender a própria classe, bem como uma classe derivada de outras. Para isso, no primeiro parâmetro de entrada é atribuído o *true*, uma vez que permite fazer uma cópia em profundidade de todos os objetos dos restantes parâmetros de entrada para o objeto do segundo parâmetro. No exemplo 5.25, a classe *ShapesForm* irá ter todas as propriedades da classe *GenericObject*. De seguida, poder-se-á continuar a extender a própria classe.

Exemplo 5.25: Extensão de uma classe.

```
define([
    'jquery',
    'models/Shapes/genericObjectModel',
    'models/Sorian/userModel',
    'custom'
], function ($, GenericObject, User) {

    function ShapesForm(params) {
        this.init(params);
    }
    $.extend(true, ShapesForm.prototype, GenericObject.
       prototype);

    $.extend(ShapesForm.prototype, {
        //...
    });

    return ShapesForm;
});
```

5.2.2 *View*

As *views* em Silverlight são escritas em XAML, contendo os controladores descritos no capítulo *Mapeamento de controladores em Silverlight para HTML5* (3.1.5). Com base nisso, será abordado como migrar cada um dos controlos para HTML, com o auxílio das bibliotecas Knockout-Kendo e FabricJS, se necessário, não contendo ainda qualquer tipo de informação sobre os *bindings*.

5.2.2.1 *Button*

Exemplo 5.26: Implementação de um botão em Silverlight.

```
<Button Content="Click me!"></Button>
```

O controlo *button* em Silverlight pode ser migrado facilmente para HTML5, uma vez que existe neste último, e para isso basta escrever o elemento *button*, remover o atributo *Content* e adicioná-lo entre o abrir e fechar da *tag button*, como mostra no exemplo 5.27. Uma vez que este controlo é suportado pelo KendoUI, é inicializado dentro do atributo *data-bind*, seguido de *kendoButton*.

Exemplo 5.27: Implementação de um botão em HTML5.

```
<button data-bind="kendoButton:␣..."">Click me!</button>
```

5.2.2.2 *HyperlinkButton*

Exemplo 5.28: Implementação de um *HyperlinkButton* em Silverlight.

```
<HyperlinkButton Content="Google" NavigateUri="http://google.
   com" TargetName="_blank"/>
```

Uma vez que um *HyperlinkButton* em Silverlight é normalmente adicionado conforme o exemplo 5.28, o processo de migração deste controlo baseia-se em alterar os nomes de *HyperlinkButton* para *a*, *NavigateUri* para *href* e *TargetName* para *target*. Por fim, deve ser removido o atributo *Content* e aplicado o conteúdo entre as *tags* de abertura e fecho, como mostra o exemplo 5.29.

Exemplo 5.29: Implementação de um botão com hiperligação em HTML5.

```
<a href="http://google.com" target="_blank">Google</a>
```

5.2.2.3 *CheckBox, RadioButton, Slider, TextBox, PasswordBox, Datepicker*

A migração dos controlos *CheckBox, RadioButton, Slider, TextBox, Datepicker* e *PasswordBox* para HTML5 resume-se a um único elemento, sendo este o *input*. Para especificar o tipo de controlo, basta alterar o atributo *type* de acordo com o pretendido, como está especificado na secção de *Mapeamento de controladores em Silverlight para HTML5* (3.1.5).

Exemplo 5.30: Implementação de uma *CheckBox* em Silverlight.

```
<CheckBox Content="Yes" />
```

O controlo *CheckBox* especificado no exemplo 5.30 traduz-se em HTML conforme está representado no exemplo 5.31.

Exemplo 5.31: Implementação de uma *CheckBox* em HTML5.

```
<input type="checkbox" />Yes
```

Os controlos *Slider* e *DatePicker* devem ser inicializados adicionando o atributo *data-bind* e especificando *kendoSlider* e *kendoDatePicker*, uma vez que o KendoUI possui suporte para os mesmos.

5.2.2.4 *ComboBox* e *ListBox*

Exemplo 5.32: Implementação de uma *ComboBox* em Silverlight.

```
<ComboBox>
    <ComboBoxItem Content="Coffe"></ComboBoxItem>
</ComboBox>
```

Em Silverlight, a definição de uma *ComboBox* ou *ListBox* está estruturada da mesma forma que em HTML5, embora com nomes diferentes. As *tags* *ComboBox* e *ListBox* são traduzidas para *Select*, e as *ComboBoxItem* e *ListBoxItem* para *Option*. Uma vez que existem os controlos na biblioteca KendoUI, para instanciar o controlo basta adicionar uma *input* com o atributo *data-bind* e o tipo *kendoComboBox*, se for uma *ComboBox*, ou *kendoDropDownList* se for uma *ListBox*.

Exemplo 5.33: Implementação de uma *ComboBox* em HTML5.

```
<input data-bind="kendoComboBox: /..." />
```

5.2.2.5 *Calendar*

O controlo *Calendar* do Silverlight não existe de forma nativa no HTML5, no entanto, este está representado na biblioteca KendoUI, fazendo o mesmo que em Silverlight.

A inicialização em Silverlight faz-se através do *namespace sdk*, seguido de *Calendar*, como mostra o exemplo 5.34.

Exemplo 5.34: Implementação de uma *Calendar* em Silverlight.

```
<sdk:Calendar />
```

Em HTML5 é igualmente fácil, bastando declarar uma *div* e inicializar com o *kendoCalendar*, como está representado no exemplo 5.35.

Exemplo 5.35: Implementação de um *Calendar* em HTML5.

```
<div data-bind="kendoCalendar: /..."> </div>
```

5.2.2.6 *TextBlock*

O *TextBlock* em Silverlight pode corresponder a vários elementos em HTML5, permitindo então ser *p*, *span* e *h1* até *h6*, sendo usada a que melhor se enquadrar com o caso.

O exemplo 5.36 define uma *TextBlock* com um pequeno texto integrado.

Exemplo 5.36: Implementação de uma *TextBlock* em Silverlight.

```
<TextBlock Text="Sample␣text␣to␣display"></TextBlock>
```

Independentemente do elemento escolhido para a migração, a *tag TextBlock* é traduzida pela *tag* escolhida em HTML5 e é removido o atributo *Text*, adicionando-o entre o abrir e fechar do elemento, como é mostrado no exemplo 5.37.

Exemplo 5.37: Implementação de uma *TextBlock* em HTML5.

```
<span>Sample text to display</span>
```

5.2.2.7 *ProgressBar*

O controlo de *ProgressBar* em Silverlight corresponde ao *Progress* em HTML5. O KendoUI possui igualmente este controlo, no entanto, deve ser utilizado o nativo, uma vez que contém um maior número de funcionalidades.

Exemplo 5.38: Implementação de uma *ProgressBar* em Silverlight.

```
<ProgressBar />
```

Em HTML5 a inicialização do controlador é igual, alterando o *ProgressBar* para *Progress*. No entanto, como é para usar o do KendoUI, esta é feita de forma ligeiramente diferente, uma vez que é adicionado o elemento *div* e, posteriormente, inicializado com *kendoProgressBar*.

Exemplo 5.39: Implementação de uma *ProgressBar* em HTML5.

```
<div data-bind="kendoProgressBar:␣/..."> </div>
```

5.2.2.8 *RichTextBlock* e *RichTextBox*

Os controlos *RichTextBlock* e *RichTextBox* não são suportados nativamente em HTML5, necessitando, uma vez mais, do suporte do KendoUI.

A inicialização destes controlos é igual, alterando simplesmente o nome de cada um deles. O exemplo 5.40 mostra a representação de uma *RichTextBox*.

Exemplo 5.40: Implementação de uma *RichTextBox* em Silverlight.

```
<RichTextBox VerticalScrollBarVisibility="Auto">
    <Paragraph>A RichTextBox with <Bold>initial content</Bold
       > in it.</Paragraph>
</RichTextBox>
```

No KendoUI apenas é usado o *Editor* para representar estes dois controlos, uma vez que depois é personalizado com o atributo *contenteditable*. No caso de o valor ser

true, desempenha o papel de *RichTextBox*, e caso seja *false*, assume-se como *RichTextBlock*. Para a migração do controlo dever-se-á adicionar uma *textarea* e inicializar com *kendoEditor*.

Exemplo 5.41: Implementação de uma *RichTextBox* em HTML5.

```
<textarea rows="10" cols="20" data-bind="kendoEditor: { value
   : content, tools: ['bold', 'italic'] }" > </textarea>
```

5.2.2.9 *AutoCompleteBox*

O controlo *AutoCompleteBox* corresponde ao elemento *input* associado ao *datalist* em HTML5, no entanto, uma vez que o KendoUI possui este controlo bem desenvolvido, será esse então o selecionado.

Exemplo 5.42: Implementação de uma *AutoCompleteBox* em Silverlight.

```
<sdk:AutoCompleteBox />
```

A migração do controlo para HTML5 é igualmente simples, bastando alterar o *namespace SDK* seguido de *AutoCompleteBox*, como mostra o exemplo 5.42, por *input*, inicializando este com *kendoAutoComplete*, como demonstrado em 5.43.

Exemplo 5.43: Implementação de uma *AutoCompleteBox* em HTML5.

```
<input data-bind="kendoAutoComplete" />
```

5.2.2.10 *DataGrid*

A migração desta informação passa por compactar todas as *tags sdk:DataGrid* para *table*, *sdk:DataGrid.Columns* para *tr* e seguido de *thead*, e por fim *sdk:DataGridTextColumn* para *th*, onde é removido o atributo *header* e adicionado esse conteúdo entre o abrir e fechar desta última *tag*.

Exemplo 5.44: Implementação de uma *DataGrid* em Silverlight.

```
<sdk:DataGrid>
    <sdk:DataGrid.Columns>
        <sdk:DataGridTextColumn Header="First Name"/>
    </sdk:DataGrid.Columns>
</sdk:DataGrid>
```

Uma vez usado o KendoUI não será necessário grande parte dessas *tags* sendo então definida uma *div* e inicializando com *kendoGrid*.

No exemplo 5.45 está representada a migração de uma *DataGrid* para HTML5.

Exemplo 5.45: Implementação de uma *DataGrid* em HTML5.

```
<div data-bind="kendoGrid: //..."> </div>
```

5.2.2.11 *DataPager*

O controlo *DataPager* não existe em HTML5 nem na biblioteca auxiliar KendoUI, uma vez que normalmente está associado ao controlo necessário, como por exemplo a *Grid*. Deste modo, o controlo específico não poderá ser migrado *naturalmente*, mas deverá ser analisado no momento da migração se será necessário simular este tipo de comportamento.

5.2.2.12 *TreeView*

O controlo *TreeView* não é suportado nativamente pelo HTML5, no entanto a biblioteca do KendoUI suporta-a. Este controlador é representado estruturalmente da mesma maneira, sendo apenas necessário alterar o nome dos *elementos*.

Exemplo 5.46: Implementação de uma *TreeView* em Silverlight.

```
<sdk:TreeView>
    <sdk:TreeViewItem Header="TreeViewItem containing other
       items.">
        <sdk:TreeViewItem.Items>
            <sdk:TreeViewItem>
                <TextBlock Text="Item 1" Margin="2"/>
            </sdk:TreeViewItem>
        </sdk:TreeViewItem.Items>
    </sdk:TreeViewItem>
</sdk:TreeView>
```

Para se proceder à migração deste *controlo* será necessário alterar as *tags sdk:TreeView* e *sdk:TreeViewItem.Items* por *ul*, inicializando com *kendoTreeView*, e a *sdk:TreeViewItem* por *li*, como mostra no exemplo 5.47.

Exemplo 5.47: Implementação de uma *TreeView* em HTML5.

```
<ul data-bind="kendoTreeView: {}">
    <li>
        <span>TreeViewItem containing other items.</span>
        <ul>
            <li>Item 1</li>
        </ul>
    </li>
</ul>
```

5.2.2.13 *Image*

O controlo *image* em Silverlight é muito simples de converter para HTML5, uma vez que apenas é necessário alterar o nome da *tag* para *img* e mudar o atributo *Source* para *src*.

Exemplo 5.48: Implementação de uma *Image* em Silverlight.

```
<Image Source="Test.jpg"/>
```

De acordo com o que foi dito, o exemplo 5.48 é convertido em *exHTMLImage*.

Exemplo 5.49: Implementação de uma *Image* em HTML5.

```
<img src="Test.jpg" />
```

5.2.2.14 *MediaElement*

O controlo de *MediaElement* em Silverlight corresponde à junção dos elementos *audio* e *video* em HTML5, uma vez que estes permitem reproduzir ficheiros de áudio e/ou vídeo.

Exemplo 5.50: Implementação de um *MediaElement* em Silverlight.

```
<MediaElement Source="media\numbers.wmv" />
```

Uma vez que o suporte para a descodificação dos mesmos varia entre browsers, é necessário adicionar várias fontes em HTML5. Para se proceder à migração do controlo é então essencial alterar o nome do controlo para *video* ou *audio* e adicionar as fontes a partir da *tag source* e atributo *src*, como mostra o exemplo 5.51.

Exemplo 5.51: Implementação de um *MediaElement* em HTML5.

```
<video width="320" height="240" controls>
    <source src="media\numbers.mp4" type="video/mp4">
    <source src="media\numbers.ogg" type="video/ogg">
    Your browser does not support the video tag.
</video>
```

5.2.2.15 *WebBrowser*

O controlo *WebBrowser* pode ser facilmente migrado para *HTLM5*, bastando alterar o nome para *iframe* e o atributo *source* para *src*.

Exemplo 5.52: Implementação de um *WebBrowser* em Silverlight.

```
<WebBrowser Source="http://google.com"/>
```

Conforme o que foi dito, o exemplo 5.52 passa a ser idêntico ao 5.53.

Exemplo 5.53: Implementação de um *WebBrowser* em HTML5.

```
<iframe src="http://google.com"></iframe>
```

5.2.2.16 *Border*

O controlo *Border* não existe em HTML5, uma vez que pode ser definido em qualquer elemento HTML através de CSS, a partir da propriedade *border*.

5.2.2.17 *Canvas*

O controlo *canvas* existe em HTML5, no entanto, uma vez que com o auxílio da biblioteca FabricJS é possível tirar um melhor partido do mesmo, a migração será efetuada nesse sentido.

Exemplo 5.54: Implementação de um *Canvas* em Silverlight.

```
<Canvas Height="400" Width="400"></Canvas>
```

Tanto em Silverlight como em HTML5 a declaração do canvas é feita da mesma forma, restando neste último ser inicializado a partir do FabricJS, como mostra no exemplo 5.55.

Exemplo 5.55: Implementação de um *Canvas* em HTML5.

```
<canvas id="myCanvas"></canvas>
<script>
    var canvas = new fabric.Canvas('myCanvas')
</script>
```

5.2.2.18 *Grid*

O controlo *grid* em Silverlight é relativamente semelhante ao *DataGrid* (5.2.2.10), sendo então necessário substituir as *tags* *Grid* por *table*, *Grid.ColumnDefinitions* e *Grid.RowDefinitions* por *tr*, e *RowDefinition* por *td*.

Exemplo 5.56: Implementação de um *Grid* em Silverlight.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition />
        <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
```

```
        <RowDefinition />
    </Grid.RowDefinitions>

    <TextBlock Grid.Row="0" Grid.Column="0">Title 1</
       TextBlock>
    <TextBlock Grid.Row="0" Grid.Column="1">Title 2</
       TextBlock>

    <TextBlock Grid.Row="1" Grid.Column="0">BMW</TextBlock>
    <TextBlock Grid.Row="1" Grid.Column="1">Renault</
       TextBlock>
</Grid>
```

Como pode ser observado no exemplo 5.56, cada elemento referente às diferentes linhas e colunas da tabela está contido apenas no interior de *TextBlock*, sendo então necessário ainda substituir o conteúdo de cada um para dentro das *tags th* e *td* correspondentes, como se pode observar no exemplo 5.57.

Exemplo 5.57: Implementação de um *Grid* em HTML5.

```
<table>
    <tr>
        <th>Title 1</th>
        <th>Title 2</th>
    </tr>
    <tr>
        <td>BMW</td>
        <td>Renault</td>
    </tr>
</table>
```

5.2.2.19 *GridSplitter*

O controlo *GridSplitter* em Silverlight funciona com o auxílio do controlo *Grid*, que aplica funcionalidades ao primeiro. Em HTML5, além de este não ser suportado nativamente, o suporte oferecido pelo KendoUI não vai de encontro à estrutura existente em Silverlight, ou seja, em vez de aplicar o *GridSplitter* a uma *table*, este aplica a *divs*.

Exemplo 5.58: Implementação de um *GridSplitter* em Silverlight.

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
```

```
    <TextBlock Grid.Row="0" Grid.Column="0">Title 1</
       TextBlock>
    <TextBlock Grid.Row="0" Grid.Column="1">Title 2</
       TextBlock>

    <GridSplitter Grid.Row="0" Grid.Column ="0" Grid.
       ColumnSpan="2" />
</Grid>
```

Para migrar este controlo é necessário estruturar, inicialmente, todas as colunas com *divs* e envolver as mesmas com outra *div*, inicializando o *KendoSplitter*, como mostra no exemplo 5.59

Exemplo 5.59: Implementação de um *GridSplitter* em HTML5.

```
<div data-bind="kendoSplitter:␣{}">
    <div>Pane One</div>
    <div>Pane Two</div>
</div>
```

5.2.2.20 *StackPanel*

O controlo *StackPanel* em Silverlight é aquele que permite agrupar outros controlos, sendo idêntico à *div* em HTML5.

Exemplo 5.60: Implementação de um *StackPanel* em Silverlight.

```
<StackPanel></sctackPanel>
```

No momento da migração, o *StackPanel* será então alterado para *div*.

Exemplo 5.61: Implementação de um *StackPanel* em HTML5.

```
<div></div>
```

5.2.2.21 *ScrollBar, ScrollViewer*

Os controlos *ScrollBar* e *ScrollViewer* não existem em HTML5, uma vez que podem ser definidos em qualquer elemento HTML através de CSS, a partir da propriedade *overflow*.

5.2.2.22 *TabControl*

O controlo *TabControl* em Silverlight não possui suporte nativo em HTML5 mas, mais uma vez, é mantido pela biblioteca do KendoUI. A migração deste controlo é

acessível, embora não seja organizado da mesma forma que em Silverlight.

Exemplo 5.62: Implementação de um *TabControl* em Silverlight.

```
<sdk:TabControl  Height="200" Width="200">
    <sdk:TabItem Header="Tab␣1">
        /...
    </sdk:TabItem>
    <sdk:TabItem Header="Tab␣2">
        //..
    </sdk:TabItem>
</sdk:TabControl>
```

Para esta migração é necessário adicionar um *container*, neste caso uma *div*, que será inicializado com *kendoTabStrip*. De seguida, dever-se-á adicionar uma lista não ordenada (*UL*) e percorrer todos os *TabItem* em Silverlight, adicionando um elemento *LI* por cada um deles. Entre cada abrir e fechar de *tag* deve ser adicionado o conteúdo presente no atributo *Header*. Por fim, após a lista não ordenada, é adicionada uma *div* e o seu conteúdo por cada *item*.

Exemplo 5.63: Implementação de um *TabControl* em HTML5.

```
<div data-bind="kendoTabStrip:␣{}">
    <ul>
        <li class="k-state-active">Tab 1</li>
        <li>Tab 2</li>
    </ul>
    <div>
        <div class="for-tab-1">// ... </div>
    </div>
    <div>
        <div class="for-tab-2">// ... </div>
    </div>
</div>
```

5.2.2.23 *DescriptionViewer, ValidationSummary*

Os controlos *DescriptionViewer* e *ValidationSummary* funcionam de uma maneira um pouco diferente em HTML5, mesmo com o auxílio do KendoUI. No entanto, uma vez que podem ser minimamente simulados, foram migrados com a adaptação que o KendoUI sugere, embora funcionem os dois em conjunto.

Exemplo 5.64: Implementação de um *DescriptionViewer* em Silverlight.

```
 <TextBox Text=""/>
<sdk:DescriptionViewer Description="Email␣format␣is␣not␣valid
   "/>
```

Dessa forma, uma vez dado um tipo de *input*, o mesmo é migrado como anteriormente descrito para HTML5 e depois, caso haja um *DescriptionViewer* e/ou um *ValidationSummary*, são adicionados os atributos *required*, *validationMessage* e *data-email-msg*, conforme especificado no exemplo 5.65.

Exemplo 5.65: Implementação de um *DescriptionViewer* em HTML5.

```
<input type="email" placeholder="my@email.com" required
   validationMessage="Enter␣{0}" data-email-msg="Email␣format
   ␣is␣not␣valid" />
```

5.2.2.24 *Label*

O controlo *Label*, à semelhança de outros controlos, é migrado mantendo o mesmo nome e removendo o atributo *Content*, sendo que o seu conteúdo é adicionado entre o abrir e fechar da *tag*.

Exemplo 5.66: Implementação de um *Label* em Silverlight.

```
<sdk:Label Content="Test␣Content␣Label" />
```

O exemplo 5.67 mostra a migração para HTML5 do exemplo 5.66 em Silverlight.

Exemplo 5.67: Implementação de um *Label* em HTML5.

```
<label>Test Content Label</label>
```

5.2.2.25 *ToolTip*

O controlo *ToolTip* em Silverlight não possui nenhum controlo associado em HTML5, mas o KendoUI consegue suportá-lo.

Exemplo 5.68: Implementação de um *ToolTip* em Silverlight.

```
<ToolTip>
    <TextBlock Text="Image␣and␣text"></TextBlock>
</ToolTip>
```

A migração de uma *ToolTip* para HTML5 passa por definir a zona onde se pretende que apareça e inicializá-la com *KendoToolTip*. Na inicialização é possível adicionar o campo *content* onde se introduz o conteúdo da *TextBlock* apresentada no exemplo 5.68.

Exemplo 5.69: Implementação de um *ToolTip* em HTML5.

```
<div data-bind="kendoTooltip:␣{␣content:␣'Image␣and␣text'␣}">
    Hover Me!
</div>
```

5.2.2.26 *OpenFileDialog, SaveFileDialog*

Os controlos *OpenFileDialog* e *SaveFileDialog* em Silverlight, uma vez chamados a partir do método do evento *OnClick* de um botão, são inicializados no ficheiro *.cs* associado à *View*, com *new OpenFileDialog()*, como é apresentado no exemplo 5.70.

Exemplo 5.70: Implementação de um *OpenFileDialog* em Silverlight.

```
// .xaml
<Button Content="Open␣File␣Click="bOpenFileDialog_Click"␣/>

//␣.CS␣File
private␣void␣bOpenFileDialog_Click(object␣sender,␣
   RoutedEventArgs␣e)
{
␣␣␣␣OpenFileDialog␣openFileDialog1␣=␣new␣OpenFileDialog();
}
```

Embora seja suportado pelo HTML5, o controlo *Upload* no KendoUI é bastante melhor. A migração deste controlo passa por substituir o *Button* em Silverlight por um controlo *Input* do tipo *File*. Porém, isso deve estar inserido numa *form*, inicializada com *KendoUpload*, como mostra o exemplo 5.71.

Exemplo 5.71: Implementação de um *OpenFileDialog* em HTML5.

```
<form method="post" data-bind="kendoUpload:␣{}">
    <div>
        <input name="files[]" type="file" />
    </div>
</form>
```

5.2.2.27 *ChildWindow, Popup*

Os controlos *ChildWindow* ou *Popup* em Silverlight são suportados por um único controlo em KendoUI: *Window*. A migração deste controlo passa por renomear as *tags sdk:ChildWindow* ou *popup* por *div* e posteriormente inicializá-la com *KendoWindow*.

Exemplo 5.72: Implementação de um *ChildWindow* em Silverlight.

```
<sdk:ChildWindow>
    // ...
</sdk:ChildWindow>
```

No exemplo 5.73 está apresentada a migração para HTML5 do exemplo *exSilverlightChildWindow* em Silverlight.

Exemplo 5.73: Implementação de um *ChildWindow* em HTML5.

```
<div data-bind="kendoWindow:␣//...">
```

```
    Window Content
</div>
```

5.2.2.28 *RepeatButton*, *DrawingSurface*, *MultiScaleImage*, *ViewBox*, *Frame*, *Page*

Os controlos *DrawingSurface*, *MultiScaleImage*, *ViewBox*, *Frame* e *Page* não são suportados nativamente em HTML5 nem suportados pelo KendoUI. Alguns destes controlos poderão ser simulados consoante a sua necessidade.

5.2.3 *ViewModel*

Uma vez concluída a migração das *Views*, é necessário terminar os *ViewModels*, embora ainda seja necessário voltar às primeiras para adicionar os *Bindings* correspondentes.

Como tal, nesta subsecção será abordado o processo de migração dos *bindings*, atribuindo as funcionalidades que o KnockoutJS oferece ao juntar com tudo o que foi feito anteriormente.

Deste modo, é necessário criar um *ViewModel*, declarando novamente a estrutura inicial do RequireJS com todas as dependências, inicializando depois a função com o nome do *ViewModel*, como mostra o exemplo 5.74.

Exemplo 5.74: Implementação de um *ViewModel* em HTML5.

```
define([jquery, ...], function ($, ...) {

    function ExampleViewModel () {
        var self = this;
    };

    return ExampleViewModel;
});
```

Em KnockoutJS temos o *observable* e o *observableArray*, em que o primeiro permite fazer *binding* a um único objeto e o segundo a uma coleção de objetos.

Em Silverlight, o modo de fazer *binding* a um objeto na *view* é declarando entre chavetas o código *Binding*, seguido do nome da propriedade, conforme está especificado no exemplo 5.75.

Exemplo 5.75: Declaração do *Binding* na *view* em Silverlight.

```
<TextBox Text="{Binding␣FirstName,␣Mode=OneWay}" />
```

No exemplo 5.76 está apresentada a propriedade que está a fazer *Binding* da *view*, sendo enviada uma notificação quando esta for alterada, de forma a que a *view* atualize o valor.

Exemplo 5.76: Implementação do *Binding* em Silverlight.

```
private string _FirstName;
public string FirstName
{
    get { return _FirstName; }
    set
    {
        _FirstName = value;
        NotifyPropertyChanged("FirstName");
    }
}
```

A migração para HTML5 é muito mais simples, uma vez que o *binding* na *view* apenas necessita de adicionar o atributo *data-bind*, definir o tipo de *binding*, seguido da propriedade. O exemplo 5.77 mostra a migração para HTML5 do exemplo 5.75 em Silverlight.

Exemplo 5.77: Declaração do *Binding* na *View* em HTML5.

```
<span data-bind="text: FirstName"></span>
```

A migração da implementação é muito mais compacta em JavaScript, usando apenas uma única linha de código, e passando o nome da propriedade seguido de *ko.observable()*, como mostra no exemplo 5.78.

Exemplo 5.78: Implementação do *Binding* no *ViewModel* em HTML5.

```
self.FirstName: ko.observable() // Initially blank
```

Uma vez que os *Bindings* na parte do *ViewModel* são sempre declarados da mesma forma, apenas é necessário focar nos tipos de *binding* suportados pelo KnockoutJS. A nível de controlo de texto e aparência só existem seis tipos: *text*, HTML, CSS, *style*, *visible* e *attr*.

Os restantes *bindings* poderão igualmente ser utilizados, uma vez que permitem trabalhar com campos de formulário. Estes são o *click*, o *events*, o *submit*, o *enable*, o *disable*, o *value*, o *textInput*, o *hasFocus*, o *checked*, o *options* e o *selectedOptions*.

6 Resultados da migração

Neste capítulo são apresentados os resultados do desenvolvimento da migração do caso de estudo para HTML5.

6.1 Resultados HTML5

Com base no modelo de migração definido, o resultado final obtido foi satisfatório, uma vez que toda a aplicação foi migrada com sucesso, como mostra a figura 6.1. No entanto, dos cerca de cinquenta controlos em Silverlight, apenas foi necessário proceder à migração daqueles que se encontravam presentes no caso de estudo, nomeadamente o *StackPanel*, o *Grid*, o *Border*, o *Button*, o *Canvas*, o *ListView*, o *ComboBox*, o *TextBlock*, o *RichTextBlock*, o *Calendar*, o *DatePicker*, o *CheckBox*, o *RadioButton*, o *TextBox*, o *Image*, o *Label* e o *ToolTip*.

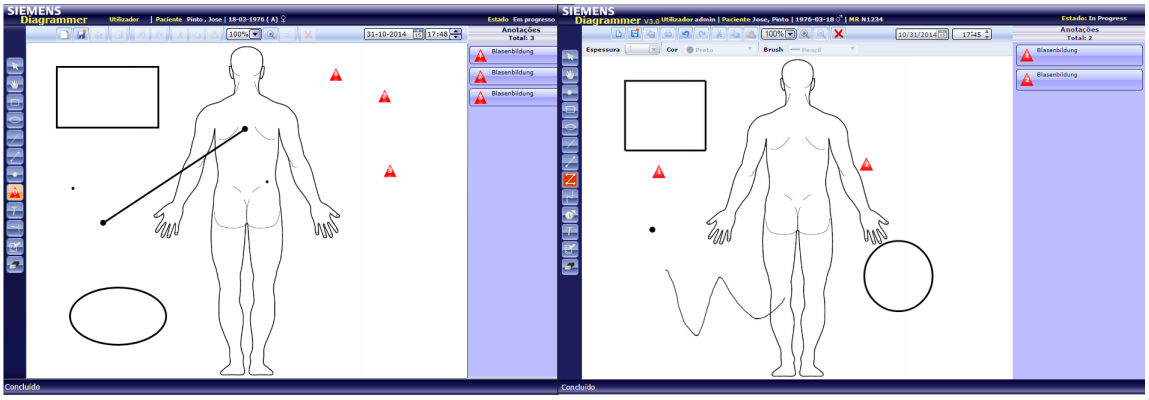


Figura 6.1: Diagrammer em Silverlight à esquerda e versão em HTML5 à direita

Os controlos foram migrados facilmente, tendo como base o modelo de migração abordado na secção 5.2. No entanto, era necessário manter todo o estilo da aplicação em Silverlight após a sua migração para HTML5, o que se revelou uma tarefa bastante demorada, pois não existia um processo definido para migrar estilos. Apesar disso, o resultado final ficou muito parecido com o original.

A biblioteca Knockout-Kendo, que permitia fazer a ligação entre os controlos do KendoUI com *bindings* da Knockout, foi utilizada quando ainda não suportava todos os controlos nem as funcionalidades oferecidos pelo KendoUI, pelo que se tornou um entrave, obrigando a desenvolver algum trabalho extra com *templating* e *binding*.

De forma geral, a migração ocorreu como esperado, mas à medida que a aplicação ia sendo testada, notou-se uma grande perda de performance na *ListView*, quando esta

tinha que adicionar/remover/alterar algum objeto da lista. Este problema foi originado no KnockoutJS, pois este fazia *binding* à lista toda e não a cada uma das linhas, pelo que, de cada vez que um dos elementos era alterado, o controlador do KendoUI era obrigado a renderizar toda a lista. Para contornar este problema, aplicou-se o *binding* à lista e a cada elemento da lista, resolvendo desta forma as questões de performance.

A biblioteca FabricJS também originou muito trabalho extra, uma vez que foi necessário alterar o comportamento nativo de algumas das funcionalidades, acrescentar outras e corrigir alguns *bugs*, porém esta solução revelou-se mais eficaz do que ter que implementar todas as funcionalidades de raiz.

Outro dos grandes problemas durante o desenvolvimento foi garantir que o DOM não bloqueava, uma vez que, em funções com ciclos longos, é propício haver uma degradação muito elevada da performance da aplicação. Para isso, recorreu-se aos *web workers*, no entanto em *browsers* mais antigos foi necessário simular uma solução parecida, ou otimizar a função ou ciclo, em vez de usar bibliotecas que o fizessem automaticamente.

7 Conclusões e Trabalho Futuro

Uma vez concluído este trabalho de investigação, posso afirmar que estou bastante satisfeito com os resultados, pois consegui alcançar todos os objetivos propostos de forma positiva. A aplicação migrada correspondeu às expectativas, tendo ainda sido possível adicionar novas funcionalidades. Além disso, reconheço que este último ano de Mestrado foi intenso, mas ao mesmo tempo fez-me crescer muito, tanto profissional como pessoalmente. A experiência adquirida na Siemens será de enorme valor no meu futuro profissional, e os conhecimentos adquiridos com o desenvolvimento deste trabalho deixaram-me muito melhor preparado para enfrentar os projetos que possam advir.

Durante a investigação de ambas as tecnologias e o desenvolvimento do caso prático, fiquei reticente se realmente a migração se ia dar de forma pacífica, uma vez que o Silverlight mantinha uma base de controlos bastante alargada e uma performance muito boa na aplicação. A escolha do conjunto de bibliotecas que seriam utilizadas para proceder à migração do caso de estudo, e que foram imprescindíveis para alcançar os objetivos, facilitou imensamente a implementação de cada um dos controlos. No entanto, à medida que o projeto foi crescendo, e com a adição de mais funcionalidades e controladores, deparei-me com a redução substancial da performance, dificuldade esta que só pôde ser contornada com realização de alguns ajustes customizados.

Quanto à questão relativa à necessidade da migração de uma aplicação desenvolvida em Silverlight para HTML5, a resposta ainda não é consensual entre os *developers* e empresas. Do meu ponto de vista, a migração de uma aplicação está dependente não só da empresa, mas também do *target* de cada aplicação, nomeadamente se esta é para ser utilizada em equipamentos móveis, se terá que ser mantida ou se será para introduzir novas funcionalidades no futuro, a necessidade de ter uma boa performance, se a empresa detém recursos no momento e se o investimento terá retorno, entre outras questões.

Este trabalho permitiu ainda partilhar algum conhecimento e correção de *bugs* na biblioteca FabricJS, auxiliando a sua evolução no repositório do GitHub.

Como trabalho futuro, seria proveitoso investigar ou desenvolver uma ferramenta que auxiliasse numa migração parcialmente automatizada. Enquanto isso não acontece, esta tese, com toda a sua investigação subjacente, bem como a indicação das bibliotecas utilizadas e os mapeamentos dos controlos abordados, poderá ser um bom auxiliar para migrações futuras, podendo ainda ser dada continuidade ao caso de estudo abordado. Porém, há que ter em conta que, caso os projetos utilizem bibliotecas auxiliares em Silverlight, será sempre necessário adaptar a migração para os novos controlos associados.

Referências

[1] Tim Berners-Lee. *First mention of HTML Tags on the www-talk mailing list*. 1991. URL: http://lists.w3.org/Archives/Public/www-talk/1991SepOct/0003.html (acedido em 11/01/2014) (ver p. 5).

[2] Tim Berners-Lee. *HTML Tags*. 1992. URL: http://info.cern.ch/hypertext/WWW/MarkUp/Tags.html (acedido em 11/01/2014) (ver p. 5).

[3] Jeremy Keith. *HTML5 for Web Designers*. First. A Book Apart, 2010. Cap. 1 A Brief History of Markup (ver p. 5).

[4] Mark Pilgrim. *HTML5 Up and Running*. First. O'Reilly Media, 2010. Cap. 1 How Did We Get Here? (Ver p. 5).

[5] Dave Raggett et al. *Ragget on HTML 4*. First. Addison Wesley, 1998. Cap. 2 A history of HTML (ver p. 5).

[6] T. Berners-Lee. *Hypertext Markup Language - 2.0*. 1995. URL: http://www.ietf.org/rfc/rfc1866.txt (acedido em 11/01/2014) (ver p. 5).

[7] D. Raggett. *HTML 3.2 Reference Specification*. 1997. URL: http://www.w3.org/TR/REC-html32 (acedido em 11/01/2014) (ver p. 5).

[8] D. Raggett, A. Le Hors e I. Jacobs. *HTML 4.0 Reference Specification*. 1998. URL: http://www.w3.org/TR/1998/REC-html40-19980424/ (acedido em 11/01/2014) (ver p. 5).

[9] D. Raggett, A. Le Hors e I. Jacobs. *HTML 4.1 Reference Specification*. 1999. URL: http://www.w3.org/TR/1999/REC-html401-19991224/ (acedido em 11/01/2014) (ver p. 5).

[10] Simon Sarris. *HTML5 UNLEASHED*. First. Sams Publishing, 2013. Cap. 1 How Did We End Up Here?, pp. 7, 8 (ver pp. 5, 7, 8).

[11] Simon Sarris. *HTML5 UNLEASHED*. First. Sams Publishing, 2013. Cap. 1 How Did We End Up Here?, p. 9 (ver p. 5).

[12] Microsoft. URL: http://msdn.microsoft.com/en-us/library/cc838158(v=vs.95).aspx (acedido em 11/01/2014) (ver p. 6).

[13] Microsoft. URL: http://msdn.microsoft.com/en-us/library/bb404700(v=vs.95).aspx (acedido em 11/01/2014) (ver pp. 6, 7).

[14] Devin Rader et al. *Silverlight 1.0*. First. Sams Publishing, 2013. Cap. 1 How Did We End Up Here?, pp. 7, 8 (ver p. 6).

[15] Laurence Moroney. *Introducing Microsoft Silverlight 2.0*. Second. Microsoft Press, 2008. Cap. 1 Introducing Silverlight 2 (ver p. 6).

[16] Microsoft. *Microsoft Silverlight Release History*. URL: http://www.microsoft.com/getsilverlight/locale/en-us/html/Microsoft%20Silverlight%20Release%20History.htm (acedido em 11/01/2014) (ver p. 6).

[17] Robert Lair. *Beginning Silverlight 5 in C#*. Fourth. Apress, 2012. Cap. 1 Welcome to Silverlight 5 (ver pp. 6, 7).

[18] Microsoft. *Compatible Operating Systems and Browsers*. URL: http://www.microsoft.com/getsilverlight/locale/en-us/html/installation-win-SL5.html (acedido em 11/01/2014) (ver p. 6).

[19] Microsoft. *Microsoft Support Lifecycle*. URL: http://support.microsoft.com/lifecycle/?LN=en-us&c2=12905 (acedido em 11/01/2014) (ver p. 7).

[20] Microsoft. *Educationalist Chooses Microsofts Public Private Cloud as its Platform for Personalized Learning*. URL: http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=710000003418 (acedido em 11/01/2014) (ver p. 7).

[21] Microsoft. *Educationalist Chooses Microsofts Public Private Cloud as its Platform for Personalized Learning*. URL: http://www.microsoft.com/casestudies/Bing-Maps/Kinetic/Media-Agency-Delivers-Competitive-Client-Service-with-Integrated-Mapping-Tool/710000002338 (acedido em 11/01/2014) (ver p. 7).

[22] World Wide Web Consortium. *HTML5 W3C Proposed Recommendation 16 September 2014*. URL: http://www.w3.org/TR/2014/PR-html5-20140916/ (acedido em 23/10/2014) (ver pp. 8, 9).

[23] Simon Sarris. *HTML5 UNLEASHED*. First. Sams Publishing, 2013. Cap. 1 How Did We End Up Here?, pp. 10, 11, 12 (ver p. 8).

[24] Sergey Mavrody. *Sergey's HTML5 & CSS3 Quick Reference*. Third. Belisso, 2012 (ver p. 8).

[25] World Wide Web Consortium. *HTML5 W3C Working Draft 22 January 2008*. URL: http://www.w3.org/TR/2008/WD-html5-20080122/ (acedido em 11/01/2014) (ver p. 9).

[26] Ian Hickson. *Update on the relationship between the WHATWG HTML living standard and the W3C HTML5 specification*. URL: http://lists.w3.org/Archives/Public/public-whatwg-archive/2012Jul/0119.html (acedido em 11/01/2014) (ver p. 9).

[27] Simon Sarris. *HTML5 UNLEASHED*. First. Sams Publishing, 2013. Cap. 2 Important Concepts for HTML5, pp. 15, 16 (ver p. 9).

[28] Bruce Lawson e Remy Sharp. *Introducing HTML5*. Second. New Riders, 2011 (ver p. 9).

[29] Appcelerator. «Native vs. HTML5 Mobile App Development». Em: (2011). URL: http://www.appcelerator.com.s3.amazonaws.com/pdf/appcelerator-whitepaper-native-html5.pdf (acedido em 11/01/2014) (ver pp. 9, 10).

[30] Adrian W. West. *Practical HTML5 Projects*. First. Apress, 2012. Cap. 16 Search Engine Optimization (ver p. 10).

[31] Microsoft. *Santa Tracker Moves to New Service, Gives Real-Time Insight to More Than 22 Million*. URL: http://www.microsoft.com/casestudies/Case_Study_Detail.aspx?casestudyid=710000003227 (acedido em 11/01/2014) (ver p. 10).

[32] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/syntax.html (acedido em 21/06/2014) (ver p. 13).

[33] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/html.html (acedido em 21/06/2014) (ver p. 13).

[34] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/head.html (acedido em 21/06/2014) (ver p. 14).

[35] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/title.html (acedido em 21/06/2014) (ver p. 14).

[36] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/base.html (acedido em 21/06/2014) (ver p. 14).

[37] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/link.html (acedido em 21/06/2014) (ver p. 14).

[38] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/meta.html (acedido em 21/06/2014) (ver p. 14).

[39] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/style.html (acedido em 21/06/2014) (ver p. 14).

[40] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/script.html (acedido em 21/06/2014) (ver p. 14).

[41] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/noscript.html (acedido em 21/06/2014) (ver p. 14).

[42] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/body.html (acedido em 21/06/2014) (ver p. 14).

[43] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/section.html (acedido em 21/06/2014) (ver p. 14).

[44] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/nav.html (acedido em 21/06/2014) (ver p. 14).

[45] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/article.html (acedido em 21/06/2014) (ver p. 15).

[46] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/aside.html (acedido em 21/06/2014) (ver p. 15).

[47] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/h1.html (acedido em 21/06/2014) (ver p. 15).

[48] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/h2.html (acedido em 21/06/2014) (ver p. 15).

[49] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/h3.html (acedido em 21/06/2014) (ver p. 15).

[50] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/h4.html (acedido em 21/06/2014) (ver p. 15).

[51] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/h5.html (acedido em 21/06/2014) (ver p. 15).

[52] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/h6.html (acedido em 21/06/2014) (ver p. 15).

[53] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/hgroup.html (acedido em 21/06/2014) (ver p. 15).

[54] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/header.html (acedido em 21/06/2014) (ver p. 15).

[55] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/footer.html (acedido em 21/06/2014) (ver p. 15).

[56] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/address.html (acedido em 21/06/2014) (ver p. 15).

[57] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/p.html (acedido em 21/06/2014) (ver p. 15).

[58] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/hr.html (acedido em 21/06/2014) (ver p. 15).

[59] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/br.html (acedido em 21/06/2014) (ver p. 15).

[60] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/pre.html (acedido em 21/06/2014) (ver p. 15).

[61] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/blockquote.html (acedido em 21/06/2014) (ver p. 15).

[62] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/ol.html (acedido em 21/06/2014) (ver p. 15).

[63] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/ul.html (acedido em 21/06/2014) (ver p. 15).

[64] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/li.html (acedido em 21/06/2014) (ver p. 15).

[65] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/dl.html (acedido em 21/06/2014) (ver p. 15).

[66] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/dt.html (acedido em 21/06/2014) (ver p. 15).

[67] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/dd.html (acedido em 21/06/2014) (ver p. 15).

[68] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/figure.html (acedido em 21/06/2014) (ver p. 15).

[69] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/figcaption.html (acedido em 21/06/2014) (ver p. 15).

[70] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/div.html (acedido em 21/06/2014) (ver p. 15).

[71] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/a.html (acedido em 21/06/2014) (ver p. 16).

[72] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/em.html (acedido em 21/06/2014) (ver p. 16).

[73] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/strong.html (acedido em 21/06/2014) (ver p. 16).

[74] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/small.html (acedido em 21/06/2014) (ver p. 16).

[75] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/s.html (acedido em 21/06/2014) (ver p. 16).

[76] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/cite.html (acedido em 21/06/2014) (ver p. 16).

[77] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/q.html (acedido em 21/06/2014) (ver p. 16).

[78] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/dfn.html (acedido em 21/06/2014) (ver p. 16).

[79] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/abbr.html (acedido em 21/06/2014) (ver p. 16).

[80] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/time.html (acedido em 21/06/2014) (ver p. 16).

[81] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/code.html (acedido em 21/06/2014) (ver p. 16).

[82] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/var.html (acedido em 21/06/2014) (ver p. 16).

[83] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/samp.html (acedido em 21/06/2014) (ver p. 16).

[84] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/kbd.html (acedido em 21/06/2014) (ver p. 16).

[85] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/sub.html (acedido em 21/06/2014) (ver p. 16).

[86] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/sup.html (acedido em 21/06/2014) (ver p. 16).

[87] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/i.html (acedido em 21/06/2014) (ver p. 16).

[88] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/b.html (acedido em 21/06/2014) (ver p. 16).

[89] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/u.html (acedido em 21/06/2014) (ver p. 16).

[90] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/mark.html (acedido em 21/06/2014) (ver p. 16).

[91] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/ruby.html (acedido em 21/06/2014) (ver p. 16).

[92] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/rt.html (acedido em 21/06/2014) (ver p. 16).

[93] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/rp.html (acedido em 21/06/2014) (ver p. 16).

[94] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/bdi.html (acedido em 21/06/2014) (ver p. 16).

[95] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/bdo.html (acedido em 21/06/2014) (ver p. 16).

[96] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/span.html (acedido em 21/06/2014) (ver p. 16).

[97] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/ins.html (acedido em 21/06/2014) (ver p. 16).

[98] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/del.html (acedido em 21/06/2014) (ver p. 17).

[99] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/img.html (acedido em 21/06/2014) (ver p. 17).

[100] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/iframe.html (acedido em 21/06/2014) (ver p. 17).

[101] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/embed.html (acedido em 21/06/2014) (ver p. 17).

[102] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/object.html (acedido em 21/06/2014) (ver p. 17).

[103] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/param.html (acedido em 21/06/2014) (ver p. 17).

[104] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/video.html (acedido em 21/06/2014) (ver pp. 17, 27).

[105] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/audio.html (acedido em 21/06/2014) (ver p. 17).

[106] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/source.html (acedido em 21/06/2014) (ver p. 17).

[107] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/track.html (acedido em 21/06/2014) (ver p. 17).

[108] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/canvas.html (acedido em 21/06/2014) (ver p. 17).

[109] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/map.html (acedido em 21/06/2014) (ver p. 17).

[110] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/area.html (acedido em 21/06/2014) (ver p. 17).

[111] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/table.html (acedido em 21/06/2014) (ver p. 17).

[112] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/caption.html (acedido em 21/06/2014) (ver p. 17).

[113] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/colgroup.html (acedido em 21/06/2014) (ver p. 17).

[114] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/col.html (acedido em 21/06/2014) (ver p. 17).

[115] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/tbody.html (acedido em 21/06/2014) (ver p. 17).

[116] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/thead.html (acedido em 21/06/2014) (ver p. 17).

[117] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/tfoot.html (acedido em 21/06/2014) (ver p. 17).

[118] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/tr.html (acedido em 21/06/2014) (ver p. 18).

[119] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/td.html (acedido em 21/06/2014) (ver p. 18).

[120] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/th.html (acedido em 21/06/2014) (ver p. 18).

[121] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/form.html (acedido em 21/06/2014) (ver p. 18).

[122] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/fieldset.html (acedido em 21/06/2014) (ver p. 18).

[123] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/legend.html (acedido em 21/06/2014) (ver p. 18).

[124] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/label.html (acedido em 21/06/2014) (ver p. 18).

[125] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/input.html (acedido em 21/06/2014) (ver p. 18).

[126] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/button.html (acedido em 21/06/2014) (ver p. 18).

[127] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/select.html (acedido em 21/06/2014) (ver p. 18).

[128] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/datalist.html (acedido em 21/06/2014) (ver p. 18).

[129] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/optgroup.html (acedido em 21/06/2014) (ver p. 18).

[130] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/option.html (acedido em 21/06/2014) (ver p. 18).

[131] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/textarea.html (acedido em 21/06/2014) (ver p. 18).

[132] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/keygen.html (acedido em 21/06/2014) (ver p. 18).

[133] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/output.html (acedido em 21/06/2014) (ver p. 18).

[134] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/progress.html (acedido em 21/06/2014) (ver p. 18).

[135] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/meter.html (acedido em 21/06/2014) (ver p. 18).

[136] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/details.html (acedido em 21/06/2014) (ver p. 19).

[137] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/summary.html (acedido em 21/06/2014) (ver p. 19).

[138] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/command.html (acedido em 21/06/2014) (ver p. 19).

[139] W3. *HTML: The Markup Language (an HTML language reference)*. URL: http://www.w3.org/TR/html-markup/menu.html (acedido em 21/06/2014) (ver p. 19).

[140] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=html5semantic (acedido em 21/07/2014) (ver p. 23).

[141] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=form-validation (acedido em 21/07/2014) (ver p. 24).

[142] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=canvas (acedido em 21/07/2014) (ver p. 26).

[143] W3 Schools. *HTML5 Audio*. URL: http://www.w3schools.com/html/html5_audio.asp (acedido em 21/06/2014) (ver p. 27).

[144] W3 Schools. *HTML5 Vdieo*. URL: http://www.w3schools.com/html/html5_video.asp (acedido em 21/06/2014) (ver p. 27).

[145] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/ (acedido em 11/08/2014) (ver p. 27).

[146] HTML5TEST. *HTML5TEST: how well does your browser support html5?* URL: http://html5test.com/ (acedido em 11/08/2014) (ver p. 27).

[147] W3 Schools. *HTML5 Vdieo*. URL: http://www.w3.org/TR/html-markup/source.html (acedido em 11/08/2014) (ver p. 28).

[148] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=dragndrop (acedido em 11/08/2014) (ver p. 29).

[149] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=namevalue-storage (acedido em 11/08/2014) (ver p. 31).

[150] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=offline-apps (acedido em 11/08/2014) (ver p. 33).

[151] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=geolocation (acedido em 11/08/2014) (ver p. 35).

[152] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=websockets (acedido em 11/08/2014) (ver p. 37).

[153] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=webworkers (acedido em 11/08/2014) (ver p. 38).

[154] Can I use. *Can I use: New semantic elements*. URL: http://caniuse.com/#feat=eventsource (acedido em 11/08/2014) (ver p. 40).

[155] Nicholas Zakas. *Maintainable JavaScript*. First. O'Reilly Media, 2012 (ver p. 57).