Projeto Integrado

BILL MATE

Engenharia de Aplicações 13/14

A PARTILHA DE DESPESAS NUNCA FOI TÃO FÁCIL

Universidade do Minho

BRAGA, 2013–2014

BILLMATE
Partilha de Despesas

André Santos PG25329
Universidade do Minho

Pedro Leite PG25330
Universidade do Minho

Francisco Neves PG25334
Universidade do Minho

Ricardo Branco PG25339
Universidade do Minho

Universidade do Minho, Departamento de Informática

RESUMO

Os projetos devem ser realizados com base em ideias novas, e não se devem focar em reinventar a roda. Foi este objetivo que moveu o grupo, e que desde o inicio pensou em realizar algo diferente e que fizesse a diferença no dia-a-dia do público-alvo.

Desde cedo surgiram várias ideias, e todas elas tinham bastante potencial, porém, o grupo queria algo que fosse útil para uso próprio e para aquelas que nos rodeiam. Assim, idealizou-se uma aplicação que pudesse ser realizada em contexto académico mas que no final da sua implementação pudesse ficar online para todos utilizarem. Com isto, pensou-se sobre uma aplicação que ajudasse na divisão de despesas entre um grupo de pessoas.

Dito isto, iniciou-se uma pesquisa que permitiu concluir que já existem algumas com o mesmo objetivo, mas, constatou-se que todas elas tentam ser muito genéricas, perdendo alguma abstração aquando da sua integração com diferentes tipos de utilizadores, e no caso dos estudantes, o grupo verificou que as existentes não abordam conceitos como ”pagamento da eletricidade”, ou um simples ”pagamento da internet”.

Com base nesta investigação, focou-se o público-alvo para estudantes, e inseriram-se todos os conceitos importantes para uma gestão organizada das despesas dos mesmos.

Área de aplicação: Desenvolvimento de aplicação para suportar o processo da partilha de despesas entre estudantes. Implementação de infra-estrutura aplicacional, avaliação e desempenho e escalabilidade.

Palavras-chave: Tomcat, Grails, Spring, WebService, CentOS, LVS, DRBD, NFS, ISCSI, MPATH, REDIS.

RESUMO DE CONTEÚDOS

Conteúdo

LISTA DE FIGURAS

LISTA DE TABELAS

PREFÁCIO

PEDRO LEITE

Braga, Portugal
Abril, 2014

ANDRÉ SANTOS

Braga, Portugal
Abril, 2014

FRANCISCO NEVES

Braga, Portugal
Abril, 2014

RICARDO BRANCO

Braga, Portugal
Abril, 2014

ACRÓNIMOS

DRBD	Distributed Replicated Block Device
HTML	Hypertext Mark Language
ICD	Infraestrutura de Centros de Dados
IDE	Integrated Development Environment
ISCSI	Internet Small Computer System Interface
LVS	Linux Virtual Server
MPATH	Multipath
NFS	Network File System
SO	Sistema Operativo
UCE	Unidade Curricular de Especialização
VM	Virtual Machine

GLOSSÁRIO

Android	É um sistema operativo baseado no núcleo do Linux para dispositivos móveis, desenvolvido pela *Open Handset Alliance* que é liderada pela *Google*.
Git	Git é um sistema de controlo de versões distribuído e um sistema de gestão de código fonte, com ênfase na velocidade. Começou por ser projetado e desenvolvido para o desenvolvimento do kernel Linux, mas foi adotado por muitos outros projetos, porque este permite que cada diretório de trabalho do Git seja um repositório com um histórico completo e habilidade total de acompanhamento das revisões, não dependente de acesso a uma rede ou a um servidor central.
GitHub	É um serviço *web hosting* para o desenvolvimento de projetos de software que usa o sistema de controlo de versões do Git. Este permite que todos trabalhem em um repositório único criando vários ”ramos”, que mais tarde serão unidos após a aceitação de todos os colaboradores desse repositório.
Grails	*Framework* para desenvolvimento de aplicações para web, utilizando a linguagem Groovy. Permite criar uma *Framework* de alta produtividade em *JAVA*. Segue o paradigma da programação por convenção que torna os detalhes de configuração transparentes para o programador.
Groovy	Linguagem de programação orientada aos objetos, similar ao *JAVA*, que se apresenta como alternativa. Apesar desta semelhança, possui características de *Python* e *Ruby*. É compilada em *bytecode*, integrando-se facilmente com outras bibliotecas em *JAVA*.

IntelliJ	*JAVA IDE* que pertence à *JetBrains*, disponível com a edição de comunidade e uma edição comercial.
Trello	É uma aplicação *web-based* para a gestão de projetos que utiliza o paradigma conhecido por *kanban*. Os projetos são representados por *boards*, e cada uma tem várias listas, que podem ser entendidas como listas de tarefas. Cada lista contém vários *cards*, em que cada uma corresponde a uma tarefa, e representam o fluxo que a lista de tarefas terá. Deste modo, os utilizadores podem ser associados aos *cards*. Uma vez que cada *board* corresponde a um projeto, é possível criar uma *organization* que será o agrupamento de *boards*.

Secção 1

INTRODUÇÃO

O trabalho que se encontra documentado neste relatório é relativo ao Projeto Integrado enquadrado na UCE de Engenharia de Aplicações, sendo que o objetivo deste projeto é conseguir utilizar os conceitos abordados nos quatro módulos desta UCE para se conseguir construir uma plataforma que suporte a aplicação com alta disponibilidade e com segurança.

Com base neste objetivo, delineou-se o desenvolvimento de uma aplicação de despesas partilhadas entre estudantes. Esta ideia surge porque, a maioria dos jovens começa a ter mais necessidades de organização aquando da sua chegada à universidade, que é quando normalmente sai da casa dos pais, e precisa de começar a pagar as suas obrigações, sejam elas de eletricidade, de gás, entre outras. Com base nisto, é certo que por várias vezes, as dívidas caem em esquecimento, e muitas são as histórias de terem ficado sem eletricidade, sem gás ou sem internet por falta de pagamento.

O nome *BillMate* é uma concatenação de *Bill* com *Mate*, que significa exatamente aquilo que esta aplicação faz, que é a partilha de despesas entre colegas. Esta é uma excelente ferramenta para os estudantes, porque é feita por estudantes, que sabem exatamente as suas necessidades. É certo que não há melhor cliente do que o próprio cliente, uma vez que conhece todas as suas necessidades. Além disso, a facilidade de confrontar o público-alvo com esta aplicação, irá permitir que esta seja mais cómoda para o utilizador final.

O trabalho que se documenta neste relatório, descreve todo o processo de criação de uma aplicação que suporta a divisão de despesas de estudantes, ajudando a fazer a divisão entre os diversos grupos em que se encontra, para que este saiba em qualquer momento, as despesas que já estão pagas, as datas limite de pagamento, as próximas despesas, as

pessoas que já pagaram, entre vários outros componentes que se referirão nos capítulos posteriores.

Os capítulos encontram-se organizados estrategicamente, apresentando-se Inicialmente uma motivação para a realização deste projeto, de modo que o utilizador final perceba quais os problemas que esta aplicação vem resolver. Esta conclusão do utilizador final terá mais ênfase quando confrontado com os casos de estudo que aparecerão posteriormente. Após estes, serão verificados os requisitos e o planeamento tomado em consideração. Com isto, entrar-se-à em detalhes mais técnicos, onde será descrita toda a infraestrutura que suportará a aplicação, passando pela modelação, interface e implementação da codificação do sistema. Terminando todo este processo serão apresentados os resultados de desempenho da aplicação na infraestrutura desenvolvida.

Secção 2

MOTIVAÇÃO

Todas as pessoas necessitam de organizar os seus débitos e os seus ativos, porém, por lapso ou por falta de tempo, nem sempre se lembram de todas as suas obrigações. No caso dos estudantes universitários, isto ainda é um fator mais agravante, porque estes começam a sentir esta necessidade quando vão para a universidade, porque a maior parte destes, saem da casa dos pais nesta altura, o que implica que iniciam novas responsabilidades que até este momento não tinham.

Muitas são as histórias de falta de eletricidade, falta de gás, ou até mesmo de falta de internet por atraso no pagamento. Esta aplicação tende a responder a estas necessidades. Estes problemas são contornados devido aos alertas e notificações que são enviados ao utilizador para que este não se esqueça das suas obrigações.

A grande vantagem desta aplicação, deve-se ao facto de ser elaborada por estudantes, e destinar-se a estudantes, pois, quem está a fazer a aplicação sabe as necessidades para este contexto. Não há melhor cliente do que quem elabora o produto, porque sabe todos os seus requisitos. O feedback demonstrado pelo público-alvo tem sido bastante positivo, demonstrando bastante interesse pelo resultado final desta aplicação.

Esta aplicação é bastante ambiciosa, porque a ambição do grupo também é bastante relevante, uma vez que no final deste trabalho, pretende-se que esta aplicação não fique ”apenas no papel”, e que saia do contexto académico, ficando online e disponível para todas as pessoas.

Secção 3

CASOS DE ESTUDO

De seguida serão apresentados alguns casos de estudo, nos quais se verifica o comportamento em algumas das situações que o grupo identificou. Apesar de apenas se apresentarem três casos de estudo, é fácil de compreender, que muitos outros cenários se enquandram nos apresentados.

3.1 Caso 1 - Habitação

O Pedro, o André, o Francisco e o Ricardo são estudantes universitários e vivem todos na mesma habitação. Para uma melhor organização das despesas utilizam uma aplicação para fazer esta gestão. O Pedro criou uma habitação e associou os 4 amigos à mesma. Nesta habitação criou uma despesa regular de eletricidade para o dia 23 de Julho, que se repete mensalmente. No dia 23 de Julho, o Pedro recebe o aviso da despesa regular e cria a despesa da eletricidade associando a imagem da fatura à despesa.

Os restantes elementos da casa recebem o aviso indicando o valor monetário que devem ao Pedro, e efetuam o pagamento. Todos fazem o pagamento, e após o Pedro confirmar o pagamento de todos os elementos, a despesa fica resolvida. Por uma questão de organização, o Pedro decide adicionar ainda o comprovativo do recibo para que todos possam consultar e fazer download dos documentos.

3.2 Caso 2 - Boleias

O André, o Francisco e o Ricardo vivem na mesma área de residência e partilham o mesmo carro para irem para a universidade. Além deles, ainda costumam dar boleia ao Pedro, mas o Pedro mora bastante mais perto da universidade, pelo que ficou decidido que o Pedro iria pagar apenas 10% da despesa.

O André, que é o condutor, criou um grupo para boleias e associou-os aos 4 a esse grupo. De seguida colocou o valor da despesa total, e definiu as percentagens de pagamento. Para o Pedro apenas colocou 10% e para os restantes dividiu de forma equitativa (30%).

Todos fizeram o pagamento e confirmaram o mesmo. O André após receber o dinheiro também confirmou a receção do mesmo. Uma vez que tanto o credor como o devedor confirmaram, então a dívida é confirmada e encerrada.

No final, a despesa da boleia é confirmada e encerrada, uma vez que já todos pagaram o pagameto.

3.3 Caso 3 - Jantar

O Pedro, o André, o Francisco e o Ricardo foram jantar a seguir às aulas, mas quem pagou toda a despesa foi o Pedro. Com o objetivo de controlar quem já pagou e quem ainda falta pagar, o Pedro criou um grupo e associou os amigos à mesma, e dividiu a despesa pelos seus amigos.

Os seus amigos recebem a notificação, procedem ao pagamento e confirmam o pagamento. De seguida o Pedro confirma que recebeu a quantia monetária. Mas, por lapso, o André esqueceu-se de pagar a despesa, e como o Pedro não definiu um limite de pagamento, o André não recebeu nenhuma notificação a lembrar da sua dívida.

O Pedro, ao analisar o seu histórico e e sua lista de débitos, verificou que tinha uma quantia ainda por receber. Assim, o Pedro relembra o André da sua dívida, e este por sua vez, procede ao pagamento ao Pedro e confirma o mesmo. Da mesma forma, o Pedro confirma que recebeu a quantia do André, e a despesa criada fica liquidada.

Uma vez que o Pedro tem todas as despesas liquidadas, ele gera um relatório detalhado para verificar as suas transações e analisar os seus gastos.

Secção 4

PLANEAMENTO

A fase de planeamento é crucial para a boa execução de qualquer projeto. Desde cedo que o grupo definiu estratégias para o bom entendimento e para seguir um fluxo de trabalho comum.

O planeamento passou por um estudo de mercado, gestão do projeto de acordo com ferramentas a utilizar e as respetivas plataformas. Nas seguintes seguintes, será descrito com mais pormenor cada um dos pontos referidos.

4.1 Estudo do mercado

Ao longo do processo de decisão da área de negócio sobre a qual o grupo de trabalho iria incidir, surgiram várias ideias, que foram descartadas porque existiam várias alternativas que eram muito boas, nas quais o acréscimo de funcionalidades não seria a melhor opção porque não iria abranger o público-alvo que era objetivo.

Com isto, idealizou-se o projeto de partilha de despesas, mas ao se verificarem várias fontes de dados, as possibilidades com as *keywords* "split"e "bill", retornavam muitos resultados, contudo, em nenhum destes se verificaram possibilidades que tinham como público-alvo os estudantes, aparecendo todos como genéricos, o que como é óbvio, não se centraliza nas necessidades que estes têm. Com base nisto, não se encontraram concorrentes diretos a esta área de negócio.

A falta de concorrência não permite dar o conhecimento sobre os requisitos operacionais que um projeto desta dimensão terá, porque através destes, é possível ter conhecimentos sobre:

- Avaliação do crescimento do mercado
- Analisar o comportamento e as tendências do setor.
- Reconhecer estratégias vencedoras.
- Identificar os concorrentes e o seu valor.

Apesar desta falta de resultados, a área de negócio na qual o projeto se incide, terá uma boa aceitação pelos utilizadores, uma vez que o grupo sabe quais as necessidades dos estudantes, porque todos os elementos do grupo são estudantes e sabem que esta aplicação vai ser bastante útil.

Uma estratégia levada em consideração pelo grupo, é não introduzir qualquer custo ao utilizador. Com isto, espera-se que estes experimentem a aplicação apenas por curiosidade. Com base nesta experimentação, é necessário mostrar ao utilizador as vantagens induzidas por esta perante as suas concorrentes. Sabendo que esta tem como público-alvo os estudantes, é preciso que esta seja simplista, de modo a cativar a atenção dos mesmos.

O comportamento natural dos estudantes é seguir as tendências, e se o utilizador que experimentar gostar da aplicação, então o número de estudantes a usufruir da mesma irá crescer. Este crescimento irá induzir uma maior divulgação que permitirá aumentar a tendência da sua utilização.

Durante este estudo de mercado, verificaram-se dois cenários, sendo que o primeiro refere-se às aplicações concorrentes, nas quais existe uma lógica de negócio disponibilizada para qualquer pessoa que utiliza a aplicação, e o segundo cenário refere-se às alternativas, que passam por técnicas primitivas, tal como, documentos de texto. Nas duas subsecções seguintes apresentam as diferentes opções com mais detalhe.

4.1.1 Aplicações Concorrentes

Todas as aplicações, mesmo que tenham o mesmo tema, são sempre realizadas de forma diferente. Variam essencialmente, nas suas funcionalidades, nos seus algoritmos, na sua simplicidade, e obviamente na interface gráfica.

No contexto do projeto em desenvolvimento, tal como já foi referido, a maioria das aplicações existentes tentam ser genéricas para poderem obter um maior número de utilizadores, e com isto reparou-se que estas falham em alguns aspetos, tal como a abstração que se tem perante um tema. A título de exemplo, para um casal que habita numa casa própria, uma despesa de eletricidade pode ter uma semântica, mas para um estudante que partilha a casa com mais colegas terá outro significado.

O aumento da abstração dos conceitos é o principal problema que as aplicações concorrentes têm, porque para se conseguir abranger todos os tipos de grupos, é necessário ter conhecimento de todos os estilos de pessoas que irão utilizar a aplicação, e isso será muito complicado.

Deste modo, serão apresentadas algumas aplicações concorrentes, que apesar de não serem totalmente orientadas para os estudantes, permitem que possam ser usadas pelos mesmos.

Split Wise

1. Permite criar grupos de habitação e de viagens.

2. Em qualquer dos grupos, é possível criar despesas que são associadas a amigos. Estas são divididades forma equitativa por todos, contudo, pode-se mudar para pôr exatamente as quantias ou dividir por percentagens.
3. Triangular as despesas
4. Verificar gráficos
5. Realizar comentários nas despesas.
6. Login com o google+
7. Aplicação móvel para android e ios
8. Envio de lembretes de pagamento aos amigos.
9. Definir prazos de pagamento.

Split a Bill

1. Criar nova despesa fatura e criar várias despesas para essa fatura.
2. Aplicação móvel para ios.
3. Tem conversações para a fatura.
4. Partilhar a despesa com vários amigos mas apenas de forma equitativa.
5. Envio de lembretes de pagamento aos amigos.

We Split

1. Criar um grupo
2. Dentro do grupo criar fatura ou despesa
3. Dividir despesa de forma automática (equitativa) ou manual.
4. Adicionar notas à despesa.
5. Aplicação móvel para android.

Bill Pin

1. Aplicação móvel para ios e android.
2. Apenas cria despesas sem agrupar por grupos ou faturas.
3. Adicionar despesa e associar amigos.
4. Gerar balanço de contas para email.

Bills Up

1. Aplicação móvel para ios e android e windows phone.

2. Criar grupos e associar amigos.

3. Triangular.

4. Criar Despesa e associar amigos.

5. Importar e Exportar dados.

4.2 Alternativas

De seguida serão apresentadas algumas soluções que apesar de não serem exatamente iguais à que se está a representar neste documento, apresentam algumas características comuns. Assim, serão apresentadas algumas vantagens e desvantagens.

4.2.1 Documento de Texto

Esta forma de divisão de contas, é o mais básico que pode ser realizado, contudo é muito desorganizado e certamente centralizado. Assim, o utilizador regista de forma textual todas as suas dívidas, assim como todas as quantias que têm a receber. Esta até pode ser uma boa solução, se a cardinalidade dos registos for baixa, contudo, quando o volume de dados aumento, a complexidade na gestão da informação também será muito maior.

Além disto, podem haver muitos problemas com valores inconsistentes, porque como não há sincronização entre os utilizadores, estes podem colocar valores distintos nos seus registos.

Importa ainda salientar, que com esta abordagem, toda a consulta e gestão de dívida é muito mais complicada, porque o processo não é automático.

Este formato é completamente desaconselhado, mas acredita-se que ainda há muitos grupos de pessoas que preferem utilizar este tipo de organização.

4.2.2 Folhas de Cálculo

Contrariamente à aboragem apresentada na subsecção anterior, a utilização de folhas de cálculo (ficheiros Excel, etc) é uma melhor alternativa. Mas, o problema da sincronização continua presente. É certo que com as tecnologias habituais, já é possível sincronizar os mesmos, porém o processo de partilha era mais complicado, uma vez que seria necessário partilhar o documento apenas com as pessoas que lhe dizem respeito, tendo no limite, um ficheiro para cada evento.

Além disso, a consulta de valores necessários para a tomada de decisão é mais delicada, sendo necessário analisar todos os ficheiros e agregar apenas os valores oportunos.

A vantagem da utilização desta abordagem deve-se ao facto de que cada utilizador cria o ficheiro de acordo com as suas necessidades, ajustando-o aos seus gostos. Mas, é óbvio que este requer algum tempo para ser construído, obrigando a que o utilizador tenha algum conhecimento para que as possa ajustar às suas necessidades. Além disso, é certo que estas estarão mais suscetíveis a erros.

4.3 Gestão do Projeto

Para a melhor organização do grupo, foram delineadas algumas estratégias para que o trabalho individual possa surtir efeito, sem serem realizadas tarefas redundantes. Apesar do trabalho ser sempre realizado em grupo, no final de cada semana é realizada uma reunião para delinear objetivos e definir prioridades. Serão ainda definidas as *deadlines* para as tarefas.

Deste modo, serão realizadas regularmente reuniões de grupo, para que seja feito o ponto de situação, e além disso serão definidas *deadlines* para as tarefas. A atribuição de prazos tem por base o auxílio do diagrama de gantt que foi previamente realizado.

Apesar do trabalho ser feito diariamente de forma presencial com todo o grupo, utilizaram-se ferramentas que auxiliaram a organização do trabalho. Ferramentas como o *Trello* e o *Git* foram indispensáveis para a boa execução do projeto. O *Git* é uma ferramenta de controlo de versões muito poderosa, que permite que cada utilizador trabalhe de forma independente, e que o *merge* do trabalho seja feito de forma quase transparente. O desenvolvimento seguiu o paradigma *feature branch workflow*, que indica que cada nova *feature* seja realizada em *branches* independentes. No final de cada *feature* estar terminada, é feito um *pull request*, com as alterações realizadas. Este é verificado por todos os elementos do grupo, e só quando todos aceitarem, é que se pode propagar estas alterações no repositório principal.

Um ponto fundamental para o sucesso de qualquer aplicação, centra-se na escolha das ferramentas a utilizar. Neste caso existem muitas escolhas que devem ser bem ponderadas, porque os recursos necessários são elevados.

No começo do planeamento de um projeto, é preciso estruturar e modelar o problema. Assim, para a sua modelação optou-se pelo visual paradigm, porque é uma das melhores ferramentas de modelação existentes no mercado, e permite que após a modelação seja possível gerar a base de dados e o código de uma forma muito rápida. Além disto, é preciso ainda criar-se os mockups que serão desenvolvidos em *Balsamiq Mockups*.

Durante a fase de desenvolvimento será utilizado o IntelliJ como IDE, porque é uma ferramenta que tem muita potencialidade para um projeto desta dimensão, e possui a vantagem de permitir a integração com o visual paradigm, permitindo realizar várias operações de modelação diretamente no IDE.

Antes de começar a desenvolver o código para a aplicação é preciso construir uma infraestrutura que suporte a mesma. Assim, para a fase de desenvolvimento, esta será criada nos computadores pessoais utilizando a vmware com máquinas virtuais de CentOS. Ainda na fase de implementação da mesma, é preciso instalar vários serviços, contudo, um serviço fundamental será o motor de base de dados. Para esta optou-se pela utilização do PostgreSQL.

Relativamente à estrutura da aplicação, esta será realizada de acordo com um modelo em 3 camadas (camada de dados, de negócio e de apresentação), sendo que para cada uma destas será utilizada uma framework. De acordo com um estudo realizado anteriormente pelo grupo, a melhor framework para a camada de dados é o Hibernate, para a camada de negócio é o Spring Framework e para a camada de apresentação é o Grails.

4.4 Plataformas

Durante o planeamento do projeto ficou delineado que o objetivo será desenvolver uma aplicação que estará disponível em suporte web e em suporte mobile.

Sabe-se que o crescimento das pessoas que navegam diariamente na internet através de um computador está a estagnar, contudo, o crescimento na navegação através de dispositivos móveis está a crescer como nunca. Principalmente desde o ano de 2010, o paradigma tem-se alterado de web para mobile. Com isto, não se quer dizer que o desenvolvimento *web* já esteja estagnado, pelo contrário, mas cada vez mais é necessário desenvolver sites *responsive*, para que estes se ajustem às pequenas telas dos dispositivos móveis.

Para a escolha foram necessários alguns fatores de decisão, tais como:

Funcionalidades do dispositivo
Apesar das aplicações *web* poderem usufruir das funcionalidades dos dispositivos, não o conseguem da mesma forma que as aplicações nativas, uma vez que estas são realizadas especificamente para o dispositivo que a está a executar.

Funcionamento offline
Apesar da funcionalidade *offline* estar presente como um objetivo futuro do grupo, é certo que para se conseguir isso, é mais fácil com as aplicações nativas, porque além do *browser* possuír uma cache, ainda é limitada comparativamente ao nativo.

Descoberta
Como se pretende que esta seja divulgada, é certo que as aplicações *web* são encontradas de forma mais fácil, porque esta está constantemente na *web*, e quando se pesquisa uma determinada *keyword*, é muito provável que apareça nos resultados da pesquisa.

Velocidade
A velocidade é um fator crucial para esta aplicação, e neste caso as aplicações nativas são mais rápidas por terem acesso direto ao sistema operativo e além disso são programadas na linguagem nativa do dispositivo.

Instalação
Alguns utilizadores não gostam de ter de instalar aplicações para usufruir das funcionalidades, deste modo a escolha do grupo recai sobre a construção da aplicação *web*, contudo, tal como já foi referido anteriormente atras, com esta não é possível obter o mesmo nível da *User Experience* que o obtido com as aplicações nativas.

Manutenção
Devido a atualizações, é necessário que a aplicação sofra manutenção, deste modo, opta-se pelas aplicações *web* pois podem ser atualizados com a frequência necessária como se fosse uma página da internet. As aplicações nativas têm mais problemas neste sentido, porque é necessário que o utilizador esteja atento às atualizações que vão surgindo.

Cross-Platform
Como é óbvio, a aplicação *web* apenas necessita de um *browser* para poder executar, enquanto que as aplicações nativas são feitas exclusivamente para um sistema operativo.

Interface ao utilizador
Como se pretende aumentar a *User Experience*, a aplicação nativa consegue tirar mais proveito dos componentes do dispositivo. Mas, isto não significa que não se consiga bons resultados com uma aplicação *web*, mas como foi dito, a usabilidade não será a mesma

Uma vez que todos os fatores retratados anteriormente foram importantes para a escolha do tipo da aplicação, é fácil de perceber que neste caso não há um tipo que contenha todas estas características. Assim, para o grupo conseguir atingir os objetivos para esta aplicação, optou por desenvolver uma aplicação com um conteúdo *web* e um conteúdo *mobile*, sendo que numa primeira instância apenas terá suporte em *Android e iOS*.

Secção 5

ANÁLISE DE REQUISITOS

5.1 Entidades

Após uma análise aos objetivos do sistema *BillMate*, definiram-se como entidades, as que se encontram na lista seguinte. Essas entidades que são listadas são as candidatas a serem classes aquando da modelação do sistema.

- Acção - Corresponde ao histórico de operações. É gerada uma acção quando é executada qualquer operação no sistema. Todas as notificações serão geradas através das acções.
- Casa - Tipo de grupo que o utilizador pode criar e tem características específicas para uma casa.
- Coletivo - Tipo de grupo que o utilizador pode criar e tem características mais genéricas que podem ser usadas para outros tipos de círculos.
- Tipo de despesa - Define o tipo de despesa que vai ser utilizado para as despesas.
- Dívida - Indica o valor que cada utilizador deve em cada despesa.
- Débito Direto - Entidade que permite criar débitos diretos, para tornar o processo de criação de despesas automático.
- Despesa - Identifica as despesas que os utilizadores têm para os vários grupos

- Pagamento - Entidade que representa o pagamento de um utilizador para uma determinada despesa.
- Utilizador referenciado - Utilizador que não está registado mas que pode ser utilizado no processo de divisão de despesas.
- Utilizador registado - Utilizador que está devidamente registado no sistema.
- Despesa regular - Despesa que é regular de acordo com uma periodicidade. Permite relembrar o utilizador que tem de criar a despesa e permite que seja criada mais rapidamente.
- Subscrição - Utilizador subscreve-se para receber novidades do sistema.
- Notificação do sistema - Aviso que é apresentado ao utilizador sobre acções que lhe estão associadas.

5.2 Funcionalidades

De modo a ser possível começar a modelar o problema, é preciso que seja realizada uma análise às funcionalidades que o sistema terá. De seguida apresentam-se as que foram identificadas aquando do levantamento e análise de requisitos.

- Login
- Registo
- Criar casa
- Criar grupo
- Criar despesa
- Criar despesa regular
- Criar tipo de despesa
- Alterar definições do Utilizador
- Receber notificações
- Marcar notificações como lidas
- Editar despesa
- Editar despesa regular
- Editar casa
- Editar grupo
- Adicionar/Remover pessoas à casa
- Adicionar/Remover pessoas à despesa
- Adicionar/Remover pessoas à despesa regular

- Consultar e filtrar relatórios
- Listar despesas
- Listar despesas regulares
- Efetuar pagamentos
- Confirmar pagamentos

5.3 Pressupostos

Durante a fase do levantamento e análise de requisitos, é preciso começar a tirar conclusões e a resolver problemas que vão sendo identificados. Nesta fase começa a ser necessário haver pressupostos, que irão influenciar a modelação do sistema.

- Um grupo não tem um responsável, e todos os que estiverem nesse grupo podem desempenhar funções de administrador.
- O utilizador em sessão pode criar uma despesa sem se incluir a ele próprio.
- O aviso de notificação aparece 5 dias antes da data de pagamento.
- Ao criar um grupo é possível adicionar utilizadores referenciados à mesma. O nome do referenciado é o prefixo do email que tem de ser inserido.
- Só é possível adicionar utilizadores à despesa regular, caso eles estejam no círculo.
- O utilizador pode fechar a despesa caso ainda não hajam pagamentos.
- O utilizador só pode sair do grupo ou da despesa caso tenha as suas dívidas resolvidas.
- O utilizador só pode fechar o grupo, caso todas as dívidas estejam resolvidas.
- Se a despesa tiver débito direto, cria automaticamente a despesa e envia as notificações aos colegas da casa, sem o responsável da despesa ter qualquer trabalho.

Secção 6

INFRAESTRUTURA

6.1 Requisitos

6.2 Componentes

6.3 Implementação

Secção 7

MODELAÇÃO

Após o levantamento e a análise de requisitos, o processo de modelação torna-se mais simplificado. Obviamente que não é um processo simples, mas já se conhecem os requisitos que devem ser tomados em consideração na modelação.

Nas próximas subsecções, irão ser apresentados três diagramas que o grupo considerou ser mais relevantes para a realização do projeto. Inicialmente elaborou-se o diagrama de use case, onde se descreve as funcionalidades que são precisas. Após se ter o conhecimento das funcionalidades, definiu-se o relacionamento presente entre as várias entidades do sistema. Por fim, indicam-se as classes que são representadas no sistema.

De seguida apresenta-se de forma mais detalhada a modelação de cada um dos diagramas.

7.1 Diagrama de Use Case

O diagrama de *use case* é feito para se perceber as funcionalidades do sistema. Este, representa a interação entre um utilizador e o sistema. Com a elaboração destes, conseguiu-se perceber a unidade de um trabalho significante. Cada caso dos que serão apresentados na imagem seguinte, descreve a funcionalidade que irá ser construída no sistema proposto.

Importa salientar que com este diagrama, não se pretende que definir como o software deverá ser construído, mas sim como se deve comportar quando estiver pronto.

O desenvolvimento de um *software* é algo bastante complexo, e o desenvolvimento dos diagramas de *use case* descrevem uma ”fatia”do que o *software* deverá oferecer.

Estes, serão também os mais indicados para o cliente final visualizar, porque são construídos com linguagem natural, facilmente percetível por qualquer pessoa.

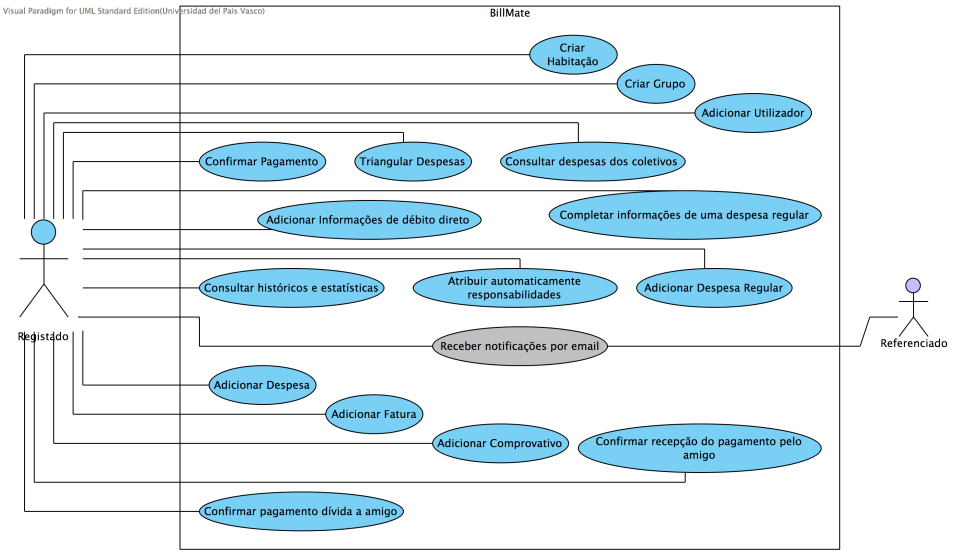

Figura 7.1 Diagrama de use case

7.2 Diagrama de Modelo de Domínio

Tal como o próprio nome indica, domínio, é utilizado para denotar áreas funcionais dentro de sistemas que exibem funcionalidades similares. Este diagrama pode ser interpretado como sendo uma coleção de componentes de software que partilham um determinado conjunto de características.

O objetivo desta análise deve-se ao facto de se puder analisar a informação que é identificada, capturada e organizada, para que se possa reutilizar na interação entre os domínios. É certo que esta reutilização está a ser vista a um nível de abstração muito elevado, uma vez que neste momento apenas se está a analisar o domínio, mas é útil aquando da construção do diagrama de classes. Apesar de não ser este o objetivo, esta modelação será útil se for necessário que as funcionalidades sejam reutilizadas para múltiplos sistemas.

Tal como se pode analisar pela imagem 7.2, um utilizador é uma das entidades principais do sistema, uma vez que é este que despoleta as ações. Este utilizador pode ser classificado como registado ou referenciado. Este distinção deve-se ao facto de um utilizador não ser registado e puder ser utilizado na aplicação para partilhar despesas. Cada utilizador encontra-se em um ou vários círculos, sendo que um círculo pode ser classificado como um tipo específico (casa), e um tipo mais genérico (colectivos). Um determinado utilizador que se encontra em um determinado círculo tem despesas, que são partilhadas com os restantes utilizadores daquele mesmo círculo. Definiu-se o tipo despesa no modelo de domínio, de modo a agrupar as despesas por categorias, sendo elas por exemplo, de eletricidade, de gás, entre outras, que podem até ser personalizadas. As despesas regulares são criadas

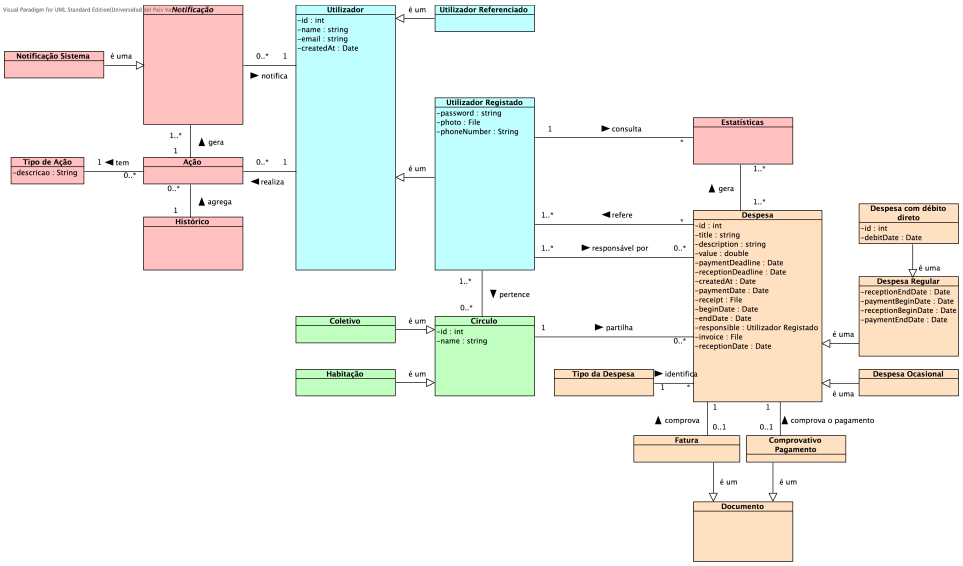

Figura 7.2 Diagrama do modelo de domínio

para alertar os utilizadores quando se aproxima a data de receção da fatura. Esta data, terá de ser obviamente definida pelo utilizador, que além desta data define a periodicidade com que esta se repete, normalmente mensal, mas é personalizável. Cada despesa pode ter uma fatura e um recibo, assim como um débito direto. Todas as ações que são feitas pelos utilizadores, geram notificações para darem feedback constante ao utilizador.

7.3 Diagrama de Classes

O diagrama de classes é uma das peças de modelação que mais fazem sentido utilizar, porque é uma representação da estrutura e das relações das classes que servem de modelo para objetos.

Estes definem todas as classes que o sistema necessita de ter e é a base para a construção dos restantes diagramas de comunicação, sequência e de estados. Com a utilização deste diagrama é possível visualizar a representação da estrutura do sistema recorrendo ao conceito de classes e relações. Este modelo resulta de um processo de seleção onde são identificados os objetos relevantes do sistema em estudo e que se pretende descrever no seu ambiente.

Optou-se pela utilização deste diagrama, pois deste modo consegue-se visualizar como cada classe se relaciona com as restantes, tendo como objetivo a satisfação dos requisitos funcionais definidos para o sistema em estudo. A legibilidade do mesmo permite que a transição para implementação de código seja facilmente interpretada.

Inicialmente optou-se pela utilização de herança, contudo, quando se estava a fazer a modelação reparou-se que não estava a ser feita a correta reutilização dos recursos, sendo necessário instanciar o objeto para o poder reutilizar. Um exemplo simples é o caso do utilizador referenciado. Supondo que existe um utilizador que é referenciado com o email abc@abc.com. Se for implementada a herança, quando ele se regista é preciso instanciar um utilizador registado. Com composição, pode-se utilizar o *design pattern state*, que faz

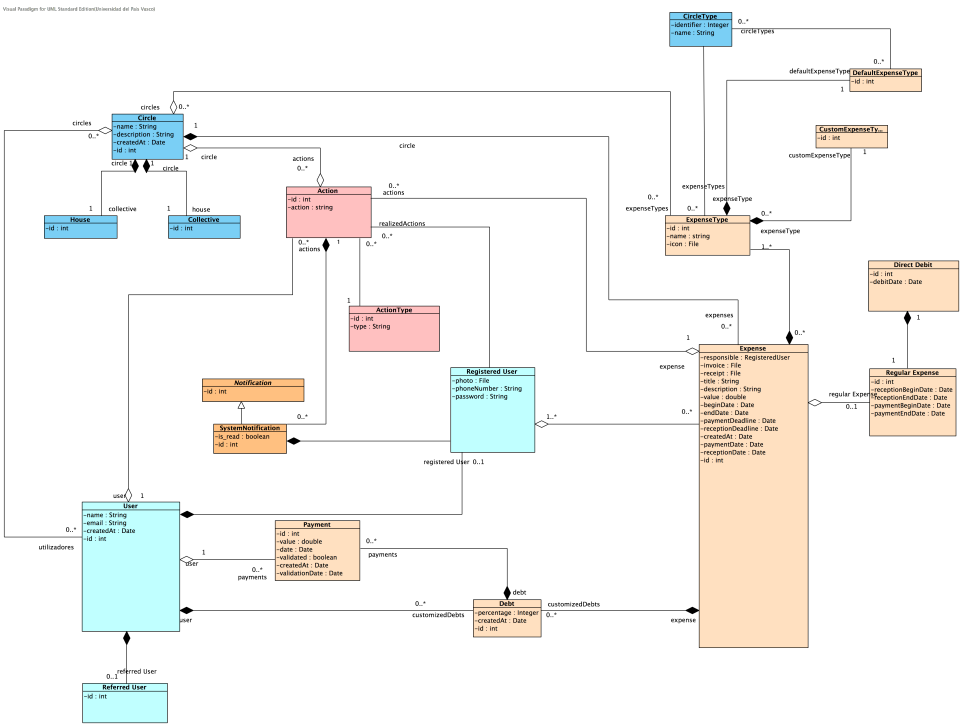

Figura 7.3 Diagrama do modelo de domínio

com que não seja necessário uma nova instância, mudando apenas o estado de utilizador registado para utilizador referenciado.

Normalmente, deve-se preferir a composição sobre a herança, mas obviamente que existem exceções. Basicamente, usa-se a herança quando se sabe que a superclasse não vai variar, porque caso contrário será necessário alterar todas as classes que a implementam. Neste caso, está-se claramente a ver que a herança era a pior escolha em todos os casos, porque, quer-se um relacionamento do tipo ”tem um”, por exemplo, o utilizador registado tem um utilizador.

De seguida serão apresentadas todas as classes do sistema de uma forma mais detalhada, explicando os seus relacionamentos e a sua cardinalidade.

- ***User***
  O *registered user* e o *referred user* têm um *user*. Aqui tem-se a composição de ambos devido ao que foi explicado na introdução desta secção. Sendo o objetivo desta aplicação a partilha de despesas entre várias pessoas, então, a classe *user* tem uma *debt* que a relaciona com a classe *expense*. Esta classe tem ainda outro relacionamento com a classe *payment* que o relaciona com a classe *debt*. Isto ocorre porque um utilizador pode fazer vários pagamentos para pagar uma despesa.

- ***Circle***
  A classe *house* e *collective* têm um *circle*. Com isto, verifica-se que um *circle* tem um ou mais *users*, mas os *users* podem não estar em nenhum *circle*. O *circle* tem um ou mais *expenseTypes*, que serão utilizados para criar as despesas para um determinado círculo. Esta classe contém *expenses* efetuadas por membros do círculo.

- ***Expense***
  Esta é a classe mais importante do sistema, uma vez que se relaciona com todas as classes. Este é bastante significativo porque indica os utilizadores que têm pagamentos em dívidas realizadas num determinado círculo.

- ***Payment***
  Os *payments* indicam os utilizadores que têm pagamentos numa determinada dívida. Um *user* pode ter vários *payments*, porque pode realizar vários pagamentos para pagar uma dívida.

- ***Debt***
  Como já deu para perceber anteriormente, a classe *Debt* indica a dívida que um utilizador tem numa determinada despesa. Um *user* pode ter várias *debts*, mas cada *debt* pertence a um *user*. Do mesmo modo, uma *expense* pode ter zero ou várias *debts*, mas cada *debt* é relativa apenas a uma *expense*.

- ***Action***
  A classe *action* é gerada sempre que qualquer utilizador realizar uma determinada tarefa no sistema.

- ***Notification***
  As notificações serão geradas a partir das *actions* realizadas pelos *users*.

- ***Regular Expense***
  Esta classe é criada sempre que um utilizador quiser criar uma despesa regular, que basicamente o alerta sobre as próximas faturas, para que este não se esqueça de criar as despesas. Com isto, é possível criar uma despesa sem esforço para o utilizador, porque os atributos da despesa são preenchidos com os atributos da despesa regular.

- ***Expense Type***
  Esta classe tem os tipos de despesa que são apresentados no momento em que se criam despesas, porque é obrigatório que cada despesa tenha um tipo de despesa para se efetuar uma organização de despesas mais eficaz na lógica de negócio.

- ***Circle Type***
  Esta classe armazena os tipos de despesa que são apresentados no círculo no momento da sua criação. As que aparecem ao utilizador são as *default expense types*, mas caso queira criar uma nova será adicionada ao *custom expense type*.

Secção 8

INTERFACE

8.1 Prototipagem

8.2 Responsiveness

8.3 Mockups

8.3.1 Web

8.3.2 Mobile

Secção 9

IMPLEMENTAÇÃO

9.1 Servidor

9.1.1 Framework Web - Grails

Nesta secção pretende-se descrever as características da *framework web Grails*, e explicitar as razões que levaram à escolha da mesma para o desenvolvimento da plataforma *BillMate*.

O *Grails* é uma *framework web*, de código aberto, que incorpora o padrão de desenvolvimento *MVC* e que defende o paradigma de convenção em detrimento da configuração, para a linguagem *Groovy*. O seu principal objetivo é ser uma *framework* de alta produtividade, e para isso utiliza as *frameworks Hibernate* e *Spring*.

O *Grails* é construido sobre a linguagem *Java*. Essa característica permite que a sua integração com bibliotecas e *frameworks Java* seja natural.

Uma das principais vantagens na sua utilização é o grande conjunto de *plugins* que esta disponibiliza e que permitem acelerar significativamente o processo de desenvolvimento.

Groovy

O *Groovy* é uma linguagem de programação orientada aos objetos desenvolvida como alternativa à linguagem de programação *Java* e que apresenta características do *Python*, *Ruby* e *Smalltalk*. É uma linguagem que pretende oferecer mecanismos que não estão pre-

sentes no *Java* e melhorar a legibilidade do código produzido.

Arquitetura

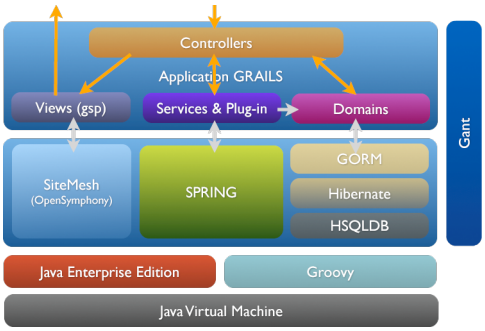

Figura 9.1 Arquitetura Grails

A figura 9.1 apresenta a arquitetura de uma aplicação *Grails*. A sua estrutura será indicada na seguinte lista com mais detalhe.

- ***Domain classes***

  As *domain classes* são dos componentes mais importantes de uma aplicação *web* em *Grails*. Cada *domain class* vai corresponder a uma entidade do sistema, estas entidades são normalmente persistidas e estão associadas a uma tabela da base de dados.

  No contexto de uma aplicação *Java Enterprise Edition (JEE)* e utilizando componentes *Enterprise Java Beans (EJB)*, as *domain classes* poderiam representar *entity beans*.

- ***Controllers***

  Os controladores, à semelhança de outras *frameworks web MVC*, são responsáveis por enviar e receber dados das *views* e gerir redirecionamentos e respostas. Estas classes comunicam com as *domain classes*, com os *services* e com as *views*.

- ***Views***

  As *views* são os componentes responsáveis por apresentar dados ao utilizador.

  Para facilitar a integração de conteúdo dinâmico com conteúdo estático, o *Grails* utiliza *Groovy Server Pages (GSP)* que é uma linguagem de apresentação semelhante ao *Java Server Pages (JSP)*.

- ***Services***

  O *Grails* utiliza os *services* para separar a lógica de negócio dos restantes componentes.

Os *services* podem facilmente fazer uso de funcionalidades de injeção de dependências e são facilmente acessíveis a partir dos controladores.

No contexto de uma aplicação *Java Enterprise Edition (JEE)* e utilizando componentes *Enterprise Java Beans (EJB)*, as *domain classes* poderiam representar *session beans*.

- ***Tag libraries***

  As *tag libraries* são classes que funcionam como *helpers* para a geração de conteúdo dinâmico nas *views*.

Spring

Como referido anteriormente o *Grails* tem a *framework Spring* embebida.
Este acaba por se definir como uma aplicação *Spring MVC* "disfarçada", tal como é indicado na sua documentação oficial.
Consequentemente a *framework* utiliza os mecanismos de gestão de *beans* do *Spring*.
Spring beans

Por omissão os *spring beans* são *singletons*, ou melhor, apenas uma instância de cada *bean* é criada.
No entanto esta omissão nem sempre é a mais indicada, principalmente quando queremos *beans* mutáveis, pelo que é possível definir um *scope* para o referido *bean*.
O Spring suporta cinco *scopes*:

- **singleton**

  Como referido, apenas uma instancia é criada criada e disponibilizada no *Spring IoC container*. Sempre que o *bean* é invocado, é sempre a mesma instância que é disponíbilizada.

- **prototype**

  Permite receber uma nova instância do *bean* de cada vez que ele é invocado. Com este *scope* já é possível ter estados mutáveis nos beans, uma vez que cada invoção recebe uma cópia diferente do objeto.

- **request**

  Atribui uma instância por ciclo de vida de um *HTTP request*, o que significa que para cada pedido uma nova instância do *bean* será criada.

- **session**

  Uma nova instância é criada por ciclo de vida de uma *HTTP session*.

- **global**

  Abribui uma instância por ciclo de vida de uma *global HTTP session*.

Hibernate e GORM

Como referido anteriormente o *Grails* tem a *framework Hibernate* integrada.

Para fazer o mapeamento entre o paradigma relacional e o dos objetos o *Grails* oferece um mecanismo baseado em *Hibernate*, que é o *Grails Object Relational Mapping (GORM)*.

O *GORM* oferece várias facilidades em relação ao *Hibernate*. No *GORM* não é necessário utilizar anotações ou extender classes especiciais para tornar a classe persistente. No *GORM* cada *domain class* representa uma tabela na base de dados e cada atributo representa uma coluna das referidas tabelas.

O *GORM* oferece assim um nível de abstração superior. Este não é mais do que uma interface (*Facade*) para o *Hibernate*.

9.1.2 Base de dados - PostgreSQL

A escolha de um Sistema de Gestão de Bases de Dados (*DBMS*) torna-se fulcral a partir do momento em que se pretende obter escalabilidade, confiança e técnicas de replicação por este fornecidas. Desta forma, é preciso pesquisar quais oferecem estas garantias e quais as que têm um maior suporte, ou até mesmo uma comunidade de suporte, capaz de responder às mais variadas dúvidas sobre o produto em questão.

O open-source costuma, por si só, ter muitos adeptos e, quando é algo que é usado por grandes companhias, torna-se uma das prioridades na hora da escolha. O mesmo se sucedeu no que toca ao *BillMate*, ou seja, acabou por se escolher um *DBMS open-source*, com uma excelente reputação, quer a nível pessoal, quer a nível empresarial: *PostgreSQL*.

Resumindo, os factores que influenciaram a escolha do *PostgreSQL* foram:

- Solução open-source, o que contribui para um estudo fácil do código, se necessário;
- Comunidade que contribui para uma curva de aprendizagem não tão elevada, devido à resolução de problemas;
- *Cross-platform* é uma mais valia para se adaptar a qualquer infraestrutura e sistema que a compõe;
- Escalabidade, com o objectivo de atingir um desempenho suficiente para atender múltiplos clientes.

9.2 Cliente

9.2.1 Web

HTML5

O *HTML5*, na sua lenta chegada, iniciou uma simplificação daquilo que se conhecia como *HTML*. Algumas *tags* foram removidas (algumas delas cujo efeito seria o mesmo que aplicar um estilo) e novas foram introduzidas, tornando o código cada vez mais legível e facilmente editável, assim que for necessário, devido aos seus nomes simples e cada vez mais próximos da linguagem do dia-a-dia. A seleção desta tecnologia para o cliente foi vista como uma mais valia devido à simplicidade e furor, pelo qual é respomsável, em

volta dos *developers*.

CSS3

O *CSS3* mudou completamente a imagem dos estilos, à medida que foi aparecendo. Várias animações deixam de ser apenas do domínio do *JavaScript* e passam a ser totalmente personalizáveis naquilo que antes se chamaria apenas ”estilos”. Para um conforto e dinâmica da aparência do serviço, optou-se por adotar a versão 3 do famoso *CSS*. Note, também, que o uso da *framework BootStrap 3* tem como base o *CSS3*, na sua maioria.

JavaScript

Já não sendo visto tanto como um atalho para animações ou alterações muito focadas no *DOM*, esta linguagem traz uma flexibilidade de processamento do lado cliente, o que pode reduzir o processamento no lado servidor e, por essa grande vantagem, foi uma tecnologia escolhida e valorizada na implementação do cliente. Para uma maior flexibilidade, optou-se,também por usar *jQuery*.

Ajax

Não é uma linguagem! Cai no vasto leque de tecnologias existentes para a *web*. Usa *JavaScript* como base, visto que corre num *browser* e tira partido do facto de não haver (ou haver pouca) transferência de *HTML* por parte de cada pedido ao servidor. Foca-se no uso de *JSON*, maioritariamente e, por isso, a inexistência do *overload* dado pela transmissão de *HTML* não prejudica, muito pelo contrário, a *performance* do serviço.

9.2.2 Mobile

PhoneGap

Uma vez implementado o cliente *web* com as tecnologias mencionadas, seria de bom grado obter uma aplicação *mobile* multi-plataforma sem o esforço de desenvolver uma por cada sistema operativo. Desta forma, optou-se por experimentar a *framework PhoneGap* que permite ao programador criar estas aplicações *mobile*, usando apenas *API's web standardizadas*. Assim, muito daquilo que foi implementado no cliente pôde ser aproveitado para iniciar o desenvolvimento da aplicação *mobile* do serviço.

Secção 10

ANALÍSE DE DESEMPENHO

Secção 11

TRABALHO FUTURO

Secção 12

CONCLUSÃO

BIBLIOGRAFIA

REFERÊNCIAS WWW

APÊNDICE A
WEB DASHBOARD

APÊNDICE B

MOBILE DASHBOARD