Instituto Superior de Engenharia do Porto
Departamento de Engenharia Electrotécnica
Rua Dr. António Bernardino de Almeida 431, 4200-072 Porto

Sistema de Apoio à Racionalização da Utilização de Energia Eléctrica

Dissertação de Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização de Telecomunicações

Tiago Alexandre Alves Teixeira

Orientacão do ISEP: Professora Doutora Maria Benedita Campos Neves Malheiro
Supervisão da Itron: Eng.º Jorge Costa
Eng.º Jorge Cardoso

Ano Lectivo: 2011-2012

Resumo

Esta dissertação descreve o sistema de apoio à racionalização da utilização de energia eléctrica desenvolvido no âmbito da unidade curricular de Tese/Dissertação. O domínio de aplicação enquadra-se no contexto da Directiva da União Europeia 2006/32/EC que declara ser necessário colocar à disposição dos consumidores a informação e os meios que promovam a redução do consumo e o aumento da eficiência energética individual.

O objectivo é o desenvolvimento de uma solução que permita a representação gráfica do consumo/produção, a definição de tectos de consumo, a geração automática de alertas e alarmes, a comparação anónima com clientes com perfil idêntico por região e a previsão de consumo/produção no caso de clientes industriais.

Trata-se de um sistema distribuído composto por *front-end* e *back-end*. O *front-end* é composto pelas aplicações de interface com o utilizador desenvolvidas para dispositivos móveis Android e navegadores *Web*. O *back-end* efectua o armazenamento e processamento de informação e encontra-se alojado numa plataforma de *cloud computing* – o Google App Engine – que disponibiliza uma interface padrão do tipo serviço *Web*. Esta opção assegura interoperabilidade, escalabilidade e robustez ao sistema.

Descreve-se em detalhe a concepção, desenvolvimento e teste do protótipo realizado, incluindo: (*i*) as funcionalidades de gestão e análise de consumo e produção de energia implementadas; (*ii*) as estruturas de dados; (*iii*) a base de dados e o serviço *Web*; e (*iv*) os testes e a depuração efectuados. Por fim, apresenta-se o balanço deste projecto e efectuam-se sugestões de melhoria.

Abstract

This MSc. dissertation describes the support system that was develop to promote the rational use of electric energy. The project application domain falls within the context of European Union Directive 2006/32/EC which states that consumers should be empowered with the means to reduce their individual electricity consumption and implement energy saving methods and devices to increase energy efficiency.

The aim is to develop a solution to provide a graphical representation of the consumption/production patterns, setup consuming ceilings, generate automatic alerts and alarms, compare anonymously consumers with identical profiles by region and predict, in the case of industrial installations, the expected consumption/production values.

The outcome is a distributed system organized in two main blocks: front-end and back-end. The front-end includes user interface applications for Android mobile devices and Web browsers. The back-end provides data storage and processing and is installed in a cloud computing platform – The Google App Engine – which provides a standard Web service interface. This option ensures interoperability, scalability and robustness to the system.

The design, implementation and testing of the prototype are detailed, including: (*i*) the management and analysis features for energy consuming and production; (*ii*) the data structures; (*iii*) the database and the Web service; and (*iv*) the testing and debugging. Finally, the outcome is discussed, the limitations are identified and improvement suggestions are made.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Excertos de Código

Lista de Equações

Glossário

Abreviatura	Descrição	Página
ISEP	Instituto Superior de Engenharia do Porto	xvii
UE	União Europeia	1
TIC	Tecnologias de Informação e Comunicação	2
C&I	Comercial & Industrial	2
IHD	*In-Home Displays*	5
VET	Visible Energy Trial	6
SO	Sistema Operativo	8
AMR	*Automatic Meter Reading*	10
RF	Radiofrequência	10
GSM	*Global System for Mobile Communications*	10
GPRS	*General Packet Radio Service*	10
PLC	*Power Line Communication*	10
DLMS	*Device Language Message specification*	11
OBIS	*OBject Identification System*	11
LCD	*Liquid Crystal Display*	13
AT	Alta Tensão	13
MT	Média Tensão	13
PDA	*Personal Digital Assistant*	14
JVM	*Java Virtual Machine*	17
API	*Application Programming Interface*	17
HTML	*HyperText Markup Language*	18
SGML	*Standard Generalized Mark-up Language*	18
SOAP	*Simple Object Access Protocol*	19
WSDL	*Web Service Description Language*	19
UDDI	*Universal Description, Discovery and Integration*	19
RPC	*Remote Procedure Call*	20
URI	*Uniform Resource Identifier*	20
REST	*REpresentational State Transfer*	20
HTTP	*HyperText Transfer Protocol*	20
SMTP	*Simple Mail Transfer Protocol*	20
PaaS	*Platform-as-a-Service*	22

Abreviatura	Descrição	Página
GAE	*Google App Engine*	22
AJAX	*Asynchronous JavaScript And* XML	25
DOM	*Document Object Model*	25
SDK	*Software Development Kit*	26
GPS	*Global System Position*	27
GUI	*Graphical User Interface*	28
CPU	*Central Processing Unit*	29
SQL	*Structured Query Language*	32
BD	Base de Dados	32
URL	*Uniform Resource Locater*	34
GWT	*Google Web Toolkit*	38
IDE	*Integrated Development Environments*	43
ADT	*Android Development Tools*	45
AVD	*Android Virtual Device*	46
QR	*Quick Response*	104

Agradecimentos

Gostaria de agradecer a todos aqueles que ao longo deste período sempre me apoiaram e sempre me disseram que era possível.

A toda a equipa da Itron - Sistemas de Medição Lda., fica a minha gratidão pela forma como me trataram ao longo deste período, em especial ao Jorge Costa e Jorge Cardoso. À orientadora do Instituto Superior de Engenharia do Porto (ISEP), a Professora Doutora Benedita Malheiro, um especial obrigado pela paciência, orientação e preparação de todo o trabalho desenvolvido.

Aos meus amigos, aqueles que estiveram comigo desde o principio, obrigado e boa sorte.

À minha namorada, Daniela Peixoto, fica a minha mais sincera gratidão pela força que me deu em todo o trabalho, quer na fase de desenvolvimento, quer na elaboração do relatório e acima de tudo pela crença que teve em mim.

Para finalizar, aos meus pais e ao meu irmão agradeço por sempre me ajudarem com as melhores palavras e pelo auxílio nos bons e maus momentos.

A todos vocês o mais sincero obrigado.

Capítulo 1

Introdução

Neste capítulo enquadra-se o leitor no projecto desenvolvido no âmbito da unidade curricular de Tese/Dissertação, apresentando-se o problema, a motivação, os objectivos, o planeamento e a organização deste documento.

1.1 Contexto

A Itron - Sistemas de Medição, Lda. é uma empresa integrada no Grupo Itron (`http://www.itron.com`), líder mundial no fornecimento de produtos de contagem, sistemas e serviços para os mercados de electricidade, água e gás, com mais de 100 anos de experiência no fornecimento de soluções tecnologicamente evoluídas e com representação em mais de 30 países. Em Portugal, o departamento de vendas e desenvolvimento são as fontes de sustentabilidade da empresa.

Neste contexto, as directivas apresentadas pela União Europeia (UE) para o início do novo século têm em consideração uma redução dos consumos energéticos. Segundo esta entidade, os estados membros devem ter um papel activo na implementação dos meios necessários para alcançar essa meta, nomeadamente, devem ser criadas condições para o desenvolvimento e promoção de novos serviços que permitam ao cliente final ter um papel preponderante na racionalização da utilização dos recursos energéticos. Em particular, a Directiva 2006/32/EC prevê o fornecimento de informação ao consumidor final, *e.g.*, indicadores energéticos como diagramas de carga[1], e ferramentas que lhe permitam adoptar um papel pró-activo em prol do objectivo proposto [1].

[1] Gráfico que representa a quantidade de energia consumida durante um intervalo de tempo.

1.2 Problema

A atribuição ao consumidor do papel principal na desejada racionalização da utilização dos recursos energéticos levanta um conjunto de questões prementes: Como motivar o consumidor? Que informação devem os fornecedores de energia fornecer aos seus clientes? Qual a melhor forma de disponibilizar essa informação? Como identificar e catalogar os perfis de consumo?

Para ajudar a responder a estas e outras questões surgiu a ideia de se desenvolver uma solução informática, baseada nas novas Tecnologias de Informação e Comunicação (TIC), que permita ao utilizador participar e gerir os seus encargos energéticos com base no seu perfil, na sua informação de consumo e na análise comparativa com clientes da mesma área de residência e com a mesma potência contratada.

1.3 Motivação

Para a Itron, o desenvolvimento desta solução vai permitir aumentar o leque de soluções de monitorização remota nos mercados da microprodução e miniprodução[2], residencial e Comercial e Industrial (C&I), incluindo tecnologias emergentes nunca exploradas localmente pela empresa como, *e.g.*, aplicações para dispositivos móveis ou serviços de *cloud computing*. A realização desta solução antecipa aquilo que a empresa prevê vir a ser uma necessidade futura do mercado e permite criar *know-how*[3] no domínio das tecnologias referidas.

A nível individual, a execução deste projecto contribuiu para o enriquecimento de conhecimentos e competências quer como finalista do Mestrado em Engenharia Electrotécnica e Computadores, quer como funcionário da Itron.

1.4 Objectivos

Esta tese pretende demonstrar que é possível através deste tipo de solução consciencializar e motivar os utilizadores para a adopção de comportamentos que promovam a racionalização e o aumento da eficiência energética nas suas instalações.

O objectivo desta ferramenta é dotar o consumidor final dos meios necessários e potenciadores para uma gestão mais eficiente dos gastos energéticos e, consequentemente, contribuir para a redução do valor da sua factura energética. Este

[2]Fornecimento de energia à rede proveniente de fontes renováveis [2].
[3]Conhecimento para efectuar uma determinada tarefa.

instrumento não deverá contemplar apenas os contadores inteligentes[4], mas também os contadores residenciais actualmente instalados, motivando também os consumidores residenciais para uma gestão mais eficiente. A solução deve armazenar a informação num centro de dados remoto e apresentar as seguintes funcionalidades:

1. Gerais:
    - Autenticação dos utilizadores;
    - Caracterização das instalações e contadores;
    - Criação de mapas de consumos/produções com base nas diferentes regiões e instalações dos utilizadores.
2. Mercado residencial e C&I:
    - Introdução e comunicação de leituras;
    - Consulta de consumos;
    - Análise do peso de cada tarifa no total consumido;
    - Alarmes de consumo.
3. Mercado de microprodução e miniprodução:
    - Consulta detalhada ou agregada de produção, em energia, potência ou capital;
    - Previsão de amortização do investimento;
    - Alarmes de mínimo de produção.

A nível de equipamento será necessário um dispositivo móvel, *e.g.*, *smartphone*, com ligação à Internet. Para o desenvolvimento recorrer-se-à, sempre que possível, a soluções gratuitas ou pré-existentes na empresa. O armazenamento de dados será alojado numa *cloud* e apresentará uma interface do tipo serviço *Web*, assegurando um elevado grau de interoperabilidade, disponibilidade e padronização.

1.5 Planeamento do Projecto

Na Figura 1.1 apresenta-se o planeamento do projecto. Começou por se fazer um estudo dos sistemas existentes e das tecnologias emergentes de suporte ao desenvolvimento desta solução. Definiu-se uma estratégia e planeou-se toda a ordem de trabalhos. Procedeu-se a um estudo prévio das principais tecnologias a

[4]Do inglês *smart meter*.

adoptar para a implementação da solução, seguido da aplicação dos conhecimentos adquiridos. Concluída a etapa de desenvolvimento, o projecto entrou na fase de testes e depuração.

	Task Name	Durat	Start	Finish
1	Relatório	172 day	Thu 01-03-12	Fri 26-10-12
2	**Estudos e Planeamento**	**36 days**	**Thu 01-03-12**	**Thu 19-04-12**
3	Estado da arte	4 days	Thu 01-03-12	Tue 06-03-12
4	Planeamento e objectivos propostos	1 day	Wed 07-03-12	Wed 07-03-12
5	Estudos preliminares Android	10 days	Fri 09-03-12	Thu 22-03-12
6	Estudos preliminares Google App Engine	10 days	Fri 23-03-12	Thu 05-04-12
7	Estudos preliminares serviços web	10 days	Fri 06-04-12	Thu 19-04-12
8	**Desenvolvimento**	**95 days**	**Fri 20-04-12**	**Thu 30-08-12**
9	Gestão de utilizadores	10 days	Fri 20-04-12	Thu 03-05-12
10	Gestão de instalações	10 days	Fri 04-05-12	Thu 17-05-12
11	Gestão de contadores	10 days	Fri 18-05-12	Thu 31-05-12
12	Submissão de leituras	10 days	Fri 01-06-12	Thu 14-06-12
13	Notificações	10 days	Fri 15-06-12	Thu 28-06-12
14	Recolha e tratamento dos dados ACE Vision	10 days	Fri 29-06-12	Thu 12-07-12
15	Análises gráficas	15 days	Fri 13-07-12	Thu 02-08-12
16	Tectos de consumo	5 days	Fri 03-08-12	Thu 09-08-12
17	Alarmes de consumo/produção	5 days	Fri 10-08-12	Thu 16-08-12
18	Criação de modelos de previsão	10 days	Fri 17-08-12	Thu 30-08-12
19	Testes	10 days	Fri 31-08-12	Thu 13-09-12
20	Melhorias	5 days	Fri 14-09-12	Thu 20-09-12

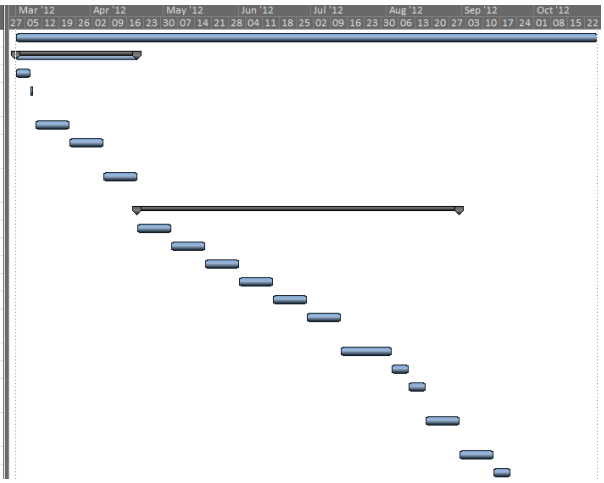

Figura 1.1: Planeamento do projecto.

1.6 Estrutura da Dissertação

Esta dissertação encontra-se subdividida em cinco capítulos. O Capítulo 1 introduz a temática, os objectivos, a motivação e o planeamento do projecto. O Capítulo 2 apresenta o estado da arte no domínio dos sistemas inteligentes de medição de energia eléctrica, incluindo uma análise comparativa dos sistemas mais representativos. No Capítulo 3 descrevem-se as tecnologias de desenvolvimento adoptadas, destacando-se o Google App Engine, o Google Web Toolkit, o Android SDK e a Google Maps API. O Capítulo 4 detalha o desenvolvimento do sistema, referindo a sua arquitectura, as funcionalidades, os testes e depuração e, ainda, a análise do tráfego gerado e do tempo de comunicação com o servidor App Engine e com o servidor Google Maps. No Capítulo 5 apresenta-se o balanço do projecto e identificam-se eventuais desenvolvimentos futuros.

Capítulo 2

Sistemas Inteligentes de Medição

Neste capítulo apresenta-se o estado da arte dos sistemas inteligentes de medição, referindo-se alguns dos sistemas mais representativos, incluindo aplicações móveis, assim como as arquitecturas, a modalidade de recolha de dados, os protocolos e os equipamentos típicos.

2.1 Medição Inteligente de Energia

Segundo o *European Smart Metering Landscape Report* [3], estão a ser implementados nos estados membros da UE diversos sistemas com o objectivo de alcançar as metas europeias para o sector energético. Estas medidas visam colocar o cliente final numa posição privilegiada para a tomada de acções conducentes à redução do consumo. Trata-se essencialmente de projectos pilotos que pretendem ajudar a encontrar as melhores soluções para alcançar este objectivo estratégico.

Uma análise superficial aos serviços de contagem inteligente instalados na Europa, referidos no *European Smart Metering Landscape Report* [3], permitem concluir que todos estes sistemas têm como alicerce a recolha de dados de consumo e a apresentação do diagrama de carga do contador. A apresentação dos resultados divide-se em dois grandes grupos, *In-Home Displays* (IHD)[1] e sítios próprios na Internet, onde podem ser consultados os dados. Adicionalmente, existem alguns sistemas que oferecem aplicações próprias para *smartphones*.

[1] Mostradores colocados em casa dos consumidores, para visualização dos registos do contador.

2.2 Visible Energy Trial

O Visible Energy Trial (VET) é um projecto que está a ser implementado no leste de Inglaterra. Este projecto piloto consiste na distribuição de dispositivos tipo IHD que permitem a visualização do diagrama de carga dos respectivos consumidores. O aparelho permite armazenar um histórico até seis meses para análise e tem a possibilidade de ser integrado com aplicações para dispositivos móveis assim como com um portal *Web* [3]. Além da visualização básica, este dispositivo permite também o estabelecimento de metas de consumo diárias e respectivos indicadores no caso destas virem a ser excedidas [4]. Na Figura 2.1 apresenta-se um dispositivo VET.

Figura 2.1: Exemplo de um dispositivo VET [3].

2.3 EcoreAction

Este sistema é baseado num portal *Web* que pode ser consultado pelos clientes das empresas distribuidoras de energia eléctrica. A redução que proporciona é baseada no *feedback* que fornece aos utilizadores que inclui a exibição do histórico do diagrama de carga, a possibilidade de previsão e comparação de gastos, a disponibilização de conselhos de poupança e, ainda, o estabelecimento de tectos de consumo [5].

Figura 2.2: Exemplo da página EcoreAction [5].

2.4 ACE Vision

O *ACE Vision* é uma aplicação Itron baseada num servidor *Web*. Esta solução, que se destina a clientes com contadores inteligentes com perfil do tipo C&I de microprodução e miniprodução, garante o acesso ao diagrama de carga dos contadores (energia activa e energia reactiva), à definição de alertas baseados no consumo, à visualização dos totais mensais divididos por tarifas e, ainda, analisar o factor de potência através de uma página *Web*.

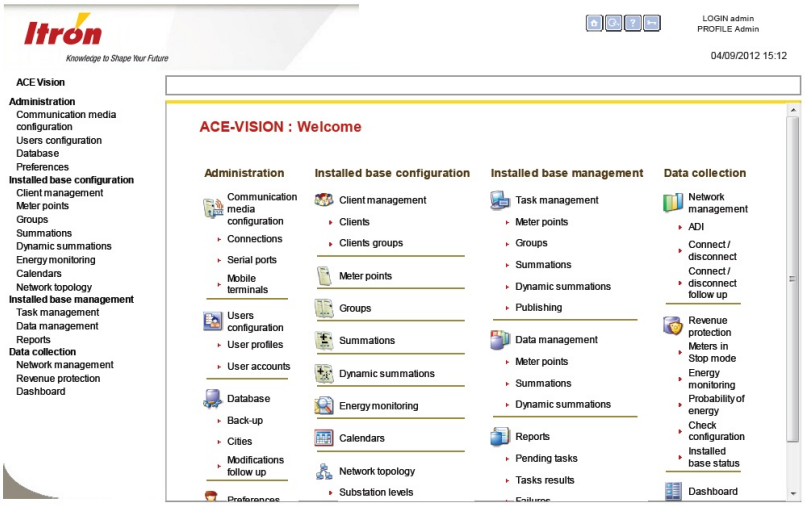

Figura 2.3: Exemplo da página ACE Vision [6].

2.5 Aplicações Móveis

Uma das abordagens adoptadas para incutir e promover uma cultura de poupança e de racionalização da utilização de energia nos consumidores consiste na

criação de aplicações para dispositivos móveis, e.g., *smartphones*, devido à facilidade de transporte, acesso à Internet, considerável capacidade de processamento, familiaridade com a interface e proximidade ao consumidor. Estes equipamentos representam um instrumento valioso neste contexto, existindo já algumas aplicações desenvolvidas para o efeito.

2.5.1 EirGrid

Este projecto, relacionado com a produção de energia, permite a visualização e tratamento da informação através de dispositivos móveis com sistema operativo (SO) iOS[2]. Esta aplicação permite a visualização dos valores de energia assim como as emissões de CO_2 quer em modo gráfico, quer em modo de texto [7].

2.5.2 MeterReading

O MeterReading é uma aplicação desenvolvida para equipamentos móveis com sistema operativo Android que se destina a clientes residenciais. O consumidor final introduz com a frequência desejada o valor das leituras no sistema e, assim que esteja disponível um conjunto mínimo de dados, consegue obter uma colecção de gráficos de apoio à tomada de decisões conducentes à redução dos consumos de electricidade [8].

2.6 Arquitecturas

A nível de arquitectura, as soluções estudadas dividem-se em dois tipos: soluções monolíticas ou *stand-alone*[3] e soluções distribuídas[4]. A escolha do tipo de arquitectura a adoptar depende dos requisitos funcionais, nomeadamente, as metodologias de apresentação, recolha, processamento e armazenamento de dados e os níveis de acessibilidade, segurança, padronização e interoperabilidade desejados.

2.6.1 Sistemas Monolíticos

Quando o armazenamento, tratamento e apresentação da informação recolhida é feito num sistema local a solução é monolítica ou *stand-alone*. São sistemas

[2] iPhone Operating System é o SO da Apple Inc. que é instalado nos equipamentos das famílias iPhone e iPad.

[3] Aplicações autossuficientes, *i.e.*, sem necessidade de programas ou equipamentos adicionais [9].

[4] Aplicações caracterizadas pela divisão de processamento (por módulos distintos distribuídos fisicamente) e que se apresentam ao utilizador final como uma solução única [10].

alojados num equipamento único que lê, processa, armazena e apresenta num *display* do próprio equipamento de contagem ou num equipamento adicional a informação, como se apresenta na Figura 2.4 e Figura 2.5, respectivamente.

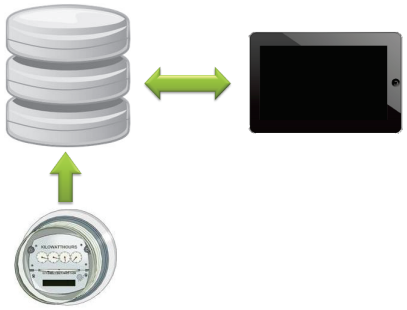

Figura 2.4: Solução *stand-alone* com contador como armazenamento.

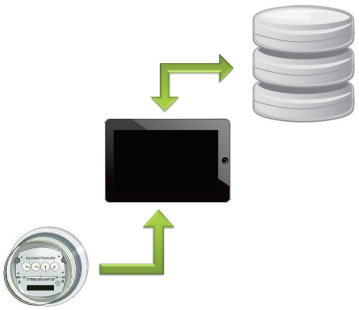

Figura 2.5: Solução *stand-alone* com *display* como armazenamento.

As vantagens desta abordagem são a manutenção da privacidade da informação e a garantia de acesso aos dados, uma vez que o armazenamento é feito localmente. Por outro lado, não permite a partilha dos registos com uma comunidade de utilizadores nas mesmas condições e sofre de grandes limitações ao nível da capacidade de armazenamento disponível. Constituem exemplos deste tipo de sistemas o *MeterReading* apresentado na Secção 2.5.2 e o VET descrito na Secção 2.2.

2.6.2 Sistemas Distribuídos

Os sistemas distribuídos são constituídos por módulos com papéis bem definidos que se encontram fisicamente distribuídos e que requerem ligação permanente à Internet. As aplicações *Web* são um exemplo deste tipo de soluções que recorrem a servidores distintos para o armazenamento e para o processamento da informação

e, ainda, a uma aplicação cliente (o navegador *Web*). É o caso dos portais *Web* EcoreAction da Secção 2.3 e do ACE Vision da Secção 2.4.

Na Figura 2.6 apresenta-se a arquitectura de um sistema inteligente distribuído de medição. As principais vantagens desta arquitectura são a elevada capacidade de armazenamento, a facilidade de acesso e de distribuição da informação e a possibilidade de adoptar diversos equipamentos de interface do utilizador. Os pontos fracos desta solução prendem-se com a vulnerabilidade da informação dos utilizadores e a obrigatoriedade de criação e manutenção de cópias de segurança dos dados, uma vez que uma falha no sistema pode levar à perda das leituras de todos os consumidores.

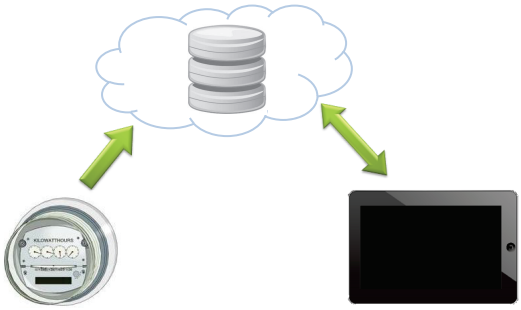

Figura 2.6: Solução distribuída.

2.6.3 Recolha de Dados

A evolução das TIC e dos sistemas de medição conduziu à criação de contadores com leitura automática – *Automatic Meter Reading* (AMR) – que permitem a recolha remota das leituras através de uma ligação de dados suportada por radiofrequência (RF), por comunicação móvel, *e.g.*, *Global System for Mobile Communications* (GSM) e *General Packet Radio Service* (GPRS), ou pela rede de energia eléctrica – *Power Line Communication* (PLC). Esta abordagem coexiste com o sistema tradicional de recolha manual das leituras efectuadas por uma pessoa que se desloca até ao local e anota os valores apresentados no mostrador do contador. Este é o caso do sistema *MeterReading* descrito na Secção 2.5.2 que atribui ao consumidor o papel da recolha tradicional das leituras – ver Figura 2.7. As soluções EcoreAction da Secção 2.3 e ACE Vision da Secção 2.4 fazem uso das técnicas de AMR – ver Figura 2.8.

Figura 2.7: Recolha manual de leituras.

Figura 2.8: Recolha automática de leituras.

2.7 Protocolos de Interface

O aparecimento dos sistemas de recolha automática conduziu à necessidade de padronização das interfaces dos sistemas de contagem para evitar a a dispersão de esforços e o desenvolvimento de soluções proprietárias. Nesse sentido, a Device Language Message Specification (DLMS) User Association definiu a especificação COmpanion Specification for Energy Metering (COSEM) designada DLMS/COSEM.

2.7.1 DLMS/COSEM

A especificação DLMS/COSEM abrange três componentes [11]: (*i*) modelação; (*ii*) mensagens; e (*iii*) transporte.

Na Figura 2.9 é possível analisar os diferentes componentes da especificação DLMS/COSEM. A informação é trocada segundo o modelo cliente/servidor, em que o contador desempenha o papel de entidade servidora responsável pela recepção e tratamento dos pedidos e os sistemas de recolha de dados funcionam como clientes encarregues de enviar pedidos ao servidor sempre que for requisitada informação sobre o contador. Os dados trocados durante o processo de comunicação consistem em objectos do sistema de identificação – OBject Identification System (OBIS) [12].

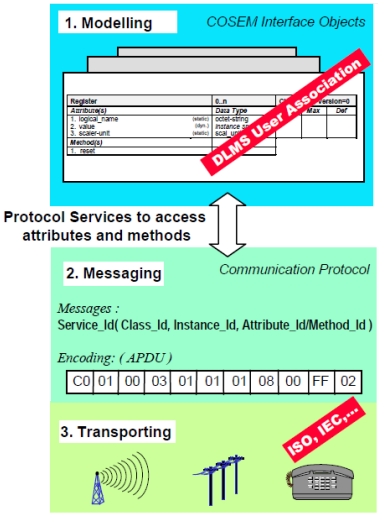

Figura 2.9: Componentes da especificação DLMS/COSEM [11].

2.8 Equipamentos

Os sistemas inteligentes de medição recorrem a dois tipos de equipamentos: os equipamentos de medição – os contadores – e os dispositivos de interface com o consumidor.

2.8.1 Equipamentos de Contagem

Um contador de electricidade é um dispositivo intermédio entre a rede de distribuição eléctrica e os locais de consumo ou produção (residências, indústrias, comércio, serviços ou centrais de produção) que permite medir a quantidade de energia eléctrica consumida ou produzida na instalação.

2.8.1.1 Contadores Residenciais

São designados contadores residenciais todos os contadores de baixa potência instalados em residências e empresas. Actualmente, neste tipo de instalações coexistem sistemas antigos – contadores electromecânicos – e sistemas modernos – contadores electrónicos.

Contador electromecânico: É um dispositivo analógico que efectua medições da respectiva grandeza com base na rotação de um disco que gira de forma directamente proporcional à energia consumida. A sua principal função é

medir o número de rotações por unidade de energia consumida, por exemplo, 600 Rot/kWh [13]. Na Figura 2.10 apresenta-se um exemplo de um contador electromecânico residencial.

Contador electrónico: É um dispositivo digital que possui maior precisão e integra um mostrador de cristais líquidos – *Liquid Crystal Display* (LCD). Permite efectuar a leitura e armazenamento do factor de potência, da energia consumida em diferentes horários e com múltiplas tarifas [14] e, em modelos mais recentes, permitem realizar a leitura remota através de PLC ou GPRS. Na Figura 2.11 apresenta-se um exemplo de um contador electrónico residencial.

Figura 2.10: Contador electromecânico trifásico Actaris P30 [15].

Figura 2.11: Contador electrónico monofásico Itron ACE2000 [14].

2.8.1.2 Contadores C&I

Os contadores utilizados nas instalações que requerem potências de ordem superior, nomeadamente regimes de Média Tensão (MT) e Alta Tensão (AT), designam-se contadores industriais. São mais precisos que os contadores residenciais e medem a energia consumida nos diferentes quadrantes assim como a

energia reactiva. Os modelos mais avançados permitem trabalhar com módulos de controlo, nomeadamente entradas e saídas de impulsos e integram módulos de comunicação RS232 ou RS485. Na Figura 2.12 apresenta-se um contador C&I.

2.8.1.3 Contadores de Produção

Estes dispositivos destinam-se à contagem de energia fornecida à rede eléctrica. Estes contadores, que são semelhantes aos descritos na Secção 2.8.1.2, incluem a particularidade de contar no sentido inverso, *i.e.*, da instalação para a rede eléctrica. Na Figura 2.12 é visível um contador de produção bidireccional, *i.e.*, que tem a capacidade de medir a produção e o consumo de energia [16].

Figura 2.12: Contador C&I Itron ACE SL7000 [16].

2.8.2 Interface com Utilizador

O equipamento de interface com o utilizador condiciona a forma como se acede aos dados do contador e influencia a metodologia de utilização dessa informação.

2.8.2.1 Dispositivos Móveis

Estes dispositivos sintetizam as funcionalidades de um telefone móvel e de um assistente pessoal digital – *Personal Digital Assistant* (PDA). Possuem um elevado poder de processamento, capacidade de acesso à Internet, uma interface com o utilizador amigável e familiar e permitem o desenvolvimento de novas aplicações. Na Secção 2.5 foram apresentadas soluções que utilizam estes dispositivos como interface com o utilizador.

2.8.2.2 *In-Home Displays*

Os IHD são monitores instalados nos locais de consumo que permitem a consulta e a configuração do sistema de gestão dos consumos de energia eléctrica. Estes

dispositivos são caracterizados pelo reduzido espaço que ocupam e fácil interacção com o utilizador [17]. Na secção 2.2 ilustra-se um sistema que usa esta abordagem.

2.9 Conclusão

O presente capítulo resume o estado da arte dos sistemas inteligentes de medição. Os sistemas analisados, que se baseiam na informação automaticamente recolhida ou manualmente inserida, suportam-se no envolvimento dos utilizadores. Possuem arquitecturas diversas, interagem com os equipamentos de contagem de forma automática ou manual e apresentam interfaces com o utilizador de características distintas. No caso dos contadores antigos sem acesso remoto, é habitual encontrarem-se soluções *stand-alone* e, no caso de contadores modernos com acesso remoto, é frequente adoptarem-se soluções distribuídas. A nível de interface com o utilizador, os sistemas inteligentes de medição recorrem a portais *Web*, IHD e *smartphones*.

No capítulo seguinte apresentam-se as tecnologias de desenvolvimento adoptadas para a realização deste projecto.

Capítulo 3

Tecnologias de Desenvolvimento

Neste capítulo descrevem-se as tecnologias, a instalação e configuração do ambiente de desenvolvimento adoptado neste trabalho. Apresentam-se as linguagens de programação e anotação utilizadas assim como as tecnologias de serviços Web, de desenvolvimento do lado do servidor e do lado do cliente e o ambiente de desenvolvimento adoptados.

3.1 Linguagens de Programação

3.1.1 Java

Java é uma linguagem de programação orientada por objectos, do tipo código aberto, que foi originalmente lançada pela Sun Microsystems. As aplicações desenvolvidas nesta linguagem são independentes da plataforma. Esta característica resulta do facto do compilador de Java, em vez de gerar código binário como um compilador normal, cria um código intermédio designado *bytecodes*. Os *bytecodes* são instruções semelhantes ao código máquina, mas que não se encontram relacionados com nenhum processador [18]. Para vincar esta propriedade, a Sun Microsystems adoptou o lema *"Write once, run anywhere"*. Os programas Java são compilados em *bytecodes* e interpretados pela máquina virtual Java – *Java Virtual Machine* (JVM) – ver Figura 3.1. Outra das características do Java é o princípio da reutilização de código baseado nos mecanismos de herança e composição intrínsecos da linguagem. Esta reutilização pode incluir quer código desenvolvido de raiz, quer de interfaces de programação de aplicações – *Application Programming Interface* (API). Para além das API de Java oficiais, que consistem

num vasto conjunto de componentes de *software* organizado em bibliotecas, estão disponíveis inúmeras API desenvolvidas por terceiros.

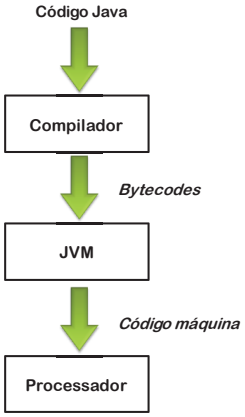

Figura 3.1: Compilação e interpretação Java.

3.2 Linguagens de Anotação

Na *Web* são usadas linguagens de anotação para descrever propriedades da informação. O objectivo é criar um mecanismo amigável e perceptível quer para humanos, quer para aplicações que possibilite a compreensão da informação representada [19]. Estas linguagens, ao contrário das habituais linguagens de programação, servem apenas para anotar a informação [20].

3.2.1 *HyperText Markup Language*

A *HyperText Markup Language* (HTML) é uma linguagem da *Web* destinada a apresentar dados que é interpretada pelo navegador *Web*. O Excerto de Código 3.1 apresenta um exemplo de um documento HTML. O HTML teve as suas origens na *Standard Generalized Mark-up Language* (SGML) que se destina a estruturar texto. A simplicidade transformou-a na linguagem de apresentação de dados da *Web* [21]. Actualmente o HTML encontra-se na versão HTML5 que integra funcionalidades como armazenamento local ou multiprocessamento [22].

3.2.2 *eXtensible Markup Language*

A *eXtensible Markup Language* (XML) é uma linguagem de anotação destinada ao armazenamento de dados. Um documento XML permite, de forma simples e flexível, o armazenamento de informação estruturada como se apresenta no

Excerto de Código 3.1 Exemplo de código HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <!-- Isto é um comentário -->
    <title>Esta é a minha primeira página<title>
    <body>
        <h2>Olá Mundo!<h2>
        <p>Esta página é um exemplo
    </body>
</html>
```

Extracto de Código 3.2. Ao contrário do HTML, o XML permite definir anotações próprias para a descrição da informação. Esta característica permite que um documento XML possa ser interpretado por aplicações, desde que conheçam a estrutura e o significado das suas anotações. Esta característica tornou o XML numa linguagem amplamente adoptada para troca de informação entre aplicações [20].

Excerto de Código 3.2 Exemplo de um documento XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.w3schools.com">
    <!-- Isto é um comentário -->
    <nota>
        <de>Emissor</de>
        <para>Receptor</para>
        <cabecalho>Mensagem</cabecalho>
        <corpo>Esta é uma mensagem de teste</corpo>
    </nota>
```

3.3 Serviços *Web*

Um serviço *Web* é um componente de *software* concebido para garantir interoperabilidade entre plataformas ligadas em rede. Esta característica resulta do facto dos serviços *Web* serem independentes de qualquer linguagem de programação, SO e plataforma de *hardware* [23]. De uma forma simples, os serviços *Web* adoptam a linguagem XML como suporte para o conjunto de tecnologias que utilizam: (*i*) o protocolo de troca de mensagens *Simple Object Access Protocol* (SOAP); (*ii*) a linguagem de descrição dos serviços *Web Service Description Language* (WSDL); e (*iii*) o sistema de publicação e pesquisa de serviços *Universal Description, Discovery and Integration* (UDDI). Na Figura 3.2 apresenta-se o ciclo de vida de um serviço *Web*: (*i*) o serviço *Web* é desenvolvido e o respectivo WSDL é publicado no serviço de registo de serviços *Web* UDDI; (*ii*) os clientes que pretendam consumir o serviço interrogam o UDDI e descarregam o ficheiro WSDL de descrição do serviço; (*iii*) o cliente interpreta o conteúdo do WSDL descarregado e interage directamente com o serviço através de mensagens SOAP.

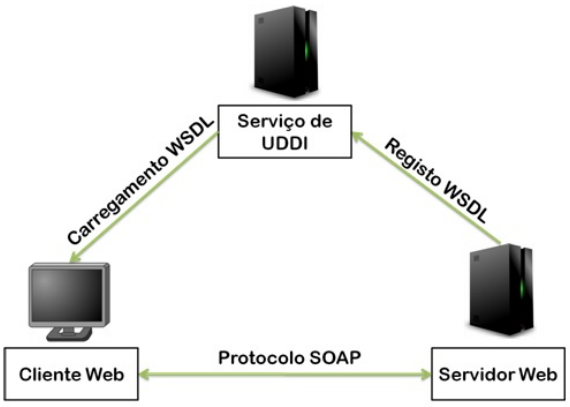

Figura 3.2: Ciclo de vida de um serviço *Web*.

Podem ser encontrados dois tipos de serviços *Web*:

Remote Procedure Call (RPC): São serviços que, com base no WSDL, invocam remotamente as operações disponíveis. O cliente interage com o serviço através do *Uniform Resource Identifier* (URI) especificado no WSDL.

REpresentational State Transfer (REST): São serviços que utilizam o protocolo *HyperText Transfer Protocol* (HTTP) para interagir com o serviço, recorrendo a mensagens HTTP GET, POST, PUT e DELETE. O cliente interage com os serviços através do URI especificado no WSDL.

3.3.1 *Simple Object Access Protocol*

O SOAP é um protocolo de nível de aplicação destinado à troca de mensagens XML encapsuladas em HTTP ou *Simple Mail Transfer Protocol* (SMTP) em ambientes distribuídos, independentemente do tipo de plataforma ou linguagem de programação adoptada. A troca de informação é feita de forma unidireccional, podendo ser processada por um ou mais nós. No entanto, podem ser implementados cenários de pedido/resposta como representado na Figura 3.3 [23].

A nível estrutural, uma mensagem SOAP é um documento XML que consiste num envelope constituído por um cabeçalho e um corpo. O cabeçalho é um elemento opcional na estrutura da mensagem, que contém informação de controlo. O corpo da mensagem SOAP contém a informação que deverá ser enviada para o receptor, podendo ou não conter campos opcionais de verificação e controlo de erros. No Extracto de Código 3.3 e no Extracto de Código 3.4 ilustra-se um pedido e resposta SOAP, respectivamente.

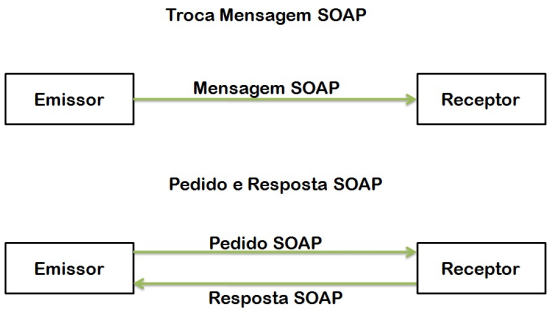

Figura 3.3: Troca de mensagens SOAP [23].

Excerto de Código 3.3 Exemplo de um pedido SOAP.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
"xmlns:ser="http://service.web.com/">
    <soapenv:Header/>
    <soapenv:Body>
        <ser:sendUser>
            <userId>user</userId>
            <userPassword>xxxxxxxx</userPassword>
        </ser:sendUser>
    </soapenv:Body>
</soapenv:Envelope>
```

Excerto de Código 3.4 Exemplo de uma resposta SOAP.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns3:sendUserResponse xmlns:ns3="http://service.web.com/">
        <return>
            <email>teste@teste.com</email>
            <name>Teste</name>
            <userId>user</userId>
            <userPassword>xxxxxxxx</userPassword>
        </return>
        </ns3:sendUserResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

3.3.2 *Web Service Description Language*

O WSDL é um dialecto de XML que descreve a interface de um serviço *Web*. É o alicerce de interacção com os serviços pois descreve os tipos de dados enviados, o tipo de operações que se podem invocar e, ainda, a correspondência entre as operações e portos associados ao URI do serviço. No Extracto de Código 3.5 é possível analisar a estrutura de um WSDL genérico.

Excerto de Código 3.5 Exemplo de estrutura de um WSDL.

```
<?xml version="1.0"?>
<definitions targetNamespace = "http://service.web.com/"
    name="WebServiceExemplo" xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:wsp="http://www.w3.org/ns/ws-policy" xmlns:tns="http://service.web.com/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <!-- Tipos abstractos -->
    <types>
    </types>
    <!-- Definição de mensagens abstractas -->
    <message>
    </message>
    <!-- Definição dos tipos de portos abstractos -->
    <portType>
    </portType>
    <!-- Definição das ligações concretas -->
    <binding>
    </binding>
    <!-- Definição do serviço Web concreto -->
    <service>
        <port>
        </port>
     </service>
</definitions>
```

3.4 *Back-end*

Numa arquitectura de *software* o *back-end* é o sistema de processamento remoto da aplicação. Neste projecto o *back-end* foi instalado na plataforma de *cloud computing* do tipo *Platform-as-a-Service* (PaaS) Google App Engine (GAE).

3.4.1 Google App Engine

O GAE é um servidor de aplicações e serviços *Web*. As aplicações desenvolvidas para esta plataforma são armazenadas na infra-estrutura de *cloud computing* da Google e têm como principal característica garantir a escalabilidade de armazenamento e processamento em função das necessidades da aplicação. O GAE possibilita o armazenamento de aplicações escritas em três linguagens de programação: Java, Python e GO. Além de garantir uma infra-estrutura de alojamento e processamento da aplicação, o GAE fornece ainda um conjunto de API específicas, *e.g.*, envio e recepção de *emails* ou agendamento de tarefas (*cron jobs*).

A nível de armazenamento de informação, designado Datastore, o GAE fornece um ambiente baseado em entidades. Neste ambiente deixam de existir tabelas de dados estruturadas. Cada objecto armazenado passa a ser de um determinado tipo (*kind*), podendo objectos do mesmo tipo ter propriedades diferentes. Uma propriedade pode ser de diferentes tipos (texto, booleano, numérico ou data).

O acesso aos dados é efectuado através de uma *query* ao tipo da entidade e às suas propriedades. Cada entidade possui um identificador único (*key*) baseado no tipo de entidade e nas suas relações. No Extracto de Código 3.6 exemplifica-se a criação de uma entidade e no Extracto de Código 3.7 ilustram-se exemplos de acções relativas a entidades armazenadas no GAE.

Excerto de Código 3.6 Criação de uma entidade GAE.

```
1    public static String createEntity(String id) {
2        Entity entity = getUser(id);
3        if (entity != null) {
4            return "Entity already exist";
5        } else {
6            // Criação da nova entidade
7            entity = new Entity("EntityName", id);
8            entity.setProperty("EntityProperty1", "property1");
9            persistEntity(entity);
10           return "Entity Created";
11       }
12   }
```

Excerto de Código 3.7 Operações de escrita, pesquisa e eliminação de entidades GAE.

```
1    // Método de escrita de uma entidade
2    public static void persistEntity(Entity entity) {
3        Transaction transaction = datastore.beginTransaction();
4        try {
5            datastore.put(entity);
6            transaction.commit();
7        } finally {
8            if (transaction.isActive()) {
9                transaction.rollback();
10           }
11       }
12   }
13   // Método para apagar uma entidade
14   public static void deleteEntity(Key key) {
15       datastore.delete(key);
16   }
17   // Método de pesquisa de uma entidade pela sua chave
18   public static Entity findEntity(Key key) {
19       try {
20           return datastore.get(key);
21       } catch (EntityNotFoundException e) {
22           return null;
23       }
24   }
25   // Método de pesquisa de uma entidade com base na sua chave e
26   // propriedades
27   public static Entity findEntity(String kind, String propertyName,
28           String propertyValue) {
29       Query q = new Query(kind);
30       q.setFilter(new FilterPredicate(propertyName, FilterOperator.EQUAL,
31               propertyValue));
32       List<Entity> results = getDataStoreService().prepare(q).asList(
33               FetchOptions.Builder.withDefaults());
34       if (!results.isEmpty()) {
35           return (Entity) results.remove(0);
36       }
37       return null;
38   }
```

A nível de manutenção de aplicações, o GAE fornece uma consola de administração. A consola de administração GAE é uma página *Web* onde o administrador

pode consultar os *logs* da aplicação, fazer uma análise e edição dos dados armazenados, consultar os recursos consumidos pelas aplicações, analisar tarefas pendentes, *etc.* É através desta consola que o administrador pode alterar características como o nome da aplicação e o nível de desempenho do servidor. Nas Figuras 3.4 e 3.5 estão representados os painéis da consola do administrador relativas aos recursos da aplicação e à base de dados.

Durante a fase de desenvolvimento é possível trabalhar num servidor local no porto 8888, ficando acessível através do endereço http://localhost:8888.

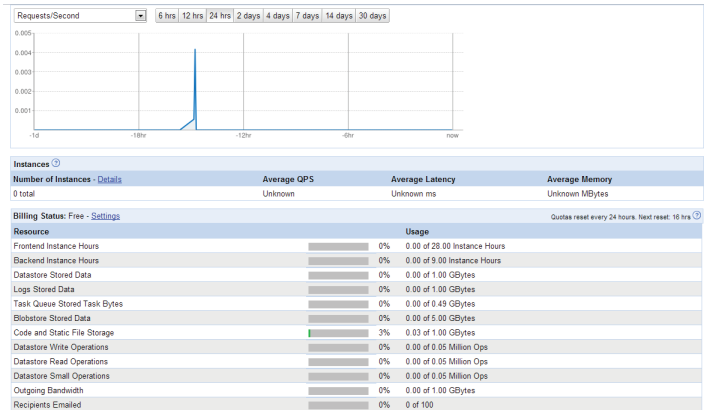

Figura 3.4: Consola de administração GAE: recursos.

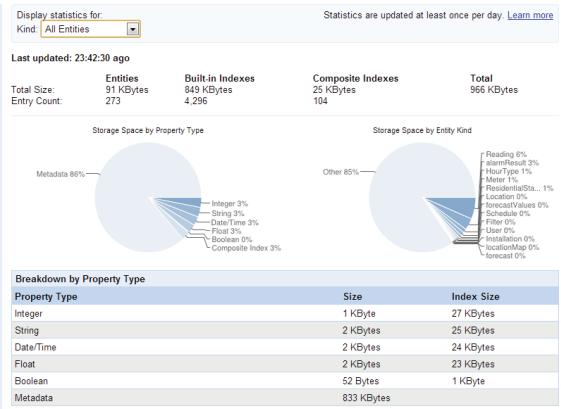

Figura 3.5: Consola de administração GAE: armazenamento.

3.5 *Front-end*

O *front-end* consiste nas aplicações de interface com o utilizador que, por sua vez, são clientes do sistema de *back-end*. Neste contexto, incluem-se as tecnologias de suporte ao desenvolvimento da aplicação móvel e da aplicação *Web*.

3.5.1 Linguagens

3.5.1.1 JavaScript

O JavaScript é uma linguagem de *scripting*, baseada em Java, de fácil compreensão e edição, os *scripts* podem ser declarados nos documentos HTML ou em ficheiros externos. Esta linguagem confere uma maior interactividade à página em desenvolvimento, permite a inclusão de objectos dinâmicos, como caixas de texto ou botões e permite efectuar validações ainda no lado do cliente. Para promover a reutilização de código e melhorar a organização dos documentos HTML, os *scripts* devem ser definidos em ficheiros externos que são carregados pelos documentos HTML. O JavaScript torna as páginas *Web* interactivas do lado do cliente efectuando a validação local de dados ou gerando código de personalizações. No Excerto de Código 3.8 ilustra-se um exemplo de código JavaScript.

Excerto de Código 3.8 Exemplo de código JavaScript.

```
1    <script type="text/javascript">
2        document.write("Olá Mundo!")
3    </script>
```

3.5.1.2 *Asynchronous JavaScript and XML*

O refrescamento ou alteração de uma página *Web* obriga por omissão ao seu recarregamento integral. O *Asynchronous JavaScript And* XML (AJAX) veio colmatar este comportamento, permitindo a actualização assíncrona de porções de uma página *Web* em função dos eventos gerados pelos utilizadores. O AJAX consiste num conjunto de tecnologias que agrupa XML, JavaScript para aceder ao *Document Object Model* (DOM) da página e o objecto `HTTP Request` para comunicar assincronamente com o servidor. Nesta abordagem os pedidos e resposta são mensagens XML. Se o evento gerado pelo utilizador desencadear uma simples validação de dados, o JavaScript processa localmente o pedido, caso contrário, é enviado um pedido assíncrono ao servidor [24]. As diferenças entre estas duas utilizações são apresentadas na Figura 3.6. O AJAX é utilizado em aplicações *Web* como o Google Maps ou o Google Earth.

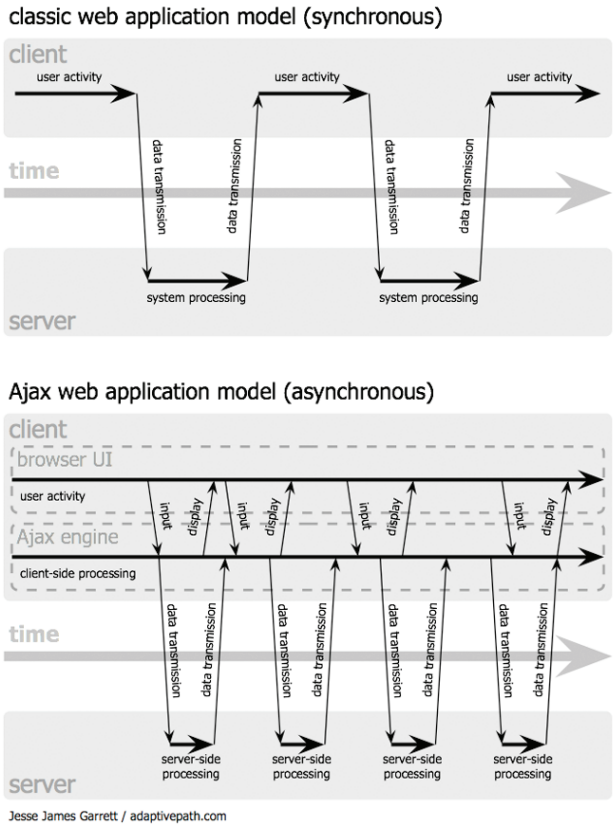

Figura 3.6: Método de funcionamento AJAX [24].

3.5.2 Android

O Android é uma plataforma de desenvolvimento para dispositivos móveis que integra um SO baseado em Linux. Esta plataforma encontra-se em expansão em Portugal e no mercado Europeu. Nas Figuras 3.7 e 3.8 apresenta-se a distribuição da utilização dos principais SO móveis na Europa e em Portugal, respectivamente, segundo o *Stat Counter* [25].

O desenvolvimento para a plataforma Android é feito em Java e é necessário um *kit* de desenvolvimento – *Software Development Kit* (SDK) – próprio que fornece as bibliotecas e as ferramentas necessárias à criação de aplicações. Existem várias versões do SDK Android que se apresentam na Tabela 3.1. Na Figura 3.9 está representada a distribuição dos utilizadores pelas diversas versões do SDK. A escolha do SDK procurou abranger o maior número possível de utilizadores. Dado que a compatibilidade de um SDK mais antigo com os SDK posteriores é garantida, escolheu-se uma versão inferior à mais recente. O desenvolvimento foi efectuado com o SDK Android 2.2 (*Froyo*).

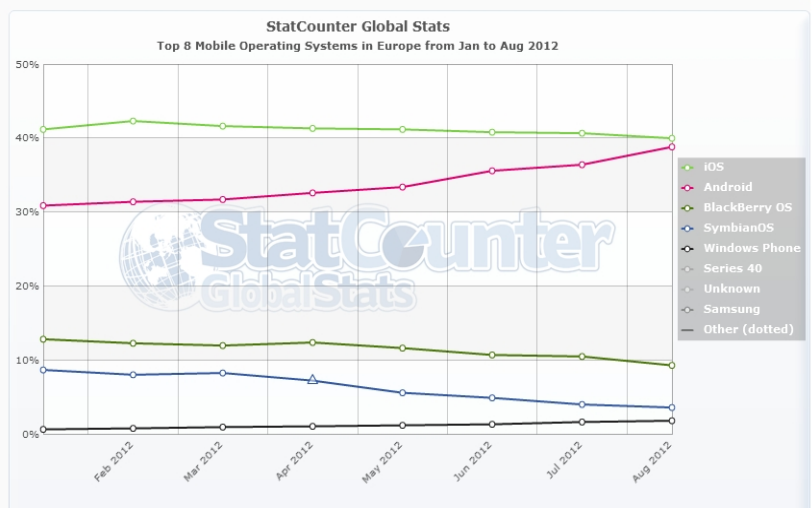

Figura 3.7: Evolução da distribuição de SO móveis na Europa (Jan-Ago 2012) [25].

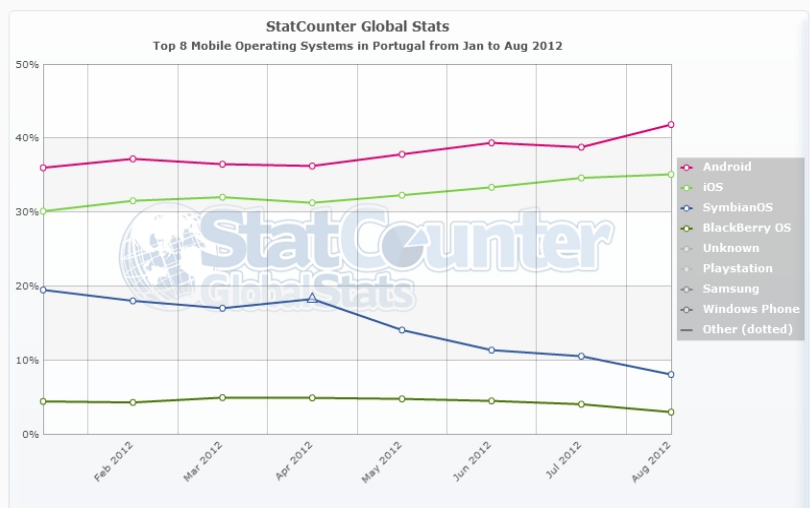

Figura 3.8: Evolução da distribuição de SO móveis em Portugal (Jan-Ago 2012) [25].

Dados os pressupostos da Secção 1.4, o equipamento deve ter acesso à Internet e incluir um sensor de *Global Positioning System* (GPS). O modelo seleccionado foi o htc Desire. Na Tabela 3.2 apresentam-se as principais características deste equipamento.

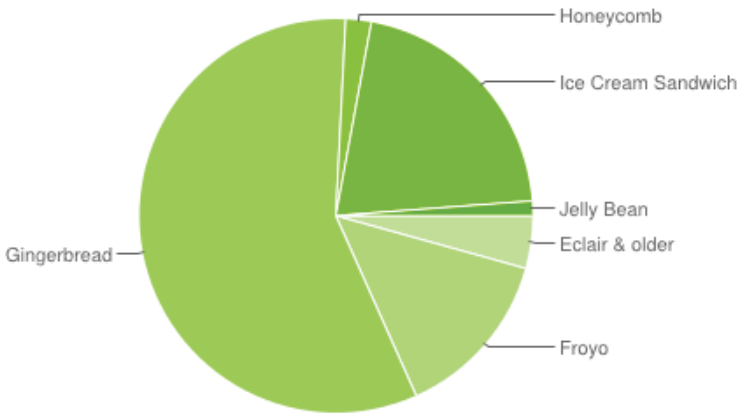

Figura 3.9: Distribuição do SDK Android (Set 2012) [26].

Tabela 3.1: Tabela SDK Android (Set 2012) [26].

Nome	**Versão**	**Nível da API**
Cupcake	1.5	3
Donut	1.6	4
Eclair	2.1	7
Froyo	2.2	8
Gingerbread	2.3 a 2.3.2	9
	2.3.3 a 2.3.7	10
Honeycomb	3.1	12
	3.2	13
Ice Cream Sandwich	4.0 a 4.0.2	14
	4.03 a 4.04	15
Jelly Bean	4.1	16

3.5.2.1 Actividades

Em Android uma aplicação é composta por várias actividades. Uma actividade corresponde a uma janela de interface com o utilizador – *Graphical User Interface* (GUI) – que se destina a um objectivo concreto. Quando é feito o arranque de uma aplicação Android é exibida a actividade principal (*main*) que poderá iniciar outras actividades. Cada aplicação só pode ter uma actividade visível. Cada actividade tem um ciclo de vida com quatro estados diferentes:

- **Em execução**: Actividade que está a ser utilizada pelo utilizador da aplicação.
- **Em pausa**: A actividade está em execução, podendo estar visível, mas está a decorrer outra que capta a atenção do utilizador. Este estado acontece, *e.g.*, quando um utilizador atende uma chamada telefónica. O sistema neste estado armazena todas as características da actividade incluindo a sua

Tabela 3.2: Características gerais do htc Desire.

Tamanho	Dimensões	119 x 60 x 11.9 mm
	Peso	135 g
Ecrã	Características	480 x 800 pixels; 3,7 ”
		Ecrã táctil
		Multi-touch
Processamento	Memória	576 MB RAM, 512 MB ROM
	Processador	1 GHz
Recursos	GPRS	
	Alarme	
	GPS	
	Bússola digital	
	Giroscópio	
	Bluetooth	
	Wi-Fi	

interface. A nível de *hardware* o sistema continua a consumir os recursos atribuídos à aplicação. Os recursos da actividade só são libertados se o sistema necessitar de memória.

- ***Parada***: A actividade continua em memória, mas deixa de ser armazenada a sua interface. Por exemplo, ao chamar uma nova actividade, a anterior fica parada.

- **Destruída ou finalizada**: Significa que os recursos associados à actividade foram libertados.

A comutação entre estados de uma actividade pode ser controlada através da reescrita de métodos apropriados [27].

- **onCreate()**: Método de preparação da fase de iniciação da actividade. Após a execução deste método a actividade fica “Em execução”.

- **onStart()**: Método de preparação do arranque da aplicação. Sempre que uma aplicação regressa do estado de paragem este é o ponto de restauro da actividade.

- **onResume()**: Método de preparação da retoma de uma actividade. Este é o ponto de regresso após uma pausa.

- **onPause()**: Método de preparação da pausa da actividade. É neste método que devem ser parados os processos que consomem *Central Processing Unit* (CPU) e memória.

- **onStop()**: Método de preparação de paragem da actividade.

- **onDestroy()**: Método de preparação da finalização da actividade. Pode ser executado quando o sistema destrói a actividade ou então porque foi executado o método `finish()`, significando que a actividade foi terminada pela própria aplicação.

Na Figura 3.10 está representado de forma gráfica o ciclo de vida de uma actividade. No Extracto de Código 3.9 está ilustrado um exemplo de criação de uma actividade Android. A gestão das actividades é feita através do manifesto. O manifesto é um documento XML que contém toda a informação sobre a aplicação, incluindo a lista de actividades e as permissões de execução da aplicação.

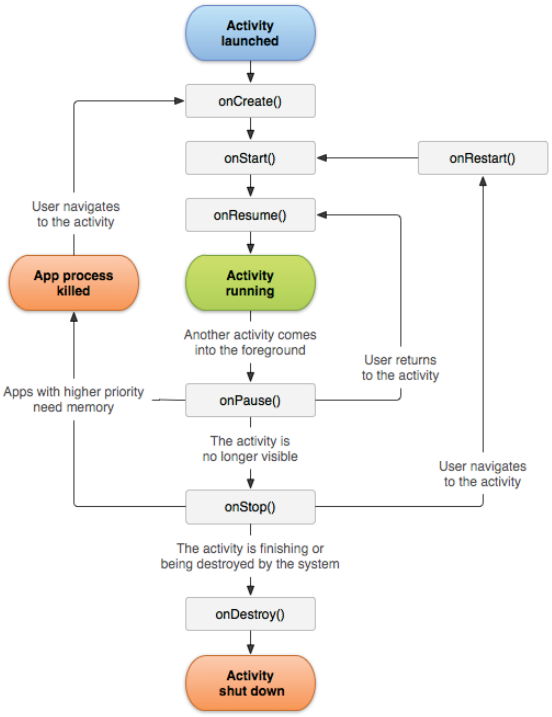

Figura 3.10: Ciclo de vida de uma actividade [27].

Excerto de Código 3.9 Exemplo de actividade Android.

```
1    public class TestActivity extends Activity {
2    /** Called when the activity is first created. */
3    @Override
4    public void onCreate(Bundle savedInstanceState) {
5        super.onCreate(savedInstanceState);
6        setContentView(R.layout.main);
7        }
8    }
```

3.5.2.2 Desenho da Interface

O *layout* consiste na configuração e disposição dos componentes gráficos da interface. Em Android, a definição de um *layout* é feita através de um documento XML que define o aspecto de uma actividade. Alternativamente, podem ser adicionados elementos visuais à actividade em tempo de execução. Esta abordagem permite não só uma melhor separação entre os componentes visuais e de controlo, mas também possibilita alterações à interface em função dos eventos gerados pelo utilizador. No Extracto de Código 3.10 apresenta-se a definição do *layout* ilustrado na Figura 3.11. Para criar a interface definida no Extracto de Código 3.10, basta invocar o método `setContentView()`, no método `onCreate()` da actividade. O Extracto de Código 3.11, exemplifica esta operação especificando o nome do ficheiro XML na forma `R.layout.nomeLayout`.

Excerto de Código 3.10 Exemplo de *layout* Android.

```
1   <RelativeLayout
2   xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:layout_width="match_parent"
5       android:layout_height="match_parent" >
6       <TextView
7           android:id="@+id/newText"
8           android:layout_width="wrap_content"
9           android:layout_height="wrap_content"
10          android:padding="@dimen/padding_medium"
11          android:text="@string/hello_world" />
12      <Button
13          android:id="@+id/newButton"
14          android:layout_width="wrap_content"
15          android:layout_height="wrap_content"
16          android:layout_centerHorizontal="true"
17          android:layout_centerVertical="true"
18          android:padding="@dimen/padding_medium"
19          android:text="@string/hello_world"/>
20  </RelativeLayout>
```

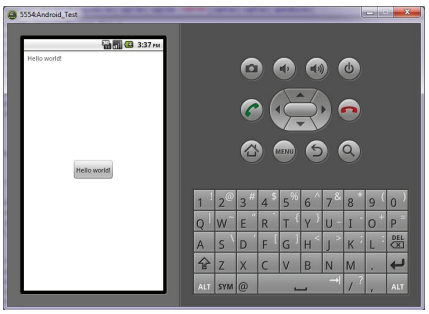

Figura 3.11: Exemplo de *Layout* de uma actividade Android.

Cada componente visual é identificado por um valor inteiro e é representado no `ficheiro de layout` como "`android:id=`". Este identificador permite referir os

Excerto de Código 3.11 Aplicação de um *layout* a uma actividade.

```
1    import android.os.Bundle;
2    import android.app.Activity;
3
4    public class MainActivity extends Activity {
5        @Override
6        public void onCreate(Bundle savedInstanceState) {
7            super.onCreate(savedInstanceState);
8            setContentView(R.layout.activity_main);
9        }
10   }
```

elementos visuais no código – ver o Extracto de Código 3.12. É ainda possível personalizar a posição de cada objecto gráfico em função do tipo de *layout* usado, e a associação a eventos, como se ilustra no Extracto de Código 3.12.

Excerto de Código 3.12 Referência ao objecto de *layout* no código da aplicação móvel.

```
1        Button helloWorld = (Button) findViewById(R.id.newButton);
2        helloWorld.setOnClickListener(new View.OnClickListener() {
3            @Override
4            public void onClick(View v) {
5                // TODO
6            }
7        });
```

3.5.2.3 SQLite

O SQLite é uma biblioteca que implementa um sistema de Base de Dados (BD) relacional *Structured Query Language* (SQL) local. A principal diferença entre o SQLite e um servidor de BD convencional SQL é que não é um servidor. Trata-se de uma aplicação local que lê e escreve os dados em ficheiros. A sintaxe dos comandos é idêntica aos servidores de BD comuns assim como os tipos de dados suportados. A implementação do SQLite é apenas recomendada para sistemas de baixa complexidade e com poucos acessos. Mais informações relativas à utilização desta tecnologia podem ser encontradas no próprio sítio oficial do SQLite (`http://www.sqlite.org/`).

3.5.2.4 kSOAP

O kSOAP é um cliente de serviços *Web* para aplicações Java J2ME[1]. Esta biblioteca permite desenvolver aplicações para a plataforma J2ME capazes de interagir com os serviços *Web*. No sítio oficial do kSOAP encontra-se mais informação acerca deste projecto (`http://ksoap2.sourceforge.net/`).

[1] *Java Plataform, Micro Edition* é a plataforma Java para dispositivos compactos como *smartphones*.

Para interagir com um serviço *Web* é necessário utilizar os seguintes elementos da API do kSOAP [28]:

`SoapPrimitive`: Classe que permite encapsular tipos primitivos de dados num envelope SOAP.

`SoapObject`: Classe que permite encapsular tipos de dados complexos no corpo de um envelope SOAP. Para criar um objecto desta classe é necessário conhecer o `namespace` e o nome do método que se pretende invocar. Depois de criado o objecto é necessário preenchê-lo com a informação desejada através do método `addProperty` como se demonstra no Extracto de Código 3.13.

Excerto de Código 3.13 Criação de um objecto `SoapObject`.

```
1   import org.ksoap2.serialization.SoapObject;
2   String METHOD_NAME = "method1";
3   String NAMESPACE = "http://service.web.com/";
4
5   // Criação do objecto da classe SoapObject
6   // que permitirá preencher o corpo do envelope SOAP
7   SoapObject request = new SoapObject(NAMESPACE, METHOD_NAME);
8
9   // Preenchimento das propriedades do objecto
10  request.addProperty(Nome da Propriedade 1, Valor1);
11  request.addProperty(Nome da Propriedade 2, Valor2);
```

`SoapSerializationEnvelope`: Classe utilizada para representar um envelope SOAP composto por cabeçalho e corpo. A diferença entre este objecto e o objecto `SoapEnvelope` é que este permite utilizar objectos serializados (gravação do estado de um objecto para utilização futura). Para criar um objecto desta classe é necessário identificar qual a versão do protocolo SOAP a utilizar. Depois de criado o objecto basta, através do método `setOutputSoapObject`, declarar o objecto SOAP a enviar na chamada ao serviço. No Extracto de Código 3.14 é demonstrada a criação de uma instância da classe.

Excerto de Código 3.14 Criação de um objecto `SoapSerializationEnvelope`.

```
1
2   import org.ksoap2.serialization.SoapSerializationEnvelope;
3
4   // Criação do objecto Envelope
5   SoapSerializationEnvelope soapEnvelope = new
6   SoapSerializationEnvelope(SoapEnvelope.VER11);
7
8   soapEnvelope.setOutputSoapObject(request); // Atribuição do
9                                              //objecto SOAP a enviar
```

`Marshal`: Interface que permite a serialização de objectos complexos. No Extracto de Código 3.15 é possível analisar o exemplo da utilização da seriali-

zação de objectos `Date` e `Float` através das classes MarshalDate e MarshalFloat, respectivamente. Depois de criados os objectos basta registá-los no envelope SOAP, utilizando o método `register`.

Excerto de Código 3.15 Serialização de objectos não primitivos em kSOAP.

```
1  import org.ksoap2.serialization.MarshalDate;
2  import org.ksoap2.serialization.MarshalFloat;
3
4  MarshalDate md = new MarshalDate();   // Criação de um objecto MarshalDate
5  md.register(soapEnvelope);            // Regista o objecto criado no envelope
6
7  // Processo semelhante com float
8  MarshalFloat mf = new MarshalFloat();
9  mf.register(soapEnvelope);
```

`Transport`: Classe abstracta usada pela camada de transporte herdada pelas subclasses `HttpTransport` e `HttpTransportSE`. A primeira classe permite trocar mensagens SOAP sobre HTTP em plataformas J2ME. A classe `HttpTransportSE` é semelhante à anterior, mas destina-se à plataforma J2SE[2]. No Extracto de Código 3.16 é ilustrada a utilização desta classe. Depois de criada uma instância da classe especificando o *Uniform Resource Locator* (URL) do serviço *Web* pode-se invocar a operação e enviar o envelope SOAP.

Excerto de Código 3.16 Utilização da classe de transporte do kSOAP.

```
1  import org.ksoap2.transport.HttpTransportSE;
2
3  String URL = "http://service.web.com/webservice";
4  String SOAP_ACTION = "http://service.web.com/method1";
5
6  // Criação do Objecto de transporte com a definição do URL de ligação
7  HttpTransportSE transport = new HttpTransportSE(URL);
8
9  transport.call(SOAP_ACTION, soapEnvelope); // Método para envio do
10                                            // pedido ao servidor
```

Após a invocação de uma operação de um serviço *Web* basta criar um objecto do tipo `SoapObject` ou `SoapPrimitive`, dependendo do tipo de dados que se espera receber, para obter a resposta. Depois de invocar o método `getResponse` do objecto `SoapEnvelope`, deve efectuar-se um *cast* para o tipo de dados pretendido. No Extracto de Código 3.17 ilustra-se um exemplo este processo.

3.5.2.5 AChartEngine

O AChartEngine é uma API gratuita para Android que permite a criação de gráficos. Os gráficos podem ser de vários tipos, *e.g.*, linhas, de barras ou tartes. A API pode ser descarregada a partir do sítio oficial do projecto na *Web*

[2] *Java Plataform, Standard Edition.*

Excerto de Código 3.17 Obtenção da resposta a um pedido SOAP através do kSOAP.

```
1    SoapPrimitive result = (SoapPrimitive)
2                soapEnvelope.getResponse(); // Resposta a um pedido SOAP
3                                            // resultado dados primitivos
4
5    /*
6    SoapObject response = (SoapObject)
7                    soapEnvelope.getResponse(); // Resposta a um pedido SOAP
8                                                // resultado dados complexos
9    */
```

(`http://www.achartengine.org/`). A versão é a 1.0.0. Nesta API, um gráfico é constituído por séries de dados que se designam `dataset`. No caso de um gráfico tarte cada fatia corresponde a uma série – ver Extracto de Código 3.18; no caso de num gráfico linear cada série representa uma linha do gráfico – ver Extracto de Código 3.19. Antes de se preencher o `dataset` é necessário definir um objecto para processar a renderização[3] do desenho. Cada série tem o seu objecto de renderização. Depois de configurados os `dataset`, é necessário criar o objecto que representa o gráfico através da classe `ChartFactory`. Este objecto permite ainda adicionar eventos ao gráfico *e.g.*, cliques. O resultado deve ser associado a um objecto de visualização Android como *e.g.*, *LinearLayout* ou um *RelativeLayout*. No Extracto de Código 3.20 exemplifica-se a configuração de um gráfico tarte e no Extracto de Código 3.21 ilustra-se a criação de um gráfico linear, Figuras 3.12 e 3.13, respectivamente.

Excerto de Código 3.18 Configuração de uma série num gráfico tarte.

```
1    // Criação do objecto de renderização do gráfico
2    private DefaultRenderer globalRenderer = new DefaultRenderer();
3    // Criação de objecto das séries
4    private CategorySeries series = new CategorySeries("");
5
6    public void dataSetDemo() {
7        // Implementação da primeira fatia/série
8        series.add("Test1", 1.0); // Adição dos valores da fatia
9
10       // Criação de um objecto de renderização para a série
11       SimpleSeriesRenderer seriesRenderer = new SimpleSeriesRenderer();
12       seriesRenderer.setColor(COLORS[(series.getItemCount() - 1) % COLORS.length]);
13
14       // Adição do objecto de renderização da série no objecto de renderização global
15       globalRenderer.addSeriesRenderer(seriesRenderer);
16
17       // Implementação da segunda fatia/série
18       series.add("Test2", 2.0);
19       SimpleSeriesRenderer seriesRenderer2 = new SimpleSeriesRenderer();
20       seriesRenderer2.setDisplayChartValues(true);
21       seriesRenderer2.setColor(COLORS[(series.getItemCount() - 1) % COLORS.length]);
22       globalRenderer.addSeriesRenderer(seriesRenderer2);
23       seriesRenderer2.setGradientStart(5.0, Color.BLUE);
24   }
```

[3]Processo digital que origina imagens.

Excerto de Código 3.19 Configuração de uma série num gráfico linear.

```
1   // Objecto de renderização do gráfico
2   private XYMultipleSeriesRenderer renderer = new XYMultipleSeriesRenderer();
3   // Objecto do dataset
4   private XYMultipleSeriesDataset dataset = new XYMultipleSeriesDataset();
5   public void dataSetDemo(){
6       // Criação do objecto de renderização das séries
7       XYSeriesRenderer seriesRenderer = new XYSeriesRenderer();
8       // Criação das séries
9       TimeSeries timeSeries = new TimeSeries("Test 1");
10      TimeSeries timeSeries2 = new TimeSeries("Test 2");
11      // Simulação de dados
12      Date date = new Date(2012, 1, 1, 0, 0);
13      timeSeries.add(date, 1.0); // Adição de dados à série
14      timeSeries2.add(date, 100.0);
15      date = new Date(2012, 1, 1, 20, 15);
16      timeSeries.add(date, 2.0);
17      timeSeries2.add(date, 200.0);
18      date = new Date(2012, 9, 15, 23, 52);
19      timeSeries.add(date, 4.0);
20      timeSeries2.add(date, 400.0);
21      dataset.addSeries(timeSeries); // Adição das séries ao dataset
22      dataset.addSeries(timeSeries2);
23      // Configuração dos objectos de renderização das séries
24      seriesRenderer.setColor(Color.BLUE);
25      seriesRenderer.setPointStyle(PointStyle.SQUARE);
26      seriesRenderer.setFillPoints(true);
27      renderer.addSeriesRenderer(seriesRenderer);
28      seriesRenderer = new XYSeriesRenderer();
29      seriesRenderer.setColor(Color.RED);
30      seriesRenderer.setPointStyle(PointStyle.CIRCLE);
31      seriesRenderer.setFillPoints(true);
32      renderer.addSeriesRenderer(seriesRenderer);
33  }
```

Excerto de Código 3.20 Configuração de um gráfico tarte.

```
1   public void buildChart(LinearLayout layout, final Context context, String title) {
2       // Configuração do objecto de renderização do gráfico
3       globalRenderer.setChartTitle(title);
4       globalRenderer.setApplyBackgroundColor(true);
5       globalRenderer.setBackgroundColor(Color.WHITE);
6       globalRenderer.setMargins(new int[] { 20, 20, 20, 20 });
7       globalRenderer.setZoomButtonsVisible(true);
8       globalRenderer.setSelectableBuffer(10);
9       globalRenderer.setClickEnabled(true);
10      globalRenderer.setLabelsColor(Color.BLACK);
11      globalRenderer.setAntialiasing(true);
12      // Criação do gráfico
13      final GraphicalView pieChart = ChartFactory.getPieChartView(
14              context, series, globalRenderer);
15      pieChart.setBackgroundColor(Color.WHITE);
16      // Configuração do objecto de visualização Android que irá conter o gráfico
17      layout.addView(pieChart, new LayoutParams(
18                      LayoutParams.WRAP_CONTENT,LayoutParams.WRAP_CONTENT));
19      // Adição do evento ao objecto gráfico que exibe o valor do ponto clicado
20      pieChart.setOnClickListener(new View.OnClickListener() {
21          @Override
22          public void onClick(View v) {
23              SeriesSelection seriesSelection =
24                      pieChart.getCurrentSeriesAndPoint();
25              if (seriesSelection != null) {
26                  Toast.makeText(
27                      context,
28                      series.getCategory(seriesSelection.getPointIndex())
29                      + ": " + seriesSelection.getValue(),
30                      Toast.LENGTH_SHORT).show();
31              }
32          }
33      });
34  }
```

Figura 3.12: Exemplo de um gráfico tarte.

Excerto de Código 3.21 Configuração de um gráfico linear.

```
1   public void buildChart(LinearLayout layout, final Context context,
2           String title, String yTitle){
3       // Configuração do objecto de renderização do gráfico
4       renderer.setChartTitle(title);
5       renderer.setChartTitleTextSize(20);
6       renderer.setApplyBackgroundColor(true);
7       renderer.setBackgroundColor(Color.WHITE);
8       renderer.setAntialiasing(true);
9       renderer.setMargins(new int[] { 20, 20, 20, 20 });
10      renderer.setMarginsColor(Color.WHITE);
11      renderer.setZoomButtonsVisible(true);
12      renderer.setStartAngle(90);
13      renderer.setClickEnabled(true);
14      renderer.setSelectableBuffer(10);
15      renderer.setLabelsColor(Color.BLACK);
16      renderer.setYTitle(yTitle);
17      // Criação do gráfico
18      final GraphicalView timeChart = ChartFactory.getTimeChartView(
19              context, dataset, renderer, "dd-MM hh:mm");
20      timeChart.setBackgroundColor(Color.WHITE);
21      // Configuração do objecto de visualização Android que
22      // irá conter o gráfico
23      layout.addView(timeChart, new LayoutParams(LayoutParams.WRAP_CONTENT,
24              LayoutParams.WRAP_CONTENT));
25      // Adição de um evento ao objecto gráfico
26      // que exibe o valor do ponto clicado
27      timeChart.setOnClickListener(new View.OnClickListener() {
28          @Override
29          public void onClick(View v) {
30              SeriesSelection seriesSelection = timeChart
31                      .getCurrentSeriesAndPoint();
32              if (seriesSelection != null) {
33                  Toast.makeText(
34                          context, dataset.getSeriesAt(seriesSelection.getSeriesIndex()).getTitle()
35                          + ":" + seriesSelection.getValue(),
36                          Toast.LENGTH_SHORT).show();
37              }
38          }
39      });
40  }
```

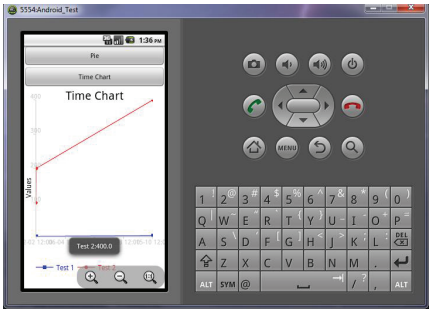

Figura 3.13: Exemplo de um gráfico linear.

3.5.3 Aplicação *Web*

O desenvolvimento da aplicação *Web* suporta-se num conjunto de tecnologias que incluem o Google Web Toolkit (GWT), a API Google Visualization e a API Google Maps.

3.5.3.1 Google Web Toolkit Framework

O GWT é um conjunto de ferramentas para o desenvolvimento de páginas *Web*. Permite desenvolver as páginas integralmente em linguagem Java com elementos que recorrem a *JavaScript*. Além desta característica, o GWT permite efectuar a depuração do código durante a fase de desenvolvimento.

O GWT possibilita a escrita de código quer para o lado do cliente (navegadores *Web*), quer do lado do servidor (aplicação servidora). O processo de comunicação entre cliente e servidor é realizado através de um serviço RPC. Para efectuar o processo de comunicação são necessárias três classes:

- Uma interface definida do lado do cliente que especializa a interface *RemoteService*, onde se declaram as assinaturas de todos os métodos do serviço – ver Extracto de Código 3.22;

- Uma interface assíncrona com o serviço definida do lado do cliente para assegurar que as chamadas aos métodos do serviço são feitas de forma assíncrona – ver Extracto de Código 3.23;

- Uma classe do lado do servidor que especializa *RemoteServiceServlet* e implemente a interface *RemoteService* criada – ver Extracto de Código 3.24.

Excerto de Código 3.22 Declaração da interface dos métodos de um serviço GWT.

```
1    @RemoteServiceRelativePath("serviceExample")
2    public interface ServiceExample extends RemoteService {
3        String sayHello() throws IllegalArgumentException;
4        String sayGoodBye() throws IllegalArgumentException;
5    }
```

Excerto de Código 3.23 Declaração de uma interface assíncrona.

```
1    @RemoteServiceRelativePath("serviceExample")
2    public interface ServiceExampleAsync {
3        String sayHello(AsyncCallback<String> callback) throws IllegalArgumentException;
4        String sayGoodBye(AsyncCallback<String> callback) throws IllegalArgumentException;
5    }
```

Excerto de Código 3.24 Implementação de uma interface GWT.

```
1    public class WSIntermediate extends RemoteServiceServlet
2    implements ServiceExample {
3        @Override
4        public String sayHello(){
5            return "Hello";
6        }
7        @Override
8        public String sayGoodBye(){
9            return "Good Bye";
10       }
11   }
```

3.5.3.2 Google Web Toolkit Designer

O GWT Designer é uma ferramenta que permite criar interfaces para o GWT do tipo "*What you see is what you get*". Apresenta um painel de criação onde é possível arrastar os componentes gráficos e dispô-los no contentor sem necessidade de escrever código. No entanto, o GWT Designer fornece o código gerado para análise e edição. Além da criação dos *layouts*, é possível configurar os objectos adicionados e implementar eventos. As principais áreas de trabalho oferecidas pelo GWT Designer são:

1. **Estrutura**: Painel de verificação da estrutura do *layout* em desenvolvimento de forma hierárquica com as relações entre os componentes inseridos.

2. **Paleta**: Painel em que se encontram os diversos objectos gráficos que se podem inserir na interface.

3. **Propriedades**: Painel que permite visualizar e editar as diversas propriedades e eventos dos componentes inseridos

4. **Área de desenho**: Painel que apresenta o *layout* em desenvolvimento.

Na Figura 3.14 apresenta-se o ambiente de desenvolvimento oferecido pelo GWT Designer indicando-se as áreas descritas.

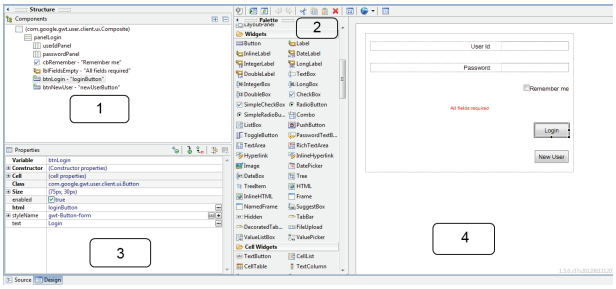

Figura 3.14: Exemplo de *layout* de uma actividade Android.

3.5.3.3 Google Visualization API

É uma API para o GWT baseada em JavaScript que permite a inclusão de gráficos numa página *Web*. Esta API providencia diversos tipos de gráficos (tarte, linear, barras, mapas, *etc.*). Os gráficos são funções de JavaScript que apresentam imagens vectoriais processadas em HTML5. Esta característica garante a compatibilidade com clientes *Web* e clientes móveis, como Android ou iPhone. As imagens geradas através desta API garantem um elevado nível de interacção com o utilizador, através do processamento de eventos [29]. Esta API pode ser encontrada no *site* `http://code.google.com/p/gwt-google-apis/downloads/list`.

A especificação dos valores a apresentar nos gráficos é efectuada através de tabelas de dados (`Data Tables`) JavaScript. Nos Extractos de Código 3.25 e 3.26 estão representados exemplos da definição de tabelas de dados para um gráfico de tarte e um gráfico linear, respectivamente. No caso do gráfico de tarte os valores da primeira coluna correspondem às designações de cada fatia e os valores da segunda coluna representam o valor da fatia, que será apresentado em valor percentual. No segundo caso, a primeira coluna contém o intervalo temporal e as duas colunas restantes as séries de dados a representar.

É ainda possível configurar diversas propriedades dos gráficos através do objecto `Options` como o título, as legendas ou o tamanho do gráfico. No Extracto de Código 3.27 efectua-se a criação e configuração do objecto para o tipo de gráfico tarte.

Para finalizar, é necessário criar a imagem do gráfico. Este processo é ilustrado no Extracto de Código 3.28 e consiste na realização de uma chamada aos servidores da Google submetendo os dados do gráfico. Nas Figuras 3.15 e 3.16 são apresentados os exemplos de criação de um gráfico de tarte e *AnnotatedTimeLine*, respectivamente.

Excerto de Código 3.25 Configuração de um gráfico tarte.

```
1    private AbstractDataTable createDataset() {
2            DataTable data = DataTable.create();
3
4        // Adição das colunas
5        data.addColumn(ColumnType.STRING, "Test Description");
6        data.addColumn(ColumnType.NUMBER, "Test Value");
7        data.addRows(2); //Número de linhas a adicionar
8
9        // Configuração da primeira fatia da tarte
10       data.setValue(0, 0, "Test1");
11       data.setValue(0, 1, 1);
12
13       // Configuração da segunda fatia da tarte.
14       data.setValue(1, 0, "Test2");
15       data.setValue(1, 1, 2);
16       return data;
17   }
```

Excerto de Código 3.26 Configuração um gráfico `AnnotatedTimeLine`.

```
1    private AbstractDataTable createDataset() {
2        DataTable data = DataTable.create();
3
4        // Adição das colunas
5        data.addColumn(ColumnType.DATE, "Test Description");
6        data.addColumn(ColumnType.NUMBER, "Test Value 1");
7        data.addColumn(ColumnType.NUMBER, "Test Value 2");
8        data.addRows(3); //Número de linhas a adicionar
9
10       // Configuração dos valores da primeira coluna, eixo XX
11       Date date = new Date();
12       date.setTime(1325376000);
13       data.setValue(0,0, date);
14       date.setTime(1325459700);
15       data.setValue(1,0, date);
16       date.setTime(1328451300);
17       data.setValue(2,0, date);
18
19       // Configuração da série "Test Value 1"
20       data.setValue(0, 1, 1);
21       data.setValue(1, 1, 2);
22       data.setValue(2, 1, 3);
23
24       // Configuração da série "Test Value 2"
25       data.setValue(0, 2, 100);
26       data.setValue(1, 2, 200);
27       data.setValue(2, 2, 300);
28
29       return data;
30   }
```

Excerto de Código 3.27 Configuração das opções de um gráfico tarte.

```
1    private Options createOptions() {
2
3        PieOptions options = PieChart.PieOptions.create();
4
5        options.setWidth(400);
6        options.setHeight(400);
7        options.set3D(true);
8        options.setTitle("Pie Chart Example");
9
10       return options;
11   }
```

Excerto de Código 3.28 Criação de um gráfico tarte.

```
1    public PieChartBuilder(final Panel panel){
2         Runnable onLoadCallback = new Runnable() {
3            public void run() {
4
5                // Criar o gráfico
6                PieChart pie = new PieChart(createDataset(), createOptions());
7
8                // Adicionar o gráfico ao painel
9                panel.add(pie);
10           }
11       };
12       // Criar o gráfico após o carregamento da API
13       VisualizationUtils.loadVisualizationApi(onLoadCallback, PieChart.PACKAGE);
14   }
```

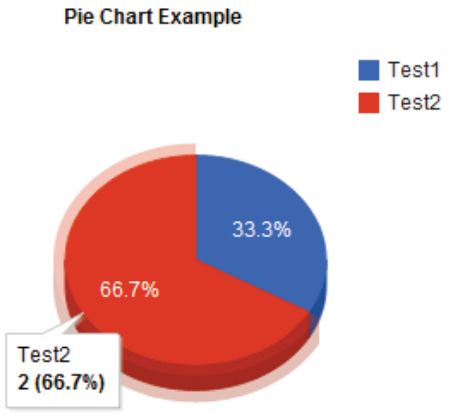

Figura 3.15: Exemplo de um gráfico tarte.

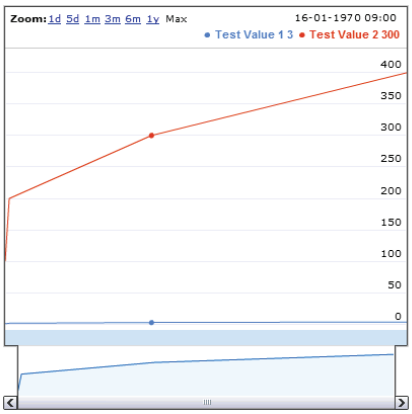

Figura 3.16: Exemplo de um gráfico *AnnotatedTimeLine*.

3.5.4 Google Maps API

É uma API que permite a inclusão de mapas personalizados do *Google Maps* em páginas *Web* ou aplicações móveis.

A utilização desta API no âmbito de aplicações *Web* obriga à obtenção de uma chave própria associada a uma conta Google. Para a obtenção de uma chave de acesso à API é necessário utilizar a ferramenta `keytool` para criar um par de chaves para a aplicação. Depois de obtida a chave há-que adicioná-la ao *layout* que incorpora o mapa. Por último é necessário o ficheiro para instalar a biblioteca externa do GWT, disponível no sítio oficial das API do Google Web Toolkit (`http://code.google.com/p/gwt-google-apis/downloads/list`).

3.6 Ambiente de Desenvolvimento Integrado

O desenvolvimento de aplicações computacionais fica consideravelmente facilitado com a utilização de ambientes de desenvolvimento integrado[4]. Os principais IDE *open source* para desenvolvimento em Java são o Eclipse, disponível em `http://www.eclipse.org/downloads` e o NetBeans, `http://netbeans.org/downloads/`. Uma vez que existem *plug-ins* específicos das ferramentas Google para o Eclipse adoptou-se este IDE para efeitos de desenvolvimento da solução.

O desenvolvimento na linguagem Java requer a instalação da plataforma Java - Java Standard Edition Development Kit (J2SE JDK) disponível em `http://www.oracle.com/technetwork/java/javase/downloads/index.html`. Depois da descarga e instalação do J2SE SDK deve-se proceder à sua configuração, segundo as instruções disponíveis no sítio oficial da linguagem.

3.6.1 Adição de *Plug-ins*

O Eclipse permite a adição de *plug-ins* que facilitam o desenvolvimento de aplicações para determinados ambientes. Um *plug-in* é um módulo de *software* concebido para adicionar funcionalidades a aplicações de maior dimensão. Para adicionar um *plug-in* ao Eclipse basta seguir os seguintes passos:

1. No separador *Help* aceder à opção "*Install New Software*";
2. Na nova janela clicar em "*Add*" e especificar o URL do *plug-in* e adicionar um novo repositório de informação – ver Figura 3.17;
3. Seleccionar o(s) *plug-in*(s) a adicionar ao Eclipse.

[4]Do inglês *Integrated Development Environment* (IDE)

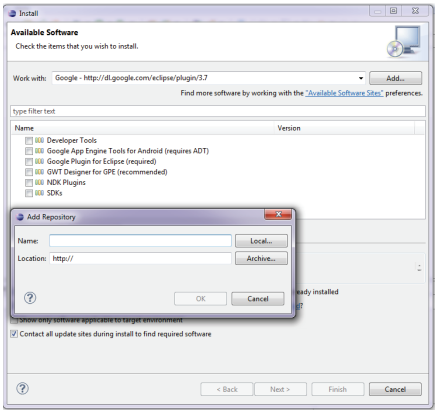

Figura 3.17: Adição de *plug-in* ao Eclipse.

3.6.2 Google *Plug-in*

A melhor forma de utilizar as ferramentas Google no Eclipse é através da instalação do *plug-in Google Plugin for Eclipse*, `https://dl.google.com/eclipse/plugin/`. Este *plug-in* contém tudo o que é necessário para se desenvolver com as ferramentas GAE e GWT. O GWT *Designer* tem também um *plug-in* para o GWT que se encontra disponível através do URL `http://dl.google.com/eclipse/inst/d2gwt/latest/`.

A criação de um novo projecto GAE ou GWT é feito através do botão da barra de ferramentas "*Google Services and Development Tools*" que fica disponível após a instalação do *plug-in*. Seleccionando-se a opção "*New Web Application Project...*", configurado-se o projecto e indicando-se se o projecto utiliza o GAE, o GWT ou ambos – ver Figura 3.18.

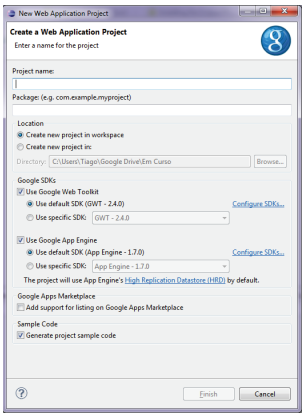

Figura 3.18: Criação de novo projecto Google *Plug-in*.

Para exportar um projecto para o *App Engine*, basta pressionar o mesmo botão, seleccionar a opção "*Deploy to App Engine*" e configurar as opções do projecto – ver Figura 3.19.

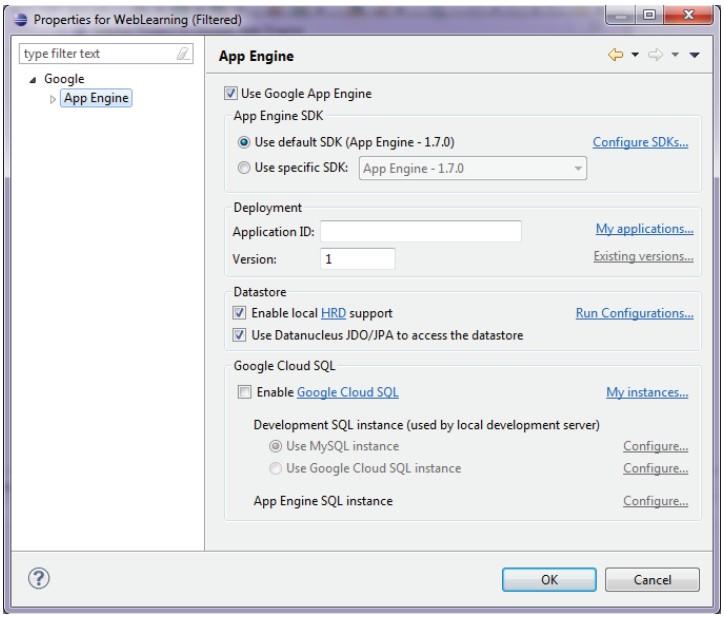

Figura 3.19: Migração do desenvolvimento para o *App Engine*.

3.6.3 Android SDK

O desenvolvimento para a plataforma Android requer a instalação do SDK do Android que está disponível em `http://developer.android.com/sdk/index.html`. O pacote descarregado contém as ferramentas do SDK que incluem o Android SDK Manager. O Android SDK Manager permite seleccionar as API e o SDK específico que se pretende descarregar e instalar – ver Figura 3.20. O Android SDK possui um *plug-in* para o Eclipse designado Android Development Tools (ADT) que está disponível em `https://dl-ssl.google.com/android/eclipse/`. A criação e desenvolvimento de projectos para Android no Eclipse após a instalação do ADT é transparente para o utilizador.

A criação de um novo projecto Android no Eclipse segue os seguintes passos:

1. Na barra de ferramentas seleccionar "*File*" -> "*New*" -> "*Project*" e na nova janela escolher a opção "*Android Application Project*";

2. Na nova janela introduzir o nome do projecto e os SDK seleccionados para a aplicação – ver Figura 3.21.

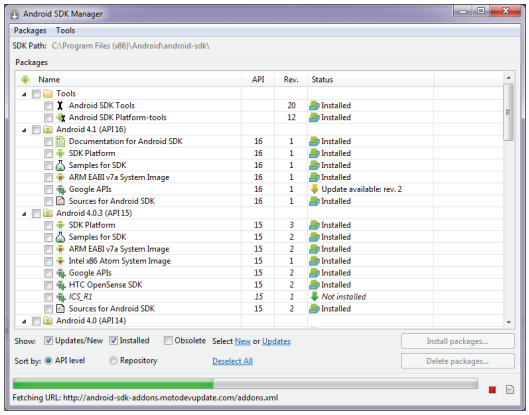

Figura 3.20: Aplicação do Android SDK Tools.

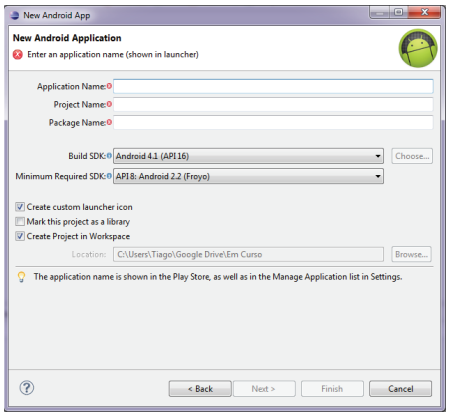

Figura 3.21: Criação de um projecto Android.

As aplicações Android podem ser executadas num dispositivo físico configurando o equipamento para o modo de depuração USB (Definições -> Aplicações -> Desenvolvimento e seleccionar a opção Depuramento USB) ou através de um emulador. Para configurar um emulador no Eclipse carrega-se no botão Android *Virtual Device manager* (AVD) da barra de ferramentas e configurar-se um novo dispositivo, Figura 3.22.

3.6.4 Adição de Bibliotecas

Nos projectos Eclipse a adição de bibliotecas externas obriga a uma reconfiguração do *build path* da aplicação. A adição de bibliotecas externas consiste nas seguintes operações:

1. Criação de uma pasta com o nome "*libs*" na raiz do projecto.

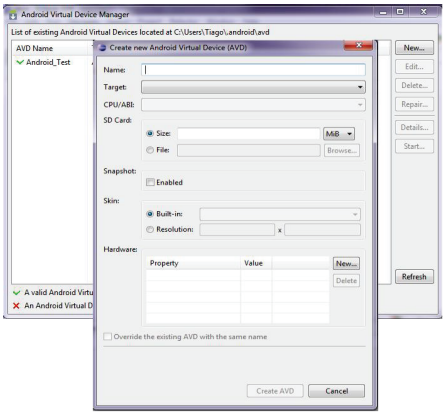

Figura 3.22: Criação de um emulador Android.

2. Inclusão das bibliotecas ao directório criado.

3. Na barra de ferramentas do Ecplise seleccionar a opção "*Project*" -> "*Properties*", na árvore da janela que aparece, seleccionar a opção "*Java Build Path*" – ver Figura 3.23.

4. Seleccionar a opção "*Add External JARs...*" e navegar até ao directório criado com as bibliotecas desejadas.

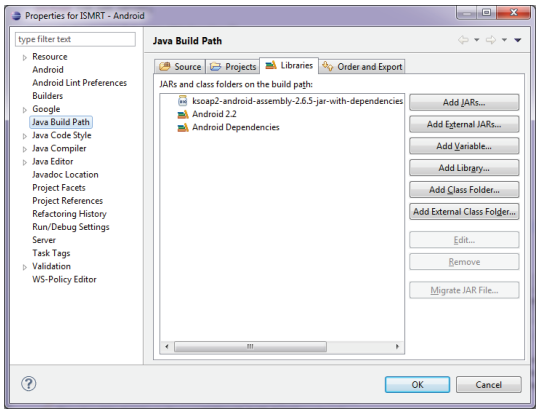

Figura 3.23: Janela de configuração do "*Build Path*" do Ecplise.

3.7 Conclusão

Foram apresentadas as tecnologias utilizadas no desenvolvimento do sistema. Em termos de linguagens existe uma forte componente Java no desenvolvimento An-

droid e *Web* assim como XML nas definições de *layout* Android e no protocolo SOAP. A nível de *back-end*, as soluções tecnológicas passam pela adopção da plataforma de *cloud computing* da Google, que disponibiliza capacidade de processamento e de armazenamento. A nível de *front-end* seleccionou-se a plataforma Android, o SQLite e o kSOAP para a aplicação móvel e o GWT Design e o GWT para o desenvolvimento da aplicação *Web*. Em ambos os casos recorreu-se a bibliotecas auxiliares para a criação dos gráficos. Por último, o IDE escolhido foi o Eclipse porque possui *plug-ins* para todas as tecnologias seleccionadas: Android, GWT e GAE.

No capítulo seguinte descreve-se o desenvolvimento do sistema com base nas tecnologias seleccionadas.

Capítulo 4

Desenvolvimento do Sistema

Neste capítulo descreve-se o desenvolvimento do sistema apresentando-se a arquitectura, as funcionalidades dos seus componentes, incluindo as aplicações de interface Android e Web de front-end, o sistema de back-end alojado no GAE, os testes e a depuração efectuados e os resultados obtidos.

4.1 Arquitectura

O sistema envolve desde o equipamento de contagem de energia eléctrica, à recolha, armazenamento, processamento e apresentação de informação ao utilizador. A arquitectura da plataforma está organizada em dois grandes blocos: *back-end* e *front-end*. O *front-end* inclui as aplicações Android e *Web* de interface com o utilizador e permite a recolha, armazenamento temporário e apresentação de dados. O *back-end* é constituído pelo serviço e pela base de dados alojada no GAE e fornece a capacidade de armazenamento e processamento de dados.

A secção 1.4 permite identificar quais as principais características que devem ser contempladas na arquitectura da plataforma a desenvolver. Segundo o documento *Smart Grid Data Cloud* [30] estima-se que as *smart grids* originem dezenas de *gigabytes* de dados por dia. Estes dados obrigam a que o sistema a desenvolver tenha um grande potencial de armazenamento e processamento de informação. Tornou-se imperativo a utilização de sistemas distribuídos, assegurando capacidade de armazenamento e de processamento escaláveis e a independência das características dos equipamentos de acesso à informação. Uma vez que todo o *back-end* se encontra alojado num único nó central, existe a necessidade de manter o sistema sempre activo e com *backups* constantes. A Tabela 4.1 enumera as características enunciadas.

Tabela 4.1: Comparação entre sistemas distribuídos e *stand-alone*.

Modelo	***Cloud***	***Stand-alone***
Armazenamento escalável	✓	
Portabilidade	✓	
Independência de servidores externos		✓
Custo de manutenção	✓	✓

O método de colheita de dados é misto. A integração dos contadores C&I com AMR na plataforma é efectuada através do ACE Vision 2.4. Contudo, nos contadores residenciais esta possibilidade não se encontra disponível, o que obriga à adopção da recolha manual. Na Tabela 4.2 compara-se o impacto de cada abordagem.

Tabela 4.2: Comparação entre recolha manual e automática.

Recolha	**Automática**	**Manual**
Processo automático	✓	
Possibilidade de erros		✓
Custos associados	✓	

A arquitectura do sistema é detalhada na Figura 4.1. O *front-end* inclui duas aplicações:

Aplicação *Web*: Permite ao utilizador das instalações residenciais aceder às funções da plataforma através de um dispositivo com acesso à Internet. Incorpora as funcionalidades de gestão de utilizadores, instalações e tectos de consumo. Permite ainda a submissão, análise e comparação, interagindo com o serviço Google Maps, de leituras dos contadores.

Aplicação Móvel: Fornece as mesmas funcionalidades que a aplicação *Web*, quer para instalações residenciais, quer a instalações industriais. As instalações industriais não foram incluídas nas funcionalidades da aplicação *Web* porque já existe uma aplicação Web – ACE Vision – para este tipo de instalações. Permite o agendamento de leituras residenciais através do uso de notificações e o armazenamento local da informação dos utilizadores, instalações, filtros de pesquisa e modelos de previsão para minimizar o volume de dados trocado com o servidor. Esta informação é igualmente armazenada no sistema central e pode ser devolvida ao dispositivo móvel caso o utilizador assim o pretenda.

O *back-end* suporta dois módulos:

Módulo App Engine: Disponibiliza os serviços *Web* de interacção com a BD do ACE Vision e a BD do Datastore. O sistema ACE Vision, que também

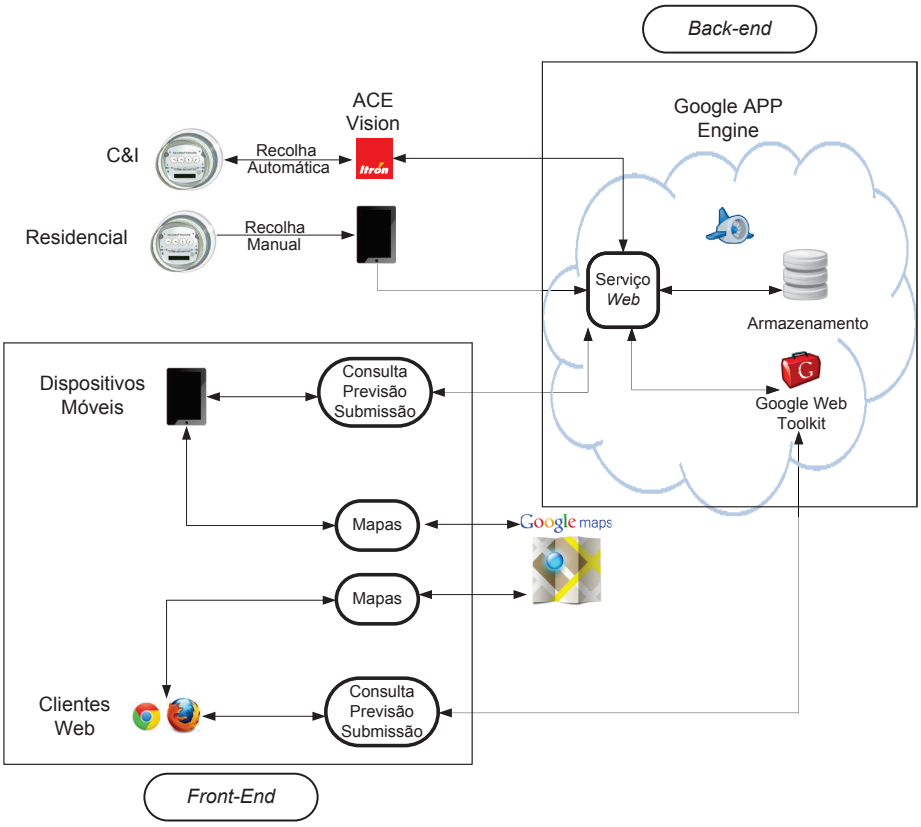

Figura 4.1: Arquitectura do sistema.

expõe um serviço *Web* para interacção, armazena a informação relativa aos utilizadores com contadores C&I. O servidor GWT também está alojado neste nó. Todas as operações de leitura e escrita no sistema de armazenamento são feitas recorrendo a este serviço. Tanto a aplicação *Web* como o dispositivo móvel invocam as operações do serviço *Web* para realizarem as suas tarefas. A interacção baseia-se num sistema de pedido/resposta ao sistema central. O acesso à informação concentrada no servidor ACE Vision encontra-se acessível através do mesmo método. No entanto, apenas são possíveis operações de leitura à mesma.

Módulo Datastore: O Datastore comporta os dados de utilizadores, instalações, contadores, leituras de contadores residenciais, filtros e tarefas.

4.2 Funcionalidades

As funcionalidades criadas dividem-se em dois conjuntos: (i) gestão da informação armazenada no Datastore dos utilizadores, instalações, contadores e leituras;

(*ii*) e processamento da informação, incluindo a análise e previsão de consumo e produção.

4.2.1 Utilizadores

Os utilizadores são representados na memória volátil através da classe `Utilizador` que possui a estrutura de dados apresentada na Tabela 4.3.

Tabela 4.3: Estrutura de dados da classe `Utilizador`.

Campo	Tipo
id	`String`
nome	`String`
password	`String`
email	`String`

As funções de gestão dos utilizadores são:

Criação: A criação de um utilizador obriga ao preenchimento da estrutura da Tabela 4.3. O utilizador deve introduzir um *username*, o seu nome, endereço de *email*, senha e a sua confirmação. Ao concluir este processo, o sistema verifica localmente se as iniciais do nome estão em maiúsculas, a estrutura do *email*, se a senha contém pelo menos 8 caracteres e se foi correctamente confirmada. Depois desta análise ao formulário, é enviado um pedido de criação de um novo utilizador ao serviço, que verifica se o *username* escolhido está disponível e, em função do resultado, confirma ou rejeita o pedido.

Validação: Após a criação do utilizador, o serviço envia um *email* de confirmação para o endereço inserido. No corpo da mensagem é introduzida uma hiperligação para a página do serviço responsável por este processo de confirmação. A confirmação é realizada com base na data exacta da criação e no nome do utilizador inseridos na hiperligação. Enquanto não proceder a esta validação o utilizador apenas pode alterar o endereço de *email*.

Edição: O utilizador pode alterar todos os seus dados, exceptuando o seu *username*. Este processo acarreta uma nova validação dos dados inseridos, quer do lado do cliente, quer do lado do servidor. No caso da alteração do endereço de correio electrónico é mesmo necessário repetir o processo de validação no servidor.

Eliminação: A remoção de um utilizador implica a eliminação total da informação armazenada no servidor.

Recuperação: A recuperação dos dados do utilizador consiste na actualização da informação armazenada na base de dados do dispositivo móvel com os dados guardados no Datastore, uma vez que podem ter sido alterados através da aplicação *Web*.

Gestão de Sessões: A utilização tanto da aplicação móvel como da aplicação *Web* requer a autenticação do utilizador que resulta na criação de uma sessão. Enquanto que no caso da aplicação *Web* a validação é feita directamente no sistema central, no caso do dispositivo móvel a validação é feita localmente. Para evitar o transtorno de proceder à autenticação todas as vezes que a aplicação é acedida, o utilizador pode armazenar localmente os dados de sessão. No dispositivo móvel esta funcionalidade é implementada recorrendo ao objecto `SharedPreferences` que permite guardar determinadas propriedades em memória mesmo depois da aplicação ser encerrada. O Excerto de Código 4.1 exemplifica este processo. Na aplicação *Web* a gestão das sessões é feita recorrendo a *cookies*. Por omissão, a duração de uma sessão é de uma hora. No entanto, caso o utilizador pretenda manter a informação de sessão por um período de um mês (30 dias), poderá especificá-lo na interface. A partir do momento em que o utilizador se desliga, o *cookie* é removido e a sessão é terminada. No Excerto de Código 4.2 apresenta-se o processo de criação e obtenção de um *cookie* de sessão GWT.

Excerto de Código 4.1 Utilização do objecto `SharedPreferences`.

```
1   public static final String PREFERECES_NAME = "Preferences_Name";
2   public static final String KEY_ID = "id";
3   private SharedPreferences mPreferences;
4
5   private SharedPreferences getSharedPreferences(){
6       return getSharedPreferences(PREFERECES_NAME, MODE_PRIVATE);
7   }
8
9   public String getPreference(String key) {
10      String value;
11      value = mPreferences.getString(key, "");
12      return value;
13  }
14
15  public void setPreference(String key, String value) {
16      SharedPreferences.Editor edit = mPreferences.edit();
17      edit.putString(key, value);
18      edit.commit();
19  }
```

4.2.2 Instalações

As instalações eléctricas são representadas na memória volátil através da classe `Instalacao` que inclui a designação da instalação, a potência contratada, o proprietário, a localização e os contadores associados à instalação. Na Tabela 4.4 apresenta-se a estrutura da classe `Instalacao` e na Figura 4.2 as suas relações.

Excerto de Código 4.2 Utilização de `Cookies` no GWT.

```
1   public static final String SESSION_NAME = "session";
2
3   public void createCookie(String cookieId){
4       final long DURATION = 1000 * 60 * 60 * 1; //1 hour
5       Date expires = new Date(System.currentTimeMillis() + DURATION);
6       Cookies.setCookie(SESSION_NAME, cookieId, expires);
7   }
8
9   public Cookie getCookie(){
10      return Cookies.getCookie(SESSION_NAME);
11  }
```

Tabela 4.4: Estrutura de dados da classe `Instalacao`.

Campo	Tipo
proprietário	`Utilizador`
nome	`String`
localização	`Localizacao`
contadores	<`Contador`>

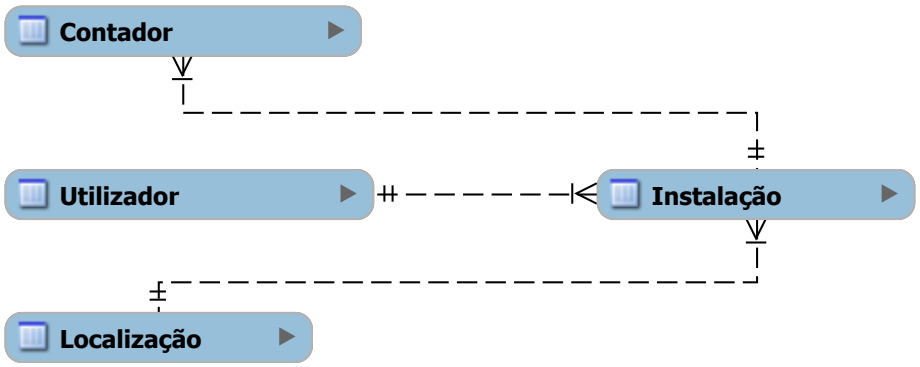

Figura 4.2: Relação da classe `Instalacao` com as restantes classes da aplicação.

As operações associadas a uma instalação são:

Criação: A criação de uma instalação está dividida em três fases. A introdução das definições gerais (nome e potência contratada), introdução da localização e a criação dos contadores que a constituem. O serviço verifica se o nome escolhido está disponível para aquele proprietário e responde em concordância. Ao nível dos dispositivos móveis a verificação do nome é feita localmente recorrendo à BD SQLite do equipamento.

Edição: A edição de uma instalação permite a alteração do nome e potência contratada e introduzir ou remover contadores. Não é possível editar a sua localização.

Eliminação: A eliminação de uma instalação implica a remoção da BD de toda a informação que lhe está associada (contadores, localização e filtros de pesquisa).

Recuperação: A recuperação de instalações permite sincronizar os dados da aplicação móvel com os dados do Datastore. Esta operação elimina todos os dados da instalação na BD e substitui-os pelos provenientes do Datastore.

4.2.2.1 Contadores

O contador é um conceito essencial para a aplicação, uma vez que todo o consumo ou produção está associado a um contador. Foram contemplados três tipos de contadores:

Residencial: contador sem sistema de recolha AMR ou memória para o armazenamento de curva de carga.

C&I: contador inteligente de importação de energia com AMR. A informação destes contadores é recolhida através da interacção com a aplicação ACE Vision.

Produção: contador de produção idêntico ao C&I que regista a energia exportada para a rede.

Um contador genérico é caracterizado pela marca, número de série, fluxo de energia (importada ou exportada) e número de tarifas associadas. Adicionalmente os contadores C&I incluem o número de telefone para transmissão por GSM e a frequência das leituras. A estrutura do contador de produção inclui, para além da estrutura de um contador C&I, o valor do projecto de micro ou miniprodução, o valor amortizado do projecto e a data de previsão da amortização. Nas Tabelas 4.5, 4.6 e 4.7 estão representadas as estruturas destas classes.

Tabela 4.5: Estrutura de um `Contador Residencial`.

Campo	Tipo
marca	`String`
número de série	`String`
fluxo de energia	`int`
tipo de contador	`int`
número de tarifas	`int`

As funções de gestão de contadores envolvem os seguintes processos:

Criação: A criação de um contador envolve a introdução dos dados representados nas Tabelas 4.5, 4.6 e 4.7. A criação de um contador só fica concluída

Tabela 4.6: Estrutura específica de um `Contador C&I`.

Campo	Tipo
frequência de leitura	`int`
número de telefone	`String`

Tabela 4.7: Estrutura específica de um `Contador de Producao`.

Campo	Tipo
custo total do projecto	`float`
valor amortizado	`float`
data prevista de amortização total	`Date`

após a introdução das respectivas tarifas (Secção 4.2.3.1). No caso dos contadores C&I, após a verificação da unicidade do número de série, é enviado um *email* ao administrador do sistema ACE Vision com a informação do contador para a criação do mesmo. A definição de um contador deste tipo acarreta a criação de um objecto de `Verificacao de Leituras` apresentado na Tabela 4.8.

Tabela 4.8: Estrutura de um objecto da classe `Verificacao de Leituras`.

Campo	Tipo
identificador do contador	`String`
data da última leitura	`Date`
número de semanas até à próxima leitura	`int`

Edição: Nos contadores residenciais apenas se podem editar as tarifas. Nos contadores C&I e produção pode-se alterar, além das tarifas, a frequência de leitura e o telefone associado ao contador. Os restantes parâmetros não são editáveis.

Eliminação: A eliminação de um contador envolve a perda de toda a informação associada.

Actualização: No caso dos contadores C&I e de produção é possível configurar alertas de consumo (Secção 4.2.4.2). Esta função lista os alarmes armazenados no Datastore. No caso específico dos contadores de produção, é possível actualizar o valor amortizado do projecto. Este cálculo é baseado no valor do fecho de facturação do contador e no preço unitário da tarifa associada. Primeiro calcula-se o total facturado desde o início da produção através da Equação 4.1, onde f_a representa o acumulado facturado actual, f_i a facturação do período i e n o número de períodos de facturação que decorreram desde o início da produção.

Equação 4.1: Valor facturado acumulado actual.

$$f_a = \sum_{i=1}^{n} f_i \tag{4.1}$$

De seguida, calcula-se o valor amortizado actual através da Equação 4.2, onde a_a é o valor amortizado actual, f_a é o valor facturado acumulado actual da Equação 4.1 e a_f representa a amortização final, *i.e.* corresponde ao custo do projecto.

Equação 4.2: Valor amortizado actual.

$$a_a = a_f - f_a \tag{4.2}$$

A previsão do tempo restante para a amortização do custo do projecto obtém-se através da Equação 4.3, onde Δt_a representa o período desde o início da produção até à actualidade, Δt_{af} o período restante para a amortização final, a_a o valor da amortização actual e a_f o valor da amortização final.

Equação 4.3: Previsão do tempo restante da amortização.

$$\frac{\Delta t_a}{\Delta t_{af}} = \frac{a_a}{(a_f - a_a)} \tag{4.3}$$

A resposta a este pedido é um objecto da classe `Actualizacao dos Contadores C&I` ilustrada na Tabela 4.9.

Tabela 4.9: Estrutura de uma `Actualizacao dos Contadores C&I`.

Campo	**Tipo**
valor amortizado	`float`
alarmes	<`Resultado Alarmes`>
localidade	`String`
data prevista de amortização total	`Date`

4.2.2.2 Localização

A localização de uma instalação inclui o país, a região, a localidade, o código postal e as respectivas coordenadas geodésicas (latitude e longitude) – ver Tabela 4.10.

O utilizador só pode criar localizações, *i.e.*, não pode alterar ou remover as localizações criadas.

Tabela 4.10: Estrutura de uma `Localizacao`.

Campo	Tipo
país	`String`
região	`String`
cidade	`String`
código postal	`int`
latitude	`float`
longitude	`float`

Dispositivo móvel: A obtenção da localização de uma instalação pode ser efectuada de forma automática ou de forma semi-automática. No primeiro caso, é usado o sensor de posicionamento do equipamento para recolher as coordenadas da sua localização actual (usando o `LocationManager` do Android) e recorre-se ao método `getFromLocation` da classe `Geocoder` para obter as restantes informações. No segundo caso, o utilizador introduz a morada e recorre-se à classe `Geocoder` para obter as coordenadas geodésicas, usando o método `getFromLocationName`. O Excerto de Código 4.3 descreve estes processos.

Excerto de Código 4.3 Utilização do `Geocoder` no Android.

```
1   private android.location.Location location;
2   private Geocoder geocoder;
3   private void getProvider() {
4       try {
5           Criteria criteria = new Criteria();
6           provider = locationManager.getBestProvider(criteria, true);
7           if (provider != null && action.equals(KEY_ACTION_CLICKED)) {
8               location = locationManager.getLastKnownLocation(provider);
9           }
10          locationManager.requestLocationUpdates(provider, 60000, 10,
11                  locationListener);
12      } catch (Exception e) {
13      }
14  }
15  private void getFullAddress() {
16      getProvider();
17      if (location != null) {
18          List<Address> addresses = null;
19          try {
20              addresses = geocoder.getFromLocation(location.getLatitude(),
21                      location.getLongitude(), 6);
22          } catch (IOException e) {
23          }
24      }
25  }
26  private void getGPSCoordinates() {
27      getProvider();
28      StringBuilder address = new StringBuilder(); // Preencher com o endereço
29      List<Address> addresses = null;
30      try {
31          addresses = geocoder.getFromLocationName(address.toString(), 6);
32      } catch {
33      }
34  }
```

Aplicação *Web*: A especificação da localização de uma instalação na aplicação *Web* envolve o preenchimento de um formulário com os dados da morada da

instalação. As coordenadas geodésicas da localização são obtidas através da classe `GeocoderRequest`. No final deste processo é exibido um mapa com um marcador no local da instalação – ver Figura 4.3. O Excerto de Código 4.4 apresenta a implementação deste processo.

Excerto de Código 4.4 Utilização do `Geocoder` no GWT.

```
1   private Geocoder geocoder;
2   private GoogleMap map;
3       public void codeAddress(String address) {
4       GeocoderRequest request = GeocoderRequest.create();
5       request.setAddress(address);
6       geocoder = Geocoder.create();
7       geocoder.geocode(request, new Callback() {
8       public void handle(JsArray<GeocoderResult> results, GeocoderStatus status) {
9               if (status == GeocoderStatus.OK) {
10              LatLng myLatLng = LatLng.create(38.7, -9.183333);
11
12              MapOptions mapOptions = MapOptions.create();
13              mapOptions.setZoom(8.0);
14              mapOptions.setCenter(myLatLng);
15              mapOptions.setMapTypeId(MapTypeId.HYBRID);
16
17              GeocoderResult location = results.get(0);
18              map = GoogleMap.create(panelMaps.getElement(), mapOptions);
19              map.setCenter(location.getGeometry().getLocation());
20
21              MarkerOptions markerOpts = MarkerOptions.create();
22              markerOpts.setMap(map);
23              markerOpts.setPosition(location.getGeometry().getLocation());
24              Marker.create(markerOpts);
25              }
26          }
27      });
28  }
```

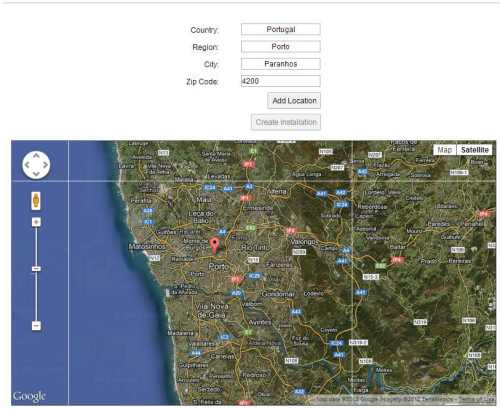

Figura 4.3: Criação de uma localização na aplicação *Web*.

4.2.3 Leituras

Uma leitura representa o valor de energia consumida ou produzida num determinado período de tempo. As leituras no caso dos contadores residenciais são

introduzidas manualmente pelo utilizador. Se utilizar a aplicação móvel para submeter as leituras e não existir ligação à Internet estas são armazenadas localmente. Nos contadores C&I as leituras são obtidas directamente do ACE Vision. A `Leitura` apresenta a estrutura da Tabela 4.11. As leituras estão associadas a um contador. Na Figura 4.4 revela-se a relação entre as leituras, as tarifas e o contador.

As leituras originárias do ACE Vision possuem uma outra estrutura designada perfil de carga ou `LoadProfile` – ver Tabela 4.12. Esta estrutura contém o intervalo das datas da recolha, o contador, a unidade das leituras e a lista de pares valor-data. Estes pares de `Valores LP` possuem a estrutura da Tabela 4.13. Na Figura 4.5 descrevem-se as relações entre o contador, a tarifa, o perfil de carga e os pares leitura-data.

Tabela 4.11: Estrutura de uma `Leitura`.

Campo	**Tipo**
data da leitura	`Date`
identificador do `Time Zone`	`String`
identificador da tarifa	`int`
unidade	`int`
identificador do contador	`String`
valor de energia	`float`

Tabela 4.12: Estrutura do `LoadProfile`.

Campo	**Tipo**
limite das datas recolhidas	<Date>
filtro LP	`Filtro LP`
unidade	`Unidade ACE Vision`
contador	`Contador`
valores lp	<`Valores LP`>

Tabela 4.13: Estrutura de um objecto `Valores LP`.

Campo	**Tipo**
valor	`float`
data	`Date`

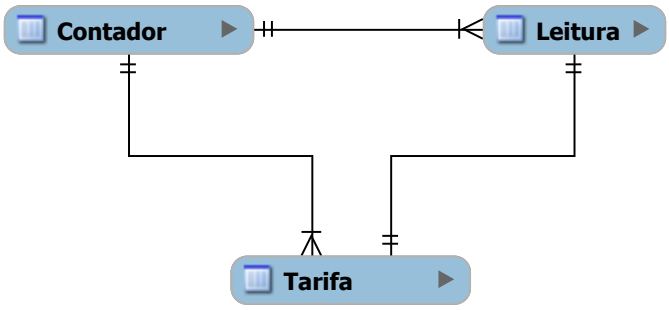

Figura 4.4: Relações da classe `Leitura`.

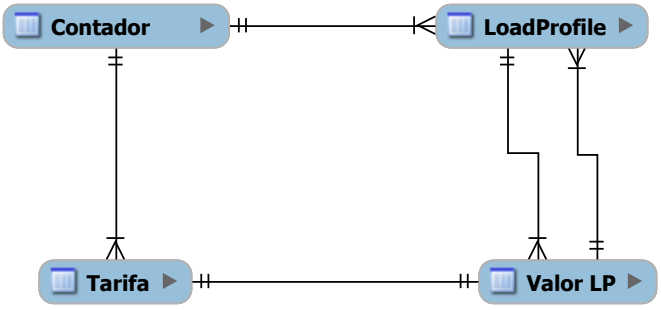

Figura 4.5: Relações da classe `LoadProfile`.

4.2.3.1 Tarifas

As tarifas permitem determinar o preço da energia consumida ou produzida (€/kWh). A tarifa apresenta a estrutura da Tabela 4.14.

Tabela 4.14: Estrutura de um objecto `Tarifa`.

Campo	Tipo
preço unitário	`float`
unidade	`int`
identificador do contador	`String`
identificador da tarifa	`int`
em uso	`boolean`
data de validação	`Date`

Em relação às tarifas, o utilizador pode efectuar as seguintes operações:

Criação: A criação de uma tarifa é efectuada no âmbito da criação e edição de um contador e são criadas tantas tarifas quantas as que se aplicam ao contador. Este processo envolve o preenchimento dos campos unidade, preço

unitário e identificador de tarifa; os restantes campos são preenchidos automaticamente. Por omissão o campo `em uso` é colocado no estado verdadeiro e a data de validação armazenada corresponde à data de criação da tarifa.

Edição: A edição de uma tarifa efectua-se quando são alteradas as tarifas de um contador. As tarifas existentes passam a ficar em desuso (campo `em uso` passa a falso), mas permanecem disponíveis para assegurar a realização de cálculos relativos a leituras passadas.

4.2.3.2 Submissão de Leituras

No caso específico dos contadores residenciais, é necessário proceder à recolha e submissão manual das leituras. Tanto na aplicação *Web* como no dispositivo móvel existem formulários próprios para esta operação. O utilizador especifica a unidade da leitura, a tarifa e o valor do contador. Na aplicação móvel a data e hora da leitura são recolhidas automaticamente; na aplicação *Web* é necessário preencher manualmente o campo. O armazenamento da leitura de um contador no Datastore origina a criação de um objecto da classe `Estatistica` representado na Tabela 4.15. Este objecto é actualizado sempre que é introduzida uma nova leitura. No dispositivo móvel existe um menu de introdução rápida de leituras que detecta automaticamente a localização do utilizador e exibe apenas os contadores das instalações que estão na sua proximidade. Caso não sejam detectadas quaisquer instalações, são mostrados todos os contadores do utilizador.

Tabela 4.15: Estrutura de um objecto `Estatistica`.

Campo	Tipo
identificador de tarifa	`int`
média	`float`
média monetária	`float`
identificador da tarifa	`int`
número da introdução	`int`
última introdução	`Date`
unidade	`int`
frequência	`long`
contador	`Contador`
total	`float`
total monetário	`float`

4.2.3.3 Calendarização de Leituras

No domínio móvel é possível agendar leituras e configurar as respectivas notificações. Este processo é efectuado recorrendo às classes `AlarmManager`, `BroadCastReceiver` e `Notification` do Android que permitem configurar a data

da notificação, a actividade que processa a notificação e as características da notificação (o estado do indicador de presença, o toque, *etc.*). Recorre-se ainda à classe `PowerManager.WakeLock` para acordar o dispositivo no momento da notificação. No Excerto de Código 4.5 exemplifica-se este processo.

As diversas funções disponibilizadas para a calendarização de leituras são:

Criação: A criação de múltiplas notificações consiste na selecção do contador e na especificação da data de notificação. São também criadas notificações que se repetem periodicamente (minutos, horas ou dias).

Diferimento: Como a introdução de leituras está directamente relacionada com a localização do utilizador, é possível adiar a notificação por um determinado período de tempo, configurável na aplicação (por omissão 15 min). É também possível activar, de forma automática, a detecção da posição do utilizador, garantindo que a notificação só ocorre quando se encontra na proximidade da instalação.

Cancelamento: O utilizador pode cancelar a notificação. Se a notificação for periódica, o cancelamento apenas afecta a próxima notificação agendada. O cancelamento de uma notificação singular (uma única ocorrência) corresponde à sua eliminação.

Eliminação : Esta opção permite eliminar as notificações periódicas.

Execução: Esta funcionalidade permite executar imediatamente uma notificação agendada. No caso de uma notificação singular, a tarefa é eliminada após a introdução da leitura. No caso de uma notificação periódica, o sistema agenda a próxima notificação.

Excerto de Código 4.5 Criação de notificações no Android.

```
1   public class SetAlarm{
2       private int processId,frequencyValue,frequencyUnit;
3       private Date taskDate;
4       public SetAlarm(Context context) {
5           Calendar calendar = Calendar.getInstance();
6           calendar.setTime(taskDate);
7           Intent intent = new Intent(context, AlarmHandler.class);
8           intent.putExtra(AlarmHandler.ALARM_PROCESS_ID, processId);
9           PendingIntent sender = PendingIntent.getBroadcast(context, processId,
10                  intent, PendingIntent.FLAG_UPDATE_CURRENT);
11          AlarmManager am =
12              (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
13          am.setRepeating(
14              AlarmManager.RTC_WAKEUP,
15              calendar.getTimeInMillis(),
16              setNewPeriod(frequencyUnit, frequencyValue), sender);
17      }
18      public void cancelAlarm(Context context) {
19          Intent intent = new Intent(context, AlarmHandler.class);
20          intent.putExtra(AlarmHandler.ALARM_PROCESS_ID, processId);
21          PendingIntent sender = PendingIntent.getBroadcast(
22              context, processId, intent, PendingIntent.FLAG_UPDATE_CURRENT);
23          AlarmManager am =
24              (AlarmManager) context.getSystemService(Context.ALARM_SERVICE);
25          am.cancel(sender);
26      }
27  }
28  public class AlarmHandler extends BroadcastReceiver {
29      public static final String ALARM_PROCESS_ID = "alarmProcessID";
30      @Override
31      public void onReceive(Context context, Intent intent) {
32          try {
33          Bundle extras = intent.getExtras();
34          WakeScreen.acquire(context);
35          NotificationManager notificationManager = (NotificationManager)
36              context.getSystemService(Context.NOTIFICATION_SERVICE);
37          if (notification == null) {
38              notification = new Notification(
39                  R.drawable.ic_launcher,"Notification Text",
40                  System.currentTimeMillis());
41          }
42          notification.flags |= Notification.FLAG_AUTO_CANCEL;
43          notification.flags |= Notification.FLAG_INSISTENT;
44          notification.ledARGB = 0xff00ff00;
45          notification.defaults |= Notification.DEFAULT_SOUND;
46
47          PendingIntent activity = PendingIntent.getActivity(
48              context,
49              extras.getInt(ALARM_PROCESS_ID),
50              intent, PendingIntent.FLAG_UPDATE_CURRENT);
51          } catch (Exception e) {}
52      }
53  }
54  public abstract class WakeScreen {
55      private static PowerManager.WakeLock wakeScreen;
56      public static void acquire(Context context) {
57          if (wakeScreen != null) wakeScreen.release();
58          PowerManager pm = (PowerManager) context.getSystemService(Context.POWER_SERVICE);
59          wakeScreen = pm.newWakeLock(
60              PowerManager.FULL_WAKE_LOCK |
61              PowerManager.ACQUIRE_CAUSES_WAKEUP |
62              PowerManager.ON_AFTER_RELEASE,
63              String.valueOf(Calendar.getInstance().getTimeInMillis()));
64                  wakeScreen.setReferenceCounted(false);
65                  wakeScreen.acquire();
66      }
67      public static void release() {
68          if (wakeScreen != null){
69              wakeScreen.release();
70          } else { wakeScreen = null; }
71      }
72  }
```

4.2.4 Análise de Consumo e Produção

O armazenamento das leituras permite a apresentação gráfica da evolução do consumo ou produção energética da instalação.

No domínio residencial, quando o utilizador consulta as leituras de um contador obtém uma janela de definição do filtro de pesquisa baseado na escolha de um período pré-definido ou na definição de um período específico (semana corrente, semana passada, mês corrente ou mês passado). A aplicação *Web* ou móvel organiza a informação recebida e mostra a frequência, os valores médio e total da energia consumida ou produzida e o preço associado do período seleccionado. Constrói também um gráfico com a evolução do consumo ou produção, a distribuição do consumo por tarifa e a distribuição dos gastos financeiros por tarifa no período especificado – ver Figura 4.6. Por fim, é apresenta a lista das leituras do período.

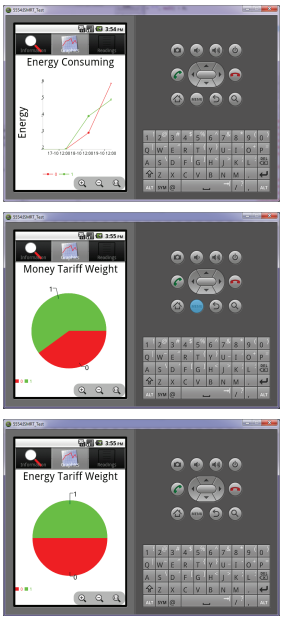

Figura 4.6: Gráficos da aplicação móvel

No domínio industrial, a pesquisa das leituras é igualmente baseada em filtros. Nestes contadores, para além da informação da curva de carga, é também possível recolher os dados do fecho de facturação e da potência máxima. Os dados de fecho de facturação são os valores totais das leituras do contador na data de emissão da factura. Os valores de potência máxima correspondem às 5 leituras mais elevadas no período de facturação. Estes objectos estão caracterizados nas Tabelas 4.16, 4.17 e 4.18.

Tabela 4.16: Estrutura de um objecto `Fim de Facturacao`.

Campo	**Tipo**
valor por tarifa	`float`
total	`float`
unidade	`Unidade ACE Vision`
contador	`Contador`
data	`Date`
filtro	`Filtro Fim de Facturacao`

Tabela 4.17: Estrutura de um objecto `Potencia Maxima`.

Campo	**Tipo**
valor por tarifa	`float`
total	`float`
unidade	`Unidade ACE Vision`
contador	`Contador`
filtro	`Filtro Potencia Maxima`
valores	<`Potencia`>

Tabela 4.18: Constituição de um objecto `Potencia`.

Campo	**Tipo**
valor	`float`
data	`Date`

Cada objecto do ACE Vision tem um filtro associado definido através da classe `Filtro`. Esta classe permite formatar os dados e escolher o período pretendido. Um filtro caracteriza-se por um nome, tipo de dados que devolve, unidade dos dados recebidos e tipo de data (mês actual, mês passado, total ou personalizado). Existem filtros de potência (`Filtro`), fim de facturação (`Filtro Fim de Facturacao`) e perfil de carga (`Filtro Curva de Carga`), que possuem configurações específicas. As estruturas destes filtros são apresentadas nas Tabelas 4.19, 4.20 e 4.21.

Tabela 4.19: Estrutura de um objecto `Filtro`.

Campo	**Tipo**
nome	`String`
unidade	`Unidade ACE Vision`
limite de datas	<`Date`>
modo de data	`int`
tipo de filtro	`int`

Assim, é possível efectuar três tipos de análises distintas nos contadores inteligentes:

- **Potência máxima**: É apresentada a lista dos cinco valores de potência

Tabela 4.20: Estrutura específica de um objecto `Filtro Fim de Facturacao`.

Campo	Tipo
tipo de soma	`int`

Tabela 4.21: Estrutura específica de um objecto `Filtro Curva de Carga`.

Campo	Tipo
modo de integração	`int`
tipo de curva de carga	`int`

máxima registadas no período, o gráfico com a evolução temporal, a média dos valores obtidos e o valor máximo registado.

- **Fim de facturação**: São apresentadas as médias do consumo ou de produção total por tarifa, um gráfico da respectiva evolução temporal e a lista das leituras.

- **Curva de carga**: A análise dos dados é igual aos contadores residenciais. No caso dos contadores de produção, é possível traçar a evolução das vendas.

4.2.4.1 Tectos de Consumo

O utilizador residencial para monitorar o consumo e consequentemente os gastos energéticos pode configurar tectos de consumo. Um tecto de consumo corresponde ao valor máximo que o utilizador pretende gastar num determinado período de tempo. É representado pela classe `Tecto de Consumo` definida na Tabela 4.22.

Tabela 4.22: Estrutura de um objecto `Tecto de Consumo`.

Campo	Tipo
identificador	`String`
contador	`Contador`
limite de datas	<`Date`>
valores actuais	<`Valores de Tectos de Consumo`>
valores máximos	<`Valores de Tectos de Consumo`>

Relativamente aos tectos de consumo o utilizador pode:

Criação: Para criar um tecto de consumo deve introduzir uma data inicial e final, os valores actuais do consumo de energia e os máximos que pretende gastar nesse período. Com estes dados o sistema calcula, com base na tarifa actual, o valor máximo de energia a consumir. São associados dois objectos por cada tarifa da classe `Valor de Tecto de Consumo` representado na Tabela 4.23. Um dos objectos contém os consumos máximos estabelecidos pelo utilizador, o outro objecto guarda o total que o utilizador consumiu e o

respectivo valor monetário, actualizado sempre que são introduzidas novas leituras.

Tabela 4.23: Estrutura de um objecto `Valor Tecto de Consumo`.

Campo	Tipo
identificador de tarifa	`int`
valor	`float`
preço	`float`

Eliminação: Elimina todos os registos associados a um tecto de consumo.

Análise: A análise dos tectos de consumo é idêntica à análise das leituras. Inclui adicionalmente a previsão do valor do consumo no final do prazo do tecto. Esta previsão assume uma proporcionalidade directa entre o tempo decorrido e o valor consumido. Quando existem tectos de consumo definidos no sistema, e se submete uma nova leitura o nó central verifica se algum desses tectos alcançou 75 % do valor máximo estabelecido. Em caso afirmativo, o cliente é alertado acerca da percentagem alcançada.

Actualização: Como podem ser criados tectos de consumo, tanto na aplicação *Web* como na aplicação móvel, é possível sincronizar a informação dos tectos de consumo, com base nos dados armazenados no Datastore.

4.2.4.2 Alarmes de Consumo e Produção

Como a frequência das leituras dos contadores inteligentes é conhecida, é possível definir alarmes. Para criar um alarme deste tipo é necessário introduzir o tecto de energia (tecto de produção é um valor mínimo e o tecto de consumo é um valor máximo) e o período de tempo mínimo que foi ultrapassado o tecto. Nos contadores de produção o tempo de validação do alarme está entre as 09:00 e as 17:00, para evitar o disparo do alarme. Na Tabela 4.24 representa-se a estrutura de um alarme e na Tabela 4.25 a resposta a uma actualização do contador.

Tabela 4.24: Estrutura de um objecto `Alarme`.

Campo	Tipo
nome	`String`
identificador de contador	`String`
valor em kWh	`float`
tempo em minutos	`int`

4.2.4.3 Comparação de Consumo e Produção

O utilizador pode comparar o seu consumo e gastos médios com os dos consumidores de uma dada região com potência contratada idêntica. Podem ser definidos

Tabela 4.25: Estrutura de um objecto `Resultado Alarmes`.

Campo	Tipo
identificador	`String`
identificador de contador	`String`
valor em kWh	`float`
limite de datas	<`Date`>

para cada instalação filtros de pesquisa específico. Na Tabela 4.26 apresenta-se a estrutura dos filtros de regiões. As regiões são definidas através da classe `Localizacao`, permitindo ao utilizador efectuar comparações ao nível do país, região, localidade ou código postal.

Tabela 4.26: Estrutura de um objecto `Filtro de regiao`.

Campo	Tipo
nome	`String`
instalação	`Instalacao`
localizações	<`Localizacao`>

Tal como no caso da introdução de localizações, também nestes filtros são usados métodos de resolução automática das coordenadas geográficas dos locais introduzidos. Os resultados são apresentados num mapa com marcadores posicionados sobre as localizações assinaladas. Cada marcador tem uma cor que ilustra o resultado da comparação: vermelho, preto e verde. São comparadas as médias dos consumos energéticos e gastos associados de cada região com o contador do utilizador. Se ambas as médias forem superiores o marcador apresenta uma cor vermelha; se apenas uma das médias for superior adopta-se a cor preta; por fim se ambas as médias forem inferiores, a cor do marcador é verde. No caso dos contadores de produção quando as médias são superiores é sinónimo de melhores condições, as cores invertem-se. Na Figura 4.7 ilustra-se o aspecto de um filtro de região.

O utilizador pode apagar e actualizar os filtros de região.

4.2.5 Previsão de Consumo e Produção

O consumo e produção de energia tende a ser cíclico. Na Figura 4.8 são apresentados dois exemplos da evolução da produção de dois contadores distintos. Este comportamento permite, a partir de um conjunto de leituras adequadas, determinar a evolução futura, através de uma regressão polinomial, do consumo ou produção.

Estabeleceu-se que os modelos gerados pelo servidor têm uma periodicidade diária e que o utilizador pode obter uma projecção futura que se prolonga até 150 % do período da amostra usada para gerar o modelo. Para se efectuar uma predição,

Figura 4.7: Exemplo de uma comparação de consumo.

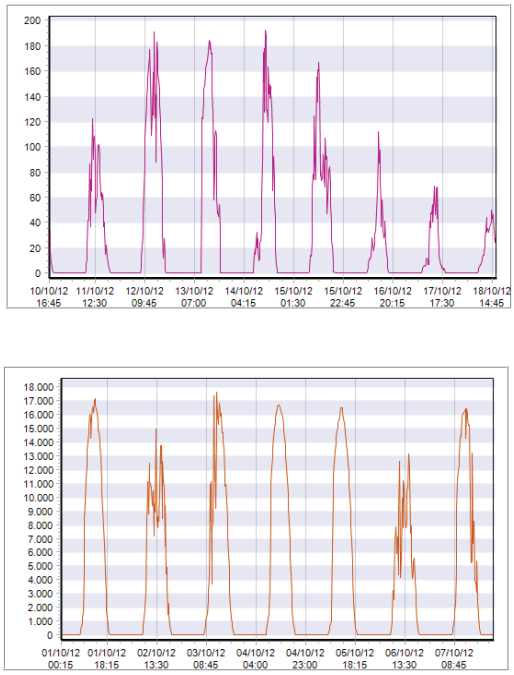

Figura 4.8: Evolução da produção de energia de dois contadores.

é necessário possuir-se um histórico de pelo menos 30 d. A partir deste histórico o utilizador pode prever até 15 d de consumo ou produção.

A regressão linear permite, dada uma equação polinomial de grau n representada pela Equação 4.4 e uma amostra com p pontos, determinar uma equação de grau $p-1$ que representa o comportamento da amostra.

Equação 4.4: Função polinomial de grau n.

$$y(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \cdots a_n x^n \tag{4.4}$$

Para determinar essa função polinomial é necessário resolver a Equação 4.5.

Equação 4.5: Matriz genérica de uma regressão polinomial.

$$\begin{pmatrix} p & \sum x_i & \sum x_i^2 & \sum x_i^3 & \dots & \sum x_i^n \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \dots & \sum x_i^{n+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \dots & \sum x_i^{n+2} \\ \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \sum x_i^6 & \dots & \sum x_i^{n+3} \\ \dots & & & & & \\ \sum x_i^n & \sum x_i^{n+1} & \sum x_i^{n+2} & \sum x_i^{n+3} & \dots & \sum x_i^{n+n} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \dots \\ a_n \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \\ \sum x_i^2 y_i \\ \sum x_i^3 y_i \\ \dots \\ \sum x_i^n y_i \end{pmatrix} \tag{4.5}$$

Em primeiro lugar foi criada uma matriz com $n + 2$ colunas – ver Equação 4.6.

Equação 4.6: Matriz de determinação dos coeficientes.

$$\begin{pmatrix} p & \sum x_i & \sum x_i^2 & \sum x_i^3 & \dots & \sum x_i^n & \sum y_i \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \dots & \sum x_i^{n+1} & \sum x_i y_i \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \dots & \sum x_i^{n+2} & \sum x_i^2 y_i \\ \sum x_i^3 & \sum x_i^4 & \sum x_i^5 & \sum x_i^6 & \dots & \sum x_i^{n+3} & \sum x_i^3 y_i \\ \dots & & & & & & \\ \sum x_i^n & \sum x_i^{n+1} & \sum x_i^{n+2} & \sum x_i^{n+3} & \dots & \sum x_i^{n+n} & \sum x_i^n y_i \end{pmatrix} \tag{4.6}$$

Em segundo lugar, através da eliminação de Gauss-Jordan, obtém-se a matriz da Equação 4.7.

Equação 4.7: Resultado da eliminação de Gauss-Jordan.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & c0 \\ 0 & 1 & 0 & 0 & \dots & 0 & c1 \\ 0 & 0 & 1 & 0 & \dots & 0 & c2 \\ 0 & 0 & 0 & 1 & \dots & 0 & c3 \\ \dots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & 1 & cn \end{pmatrix} \tag{4.7}$$

Por último, os coeficientes da equação polinomial da regressão linear correspondem aos valores da última coluna da matriz da Equação 4.7 – ver a Equação 4.8 [31].

Equação 4.8: Função Polinomial resultante.

$$y(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \ldots + c_n x^n \tag{4.8}$$

Este cálculo só está disponível na aplicação móvel e para contadores inteligentes. Por essa razão, os modelos são armazenados na BD local que recorre ao serviço apenas para a obtenção dos coeficientes. Sempre que o utilizador pretende visualizar os resultados do modelo criado, a aplicação calcula, com base na equação obtida, os respectivos valores. Além de criar é possível também eliminar os modelos armazenados na BD local.

4.2.6 App Engine

O Google App Engine aloja o serviço *Web* que disponibiliza as operações descritas, armazena no Datastore as instancias das estruturas descritas e implementa um *cron job* para verificar diariamente os alarmes e actualizar a data de previsão de amortização dos contadores de produção.

4.2.6.1 Serviço *Web*

O serviço *Web* expõe o conjunto de operações apresentadas na Tabela 4.27. Para as invocar é necessário enviar as credenciais do utilizador. A Tabela 4.28 lista os métodos criados para a análise das leituras.

4.2.6.2 Datastore

O Datastore armazena entidades. Cada entidade possui uma chave e pode estar relacionada com outra entidade. Entidades com o mesmo nome podem ter propriedades diferentes, ou propriedades iguais com diferentes tipos de dados. Na solução desenvolvida foram criadas 15 tipos de entidades que se apresentam na Tabela 4.29.

4.2.6.3 Agendamento de Tarefas

Um *cron job* é uma tarefa programada para executar periodicamente. Na solução proposta foi desenvolvido um *cron job* para a actualização dos dados dos contadores inteligentes. Esta actualização detecta a existência de alarmes relativos a novas leituras e no caso dos contadores de produção, corrige o valor amortizado e a data prevista para a amortização total. No Excerto de Código 4.6 ilustra-se a configuração de um *cron job* na plataforma App Engine. A configuração é feita

Tabela 4.27: Operações do serviço *Web*.

Secção	**Operação**	***In***	***Out***
Utilizador	Criar `Utilizador`	`Utilizador`	`String`
	Editar *Password*	*username*	`String`
		Password antiga	
		Nova *Password*	
	Editar Perfil	`Utilizador`	`String`
	Apagar `Utilizador`	`Utilizador`	`String`
	Actualizar `Utilizador`	Credenciais do Utilizador	`Utilizador`
Instalação	Criar Instalação	`Instalacao`	`String`
	Editar Instalação	Novo nome da `Instalacao`	`String`
		Nome antigo da Instalação	
		Potência da Instalação	
	Apagar Instalação	`Instalacao`	`String`
	Actualizar Instalação	Credenciais do Utilizador	<`Instalacao`>
Contador	Criar Contador Residencial	`Contador Residencial`	`String`
		Nome da Instalação	
	Criar Contador C&I	`Contador C&I`	`String`
		Nome da Instalação	
	Criar Contador Produção	`Contador Produção`	`String`
		Nome da Instalação	
	Editar Contador Residencial	`Contador Residencial`	`String`
		Nome da Instalação	
	Editar Contador C&I	`Contador C&I`	`String`
		Nome da Instalação	
	Editar Contador Produção	`Contador Produção`	`String`
		Nome da Instalação	
	Apagar Contador	`Contador`	`String`
		Nome da Instalação	
	Actualizar Contador C&I	Identificador de Contador	`Actualizacao dos Contadores C&I`
Leituras	Criar `Leitura`	`Leitura`	`String`
	Obter leituras	identificador do contador	<`Leitura`>
		Data inicial	
		Data final	
	Criar Filtro	`Filtro`	`String`
	Criar Filtro LP	`Filtro LP`	`String`
	Criar Filtro EOB	`Filtro EOB`	`String`
	Apagar Filtro	`Filtro`	`String`
	Apagar Filtros	Credenciais do Utilizador	`String`
	Actualizar Filtros	Credenciais do Utilizador	<`Filtro`>
	Obter leituras LP	`Filtro`	`Curva de Carga`
	Obter leituras EOB	`Filtro`	<`Fim de Facturacao`>
	Obter potência máxima	`Filtro`	`Potencia maxima`

Tabela 4.28: Operações de análise do serviço *Web*.

Secção	**Operação**	***In***	***Out***
Tarifas	Criar ou Actualizar Tarifa	<Tarifa>	String
	Actualizar Tarifas	Identificador de Contador	<Tarifa>
Tectos de Consumo	Criar Tecto	Tecto de Consumo	String
	Apagar Tecto	Tecto	String
	Actualizar Tectos	Identificador do contador	<Tecto de Consumo>
Alarmes	Criar Alarme	Alarme	String
	Apagar Alarme	Alarme	String
	Apagar Alarmes	Credenciais do Utilizador	String
	Enviar Alarmes	Credenciais do Utilizador	<Alarme>
Filtro de mapa	Criar Filtro de Região	Tecto de Consumo	String
	Apagar Filtro de Região	Tecto de Consumo	String
	Apagar Filtros de Região	Credenciais do Utilizador	String
	Actualizar Filtros de Região	Nome da Instalacao	<Filtros de Regiao>
	Obter Resultados Residenciais	Filtro de Regiao Contador	Estatistica residencial
	Obter Resultados C&I	Filtro de Regiao Contador	Estatistica C&I
	Lista de Países	País	<String>
	Lista de Regiões	Região	<String>
	Lista de Cidades	Cidade	<String>
	Lista de Cód. Postal	Cód. Postal	<int>
Predição	Gerar Modelo	Contador <Date>	<Regressao>

Tabela 4.29: Lista de entidades do serviço *Web*.

Entidade	**Designação**
Utilizador	Utilizadores
Instalação	Instalações
Localização	Localizações
Contador	Armazenamento de todos os tipos de contadores
Leitura	Leituras residenciais
Tarifa	Tarifas de todos os contadores
Estatística	Estatísticas residenciais
Tecto de Consumo	Informações gerias dos tectos de consumo
Valor Tecto de Consumo	Valores actuais e máximos de um tecto de consumo
Calendarização	Datas em que são lidos os contadores inteligentes do ACE Vision
Filtro	Todos os tipos de filtros de pesquisa ACE Vision
Alarme	Informações gerais dos alarmes
Resultado Alarme	Resultados dos alarmes
Filtro de Mapa	Filtros de mapa
Localização Filtros de Mapa	Localizações dos filtros

através de um ficheiro XML designado `cron.xml` que deve ser colocado na pasta `WEB-INF` do projecto.

Excerto de Código 4.6 Agendamento de uma tarefa no App Engine.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <cronentries>
3      <cron>
4          <url>/update-statistics</url>
5          <description>Update CI and Production Meter statistics and so on.</description>
6          <schedule>every day 00:05</schedule>
7          <target>update</target>
8      </cron>
9  </cronentries>
```

Para não afectar o desempenho da máquina que serve a solução, foi configurada uma máquina de *back-end* para executar o *cron job*. No Excerto de Código 4.7 ilustra-se a configuração de uma máquina de *back-end* que, tal como no caso do *cron job*, consiste num ficheiro XML, designado `backends.xml` que se coloca na pasta `WEB-INF`.

Excerto de Código 4.7 Configuração de uma tarefa no App Engine.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <backends>
3      <backend name="update">
4          <class>B1</class>
5          <options>
6              <dynamic>true</dynamic>
7              <public>false</public>
8          </options>
9      </backend>
10 </backends>
```

4.3 Teste, Depuração e Resultados

Para verificar o correcto funcionamento das funcionalidades desenvolvidas procedeu-se a um conjunto de testes. Estes testes têm como principal objectivo verificar o correcto processamento de dados e das interfaces desenvolvidas. O conjunto de testes foi organizado em gestão de utilizadores, instalações e contadores, submissão e agendamento de leituras, análise de consumo e produção, tectos de consumo, alarmes, comparação e modelos de previsão. O ambiente de simulação foi constituído pelo servidor de produção App Engine, a aplicação *Web* instalada na mesma plataforma e um telefone htc Desire com SO Android 2.2.2.

4.3.1 Gestão de Utilizadores

Foram criados dois clientes distintos, um na aplicação *Web* e outro através da aplicação móvel. Os valores introduzidos estão representados na Tabela 4.30.

Tabela 4.30: Informação dos utilizadores criados.

Identificador Único	**Nome**	**Endereço de *email***	***Password***
Tiago	Tiago	teitiago@gmail.com	12345678
Alexandre	Alexandre	1070314@gmail.com	87654321

O primeiro utilizador (Tiago) foi criado na aplicação Android. A sequência dos menus de utilizador está numerada e é apresentada na Figura 4.9. Primeiro procedeu-se à criação do utilizador, Figura 4.9 - 1. Durante a fase de introdução foram deliberadamente provocados erros nos dados introduzidos no formulário, Figura 4.9 - 2. Depois de recebida a resposta positiva do servidor, saiu-se da aplicação para forçar a autenticação, Figura 4.9 - 5. Activou-se a opção de autenticação automática e repetiu-se o processo de autenticação, tendo sido armazenada localmente a sessão do utilizador.

O segundo utilizador (Alexandre) foi inserido na aplicação *Web*, seguindo a ordem especificada na Figura 4.10. Na Figura 4.10 - 1 criou-se o utilizador e na Figura 4.10 - 2 submeteram-se as credenciais de autenticação. As entidades correspondentes foram armazenadas no Datastore – ver Figura 4.11.

Através da Figura 4.9 - 7 e da Figura 4.9 - 9 e da 4.10 - 3 e 4.10 - 4, que servem para mudar a palavra-passe e editar o perfil respectivamente, foram alterados os utilizadores conforme representado na Tabela 4.31. Depois de confirmadas as operações de alteração dos utilizadores foram confirmados os endereços de correio electrónico, Figura 4.12 e o resultado obtido no Datastore foi o que se apresenta na Figura 4.13.

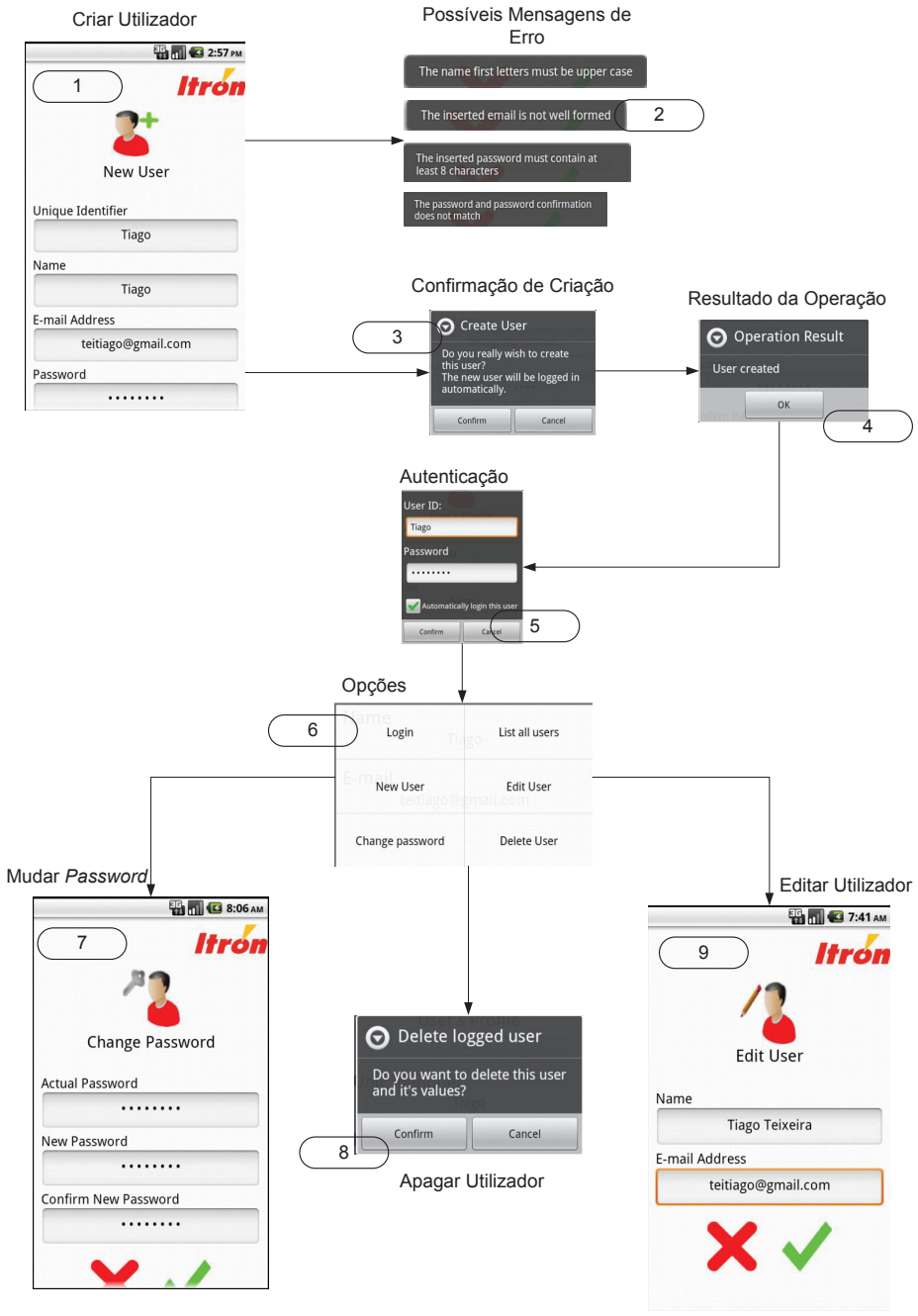

Figura 4.9: Sequência de opções do utilizador na aplicação móvel.

Tabela 4.31: Informação dos utilizadores editados.

Identificador Único	**Nome**	**Endereço de *email***	***Password***
Tiago	Tiago Teixeira	teitiago@gmail.com	11111111
Alexandre	Alexandre Alves	1070314@gmail.com	00000000

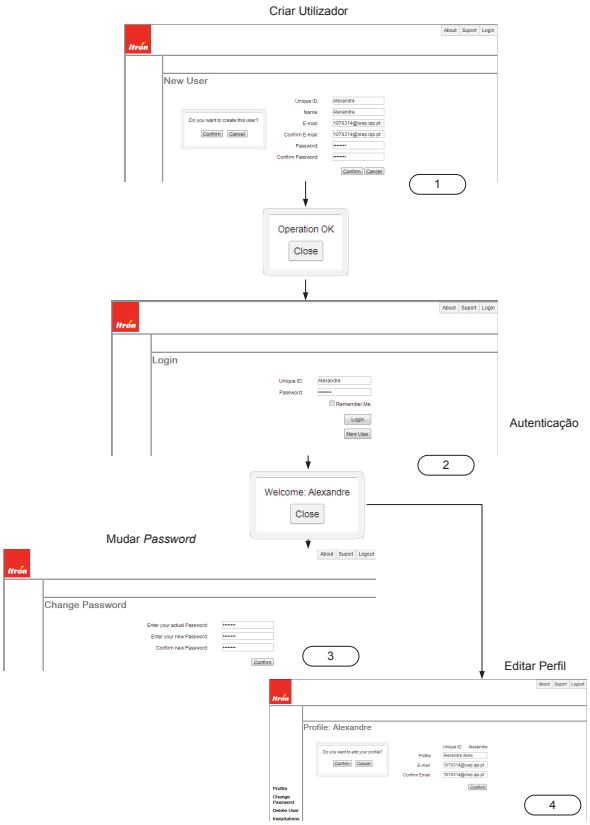

Figura 4.10: Sequência de opções do utilizador na aplicação *Web*.

User Entities

‹ Prev 20 **1-2** Next 20 ›

ID/Name	email	lastActivity	name	password
name=Alexandre	1070314@isep.ipp.pt	not validated 2012_10_27_07:57:03	Alexandre	87654321
name=Tiago	teitiago@gmail.com	not validated 2012_10_27_07:37:02	Tiago	12345678

Delete Flush Memcache ‹ Prev 20 **1-2** Next 20 ›

Figura 4.11: Resultado do Datastore após criação dos utilizadores.

teitiago@gmail.com através de 2uix4h7xygsz66weerlq.apphosting.bounces.google.com
para mim

Hello Tiago.
Please confirm your email address by clicking on the following link:
http://servicehelloworldexample.appspot.com/validate?id=Tiago&key=2012_10_27_07:37:02

If this email address was not addressed to you, please ignore it.
Many Thanks.

Figura 4.12: Exemplo de um pedido de confirmação de *email*.

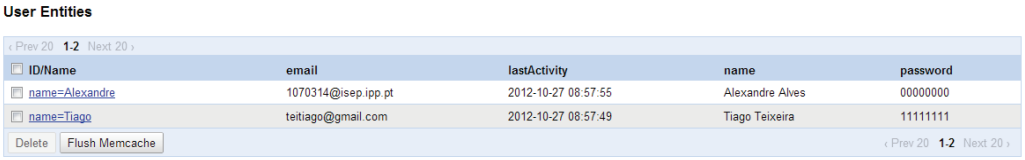

Figura 4.13: Resultado da edição de utilizadores no Datastore.

4.3.2 Instalações e Contadores

Os dois clientes introduzidos possuem instalações com contadores inteligentes representados no ACE Vision. Os dados dos contadores permanecem privados para garantir a confidencialidade da informação.

Foi criada para cada utilizador uma instalação com um contador residencial, os respectivos dados estão representados nas Tabelas 4.32 e 4.33.

Tabela 4.32: Informação geral das instalações.

Proprietário	**Nome**	**Potência**
Tiago	ISEP	6.9
Alexandre	Itron Server Room	6.9

Tabela 4.33: Localização das instalações.

País	**Região**	**Cidade**	**Cód. Postal**
Portugal	Porto	Paranhos	4200
Portugal	Braga	Vila Nova de Famalicão	4760

A cada instalação foi adicionado um contador residencial com as características gerais e de tarifas representadas nas Tabelas 4.34 e 4.35, respectivamente.

Tabela 4.34: Contadores de teste.

Instalação	**N.° de Série**	**Marca**	**Fluxo**	**Modelo**	**N.° de Tarifas**
ISEP	001	Outro	Importação	0	2
Itron Server Room	001	Itron	Importação	2000	2

Tabela 4.35: Tarifas dos contadores.

Contador	**Identificador**	**Unidade**	**Preço Unitário (€/kWh)**	**Em Uso**	**Validade**
Outro@001	0	kWh	0,10	`true`	2012-10-27
Outro@001	1	kWh	0,15	`true`	2012-10-27
Itron@001	0	kWh	0,11	`true`	2012-10-27
Itron@001	1	kWh	0,16	`true`	2012-10-27

A Figura 4.14 ilustra o processo de criação de uma instalação no dispositivo móvel. A criação consiste na introdução das definições gerais dos contadores associados e, por último, da localização. Na aplicação *Web* o processo, representado pelas Figuras 4.15 e 4.16, é idêntico diferindo apenas na ordem da adição dos contadores. No caso do dispositivo móvel esta operação é realizada durante o processo de criação da instalação e na aplicação *Web* é efectuada após a criação da instalação.

A Figura 4.16 ilustra o processo de criação de uma instalação através da aplicação móvel incluindo a criação de um contador e as respectivas tarifas.

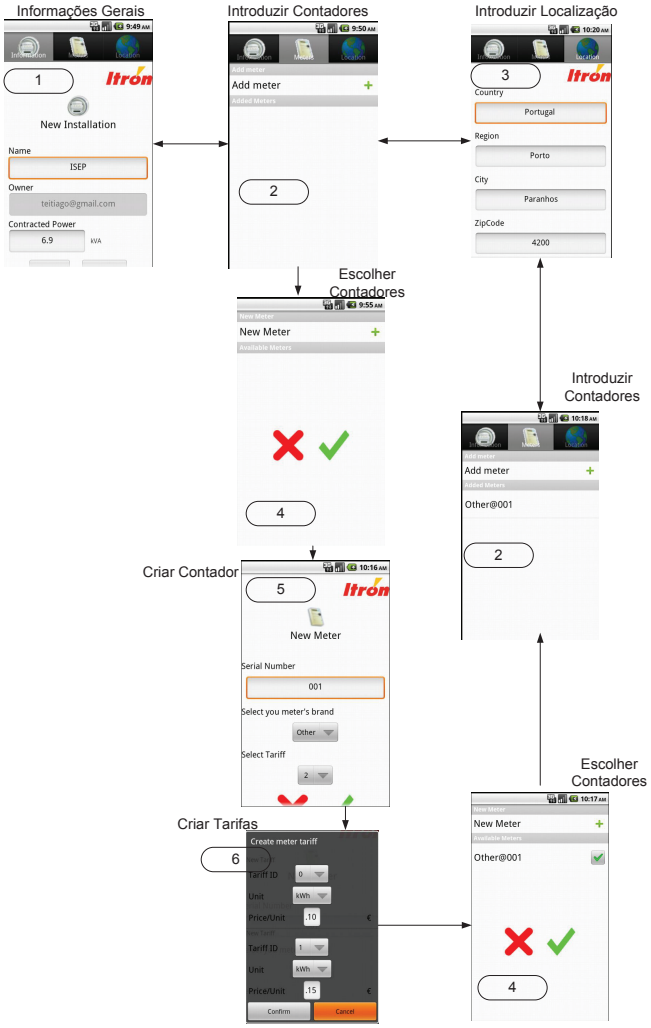

Figura 4.14: Processo de criação de instalação na aplicação móvel.

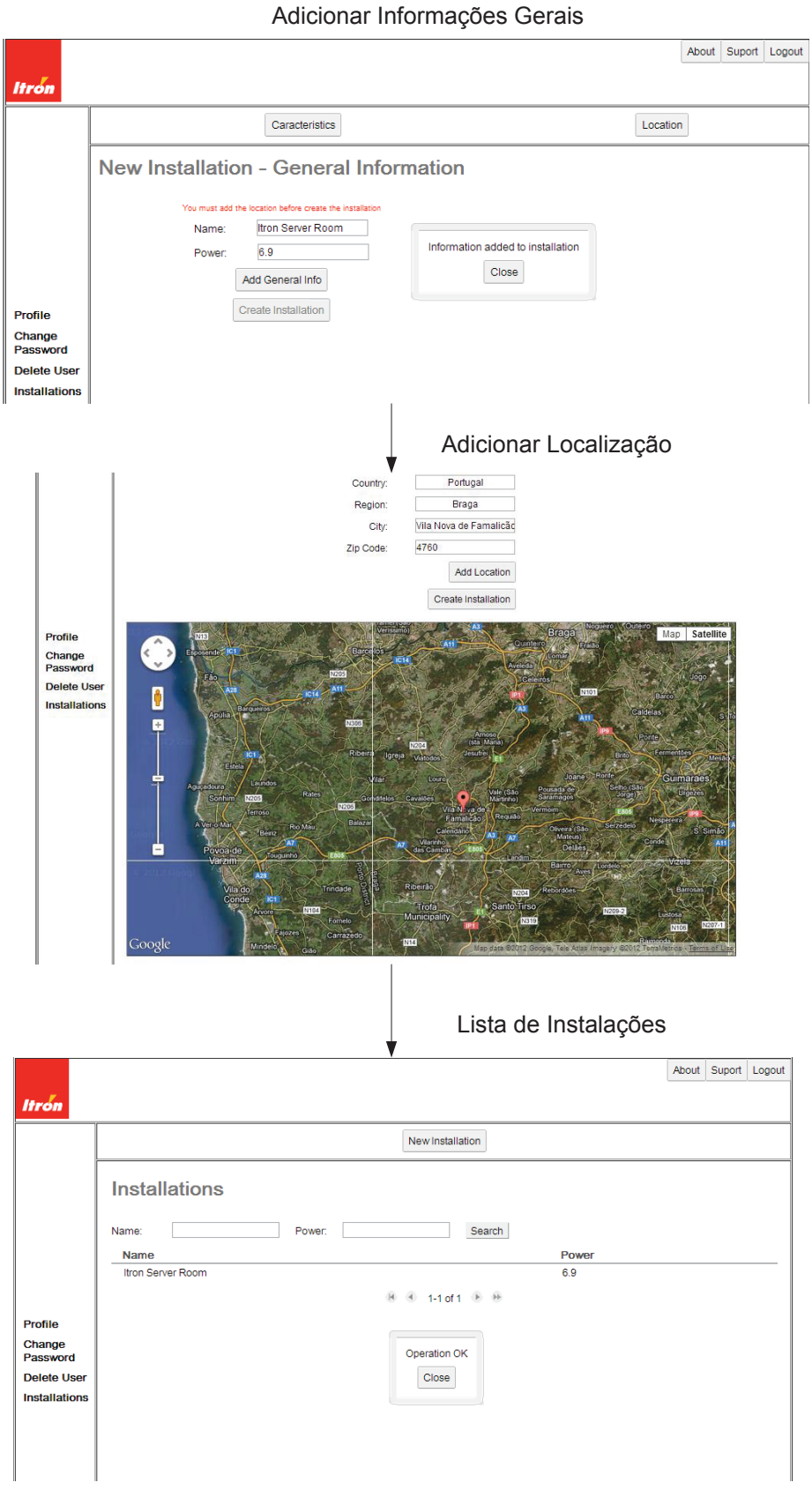

Figura 4.15: Criação de uma instalação na aplicação *Web*.

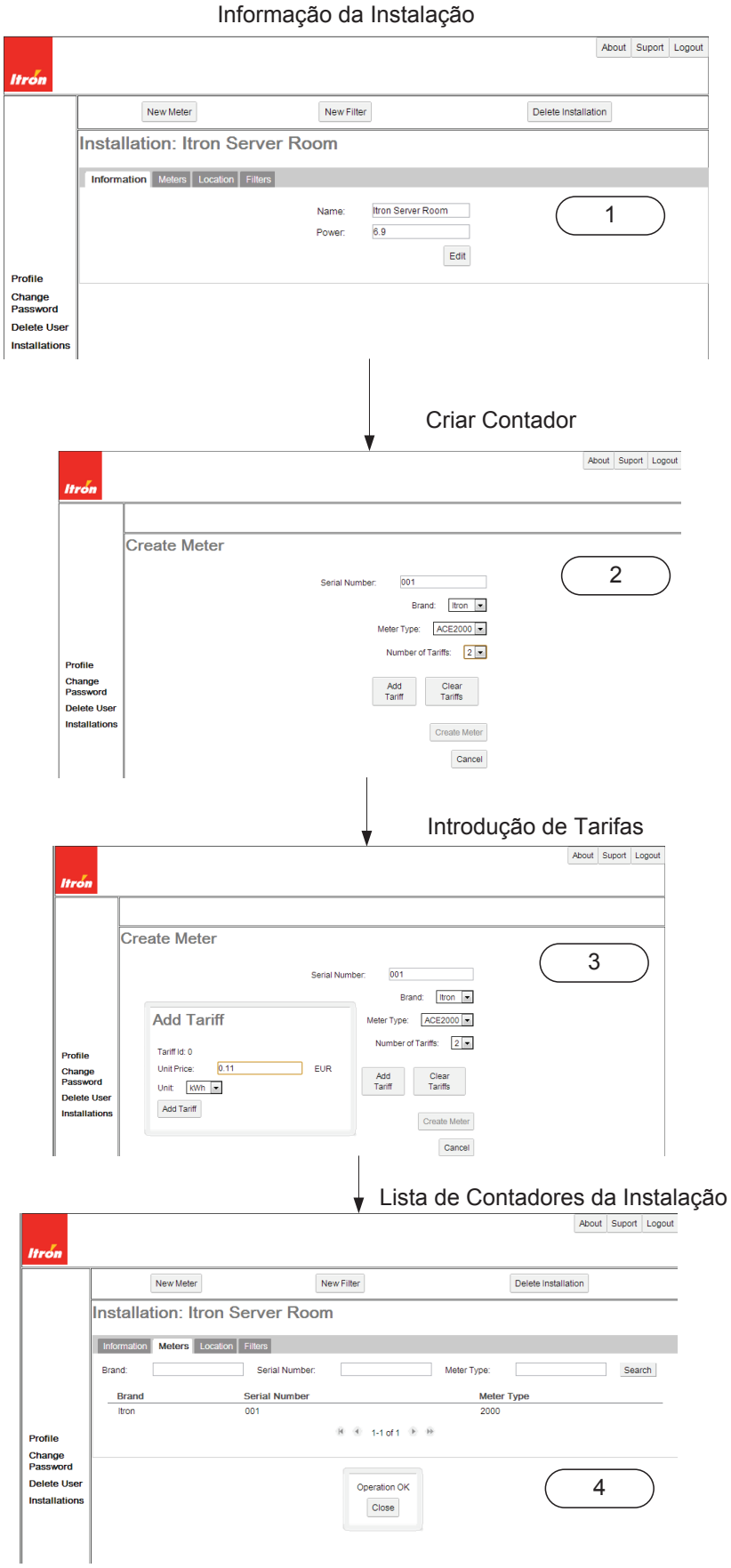

Figura 4.16: Criação de um contador e respectivas tarifas na aplicação *Web*.

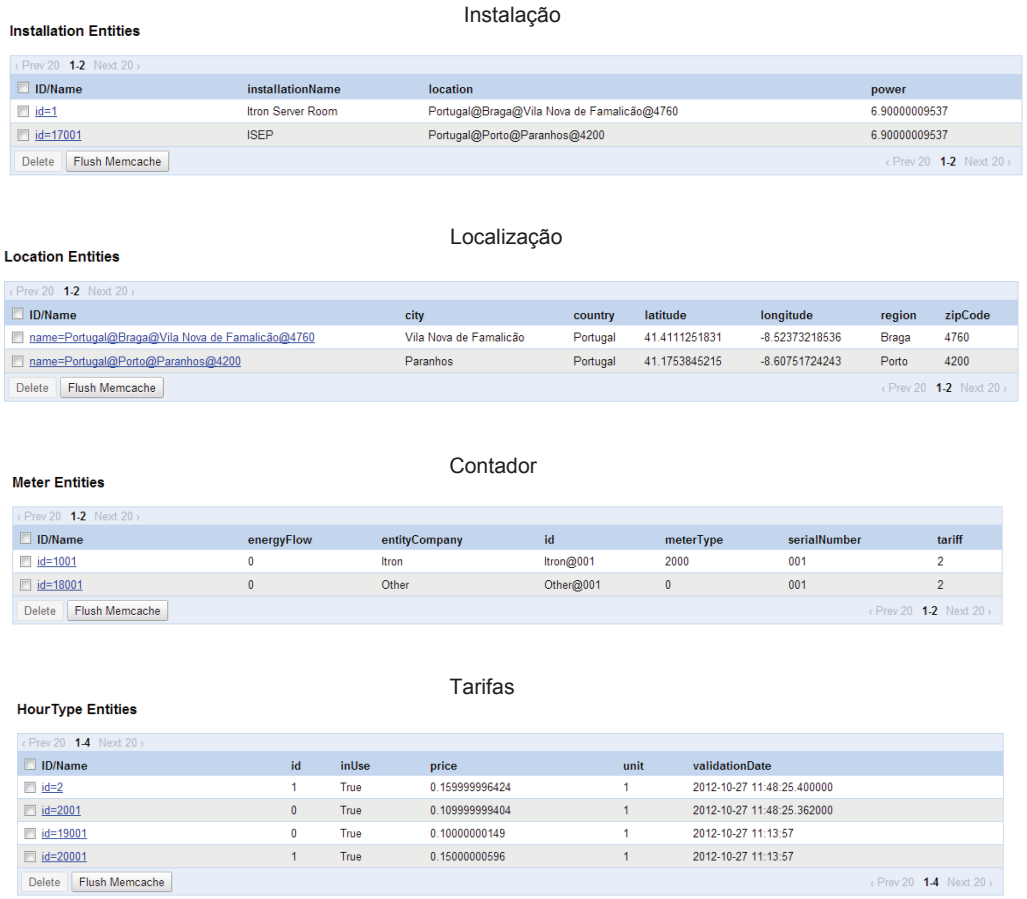

Figura 4.17: Armazenamento das instalações, contadores e tarifas no Datastore.

4.3.3 Submissão e Agendamento de Leituras

Nos contadores residenciais é necessário proceder à recolha e submissão manual das leituras do contador. Assim, quer na aplicação *Web*, quer no dispositivo móvel existem os meios necessários à submissão de leituras. Foram introduzidas quatro leituras por tarifa e por contador. As leituras introduzidas estão representadas na Tabela 4.36.

Na aplicação móvel procedeu-se ao agendamento de uma tarefa que alerta o utilizador de 5 min em 5 min para a realização de uma leitura.

Na Figura 4.18 ilustra-se a sequência de actividades necessária para o agendamento de leituras no dispositivo móvel. A Figura 4.19 apresenta o armazenamento das leituras no Datastore e as estatísticas criadas. Como se pode verificar a partir da Tabela 4.36, o valor médio das leituras armazenadas é 2,0 kWh e os gastos to-

Tabela 4.36: Leituras introduzidas.

Contador	Tarifa	Unidade	Valor
Outro@001	0	kWh	1
Itron@001	0	kWh	1
Outro@001	1	kWh	2
Itron@001	1	kWh	2
Outro@001	0	kWh	3
Itron@001	0	kWh	3
Outro@001	1	kWh	4
Itron@001	1	kWh	4
Outro@001	0	kWh	5
Itron@001	0	kWh	5
Outro@001	1	kWh	6
Itron@001	1	kWh	6
Outro@001	0	kWh	7
Itron@001	0	kWh	7
Outro@001	1	kWh	8
Itron@001	1	kWh	8

tais e médios correspondem aos valores esperados. Esta análise permitiu concluir que os cálculos apresentados pelo serviço estão correctos.

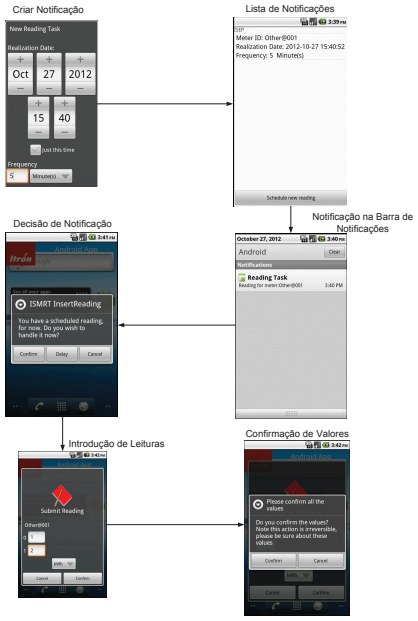

Figura 4.18: Sequência de criação de notificações e leituras.

Na Figura 4.20 está representada a sequência adoptada na aplicação *Web* para proceder à introdução de uma leitura. Na Figura 4.21 apresenta-se o conteúdo actualizado do Datastore.

Figura 4.19: Armazenamento de leituras no Datastore.

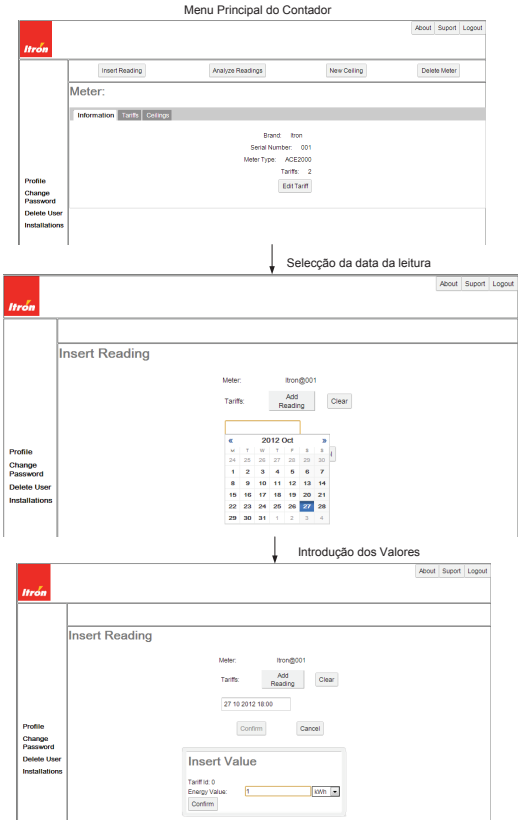

Figura 4.20: Sequência de criação de leituras na aplicação *Web*.

Lista de Leituras

Reading Entities

‹ Prev 20 **1-16** Next 20 ›

ID/Name	dateTime	hourType	timeZoneId	unit	value
id=1002	2012-10-27 18:15:00	1	Europe/London	1	8.0
id=3001	2012-10-27 18:00:00	0	Europe/London	1	1.0
id=3002	2012-10-27 18:10:00	1	Europe/London	1	6.0
id=4001	2012-10-27 18:00:00	1	Europe/London	1	2.0
id=4002	2012-10-27 18:05:00	1	Europe/London	1	4.0
id=5001	2012-10-27 18:05:00	0	Europe/London	1	3.0
id=6001	2012-10-27 18:10:00	0	Europe/London	1	5.0
id=7001	2012-10-27 18:15:00	0	Europe/London	1	7.0
id=17002	2012-10-27 15:46:13.140000	0	GMT	1	3.0
id=18002	2012-10-27 15:56:15.649000	1	GMT	1	8.0
id=20002	2012-10-27 15:42:50.403000	0	GMT	1	1.0
id=21001	2012-10-27 15:42:50.403000	1	GMT	1	2.0
id=22001	2012-10-27 15:46:13.140000	1	GMT	1	4.0
id=23001	2012-10-27 15:52:48.128000	0	GMT	1	5.0
id=24001	2012-10-27 15:52:48.128000	1	GMT	1	6.0
id=25001	2012-10-27 15:56:15.649000	0	GMT	1	7.0

Delete Flush Memcache ‹ Prev 20 **1-16** Next 20 ›

Estatisticas

ResidentialStatistic Entities

‹ Prev 20 **1-4** Next 20 ›

ID/Name	average	averageSpent	cadence	hourType	introductionNumber	lastIntroduction	lastIntroductionDate	totalSpent	unit
name=Itron@001@0	2.0	0.219999998808	0	0	4	7.0	2012-10-27 12:15:00	0.659999966621	1
name=Itron@001@1	2.0	0.319999992847	0	1	4	8.0	2012-10-27 12:15:00	0.959999978542	1
name=Other@001@0	2.0	0.20000000298	0	0	4	7.0	2012-10-27 15:56:15.649000	0.600000023842	1
name=Other@001@1	2.0	0.300000011921	0	1	4	8.0	2012-10-27 15:56:15.649000	0.900000035763	1

Delete Flush Memcache ‹ Prev 20 **1-4** Next 20 ›

Figura 4.21: Armazenamento dos dados no Datastore.

4.3.4 Análise de Consumo e Produção

Para avaliar a análise de consumo e produção foram realizados dois conjuntos de testes: um para contadores residenciais e outro para contadores inteligentes.

4.3.4.1 Análise de Consumo Residencial

Neste teste foram utilizados os dados da Tabela 4.36. No dispositivo móvel o acesso ao menu de leituras obriga à selecção prévia de um contador e à selecção da opção “obter leituras”. Indicam-se as datas da análise e submete-se o pedido – ver Figura 4.22. Caso não existam leituras relativas ao período especificado, é gerado um aviso.

Na Figura 4.23 apresentam-se as estatísticas do consumo/tarifa. Os valores apresentados foram verificados e estão correctos. A Figura 4.24 contém os gráficos obtidos, na Figura 4.24 - 1 apresenta-se os gráficos da evolução do consumo relativo às duas tarifas associadas ao contador. As duas rectas neste caso são pouco perceptíveis porque estão sobrepostas. A Figura 4.24 - 2 mostra a repartição do consumo energético pelas tarifas onde cada tarifa totalizou 6,0 kWh. A Figura

4.24 - 3 representa o preço a pagar por tarifa. O total consumido das duas tarifas foi 1,5 € correspondendo a mesma tarifa a 60 % e a segunda tarifa a 40 %. Por fim, na Figura 4.25 listam-se as leituras de cada tarifa.

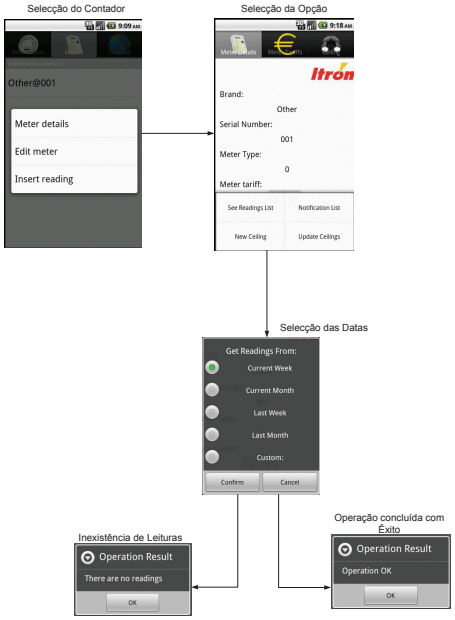

Figura 4.22: Sequência de obtenção de leituras.

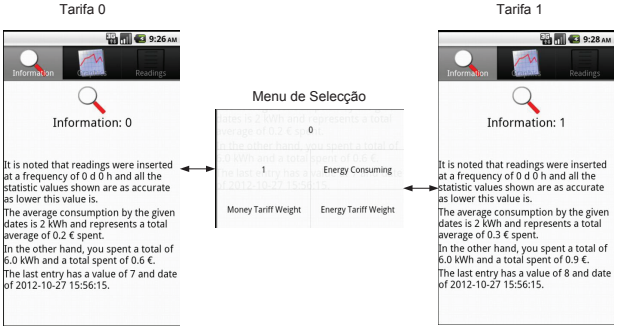

Figura 4.23: Informação da amostra das leituras obtidas.

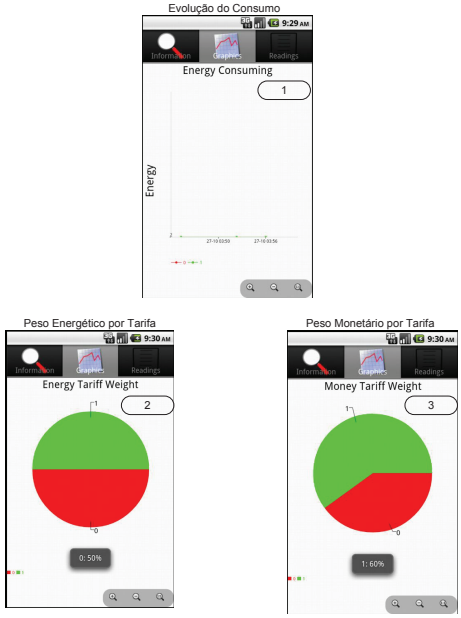

Figura 4.24: Gráficos das leituras.

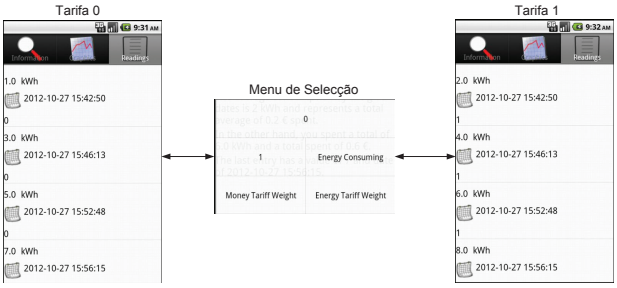

Figura 4.25: Lista de leituras.

Na Aplicação *Web* o processo é idêntico, mudando apenas a forma como é exibida a informação. Escolhe-se o contador e selecciona-se a opção de "análise de leituras", – ver Figura 4.26 - 1. Especifica-se o período de análise como se representa na Figura 4.26 - 2. Como os dados introduzidos para este contador são iguais aos do caso anterior, os resultados são os mesmos.

Os gráficos apresentados ao utilizador são iguais aos da aplicação móvel – ver Figura 4.27. A Figura 4.28 apresenta as leituras do contador relativas ao período seleccionado podendo-se aplicar filtros por tarifa.

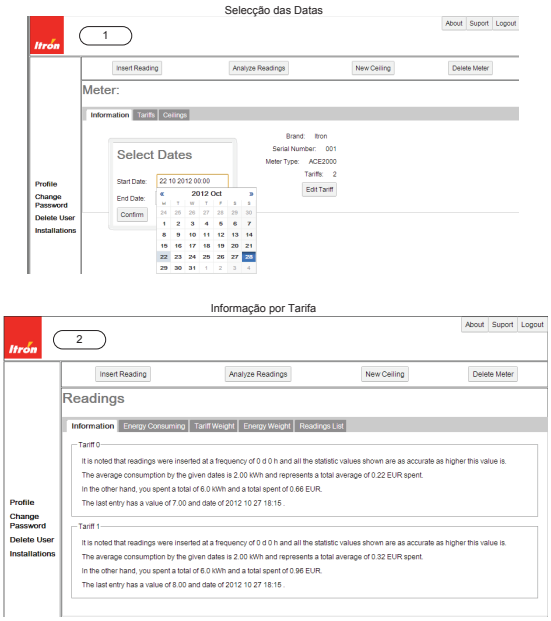

Figura 4.26: Obtenção de leituras e dados por tarifa na aplicação *Web*.

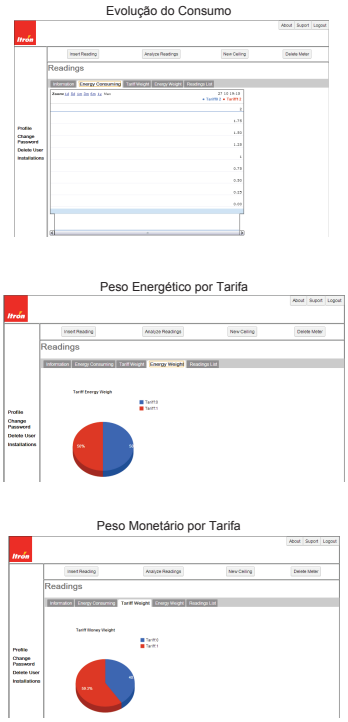

Figura 4.27: Lista de gráficos na aplicação *Web*.

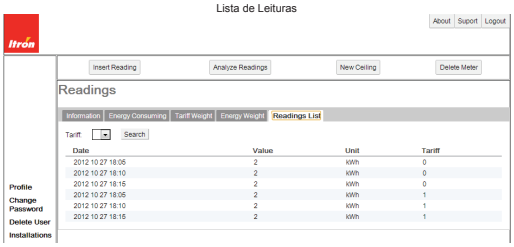

Figura 4.28: Lista de leituras na aplicação *Web*.

4.3.4.2 Análise de Consumo C&I

Adicionou-se a uma instalação de um utilizador pré-existente um contador inteligente de importação a um deles para avaliar o processamento da informação recebida. O contador escolhido foi configurado para quatro tarifas e introduziram-se as tarifas que se apresentam na Tabela 4.37.

Tabela 4.37: Tarifas dos contadores.

Identificador	**Unidade**	**Preço Unitário (€/kWh)**	**Em Uso**	**Validade**
0	kWh	0,10	`true`	2012-10-28
1	kWh	0,15	`true`	2012-10-28
2	kWh	0,20	`true`	2012-10-28
3	kWh	0,15	`true`	2012-10-28

Definiram-se três filtros de pesquisa (um filtro de curva de carga, um filtro de fim de facturação e um filtro de potência) Tabela 4.38. A Figura 4.29 exemplifica a sequência de passos adoptada neste processo. O resultado é apresentado na Figura 4.30.

Tabela 4.38: Filtros de pesquisa ACE Vision.

Nome	**Tipo**	**Modo de Data**
LP	Curva de Carga	Personalizado
EoB	Fim de Facturação	Personalizado
P	Potência Máxima	Personalizado

Recolheram-se os valores do consumo energético do intervalo [2012-09-29; 2012-09-30] e confirmou-se que os valores apresentados coincidiam com os dados armazenados no ACE Vision. Na Figura 4.31 apresenta-se o conjunto de capturas de ecrã que constituem a análise da curva de carga.

Com o filtro de fecho de facturação recolheram-se os dados de Agosto a Outubro. Os totais obtidos corresponderam aos totais no ACE Vision. A Tabela 4.39

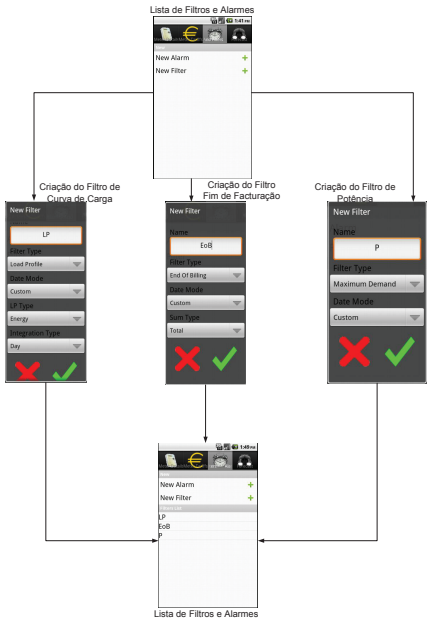

Figura 4.29: Criação de filtros de pesquisa.

Filtros de Pesquisa ACE Vision

Filter Entities

‹ Prev 20 **1-3** Next 20 ›

ID/Name	DateMode	FilterType	Name	SumType	Integration	LPType
id=42002	0	2	P	<missing>	<missing>	<missing>
id=45002	0	1	EoB	0	<missing>	<missing>
id=46001	0	0	LP	<missing>	1	0

Delete Flush Memcache ‹ Prev 20 **1-3** Next 20 ›

Figura 4.30: Aplicação dos filtros de pesquisa ao Datastore.

apresenta o cálculo da distribuição do consumo e dos custos por tarifa que são coincidentes com os valores apresentados na Figura 4.32.

Tabela 4.39: Cálculo dos pesos do fim de facturação.

Tarifa	**Energia (kWh)**	**Peso Energético**	**Preço Unitário (€/kWh)**	**Peso Monetário**
0	1606	38,2 %	0,10	23 %
1	373	8,8 %	0,15	8 %
2	1511	35,8 %	0,20	43 %
3	725	17,2 %	0,25	26 %

As potências recolhidas através do filtro de potência para o mesmo período do teste anterior coincidiram com os valores do ACE Vision. Os resultados estão ilustrados na Figura 4.33.

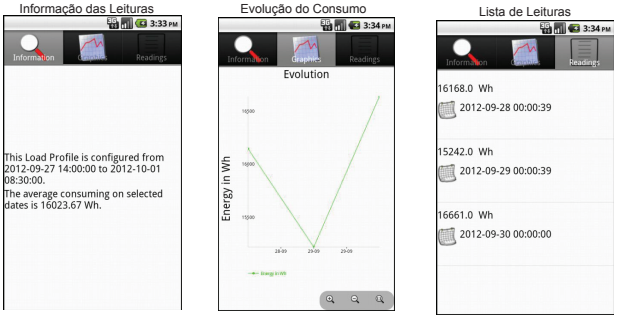

Figura 4.31: Aplicação do filtro de pesquisa de curva de carga.

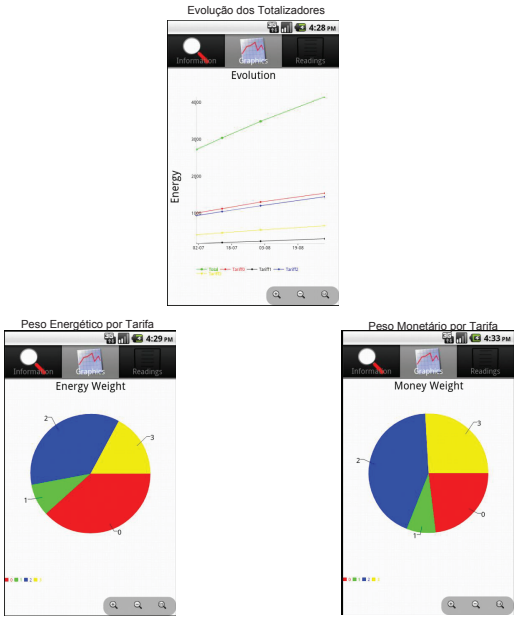

Figura 4.32: Gráficos do filtro fim de facturação.

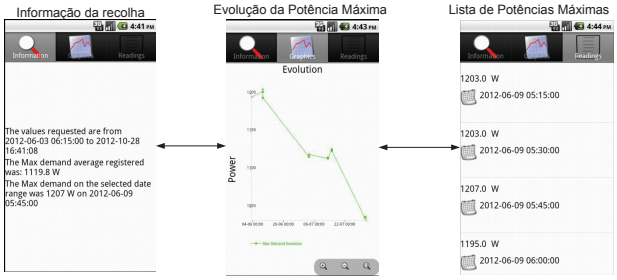

Figura 4.33: Recolha da potência máxima.

Este conjunto de testes permitiu concluir que a recolha de valores do servidor ACE Vision está operacional.

4.3.4.3 Análise de Produção

A análise de produção é semelhante à análise de consumo. A única diferença reside no facto que neste caso o fluxo de energia passa a ser de exportação em vez de importação. A apresentação dos resultados só difere no gráfico adicional que representa os ganhos monetários em função do tempo. O resultado é mostrado na Figura 4.34.

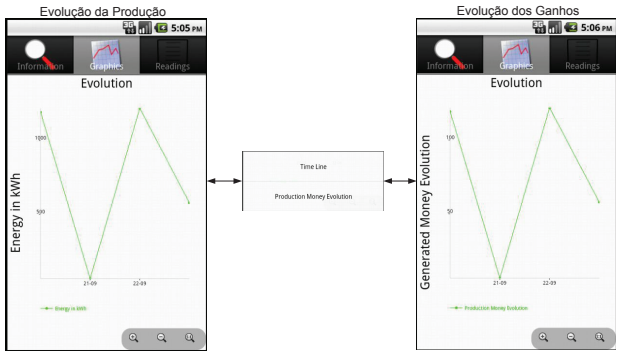

Figura 4.34: Evolução da produção de um contador.

4.3.5 Tectos de Consumo

Para testar os limites máximos monetários estabelecidos pelo utilizador foram criados dois tectos de consumo: um para a aplicação *Web* e outro através da aplicação móvel. Os testes foram definidos para os contadores residenciais já existentes com leituras já introduzidas. Na Tabela 4.40 estão definidas as características dos tectos.

Tabela 4.40: Tectos de consumo.

Contador	Data Inicial	Data Final	Tarifa	Limite Máximo
Other@001	2012-10-28	2012-11-01	0	0,50 €
Other@001	2012-10-28	2012-11-01	1	0,75 €
Itron@001	2012-10-28	2012-11-01	0	0,55 €
Itron@001	2012-10-28	2012-11-01	1	0,80 €

Introduziram-se as leituras apresentadas na Tabela 4.41.

Na Figura 4.35 ilustra-se o processo de criação de um tecto de consumo no dispositivo móvel e na Figura 4.36 na aplicação *Web*. Após a submissão da última

leitura de cada tarifa (13 kWh e 14 kWh, respectivamente), foi desencadeado um alerta em ambas as aplicações – ver Figura 4.37. O alerta foi correctamente gerado porque o total consumido e o total dispendido excederam para ambas as tarifas os 75 % estabelecidos. A Figura 4.38 apresenta os dados dos tectos de consumo armazenados no Datastore.

Tabela 4.41: Leituras introduzidas no tecto de consumo.

Contador	**Tarifa**	**Unidade**	**Valor**
Other@001	0	kWh	9
Other@001	1	kWh	10
Other@001	0	kWh	11
Other@001	1	kWh	12
Other@001	0	kWh	13
Other@001	1	kWh	14
Itron@001	0	kWh	9
Itron@001	1	kWh	10
Itron@001	0	kWh	11
Itron@001	1	kWh	12
Itron@001	0	kWh	13
Itron@001	1	kWh	14

Figura 4.35: Criação e listagem de um tecto de consumo na aplicação móvel.

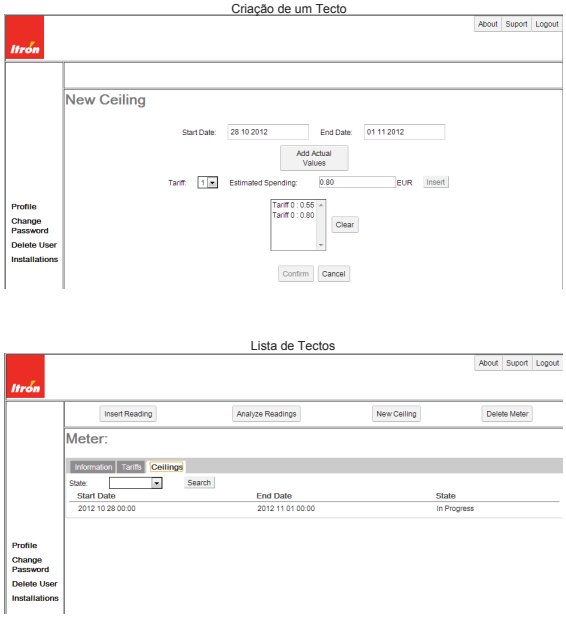

Figura 4.36: Criação e listagem de um tecto de consumo na aplicação *Web*.

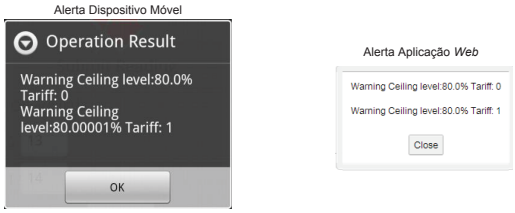

Figura 4.37: Exemplos de alertas de tectos de consumo.

Figura 4.38: Armazenamento dos tectos de consumo no Datastore.

4.3.6 Alarmes

Para testar os alarmes de consumo foi utilizado um contador inteligente de consumo. Como os alarmes são automáticos e para não se aguardar pela execução do *cron job*, alterou-se a configuração da máquina de *back-end* para passar a estar acessível. Analisaram-se os consumos do contador e escolheu-se um valor que activasse os alarmes. Na Figura 4.39 está ilustrada a criação do alarme. Acedendo ao *back-end* e ao *servlet* responsável pela actualização dos alarmes (`update.URL/update-statistic`), forçou-se a actualização do alarme. No dispositivo móvel actualizou-se a informação do contador e obteve-se o resultado da Figura 4.39. A Figura 4.40 apresenta os alarmes armazenados no Datastore.

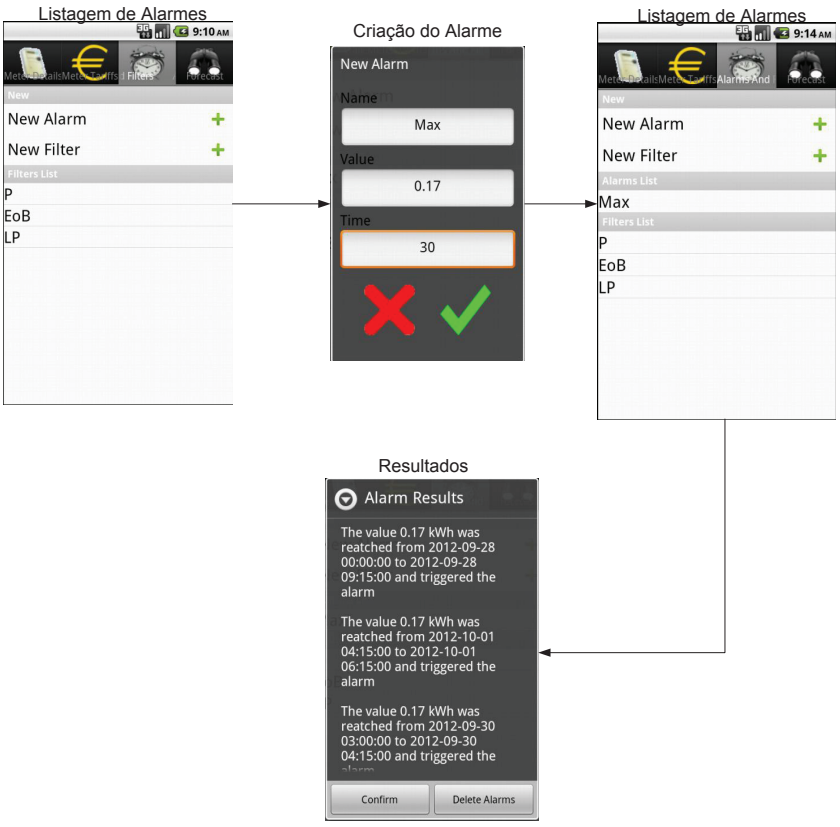

Figura 4.39: Criação e listagem de alarmes na aplicação móvel.

Figura 4.40: Armazenamento dos alarmes no Datastore.

4.3.7 Comparação Anónima de Consumos

Para testar esta funcionalidade foram utilizados os contadores residenciais criados e as respectivas leituras. No caso do dispositivo móvel estabeleceu-se uma comparação com os consumidores com um perfil de consumo idêntico da região de Braga, Figura 4.41. No caso da aplicação *Web* definiu-se uma comparação no âmbito da região do Porto – ver Figura 4.42. Na Figura 4.43 apresenta-se o armazenamento de filtros de comparação por regiões.

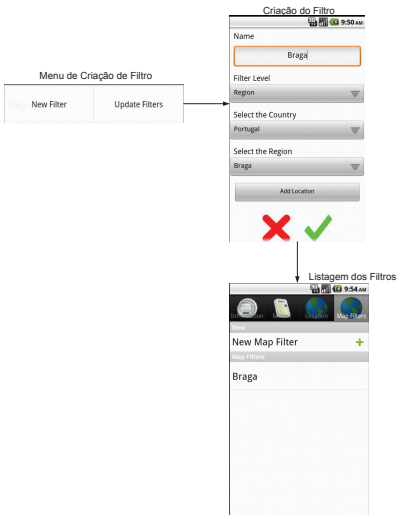

Figura 4.41: Criação e selecção de um filtro de mapa na aplicação móvel.

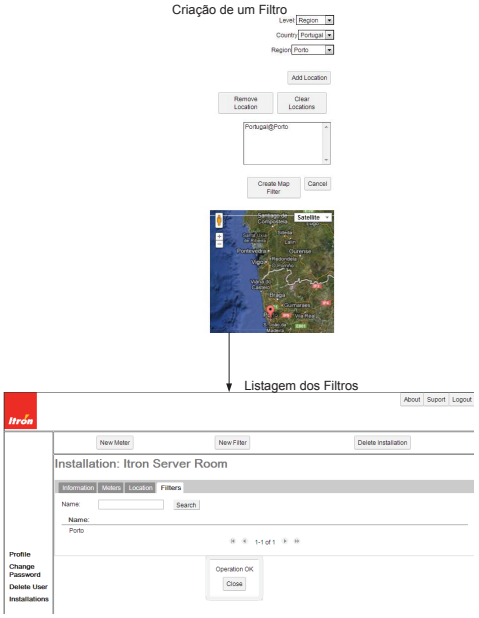

Figura 4.42: Criação e selecção de um filtro de mapa na aplicação *Web*.

Figura 4.43: Armazenamento dos filtros de região no Datastore.

Dado que a média de consumo dos dois contadores é igual (2 kWh) e as tarifas da região de Braga são mais caras 0,01 €/kWh, é de prever que, no caso da aplicação móvel, o marcador seja preto e, no caso da aplicação *Web*, como apresenta um custo superior ao da média dos consumidores da região, o marcador seja vermelho. A Figura 4.44 apresenta o resultado de cada uma das operações.

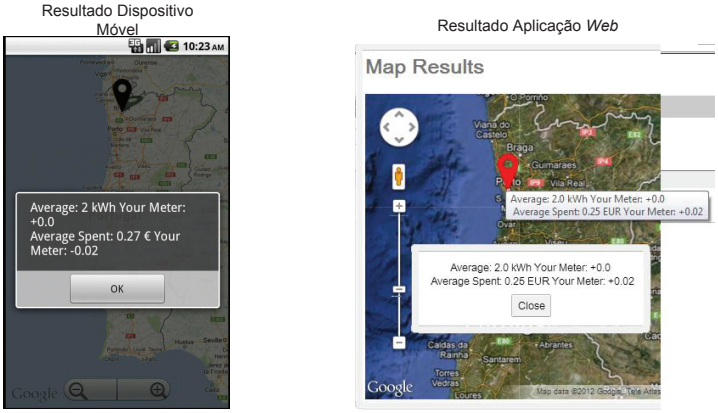

Figura 4.44: Duas comparações de consumo.

4.3.8 Modelos de Previsão

Foi escolhido um contador de produção para a realização deste teste. Na Figura 4.45 está representado o pedido de criação de um modelo e o resultado obtido.

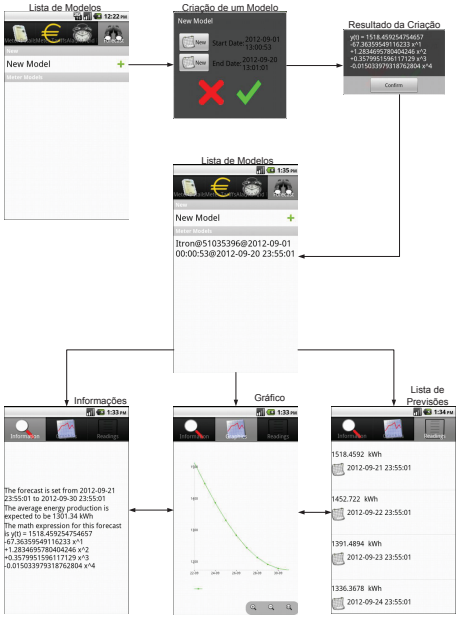

Figura 4.45: Criação de um modelo de previsão.

Para validar os resultados obtidos, procedeu-se à análise dos dados recolhidos através do Excel. Obteve-se a curva representada na Figura 4.46. É perceptível

que dois pontos fogem à média dos valores obtidos. Por essa razão, foi desenvolvido um algoritmo de correcção. O processo consiste em definir uma janela de largura igual a sete, contendo os sete pontos que antecedem o valor em análise. Se o valor não estiver compreendido entre $[\bar{x} - 30\ \%; \bar{x} + 30\ \%]$, onde $\bar{x}$ representa o valor médio dos últimos sete pontos, então o valor analisado passa a ser igual à média dos pontos, *i.e.*, $\bar{x}$. O resultado é apresentado na Figura 4.47.

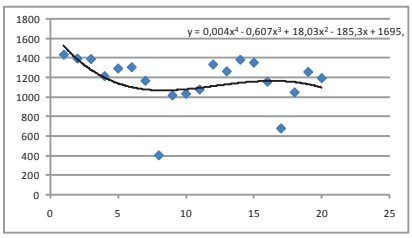

Figura 4.46: Valores de produção brutos.

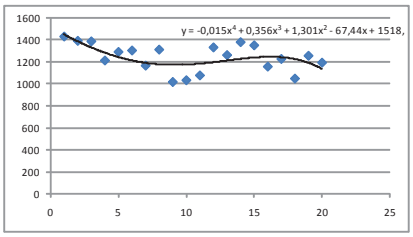

Figura 4.47: Valores de produção corrigidos.

A função obtida no Excel (Equação 4.9) e a obtida pela aplicação móvel (Equação 4.10) são idênticas.

Equação 4.9: Função polinomial obtida através do Excel.

$$y(t) = -0,02t^4 + 0,36t^3 + 1,30t^2 - 67,44t + 1518,00 \tag{4.9}$$

Equação 4.10: Função polinomial obtida através da aplicação móvel.

$$y(t) = -0,02t^4 + 0,36t^3 + 1,28t^2 - 67,36t + 1518,46 \tag{4.10}$$

4.3.9 Análise de Desempenho do Sistema

Uma vez que uma comunicação de dados móvel acarreta custos para o utilizador, realizou-se um conjunto de testes para avaliar o tempo de comunicação e o volume

de dados trocados entre a aplicação móvel e o Google App Engine através da ferramenta Wireshark.

Os testes visaram os métodos de submissão e de obtenção de informação. Nos testes de submissão de dados seguiram-se os seguintes passos: (*i*) eliminou-se a informação armazenada no Datastore; (*ii*) criou-se um utilizador; (*iii*) uma instalação com dois contadores (um residencial e outro inteligente); (*iv*) introduziram-se dez leituras; e (*v*) criou-se um tecto de consumo. Nos testes de obtenção de informação solicitaram-se dez leituras residenciais, um mês de curva de carga de um contador inteligente com actualização diária, três meses de fim de facturação e de potência máxima, os alarmes do contador (22), um modelo de previsão com um mês e um mapa de comparação. Por último, apagaram-se os dados armazenados no dispositivo móvel e recuperou-se a informação de um utilizador. Os resultados da submissão de dados apresentam-se na Tabela 4.42 e os resultados da obtenção de informação na Tabela 4.43.

Tabela 4.42: Submissão de dados através da aplicação móvel.

Operação	**Tempo de Comunicação** (**s**)	**Tempo de Resposta** (**s**)	**Pedido** (**B**)	**Resposta** (**B**)
Criar Utilizador	1,74	1,26	771	447
Criar Instalação	4,34	1,56	5404	2345
Submeter Leitura	0,95	0,30	1030	465
Criar Tecto	1,36	0,51	1490	455

Tabela 4.43: Obtenção de dados através da aplicação móvel.

Operação	**Tempo de Comunicação** (**s**)	**Tempo de Resposta** (**s**)	**Pedido** (**B**)	**Resposta** (**B**)
Consumo Residencial	0,59	0,05	963	2162
Curva de Carga	17,00	16,30	1322	3146
Fim de Facturação	7,92	7,10	1258	1530
Potencia Máxima	8,49	7,65	1229	2673
Obter Alarmes	1,65	0,35	954	4778
Gerar Modelo	15,00	14,50	1122	2787
Obter Mapa	2,28	0,65	1453	170860
Recuperar Utilizador	22,50	22,00	17555	13464

Estes testes envolveram um tráfego total de dados descarregados de 205,112 kB e de dados enviados de 34,551 kB. Este tráfego corresponde, no caso de um pacote mensal padrão de transferência de dados móveis com 500 MB de descarga incluídos, a 0,04 %. As taxas da transmissão de dados são apresentadas na Tabela 4.44. Estes valores foram conseguidos recorrendo à página `http://www.speedtest.net/` que permite medir as taxas de *download* e *upload* de uma ligação.

Tabela 4.44: Taxas de *download* e *upload* de dados.

Operação	***Download*** **(Mb/s)**	***Upload*** **(Mb/s)**
Criação	20,10	8,16
Leitura	6,96	6,95

Pode-se verificar que algumas operações são mais demoradas, designadamente a recuperação de um utilizador ou a criação de uma instalação, porque desencadeiam a troca de várias mensagens entre cliente e servidor. Os métodos de recolha dependentes da aplicação ACE Vision revelaram-se mais lentos devido à interacção adicional que acarretam. Não foi avaliada a velocidade de *upload* da máquina em que está alojada a aplicação.

4.4 Conclusão

Foi apresentada a arquitectura da solução proposta e descritos os quatro módulos constituintes: a aplicação *Web*, a aplicação móvel, o serviço e o armazenamento de dados no Datastore. A organização da informação do Datastore revelou-se de simples compreensão e uso. Do ponto de vista funcional, os testes e a depuração efectuados permitiram avaliar e confirmar o correcto funcionamento do protótipo. A adopção de um serviço *Web* do lado do *back-end* tornou o desenvolvimento de ambas as aplicações cliente mais rápido e permite a integração futura de outras aplicações.

Concluiu-se que o volume de informação trocada entre a aplicação móvel cliente e o servidor é de baixa relevância, tendo encargos insignificantes ao nível de uma assinatura mensal padrão.

No capítulo seguinte efectua-se uma análise da solução desenvolvida e propõem-se eventuais desenvolvimentos futuros.

Capítulo 5

Conclusões

Neste capítulo são avaliados os resultados alcançados face aos objectivos propostos e são identificadas as limitações e propostas sugestões de desenvolvimento futuro.

5.1 Balanço

Os objectivos propostos foram integralmente alcançados. O principal objectivo foi o desenvolvimento de uma solução capaz de dotar um utilizador dos meios necessários ao controlo e racionalização do consumo/produção de energia eléctrica. O sistema desenvolvido permite definir utilizadores, instalações, tectos de consumo (alertas) e alarmes, submeter leituras, comparar o consumo/produção com clientes com o mesmo perfil, obter a representação gráfica do consumo/produção de energia eléctrica e, no caso de instalações industriais, efectuar a predição.

Trata-se de um sistema distribuído dividido em *back-end* e *font-end*. O *back-end*, que inclui o armazenamento persistente de dados e o serviço *Web* de processamento de dados, encontra-se alojado numa plataforma de *cloud computing* – Google App Engine – assegurando interoperabilidade, escalabilidade e robustez. O *front-end* inclui duas aplicações de interface com o utilizador: a aplicação móvel para dispositivos Android e a aplicação *Web*.

O conjunto de testes efectuado às funcionalidades do *back-end* e *front-end* permitiu verificar o correcto funcionamento dos diferentes módulos. Os testes de desempenho realizados para a aplicação móvel concluíram que o volume de informação trocado com o servidor é reduzido. O maior volume de tráfego gerado ocorre durante a obtenção de mapas do servidor Google Maps ($\approx$ 180 kB). De qualquer forma, o tráfego gerado corresponde a encargos financeiros insignificantes para os detentores de uma assinatura mensal de transmissão de dados

padrão. Em termos de resposta temporal, o pior caso correspondeu à operação de recuperação de um utilizador que demorou ≈ 22 s.

5.2 Desenvolvimentos Futuros

Termina-se esta dissertação propondo um conjunto de sugestões de desenvolvimento futuro destinado a ultrapassar as limitações identificadas:

***Layout*:** A interface com o utilizador quer da aplicação *Web*, quer da aplicação móvel pode ser melhorada através da adopção de ícones dimensionados em função das restrições físicas do equipamento, da reorganização da navegação e da uniformização dos *layouts* das duas aplicações cliente.

Apresentação de Conteúdos: A informação sobre os tectos de consumo e alarmes pode ser representada em formato gráfico ao invés do formato de texto actualmente adoptado.

Encriptação: As credenciais dos utilizadores podem ser encriptadas para prevenir o acesso malicioso aos dados.

Gestão da Informação: A edição de leituras de contadores e da localização das instalações, por um lado, e a transformação da eliminação de entidades no Datastore em inactivação (mudança de estado) permitiriam corrigir eventuais lapsos do utilizador e manter o histórico das instalações que entretanto ficassem obsoletas, respectivamente.

Modelo de Predição: O modelo deve ser enriquecido para permitir alargar a predição aos contadores residenciais.

Instalações: O perfil de cada instalação deve ser enriquecido com o número e as características dos equipamentos ligados, a área da instalação e o número de utilizadores.

Contadores: A definição dos tectos de consumo deve ser baseada no modelo de predição, *i.e.*, se o utilizador definir um tecto de consumo incompatível com o seu padrão habitual, deverá ser alertado. A colocação de etiquetas de código *Quick Response* (QR) junto dos contadores permitiria a sua identificação automática.

Por último, seria interessante efectuar uma comparação entre as plataformas PaaS de *cloud computing* disponíveis e proceder à validação pelos utilizadores do conjunto de funcionalidades implementadas para recolher sugestões conducentes à melhoria da experiência de navegação.

5.3 Conclusão

O protótipo implementado permite aos utilizadores monitorar o seu consumo energético ou produção energética a partir de um dispositivo Android ou de um navegador *Web*.

A adopção de uma plataforma de *cloud computing* para alojar o *back-end* assegurou a interoperabilidade, escalabilidade e robustez da solução. A capacidade de processamento e de armazenamento oferecidas pela Google foram suficientes em toda a fase de desenvolvimento e teste.

A plataforma Android apresentou uma capacidade de processamento e qualidade de interface adequadas. A aplicação móvel recorre à informação disponibilizada pelo sensor de posicionamento GPS para determinar a sua posição. Esta informação suporta as operações de localização automática de instalações e utilizadores.

A aplicação *Web* ilustra a interoperabilidade do *back-end*. O GWT Designer permitiu desenvolver a aplicação *Web* integralmente em Java, tendo sido necessário apenas criar um documento HTML.

A solução foi integralmente implementada usando tecnologias *open source* ou de utilização gratuita. Contudo, a disponibilização da solução ao mercado requer a negociação das condições da utilização dos serviços do Google App Engine porque a quota e a capacidade de processamento gratuitas serão insuficientes.

Bibliografia

[1] The European Parliament and the Council of the European Union, "Directive 2006/32/Ec Of The European Parliament," 2006. [citado na p. 1]

[2] Ministério da Economia, da Inovação e do Desenvolvimento, "Decreto-Lei n.º 34/2011," 2011. [citado na p. 2]

[3] SmartRegions, "European Smart Metering Landscape Report," Tech. Rep., 2011. [citado na p. v, 5, 6]

[4] HARGREAVESN, Tom, NYE, Michael and BURGESS, Jacquelin, "Making Energy Visible: A Qualitative Field Study Of How Householders Interact With feedback from smart energy monitors," *Energy Policy 38*, 2010. [citado na p. 6]

[5] Ecore, *EcoreAction Brochure*. [Online]. Available: http://www.ecore.fi/temp_eng/esitteet.php [citado na p. v, 6, 7]

[6] Itron. ACE Vision. [Online]. Available: http://www.acevisionservice.com:82/ [citado na p. v, 7]

[7] EirGrid, *EirGrid Launches SmartGrid iPhone Application*, 2010. [Online]. Available: http://www.eirgrid.com/search/?q=iphone&site=eirgrid [citado na p. 8]

[8] KOLLE. Help pages for MeterReading. [Online]. Available: http://kolle.mobi/apps/meterReading/help/help.html [citado na p. 8]

[9] FARLEX. Standalone Definition Of Standalone In The Free Online Encyclopedia. [Online]. Available: http://encyclopedia2.thefreedictionary.com/standalone [citado na p. 8]

[10] TANENBAUM, Andrew and STEEN, Maarten, *Distributed Systems Principles And Paradigms*. Pearson Prentice Hall, 2007. [citado na p. 8]

[11] DLMS User Association, *Architecture and Protocols*, 2009. [citado na p. v, 11, 12]

[12] DLMS User Association. List of standard OBIS codes and COSEM objects. [Online]. Available: http://www.dlms.com/documentation/listofstandardobiscodesandmaintenanceproces/index.html [citado na p. 11]

[13] Ricardo Pinto. Contador de electricidade - Museu da Electricidade. [Online]. Available: http://wikienergia.com/~edp/index.php?title=Contador_de_electricidade [citado na p. 13]

[14] Amanda Smith, *ACE2000 type 290 Technical and User Guide*, Itron, 2004. [citado na p. v, 13]

[15] Actaris Sistemas de Medição, Lda, *P30 15(60) O Contador Trifásico de Energia Eléctrica*, Actaris, 2003. [citado na p. v, 13]

[16] Itron Sistemas de Medição, *ACE SL7000 Folheto Técnico*, Itron, 2006. [citado na p. v, 14]

[17] Atmel. In-Home Display Units. [Online]. Available: http://www.atmel.com/applications/metering/in-home-display-units/default.aspx?tab=devices [citado na p. 15]

[18] SOMERA, Guilherme, *Treinamento Profissional em Java.* Digerati Books, 2006. [citado na p. 17]

[19] MAY, Wolfgang, ALFERES, José, AMADOR, Ricardo, "Active Rules in the Semantic Web: Dealing with Language Heterogeneity," *Rules and Rule Markup Languages for the Semantic Web: First International Conference, RuleML 2005, Galway, Ireland, November 10-12, 2005, Proceedings.* [citado na p. 18]

[20] DEITEL, Harvey, *XML How To Program.* Pearson Education, Inc. [citado na p. 18, 19]

[21] RAGGETT,Dave, LAM, Jenny, ALEXANDER, Ian, KMIEC, Michael, *Raggett on HTML 4.* Addison-Wesley, 1998. [citado na p. 18]

[22] FREEMAN, Eric, ROBSON, Elisabeth, *Head First HTML5 Programming: Building Web Apps with JavaScript.* O'Reilly Media, Inc. [citado na p. 18]

[23] MONSON-HAEFEL, Richard, *J2Ee Web Services.* Addison-Wesley Professional, 2004. [citado na p. v, 19, 20, 21]

[24] GARRETT, Jesse. Ajax: A New Approach to Web Applications. [Online]. Available: http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications [citado na p. v, 25, 26]

[25] Stat Counter. StatCounter Global Stats - Browser, OS, Search Engine including Mobile Market Share. [Online]. Available: http://gs.statcounter.com [citado na p. v, 26, 27]

[26] Android Developers. Platform Versions. [Online]. Available: http://developer.android.com/about/dashboards/index.html [citado na p. v, ix, 28]

[27] Android Developers. Activities. [Online]. Available: http://developer.android.com/guide/components/activities.html [citado na p. v, 29, 30]

[28] HAUSTEIN, Stefan, SEIGEL, James. Package org.ksoap2. [Online]. Available: http://ksoap2.sourceforge.net/doc/api/ [citado na p. 33]

[29] Google developers. Getting Started with Chart Tools. [Online]. Available: http://code.google.com/p/gwt-google-apis/wiki/VisualizationGettingStarted [citado na p. 40]

[30] RUSITSCHKA, Sebnem, EGER, Kolja GERDES, Christoph, "Smart Grid Data Cloud: A Model for Utilizing Cloud Computing in the Smart Grid Domain," 2010. [citado na p. 49]

[31] LUTUS, Paul. Polynomial Regression. [Online]. Available: http://www.arachnoid.com/sage/polynomial.html [citado na p. 71]