

Mestrado em Engenharia Informática  
Estágio  
Relatório Final

# Appbase – Plataforma Interoperável para Serviços Web Centrados no Utilizador

David Marquês Francisco  
dmfranc@student.dei.uc.pt

*Orientadores:*

Professor Doutor Ernesto Costa (DEI)  
Eng. Alcides Marques (IPN)

12 de Julho de 2012



**FCTUC** DEPARTAMENTO  
DE ENGENHARIA INFORMÁTICA  
FACULDADE DE CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE COIMBRA



## Histórico de revisões

Data	Alteração	Versão
18/10/2011	Escrita do capítulo “Estado da Arte”.	0.1.0
22/10/2011	Escrita do capítulo “Introdução”.	0.2.0
23/10/2011	Adição de análise sobre <i>frameworks JavaScript client-side</i> .	0.3.0
25/10/2011	Adição do “Planeamento” e da introdução sobre Scrum.	0.4.0
02/11/2011	Actualização do Planeamento (de acordo com as indicações dadas pelo Eng. Alcides Marques e Prof. Paulo Rupino).	0.5.0
03/11/2011	Secção para <i>backlogs</i> adicionada aos anexos.	0.6.0
16/11/2011	Primeiros conteúdos do capítulo “Arquitectura e Design”.	0.7.0
22/12/2011	Especificação dos Requisitos Funcionais.	0.8.0
25/12/2011	Conclusão do capítulo “Arquitectura e Design”.	0.9.0
26/12/2011	Escrita do anexo “Análise de Riscos”.	0.10.0
12/01/2012	Primeira versão do anexo “Artefactos da Metodologia de Desenvolvimento Ágil”.	0.11.0
13/01/2012	Escrita do subcapítulo “Processos de Engenharia”.	0.12.0
18/01/2012	Inclusão do diagrama <i>Gantt</i> com os desvios face ao plano inicial.	0.13.0
19/01/2012	Escrita do capítulo “Conclusões”.	0.14.0
19/01/2012	Escrita do capítulo “Plano de Testes”.	0.15.0
21/01/2012	Escrita do capítulo “Desenho da Interface Gráfica”.	0.16.0
27/03/2012	Escrita do subcapítulo “Modelo de dados”.	1.1.0
08/06/2012	Escrita da primeira versão do “Guia do programador”.	1.2.0
16/06/2012	Inclusão dos <i>Backlogs</i> , gráficos de <i>Burndown</i> e da calendarização efectiva do segundo semestre.	1.3.0
18/06/2012	Escrita da documentação das API's <i>JavaScript</i> no “Guia do programador”.	1.4.0
19/06/2012	Actualização do capítulo “Plano de Testes”.	1.5.0
20/06/2012	Adição da descrição sobre Ambientes e Requisitos Tecnológicos.	1.6.0
01/07/2012	Actualização do capítulo “Conclusão”.	1.7.0

A versão deste documento segue o formato **X.Y.Z** (Entrega.Adição.Revisão). A alteração de conteúdo existente implica o incremento da casa **Z** e não consta do histórico de revisões; a adição de uma secção impõe o incremento da casa **Y**; a entrega do documento para avaliação (fase intermédia e final) resulta no incremento da casa **X**. A versão actual deste documento é **2.0.0**.





## Resumo

Um conjunto de entidades empresariais e instituições de *I&DT* tem como missão potenciar a presença das empresas portuguesas nos mercados das áreas estratégicas Mobilidade e Saúde. A sua concretização passa pelo desenvolvimento de serviços inovadores e de uma plataforma que possibilite a disponibilização dos mesmos [1].

Neste estágio construiu-se esse canal de promoção e comercialização de soluções de Mobilidade e Saúde, sob a forma de aplicações Web. Este canal desempenha um papel simultaneamente agregador, uma vez que é através deste que se processa a descoberta das aplicações, e integrador, pois funcionam em torno do mesmo contexto e utilizam funcionalidades comuns. Cada utilizador pode associar ao seu perfil aplicações existentes na plataforma. Esta encarrega-se de partilhar o contexto específico dessa conta e sessão pelas várias aplicações associadas, de forma a que cada acção realizada pelo utilizador possa surtir efeitos nas aplicações que usa. Colocou-se assim o desafio de integrar aplicações Web, cada uma com as suas especificidades, linguagens e tecnologias (e fisicamente dispersas por servidores remotos) num só local, garantindo sempre que para o utilizador final a experiência seja transparente e singular, ao longo da utilização da plataforma e das aplicações nela disponibilizadas.

Este documento pretende dar a conhecer o trabalho realizado pelo aluno David Marquês Francisco no âmbito da disciplina “Dissertação / Estágio” do Mestrado em Engenharia Informática da FCTUC, no ano lectivo 2011/2012.

**Palavras-chave** Aplicações Web, HTML5, Integração de Conteúdos Third-Party, Marketplace, Mashup, Mobilidade, Portal, Saúde, TICE, Web Services, Web Store, Widgets.

## **Agradecimentos**

Agradeço a todos os intervenientes na minha formação académica, em especial ao meu professor e orientador, Professor Doutor Ernesto Costa, e ao meu orientador no Instituto Pedro Nunes, o Engenheiro Alcides Marques. Gostaria de agradecer também ao Engenheiro Miguel Machado, ao Engenheiro Carlos Lopes e ao Professor Doutor Paulo Rupino que sempre estiveram disponíveis para me ajudar e que intensificaram o meu gosto pelo que faço. Agradeço ainda ao Professor Doutor Carlos Bento, ao Mestre João Quintas e ao Engenheiro Gouveia Leal pelo acompanhamento dado ao longo do projecto. Por último, mas não menos importante, agradeço à minha família, à minha namorada e aos meus colegas de equipa Diogo, Vitorino, Ferrão, Diana, Catré, David e Rebelo.

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Planeamento</b>	<b>5</b>
2.1	Metodologia de Desenvolvimento . . . . .	5
2.2	Calendário do Plano de Trabalho . . . . .	7
2.3	Outros Processos de Engenharia . . . . .	12
2.3.1	Estruturação dos repositórios de código . . . . .	12
2.3.2	Caracterização dos ambientes . . . . .	13
<b>3</b>	<b>Estado da Arte</b>	<b>15</b>
3.1	Estudo das capacidades dos <i>Web Portals</i> . . . . .	15
3.1.1	Descoberta de aplicações web . . . . .	16
3.1.2	Promoção e venda de aplicações web . . . . .	18
3.1.3	Registo das aplicações . . . . .	19
3.1.4	Utilização da aplicação através do portal . . . . .	20
3.2	Especificações relevantes . . . . .	20
3.2.1	Java Portlet Specification 2.0 . . . . .	22
3.2.2	Microsoft WebParts . . . . .	22
3.2.3	W3C Widgets . . . . .	22
3.2.4	Opera Widgets . . . . .	23
3.2.5	Google Gadgets . . . . .	23
3.2.6	Netvibes Universal Widgets . . . . .	24
3.2.7	Yahoo! Widgets, Microsoft Gadgets e Apple Dashboard Widgets	24
3.3	Tecnologias e ferramentas . . . . .	25
3.3.1	OpenSocial e Apache Shindig . . . . .	25
3.3.2	Apache Wookie . . . . .	25
3.3.3	Apache Rave . . . . .	25
3.4	Principais soluções comerciais e projectos de investigação . . . . .	26
3.4.1	Descrição geral . . . . .	26
3.4.2	Tabela comparativa . . . . .	29

<b>4</b>	<b>Análise de Requisitos</b>	<b>31</b>
4.1	Actores . . . . .	33
4.2	Requisitos funcionais . . . . .	33
4.3	Requisitos de sistema . . . . .	40
4.4	Requisitos de desempenho . . . . .	40
4.5	Requisitos de segurança . . . . .	41
4.6	Outras restrições técnicas . . . . .	41
4.7	Requisitos tecnológicos . . . . .	41
4.8	Atributos de Qualidade do Sistema . . . . .	42
4.8.1	Extensibilidade e Interoperabilidade . . . . .	42
4.8.2	Manutenabilidade . . . . .	42
4.8.3	Permutabilidade e Modularidade . . . . .	43
<b>5</b>	<b>Arquitectura e Design</b>	<b>45</b>
5.1	Problema de Engenharia . . . . .	45
5.2	Tipos de aplicações . . . . .	46
5.3	Vistas da Arquitectura . . . . .	47
5.3.1	Visão de nível 0 . . . . .	47
5.3.2	Visão de nível 1 . . . . .	50
5.4	Desenho da solução . . . . .	59
5.4.1	Estrutura do <i>back-end</i> . . . . .	59
5.4.2	Estrutura do <i>front-end</i> . . . . .	60
5.4.3	Descrição dos restantes módulos . . . . .	62
5.4.4	Modelo de dados . . . . .	65
5.4.5	Estrutura do código produzido . . . . .	67
<b>6</b>	<b>Desenho da Interface Gráfica</b>	<b>71</b>
6.1	Protótipos de baixa fidelidade . . . . .	71
6.2	Ecrãs de alta fidelidade . . . . .	74
6.3	Implementação inicial do design . . . . .	78
<b>7</b>	<b>Plano de Testes</b>	<b>83</b>
7.1	Estratégia de Desenvolvimento . . . . .	83
7.2	Especificação e elaboração dos Testes . . . . .	84
7.2.1	Testes unitários, funcionais e de integração . . . . .	84
7.2.2	Testes de aceitação pelo utilizador . . . . .	87
7.2.3	Testes de usabilidade . . . . .	88
<b>8</b>	<b>Conclusões</b>	<b>89</b>
	<b>Referências</b>	<b>93</b>

# Lista de Tabelas

3.1	Comparação das funcionalidades das várias soluções comerciais e projectos de investigação analisados. . . . .	30
5.1	Comparação entre aplicações <i>Hosted</i> e <i>Packaged</i> . . . . .	47
5.2	Resumo dos objectivos gerais de cada API <i>JavaScript</i> . . . . .	52
5.3	Vertentes do contentor de aplicações <i>Hosted</i> . . . . .	53
6.1	Protótipos de baixa fidelidade. . . . .	72
6.2	Protótipos de baixa fidelidade (continuação). . . . .	73

# Lista de Figuras

2.1	Representação da metodologia Scrum. Imagem adaptada [2]. . . . .	6
2.2	Estrutura dos repositórios de código. . . . .	13
3.1	Processo de obtenção de uma aplicação disponível numa <i>Web Application Store</i> . Imagem adaptada [3]. . . . .	16
3.2	Componentes usuais numa <i>Web Application Store</i> e num <i>Web Application Portal</i> . . . . .	17
3.3	A proliferação de <i>standards</i> . Fonte: <a href="http://xkcd.com/927/">http://xkcd.com/927/</a> . . . . .	20
3.4	Pilha de especificações relacionadas com <i>widgets</i> e aplicações web [4]. . . . .	21
3.5	Arquitectura de alto-nível do <i>Apache Shindig</i> . . . . .	25
3.6	Espaço ocupado por uma aplicação no Facebook. . . . .	26
3.7	Dois <i>gadgets</i> associados a um perfil do LinkedIn. . . . .	27
4.1	Notas escritas durante a definição do <i>scope</i> do estágio. . . . .	32

4.2	Actores do sistema. A seta representa a herança de actores (significando o mesmo que nos diagramas <i>UML</i> de casos de uso).	33
4.3	<i>Brainstorming</i> com a equipa sobre requisitos funcionais e de interface.	35
4.4	<i>Brainstorming</i> com o coordenador técnico sobre o fluxo de aprovação de aplicações.	35
4.5	Sessão para equipas técnicas do consórcio TICE.Mobilidade.	35
4.6	Apresentação das plataformas Appbase e One.Stop.Transport.	35
5.1	Visão de nível 0 da perspectiva de <i>deployment</i> da arquitectura.	49
5.2	Vista estática da arquitectura, com ligações a entidades externas e bases de dados. Como referido na vista física, as bases de dados têm um DBMS comum.	50
5.3	Visão de alto-nível da perspectiva estática da arquitectura.	51
5.4	Gestão de mudança de URL pela plataforma Appbase.	54
5.5	Execução de uma aplicação <i>packaged</i> ou <i>web layer</i> . Inspirado num diagrama de arquitectura do projecto Apache Wookie [5].	55
5.6	<i>Deploy</i> de uma aplicação <i>packaged</i> ou <i>web layer</i> .	55
5.7	Mecanismo de protecção de canais de comunicação privados.	56
5.8	Interacção entre actores no fluxo do OAuth 1.0.	57
5.9	Fluxo de aprovação de aplicações.	58
5.10	Visão de alto-nível da vista de desenvolvimento. Alguns sistemas presentes neste diagrama serão clarificados no subcapítulo “Descrição dos restantes módulos”.	59
5.11	Vista de desenvolvimento da solução, com divisão entre <i>back-end</i> e <i>front-end</i> .	64
5.12	Diagrama físico de base de dados da plataforma Appbase. A entidade <i>User</i> faz parte do módulo de autenticação da <i>framework</i> Django, enquanto que a entidade <i>Admin Log</i> faz parte da interface Django Admin. A amarelo estão representadas entidades e campos que não são actualmente utilizados na plataforma.	66
5.13	Entidades do módulo de autenticação da <i>framework</i> Django.	67
5.14	Entidades dos módulos de autorização ao acesso a API's através de chave e do protocolo OAuth 1.0.	68
5.15	Entidades do módulo de autorização ao acesso a API's através do protocolo OAuth 2.0.	68
5.16	Entidades do servidor de <i>widgets</i> Apache Wookie, responsável pela gestão de aplicações <i>packaged</i> .	69

6.1	Página <i>Home</i> de um utilizador autenticado. . . . .	74
6.2	Utilização de uma aplicação. . . . .	74
6.3	<i>Landing page</i> de uma aplicação. . . . .	75
6.4	Página <i>Home</i> de um utilizador autenticado. . . . .	75
6.5	Utilização de um <i>web layer</i> para cálculo de rotas. . . . .	76
6.6	Utilização de uma aplicação. . . . .	76
6.7	Visualização de todas as aplicações favoritas. . . . .	77
6.8	<i>Landing page</i> de uma aplicação. . . . .	77
6.9	Entrada do portal Appbase do TICE.Mobilidade, em inglês. . . . .	78
6.10	Entrada do portal Appbase do TICE.Healthy, em português. . . . .	79
6.11	Página de descoberta de aplicações. . . . .	80
6.12	Formulário de criação de uma aplicação do tipo <i>packaged</i> . . . . .	80
6.13	Algumas páginas do portal vistas num <i>smartphone</i> . As primeiras duas imagens mostram a descoberta de aplicações, enquanto a última apresenta o <i>web layer</i> de demonstração <i>Tweets Nearby</i> . . . . .	81
6.14	Guias de estilos para aplicações do projecto TICE.Mobilidade. . . . .	82
7.1	Processo a seguir na abordagem <i>Behavior Driven Development</i> . . . . .	84
7.2	<i>Screenshot</i> da execução da pilha de testes utilizando a ferramenta <i>Lettuce</i> (o <i>browser</i> é executado numa janela separada). . . . .	85
7.3	<i>Screenshot</i> do resultado da execução da pilha de testes utilizando o <i>Test Runner</i> da biblioteca <i>JUnit</i> . . . . .	86

# Lista de Acrónimos

**AJAX** Asynchronous JavaScript and XML

**API** Application Programming Interface

**BDD** Behavior Driven Development

**CSS** Cascading Style Sheets

**CTO** Chief Technology Officer

**DBMS** Database Management System

**DRY** Don't Repeat Yourself

**DOM** Document Object Model

**HTML** HyperText Markup Language

**HTTP** Hypertext Transfer Protocol

**IPN** Instituto Pedro Nunes

**I&DT** Investigação e Desenvolvimento Tecnológico

**JSON** JavaScript Object Notation

**LAS** Laboratório de Automática e Sistemas

**LIS** Laboratório de Informática e Sistemas

**MTV** Model-Template-View

**MVC** Model-View-Controller

**ORM** Object-Relational Mapping

**POI** Point Of Interest

**REST** Representational State Transfer



**SGBD** Sistema de Gestão de Base de Dados

**SOA** Service-Oriented Architecture

**SOAP** Simple Object Access Protocol

**SPA** Single Page Application

**TDD** Test Driven Development

**TICE** Tecnologias de Informação, Comunicação e Electrónica

**UI** User Interface

**UML** Unified Modeling Language

**URL** Uniform Resource Locator

**UTF-8** 8-bit Unicode Transformation Format

**UWA** Universal Widgets API

**W3C** World Wide Web Consortium

**WWW** World Wide Web

**XHTML** eXtensible Hypertext Markup Language

**XML** Extensible Markup Language



# Capítulo 1

## Introdução

### Enquadramento

Os projectos “TICE.Mobilidade – Sistema de Mobilidade Centrado no Utilizador” e “TICE.Healthy – Sistemas de Saúde e Qualidade de Vida” são dois projectos âncora inseridos no Pólo de Competitividade e Tecnologia TICE.PT e resultam de um esforço conjunto entre 29 parceiros. Estes dispõem-se a criar um ecossistema de serviços para potenciar a presença das empresas portuguesas nos mercados das áreas estratégicas do TICE.PT designadas “Mobilidade” e “Saúde e Qualidade de Vida”. Esses serviços, disponibilizados na forma de aplicações web, são suportados por uma Arquitectura Baseada em Serviços (SOA). Neste contexto, o Instituto Pedro Nunes (IPN) é responsável por garantir a correcta implementação de vários sub-projectos, destacando-se destes a plataforma desenvolvida no âmbito deste estágio [1].

O sub-projecto Appbase consiste numa plataforma de gestão e disponibilização de aplicações web, as quais serão produzidas por parceiros (e pelo próprio IPN) inseridos nas áreas da Mobilidade e Saúde. Apesar desta plataforma ser comum aos projectos TICE.Mobilidade e TICE.Healthy, a existência de módulos específicos permite tirar partido da oferta de dados existentes em cada um desses contextos. No caso do projecto TICE.Mobilidade, os módulos de extensão da plataforma Appbase estão integrados com a componente *One.Stop.Transport* — um conjunto de *Web Services* que permite a aquisição, tratamento e análise de dados para serviços de mobilidade em ambiente urbano. A plataforma Appbase torna, assim, possível agregar e registar novas aplicações que tirem partido da oferta de dados existentes e que contribuam para aproximar o fornecedor de serviços ao cliente final. Por exemplo, o utilizador poderá ter no seu telemóvel um plano intermodal actualizado ao segundo, receber sugestões de mobilidade e segurança, verificar a disponibilidade de parques de estacionamento, ou mesmo enviar informações sobre comodidades da sua cidade, eventos e notícias, tudo através de aplicações independentes. Cada aplicação pode ser criada por uma entidade distinta e seguir um modelo de negócio próprio [6].

## Relevância e Interesse

Vivemos numa época em que a Web está a sofrer mudanças significativas, que alteram a forma como se desenvolve e utiliza o seu conteúdo [7]. Com estas mudanças surgem oportunidades de negócio, não só em torno das novas capacidades das tecnologias de desenvolvimento para Web, mas também dos dispositivos móveis cada vez mais capazes. Estas são áreas estratégicas para as empresas tecnológicas portuguesas e, de forma a criar um ecossistema favorável à aposta nestes novos mercados, foram criados os projectos TICE.Mobilidade e TICE.Healthy. Ambos são de extrema relevância para que as empresas possam acompanhar as novas tendências tecnológicas. Assim, um grupo de pequenas e médias empresas pode conjuntamente levar a cabo dois projectos de grande dimensão que melhorarão os serviços prestados à sociedade, nas áreas da Mobilidade, Saúde e Qualidade de Vida. Os novos dispositivos móveis (*smartphones* e *tablets*) possuem nos seus *web browsers* capacidades similares às associadas aos *browsers* dos computadores pessoais. Estes têm evoluído à medida que se constrói um conjunto de novas especificações genericamente designadas por HTML5 [8]. Novas capacidades como armazenamento *offline* de conteúdos e geolocalização permitem finalmente a criação de aplicações web. A Web começa a deixar de estar centrada em documentos e passa a estar centrada em pessoas. Estes projectos têm assim o potencial de resolver problemas comuns existentes na nossa sociedade de formas até agora impraticáveis.

A motivação por detrás da escolha deste estágio está relacionada, em parte, com a investigação que pode ser feita em torno das novas especificações para a Web. Existe uma comunidade cada vez maior de *developers* que procura resolver problemas antigos de formas inovadoras, através de implementações baseadas nestes *standards* abertos [7]. Há também a possibilidade de inovar no que diz respeito à construção de aplicações *user-centric*, ao invés de *document-centric*. Actualmente, as aplicações e *websites* que visitamos na Web não interagem entre si de forma transparente. As contas de utilizador, os seus perfis, preferências, actividades e informações estão restritas a um proprietário e a integração entre serviços nem sempre é possível. Cabe ao utilizador navegar pelos sítios que possuem conteúdo do seu interesse. Alguns passos foram dados nesta área mas ainda existe muito espaço para investigação. Permanecem ainda desafios de agregação e integração de dados de múltiplas fontes. Em Portugal, as fontes de *Open Data* (dados abertos que podem ser utilizados por qualquer cidadão de forma livre) começam a crescer mas ainda são poucos os serviços que destas tomam partido [9].

Existe ainda um interesse pessoal pelo desenvolvimento para a Web e pela criação e utilização de *API's*. O sistema de controlo de versões *Git*, o protocolo *OAuth*, e a metodologia de desenvolvimento *Scrum* mencionados na proposta de estágio são

igualmente, na opinião do estagiário, complementos importantes para o currículo de um Engenheiro Informático. Por fim, foi a primeira oportunidade de trabalho num contexto real, com uma equipa ampla e multidisciplinar.

## Objectivos

O âmbito deste estágio consiste no desenvolvimento da plataforma Appbase e dos módulos de extensão para o projecto TICE.Mobilidade. A plataforma disponibiliza interfaces avançadas de interacção com o utilizador, tirando partido da Web 2.0 e da elevada taxa de penetração de dispositivos móveis com acesso à Internet em Portugal. Tem um papel agregador, visto que a descoberta de aplicações é feita no portal, e integrador, pois estas funcionam em torno do mesmo contexto (no caso do projecto TICE.Mobilidade, um mapa) e utilizam funcionalidades comuns (ao nível da autenticação e autorização, dados e *API's*). Apresentam-se os principais objectivos do trabalho proposto ao estagiário:

- Registo e gestão de aplicações desenvolvidas por *third-parties*;
- Autenticação de utilizadores em aplicações *third-party*, através de um mecanismo de *single sign-on*;
- Autorização no acesso a dados dos utilizadores e a outros recursos da plataforma por entidades externas;
- Desenvolvimento de solução para comunicação inter-aplicação e para partilha de contexto entre aplicações;
- Internacionalização e localização.

Durante o desenvolvimento foram ainda tidos em conta:

- Desenho e análise funcional em função de critérios de desempenho, disponibilidade e escalabilidade;
- Destaque ao nível da interacção com o utilizador;
- Forte componente de *Quality Assurance* com recurso a testes unitários, funcionais e de integração;
- Implementação de módulos e componentes segundo padrões de encapsulamento e *loose-coupling*.

Em suma, os projectos TICE têm o intuito de criar um ecossistema de serviços que potencie a presença das empresas portuguesas nos mercados das áreas estratégicas Mobilidade, Saúde e Qualidade de Vida. Por sua vez, pretendeu-se com o estágio resolver o problema de disponibilizar, gerir e utilizar aplicações web desenvolvidas por entidades distintas num só local.



# Capítulo 2

## Planeamento

Nine people can't make a baby in a month.

— *Fred Brooks, IBM, autor de “The Mythical Man-Month”*

Este capítulo apresenta a metodologia de desenvolvimento de software aplicada, o cronograma do trabalho realizado e outros processos seguidos para a gestão do projecto. Associados a este capítulo estão os anexos “Artefactos da Metodologia de Desenvolvimento Ágil”, que corresponde a uma análise detalhada das tarefas realizadas, e “Análise de Riscos”, onde os riscos identificados durante o projecto de estágio são avaliados, com indicação da sua ocorrência ou inoccorrência.

### 2.1 Metodologia de Desenvolvimento

O processo de desenvolvimento de software aplicado neste estágio é baseado na metodologia ágil Scrum, uma vez que o estágio está integrado nos projectos TICE.Mobilidade e TICE.Healthy e se segue nestes a mesma metodologia. A natureza ágil desta metodologia é vantajosa em projectos deste tipo, em que existe muita colaboração com parceiros externos, onde é expectável a ocorrência de constrangimentos e onde os requisitos são pouco estáveis.

O Scrum foi concebido com o intuito de permitir que projectos de software possam entregar incrementos de alto valor para o cliente de uma forma iterativa [10]. Tal é alcançado através de uma gestão mais eficiente dos requisitos, melhor comunicação entre os elementos da equipa e o cliente, e a descentralização da responsabilidade do gestor transferindo-a em parte para a equipa de projecto. No Scrum, os projectos progridem através de uma série de iterações designadas *Sprints*. O trabalho a

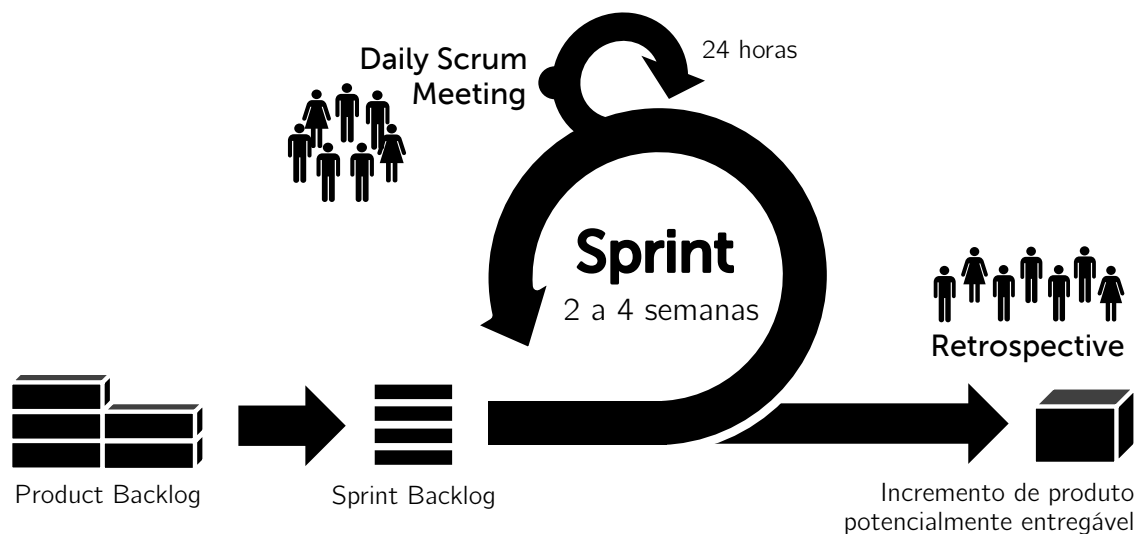


Figura 2.1: Representação da metodologia Scrum. Imagem adaptada [2].

desenvolver está listado no *Product Backlog*, uma lista de tarefas regularmente actualizada. No início de cada *Sprint*, o *Product Owner* prioriza o *Product Backlog* e a *Scrum Team* selecciona as tarefas a completar durante o próximo *Sprint*, no *Sprint Backlog*. É realizada diariamente uma breve reunião designada *Daily Scrum Meeting* [11]. A **figura 5.2** pretende ilustrar os conceitos referidos.

Foi mencionado que a metodologia empregue neste estágio é “baseada” em Scrum pelo facto de se seguirem as mesmas fases (no Scrum designadas *Pré-Game*, *Game* e *Pós-Game*), se produzirem os mesmos tipos de artefactos (*Product Backlog*, *Sprint Backlog* e gráficos *Burndown*), mas existirem alguns conceitos que foram adaptados à realidade do projecto. A título de exemplo, as *Sprint Retrospectives* aconteceram noutros moldes, de forma a coincidirem com as reuniões de coordenação que existem para os projectos.

Como referido, o plano de trabalhos do estágio foi dividido de acordo com as três fases do Scrum — *Pré-Game*, *Game* e *Pós-Game*. A fase *Pré-Game* não é subdividida em *Sprints*, não havendo dessa forma *backlogs* associados. É no entanto apresentada em anexo a lista de tarefas realizadas pelo estagiário durante cada um dos meses que constituem esta fase.

Durante a fase de *Game*, cada *Sprint* teve a duração de quatro semanas. O *Product Backlog* foi gerido através do software *web-based Redmine*, ao qual todos os elementos dos projectos TICE têm acesso. No contexto específico deste estágio, as tarefas a desempenhar pelo estagiário foram maioritariamente criadas pelo próprio, de acordo com o plano do estágio, e foi este o responsável por criar a sua porção do *Sprint Backlog*, específico do sub-projecto a que corresponde o estágio. Tal como no Scrum, é adicionada a cada tarefa uma estimativa (em horas) da sua duração e o



*Sprint Backlog* é actualizado ao longo dos 30 dias. São apresentados no anexo “Artefactos da Metodologia de Desenvolvimento Ágil” os *Sprint Backlogs* e respectivos gráficos *Burndown*.

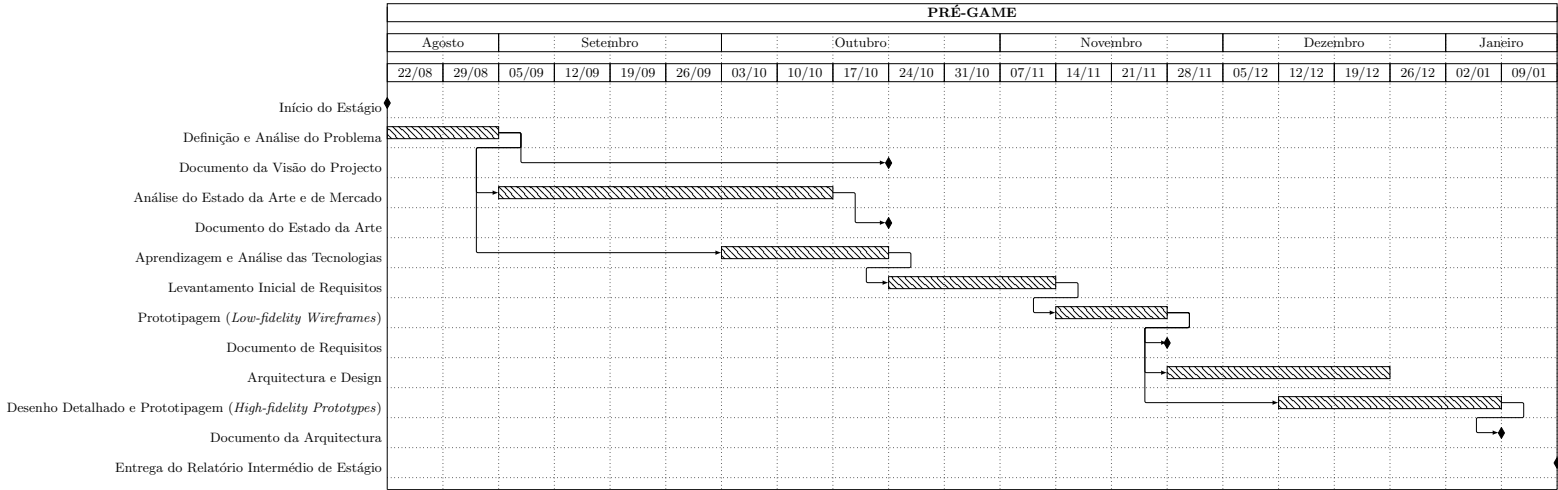
Tal como a fase *Pré-Game*, o *Pós-Game* não é subdividido em *Sprints*, não havendo por isso *backlogs* associados.

Assim como é comum no Scrum, existe uma equipa que desenvolve software de forma a resolver um problema comum. No entanto, todas as tarefas específicas da componente Appbase (o âmbito deste estágio) ficaram atribuídas ao estagiário (ou seja, existe uma *Scrum Team* mas foi mantida de forma clara a distinção entre o que foi realizado pela equipa e o que era da responsabilidade do estagiário). O *Scrum Master* é o Eng. Miguel Machado, coordenador técnico do projecto, e o *Product Owner* é o gestor dos projectos TICE, o Eng. Alcides Marques, orientador do estágio por parte do IPN. Foram feitas *Daily Meetings* entre a equipa de desenvolvimento com a participação do *Scrum Master*, durante as fases *Pré-Game* e *Game*.

## 2.2 Calendário do Plano de Trabalho

Segue-se o planeamento do estágio, representado através de dois diagramas *Gantt*. É também apresentada a calendarização do trabalho que efectivamente se desenrolou, com referência aos desvios ocorridos, durante os dois semestres.

1º Semestre — Pré-game (20 semanas)



**Início do Estágio** — 22 de Agosto de 2011

**Definição e Análise do Problema** — de 22 de Agosto até 4 de Setembro (2 semanas / 10%)

**Análise do Estado da Arte e de Mercado** — de 5 de Setembro até 16 de Outubro (6 semanas / 30%)

- Consiste na análise de *standards* e levantamento de tecnologias a usar no projecto, estudo de boas práticas e de soluções já existentes no mercado

**Aprendizagem e Análise das Tecnologias** — de 3 de Outubro até 23 de Outubro (3 semanas / 15%)

- Para além da aprendizagem e análise, inclui a configuração do ambiente, no que diz respeito à instalação da infra-estrutura de *software* necessária

**Levantamento Inicial de Requisitos** — de 24 de Outubro até 13 de Novembro (3 semanas / 15%)

- Consiste na análise de requisitos dos módulos a desenvolver
- Ter-se-á ainda em conta requisitos de desempenho, segurança, e outros não-funcionais

**Prototipagem (Low-fidelity Wireframes)** — de 14 de Novembro até 27 de Novembro (2 semanas / 10%)

- Criação de *mockups* que têm como objectivo substituir o documento de Especificação Funcional e obter *feedback* da parte dos parceiros do projecto

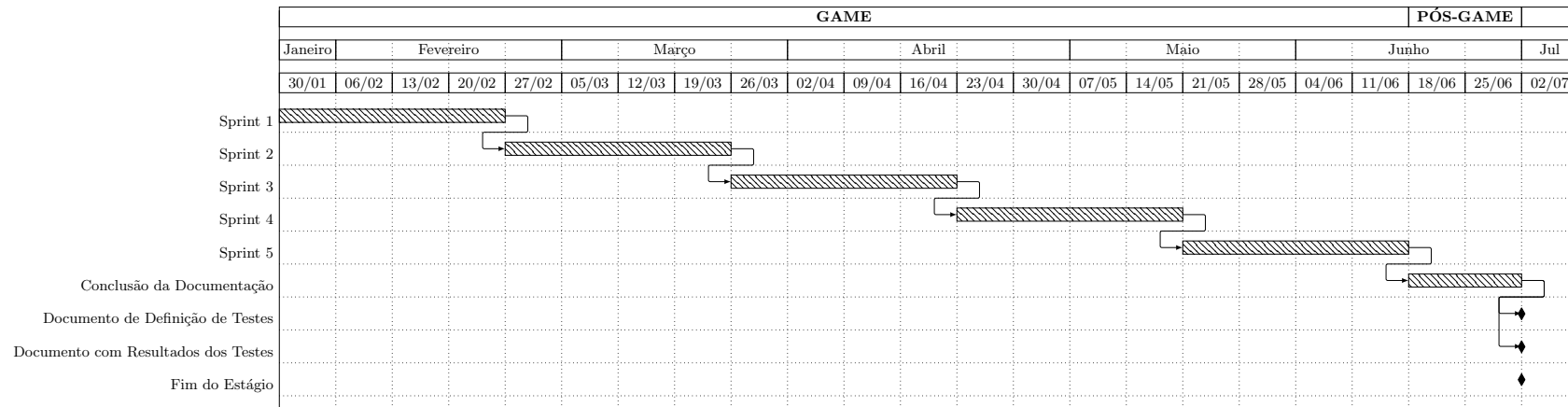
**Arquitectura e Design** — de 28 de Novembro até 25 de Dezembro (4 semanas / 20%)

- Desenho da arquitectura segundo várias perspectivas (física, estática, etc.) e concepção de um cenário de implementação
- Descrição da forma como se pretendem cumprir os atributos de qualidade

**Desenho Detalhado e Prototipagem (High-fidelity Prototypes)** — de 12 de Dezembro até 8 de Janeiro de 2012 (4 semanas / 20%)

**Entrega do Relatório Intermédio de Estágio** — 24 de Janeiro de 2012

## 2º Semestre — Game e Pós-game (22 semanas)



Todos os *sprints* seguem o ciclo de desenvolvimento Desenho, Teste, Implementação e *Deployment*, incluem a criação de testes automatizados segundo a estratégia *Behavior Driven Development* e a documentação do código-fonte. Em cada *sprint*, os requisitos e desenho poderão sofrer alterações.

**Sprint 1** — 30 de Janeiro a 26 de Fevereiro (4 semanas / 18%)

- Construção do modelo de dados
- Suporte para aplicações *packaged*
- Suporte para aplicações *hosted*
- Perfis de utilizadores

**Sprint 2** — 27 de Fevereiro a 25 de Março (4 semanas / 18%)

- *Web layers* com interacção com mapa

**Sprint 3** — 26 de Março a 22 de Abril (4 semanas / 18%)

- Integração com módulo *OAuth* (autenticação)
- Integração com módulo de permissões (autorização)

**Sprint 4** — 23 de Abril a 20 de Maio (4 semanas / 18%)

- Sistema de comentários
- Sistema de classificação de aplicações
- Funcionalidades do administrador
- Suporte para pesquisa
- Suporte para favoritos

**Sprint 5** — 21 de Maio a 17 de Junho (4 semanas / 18%)

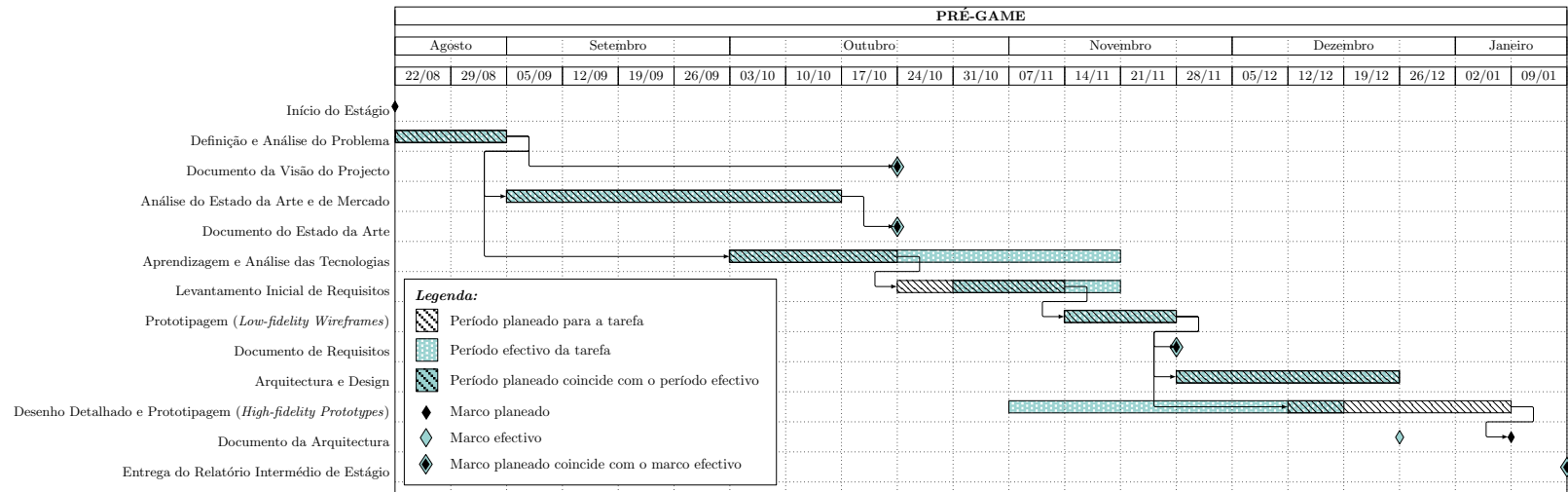
- Construção da API externa
- Integração com módulo de *logging*
- Suporte para internacionalização
- Suporte para aplicações móveis

**Conclusão da Documentação** — 18 de Junho a 01 de Julho (2 semanas / 10%)

- Conclusão e revisão final da documentação e preparação para a defesa

**Fim do Estágio** — 29 de Junho

# 1º Semestre — Desvios face ao plano inicial



## Justificação dos desvios ocorridos

### Aprendizagem e Análise das Tecnologias — prolongamento de 4 semanas

Esta fase estendeu-se por mais um mês do que o esperado pois foi necessário realizar experiências adicionais com algumas tecnologias (Apache Wookie, QUnit e Fabric) e dedicou-se mais tempo à aprendizagem (de *User Stories* e *Story Tests*, BDD, JavaScript e Backbone.js).

### Levantamento Inicial de Requisitos — adiamento de 1 semana

Tendo a fase “Aprendizagem e Análise das Tecnologias” se alongado por algumas semanas, optou-se por adiar o levantamento inicial de requisitos numa semana.

### Desenho Detalhado e Prototipagem (*High-fidelity Prototypes*) — antecipação de 5 semanas e prolongamento em 2 semanas

Esta fase consistiu no desenho de uma interface de interacção com o utilizador mais próxima da final, a partir do trabalho resultante da tarefa de criação de *Low-fidelity Wireframes*, e de um protótipo com algumas funcionalidades. A importância do protótipo aumentou no final do mês de Outubro, tendo sido pedido ao estagiário que criasse, durante o mês de Novembro, uma aplicação que demonstrasse algumas funcionalidades da plataforma. O estagiário optou por criar um protótipo da plataforma que permitisse a adição de aplicações do tipo *Hosted* e a sua utilização. Assim, esta tarefa realizou-se mais cedo, no mês de Novembro. Optou-se também por antecipar a criação da interface de interacção com o utilizador, a qual foi feita nas primeiras semanas de Dezembro. No total, demorou mais duas semanas que o esperado.

### Documento de Arquitectura — antecipação de 2 semanas

Uma vez que foi pedido ao estagiário que entregasse o relatório de estágio para revisão com pelo menos um mês de antecedência (ou seja, no dia 24 de Dezembro), este documento ficou igualmente concluído nessa data.

## 2º Semestre — Game e Pós-game (22 semanas)

### Justificação dos desvios ocorridos

As tarefas a realizar em cada *sprint* foram definidas nas reuniões de *Sprint Planning*, que se realizaram no início de cada *sprint*. Seguem-se os desvios ocorridos comparativamente ao plano inicial. Para uma descrição mais detalhada das tarefas realizadas deve ler-se o anexo “Artefactos da Metodologia de Desenvolvimento Ágil”. O diagrama *Gantt* não é apresentado neste caso por coincidir com o inicialmente planeado.

#### **Sprint 1** — 30 de Janeiro a 26 de Fevereiro (4 semanas / 18%)

- Suporte para aplicações *packaged* e *hosted*
- Autenticação e perfis de utilizadores
- Suporte para *developers*
- API's de utilizadores e de aplicações
- Sistema de notificações
- Integração inicial com a plataforma *One.Stop.Transport*
- Integração com módulo *OAuth 1.0*
- Configurações necessárias ao *deployment* de teste e produção

#### **Sprint 2** — 27 de Fevereiro a 25 de Março (4 semanas / 18%)

- Suporte para *web layers*
- *Sandbox* para desenvolvimento de *web layers*
- Estudo de mecanismos de partilha segura entre aplicações
- *Developer Dashboard* para gestão de chaves
- Suporte para acesso a API's externas por *web layers*
- Estudo da viabilidade do suporte para *OAuth 2.0*

#### **Sprint 3** — 26 de Março a 22 de Abril (4 semanas / 18%)

- Construção do modelo de dados
- Documentação para *developers*
- Novas capacidades de comunicação entre portal e aplicações
- Suporte para internacionalização
- Suporte para migrações da base de dados
- Criação de instâncias de teste e produção do TICE.Healthy

#### **Sprint 4** — 23 de Abril a 20 de Maio (4 semanas / 18%)

- Extensão das capacidades das aplicações *packaged* e *hosted*
- Integração com módulo *OAuth 2.0*
- Suporte para favoritos
- Continuação da documentação para *developers*

#### **Sprint 5** — 21 de Maio a 17 de Junho (4 semanas / 18%)

- Sistema de comentários
- Sistema de classificação de aplicações
- Funcionalidades do administrador
- Guia de estilos para a criação de aplicações
- Actualização de dependências para versões mais recentes

#### **Conclusão da Documentação** — 18 de Junho a 01 de Julho (2 semanas / 10%)

- Conclusão e revisão final da documentação e preparação para a defesa

#### **Fim do Estágio** — 29 de Junho

## 2.3 Outros Processos de Engenharia

### 2.3.1 Estruturação dos repositórios de código

Todo o sistema desenvolvido teve de servir de base para ambos projectos TICE.Mobilidade e TICE.Healthy. Como existem extensões que diferem de acordo com o projecto, dadas as especificidades que lhe estão associadas, foi preciso garantir que o estagiário pudesse desenvolver esta base ao mesmo tempo que outros desenvolveram componentes específicas a incluir na plataforma, e que fosse possível integrar código já desenvolvido no passado.

Para o efeito foi escolhido o sistema de controlo de versões Git. Este tem uma natureza distribuída, permitindo que cada programador tenha uma cópia local completa de todo o histórico de desenvolvimento e que possa realizar *commits* mesmo que esteja *offline* ou fora da rede privada da empresa [12]. Facilita ainda o *branching* e *merging* de código (isto é, a criação de “ramos” com certas características distintas e a fusão desses ramos divergentes), o que permite um desenvolvimento não linear. Por último, é de fácil integração com outras ferramentas utilizadas no projecto.

De uma perspectiva de alto nível, existem cinco *branches* remotos principais, representados na **figura 2.2**. *Branches* remotos são aqueles que existem no repositório principal, o qual está alojado nos servidores do IPN. Todo o código comum a ambos projectos está presente no *branch* remoto de desenvolvimento *appbase*. Deste surgem dois *branches* remotos para o projecto Mobilidade, um de desenvolvimento (nomeado *develop*) e um de produção (designado *master*) e dois para o projecto Healthy (também com as mesmas designações). Os *branches* remotos *master* correspondem a versões estáveis e *deployed*, enquanto que os *develop* dizem respeito a funcionalidades ainda não concluídas ou testadas. Desta forma, podem ser *pushed* para o repositório principal novos *commits* nos *branches* remotos dos projectos Mobilidade e Healthy. Podem ainda ser feitos *commits* no *branch* remoto *appbase*, os quais poderão futuramente ser adicionados novamente nos restantes *branches*. Esta operação (designada no Git por *rebase*) deve ser invocada manualmente, mas efectua a junção de código de forma automática. Em caso de conflito, cabe ao programador resolvê-los manualmente e localmente, sendo mantido todo o código pelo Git de forma segura. No que diz respeito a *branches* locais (os que não são *pushed* para o repositório principal), seguiu-se o modelo de *branching* designado *Git flow*<sup>1</sup> (coleção de boas práticas e de guias de orientação para a estruturação dos *branches* de um repositório Git).

---

<sup>1</sup>Mais informações sobre este modelo estão disponíveis em: <http://goo.gl/GDaF>.

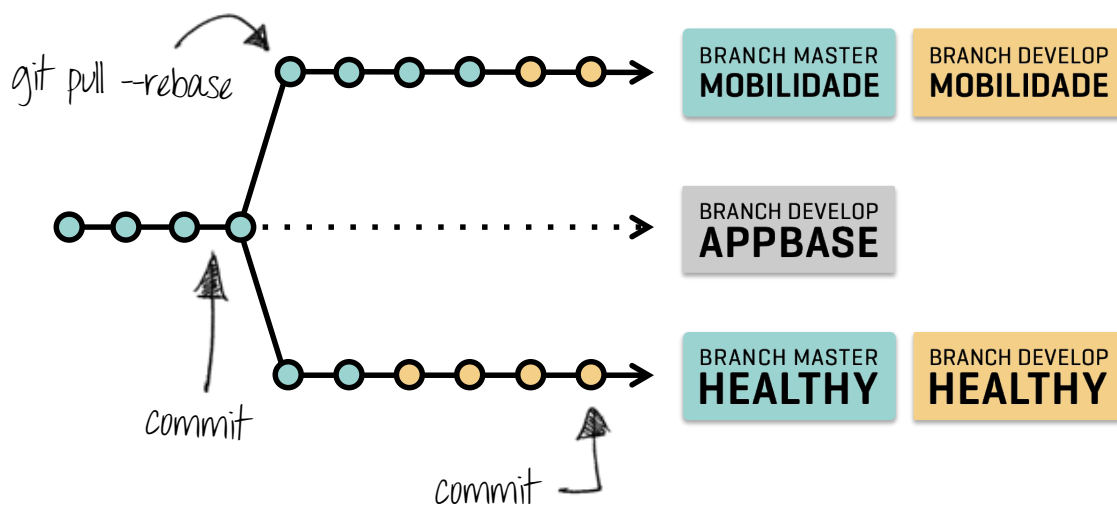


Figura 2.2: Estrutura dos repositórios de código.

### 2.3.2 Caracterização dos ambientes

Para que o projecto possa ser facilmente executado em máquinas com configurações, tecnologias e propósitos distintos, recorreu-se à definição de ambientes. Desta forma, é possível manter configurações específicas para, por exemplo, programar num computador pessoal ou testar o software numa máquina remota semelhante à visível pelo público. Cada ambiente tem um objectivo bem definido. Estes devem ser independentes pois as actividades executadas em cada um não devem interferir com os restantes. Segue-se uma breve descrição dos ambientes definidos. No capítulo “Análise de Requisitos” são indicados os requisitos tecnológicos associados a cada um desses ambientes.

#### Ambiente de desenvolvimento

É no ambiente de desenvolvimento que se programa o portal. As suas configurações permitem que o estagiário possa implementar as funcionalidades no seu computador pessoal. Trata-se de um ambiente mais leve que facilita o desenvolvimento<sup>2</sup>.

Existem três instâncias do ambiente de desenvolvimento — uma para a base comum do portal, outra para o projecto TICE.Mobilidade e outra para o projecto TICE.Healthy. São em tudo semelhantes mas executam código diferente (de acordo com o *branch Git* do projecto), utilizam bases de dados distintas, e diferem em

<sup>2</sup>A título de exemplo, é utilizado o motor de base de dados SQLite, em que a base de dados consiste num único ficheiro e que não requer servidor nem configuração. Tal tem a vantagem de não exigir esforço adicional de configuração para executar o projecto, mas não garante a escalabilidade de um motor de base de dados robusto. São também usados servidores de desenvolvimento, que se reiniciam quando se efectuam alterações no código do projecto. Da mesma forma que no caso anterior, garante comodidade mas não pode ser utilizado num ambiente de produção.

algumas das suas dependências (por exemplo, o projecto TICE.Mobilidade faz uso de bibliotecas para georeferenciação).

### Ambiente de teste

Este ambiente é utilizado exclusivamente para executar a bateria de testes. Deve por isso recorrer a uma base de dados que possa ser apagada periodicamente e pode conter ficheiros com *seed data* (dados fictícios para popular a base de dados). O ambiente tecnológico de teste tem as mesmas características do ambiente de desenvolvimento. Existem três instâncias do ambiente de teste pelas mesmas razões acima indicadas.

### Ambiente de *staging*

Este ambiente é empregue para testar a aplicação num ambiente igual ao de produção, mas de carácter privado. Sendo a configuração deste ambiente igual à de produção, oferece a garantia de que os desenvolvimentos têm o comportamento esperado na máquina de destino. Possibilita, assim, a existência de uma fase de transição. Tendo em conta os *branches* explicitados no capítulo anterior, a máquina correspondente a este ambiente possui o código dos *branches* `mobilidade-develop` ou `healthy-develop`, os quais se encontram sistematicamente “atrás” dos *branches* `mobilidade-master` e `healthy-master`, respectivamente. Assim, nenhuma alteração fica visível na máquina de produção sem que primeiro passe pela máquina de *staging*. Possui configurações ao nível da segurança e performance, da mesma forma que o ambiente de produção.

Este é actualizado no final de cada *sprint*. Existem duas instâncias do ambiente de *staging*, uma por cada projecto TICE, pelas mesmas razões acima indicadas, com a distinção de, neste caso, se tratarem de máquinas fisicamente distintas. Não existe um ambiente de *staging* para a base comum, pelas mesmas razões que serão apresentadas para o ambiente que se segue.

### Ambiente de produção

Este ambiente consiste na versão visível ao público do portal. É actualizado no começo de cada *sprint*, com os desenvolvimentos do penúltimo *sprint* (ou seja, as alterações de um *sprint* ficam durante o *sprint* seguinte no servidor de *staging* e só posteriormente passam para o servidor de produção, salvo raras excepções). Não existe um ambiente de produção para a base comum uma vez que esta não se destina ao utilizador final, e se pretende, neste momento, que não esteja visível ao público.



# Capítulo 3

## Estado da Arte

Um *web portal* é uma aplicação baseada em tecnologias web que agrega conteúdos provenientes de múltiplas fontes. O conceito de agregação é aqui aplicado como sendo a acção de juntar conteúdos de fontes distintas na mesma página web [13]. Os *web portals* são muitas vezes utilizados num contexto restrito como ponto de acesso único a informação de interesse por um público-alvo [14]. A informação é apresentada de forma consistente a fim de satisfazer diferentes requisitos e oferece usualmente mecanismos de *single sign-on* e personalização [15]. Exemplos de *web portals* bem conhecidos incluem o *iGoogle*, *Netvibes* e *Yahoo!* [16].

### 3.1 Estudo das capacidades dos *Web Portals*

Apesar do interesse em torno das recentes *Web Application Stores* (locais na Web em que são disponibilizadas e comercializadas aplicações web feitas por terceiros), estas têm diversos pontos comuns com os anteriores *Web Portals*. A **figura 3.1** descreve um possível processo a ser seguido por um utilizador de uma *Web Application Store*, como a *Chrome Web Store*, para adquirir uma aplicação. É o mesmo que é realizado para a adição de um *widget* num portal como o *iGoogle* ou o *Netvibes*.

As semelhanças entre estes *Web Portals* e as *Web Application Stores* vão desde da descoberta da aplicação (conseguida através de um catálogo, passo genericamente designado *Application Discovery*), passando pela promoção (a venda não se adequa tanto para o caso dos portais, uma vez que os *widgets* são frequentemente gratuitos) e listagem personalizada e persistente das aplicações adquiridas por um utilizador. A diferença mais visível está na utilização da aplicação em si — esta deixa de ser oferecida no contexto da *Store* assim que passa a ser utilizada, o que não acontece nos *Web Portals*. A **figura 3.2** pretende sumarizar as semelhanças e diferenças entre ambos. O termo *Web Application Portal* é utilizado como formalização da hipótese de aplicar o conceito de portal às novas aplicações web.

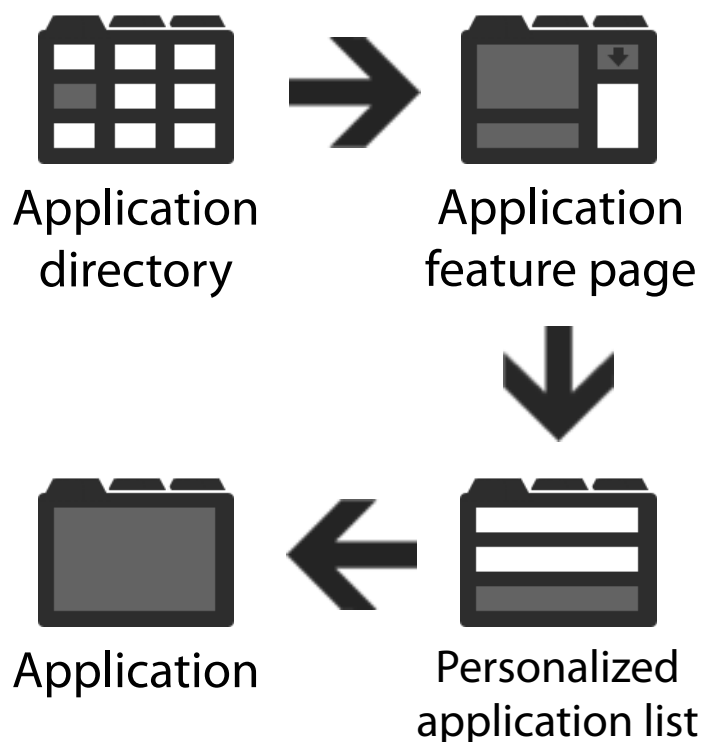


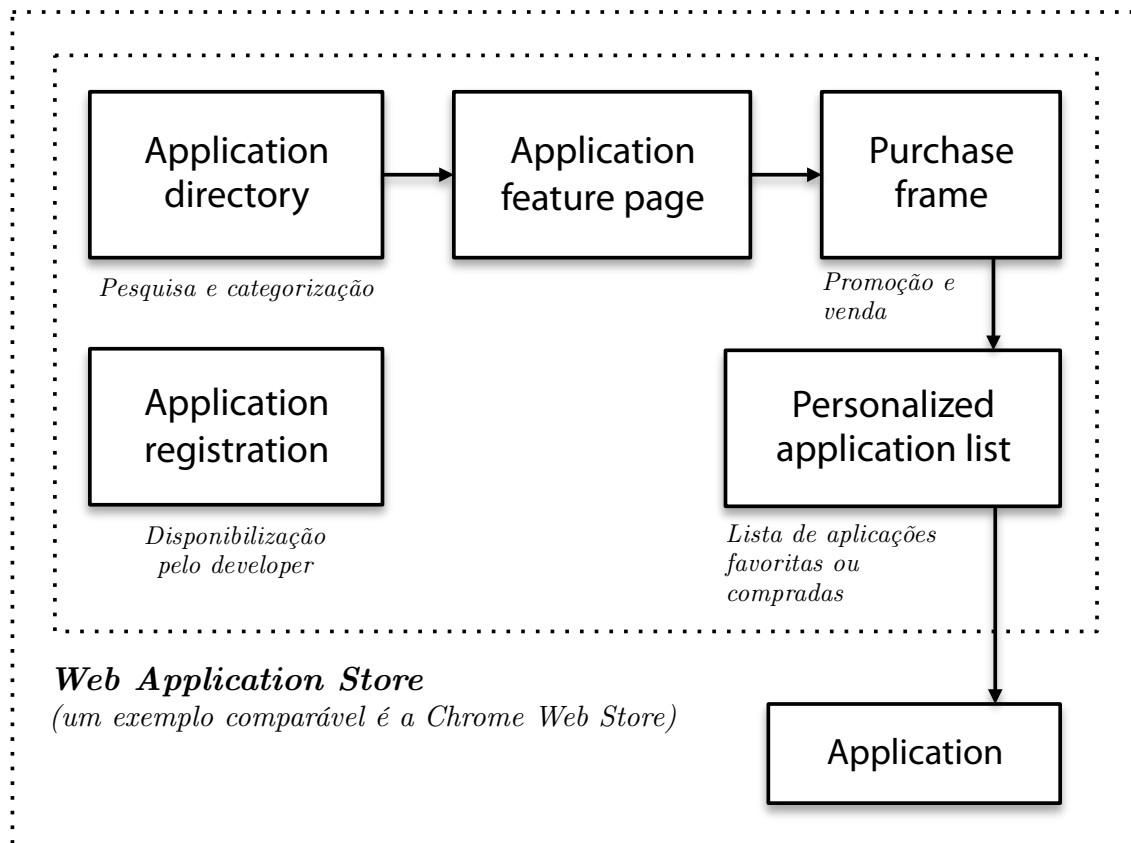
Figura 3.1: Processo de obtenção de uma aplicação disponível numa *Web Application Store*. Imagem adaptada [3].

Os quatro subcapítulos seguintes irão detalhar os conceitos chave que ambos têm em comum, nomeadamente a descoberta de aplicações, a sua promoção e venda, e disponibilização.

### 3.1.1 Descoberta de aplicações web

A descoberta de aplicações acontece normalmente através da *Store*. Nalguns casos, é também possível encontrar aplicações através de motores de pesquisa [17]. Uma situação comum na maioria das *Stores* prende-se no facto de serem frequentemente os mesmos produtores a ocuparem as zonas de destaque. Ainda assim, a inclusão de desenvolvedores de menor dimensão só foi possível graças a factores como:

- Capacidade de pesquisar dentro da *Application Store*;
- Possibilidade de conteúdo na Web conter ligações para aplicações na *Store*;
- Facilidade de utilização;
- Diversidade de modelos de negócio para além da venda ou subscrição;
- As redes sociais para permitir partilha e recomendações personalizadas.



- ⋯ Os grandes blocos ponteados delimitam o *scope* do que é uma *Web App. Store* e um *Web App. Portal*.
- Os blocos sólidos representam componentes (do ponto vista do cliente, cada um é constituído por páginas *html*).
- ➔ As setas representam ligações entre componentes (do ponto vista do cliente são hiperligações).

Figura 3.2: Componentes usuais numa *Web Application Store* e num *Web Application Portal*.

### 3.1.2 Promoção e venda de aplicações web

#### Entrada na *Web Store* vs *landing pages* tradicionais

Todas as aplicações existentes numa *Web Store* devem ter a sua página de entrada, que desempenhe o papel de promoção e venda da aplicação. Esta é apresentada ao utilizador caso este não tenha “instalado” a aplicação em causa, e fornece toda a informação que seja necessária para explicar o propósito da aplicação web e cativar o utilizador. Desta forma, a descoberta de aplicações acontece via *marketplace* e formas de pesquisa usuais [17]. O seguinte princípio de design descrito pela empresa *Google* para a *Chrome Web Store* concretiza esta situação:

Uma vez dentro da aplicação, o utilizador não deve ver *banners* externos explicando os benefícios da aplicação. O utilizador já escolheu e instalou a aplicação, e por isso não necessita que lhe seja vendida a aplicação novamente. Devem por isso apenas ser apresentados os elementos da interface que ajudem o utilizador a conseguir atingir o seu objectivo [17].

Existem diversas formas de construir uma *landing page* (ou página de entrada), e a melhor organização de conteúdos para cativar um futuro cliente depende em parte do tipo de aplicação web que se pretende vender. Desta forma, é positivo que uma das seguintes situações seja permitida:

- Deixar que o desenvolvedor construa a sua própria *landing page*, que substitua a página da entrada da aplicação na *web store*;
- Incluir na página da entrada da aplicação uma ligação que direcione o utilizador para uma página alternativa.

#### *Launch pages*

Um dos primeiros passos normalmente dados no desenvolvimento de aplicações web é a empresa responsável tornar pública uma página temporária, genericamente designada *Coming Soon page* (ou *Launching page*). Esta visa divulgar o novo serviço e reunir endereços de email de possíveis interessados [18]. Existir uma *landing page* com a descrição da aplicação que se pretende oferecer antes desta estar disponível ao público pode ter um impacto positivo. O mesmo se aplica à possibilidade deste tipo de página existir enquanto a aplicação se encontra em *private beta* (modo de utilização restrito a certos utilizadores). As *Web Application Stores* mais utilizadas actualmente (como a *Chrome Web Store*) não suportam este tipo de páginas.

### 3.1.3 Registo das aplicações

Com a finalidade de distinguir páginas web de aplicações web, a equipa que desenvolve o *Google Chrome* adicionou suporte para um tipo de aplicações designadas *Installable Web Apps*, podendo ser encontradas e “instaladas” a partir da *Chrome Web Store*. Estas podem estar alojadas em servidores externos (designadas *Hosted Apps*) ou consistirem em código que executa do lado do cliente (designadas *Packaged Apps*, e também conhecidas por *Serverless Applications*)<sup>1</sup>.

As *Hosted Apps* consistem em código-fonte existente num servidor remoto, num ficheiro de manifesto que possui metadados como o *URL* para esse servidor e o título da aplicação (e ainda informação adicional como permissões de acesso), e nos ícones que irão representar a aplicação. As *Packaged Apps* consistem em código-fonte *HTML*, *CSS*, *JavaScript*, os *assets* associados às páginas *HTML*, o ficheiro de manifesto e os ícones mencionados no caso anterior. Em ambos casos, toda a informação é reunida num único ficheiro *CRX* (um arquivo comprimido).

Os *widgets*, componentes especificados pelo *W3C*, também funcionam com base num ficheiro comprimido com o mesmo conteúdo que foi referido. Este é o *standard* seguido pelas extensões e *widgets* do *browser Opera*, e por telemóveis *WAC-compatible*. O *Google* possui vários formatos que utiliza em produtos e serviços distintos, mas todos atingem objectivos semelhantes. Para além do formato referido para as *Installable Web Apps*, possui outro formato para o *Google Enterprise Apps Marketplace* e outro para os *iGoogle Gadgets* e para o *Google Wave*. As semelhanças entre o formato de manifesto usado no *Google Chrome* e o formato especificado pelo *W3C* são tais ao ponto de, como prova de conceito, ter sido criado um *script* que faz a conversão do primeiro formato para o segundo [19]. Existe ainda um outro formato, especificado pela *Mozilla* [20]. Têm sido feitas críticas ao facto das várias empresas interessadas estarem a desenvolver os seus vários formatos ao invés de seguirem o *standard* do *W3C* e de eventualmente trabalharem em conjunto para o melhorar [21].

Há autores que defendem que esta não é a melhor forma de especificar aplicações web [22]. A alternativa que propõem foca-se na definição dos metadados no próprio código da página *HTML* principal da aplicação. Segue-se um exemplo de possíveis *metatags* e ligações a incluir numa página *HTML*, para a tornar numa aplicação web:

```
<meta name='application-name' content='...'>
<meta name='application-description' content='...'>
<link rel='application-data' href='app.json'> <!-- Optional ->
<link rel='icon' size='57x57' href='icon.png'>
```

---

<sup>1</sup>Vários autores não consideram o segundo tipo como sendo aplicações web (uma vez que muitas delas nunca interagem com um servidor para realizarem as funcionalidades que expõem) mas sim aplicações desenvolvidas com tecnologias para a Web.

### 3.1.4 Utilização da aplicação através do portal

Tecnicamente, a inclusão de uma aplicação no interior de um *Web Portal* pode ser feita de diversas formas. O capítulo seguinte detalha várias formas de resolução deste problema. Por sua vez, o capítulo “Arquitectura e Design”, mais especificamente os subcapítulos “Servidor de *widgets* para as aplicações web *packaged*” e “Contentor para as aplicações web *hosted*”, abordam o mecanismo escolhido para o projecto.

## 3.2 Especificações relevantes

Em 2001 surgiu a *Java Specification Request 168* (*JSR 168*) que introduziu o conceito de *portlet*. Um *portlet* é uma aplicação que fornece um conteúdo específico que pode ser incluído num *Web Portal* [13]. Em 2007, surgiram os *web widgets* (também conhecidos por *badges* ou *gadgets*) [23]. Estes consistem em pedaços de código *HTML* e *JavaScript*, oferecidos normalmente através de *iframes*, ou objectos *Adobe Flash*, sendo uma alternativa mais *lightweight* e menos dispendiosa do que os anteriores servidores de *web portals*. Exemplos de *widgets* incluem desde pequenas aplicações como relógios, conversores de moedas, jogos, leitores de *RSS*, etc., a aplicações bastante completas como editores de texto.

Existe actualmente um ecossistema rico de especificações — algumas complementam-se, outras são concorrentes (figura 3.4). O capítulo que se segue fará uma análise das várias soluções existentes. O estudo de especificações é baseado na análise realizada pela equipa que desenvolveu o projecto europeu *Palette* (EU FP6) [24].

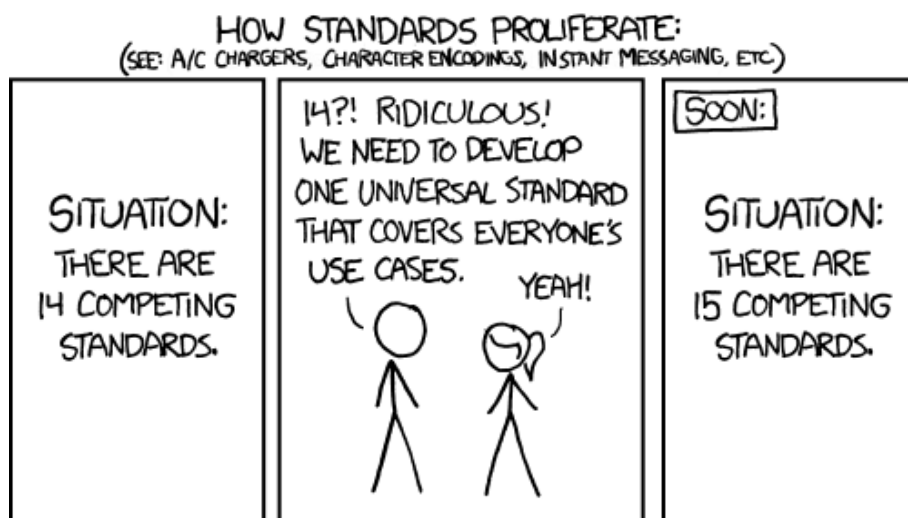
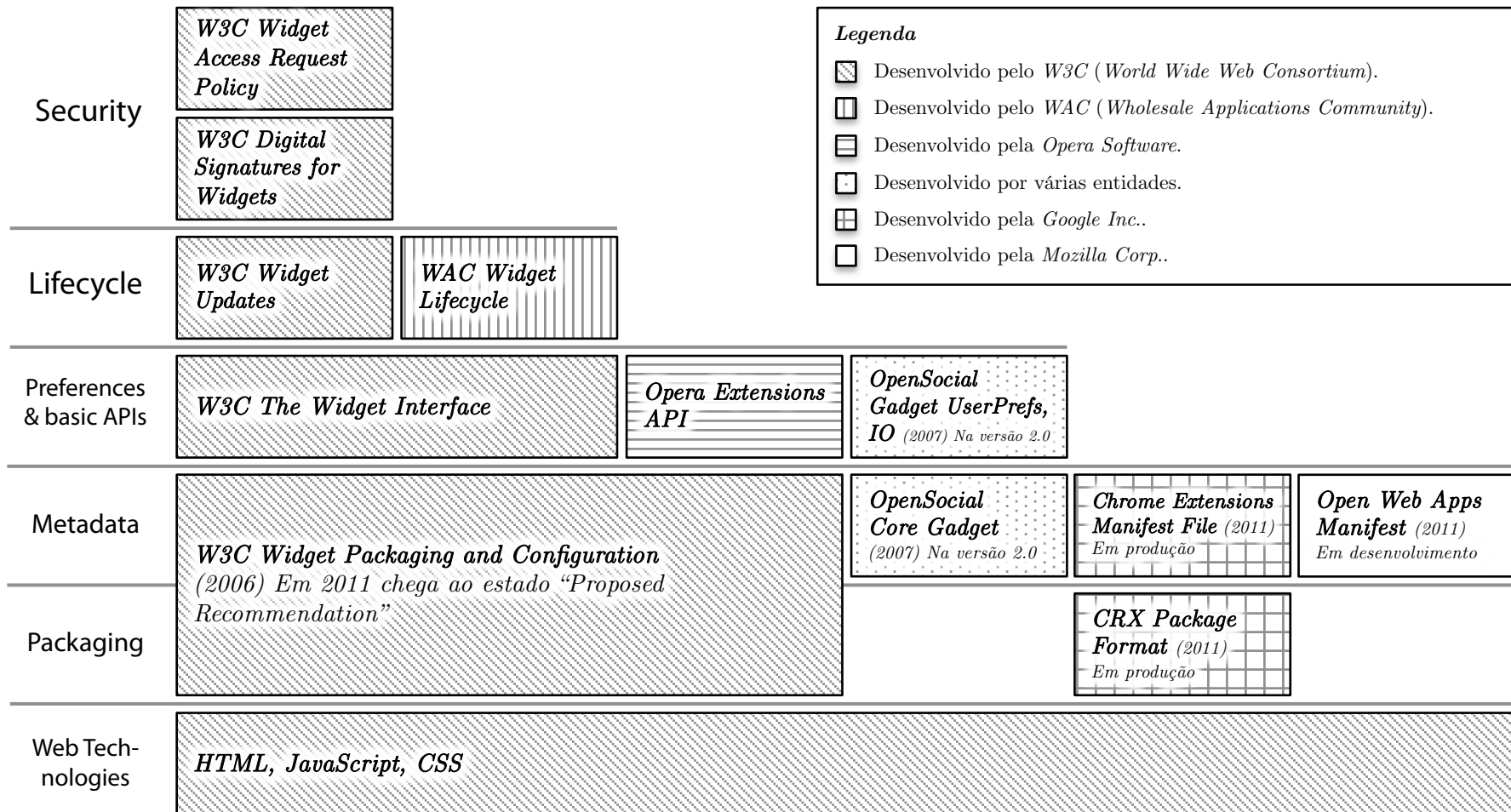


Figura 3.3: A proliferação de *standards*. Fonte: <http://xkcd.com/927/>.

Figura 3.4: Pilha de especificações relacionadas com *widgets* e aplicações web [4].

### 3.2.1 Java Portlet Specification 2.0

A *Java Specification Request 168* introduziu o conceito de *portlets* — aplicações que fornecem um conteúdo específico (na forma de fragmentos de *markup*) que pode ser incluído em qualquer *Web Portal* que respeite o *standard* [13]. Suporta a *design pattern MVC* (*Model-View-Controller*) através da definição de duas fases de acção (processamento e *rendering*), *portlet modes* (que o portal pode utilizar para definir que conteúdo deve ser gerado ou que tarefa deve ser realizada), *window states*, *portlet data models* e um formato de *packaging*. A especificação está na sua versão 2.0 [24].

Tem como principal limitação o facto de estar restrita à linguagem *Java*. Sendo um dos requisitos do projecto seguir-se uma especificação que permita a interoperabilidade entre linguagens e plataformas, esta especificação não se adequa e não foi por isso analisada com maior detalhe.

### 3.2.2 Microsoft WebParts

São semelhantes aos *portlets* em *Java* mas orientados às tecnologias e ambiente *ASP.NET* da *Microsoft* [25]. Pelas mesmas razões acima indicadas, esta especificação não se adequa e não foi por isso analisada com maior detalhe.

### 3.2.3 W3C Widgets

A especificação *Widgets 1.0* foi publicada em 2007 pelo grupo *Web Application Formats Working Group* do W3C e incide nas generalidades e princípios-chave da especificação de aplicações web, deixando de parte alguns dos aspectos mais técnicos relativos à sua construção. Como tal, serve como um ponto de partida para a criação de especificações mais complexas ou para a adição de novas características específicas da aplicação. O W3C pretende com esta especificação standardizar a forma como as aplicações web do tipo *client-side* devem ser escritas, assinadas digitalmente, protegidas, compactadas e *deployed* de forma independente da plataforma [24].

**Formato do arquivo** Os *widgets* são um arquivo comprimido de ficheiros, conforme descrito pela especificação do formato de ficheiro ZIP. Os ficheiros que constituem o *widget* devem estar na raiz do arquivo ZIP, juntamente com todos os recursos que lhe devem estar associados (na mesma directoria ou em subdirectorias) [24].

**Ficheiros associados** Todos os *widgets* devem conter os seguintes ficheiros [24]:

- Um ficheiro de manifesto com o nome *config.xml* que possui as informações necessárias para inicializar o *widget*;
- Um documento designado *index.html*. Este ficheiro é exibido ao utilizador tendo em conta as propriedades do arquivo *config.xml*.



**Ficheiro de configuração** O ficheiro *config.xml* permite especificar o nome do *widget*, as suas dimensões, bem como informações adicionais como o autor, a descrição do *widget*, o seu ícone representativo, e detalhes de segurança [24].

**Ficheiro de dados** O documento principal *index.html* é um ficheiro com as mesmas características que uma página web usual. Este pode referenciar conteúdo externo e incluir e executar linguagens de *scripting* interpretáveis, como JavaScript. Não é possível incluir linguagens para *server-side scripting* como Python [24].

### 3.2.4 Opera Widgets

Os *widgets* desenvolvidos pela empresa Opera são muito semelhantes aos *widgets* do W3C, considerando que o documento “Widgets 1.0 Working Draft” é baseado na versão inicial da especificação “Opera Widgets 1.0”. De modo semelhante aos anteriores, estes consistem em pequenas aplicações web que executam directamente no *browser* do utilizador, ou fora deste (sem a janela do *browser*), se o *browser* Opera se encontrar instalado [24].

**Formato do arquivo** Os *widgets* são comprimidos no formato ZIP e têm a sua extensão modificada para “.wdgt”. Os ficheiros que constituem o *widget* devem estar na raiz do arquivo ZIP, na mesma directoria ou em subdirectorias [24].

**Ficheiros associados** Todos os *widgets* devem conter um ficheiro de manifesto XML e um documento HTML com a mesma natureza e propósito que os ficheiros equivalentes dos *widgets* especificados pelo W3C [24].

**Ficheiro de configuração** O ficheiro de configuração dos *Opera Widgets* é idêntico ao ficheiro *config.xml* dos *widgets* especificados no documento *W3C Widgets 1.0*. Os elementos permitidos no ficheiro XML são semelhantes, com apenas algumas alterações, como o facto do elemento ID ser obrigatório [24].

**Ficheiro de dados** Para além das funcionalidades oferecidas pelo HTML e JavaScript, este ficheiro pode fazer uso de conteúdos suportados nativamente pelo *browser* Opera, como os tipos de ficheiros SVG e XML [24].

### 3.2.5 Google Gadgets

*Google Gadgets* são aplicações de dimensões reduzidas, utilizadas no contexto da *homepage* personalizada da empresa Google, designada iGoogle. Estas podem ser similarmemente utilizadas através da aplicação Google Desktop [24].

**Formato do arquivo** Um *gadget* é composto por um único ficheiro XML. Todo o conteúdo externo, como folhas de estilo, *scripts* ou imagens deve ser carregado a partir de um servidor remoto [24].

**Ficheiros associados** Um *gadget* está subdividido em três secções distintas [24]:

- *Secção de conteúdo*: possui a lógica de negócio e de apresentação que determinam as funcionalidades e o aspecto gráfico do *gadget*;
- *Preferências do utilizador*: define os controlos que permitem aos utilizadores especificar configurações para o *gadget*;
- *Preferências do gadget*: especifica características, como as suas dimensões.

### 3.2.6 Netvibes Universal Widgets

Os *Netvibes Universal Widgets* são criados através da Netvibes Universal Widgets API (UWA). Esta surge com o objectivo de permitir a criação de *widgets* que possam ser executados num grande número de plataformas (como o iGoogle, a *dashboard* da Apple, a barra lateral do Windows Vista, entre outros). Para conseguir a compatibilidade entre as várias plataformas, um *widget* UWA inclui código JavaScript que produz conteúdo específico de acordo com o ambiente de execução [24].

**Formato do arquivo** Um *widget* UWA deve ter um único ficheiro estático XHTML. Este deve ser codificado em UTF-8 e incluir um *namespace* específico. Conteúdo externo deve ser carregado de um servidor remoto através de pedidos *Ajax* [24].

**Ficheiros associados** Um *widget* UWA está estruturado nas seguintes secções:

- *Metadados*: descritos na *head* do ficheiro XHTML, utilizando *metatags* padrão;
- *Preferências*: declaradas usando as *tags* específicas para preferências, colocadas igualmente na porção *head* do arquivo XHTML;
- *Conteúdo*: inclui os arquivos de emulação (um ficheiro CSS e um ficheiro JavaScript para simular o ambiente Netvibes), *scripts inlined*, secções de estilo e o corpo estático do ficheiro XHTML [24].

### 3.2.7 Yahoo! Widgets, Microsoft Gadgets e Apple Dashboard Widgets

Não existem diferenças muito significativas entre estas especificações e as analisadas anteriormente para a definição de *widgets*. Os *Yahoo! Widgets* possuem um ponto distinto pelo facto de serem definidos inteiramente através de elementos XML, não sendo possível empregar HTML. Como tal, não são processados pelo *browser*, mas sim por um ambiente de execução específico, o Yahoo! Widgets Konfabulator [24].

### 3.3 Tecnologias e ferramentas

#### 3.3.1 OpenSocial e Apache Shindig

O *OpenSocial* possui mecanismos que permitem criar uma *Social App Platform*, o correspondente ao que se tem designado *Web Application Portal*, e *Social Apps*, que são aplicações web [26]. Tem a grande vantagem das aplicações implementadas para este sistema poderem aplicar-se em todas as plataformas baseadas no *OpenSocial*, o que inclui, entre outros, o *eBay*, *Friendster*, *GROU.PS*, *hi5*, *iGoogle*, *LinkedIn*, *MySpace*, *Netlog*, *Ning*, *orkut*, *PayPal*, e *Yahoo!* [27]. A empresa *Google* suporta o *OpenSocial* em vários dos seus produtos. Por exemplo, para utilizadores do *Google Apps*, é possível incluir qualquer *gadget* neste formato em produtos como o *Gmail*.

O *Apache Shindig* é a implementação em *Java* de referência para o *OpenSocial*. A figura 3.5 mostra a arquitectura geral do *Apache Shindig* [28].

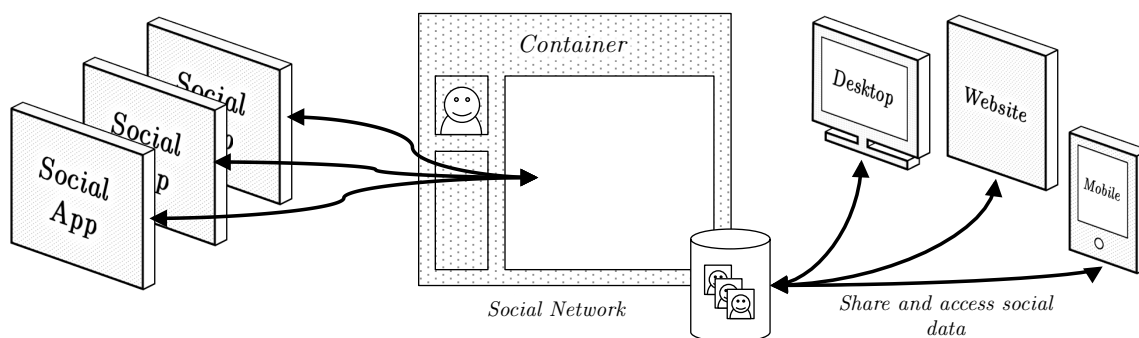


Figura 3.5: Arquitectura de alto-nível do *Apache Shindig*.

#### 3.3.2 Apache Wookie

O *Apache Wookie* é um servidor REST escrito na linguagem *Java* que permite o carregamento e *deployment* de *widgets*. Este segue a especificação de W3C Widgets, mas também permite a inclusão de *widgets* que façam uso de outras APIs, nomeadamente do *OpenSocial*. Quando utilizado conjuntamente com outra plataforma, este assegura a gestão de *widgets* [29].

#### 3.3.3 Apache Rave

O *Apache Rave* é uma plataforma social extensível escrita em *Java* que permitirá hospedar, servir e agregar *gadgets* do *OpenSocial* através de uma interface gráfica intuitiva e personalizável. Tal será conseguido através da integração de ambos projectos *Apache Shindig* e *Apache Wookie*. Esta plataforma encontrava-se (no momento da escrita do presente relatório) numa fase de desenvolvimento inicial [30].

## 3.4 Principais soluções comerciais e projectos de investigação

Nesta secção será feita uma breve análise de algumas das soluções existentes no mercado com funcionalidades equiparáveis à plataforma que se pretende desenvolver.

### 3.4.1 Descrição geral

#### iGoogle

O iGoogle é um serviço prestado pela empresa Google que consiste num portal web que suporta *web feeds* e *web gadgets*. Os *gadgets* são definidos através de uma especificação própria, aberta, designada *Google Gadgets* [31]. Tanto o desenvolvimento de aplicações como a sua utilização está aberta ao público [32]. Funciona de modo semelhante a outros portais como o My Yahoo!, Netvibes, e Pageflakes [16].

#### Facebook Apps

As aplicações do Facebook diferem largamente das soluções *client-side* analisadas anteriormente. Consistem em aplicações hospedadas num servidor externo e que são acedidas no contexto do Facebook numa página designada *Canvas* (figura 3.6). Conjuntamente, oferecem funcionalidades através de API's, sendo um exemplo os *Social Channels*, que incluem *bookmarks*, notificações, *News Feed stories*, e pesquisa [33]. O Facebook oferece ainda um sistema de autenticação, designado Facebook Connect, que permite a aplicações externas suportar contas Facebook, como mecanismo alternativo à realização de um novo registo de utilizador [34].

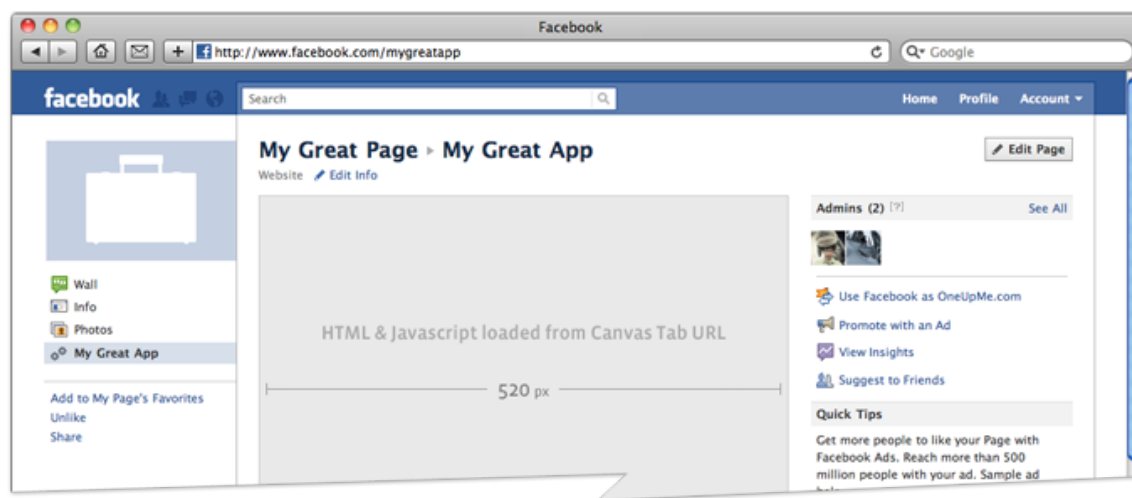


Figura 3.6: Espaço ocupado por uma aplicação no Facebook.

## LinkedIn Apps

O LinkedIn suporta aplicações que sigam a especificação OpenSocial. Estas aplicações podem ser adicionadas à *homepage* do utilizador e ao seu perfil profissional (figura 3.7) [35].

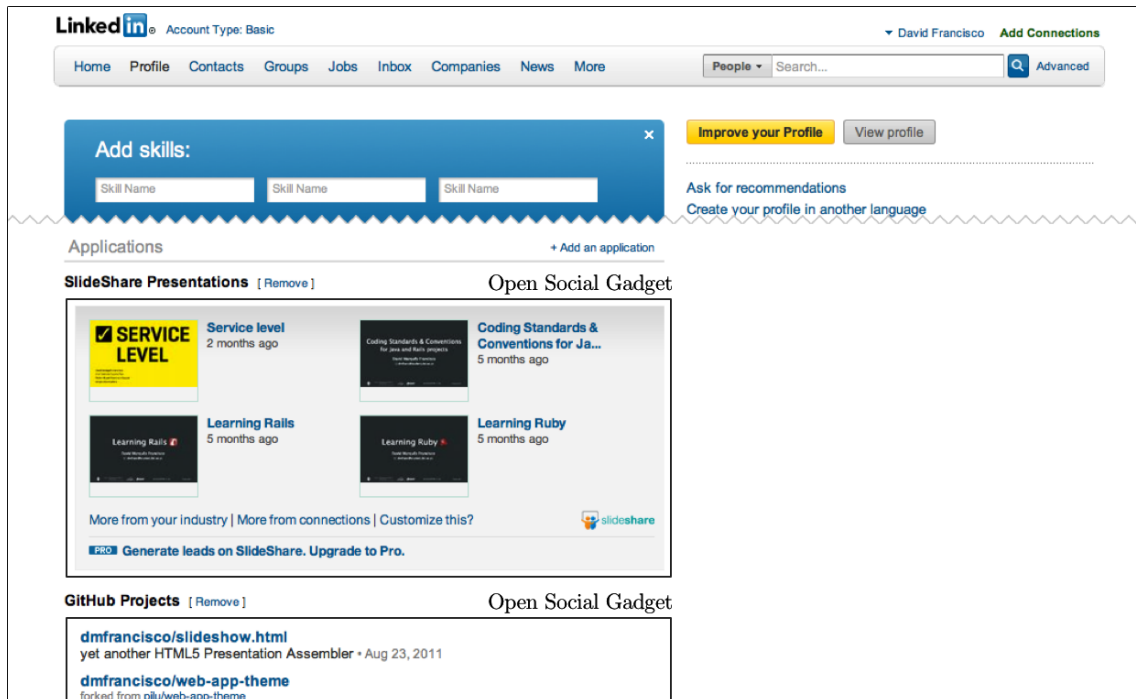


Figura 3.7: Dois *gadgets* associados a um perfil do LinkedIn.

## Twitter Apps

O Twitter não possui aplicações, pelo menos da mesma forma que os casos de estudo anteriores. Oferece uma API REST que permite interagir com a maioria das funcionalidades do *site* [36]. Disponibiliza ainda um mecanismo de autenticação e autorização que pode ser utilizado por aplicações externas (semelhante ao Facebook Connect). Para esse fim, utiliza a especificação *OAuth* [37]. As permissões que um utilizador concede a cada aplicação são granulares, de modo a limitar o acesso a dados específicos do utilizador e a funcionalidades que comuniquem em seu nome.

## Chrome Web Store

A Chrome Web Store permite a especificação de dois tipos de aplicações – *packaged*, que consistem num arquivo comprimido (que possui um ficheiro de manifesto, um ou mais ficheiros HTML, folhas de estilo e código JavaScript) [38], e *hosted*, que consistem num ponteiro para um servidor externo [39]. O mecanismo de execução de aplicações não é interoperável, estando restrito ao *browser* Chrome. A Web Store permite a compra de aplicações, através de várias modalidades de pagamento.

## Podio App Store

O Podio é um sistema web de trabalho colaborativo para empresas que garante a sua extensibilidade mediante aplicações web externas. Permite a descoberta e compra de aplicações de uma forma semelhante à Chrome Web Store [40].

## Palette e Mywiwall

O projecto *Palette* (EU FP6) consiste no desenvolvimento de uma plataforma para interoperabilidade de serviços. Consiste num projecto europeu iniciado em 2009 com o objectivo de resolver o problema da heterogeneidade de ferramentas e serviços em comunidades do sector da educação, através de mecanismos inovadores de integração. Os serviços de integração propostos incluíam pesquisa, autenticação, armazenamento unificados, um *web portal*, entre outros. Os serviços disponíveis a integrar cobriam uma vasta gama nomeadamente aplicações web, *desktop* e *web services*. O *Palette Services Portal* pretendia ainda cumprir os seguintes requisitos [24]:

- Possibilitar a integração entre serviços já existentes;
- Actuar como ponto central de acesso a todos os serviços;
- Suportar vistas personalizáveis;
- Oferecer uma visão global da actividade de grupos.

De um ponto de vista não-funcional, a plataforma viria ainda a cumprir os seguintes requisitos de alto nível [24]:

- Permitir o desenvolvimento rápido e simples de novos *web widgets*;
- Evitar necessidade de aprendizagem de um novo ambiente;
- Garantir a integração na arquitectura web através de *web services* REST;
- Ser uma solução aberta e interoperável, com a finalidade de não se ficar restrito a uma plataforma proprietária.

O projecto baseia-se na especificação do W3C, ao qual foram feitas algumas extensões. No entanto, essas extensões estão definidas num documento à parte, permitindo que os *widgets* (sem as extensões) funcionem noutras plataformas. As extensões dividem-se em várias categorias [24]:

- Gestão de preferências;
- Comunicação com servidores remotos;
- Comunicação entre *widgets*;
- *Drag & drop* de componentes entre *widgets*.

### 3.4.2 Tabela comparativa

Encontram-se listados, de seguida, os vários critérios de comparação aplicados na análise das várias soluções comerciais e projectos de investigação existentes.

**Portal (aplicações embebidas)** — As aplicações são executadas no próprio portal em que se processa a descoberta da aplicação.

**Single sign-on** — O utilizador apenas precisa de efectuar *login* na plataforma, sendo este partilhado com as aplicações que utiliza.

**Aplicações com permissões granulares** — Quando a aplicação pretende aceder a dados específicos do utilizador ou a funcionalidades que comuniquem em seu nome, esta solicita ao utilizador as permissões requeridas.

**Interface de interacção com o utilizador consistente** — As aplicações e a plataforma partilham os mesmos estilos, de forma a que aparentem ser uma só, para o utilizador.

**Cross-browser** — As funcionalidades da plataforma funcionam em todos os *browsers* populares (não se trata de uma solução específica de um fabricante).

**Suporte para aplicações web** — Disponibiliza aplicações que executem no *browser* e que façam uso das tecnologias abertas da web.

**Suporte para aplicações móveis nativas** — Disponibiliza aplicações nativas para *smartphones* e *tablets*.

**Suporte para aplicações *desktop*** — Disponibiliza aplicações para *desktop*. Estas não devem estar fortemente baseadas em tecnologias web (isto é, não devem ser aplicações maioritariamente escritas em HTML).

**Inclui sistema de pagamento** — Permite efectuar a compra ou subscrição de serviços prestados através de aplicações.

Tabela 3.1: Comparação das funcionalidades das várias soluções comerciais e projectos de investigação analisados.

	Portal ( <i>apps</i> embebidas)	<i>Single</i> <i>sign-on</i>	Aplicações com perm. granulares	UI con- sistente	<i>Cross-</i> <i>-browser</i>	Suporta web <i>apps</i>	Suporta <i>apps</i> móveis nativas	Suporta <i>apps</i> para <i>desktop</i>	Suporta pa- gamentos
iGoogle	✓	✓	✗	✗	✓	✓	✗	✗	✗
Facebook Apps	✓	✓	✓	✓	✓	✓	✓	✗	✗
LinkedIn Apps	✓	✓	✓	✓	✓	✓	✗	✗	✗
Twitter Apps	✗	✓	✓	✗	✓	✓	✓	✓	✗
Chrome Web Store	✗	✗	✗	✗	✗	✓	✗	✗	✓
Podio App Store	✓	✓	✗	✓	✓	✓	✓	✗	✓
Palette e Mywiwall	✓	✓	✗	✗	✓	✓	✗	✓	✗
Appbase (a)	✓	✓	✓	✓	✓	✓	(b)	(b)	(b)

(a) Appbase é a plataforma desenvolvida no âmbito do estágio. Quando concluída, se forem cumpridos todos os requisitos com prioridade MUST e SHOULD, esta terá as características indicadas na tabela.

(b) Corresponde a um requisito de carácter obrigatório para o projecto, mas de prioridade OPTIONAL para o estágio.



# Capítulo 4

## Análise de Requisitos

Requirements rarely lie on the surface. They're buried deep beneath layers of assumptions, misconceptions, and politics.

— *The Pragmatic Programmer book*

A especificação das funcionalidades do projecto foi feita com recurso a *user stories*. *User stories* consistem em descrições sumárias de funcionalidade expostas na perspectiva do utilizador [41] e são comuns em projectos que seguem metodologias de desenvolvimento ágil [42]. Neste capítulo serão analisados os requisitos funcionais seguindo o formato enunciado, especificados os vários actores intervenientes na plataforma, e listados requisitos de sistema, desempenho e segurança.

Associados a este capítulo estão os anexos “Lista de Requisitos”, que possui uma descrição completa de cada requisito (enquanto que nesta secção apenas são apresentados título e prioridade), e “*Story Tests*”, a especificação dos testes associados a cada requisito funcional.

Os vários requisitos apresentados e as respectivas prioridades têm como origem as candidaturas dos projectos TICE.Mobilidade e TICE.Healthy, as equipas de gestão e desenvolvimento, e parceiros, internos e externos, dos projectos TICE. A fonte de cada requisito pode ser igualmente consultada no anexo “Lista de Requisitos”.

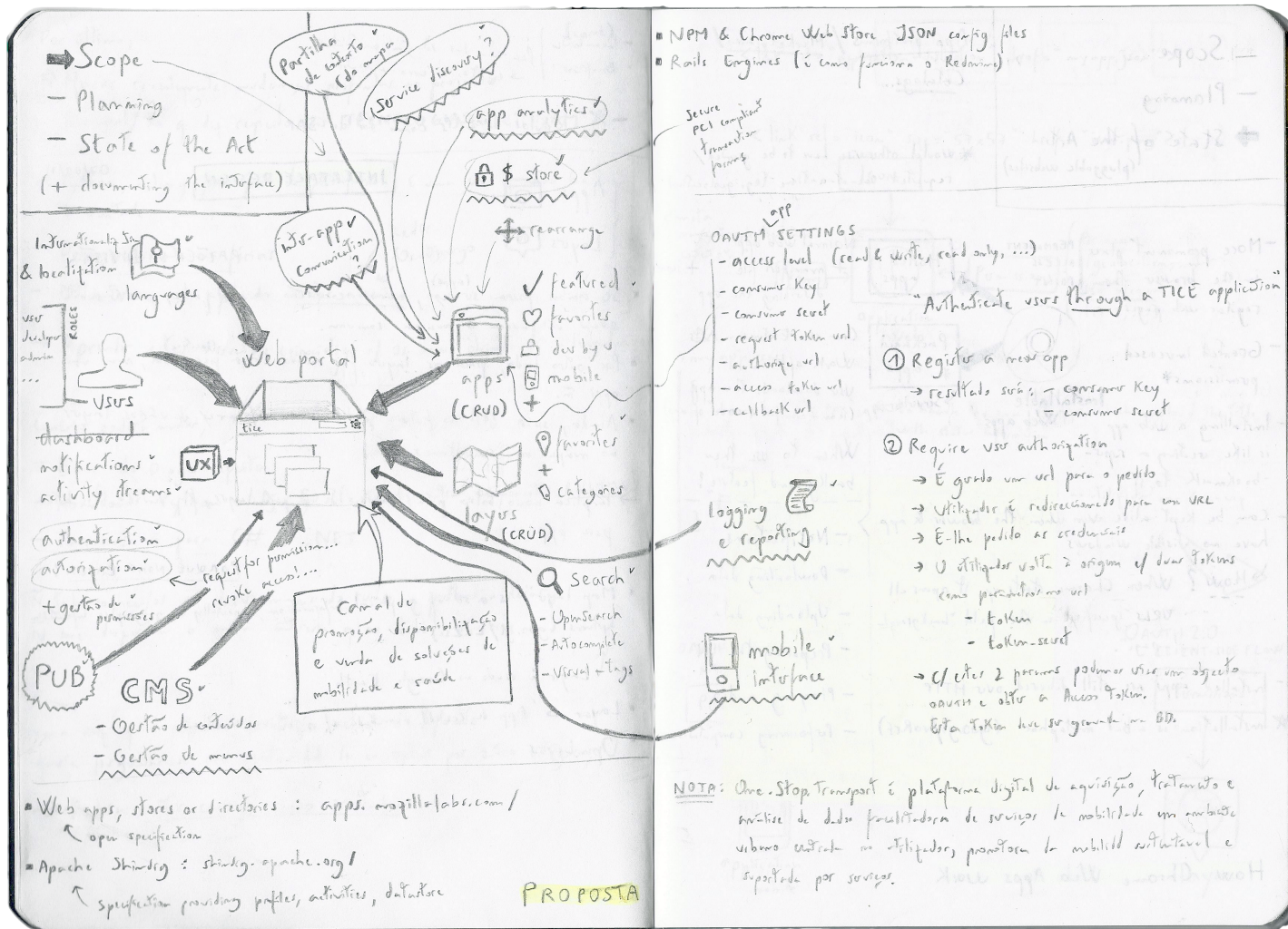


Figura 4.1: Notas escritas durante a definição do scope do estágio.



Figura 4.2: Actores do sistema. A seta representa a herança de actores (significando o mesmo que nos diagramas *UML* de casos de uso).

## 4.1 Actores

Os actores existentes no sistema são *Visitante*, *Utilizador*, *Programador*, *Gestor* e *Administrador* (figura 4.2). O *Visitante* consiste num cliente que não possui registo na plataforma, que navega no portal sem uma conta associada. O actor *Utilizador* consiste num *Visitante* que se registou na plataforma e que está autenticado. O tipo de actor *Programador*, para além das capacidades que possui, que lhe são características, pode efectuar todos os casos de uso do *Utilizador*. Um *Gestor* tem acesso a algumas zonas da interface de administração, nomeadamente à secção de aprovação de aplicações. Um *Administrador* possui todas as capacidades de um *Gestor* e pode ainda gerir as entidades da plataforma, como contas de utilizador e aplicações. No que diz respeito a este tipo de actor, é necessário que exista na plataforma a possibilidade de se lhe retirarem certas permissões. Para tal será utilizado um sistema de permissões que está fora do âmbito deste estágio. Desta forma, nas *user stories* serão descritas as capacidades de um *Administrador* que possui a totalidade das permissões.

## 4.2 Requisitos funcionais

Os requisitos funcionais foram especificados com recurso a *user stories*. *User stories* são pequenas descrições de algo que um utilizador irá fazer quando acede à plataforma, focadas no valor do resultado que obtém ao realizá-lo [43]. A definição de requisitos através de *user stories* tem a vantagem de fazer pensar no que se está a construir da perspectiva do utilizador final (uma vez que reflectem as suas necessidades e a sua interacção com o software). Capturam de uma forma simples funcionalidade e acções, evitando assim perder-se demasiado tempo com os detalhes de implementação numa fase inicial do projecto. São de fácil leitura por pessoas sem conhecimentos técnicos na área [44] e são de mapeamento directo para testes funcionais, de integração e aceitação, utilizando uma *framework* de testes do estilo *BDD* (*Behavior Driven Development*) ou *TDD* (*Test Driven Development*), como é o caso das ferramentas *Lettuce* e *Pavlov*, descritas no capítulo de Testes.

As *user stories* respondem a três questões importantes na definição de funcionalidades [45]:

- O que se pretende?
- Quem o pretende?
- Porquê?

Para as *user stories* que contemplam interacção directa com o utilizador, o formato mais utilizado na sua definição é o seguinte [42]:

- Enquanto <actor do sistema>
- Quero <acção>
- De forma a <proposta de valor>

Definir o actor implica pensar no interveniente que utilizará a funcionalidade. A acção descreve o que acontecerá (mas não como esta deverá ser executada). A proposta de valor (ou resultado) indica o propósito da funcionalidade [42]. Seguindo este *template*, uma possível *user story* seria:

- **Enquanto** utilizador
- **Quero** configurar os detalhes do meu perfil
- **De forma a** ter a minha identidade no portal

As *user stories* não se adequam à escrita de requisitos de natureza não-funcional. Nesse caso, não se realizaram restrições na forma, tendo sido descritas de modo livre. A descrição completa de cada *user story* — que inclui as variáveis versão, fonte, modo de verificação, data de identificação e estado — encontram-se no anexo “Lista de Requisitos”. Os requisitos que se encontram rasurados foram rejeitados ou adiados (podem ler-se as razões da rejeição ou postergação no já referido anexo).

As palavras-chave “*MUST*”, “*MUST NOT*”, “*SHOULD*”, “*SHOULD NOT*” e “*OPTIONAL*” presentes neste capítulo para descrever a prioridade dos requisitos devem ser interpretadas da mesma forma que são descritas no *RFC 2119*.



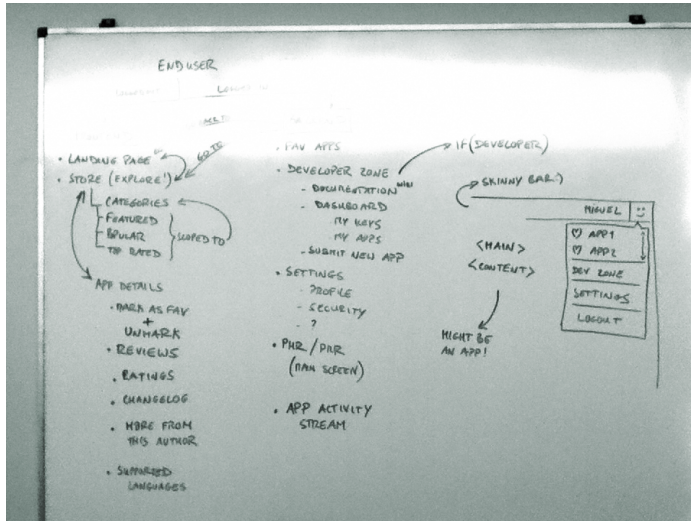


Figura 4.3: *Brainstorming* com a equipa sobre requisitos funcionais e de interface.

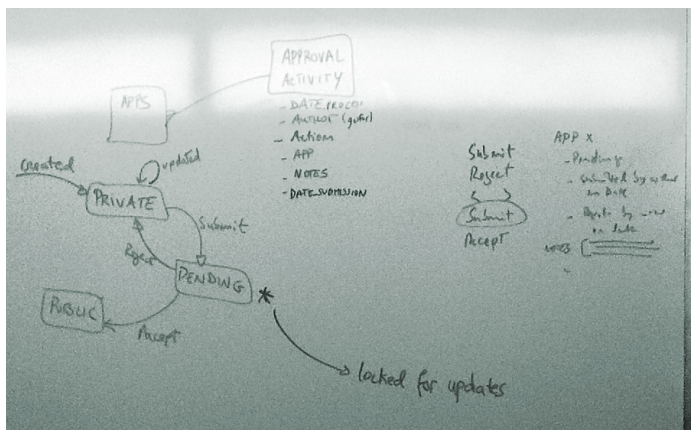


Figura 4.4: *Brainstorming* com o coordenador técnico sobre o fluxo de aprovação de aplicações.



Figura 4.5: Sessão para equipas técnicas do consórcio TICE.Mobilidade.



Figura 4.6: Apresentação das plataformas Appbase e One.Stop.Transport.

## Requisitos sobre Aplicações

User story	Resumo	Prioridade
APPS-USER-01	Listar todas as aplicações existentes	MUST
APPS-USER-02-WEB	Executar uma aplicação web	MUST
APPS-USER-03-MOB	<del>Instalar uma aplicação móvel</del>	OPTIONAL
APPS-USER-04-BIL	<del>Adquirir uma aplicação</del>	OPTIONAL
APPS-USER-05	Aceder à página de detalhes de uma aplicação	SHOULD
APPS-USER-06-OAUTH	Confiar dados a uma aplicação (com <i>OAuth</i> )	SHOULD
APPS-USER-07-SEARCH	<del>Pesquisar na plataforma</del>	OPTIONAL
APPS-DEV-01	Adicionar uma aplicação à plataforma	MUST
APPS-DEV-02	Editar uma aplicação do autor	MUST
APPS-DEV-03	Eliminar uma aplicação do autor	MUST
APPS-DEV-04	Listar aplicações do autor	MUST
APPS-ADMIN-01	Aprovar aplicações pendentes para revisão	SHOULD
APPS-ADMIN-02	Eliminar aplicações que violam regras	SHOULD
APPS-FAVS-USER-01	Adicionar uma aplicação aos favoritos	MUST
APPS-FAVS-USER-02	Listar aplicações favoritas	MUST
APPS-FAVS-USER-03	Remover aplicações dos favoritos	MUST

## Requisitos sobre Internacionalização

User story	Resumo	Prioridade
I18N-USER-01	Alterar a língua da plataforma	SHOULD
I18N-ADMIN-01	<del>Adicionar, remover e editar línguas pela interface web</del>	OPTIONAL
I18N-ADMIN-02	Adicionar, remover e editar línguas para o portal	OPTIONAL
I18N-ADMIN-03	Adicionar, remover e editar línguas para a interface de administração	OPTIONAL
I18N-NOTIF-01	Notificações na língua do utilizador	MUST

## Requisitos sobre Avaliações

User story	Resumo	Prioridade
RATE-USER-01	Escrever revisão crítica de uma aplicação web	SHOULD
RATE-USER-02	Atribuir entre 0 a 5 estrelas a uma aplicação web	SHOULD
RATE-USER-03	Reportar uma revisão crítica indevida	OPTIONAL
RATE-ADMIN-01	Eliminar revisões críticas (a aplicações) indevidas	OPTIONAL
RATE-ADMIN-02	Ler e eliminar situações reportadas por utilizadores	OPTIONAL

Requisitos sobre *Web Layers*

User story	Resumo	Prioridade
LAYERS-USER-01	Listar todos os <i>web layers</i> existentes	MUST
LAYERS-USER-02	Executar um <i>web layer</i>	MUST
LAYERS-USER-03-OAUTH	<del>Confiar dados a um <i>web layer</i> (com OAuth)</del>	OPTIONAL
LAYERS-USER-04-SEARCH	<del>Pesquisar na plataforma</del>	OPTIONAL
LAYERS-DEV-01	Adicionar um <i>web layer</i> à plataforma	MUST
LAYERS-DEV-02	Editar um <i>web layer</i> do autor	MUST
LAYERS-DEV-03	Eliminar um <i>web layer</i> do autor	MUST
LAYERS-DEV-04	Listar os <i>web layers</i> do autor	MUST
LAYERS-DEV-05	<i>Sandbox</i> para testar <i>web layers</i>	MUST
LAYERS-ADMIN-01	Aprovar <i>web layers</i> pendentes para revisão	SHOULD
LAYERS-ADMIN-02	Eliminar <i>web layers</i> que violam regras	SHOULD
LAYERS-FAVS-USER-01	Adicionar um <i>web layer</i> aos favoritos	MUST
LAYERS-FAVS-USER-02	Listar <i>web layers</i> favoritos	MUST
LAYERS-FAVS-USER-03	Remover <i>web layers</i> dos favoritos	MUST

## Requisitos sobre Contas de Utilizador

User story	Resumo	Prioridade
ACCOUNTS-USER-01	Editar perfil de utilizador	MUST
ACCOUNTS-USER-02	Autenticação no portal	MUST
ACCOUNTS-USER-03	Registo no portal	MUST
ACCOUNTS-USER-04	Expirar sessão após duas horas sem utilização	MUST
ACCOUNTS-DEV-01	Associar conta a uma conta de programador	MUST
ACCOUNTS-ADMIN-01	Remover e editar contas de utilizador através da interface web	SHOULD
ACCOUNTS-ADMIN-02	Desactivar o acesso de um utilizador	MUST
ACCOUNTS-ADMIN-03	Reactivar o acesso de um utilizador	MUST

## Requisitos sobre Programadores

User story	Resumo	Prioridade
DEV-01	Um programador é uma extensão do utilizador	MUST
DEV-02	Acesso à documentação para criação de aplicações e <i>web layers</i>	MUST
DEV-03	Obter e gerir chaves de acesso às API's da plataforma a partir da <i>dashboard</i> para programadores	MUST
DEV-04	Obter e gerir <i>token</i> e <i>secret</i> necessários à criação de <i>consumers OAuth</i> a partir da <i>dashboard</i> para programadores	MUST
DEV-05	Acesso aos guias de estilo	MUST

## Requisitos sobre Notificações

User story	Resumo	Prioridade
NOTIF-01	Notificação no portal quando acções persistem alterações	MUST
NOTIF-02	Enviar <i>feedback</i> sobre a plataforma	OPTIONAL
NOTIF-03	Notificação no portal quando aplicação muda de estado	MUST
NOTIF-04	Notificação por email quando aplicação muda de estado	SHOULD



Requisitos sobre Contentor de Aplicações e API's *JavaScript*

User story	Resumo	Prioridade
CONTAINER-01	Actualizar URL do portal ao navegar numa aplicação web	SHOULD
CONTAINER-02	As páginas da aplicação web podem ter dimensões dinâmicas e apresentar barras de deslocamento	SHOULD
CONTAINER-03	A aplicação ou <i>web layer</i> funciona em dispositivos móveis	SHOULD
CONTAINER-04	A aplicação web tem acesso aos dados pessoais de um utilizador mediante autorização	MUST
CONTAINER-05	Utilizar vários <i>web layers</i> em simultâneo	SHOULD
CONTAINER-06	Criar <i>web layers</i> que interajam com um mapa independentemente do sistema de mapas	MUST
CONTAINER-07	Aplicação web ou <i>web layer</i> possuem os métodos <i>JavaScript</i> de <i>debugging</i> usuais	SHOULD
CONTAINER-08	Aplicação web ou <i>web layer</i> acede a recursos externos	SHOULD
CONTAINER-09	Criar pontos de interesse no mapa dos <i>web layers</i>	MUST
CONTAINER-10	Escutar e registar eventos associados ao mapa comum	MUST
CONTAINER-11	Criar e fechar caixas geolocalizadas em <i>web layers</i>	MUST
CONTAINER-12	Aplicação web ou <i>web layer</i> pode interagir com outras aplicações ou <i>web layers</i> de forma pública ou privada	SHOULD
CONTAINER-13	Aplicação web ou <i>web layer</i> pode disponibilizar dados a terceiros que se queiram inscrever	SHOULD
CONTAINER-14	Aplicação web ou <i>web layer</i> pode inscrever-se a canais públicos ou privados registados por terceiros	SHOULD
CONTAINER-15	Aplicação web ou <i>web layer</i> pode cancelar subscrição a canal público ou privado registado por terceiros	SHOULD
CONTAINER-16	Localização geográfica do utilizador com autorização	OPTIONAL
CONTAINER-17	Aplicação web ou <i>web layer</i> conhece língua do utilizador	SHOULD
CONTAINER-18	Aplicação <i>packaged</i> ou <i>web layer</i> tem acesso aos metadados do arquivo comprimido	OPTIONAL
CONTAINER-19	Aplicação web tem acesso ao URL actual do portal	OPTIONAL
CONTAINER-20	Aplicação web ou <i>web layer</i> reconhece autenticação	SHOULD

CONTAINER-21	Aplicação <i>packaged</i> ou <i>web layer</i> gere dados no portal	SHOULD
CONTAINER-22	Aplicação <i>packaged</i> ou <i>web layer</i> oferece funcionalidades para segurança ao nível do <i>escaping</i> e <i>sanitization</i>	SHOULD
CONTAINER-23	Criar e apagar polígonos em <i>web layers</i>	SHOULD
CONTAINER-24	Suporte para <i>pushState</i> em aplicações web	SHOULD

### 4.3 Requisitos de sistema

ID	Título	Prioridade
SYS-01	<i>Logging</i> das exceções ocorridas na plataforma	SHOULD
SYS-02	API pública para programadores	SHOULD
SYS-03	Suporte para <i>Activity Streams</i>	OPTIONAL
SYS-04	Considerações para suporte futuro do modo <i>offline</i>	OPTIONAL
SYS-05	<i>Layout</i> fluído e responsivo	OPTIONAL
SYS-06	Usar ícones vectoriais para botões	OPTIONAL

### 4.4 Requisitos de desempenho

ID	Título	Prioridade
DES-01	Validações de formulários do lado do cliente	SHOULD
DES-02	Renderização de <i>templates</i> do lado do cliente	SHOULD
DES-03	Combinar ficheiros <i>JavaScript</i>	MUST
DES-04	Minificar ficheiros <i>JavaScript</i>	MUST
DES-05	Combinar folhas de estilo CSS	MUST
DES-06	Minificar folhas de estilo CSS	MUST
DES-07	Usar <i>image maps</i> para imagens de pequena dimensão	SHOULD
DES-08	Comprimir <i>JavaScript</i> , CSS e HTML com <i>Gzip</i>	SHOULD
DES-09	<del>Caching dos conteúdos estáticos</del>	OPTIONAL

## 4.5 Requisitos de segurança

ID	Título	Prioridade
SEG-01	Validações de formulários do lado do servidor	MUST
SEG-02	<del>Chapéus específicos dentro do papel Administrador</del>	OPTIONAL
SEG-03	Autorização de acesso a dados do utilizador	SHOULD
SEG-04	<i>Single sign-on</i> na plataforma e aplicações	SHOULD
SEG-05	Não permitir acessos não autenticados a recursos restritos	MUST
SEG-06	Encriptar palavras-chave na base de dados	MUST
SEG-07	Restringir o acesso a recursos externos por aplicações	SHOULD
SEG-08	<i>Logging</i> de certas acções realizadas na plataforma	SHOULD
SEG-09	<del>Transmissão de dados através de ligações seguras</del>	SHOULD
SEG-10	Protecção contra ataques <i>cross-site scripting</i> (XSS)	MUST
SEG-11	Protecção contra ataques <i>cross-site request forgery</i> (CSRF)	MUST
SEG-12	Protecção contra <i>SQL injection</i>	MUST
SEG-13	Obrigar à escolha de palavras-chave longas	SHOULD

## 4.6 Outras restrições técnicas

ID	Título	Prioridade
OTHER-01	<i>Deployment</i> fácil e automatizado	MUST
OTHER-02	O projecto deve ser facilmente extensível	MUST
OTHER-03	O projecto deve suportar um mecanismo de migrações de Base de Dados	SHOULD
OTHER-04	Garantir a recuperação do sistema em caso de falha de electricidade	OPTIONAL

## 4.7 Requisitos tecnológicos

Está disponível em anexo a especificação do *software* necessário para executar o projecto. Para uma lista completa das dependências do *back-end* e instruções de instalação deve consultar-se o ficheiro `requirements.txt`, na raiz do repositório de código do projecto. Todas as dependências do *front-end* estão incluídas nesse repositório.

## 4.8 Atributos de Qualidade do Sistema

Antes de se prosseguir para o próximo capítulo, onde será apresentada a visão da arquitectura, serão mencionados nesta secção de que forma se pretendem cumprir certos atributos de qualidade e objectivos arquitecturais importantes para o sistema a construir.

### 4.8.1 Extensibilidade e Interoperabilidade

Um dos objectivos da arquitectura consiste em garantir a sua extensibilidade. Uma vez que é esperado que o sistema tenha um tempo de vida longo, juntamente com o facto do período de estágio ser limitado, e de cobrir apenas uma parte do *scope* do que é esperado para a plataforma, este deve ser desenvolvido tendo em conta crescimento futuro. O esforço necessário para acomodar novas extensões, expandir o sistema com novas funcionalidades, ou modificar uma funcionalidade existente, deverá ser baixo e não exigir grandes mudanças na infra-estrutura do sistema. Estas capacidades adicionais são necessárias para evitar a obsolescência da plataforma e garantir a sua manutenibilidade.

Dada a natureza orientada a serviços da plataforma, é relativamente simples incluir novos *web services* que exponham as suas funcionalidades mediante uma interface externa. A própria plataforma tem as suas API's REST que oferecem parte das funcionalidades acessíveis através da interface gráfica. Através da utilização do projecto Piston para a criação desta interface, torna-se possível estender os formatos suportados (sendo que por *default* são suportados JSON, YAML, Pickle e XML). Ao ser um projecto Django, isolado em várias *Django Apps*, é possível adicionar ao projecto novas *Django Apps* e associar-lhe novas *routes*, sem ser necessário efectuar qualquer alteração ao código já existente.

Optou-se por modularizar o código *JavaScript* aplicando a *framework* Backbone.js e o sistema Require.js (para mais detalhes deve ler-se o anexo “Comparação de *frameworks client-side*”). Aplicou-se ainda a *design pattern* *Façade*, a qual fornece um interface uniforme a um conjunto de interfaces de um subsistema. Assim, ao garantir que cada módulo oferece as suas funcionalidades mediante um contrato explícito, torna-se simples utilizar os módulos existentes e substituí-los por outros com funcionalidades equivalentes.

### 4.8.2 Manutenibilidade

O sistema foi implementado prestando especial atenção a *coding standards* e convenções, a fim de escrever código manutenível. As normas seguidas são as sugeridas para as linguagens Python e CoffeeScript, e pela comunidade associada à *framework*

Django. Ao seguir *coding standards* e convenções conhecidas, aliados a uma boa documentação, não só se consegue garantir uma melhor manutenção do código produzido, como também melhorar a legibilidade, tornar o *refactoring* mais eficiente, mais fácil *debugging* e implementação, e melhor portabilidade. Foi também seguido um conjunto de boas práticas existentes no IPN no que diz respeito à escrita de mensagens de *commit* de código. Por último, aplicou-se a especificação *Semantic Versioning*<sup>1</sup> para atribuir versões às *releases* parciais, não só da API da plataforma, como também da interface web.

O código do *back-end* foi estruturado de acordo com a *design pattern* MVC, uma vez que encoraja os *developers* a dividirem o seu código em blocos auto-contidos e *loosely-coupled*, com objectivos bem definidos – desde gestão de dados, passando por lógica de negócio e código relacionado com a interface de interacção com o utilizador.

Por último, no que diz respeito ao *front-end*, foi criado e programado um *Interface Style Guide*. Os guias de estilo permitem definir directrizes e princípios que são essenciais para produzir um design visual uniforme e manter o controlo de requisitos de interface gráfica de interacção com o utilizador. Para permitir, não só facilitar a manutenção deste guia, como também a manutenção do código de interface de interacção com o utilizador, aplicou-se uma metalinguagem designada *Less* (a qual é compilada para CSS). Esta permite-nos definir métodos e variáveis importantes para manter um aspecto uniforme da aplicação, através da definição da palette de cores, famílias de fontes, dimensões, entre outros.

### 4.8.3 Permutabilidade e Modularidade

A plataforma web faz uso de várias *packages* e *apps* Django e bibliotecas Python. Estas são aplicações que contêm os arquivos e informações necessários para serem instaladas e executadas no sistema. São de grande importância, uma vez que podem facilmente ser usadas para estender ou modificar a funcionalidade dentro de projectos Django. Isto torna mais fácil o *deployment* do projecto e são uma boa forma de reutilização de código. Há *packages* que encapsulam funcionalidades como auditoria, versionamento de modelos, entre outros. Estes módulos são autónomos e podem ser facilmente substituídos por outros de funcionalidade equivalente.

Ao garantir-se a separação do código do servidor e do cliente, e ao seguir-se uma especificação conhecida e bastante suportada pela comunidade para o sistema de *templates*, torna-se simples reutilizar as vistas existentes para projectos em que o *back-end* está escrito noutra linguagem. Tal como referido na secção “Extensibilidade e Interoperabilidade”, os projectos Require.js e Backbone.js têm também um papel importante na modularização do código do cliente.

---

<sup>1</sup>Esta especificação está acessível em: <http://semver.org/>.



# Capítulo 5

## Arquitectura e Design

With good program architecture debugging is a breeze, because bugs will be where they should be.

— *David May, CTO da XMOS Semiconductor*

Neste capítulo é formalizada a arquitectura do sistema de acordo com várias perspectivas. É igualmente descrita a lógica, os pressupostos, e as implicações das decisões que foram tomadas. Esta, não só reflecte o que foi aprendido com o Estado da Arte, como visa suportar os requisitos levantados. Por último, é exposta a solução implementada, a qual foi inicialmente validada, de forma parcial, através da criação de um protótipo de alto-nível.

### 5.1 Problema de Engenharia

No âmbito deste estágio é pretendido resolver-se o problema de disponibilizar, gerir e utilizar aplicações web desenvolvidas por entidades distintas num só local. Para isso, será construída uma plataforma web orientada a serviços em que seja possível agregar e integrar novas aplicações, e que ofereça ao utilizador final uma interface unificada. Desta forma, e de um modo geral, foi definido que seria necessário que:

- Programadores pudessem adicionar e gerir aplicações na plataforma;
- Utilizadores finais pudessem utilizar essas aplicações na plataforma, sem ser necessário efectuar um registo para cada uma dessas aplicações, através de um mecanismo de *single sign-on*;
- As aplicações pudessem fazer uso de API's REST da plataforma e dos projectos TICE e de dados dos utilizadores, mediante a sua permissão;
- As aplicações pudessem fazer uso de API's *JavaScript* para interacção com o portal, outras aplicações e utilizadores<sup>1</sup>.

---

<sup>1</sup>API's *JavaScript* para obtenção da língua do portal, comunicação com servidores externos, *escaping* de código HTML, gestão de preferências do utilizador, persistência de dados, etc.

Para compreender as decisões tomadas na definição da arquitectura do sistema, é necessário começar por descrever os tipos de aplicações que se decidiu definir e explicar essas mesmas decisões.

## 5.2 Tipos de aplicações

A plataforma permite a publicação de aplicações web. Apesar de ser este o foco de trabalho, era requisito facultativo para o estágio o suporte para aplicações móveis nativas<sup>2</sup>. Neste último caso, a plataforma serviria apenas para disponibilizar uma ligação para a aplicação. Não obstante ter sido construída com vista a suportar evolução, a prioridade esteve sempre nas aplicações web, para as quais se decidiu especificar dois formatos:

**Packaged** Executam inteiramente no *browser* e a sua lógica de negócio está escrita em JavaScript. Podem fazer uso das novas capacidades dos *browsers* relacionadas com o HTML5, como o funcionamento em modo *offline*. Estas consistem num arquivo que segue a especificação de *widgets* do W3C (analisada no capítulo “Estado da Arte”).

**Hosted** Acessíveis remotamente, alojadas no exterior da plataforma, estas aplicações não têm qualquer limitação ao nível da linguagem de programação ou tecnologias aplicadas. O modo como se processará a sua integração na plataforma será descrita na secção “Contentor para integração das aplicações web *hosted*” no subcapítulo “Descrição dos restantes módulos”.

A necessidade de suportar dois tipos de aplicações deve-se ao facto de, embora as aplicações *packaged* sejam mais responsivas, rápidas, e interactivas, estas envolverem um modelo de programação distinto. Para além disso, o facto de serem escritas inteiramente em JavaScript pode ser limitativo em certos contextos.

Para o utilizador final, esta distinção deve ser imperceptível. Na sua perspectiva, as aplicações distinguem-se-ão entre aplicações web, móveis e *web layers*:

**Web Layers** Os *web layers* são aplicações *packaged* criadas especialmente para o projecto TICE.Mobilidade. Estas diferem das restantes por utilizarem um mapa geográfico comum, com o qual interagem. Por exemplo, o utilizador pode aplicar um *web layer* que apresente paragens de autocarros e outro que lhe permita calcular distâncias entre pontos. Ambos *web layers* interagem com o mesmo mapa, naquela localização específica escolhida pelo utilizador.

---

<sup>2</sup>Estão disponíveis em anexo mais informações sobre este requisito (APPS-USER-03-MOB).



A **tabela 5.1** apresenta um resumo dos benefícios e inconvenientes dos referidos formatos. Para mais informações sobre os vários tipos de aplicações, suas vantagens e limitações, deve ler-se a secção “Tipos de aplicações”, na “Introdução à plataforma de desenvolvimento” do “Guia do Programador”.

Tabela 5.1: Comparação entre aplicações *Hosted* e *Packaged*

Critério	Hosted App	Packaged App	Web Layer
Alojamento	A cargo do programador	Na plataforma Appbase	Na plataforma Appbase
Paradigma tecnológico	Qualquer linguagem e tecnologia	<i>JavaScript</i> executado no <i>browser</i>	<i>JavaScript</i> executado no <i>browser</i>
Componente servidor	Obrigatória <sup>(a)</sup>	Opcional	Opcional
Comunicação com portal e aplicações	Suportada, com limitações <sup>(b)</sup>	Suportada	Suportada
Persistência de dados	A cargo do programador	Suportada	Suportada

(a) A aplicação tem de estar hospedada em servidores externos à plataforma.

(b) A comunicação é processada de forma segura em *browsers* recentes, no entanto, o *fallback* aplicado para *browsers* mais antigos (Internet Explorer  $\leq 7$ , Mozilla Firefox  $\leq 2.0$ , Safari  $\leq 3.2$ , Opera  $\leq 9$ ) é inseguro, não devendo por isso ser utilizado para transmissão de dados sensíveis.

## 5.3 Vistas da Arquitectura

Nas próximas secções será descrita a arquitectura do sistema desenvolvido segundo dois níveis de abstracção — o *nível 0*, que corresponde a uma perspectiva geral do sistema, em que é apresentada a interacção com componentes externos, e o *nível 1*, em que se descreve a plataforma de uma forma mais detalhada, com todos os seus sub-componentes internos e suas interacções.

### 5.3.1 Visão de nível 0

A plataforma é baseada numa arquitectura orientada a serviços, dada a sua natureza distribuída e extensível. Esta aumenta a modularidade incentivando a divisão do sistema em serviços independentes e desacoplados.

## Perspectiva de *deployment*

De um ponto de vista físico existe um servidor acessível ao público pela rede, que fornece acesso à interface, funcional não só em computadores pessoais, como em *tablets* e *smartphones* (figura 5.1). Este executa o código da plataforma e comunica com as seguintes entidades:

**Servidor de *widgets*** A plataforma comunica com um servidor de *widgets* para a gestão de aplicações do tipo *packaged*. Pelo facto de se seguir um *standard* aberto, foi possível utilizar uma das implementações *open source* existentes.

**Servidor para geração de ambientes isolados** Uma vez que as aplicações devem interagir entre si e com o próprio portal, é necessário que o seu código possa ser executado em ambientes isolados que não comprometam a segurança dos utilizadores da plataforma. Para tal, o portal comunica com um servidor que permite a geração desses ambientes isolados.

**Servidor *OAuth*** É requisito que seja possível proceder à autorização e autenticação de utilizadores e aplicações. Desta forma, uma aplicação pode pedir permissão ao utilizador para, por exemplo, ter acesso aos seus dados pessoais e, por outro lado, a aplicação pode requisitar acesso à plataforma para aceder a certos recursos e API's. Permite também que o utilizador se registe apenas no portal, e não em cada uma das aplicações presentes, funcionando como um mecanismo de *single sign-on*.

**Servidor de eventos** Para que seja possível que as aplicações comuniquem entre si, e que tal possa acontecer entre utilizadores e sessões distintas, é necessária a existência de um servidor que possibilite essa comunicação em tempo real.

**Servidor de gestão base de dados** A plataforma Appbase tem a sua base de dados no servidor DBMS da plataforma One.Stop.Transport. O mesmo acontece para o servidor de *widgets* e para o servidor *OAuth*.

## Perspectiva estática

Ainda numa abstracção de alto-nível, de um ponto de vista estático, a arquitectura do sistema pode dividir-se nos seguintes grandes módulos:

**Portal web e interface de administração/gestão** Existe um portal web para utilizadores e programadores, e uma interface web de administração para gestores e administradores da plataforma.

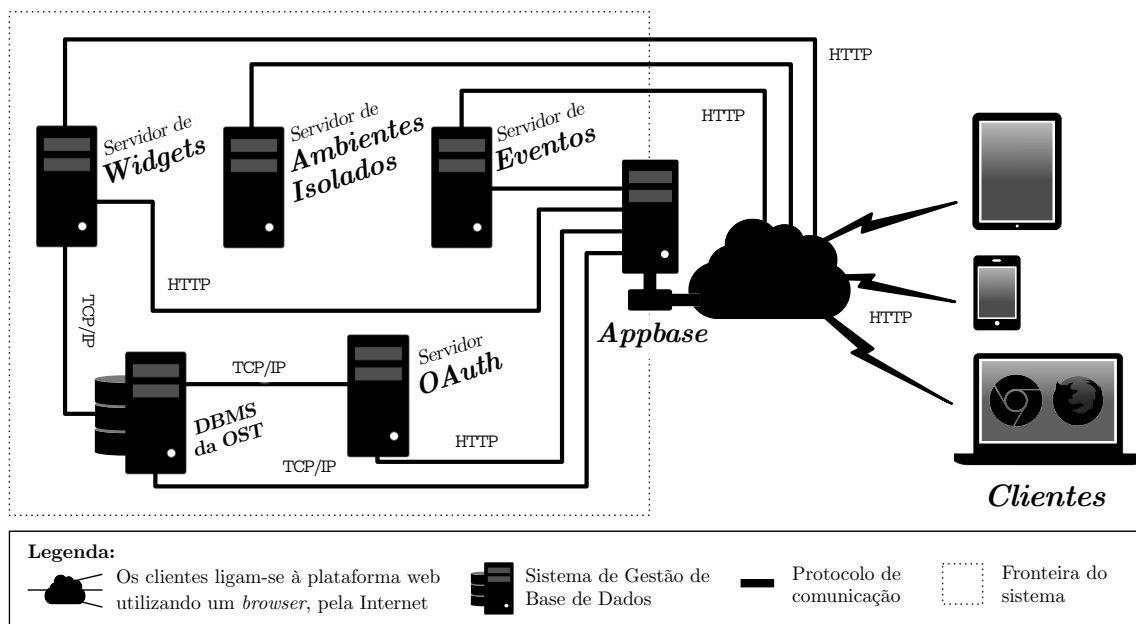


Figura 5.1: Visão de nível 0 da perspectiva de *deployment* da arquitectura.

**API's REST da plataforma Appbase** A plataforma oferece também as suas capacidades de gestão de aplicações e utilizadores mediante API's REST. As interfaces podem ser implementadas via REST ou SOAP. Contrariamente ao SOAP, que se trata de um protocolo de comunicação, o REST é uma *pattern* arquitectural baseada na troca de mensagens HTTP. Apesar de existir um largo suporte para serviços SOAP, as arquitecturas *RESTful* são tendencialmente mais leves, simples e fáceis de construir. Para além disso, o SOAP está limitado ao formato XML enquanto no REST não existem restrições no formato da resposta (que pode ser, por exemplo, HTML, XML, ou JSON) [46]. Foram construídas duas API's — de utilizadores e de aplicações — que permitem realizar as operações CRUD. Estas podem ser facilmente estendidas no futuro, sempre que tal seja necessário.

**Módulos de permissões, *logging* e *reporting*** Estava planeada a integração com um módulo de permissões, para permissões granulares ao nível dos administradores e gestores, e de *logging* e *reporting*. Ambos módulos serão comuns a vários sub-projectos TICE (o estagiário ficaria responsável apenas pela integração). De notar que, como é descrito no anexo “Lista de Requisitos” (mais especificamente nos requisitos SYS-01 e SEG-02), o módulo de permissões e de *logging* e *reporting* não foram implementados a tempo de serem integrados durante o período de estágio.

A **figura 5.2** apresenta a visão de alto-nível da vista estática, com a representação das ligações a entidades externas e a bases de dados. De notar que, apesar do utilizador final apenas aceder ao *link* da interface web, na prática, o seu *browser*

comunica internamente de forma directa com o servidor de eventos, de *widgets*, de ambientes isolados, e com as aplicações externas. A ligação efectuada com o servidor de eventos é feita através de *web sockets*, que consiste numa conexão TCP/IP, ou por pedidos *Ajax*, caso os *web sockets* não sejam suportados pelo *browser* do utilizador. As aplicações comunicam com o portal através do método *postMessage*, que permite a comunicação entre *frames* de uma página HTML através de JavaScript (não envolve por isso pedidos pela rede). Por último, algumas ligações do tipo HTTP serão no futuro substituídas por HTTPS.

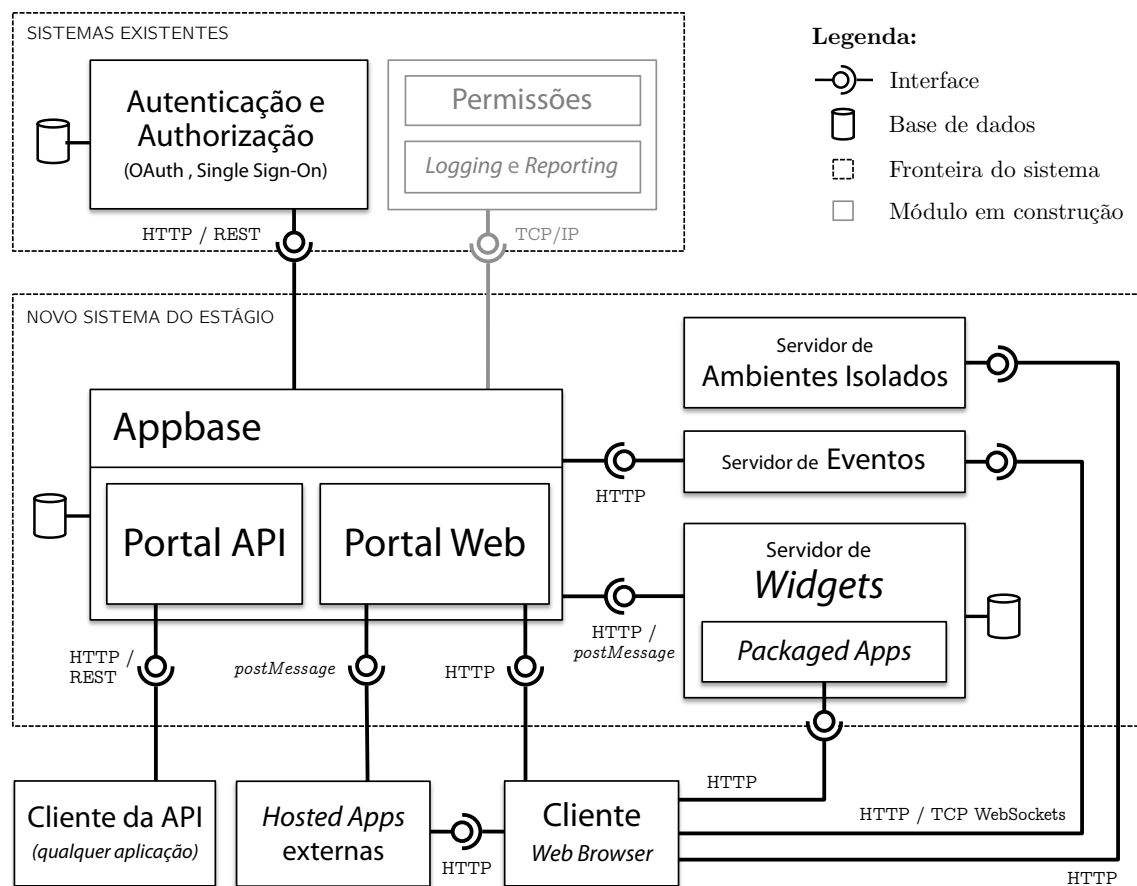


Figura 5.2: Vista estática da arquitectura, com ligações a entidades externas e bases de dados. Como referido na vista física, as bases de dados têm um DBMS comum.

### 5.3.2 Visão de nível 1

#### Perspectiva estática

A **figura 5.3** apresenta apenas a componente Appbase com maior granularidade. Os restantes servidores com que comunica são representados num único módulo “Outros servidores”, e os sistemas existentes não estão caracterizados. A descrição dos vários

módulos será feita de forma pormenorizada na secção “Desenho da Solução”, em que se apresentam as tecnologias aplicadas para cada componente da arquitectura.

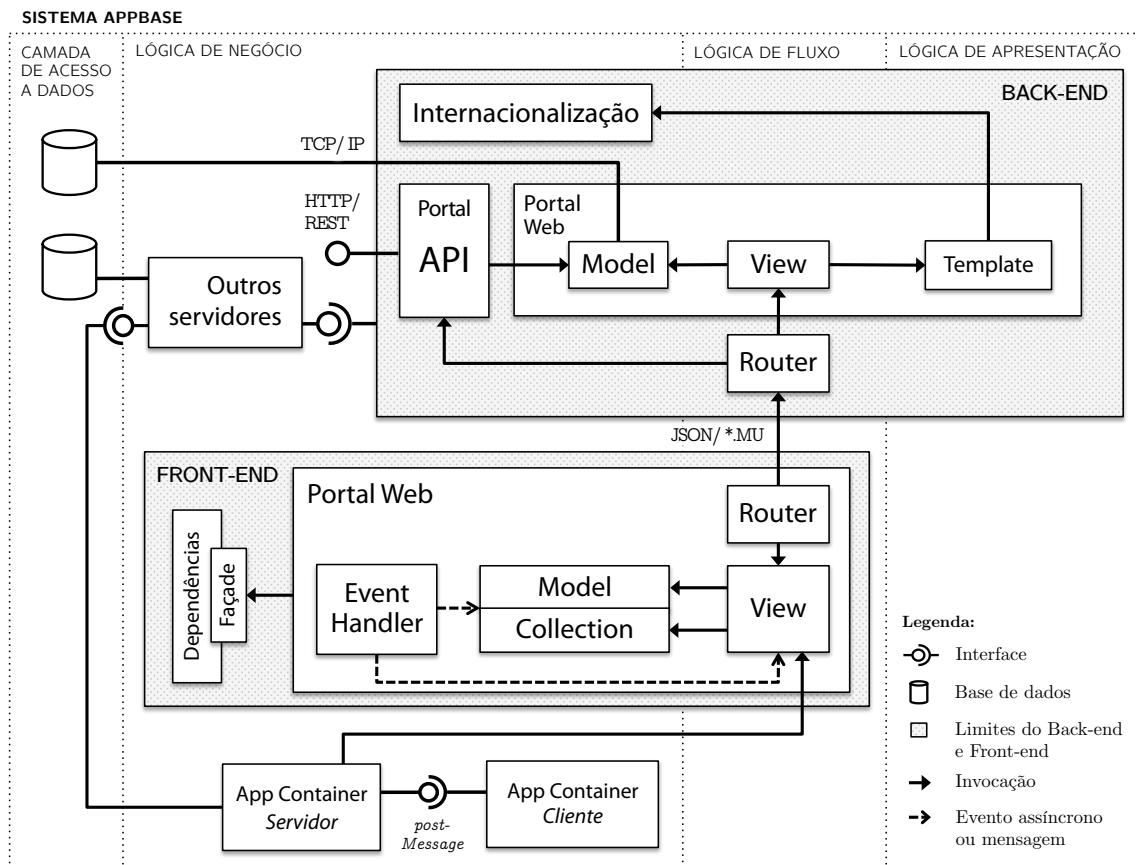


Figura 5.3: Visão de alto-nível da perspectiva estática da arquitectura.

## Perspectiva dinâmica

Não obstante a importância da perspectiva estática da arquitectura, a natureza desta plataforma requer uma análise do ponto de vista da execução. Será por isso feita, nos próximos parágrafos, uma descrição de alguns dos mecanismos mais importantes, sem descer aos detalhes técnicos inerentes à sua implementação.

**Interação entre aplicações e portal através de API's *JavaScript*** De forma a garantir a interação inter-aplicação e com o próprio portal, foram desenvolvidos vários módulos *JavaScript*<sup>3</sup>. Estes expõem publicamente apenas parte dos seus métodos, aplicando a *design pattern Façade*. Esses módulos são API's *JavaScript* que foram documentadas e podem ser utilizadas pelos *developers* nas suas aplicações.

<sup>3</sup>O conceito de módulo ou classe não existe em *JavaScript*, embora possa ser simulado. Para garantir a manutenibilidade do código usou-se uma linguagem que compila para *JavaScript* chamada *CoffeeScript*, que introduz este conceito (a compilação é feita previamente e não em *runtime*).

Cada tipo de aplicação (*hosted*, *packaged* e *web layer*) tem as suas especificidades técnicas e, por isso, a cada uma estão associados contentores, *scripts* implementados de forma a garantir o acesso correcto às API's. Tal é transparente para o *developer* (excepto nas aplicações *hosted*, nas quais precisa de importar o contentor, um ficheiro *JavaScript*, manualmente). A solução implementada é extensível, sendo fácil adicionar novas API's *JavaScript*. Foram construídas várias API's, apresentadas na **tabela 5.2** (estão disponíveis mais informações no anexo “Guia do Programador”).

Tabela 5.2: Resumo dos objectivos gerais de cada API *JavaScript*.

<i>Inter-App Communication</i>	Oferece um mecanismo de comunicação entre aplicações através de um modelo <i>Publish-Subscribe</i> . Um produtor (aplicação) pode partilhar dados de forma pública — com outros utilizadores desta ou de outras aplicações — ou privada — com todas as sessões de um único utilizador desta ou de outras aplicações. Tanto nos canais públicos como privados, o produtor pode especificar quais são os receptores autorizados. Nos canais públicos, o utilizador, que pode ser anónimo (não autenticado), é solicitado a autorizar ou rejeitar o canal quando a aplicação tenta efectuar o seu registo. Nos canais privados, o utilizador pode ter sessão iniciada em múltiplos <i>browsers</i> e dispositivos, e é solicitado a autorizar ou rejeitar o canal quando a aplicação se tenta inscrever a este.
<i>Remote Communication &amp; Debugging</i>	Conjunto de funções que tornam mais simples o <i>debugging</i> e permitem comunicação assíncrona com recursos externos.
<i>Widget Properties</i> (inclui internacionalização)	Permite o acesso às propriedades que definem a aplicação. Tal inclui os metadados definidos no ficheiro de configuração da aplicação, e propriedades relacionadas com a execução da aplicação no portal (como por exemplo, a língua em que está configurado o portal). Esta API segue a especificação <i>Widget Interface</i> <sup>4</sup> definida pelo W3C e é parcialmente implementada pelo Apache Wookie <sup>5</sup> .
<i>Widget Extensions</i> (inclui utilizadores)	Possui alguns métodos que estendem as capacidades das aplicações. Uma das funções mais importantes é a <code>userIsLoggedIn</code> , que devolve um booleano conforme um utilizador esteja ou não autenticado no portal. De notar que os dados do utilizador são protegidos por <i>OAuth</i> . Desta forma, para obter dados pessoais do utilizador autenticado é necessária a sua permissão.

<sup>1</sup>Mais informação disponível em: <http://w3.org/TR/widgets-apis/#the-widget-interface>

<sup>2</sup>Os métodos para interacção com portal não fazem parte da implementação do *Apache Wookie*.

<i>Widget Preferences</i> (inclui persistência de dados)	Permite a manipulação de uma zona de persistência de dados única da instância da aplicação. Esta permite que, por exemplo, se guardem configurações da aplicação, personalizáveis por cada utilizador. Esta API segue a especificação <i>Widget Interface</i> definida pelo W3C e é implementada pelo Apache Wookie.
<i>Wookie Utilities</i>	Funções utilitárias que ajudam na actualização dinâmica da página web com conteúdos provenientes de fontes externas, como servidores ou o input do utilizador. Esta API é implementada pela biblioteca <i>Direct Web Remoting</i> <sup>6</sup> .
<i>Maps</i>	Esta não se trata de uma única API, mas de várias: <i>Map</i> , <i>Marker</i> , <i>Event</i> , <i>InfoWindow</i> e <i>Polygon</i> . Recorreu-se à <i>design pattern Bridge</i> , na qual se separa uma abstracção da sua implementação, de forma a que ambas possam variar de forma independente. Neste caso, trata-se de uma camada sobre a API do <i>Google Maps</i> . Assim, garante-se que os <i>web layers</i> apenas utilizam as funções desejadas, e pode manter-se a mesma API para os <i>developers</i> , mesmo se no futuro escolhermos outra solução alternativa ao <i>Google Maps</i> (está previsto no projecto, após o período de estágio, migrar-se do <i>Google Maps</i> para <i>Open Layers</i> e <i>Open Street Maps</i> ).

**Execução das aplicações *Hosted*** As aplicações do tipo *Hosted* são carregadas no contexto da plataforma Appbase através de um contentor (designado por *Application Container*). Este contentor inclui um pequeno *script JavaScript* que deve ser incluído na aplicação (designado por *Application Container Client*). Este permite, por *default*, uma contextualização mais transparente com a plataforma, nomeadamente no que diz respeito aos factores sumariados na **tabela 5.3**.

Tabela 5.3: Vertentes do contentor de aplicações *Hosted*.

Dimensões dinâmicas	Permite a construção de <i>layouts</i> fluídos e responsivos, que se adequem a ecrãs de menores dimensões, como os de <i>smartphones</i> e <i>tablets</i> . Pretende evitar que a aplicação apresente uma barra de <i>scroll</i> horizontal e/ou vertical (isto é, se uma barra de <i>scroll</i> for necessária, deverá aparecer ao nível do portal).
---------------------	---

<sup>1</sup>Mais informação disponível em: [directwebremoting.org/dwr/documentation/browser/util](http://directwebremoting.org/dwr/documentation/browser/util)

Modificações no endereço URL	<p>À medida que o utilizador percorre a aplicação, isto é, carrega em <i>links</i> que o encaminham para novas páginas, o URL do portal é actualizado em conformidade. A <b>figura 5.4</b> pretende exemplificar uma aplicação que vive no endereço <code>http://myapp.com</code> e cujo ID é <code>example-app</code>. No exemplo, o utilizador carrega num <i>link</i> relativo que o encaminha para o endereço <code>/page</code> (este mecanismo funciona igualmente para caminhos absolutos). O URL do portal é alterado em conformidade, sendo adicionado o sufixo <code>/page</code> ou <code>/#/page</code>. A distinção deve-se ao facto de certos <i>browsers</i> não permitirem a modificação do URL da página visualizada sem que, para isso, seja feito um pedido ao servidor e recarregada a página completamente. De referir que, se aceder directamente pelo seu <i>browser</i> a qualquer um dos dois URL's, ambos funcionarão correctamente, em qualquer <i>browser</i>, pois tal é assegurado pelo servidor.</p>
Invocação das API's <i>JavaScript</i> através do contentor	<p>Ao incluir-se o cliente do contentor tem-se acesso às <i>API's JavaScript</i> em aplicações do tipo <i>Hosted</i>. Desta forma, apesar das aplicações <i>Hosted</i> serem externas à plataforma, podem interagir com o portal e outras aplicações, da mesma forma que o fariam se fossem executadas de forma embebida.</p>

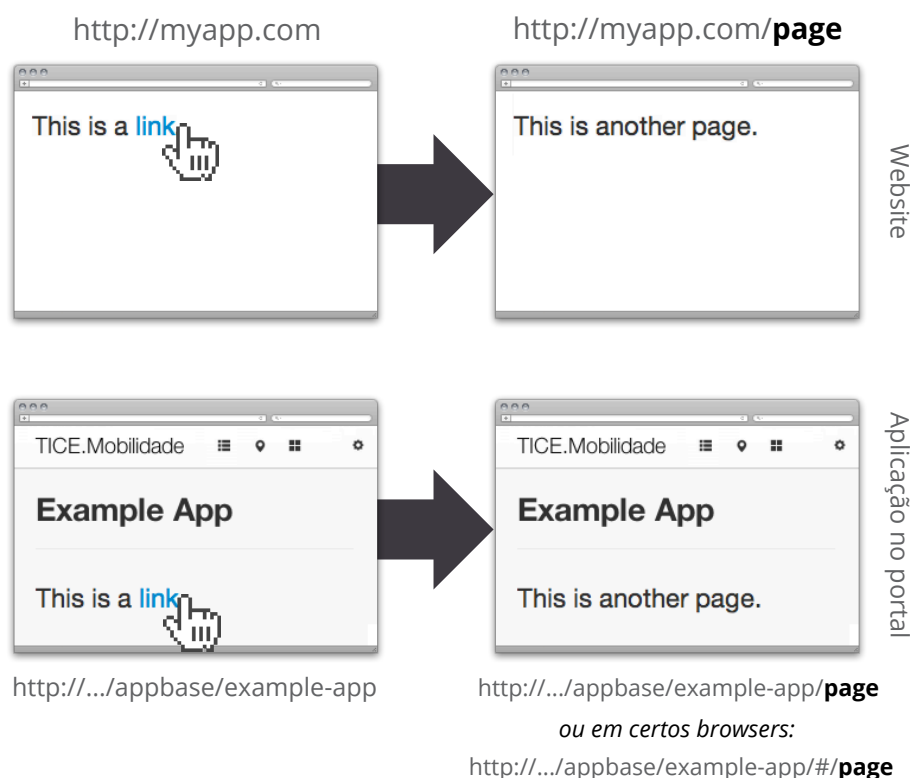


Figura 5.4: Gestão de mudança de URL pela plataforma Appbase.



**Deploy e execução das aplicações *Packaged* e *Web Layers*** O processo de execução de aplicações *packaged* e *web layers* é descrito na figura 5.5.

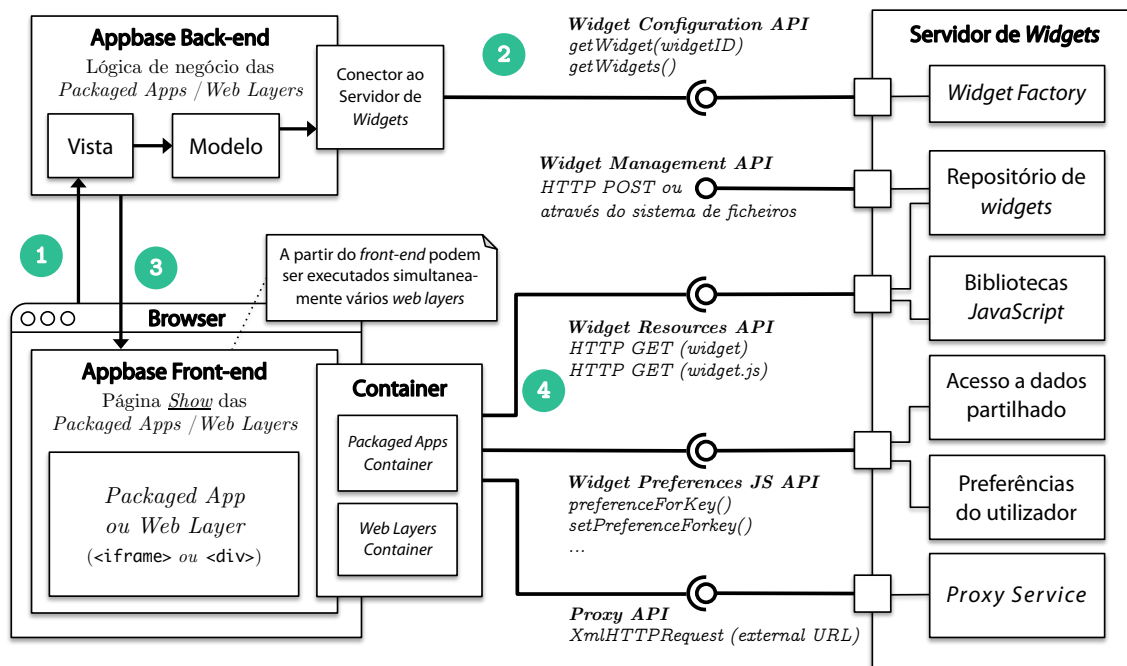


Figura 5.5: Execução de uma aplicação *packaged* ou *web layer*. Inspirado num diagrama de arquitectura do projecto Apache Wookie [5].

Ao aceder à página de uma aplicação deste tipo 1 é executada a vista correspondente que, por sua vez, comunicará através do modelo com o servidor de *widgets* 2. O servidor de *widgets* instancia o *widget* e devolve o URL da aplicação. Neste momento é possível apresentá-la ao utilizador 3. Ao ser carregada, são obtidos os recursos associados 4 necessários à execução das APIs *JavaScript*. O processo de *deployment* de aplicações é mais simples, sendo apenas necessário efectuar um pedido POST ao servidor de *widgets* com o arquivo comprimido do *widget* figura 5.6.

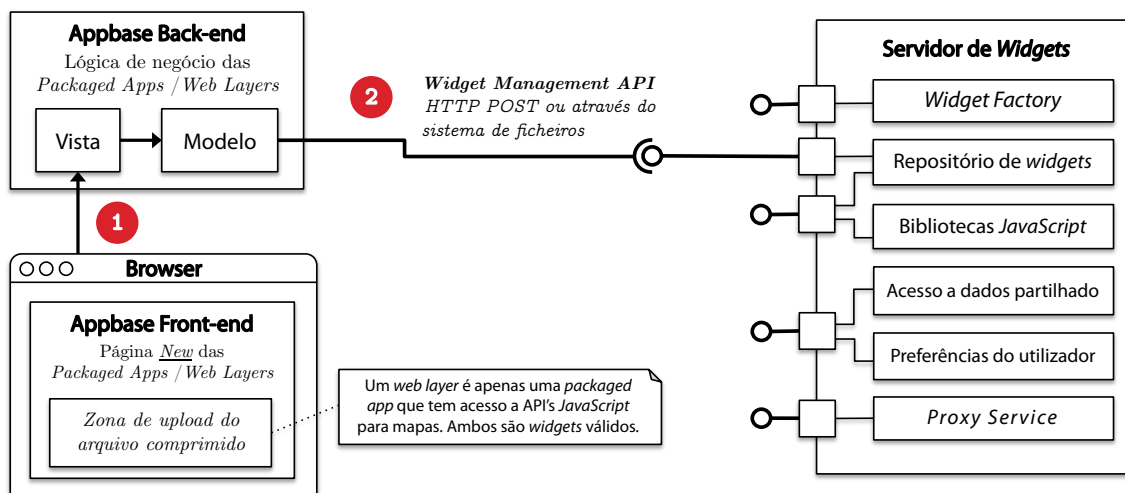


Figura 5.6: *Deploy* de uma aplicação *packaged* ou *web layer*.

**API *JavaScript* de comunicação entre aplicações** De forma a suportar comunicação através de canais privados, foi necessário definir um fluxo que garantisse a autenticidade do utilizador. Inspirado na forma como o serviço *web Pusher* disponibiliza um mecanismo semelhante [47], foi definido o seguinte processo:

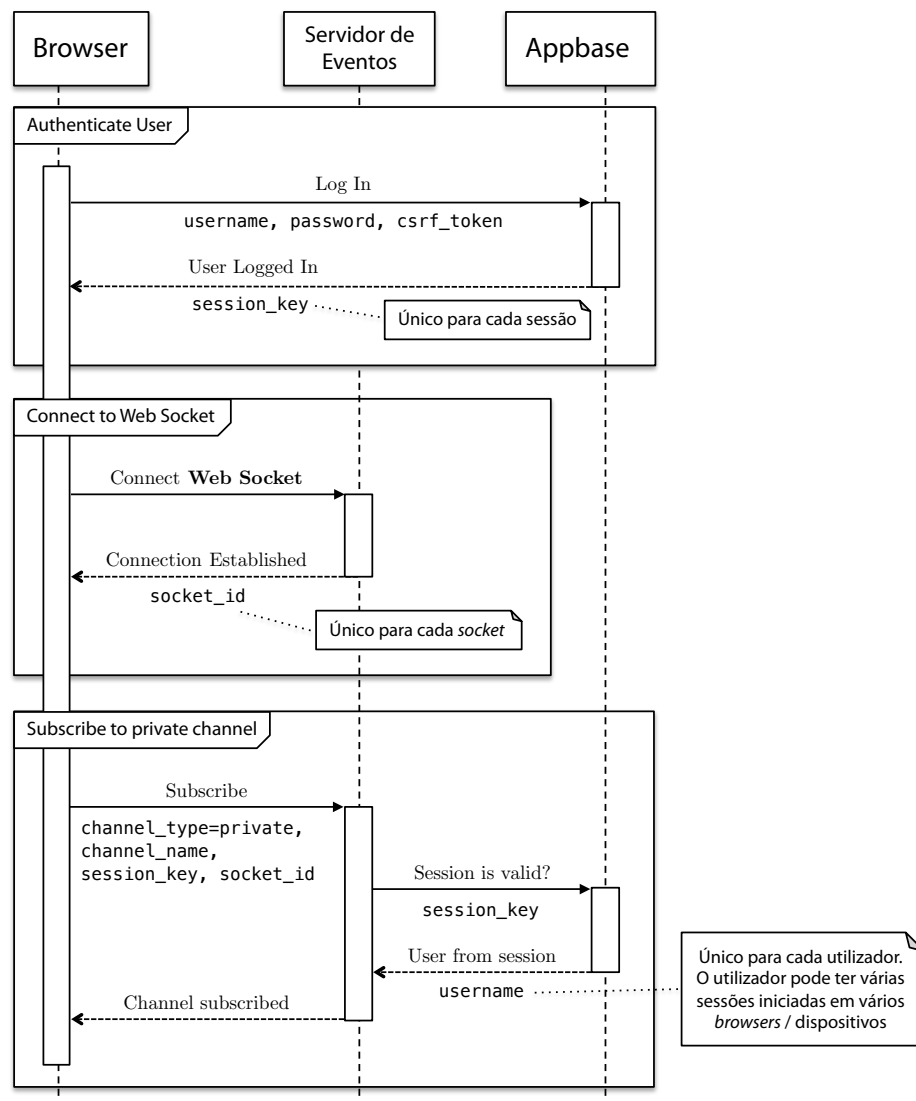


Figura 5.7: Mecanismo de protecção de canais de comunicação privados.

Em primeiro lugar é necessário que o utilizador esteja autenticado no portal para que possa fazer uso de canais privados. Ao fazer *login*, é gerada uma chave para a sessão do utilizador. Posteriormente, assim que este executa o primeiro *web layer*, uma ligação ao servidor de eventos é estabelecida. Essa ligação é unicamente identificada pelo seu *socket ID*. De seguida, ao ser feito *subscribe* a um canal privado, a chave de sessão do utilizador permite, caso seja válida, identificar o utilizador. A partir desse momento, fica subscrito ao canal privado. De notar que esta é primeira versão deste sistema de canais, podendo este fluxo ser adaptado no futuro.

**Fluxo de autorização de acesso a dados** A plataforma simplifica o processo de registo e *login* em aplicações através da utilização das contas de utilizadores da própria plataforma. Desta forma, os utilizadores não necessitam de preencher um formulário de registo para cada aplicação que utilizam, de escolher novos nomes de utilizadores, ou de memorizar várias palavras-chave. A plataforma suporta autenticação e autorização através dos protocolos OAuth 1.0 e OAuth 2.0, à semelhança do que acontece com o *Facebook Connect* ou o *Sign up with Twitter*.

O protocolo OAuth permite a partilha de recursos privados armazenados num *website* (neste caso a plataforma) com outro *website* (neste caso uma aplicação web ou móvel), sem que para isso o utilizador tenha de fornecer credenciais de acesso ao segundo *website*. O mesmo processo aplicado para este tipo de autorização é feito para a autenticação — o utilizador, ao autenticar-se perante uma aplicação, está tecnicamente a autorizar a aplicação a que tenha acesso a dados do seu perfil de utilizador. Desta forma, o utilizador final não só desfruta de uma experiência mais segura, como também mais usável.

A explicação feita acima é válida tanto para o protocolo OAuth 1.0, como para a sua nova versão, 2.0. O trabalho do estagiário neste âmbito foi o de criar a lógica necessária à autorização por parte do utilizador. A **figura 5.8** apresenta o fluxo, onde o estagiário programou as relações estabelecidas com a entidade Appbase.

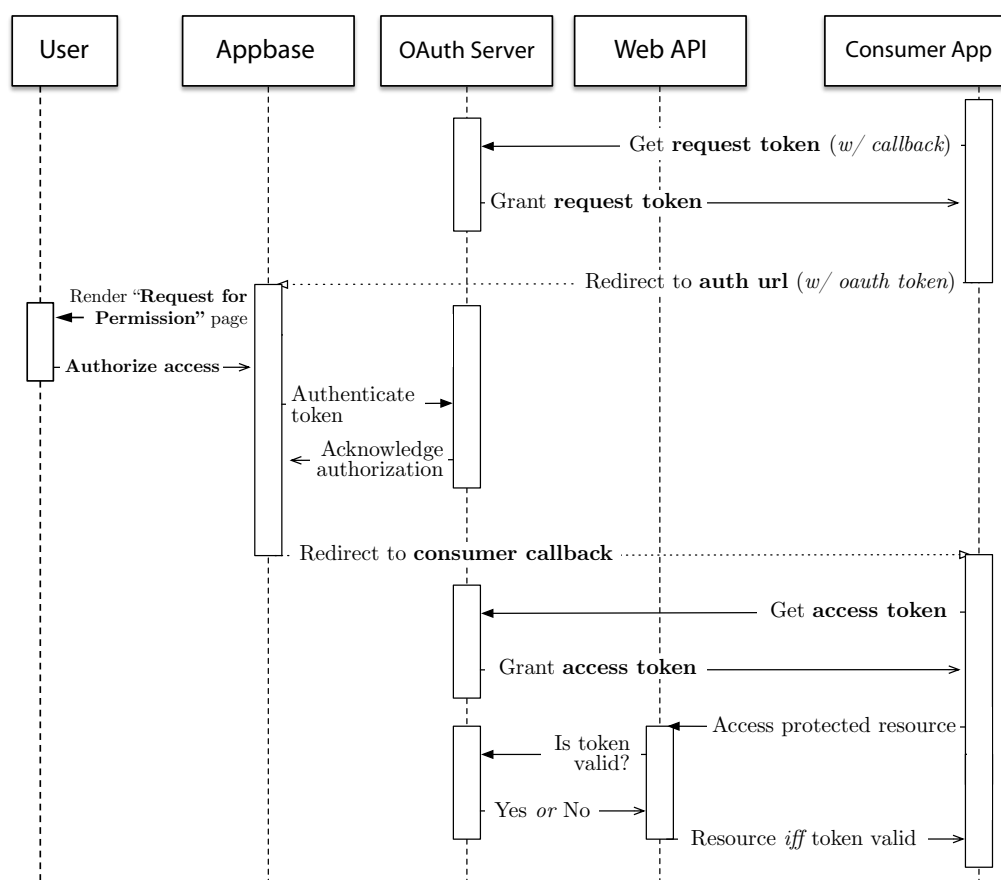


Figura 5.8: Interação entre actores no fluxo do OAuth 1.0.

**Fluxo de aprovação de aplicações** Por último, será descrito de forma breve o fluxo de aprovação de aplicações pelos gestores da plataforma. Ao se adicionar uma aplicação, esta fica apenas acessível ao seu autor. É preciso que este a submeta para aprovação, e que esta seja aprovada, para que fique disponível publicamente. A aplicação não é apagada pela equipa de gestão, mesmo que seja rejeitada. O *developer* pode efectuar alterações à aplicação e submetê-la novamente para aprovação. Enquanto a nova versão não for aprovada, os seus utilizadores continuam a ter acesso à versão anterior. A **figura 5.9** apresenta os estados em que uma aplicação pode estar (dá-se o nome *draft* a uma versão nova de uma aplicação pública).

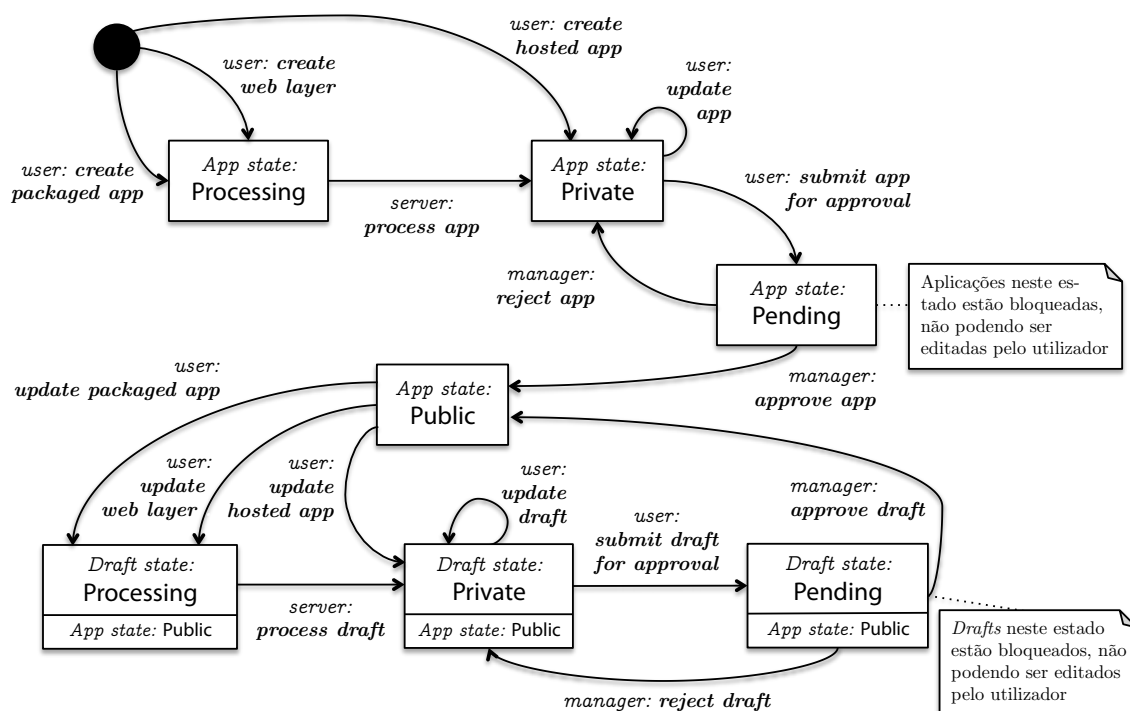


Figura 5.9: Fluxo de aprovação de aplicações.

**Memória dos Web Layers** Um desafio interessante dos *web layers* foi o de conseguir que, independentemente das acções realizadas por um *web layer* no mapa, fosse possível desfazer essas acções quando o *web layer* é fechado pelo utilizador. Não é aceitável que tal fique a cargo do *developer*. Uma vez que todas as interacções com o mapa são feitas pelas API's *JavaScript*, aplicou-se um sistema que segue a *design pattern Command*. Cada chamada à API é um “comando” que, internamente, possui as operações *execute* e *unexecute*. Ao ser despoletada a acção *execute* é registado um *listener* que invocará a operação contrária assim que o *web layer* seja terminado.

## 5.4 Desenho da solução

### 5.4.1 Estrutura do *back-end*

Para estruturar todo o código do servidor foi utilizada a *framework* Django (**figura 5.10**). Esta é uma *framework open source* de desenvolvimento para a Web, que segue a *design pattern* MVC, e que visa tanto acelerar a criação de pequenos *websites* como de sistemas web complexos. Django destaca a reutilização de componentes, o princípio DRY, e recorre ao Python para a sua definição, seja ao nível de configurações, modelos ou controladores [48]. Para além disso, oferece algumas ferramentas que melhoram a experiência de desenvolvimento, nomeadamente [46]:

- Um ORM flexível que oferece suporte para vários tipos de base de dados;
- Uma interface de administração gerada automaticamente;
- Um sistema de autenticação de utilizadores;
- Um servidor *lightweight* e *standalone* para desenvolvimento e *testing*;
- Um sistema de *caching* para melhoria de *performance* e escalabilidade.

Esta decisão foi sugerida no seio do projecto pelo facto de, no projecto Mobilidade, utilizar-se o projecto GeoDjango, que consiste num conjunto de extensões para georeferenciação e dados geo-espaciais. Desta forma, a plataforma Appbase será um projecto Django isolado em duas *Django Apps* principais (termo dado pelos autores da *framework* para referenciar uma unidade independente e reutilizável, dentro de um projecto), uma para a interface web e outra para a API. A API *RESTful* será criada com recurso ao projecto Piston, um *plugin* para Django. Este [49]:

- Relaciona-se directamente com os mecanismos internos do Django;
- Não necessita de estar intimamente ligado aos modelos;
- Permite gerar respostas em formato JSON, YAML, Pickle e XML;
- Respeita a utilização correcta do HTTP (nomeadamente, dos *status codes*);
- Tem um sistema de validação *default* (que é opcional);
- Suporta *streaming*, com baixos custos ao nível da alocação de memória.

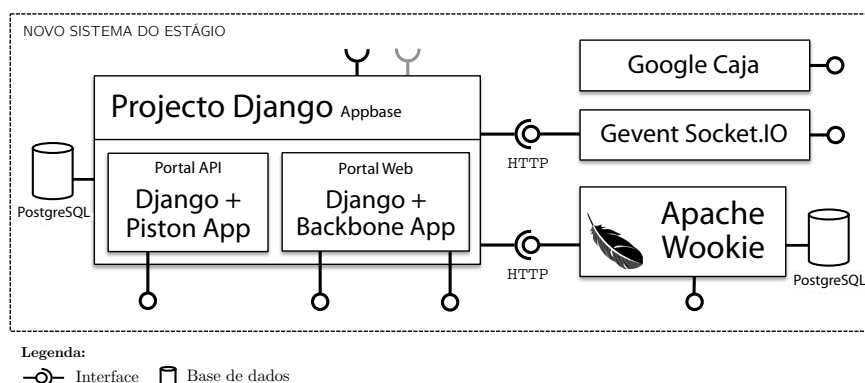


Figura 5.10: Visão de alto-nível da vista de desenvolvimento. Alguns sistemas presentes neste diagrama serão clarificados no subcapítulo “Descrição dos restantes módulos”.

### 5.4.2 Estrutura do *front-end*

As capacidades do *JavaScript* estão hoje muito além das existentes na sua implementação inicial no *browser* Netscape, e a sua performance aumentou de uma forma extraordinária graças a novos *engines* como o Google V8. Apesar do seu sucesso, o *JavaScript* ainda é muitas vezes descurado e desentendido em parte devido as suas implementações passadas (nomeadamente, no que diz respeito às incompatibilidades entre *browsers*). Hoje é uma linguagem dinâmica, orientada a objectos, com suporte a herança e eventos, e na qual podem ser aplicados *patterns*, fundamental na criação de aplicações complexas como o GMail ou Google Maps [50].

Com o aumento da quantidade e complexidade do código que é necessário criar para o *browser*, deixa de ser apenas crítico definir correctamente a arquitectura do código *server-side*, e passa a sê-lo também para todo o código que é executado do lado do cliente. Ao invés de criar grandes ficheiros *JavaScript*, que manipulam directamente a DOM e geram pedaços de HTML e CSS, o objectivo é garantir a facilidade de manutenção e extensibilidade do código, através do desacoplamento das suas partes constituintes. Optou-se por isso por seguir a *design pattern* MVC. Entre outras vantagens, torna-se possível [50]:

- Separar as preocupações da aplicação e desacoplar componentes;
- Manter uma API consistente e limpa;
- Fazer o melhor uso das tecnologias recentes para uma interacção com o utilizador com qualidade superior.

Utilizando outras técnicas e boas práticas somam-se ainda as seguintes capacidades:

- Funcionamento em vários *browsers* mantendo *fallbacks* para os mais antigos;
- *Routing* e gestão de estado no *browser*;
- Armazenamento *cross-browser*;
- *Client-side templating*;
- Gestão de dependências.

### Problemas com as *frameworks* centradas no servidor

Apesar da utilização da *framework* Django para estruturação do código do servidor ter sido previamente sugerida para este estágio pela equipa (por se utilizar GeoDjango no projecto), a sua estrutura desacoplada, que permite que seja adaptada e combinada com outros componentes, tornou-a numa excelente opção para a criação desta plataforma (um desses exemplos é descrito no subcapítulo *client-side templating*). Contrariamente ao Django, *frameworks* pesadas para desenvolvimento web têm muitas vezes alguns problemas, como [50]:

- Baixa distribuição do processamento – a UI é gerada para cada cliente, o que torna este processamento ineficiente na existência de muitos clientes;

- Alta latência na resposta ao utilizador – aplicações tradicionais não são responsivas o suficiente;
- Modelo de programação complexo – *JavaScript* é escondido ou mapeado para funcionalidade da *framework* utilizada no servidor; assim, em *runtime*, o código executado não é aquele que foi directamente escrito, o que torna o *debugging* mais difícil;
- Dificuldades em modo *offline* – adicionar capacidades para o modo *offline* é difícil quando a UI está predominantemente acoplada ao servidor;
- Perda de oportunidade ao nível da interoperabilidade e reutilização de código – a comunicação cliente-servidor fica muitas vezes escondida pela *framework*, o que torna mais difícil reaproveitar código criado para o *client-side*; o código *JavaScript* deve poder ser reaproveitado para outros projectos Web.

### *Client-side templating*

O Django possui um sistema de *templating*, para gerar o código *HTML* das páginas. No entanto, este pode ser substituído por qualquer outra solução. Optou-se por utilizar o sistema de *templating* *Mustache*, uma vez que consiste numa especificação para criação de *templates* para a qual existem diversas implementações. Para *Python* foi utilizado o *Pystache* e para *JavaScript* o *Mustache.js*. Desta forma, garantimos que, de forma transparente, os mesmos *templates* são usados no lado do cliente (se o *browser* suporta *JavaScript*) ou do lado do servidor (como *fallback*). O mesmo código poderia no futuro ser portado para projectos *Ruby*, *Java*, entre outros, sem grande esforço adicional.

Esta solução permite ainda que a geração dos dados necessários para o preenchimento dos *templates* possa ser simulada do lado do cliente, permitindo assim que, alguém da equipa que não possua o código do servidor e as suas dependências instaladas, possa mesmo assim renderizar as páginas. Isto é extremamente útil em projectos em que existe uma equipa de design responsável por criar o código *HTML*. Como é uma especificação agnóstica à linguagem, essa equipa não precisa de ter conhecimentos sobre *JavaScript* ou *Python* para criar estes *templates*.

Resumindo, procedeu-se à seguinte abordagem:

- *Client-side templating* (os *templates* são obtidos por *AJAX* à medida que são necessários e são guardados para futura utilização);
- Dados obtidos do servidor em *JSON* (estes serão sempre pedidos pois estão sujeitos a mudanças, mas são igualmente armazenados no lado do cliente para que no futuro se suporte acesso em modo *offline*);
- Permite a construção do *client-side* separado da criação do *server-side*. Ferramentas como *Mockjax*<sup>7</sup> são úteis neste campo. O *server-side* pode ser substituído sem problemas, se estiver definido um contrato.

---

<sup>7</sup>Esta ferramenta permite simular respostas a pedidos *AJAX*: <http://goo.gl/xMpu>.

### 5.4.3 Descrição dos restantes módulos

A secção anterior descreveu a forma como foram estruturados os elementos desenvolvidos de modo a tornar esta plataforma uma realidade. No entanto, a implementação esteve igualmente dependente de módulos externos, cada um com as suas interfaces, *patterns* e princípios de design. Neste subcapítulo serão analisadas essas dependências externas.

#### Servidor de *widgets* para as aplicações web *packaged*

Tanto a API do portal como a interface web comunicam com um servidor de *widgets* que permite a integração de aplicações *packaged*. Pelo facto de se seguir um *standard* aberto para a definição das aplicações deste tipo, torna-se possível usar uma das implementações *open source* existentes. A escolha do servidor recaiu sobre o Apache Wookie pois revelou ser uma alternativa mais “madura” em relação ao Apache Rave e Apache Shindig (descritos no capítulo “Estado da Arte” na secção “Tecnologias e ferramentas”). Apesar de, na opinião do estagiário, a especificação OpenSocial ser extremamente interessante para projectos deste tipo (tal fica evidente ao verificar-se que grandes empresas como o LinkedIn e MySpace apostam neste *standard*), a sua implementação de referência, o Apache Shindig, fica aquém das expectativas, ao estar mal documentada e muito atrás da versão actual da especificação (tal não é problema para grandes empresas, que podem desenvolver a sua própria implementação proprietária do *standard*, mas seria inconcebível para um projecto de estágio).

#### Contentor para integração das aplicações web *hosted*

Tecnicamente, a inclusão de uma aplicação no interior de um *Web Portal* pode ser feita de diversas formas. Estas são agrupadas de seguida em quatro categorias:

***Inline JavaScript*** A aplicação web corresponde a um pedaço de código *JavaScript* que é embebido na página HTML do portal. Esta abordagem é pouco prática, uma vez que a interface gerada estará descrita no código *JavaScript* e não em ficheiros HTML e CSS como é comum, tanto para definirem o *layout* como os estilos a aplicar [51].

***Iframe*** O seu conteúdo é descarregado de forma assíncrona, não afectando por isso o *rendering* da página que o contém. No entanto, o evento que indica que a página já foi carregada totalmente (normalmente utilizado para assegurar que o código *JavaScript* é executado após todo o conteúdo da página estar pronto) só é despoletado após o conteúdo do *iframe* também ter sido carregado. Desta forma, se a aplicação contida nesse *iframe* é lenta a carregar, o utilizador fica com a sensação de que todo o portal está lento. Possui ainda problemas resultantes da natureza estática



das suas dimensões. As dimensões de um *iframe* não se alteram conforme o seu conteúdo (o que acontece com os elementos do tipo *div*). Para além disso, ao carregar em *links* dentro de um *iframe*, o *url* da página que o contém não é actualizado (impossibilitando assim, por exemplo, o *bookmarking* de páginas específicas visitadas dentro de uma aplicação — o *url* aponta sempre para a sua raiz) [51].

**Conteúdo obtido através de pedidos *AJAX*** O conteúdo é obtido assincronamente (de forma não bloqueante). No entanto, está muitas vezes limitado por uma restrição designada *Same Origin Policy*, que obriga a que o pedido HTTP feito por *AJAX* seja direccionado ao mesmo domínio da página que contém o código que efectuou a chamada (na prática, tanto a aplicação como o portal têm de estar no mesmo servidor). Esta é, no entanto, uma limitação que tem sido combatida de diversas formas nos últimos anos [51].

***Script Tag*** É escrito um pedaço de *JavaScript* que é colocado numa *script tag* na porção *head* da página HTML do portal, o que permite que o conteúdo seja descarregado de forma assíncrona (na maioria dos *browsers*) [51].

Optou-se por incluir as aplicações *hosted* através de *iframes* com capacidades adicionais, que permitem que o *url* do portal seja actualizado à medida que se navega no interior da aplicação e para que seja sensível a eventos, como o redimensionamento do ecrã e mensagens personalizadas. Para tal, desenvolveu-se código (designado nos diagramas por *Container*) que teve como base a biblioteca aberta *Froogaloop*<sup>8</sup> da empresa Vimeo que, apesar de ter um propósito totalmente distinto (manipular um *player* de vídeo), é suficientemente genérica para que possa ser utilizada noutros contextos. Este contentor permite ainda que o sistema de interacção entre aplicações *packaged* e *web layers* possa ser igualmente empregue em aplicações *hosted*.

## Servidor para geração de ambientes isolados

Tanto as aplicações *hosted* como as *packaged* são servidas a partir de domínios distintos ao do portal. Tal adiciona segurança uma vez que os *browsers* restringem a interacção entre *frames* em que o domínio, porto ou protocolo sejam diferentes. No entanto, o que se ganha em segurança é perdido em interacção. No caso dos *web layers*, que têm de ter a capacidade de executar código JavaScript utilizando objectos comuns, como o mapa, foi necessário recorrer a outra solução. Nesse caso, optou-se por se aplicar um *web service*, designado Google Caja, que permite isolar a execução do código de forma a que a aplicação possa apenas manipular certos objectos. O sistema Google Caja segue o modelo de segurança *Object Capabilities*<sup>9</sup>.

---

<sup>8</sup>Esta biblioteca está disponível em: <http://goo.gl/sQARN>

<sup>9</sup>Mais informações sobre este modelo em: <http://goo.gl/q7WVy>

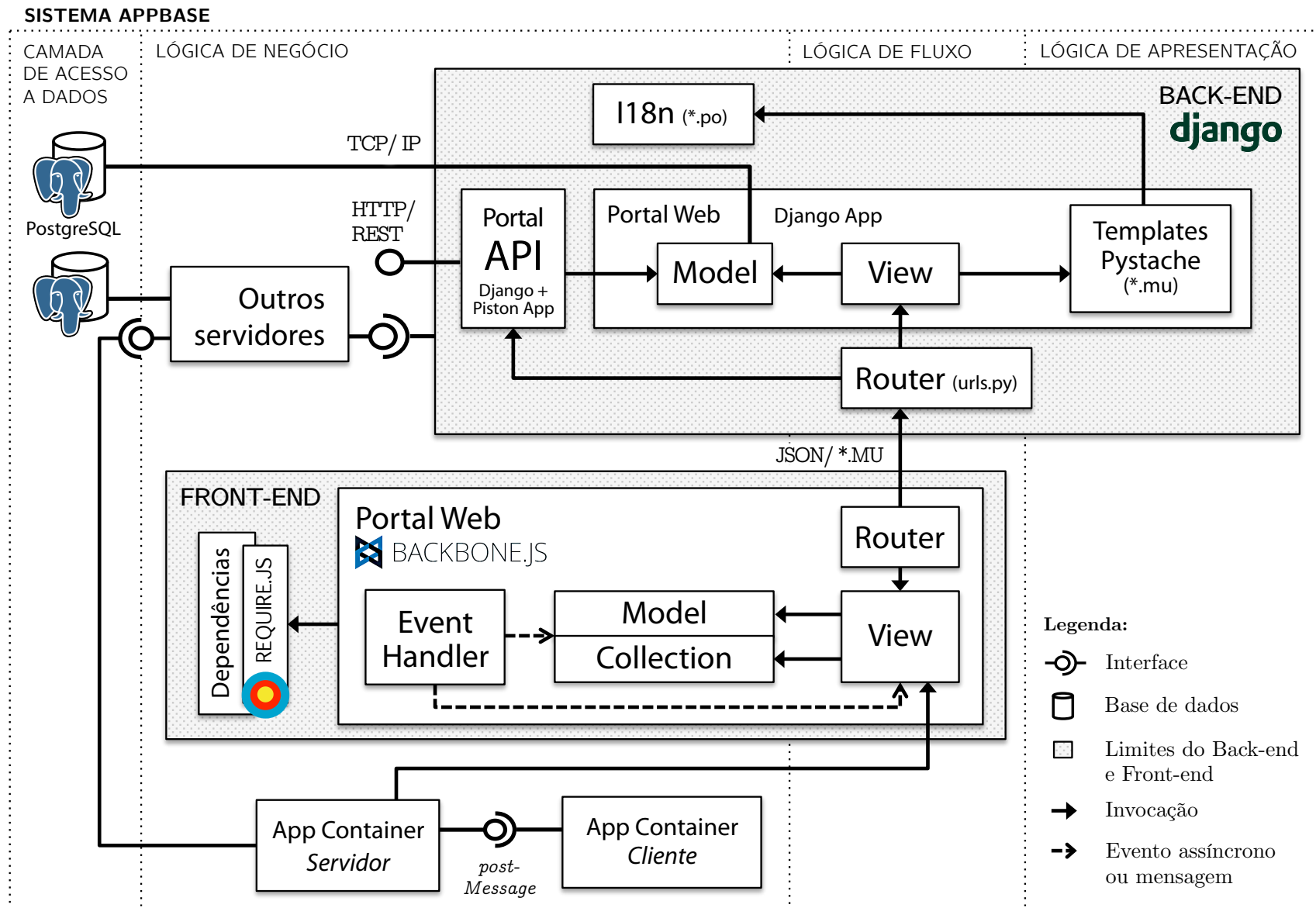


Figura 5.11: Vista de desenvolvimento da solução, com divisão entre *back-end* e *front-end*.

## Servidor de eventos

Para o servidor de eventos aplicou-se o projecto Gevent-Socket.IO, que consiste num *back-end* em *Python* para a biblioteca *JavaScript* Socket.IO. Esta tem como objectivo facilitar a criação de aplicações de tempo-real que funcionem num grande número de *browsers* e dispositivos móveis. Oferece uma única API ao *developer*, embora suporte vários mecanismos de transporte, utilizados conforme as capacidades do *browser* do cliente. Estes mecanismos vão desde os recentes *web sockets*, passando pela criação de um componente *Adobe Flash* ou *iframe* escondido, ou *Ajax long polling*.

## Gestor de base de dados

A decisão do motor de base de dados a utilizar no projecto foi sugerida pela equipa de projecto, e consiste no sistema PostgreSQL. Este consiste num DBMS *open source* relacional, *cross-platform* e amplamente suportado pela comunidade. Foi escolhido, não só mas também, pelo facto de poder ser utilizado para gestão de dados geoespaciais através de uma extensão designada PostGIS.

### 5.4.4 Modelo de dados

A plataforma Appbase está integrada com diferentes componentes que podem possuir o seu próprio modelo de dados, como é o caso do servidor Apache Wookie. Nesta secção será apresentado o modelo de dados da plataforma desenvolvida, bem como o de componentes integrados, quando tal se revelar necessário.

Segue-se uma breve descrição de cada uma das entidades apresentadas na **figura ??** (com a excepção das entidades fracas):

**App** Representa uma aplicação, qualquer que seja o seu tipo (inclui os *Web Layers*).

**Approval Activity** Permite o registo do histórico de uma aplicação, no que diz respeito às várias situações em que esta foi aprovada e rejeitada.

**User** Parte integrante do módulo de autenticação da *framework* Django. O atributo *username* consiste num identificador único que cada utilizador terá no portal.

**User Profile** Possui informações adicionais sobre um utilizador. Para além de separar um utilizador do seu perfil, consiste na forma recomendada para estender o modelo *User default* da *framework* Django.

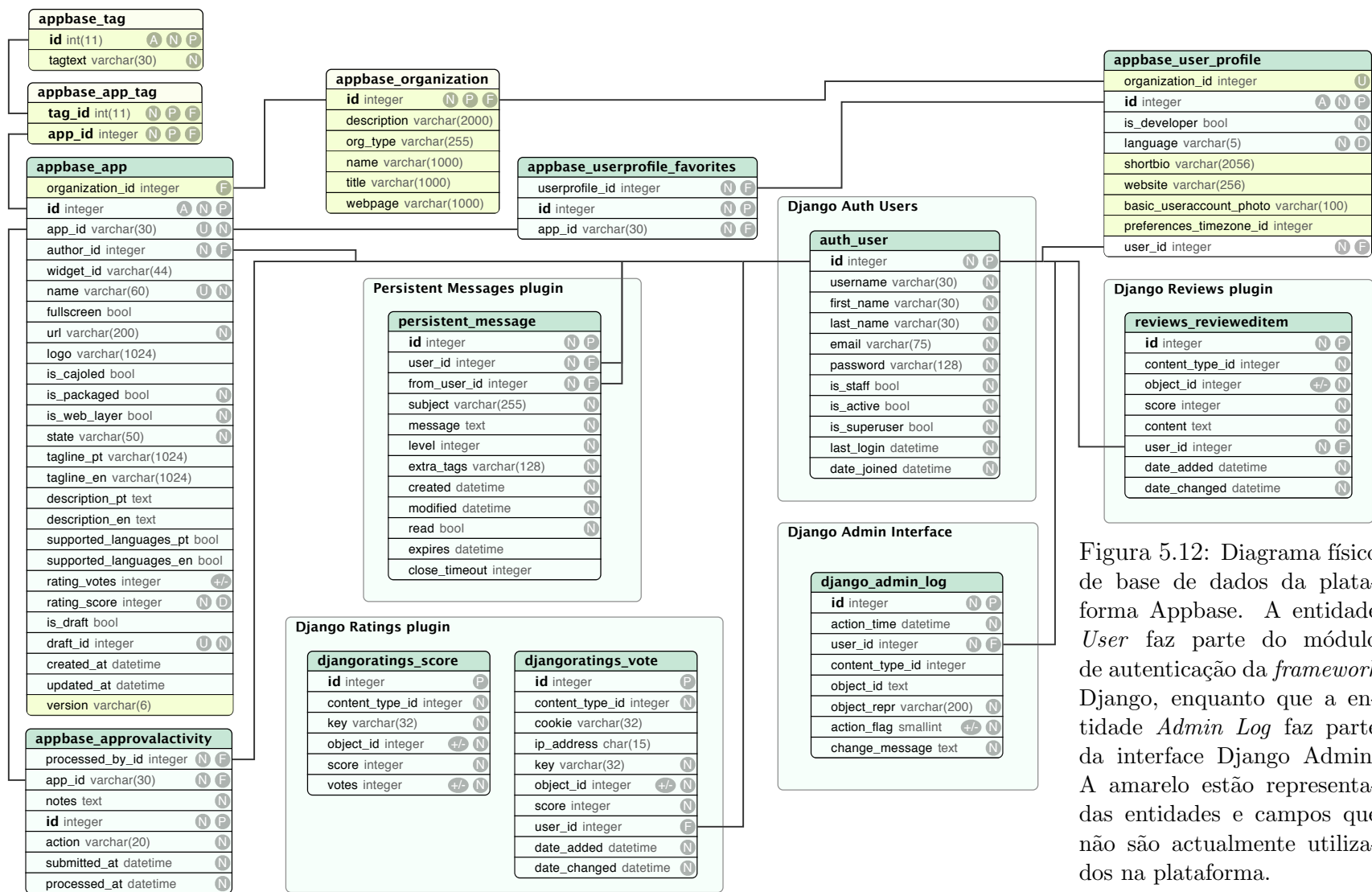


Figura 5.12: Diagrama físico de base de dados da plataforma Appbase. A entidade *User* faz parte do módulo de autenticação da *framework* Django, enquanto que a entidade *Admin Log* faz parte da interface Django Admin. A amarelo estão representadas entidades e campos que não são actualmente utilizados na plataforma.

**Persistent Message** Notificações persistentes que são apresentadas ao utilizador em casos especiais. Contrariamente às notificações de informação, sucesso e erro temporárias (que são apresentadas quando, por exemplo, o utilizador efectua uma alteração no seu perfil), as notificações persistentes ficam visíveis até que o utilizador as feche.

**Reviewed Item** Armazena os comentários feitos pelos utilizadores a aplicações.

**Vote e Score** Permite o registo das avaliações feitas a aplicações por utilizadores.

**Tag** Cada aplicação poderá ter *tags* associadas de forma a categorizá-las por temáticas.

**Organization** Permitirá que uma aplicação fique associada a uma empresa.

Para além da entidade *User* que faz parte do módulo de autenticação da *framework* Django, foram também criadas as tabelas para as entidades de permissões e grupos, embora estas não estejam actualmente a ser utilizadas na plataforma.

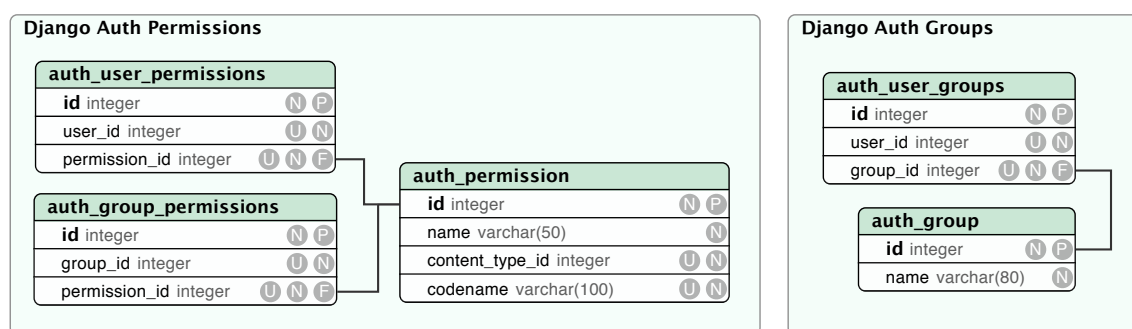


Figura 5.13: Entidades do módulo de autenticação da *framework* Django.

Para permitir que *developers* de aplicações a incluir na plataforma possam obter chaves de acesso a API's (como por exemplo, no âmbito do projecto TICE.Mobilidade, API's para acesso a dados estáticos e dinâmicos de mobilidade) e criar *consumers* OAuth (por exemplo, para acesso a dados privados de utilizador mediante a sua autorização), foi necessário que o portal estivesse integrado com o módulo de autenticação e autorização. As entidades que constituem esse módulo são representadas nas figuras 5.14 e figuras 5.14.

### 5.4.5 Estrutura do código produzido

Está disponível em anexo a descrição dos principais ficheiros do projecto Appbase, de acordo com a hierarquia de pastas do repositório de código-fonte comum aos projectos TICE.Mobilidade e TICE.Healthy.

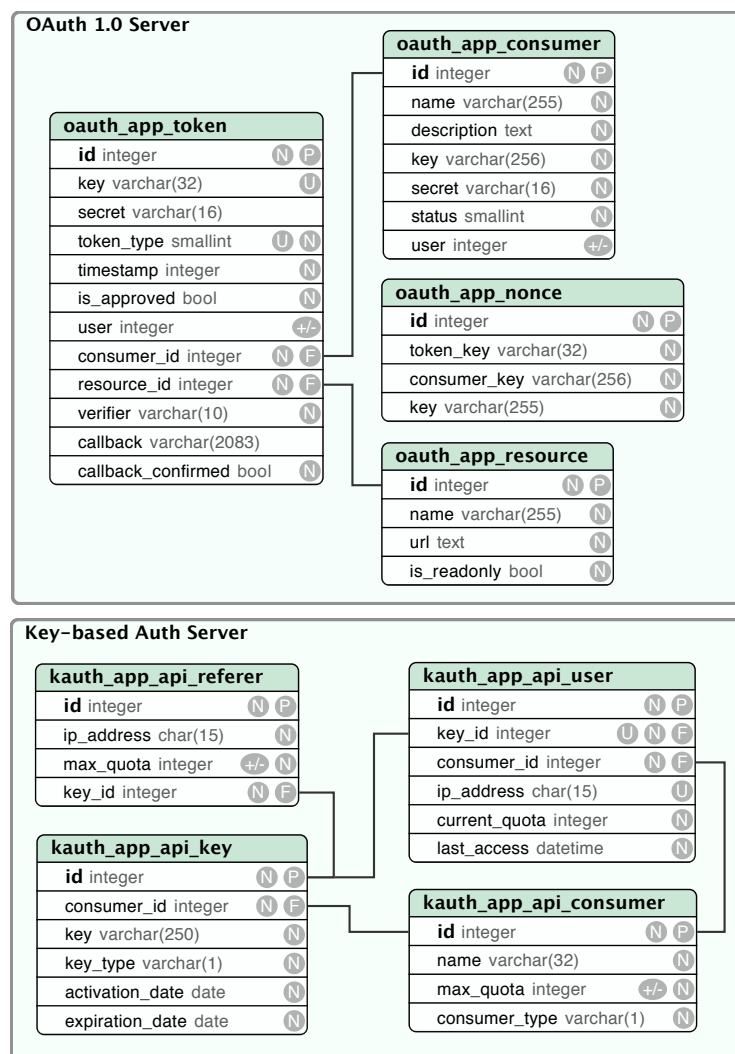


Figura 5.14: Entidades dos módulos de autorização ao acesso a API's através de chave e do protocolo OAuth 1.0.

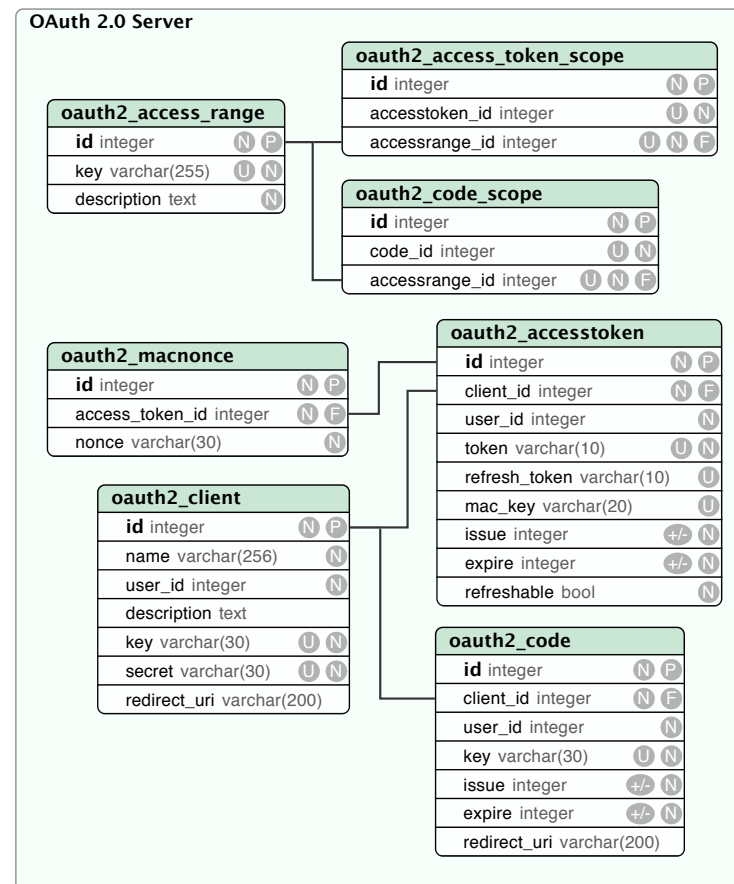


Figura 5.15: Entidades do módulo de autorização ao acesso a API's através do protocolo OAuth 2.0.







# Capítulo 6

## Desenho da Interface Gráfica

A User Interface is well-designed when the program behaves exactly how the user thought it would.

— Joel Spolsky, *CEO da Stack Exchange*

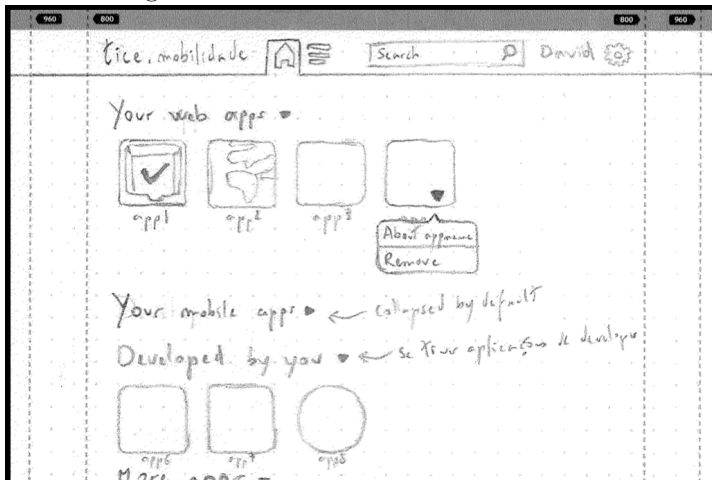
Foram criados para este projecto protótipos de baixa fidelidade e alguns ecrãs mais próximos da interface final. Esse estudo foi importante pois permitiu que a equipa de projecto ficasse com uma visão alinhada daquilo que se pretendia construir. O estudo foi posteriormente fornecido a uma empresa de design subcontratada, a *bürocratik*, a qual ficou responsável pela interface do portal. Dado que a empresa ficou igualmente responsável pela criação dos *websites* institucionais do projecto, não foram ainda incluídos no portal nenhum dos desenvolvimentos efectuados por esta. Assim, optou-se por implementar uma primeira versão de um design para o portal, criado pelo estagiário, baseado nos *mockups* inicialmente construídos.

Nesta secção serão apresentados os protótipos iniciais de baixa fidelidade, os ecrãs de maior fidelidade, e a primeira implementação do design do portal, avançados pelo estagiário.

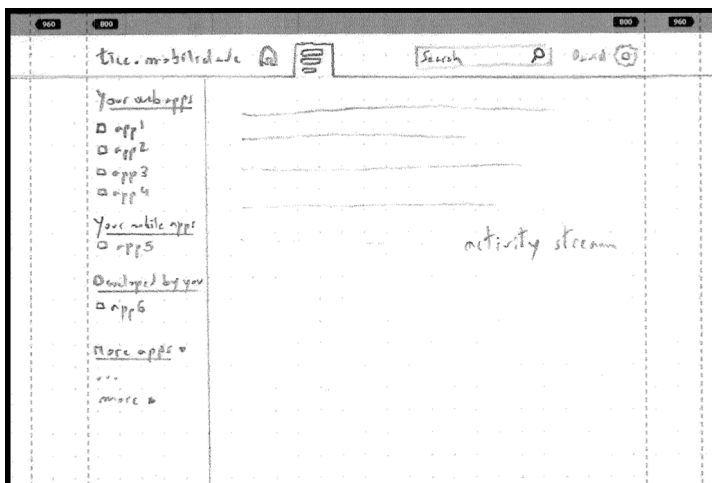
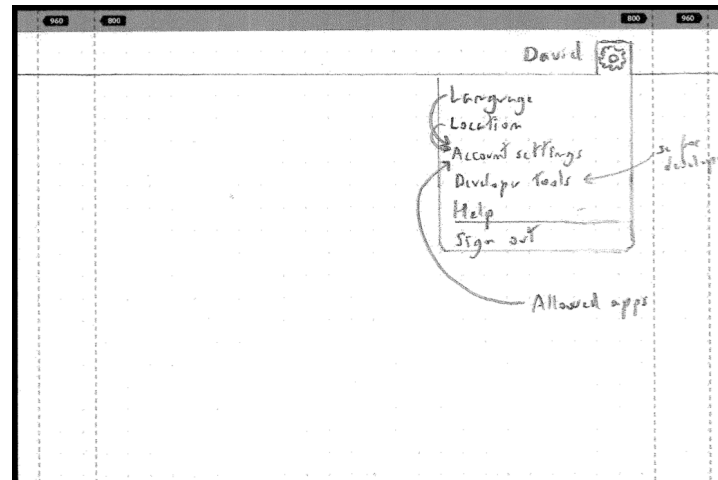
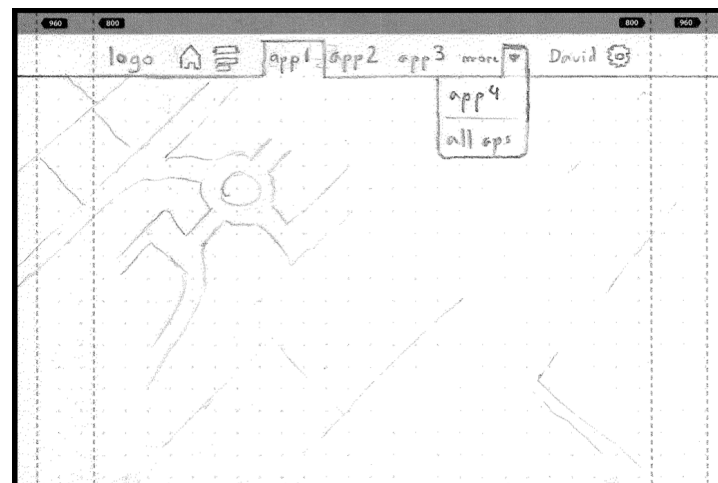
### 6.1 Protótipos de baixa fidelidade

Os protótipos de baixa fidelidade são esboços de determinada interface que permitem aos desenvolvedores testá-la de forma fácil e obter *feedback* construtivo. Um protótipo devidamente pensado pode resultar numa implementação mais rápida e precisa [52]. Com essa vantagem em mente, aplicaram-se alguns princípios e directrizes para o design da interface de interacção com o utilizador da plataforma.

Tabela 6.1: Protótipos de baixa fidelidade.

Página *Home* de um utilizador autenticado

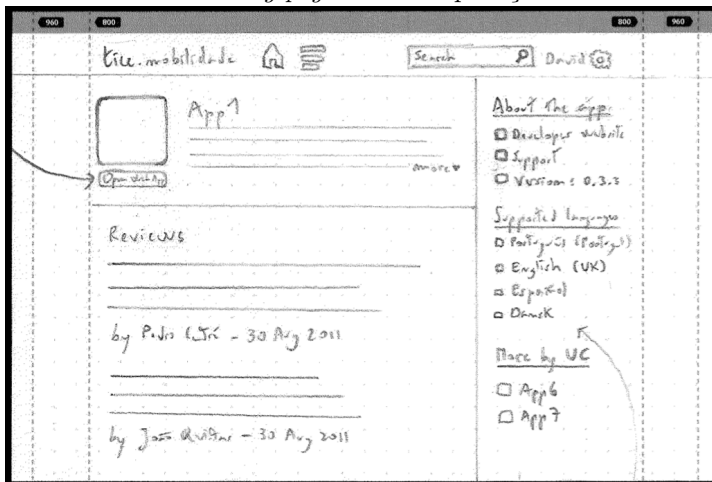
Menu de conta de utilizador

*Activity Stream*

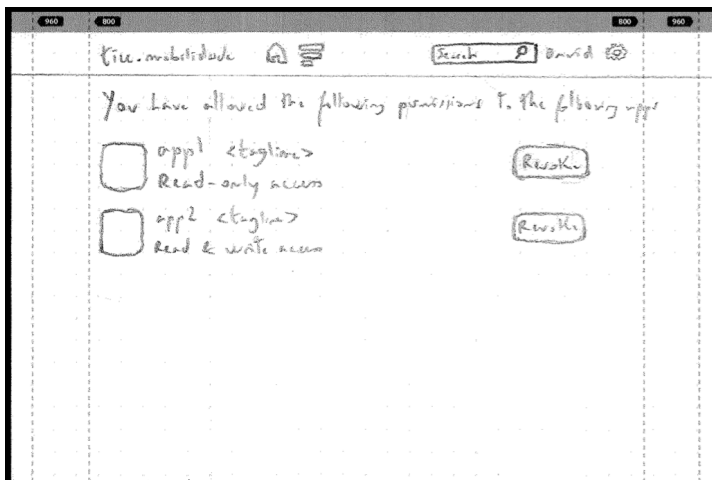
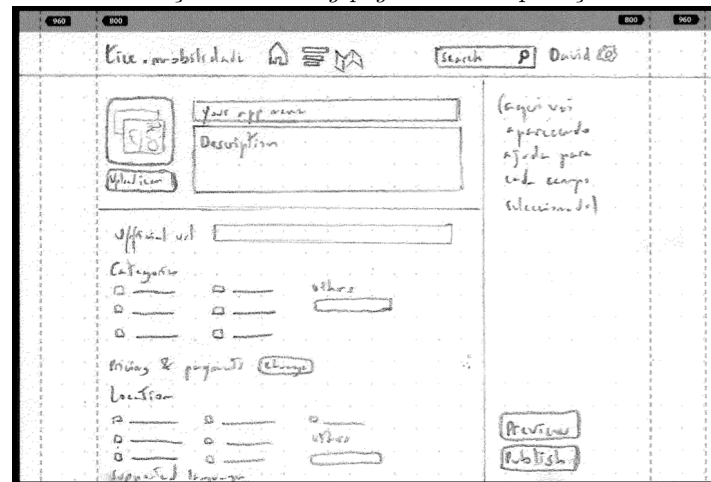
Utilização de uma aplicação

Tabela 6.2: Protótipos de baixa fidelidade (continuação).

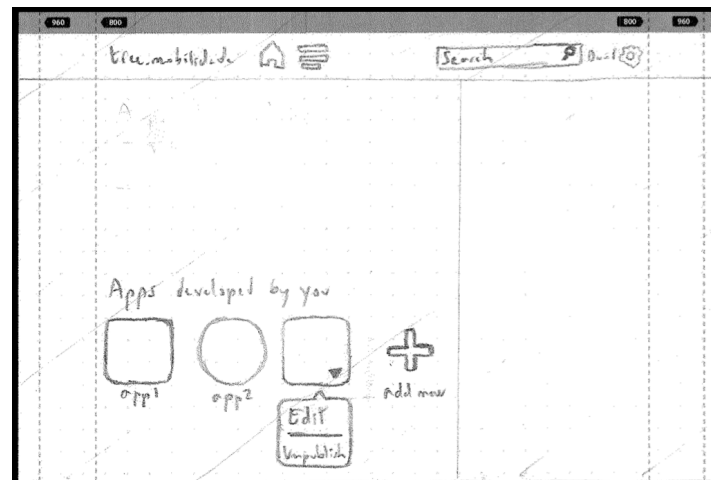
Landing page de uma aplicação



Edição da landing page de uma aplicação



Gestão de permissões das aplicações usadas por um utilizador



Lista das aplicações desenvolvidas por um programador

## 6.2 Ecrãs de alta fidelidade

As figuras que se seguem correspondem à primeira versão dos ecrãs de maior fidelidade. Apesar de apresentarem a marca do projecto TICE.Mobilidade, os conceitos são extrapoláveis para o projecto TICE.Healthy.

Ao utilizar-se uma aplicação na plataforma esta ocupará sempre uma posição central, pois cada ecrã deve ser desenhado de forma a que, quando o utilizador o observa pela primeira vez, a sua atenção esteja focada naquilo que lhe é útil à sua tarefa em mãos [52]. Haverá ainda um cabeçalho que indicará o nome da aplicação em uso, de forma a localizar o utilizador e a reduzir o esforço cognitivo, em termos da memória de curto prazo. Esse cabeçalho continuará a existir nas restantes páginas da plataforma com o fim de garantir a consistência da interface [52].

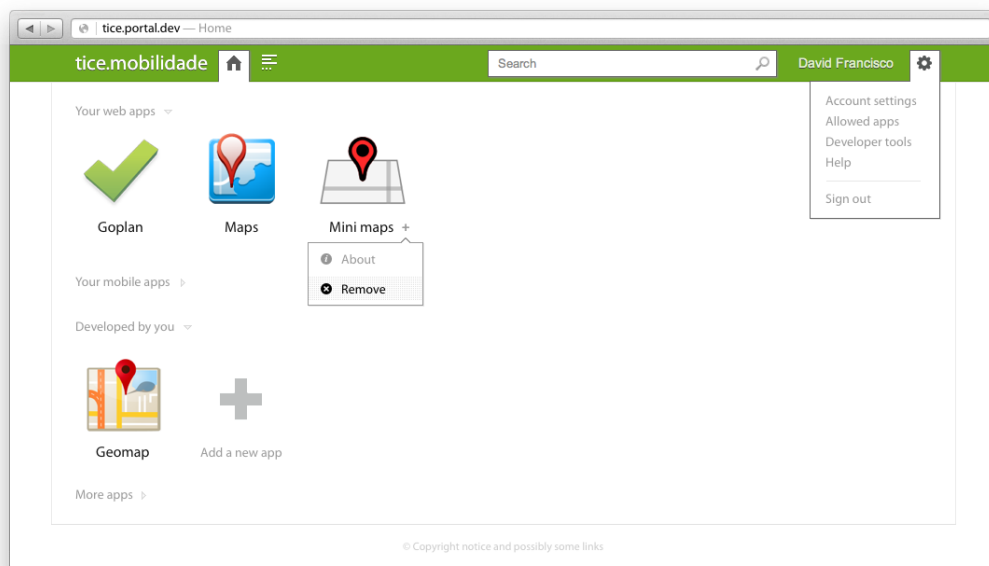


Figura 6.1: Página *Home* de um utilizador autenticado.

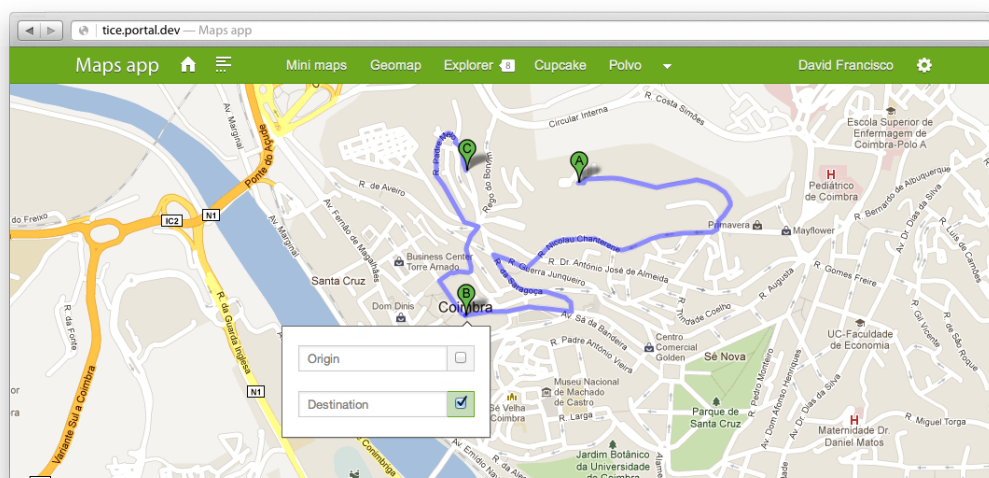


Figura 6.2: Utilização de uma aplicação.

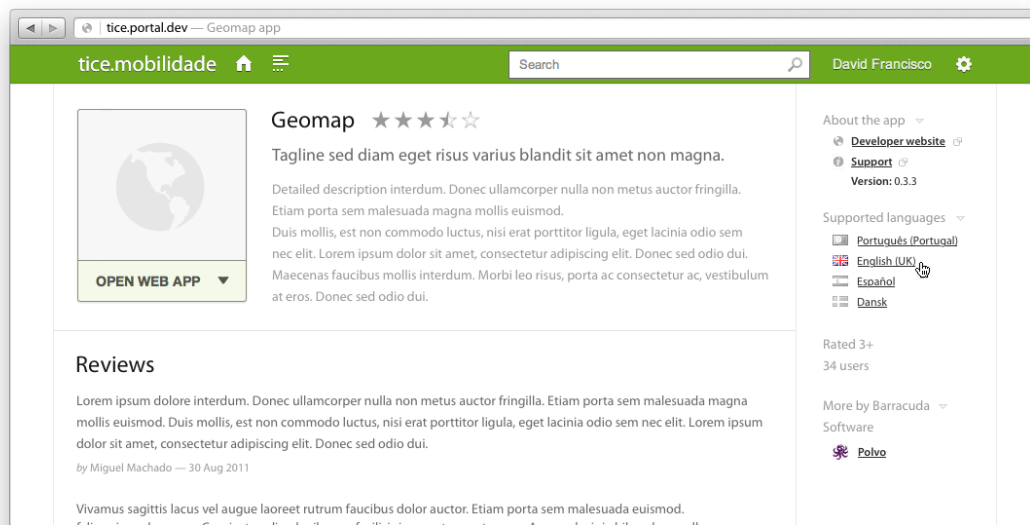


Figura 6.3: *Landing page* de uma aplicação.

Por último, será apresentada a versão mais recente dos ecrãs de alta fidelidade. Estes incluem a visualização de *web layers*, conceito específico do projecto TICE.Mobilidade. No entanto, todos os restantes aspectos são aplicáveis ao projecto TICE.Healthy.

Optou-se por separar as aplicações dos *web layers* pois, se ambos ficassem acessíveis a partir da mesma zona, seria confuso para o utilizador distinguir os que utilizam o mapa comum (os *web layers*) daqueles que não o utilizam (as aplicações). Por fim, é possível a utilização de vários *web layers* em simultâneo, o que não acontece com as aplicações.

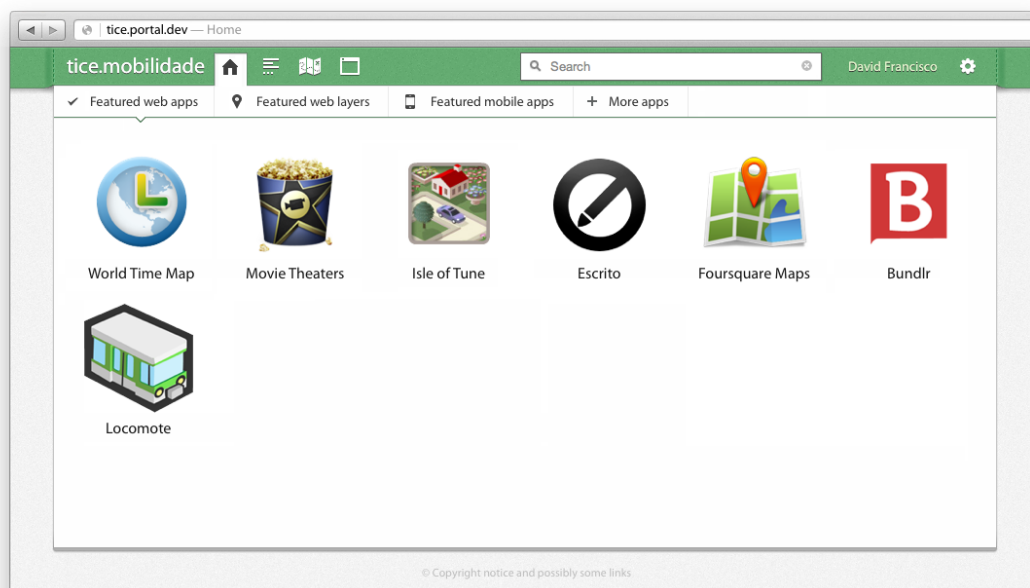


Figura 6.4: *Página Home* de um utilizador autenticado.

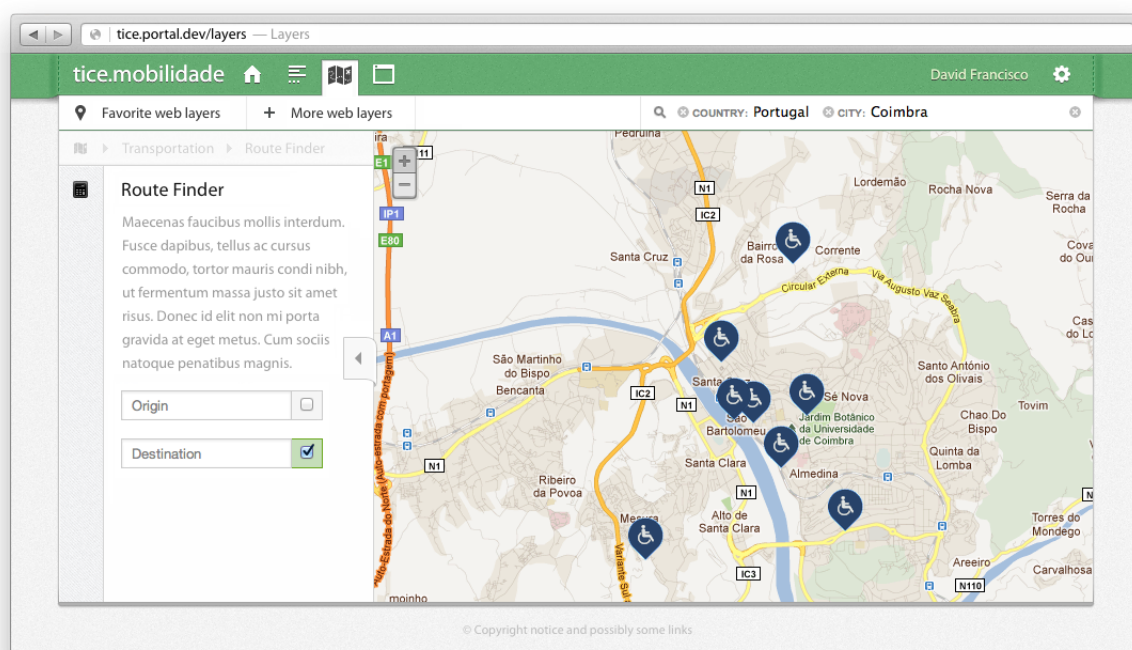


Figura 6.5: Utilização de um *web layer* para cálculo de rotas.

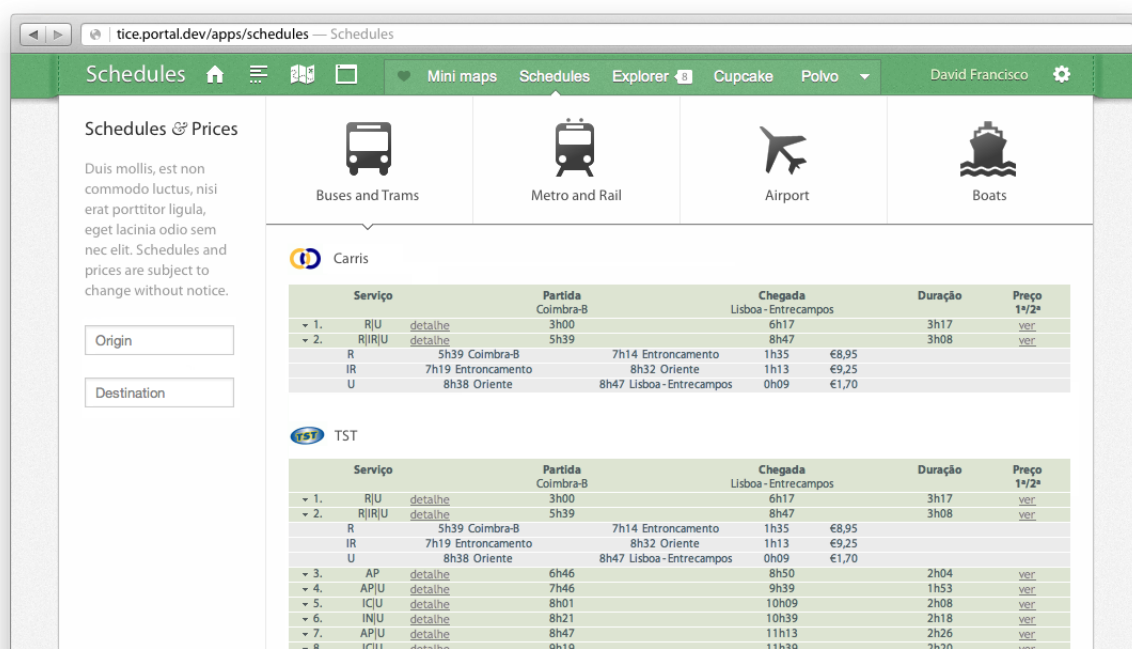


Figura 6.6: Utilização de uma aplicação.

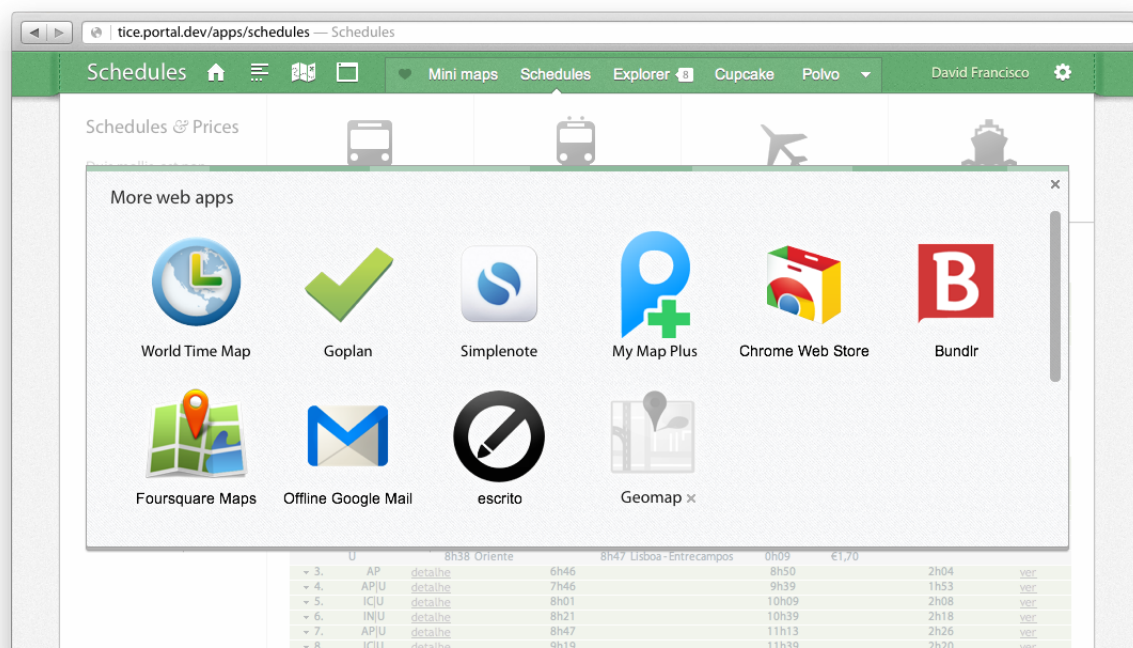


Figura 6.7: Visualização de todas as aplicações favoritas.

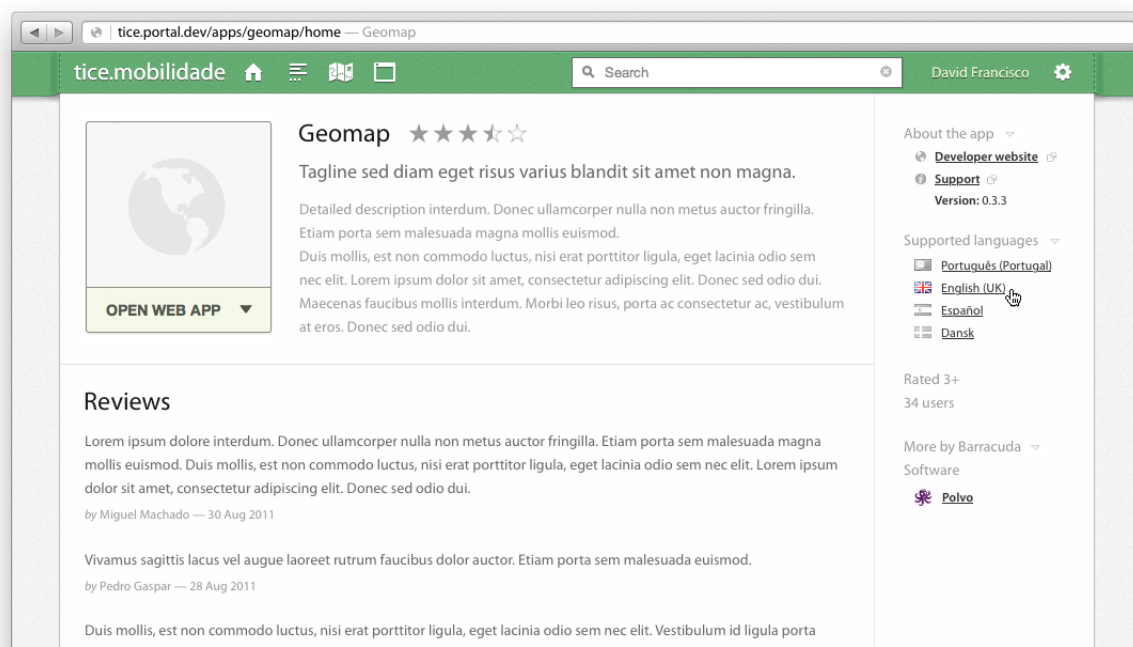


Figura 6.8: Landing page de uma aplicação.



## 6.3 Implementação inicial do design

Apesar da empresa de design estar responsável por criar a interface do portal, optou-se por se investir algum tempo numa implementação inicial baseada nos protótipos apresentados anteriormente. Com vista a preparar os *templates* para futuras iterações, os *layouts* e elementos criados têm por base uma *toolkit* simples e flexível de componentes de *User Interface* e de interacção com o utilizador em HTML, CSS e *JavaScript*, criados pela empresa Twitter. Esta *toolkit*, designada Twitter Bootstrap, está disponível sob uma licença permissiva *Apache License v2.0* <sup>1</sup>.

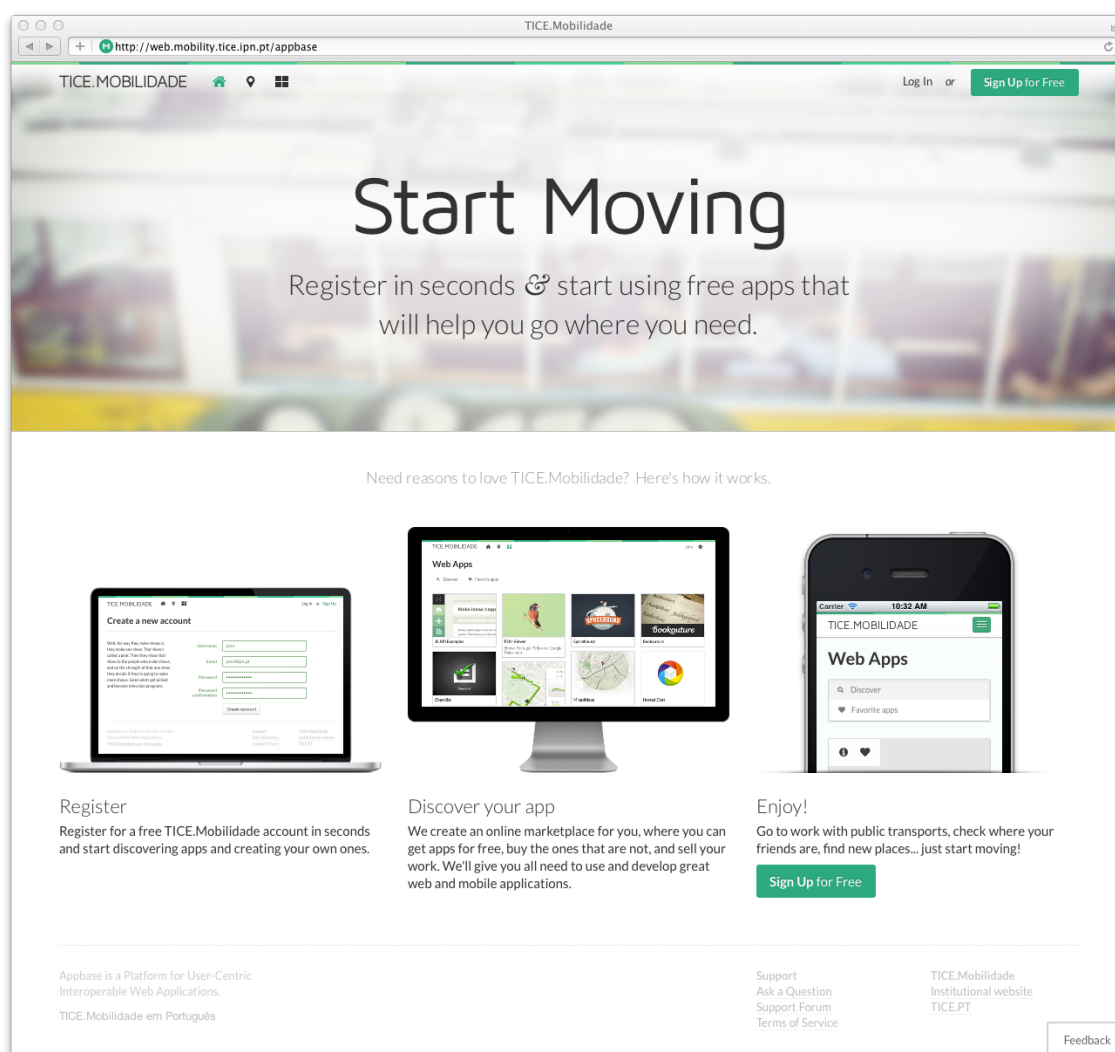


Figura 6.9: Entrada do portal Appbase do TICE.Mobilidade, em inglês.

<sup>1</sup>Esta licença pode ser consultada em: <http://apache.org/licenses/LICENSE-2.0>



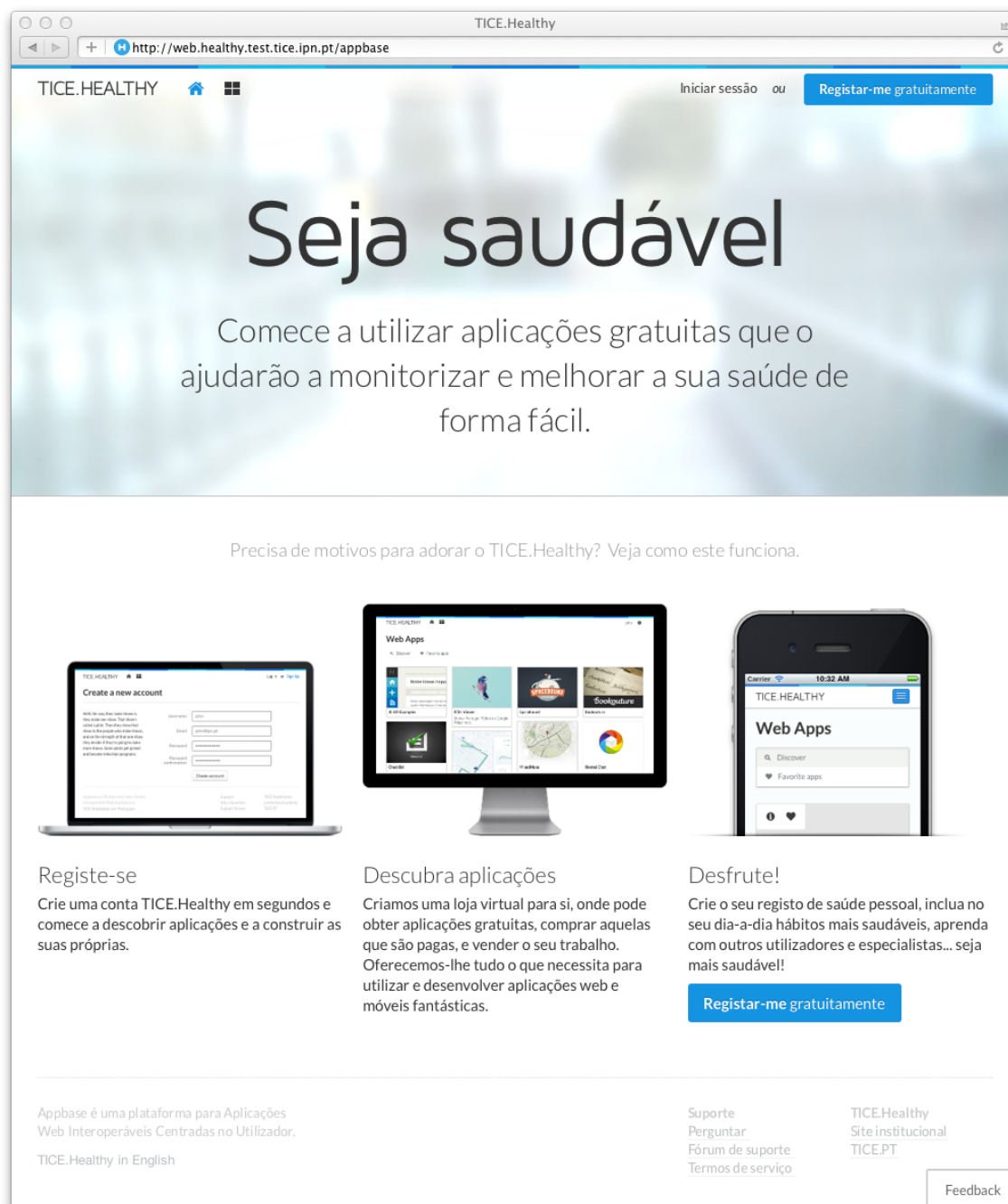


Figura 6.10: Entrada do portal Appbase do TICE.Healthy, em português.

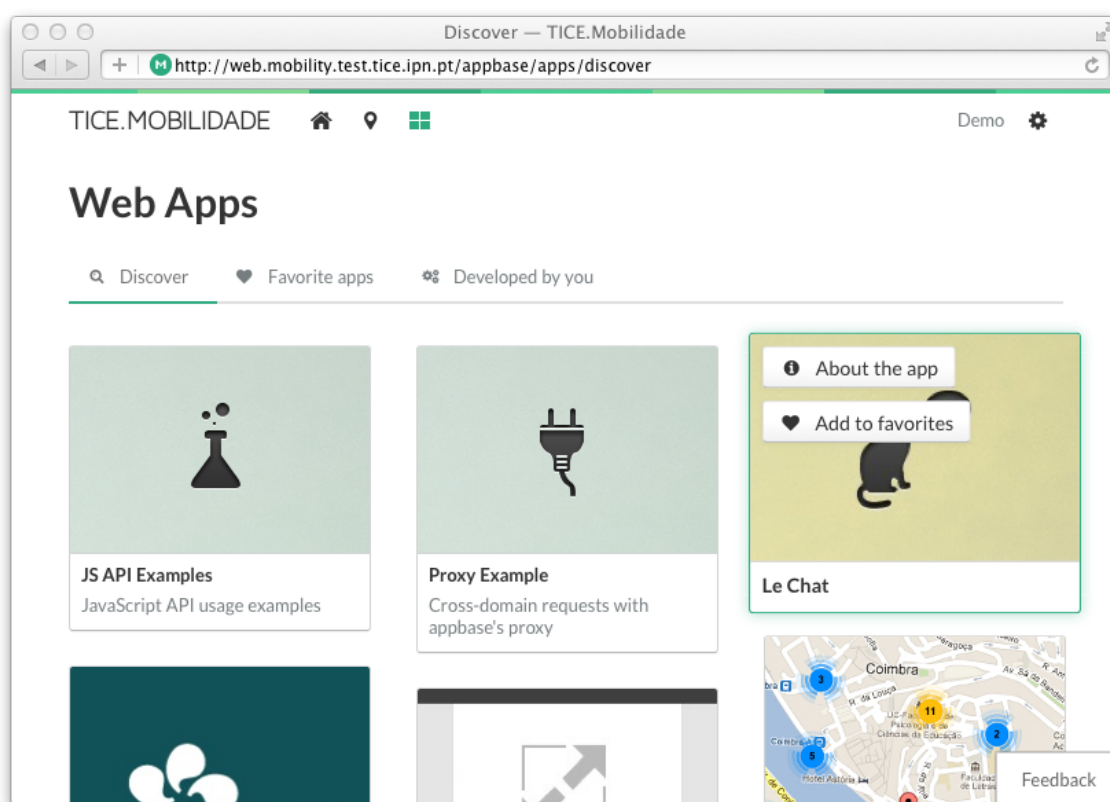
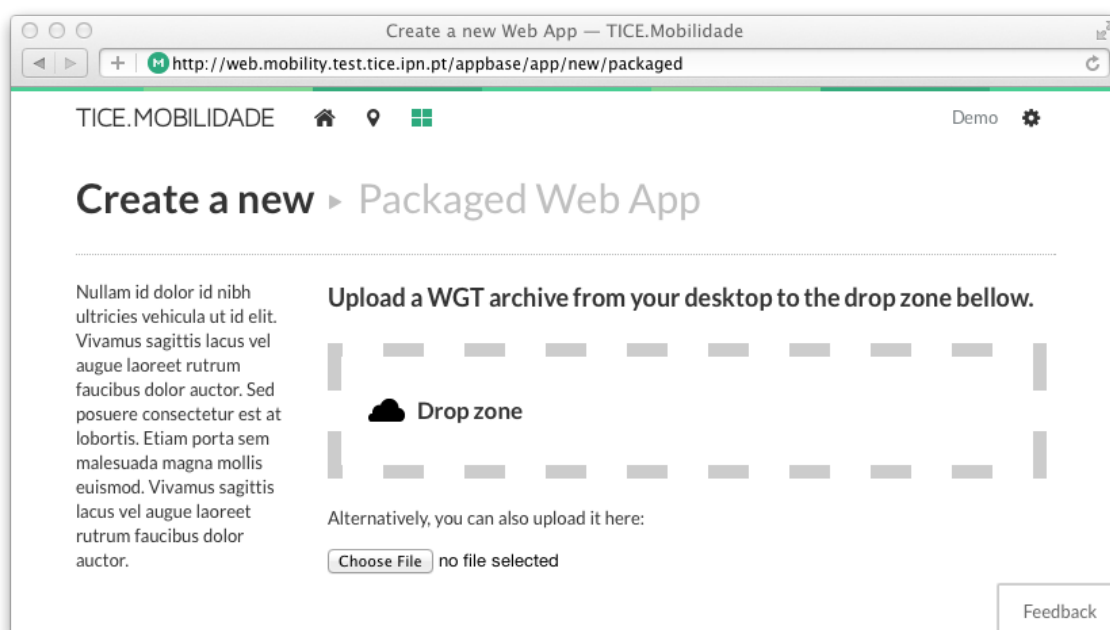


Figura 6.11: Página de descoberta de aplicações.

Figura 6.12: Formulário de criação de uma aplicação do tipo *packaged*.

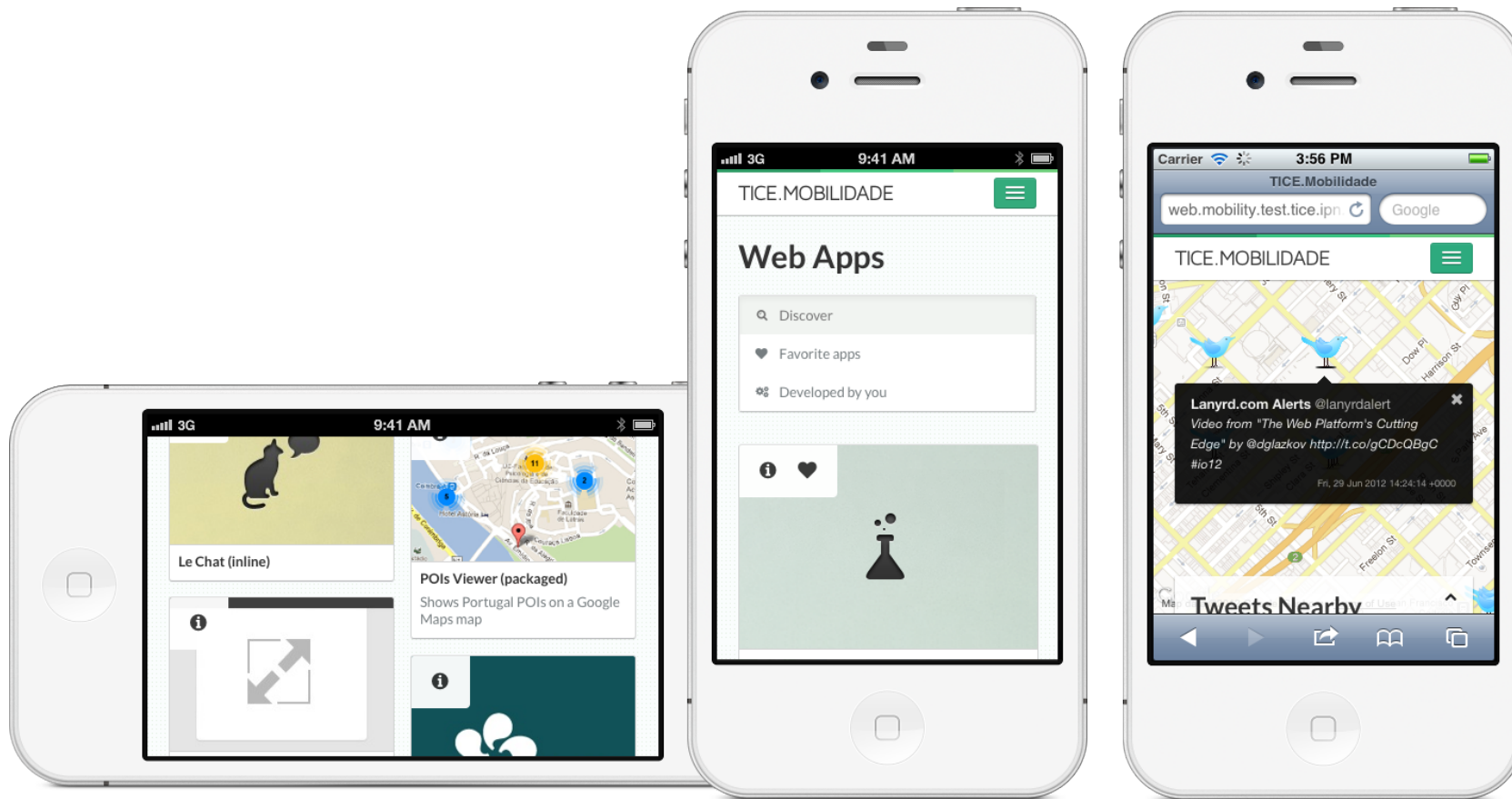


Figura 6.13: Algumas páginas do portal vistas num *smartphone*. As primeiras duas imagens mostram a descoberta de aplicações, enquanto a última apresenta o *web layer* de demonstração *Tweets Nearby*.

Visto que todas as modificações realizadas são compatíveis com a *toolkit* Twitter Bootstrap, utilizou-se como base, para os guias de estilos do portal, a documentação desse mesmo projecto. Essa documentação está disponível sob uma licença permissiva *Creative Commons BY 3.0 v2.0*<sup>2</sup>. Desta forma, os programadores de aplicações para o portal têm acesso a documentação intuitiva e completa, desenvolvida por uma comunidade diversificada de utilizadores e programadores da *toolkit*.

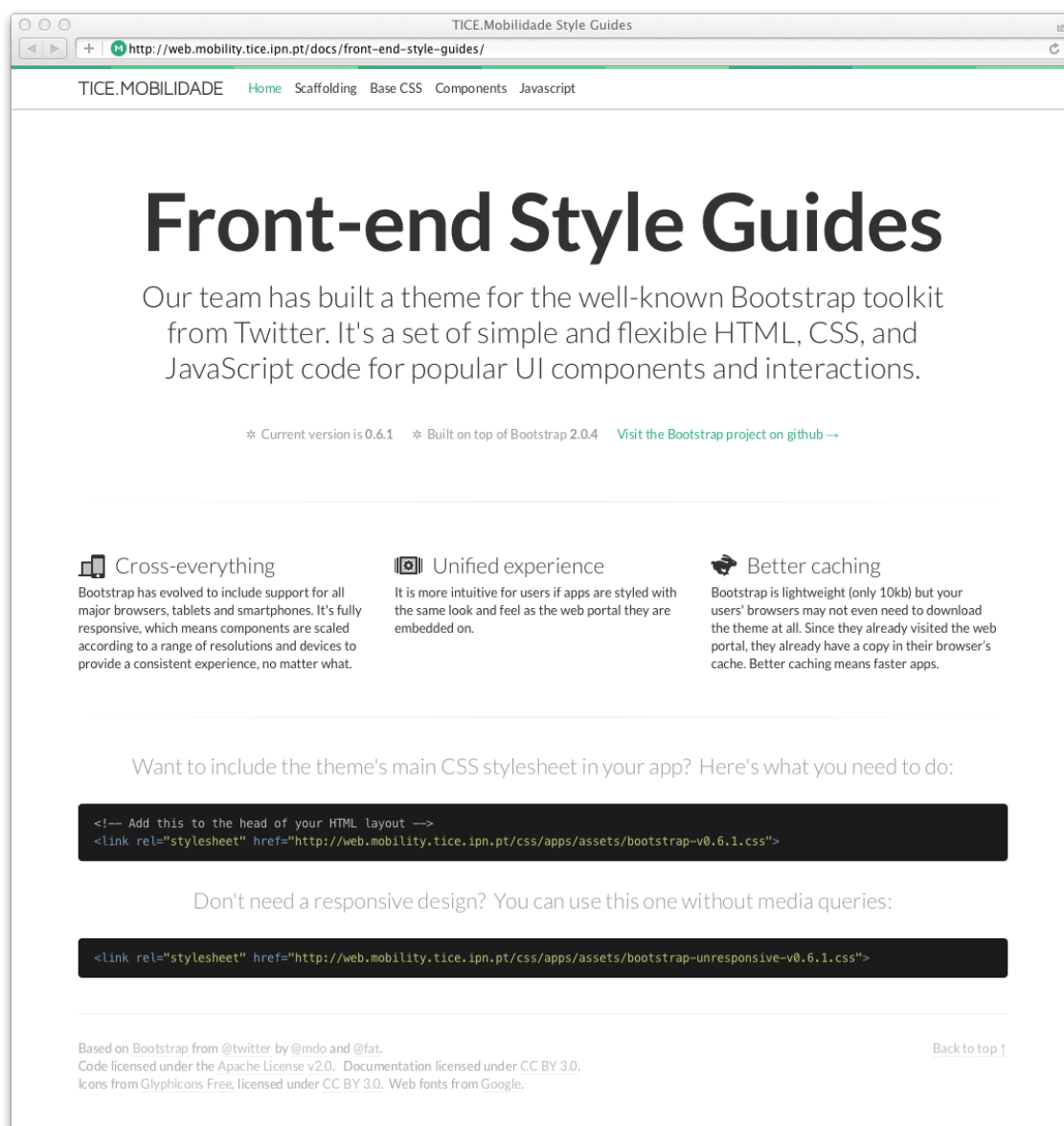


Figura 6.14: Guias de estilos para aplicações do projecto TICE.Mobilidade.

<sup>2</sup>Esta licença pode ser consultada em: <http://creativecommons.org/licenses/by/3.0/>

# Capítulo 7

## Plano de Testes

Don't code today what you can't debug tomorrow.

— *Ariya Hidayat, autor da ferramenta de teste PhantomJS*

A elaboração de testes é uma área da Engenharia de Software de extrema relevância. Neste capítulo será descrita a estratégia de desenvolvimento de código seguida, as ferramentas utilizadas e os tipos de testes elaborados. Associado a este capítulo está o anexo “Resultados dos Testes”, que contém resultados da execução da bateria de testes automatizados, de um inquérito realizado a parceiros dos projectos, e ainda algumas considerações ao nível do desempenho.

### 7.1 Estratégia de Desenvolvimento

Optou-se por se seguir a abordagem *Behavior Driven Development* (BDD) para implementação ágil dos requisitos. Esta é considerada uma evolução da estratégia *Test Driven Development* (TDD) e consiste na descrição do comportamento pretendido para determinada funcionalidade e dos passos que constituem os testes antes da sua implementação. O comportamento é usualmente descrito textualmente em linguagem natural, sendo posteriormente convertido para métodos que deverão ser programados antes da funcionalidade estar implementada. No caso particular deste estágio é útil na medida em que se estabelece um mapeamento claro entre as *User Stories* (cenários de utilização especificados no levantamento de requisitos) e esses testes, permitindo a cada momento saber que requisitos já estão a ser cumpridos e sendo suficientemente intuitivos para mostrar ao cliente, se tal for necessário.

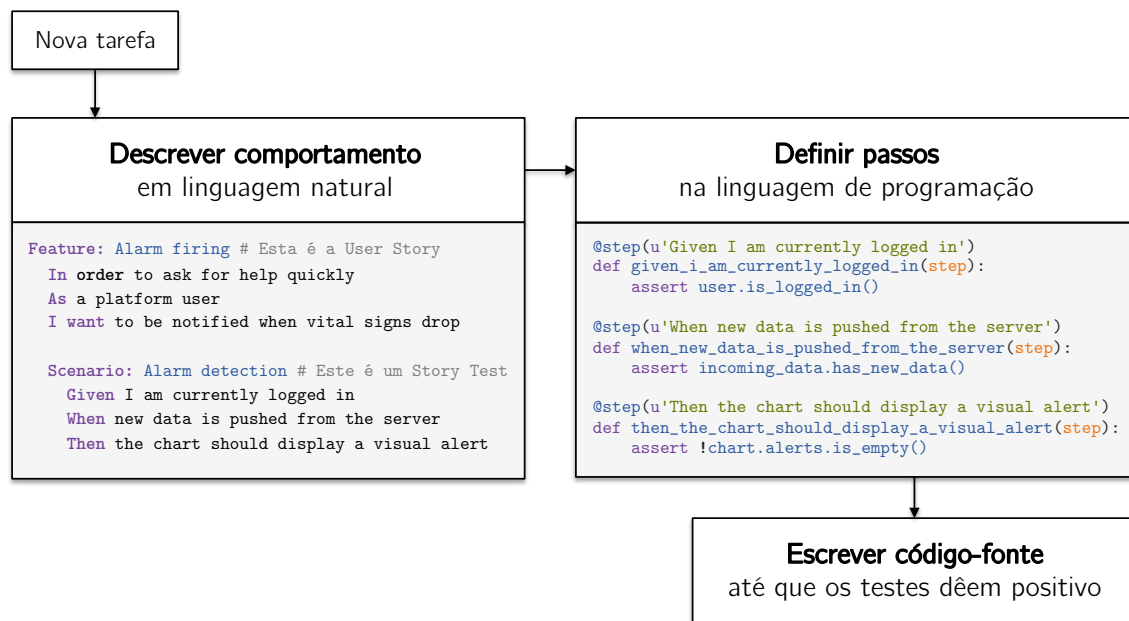


Figura 7.1: Processo a seguir na abordagem *Behavior Driven Development*.

## 7.2 Especificação e elaboração dos Testes

Os testes têm como propósito garantir a qualidade do software e, ao serem efectuadas alterações no código-fonte existente ou ao serem adicionadas novas funcionalidades, alertar o programador caso o que tenha sido desenvolvido até então não se mantenha operacional [53]. No âmbito deste estágio, foram programados testes funcionais e de integração e, sempre que se achou apropriado, testes unitários que podem ser executados de forma automatizada. Foram igualmente elaborados alguns testes não-automatizados de aceitação pelo utilizador.

### 7.2.1 Testes unitários, funcionais e de integração

#### Para o código que constitui o *Back-end*

A ferramenta aplicada para a programação de testes funcionais e de integração é conhecida pelo nome *Lettuce*<sup>1</sup>, a qual simplifica a criação de testes segundo o estilo de desenvolvimento BDD. Estes seguem o formato aplicado no levantamento de requisitos, o que facilitou a verificação da cobertura de testes do projecto. Foram criados vários *Story Tests* por cada uma das *User Stories*, estando estes presentes no anexo “*Story Tests*”.

Por outro lado, estes utilizam a ferramenta *Selenium*, o que permite que os testes criados sejam executados numa instância de um *browser* como o Google Chrome, Mozilla Firefox ou Internet Explorer. Desta forma, é possível simular o comportamento

<sup>1</sup>Mais informação sobre esta ferramenta disponível em: <http://goo.gl/NX2oS>.

real de um utilizador no que diz respeito a *clicks*, preenchimento de campos, navegação, e outros tipos de interacção. Para acelerar a criação destes testes, utilizou-se ainda: *Chrome Developer Tools*, baseado no *WebKit Web Inspector*, que permite identificar elementos dentro de uma página HTML através da sintaxe XPath; e *Selenium IDE*, um plugin para o Mozilla Firefox que possibilita gravar interacções com elementos (por exemplo, o preenchimento de um formulário) e exportá-las em código Python (para inclusão nos passos associados às *Story Tests*).

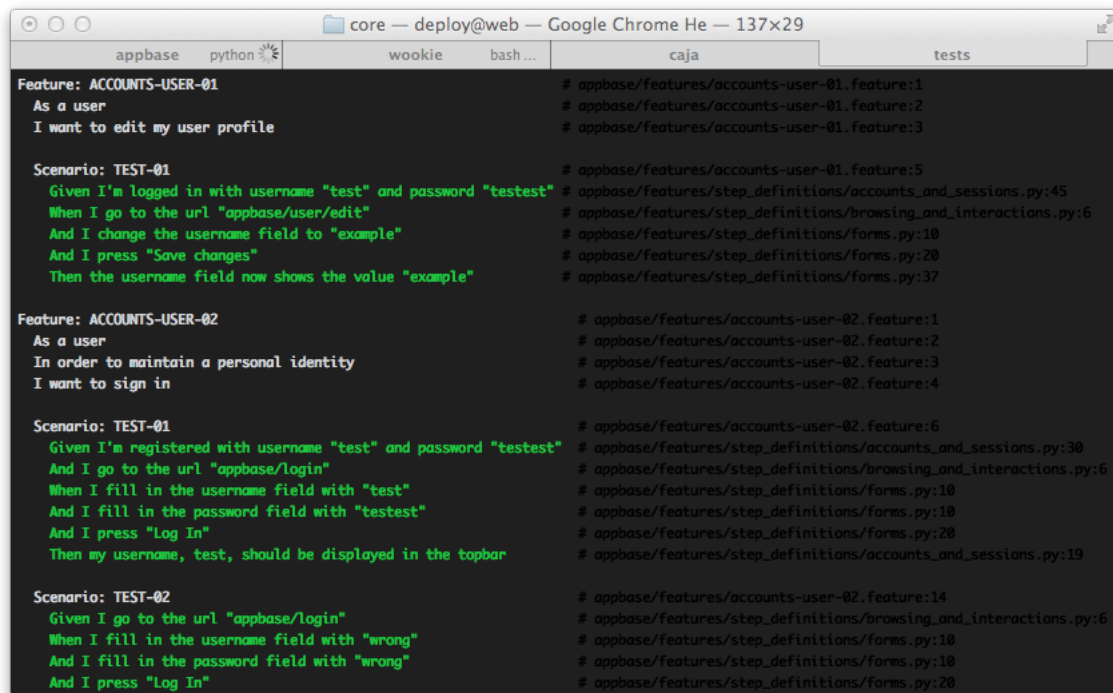


Figura 7.2: *Screenshot* da execução da pilha de testes utilizando a ferramenta *Lettuce* (o *browser* é executado numa janela separada).

### Para o código que constitui o *Front-end*

À medida que se expande a utilização de JavaScript para incluir lógica de negócio mais “crítica”, a ocorrência de erros neste código pode trazer graves problemas (como por exemplo, inconsistências lógicas, perda ou corrupção de dados) [53]. Desta forma, há valor em utilizar ferramentas específicas para teste automatizado do código que constitui o *front-end*, com o mesmo rigor com que se testa o código do *back-end*. Existem muitas opções para escrita de testes unitários e funcionais em JavaScript [53]. Após a análise de várias ferramentas, optou-se por aplicar a biblioteca *QUnit*, criada pela equipa que desenvolve a biblioteca *jQuery*. Esta possui uma interface web que permite executar os testes e analisar os resultados obtidos (figura 7.3). As suas capacidades podem ser estendidas mediante a utilização de uma biblioteca adicional, *Pavlov*, que permite a construção de testes ao estilo BDD.



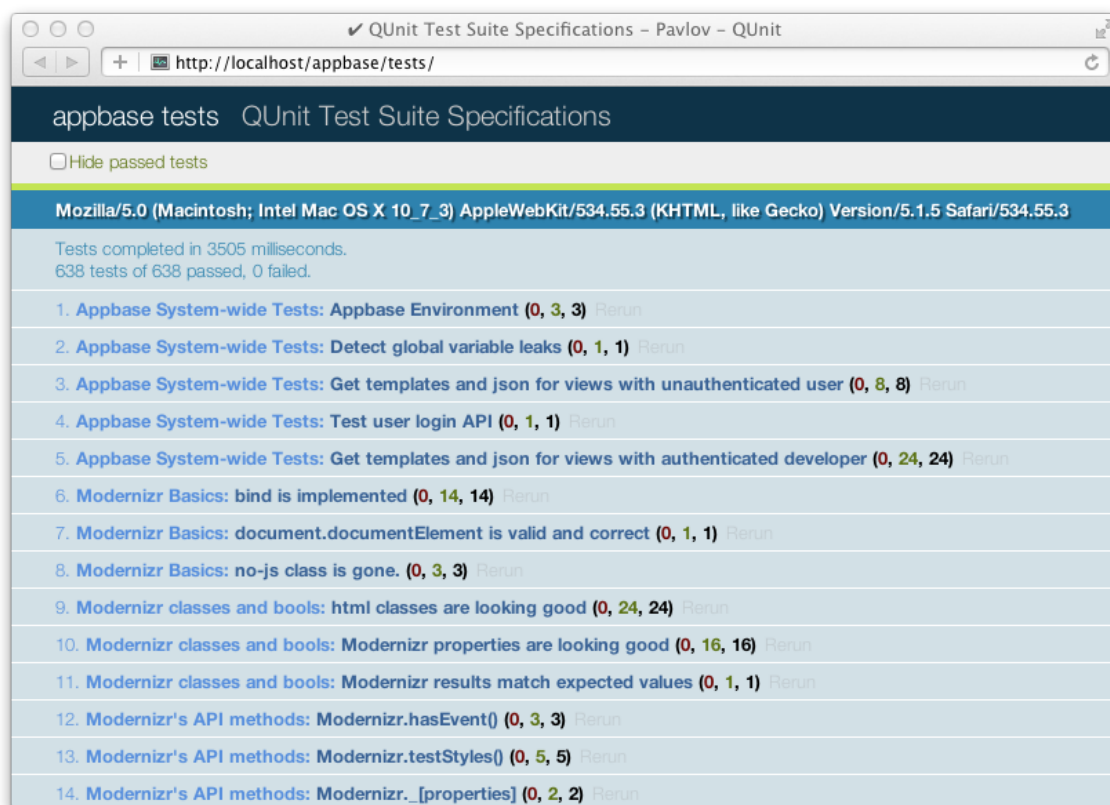


Figura 7.3: *Screenshot* do resultado da execução da pilha de testes utilizando o *Test Runner* da biblioteca *QUnit*.

Optou-se ainda por, sempre que possível, para várias das dependências de bibliotecas JavaScript de que o portal faz uso, incluir os testes dessas bibliotecas na bateria de testes do portal. As vantagens desta abordagem incluem:

- Verificar se existe conflito entre bibliotecas, por exemplo, ao nível do *naming* de variáveis e/ou funções globais, ou da extensão de objectos comuns (como o objecto `$` da biblioteca *jQuery*);
- Verificar se não existem variáveis globais *leaked* que exponham dados ou funcionalidade que possam ser obtidos ou executados pela consola JavaScript do *browser* do utilizador final, ou por extensões do *browser* que este possua;
- Quando se actualiza uma dependência, permite verificar se houve alterações de comportamento ou da sua API e agir em conformidade. Este ponto é importante pois nem sempre é suficiente executar os testes das novas versões das dependências (uma vez que estes poderão ter sido igualmente actualizados). Desta forma, como nem sempre é trivial perceber que alterações foram efectuadas nessas bibliotecas através dos seus *changelogs*, ao executar-se os testes fica-se com uma ideia clara dos problemas que poderão surgir.



## 7.2.2 Testes de aceitação pelo utilizador

Foram especificados e elaborados testes não-automatizados de aceitação pelo utilizador (do inglês *User Acceptance Testing*). Este tipo de testes tem como objectivo verificar se as *user stories* implementadas funcionam da forma esperada pelo utilizador final [54]. Definem, desta forma, o valor de negócio que cada *user story* deve entregar [55].

### Sessões de demonstração

Foi feita, no final de cada *sprint*, uma demonstração das funcionalidades implementadas no decorrer do *sprint*. Estas apresentações foram feitas à equipa de gestão do projecto, que inclui não só, mas também, os seguintes elementos:

- Eng. José Gouveia Leal, Director Executivo dos projectos TICE pelo IPN;
- Professor Doutor Carlos Bento, Director do LIS;
- Professor Doutor Jorge Dias, Director do LAS;
- Eng. Alcides Marques, Coordenador do projecto TICE.Mobilidade e Director Adjunto do LIS, na qualidade de *Product Owner*;
- Mestre João Quintas, Coordenador do projecto TICE.Healthy;
- Eng. Miguel Laginha, Coordenador Técnico dos projectos TICE, na qualidade de *Scrum Master*;
- Eng. António Cunha, Director Adjunto do LAS.

Estes elementos puderam, desta forma, criticar o trabalho desenvolvido numa perspectiva de negócio, dando assim um contributo positivo para o progresso do estágio e permitindo definir um plano adequado do trabalho a realizar em *sprints* posteriores.

### Sessão técnica de exploração da plataforma

Foi agendado para o dia 2 de Julho de 2012, nas instalações do Instituto Pedro Nunes, uma sessão de trabalho que abordou os aspectos técnicos da plataforma, orientada às equipas de desenvolvimento dos vários parceiros. Prolongou-se durante a manhã e a tarde desse dia. Este foi um passo muito importante no sentido de validar o trabalho desenvolvido e enriquecer a plataforma com aplicações de vários parceiros do consórcio. O plano inicial da sessão de trabalho incluiu os seguintes pontos <sup>2</sup>:

#### Período da Manhã

- Apresentação da arquitectura e contexto tecnológico da plataforma;
- Introdução e discussão dos diferentes tipos de aplicações.

---

<sup>2</sup>Foram também abordados aspectos da plataforma One.Stop.Transport que não se encontram aqui indicados por não se tratarem de trabalho desenvolvido neste estágio.

**Período da Tarde**

- Sessão *hands-on* de desenvolvimento de uma *packaged app* e respectiva integração no portal;
- Autenticação na plataforma via OAuth 2.0;
- Utilização de uma API “externa” (componente *server-side*);
- Utilização da API *JavaScript* do portal para internacionalização;
- Utilização de estilos do portal (presentes no Guia de Estilos).

Compareceram no *workshop* as seguintes empresas:

- SMARTMOVE — Serviços de Mobilidade, S.A.;
- MetiCube;
- Ubiwhere;
- MediaPrimer;
- HIS — e-Health Innovation Systems;
- Wizdee;
- Universidade do Minho.

Esta actividade funcionou como teste de aceitação do ponto de vista do programador. No final da sessão foi realizado um inquérito sobre o nível de satisfação dos *developers* com a plataforma Appbase, do ponto de vista da criação de aplicações e sua utilização. Os resultados estão disponíveis no anexo “Resultados dos Testes”.

**Apresentação na FCTUC e concurso de aplicações**

Haverá no dia 24 de Setembro de 2012 uma apresentação formal do projecto TICE-Mobilidade na Faculdade de Ciências e Tecnologia da Universidade de Coimbra. Prevê-se que esta se estenda a vários departamentos. Conjuntamente, haverá um concurso de desenvolvimento de aplicações no qual serão atribuídos prémios aos projectos mais inovadores.

Esta actividade funcionará como teste de aceitação do ponto de vista do utilizador final e do programador de aplicações para a plataforma.

**7.2.3 Testes de usabilidade**

Optou-se por não se elaborarem testes de usabilidade pelo facto do aspecto gráfico do portal ser da responsabilidade de uma empresa de design subcontratada. Uma vez que a interface gráfica desenvolvida no âmbito do estágio será modificada no futuro com base no trabalho que essa empresa desenvolverá, determinou-se que este tipo de testes deveria apenas realizar-se numa fase posterior. Para mais informações sobre as interações com a equipa de design deve ler-se o capítulo “Desenho da Interface Gráfica”.

# Capítulo 8

## Conclusões

### Ponto de situação

A plataforma desenvolvida no estágio encontra-se estável e pronta a alojar aplicações desenvolvidas por parceiros<sup>1</sup> e a ser apresentada a empresas e programadores<sup>2</sup>. Em retrospectiva, é interessante observar que a direcção tomada pelo estágio influenciou a importância atribuída pelos parceiros a certas componentes da plataforma. Tal é, em parte, visível quando os parceiros atribuíram maior importância às API's JavaScript do que ao directório e loja de aplicações (conceito que estava inicialmente previsto na candidatura dos projectos TICE.Mobilidade e TICE.Healthy)<sup>3</sup>, ambos implementados no âmbito do estágio.

### Obstáculos

No início do primeiro semestre, surgiu um período de indefinição quanto ao âmbito do trabalho a realizar, o que durou algumas semanas. A proposta de estágio foi então reescrita pelo estagiário, conjuntamente com os seus orientadores, passando assim a consistir na criação da plataforma de gestão, disponibilização e divulgação de aplicações.

Revelou-se também moroso e cansativo aplicar o processo de actualização do código dos dois projectos<sup>4</sup>. Dado que certos ficheiros são muito distintos na código-base e nos projectos TICE.Mobilidade e TICE.Healthy (como por exemplo, os ficheiros de tradução), cada vez que se procede a esta actualização, ocorrem centenas de conflitos que, apesar de identificados de forma automática, necessitam de ser resolvidos de forma manual. Por essa razão, o processo demora cerca de uma manhã, sendo repetido, em média, semanalmente.

---

<sup>1</sup>Com excepção de aplicações móveis nativas, como referido no requisito APPS-USER-03-MOB.

<sup>2</sup>Será feita uma apresentação formal, no mês de Setembro de 2012, em alguns departamentos da FCTUC, como referido no subcapítulo “Testes de Aceitação pelo Utilizador”.

<sup>3</sup>Está disponível no anexo “Resultados dos Testes” um inquérito feito sobre a plataforma.

<sup>4</sup>Está em causa o processo demonstrado no capítulo “Planeamento”, na secção “Estruturação dos repositórios de código”.

## Contributo

Neste projecto foram construídos alguns mecanismos menos comuns, tanto ao nível das aplicações *Hosted* como das *Packaged* e *Web Layers*. A inclusão do primeiro tipo de aplicações em *iframes* com capacidades estendidas não é frequente, mesmo em plataformas de grande sucesso como o Facebook. A comunicação entre aplicações web é algo que tem sido largamente estudado em projectos de investigação. Recentemente, a comunidade em torno do projecto Apache Wookie tem debatido formas de resolver esse problema, sendo uma das propostas mais referida muito semelhante à solução implementada neste estágio. Desta forma, teria todo o interesse transformar partes deste estágio em projectos *open source* que pudessem ser analisados e melhorados pela comunidade. Essa possibilidade está a ser considerada pela equipa de projecto.

Respeitante ao contributo dado pelo estagiário à comunidade *open source*, tal aconteceu das seguintes formas:

- Detectou um problema no projecto *Splinter* (utilizado para a realização de testes), desenvolveu o código necessário para o resolver e submeteu os testes associados à nova funcionalidade <sup>5</sup>;
- Criou projecto que adiciona ao sistema de *templates Pystache* suporte para internacionalização, seguindo a sintaxe criada pela empresa *Twitter* para o projecto *Mustache* <sup>6</sup>;
- Desenvolveu o projecto *Django Pystache* que permite a utilização de *templates Pystache* em aplicações criadas com a *framework Django* <sup>7</sup>;
- Reportou uma falha de segurança do projecto Google Caja <sup>8</sup>;
- Contribuiu para a resolução de um problema no sistema *Gevent SocketIO* (aplicado na comunicação entre aplicações) <sup>9</sup>.

No que diz respeito ao contributo dado pelo aluno ao Instituto Pedro Nunes, este deu-se mediante a adição de conteúdos para a Base de Conhecimento da instituição. Entre outros, fez um estudo comparativo das ferramentas existentes para documentação de código Python. Realizou também um *workshop* sobre a criação de aplicações web em HTML5.

---

<sup>5</sup>Mais informações disponíveis em <http://goo.gl/F4fVi> e <http://goo.gl/0uLu0>.

<sup>6</sup>Mais informações disponíveis em <https://github.com/dmfrancisco/pystache>.

<sup>7</sup>Mais informações disponíveis em <https://github.com/dmfrancisco/django-pystache>.

<sup>8</sup>Mais informações disponíveis em <http://goo.gl/1Szzt> e <http://goo.gl/cYf9S>.

<sup>9</sup>Mais informações disponíveis em <http://goo.gl/a82CD>.

### Análise crítica

Pensa-se que o resultado final correspondeu às expectativas iniciais. Todavia, há aspectos em que se poderia ter procedido melhor. Um desses aspectos diz respeito ao processo de teste do portal em versões mais antigas do *browser Internet Explorer*. Muitos problemas relacionados com a execução do portal nestes *browsers* foram apenas detectados após estarem em produção. Por outro lado, a performance do *web service* Google Caja ficou aquém das expectativas, sendo necessário, no futuro, analisar formas de acelerar o processo de execução de *web layers*.

### Lições aprendidas

O projecto prestou um enorme contributo ao estagiário do ponto de vista da formação. A aplicação dos conceitos de Engenharia de Software adquiridos no curso num contexto real foi extremamente enriquecedora. O estagiário descobriu a influência que tem existir um plano de projecto que seja descritivo e mantido actualizado e adquiriu conhecimentos sobre os aspectos envolvidos no levantamento de requisitos, criação de uma arquitectura, e gestão de riscos. Tornaram-se evidentes as vantagens associadas à execução de um processo de desenvolvimento de software como o Scrum e foi possível reconhecer a real importância que os seus conceitos têm para o desenrolar saudável de um projecto de software. A existência de objectivos mensais obrigou a uma maior disciplina de trabalho e organização pessoal. Foi também positiva a experiência que ganhou ao trabalhar numa equipa de maiores dimensões e ao participar em reuniões com *stakeholders* do projecto.

De uma perspectiva técnica, destaca-se a oportunidade de aprendizagem de novas tecnologias. Ficou clara a relevância da existência de *branches* em sistemas de controlo de versões, a vantagem associada à capacidade de se poder desenvolver funcionalidades em paralelo e a segurança que é poder-se experimentar sem ter medo de causar estragos no código produzido. É também desafiante e motivador poder explorar novas formas de integrar aplicações desenvolvidas por terceiros, paradigma cada vez mais proeminente na área de desenvolvimento de software para a web.

### Trabalho futuro

A curto prazo, prevê-se que os próximos meses venham a consistir numa fase de consolidação da plataforma Appbase, o que poderá incluir:

- Desenvolver os sistemas de pesquisa e pagamentos;
- Integrar os módulos de permissões, *logging* e *reporting*;
- Implementar a nova interface gráfica de acordo com o trabalho que será realizado pela empresa de design;

- Expandir algumas funcionalidades;
- Estudar que melhorias podem ser feitas nas funcionalidades existentes, a fim de as aprimorar e aumentar a satisfação de quem as utiliza;
- Continuar a documentação relativa ao *back-end* da plataforma e ao Guia do Programador;
- Reunir novos requisitos a serem implementados numa segunda fase de desenvolvimento.

A longo prazo, o estagiário mantém interesse em participar no desenvolvimento dos módulos dos projectos TICE, nomeadamente:

- Ajudar no desenvolvimento e integração na plataforma do *Personal Mobility Record*, um sistema que permitirá aos utilizadores gerirem a sua informação de mobilidade, no projecto TICE.Mobilidade;
- Contribuir na integração do *Personal Health Record* na plataforma, o qual poderá tornar-se a porção mais importante do portal no projecto TICE.Healthy;
- Substituir os mapas dos *Web Layers* por um sistema mais leve baseado em *Open Layers*, e torná-los mais fáceis de usar em dispositivos móveis;
- Desenvolver aplicações de mobilidade que aproveitem a oferta de dados, nomeadamente de agências de transporte, da plataforma One.Stop.Transport.

# Referências

- [1] M. I. Machado, B. Rosa, N. Cruz, A. Moreira, M. Yasmina, and P. Serra, “TICE.MOBILIDADE — Sistema de Incentivos à Investigação e Desenvolvimento Tecnológico,” Tech. Rep., 2009.
- [2] C. Lynch, “What’s Scrum and how do we use it?” 2011, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/zeJQ7>
- [3] P. M. F. Finette, “An Open Web App Marketplace : Observations and Thoughts,” 2010, date accessed: 19/09/11. [Online]. Available: <http://goo.gl/ciQP a>
- [4] S. U. o. B. Wilson, “Web Apps — A Snapshot of the Standards Landscape,” 2011, date accessed: 25/09/11. [Online]. Available: <http://goo.gl/6eLvP>
- [5] S. Wilson, “Widgets — the Wookie project,” 2008, date accessed: 08/07/12. [Online]. Available: <http://goo.gl/KE2Ox>
- [6] A. Marques, “Proposta de Estágio 740 — Estágios — Departamento Engenharia Informática,” 2011, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/aTdhI>
- [7] D. O. Manian, “This revolution needs new revolutionaries,” 2011, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/ajKfp>
- [8] “W3C HTML5 — Take Control,” date accessed: 22/01/12. [Online]. Available: <http://www.w3.org/html/logo/>
- [9] M. I. Machado, B. Rosa, N. Cruz, A. Moreira, M. Yasmina, and P. Serra, “PPS1 One.Stop.Transport — Especificação de funcionalidades,” Instituto Pedro Nunes, Tech. Rep., 2011.
- [10] Softhouse, *Scrum in Five Minutes*, 2006.
- [11] “Scrum - EPF Wiki - Eclipse,” date accessed: 22/01/12. [Online]. Available: <http://epf.eclipse.org/wikis/scrum/>
- [12] “Git - Fast Version Control System,” date accessed: 26/12/11. [Online]. Available: <http://git-scm.com/about>
- [13] S. Hepper, “Java(TM) Specification Requests - JSR#286 Portlet Specification 2.0,” IBM Corporation, Tech. Rep., 2008, date accessed: 26/09/11. [Online]. Available: <http://goo.gl/VA85z>

- [14] H. Lausen, Y. Ding, M. Stollberg, D. Fensel, R. L. Hernández, and S.-K. Han, “Semantic web portals: state-of-the-art survey,” *Journal of Knowledge Management*, vol. 9, no. 5, pp. 40–49, 2005. [Online]. Available: <http://goo.gl/m6nGD>
- [15] Y. Jie, Y. Peng, and G. Jianya, “Integration of geospatial Web services and Web portal technologies for geospatial Information sharing and processing,” in *2009 17th International Conference on Geoinformatics*. IEEE, 2009, pp. 1–4. [Online]. Available: <http://goo.gl/XA103>
- [16] J. Price, “The Battle For Your Browser’s Homepage: iGoogle vs. Netvibes vs. Pageflakes,” 2010, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/djtTf>
- [17] M. G. Mahemoff and P. G. Kinlan, “Thinking in Web Apps - Installable Web Apps - Google Code,” 2010, date accessed: 26/09/11. [Online]. Available: <http://goo.gl/E2Los>
- [18] J. Fried, D. Heinemeier, and M. Linderman, *Getting Real — The smarter, faster, easier way to build a successful web application*, 37signals, Ed., 2009. [Online]. Available: <http://goo.gl/2FJLd>
- [19] S. Wilson, “Converting Chrome Installed Web Apps into W3C Widgets,” date accessed: 22/01/12. [Online]. Available: <http://goo.gl/L9pRv>
- [20] (Mozilla Corporation), “Install Manifests - MDN,” 2011, date accessed: 24/09/11. [Online]. Available: <http://goo.gl/KNzeV>
- [21] B. O. Lawson, “Installable web apps and interoperability,” 2011, date accessed: 24/09/11. [Online]. Available: <http://goo.gl/Qu44c>
- [22] H. Sivonen, “Installable Web Apps — WHATWG Mailing List,” 2010, date accessed: 24/11/11. [Online]. Available: <http://goo.gl/3HFgO>
- [23] D. Hinchcliffe, “An Overview Of Badges And Widgets: The Fast Rise Of Viral Web Parts,” 2008. [Online]. Available: <http://goo.gl/TBFXA>
- [24] L. Haan, A. Vagner, and Y. Naudet, “Palette Web Portal Specification,” Jan. 2007. [Online]. Available: <http://goo.gl/9cHaJ>
- [25] A. M. Baron, “A Developer’s Introduction to Web Parts,” 2003, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/jJEe6>
- [26] M. Halvorson, “Share and access data on the web - OpenSocial Documentation - OpenSocial Wiki,” 2011. [Online]. Available: <http://goo.gl/jFiil>
- [27] J. Y. LeBlanc, “List of OpenSocial Containers - OpenSocial Documentation,” 2011, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/AUKIc>
- [28] “What is Apache Shindig?” date accessed: 22/01/12. [Online]. Available: <http://shindig.apache.org/>



- [29] “Apache Wookie (Incubating),” date accessed: 21/01/12. [Online]. Available: <http://incubator.apache.org/wookie/>
- [30] “Apache Rave (Incubating),” date accessed: 21/01/12. [Online]. Available: <http://incubator.apache.org/rave/>
- [31] E. Mills, “Welcome to iGoogle,” 2007, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/6qRd2>
- [32] A. Chitu, “The New iGoogle, Publicly Launched,” 2008, date accessed: 22/01/12. [Online]. Available: <http://goo.gl/QiKBK>
- [33] “Apps on Facebook.com — Facebook Developers,” date accessed: 22/01/12. [Online]. Available: <https://developers.facebook.com/docs/guides/canvas/>
- [34] “Facebook for Websites — Facebook Developers,” date accessed: 22/01/12. [Online]. Available: <https://developers.facebook.com/docs/guides/web/>
- [35] “LinkedIn Apps - LinkedIn Learning Center,” date accessed: 22/01/12. [Online]. Available: <http://learn.linkedin.com/apps/>
- [36] “REST API Resources | Twitter Developers,” date accessed: 22/01/12. [Online]. Available: <https://dev.twitter.com/docs/api>
- [37] “Authentication & Authorization | Twitter Developers,” date accessed: 22/01/12. [Online]. Available: <https://dev.twitter.com/docs/auth>
- [38] “Packaged Apps - Google Chrome Extensions - Google Code,” date accessed: 22/01/12. [Online]. Available: <http://goo.gl/xKQTR>
- [39] “Hosted Apps - Installable Web Apps - Google Code,” date accessed: 22/01/12. [Online]. Available: <http://goo.gl/NzD3>
- [40] “Podio App Store | Podio,” date accessed: 22/01/12. [Online]. Available: <https://podio.com/store>
- [41] M. Cohn, “Advantages of User Stories for Requirements,” 2004, date accessed: 26/12/11. [Online]. Available: <http://goo.gl/34j5>
- [42] J. Rasmusson, *The Agile Samurai: How Agile Masters Deliver Great Software*, ser. Pragmatic Bookshelf Series. Pragmatic Bookshelf, 2010. [Online]. Available: <http://goo.gl/FUzC0>
- [43] Courtney, “User stories: A beginner’s guide,” 2010, date accessed: 26/12/11. [Online]. Available: <http://goo.gl/OEp8I>
- [44] A. Stellman, “Building Better Software — Requirements 101: User Stories vs. Use Cases,” 2009, date accessed: 26/12/11. [Online]. Available: <http://goo.gl/jD7PS>
- [45] R. Pais, “PPS1 We.Can — Exemplo de cenário de utilização,” 2011.

- [46] D. Laginha, “One-Stop-Transport: a data platform for mobility services,” 2011.
- [47] “Authenticating users — Pusher,” date accessed: 08/07/12. [Online]. Available: <http://goo.gl/B980Y>
- [48] A. Holovaty and J. Kaplan-Moss, *The definitive guide to Django: Web development done right*, ser. The expert’s voice in Web development. Apress, 2007. [Online]. Available: <http://goo.gl/klhGa>
- [49] R. Shady, “Creating a very basic API using Python, Django and Piston | RobertShady.com,” 2010. [Online]. Available: <http://goo.gl/NBN7M>
- [50] A. MacCaw, *JavaScript Web Applications*, . O’Reilly Media, Inc., Ed. O’Reilly Media, 2011. [Online]. Available: <http://shop.oreilly.com/product/0636920018421.do>
- [51] M. M. Westin, “Building Fast Webapps, Fast - Velocity 2010,” 2010, date accessed: 23/10/11. [Online]. Available: <http://slidesha.re/rrTJhL>
- [52] B. Shneiderman and C. Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition)*. Addison-Wesley, 2010. [Online]. Available: <http://goo.gl/GIj2u>
- [53] B. Rady and R. Coffin, *Continuous Testing: With Ruby, Rails, and JavaScript*, ser. Pragmatic Bookshelf Series. Pragmatic Bookshelf, 2011. [Online]. Available: <http://goo.gl/PCsJN>
- [54] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*, ser. The Addison-Wesley Signature Series. Addison-Wesley, 2008. [Online]. Available: <http://goo.gl/mB5lc>
- [55] M. Cohn, *User Stories Applied: For Agile Software Development*, ser. The Addison-Wesley Signature Series. Addison-Wesley, 2004. [Online]. Available: <http://goo.gl/r6AvJ>