Relatório

Mestrado em Engenharia Informática - Computação Móvel

g.Core – Sistema de gestão empresarial

João Miguel Silva Sousa

Leiria, *Março* de *2015*

escola superior de tecnologia e gestão
instituto politécnico de leiria

Relatório

Mestrado em Engenharia Informática - Computação Móvel

g.Core – Software de gestão empresarial

João Miguel Silva Sousa

Estágio de Mestrado realizado sob a orientação do Doutor Carlos Fernando Almeida Grilo, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria

Leiria, *Março* de *2015*

À Minha Família

Agradecimentos

Agradeço à empresa Grifin por se ter disponibilizado a receber o meu estágio e me ter ajudado a realizá-lo, com especial agradecimento ao meu coordenador de estágio, João Clérigo, que com a sua experiência me ajudou sempre que foi necessário. Ao professor Carlos Grilo, foi incansável, sempre pronto para me ajudar, e por isso, quero agradecer toda a sua ajuda, obrigado. Quero agradecer à minha família, pai mãe e irmã pela força que me deram para realizar este trabalho. Obrigado, Valdemar, Susana e Joana. Por último, quero agradecer à Margarida Gabriel por toda a ajuda e motivação para finalizar este relatório de estágio.

Esta página foi intencionalmente deixada em branco

Resumo

Este documento tem como objetivo dar a conhecer o trabalho desenvolvido no âmbito do estágio realizado na empresa "Grifin – Soluções de Informática". O trabalho desenvolvido passou por realizar três aplicações distintas que integrassem com a aplicação g.Core, aplicação de gestão empresarial e industrial, desenvolvida pela Grifin. Uma das aplicações, de nome Relógio de Ponto, é uma aplicação de controlo de acessos que usa um leitor biométrico para identificar o utilizador. As outras duas aplicações clientes foram desenvolvidas para o sistema operativo Android e são denominadas de g.Core Mobile e g.Produção. A aplicação g.Core Mobile é otimizada para *smartphones* com o objetivo de oferecer algumas funcionalidades que estão presentes na aplicação g.Core mas agora num contexto de mobilidade. A g.Produção é uma aplicação desenvolvida apenas para *tablets*, tendo como objetivo permitir aos funcionários de uma unidade de produção da indústria de moldes registar o tempo de cada trabalho realizado.

A aplicação g.Core não está desenvolvida de forma de forma a escalar, não tendo camadas de negócio e de acesso dados bem definidas, pelo que houve a necessidade de criar uma nova aplicação, de nome g.Webservice, que disponibilize algumas funcionalidades do g.Core em forma de serviços web REST. Dessa forma, foi possível desenvolver as três aplicações clientes e preparar a aplicação g.Core para integrar outros clientes.

Palavras-chave: Integração de Sistemas, Software de Gestão Empresarial, Mobilidade

Esta página foi intencionalmente deixada em branco

Abstract

This document has the objective of presenting the work developed under the internship in the company "Grifin – Soluções de Informática". The work consisted in developing three different applications that integrate with g.Core, a business and industrial management application, developed by Grifn. One of the applications, named "Relógio de Ponto", is an access control application which uses a biometric reader to identify the user. The other two client applications have been developed for the Android platform and are called g.Core Mobile and g.Produção. The g.Core Mobile application is optimized for smartphones in order to offer some features that are present in g.Core application but now in a mobility context. The g.Produção is an application developed only for tablets, aiming to allow employees of a production unit of the mold industry to record the time of each task.

The g.Core application is not developed in a scalable way, not having well-defined business and data access layers, and there was the need of creating a new application, that was named g.Webservice, to make available some features of g.Core in the form of web REST services. Thus, it was possible to develop the three client applications and prepare g.Core application to integrate other client applications.

Key-Words: Integração de Sistemas, Software de Gestão Industrial, Mobilida

Esta página foi intencionalmente deixada em branco

Índice de Figuras

Índice de Tabelas

Esta página foi intencionalmente deixada em branco

Lista de Siglas

CRM	Customer Relationship Management
CRUD	Create, Read, Update e Delete
CRUD	Create, Read, Update e Delete
GCM	Google Cloud Message
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
MVC	Model View Controller
NFC	Near Field Communication
ORM	Object-Relational Mapping
PC	Personal Computer
REST	Representational State Transfer
RFID	Radio-Frequency Identification
RP	Relógio de Ponto
SDK	Software Development Kit
SMS	Short Message Service
SOAP	Simple Object Access Protocol
SRP	Single Responsibility Principle

Esta página foi intencionalmente deixada em branco

Índice

Introdução

Os sistemas de gestão empresarial e de produção têm vindo a ocupar cada vez mais importância. É nesse sentido que a Grifin [1] está a desenvolver o software g.Core. O g.Core pretende juntar funcionalidades de gestão de projetos, clientes, colaboradores e produção de forma a oferecer as funcionalidades necessárias às empresas do setor industrial principalmente à indústria de produção de moldes.

1.1 Entidade Acolhimento

A entidade de acolhimento para este estágio foi a empresa "Grifin – Soluções de Informática" fundada em 1999 e sediada na Marinha Grande. A Grifin tem como principal foco de negócio a assistência técnica e de suporte a empresas e particulares. Com o considerável desenvolvimento industrial existente na Marinha Grande surgiu a oportunidade de começar a desenvolver soluções de *software* ajustadas à realidade da indústria local. Após o desenvolvimento de algumas aplicações à medida para empresas da área dos moldes e plásticos, a Grifin decidiu desenvolver um *software* de gestão industrial que pudesse ser vendido a várias empresas. Com isto nasceu a aplicação g.Core, *software* de gestão empresarial, focado nas necessidades da indústria local, com principal foco na indústria dos moldes.

1.2 Aplicação g.Core

Os sistemas de gestão empresarial têm uma grande importância nos dias de hoje. Estes sistemas permitem às empresas gerir todo o seu processo de negócio como, por exemplo, realizar a gestão de recursos ou da faturação, permitindo também analisar todos os processos que já decorreram. Para tal, existem produtos de faturação adaptados à gestão empresarial. No entanto, estes produtos nem sempre oferecem todas as funcionalidades desejadas por serem demasiado genéricos. Existem setores de atividade, como por exemplo o setor industrial, que necessitam de funcionalidades mais específicas, principalmente a nível da produção. Foi para essa área que a Grifin começou a desenvolver o sistema de gestão g.Core.

O g.Core é um sistema de gestão empresarial, focado na sua generalidade, para a indústria, mas de uma forma mais específica para a indústria dos plásticos e moldes, pois estes dois setores são predominantes na região da Marinha Grande, onde a Grifin está sediada. Este

sistema, ao contrário da maioria, não consiste numa aplicação *standalone*, mas sim num portal Web. Cada empresa tem o seu g.Core, localizado num domínio único próprio.

O g.Core está dividido em vários módulos independentes, cada um com funcionalidades específicas. Os módulos são os seguintes: g.CRM, g.Project, g.Moulds e g.Point.

O módulo g.CRM tem como principal objetivo a gestão de relacionamento com entidades externas, tais como, clientes, fornecedores e fabricantes. Além da gestão dos dados deste tipo de entidades, este módulo permite também a criação de atividades e respetivos relatórios, gerir contratos e gestão de atividades comerciais.

O g.Project é um módulo destinado à gestão de projetos, onde é possível atribuir projetos a utilizadores e fazer a inserção dos custos de cada fase do projeto. Este módulo permite também analisar todos os dados referentes a projetos, nomeadamente os custos.

Como referimos acima, a indústria dos moldes é predominante na região onde a Grifin é sediada. Aproveitando esse espeto, o g.Core disponibiliza um módulo dedicado a essa indústria. O módulo g.Moulds pretende gerir toda a produção de moldes desde o planeamento até à sua fabricação.

Por fim o g.Point, um módulo que disponibiliza gestão de assiduidade e pontualidade dos funcionários de uma empresa. Permite ainda a gestão de dias de férias e horários de trabalho. Este módulo trabalha em conjunto com uma aplicação que permite registar a entrada dos funcionários e enviar esses dados para o g.Core.

Embora estes módulos possam funcionar de forma independente, o sistema permite que vários módulos estejam ativos em simultâneo e que comuniquem entre si. O sistema está desenhado de modo a que a ativação simultânea de vários módulos permita disponibilizar funcionalidades que cada um não disponibiliza de forma isolada. Por exemplo, tendo o modulo CRM, Project e g.Point ativos simultaneamente, é possível ter total controlo na conceção de projetos para clientes, pois o *software* disponibiliza informações estatísticas acerca de custos, prazos e previsões de lucro. Caso estes módulos não estivessem ativos simultaneamente não seria possível obter estatísticas tão específicas.

A organização modular do g.Core permite aos clientes obter apenas as funcionalidades de que necessitam, adquirindo apenas os módulos necessários. Num universo de milhares de pequenas, médias e grandes empresas do setor industrial na região, esta solução adapta-se consoante a necessidade da empresa tornando-se, nesse aspeto, muito escalável.

1.3 Motivação e Objetivos

Muitos setores industriais estão em constante crescimento e desenvolvimento. Isso leva à necessidade de um maior controlo e normalização de processos. O setor dos moldes é um exemplo dessa tendência, com clientes cada vez mais exigentes que impõem determinados padrões de qualidade, desempenho e organização elevados na realização de trabalhos.

Tendo em conta as exigências impostas pelo mercado atual, a maioria das áreas industriais necessitam de *software* que ajude a alcançar as métricas anteriormente referidas. Foi nesse sentido que a Grifin iniciou o projeto g.Core, um *software* de gestão industrial que pudesse ser útil para qualquer tipo de empresa do setor industrial.

Quando este estágio iniciou o projeto g.Core já tinha começado acerca de um ano e já estava em fase de produção. A minha integração neste projeto foi pensada com o propósito de desenvolver novas funcionalidades na aplicação bem como o desenvolvimento de aplicações móveis que estendessem funcionalidades do g.Core. Outro objetivo traçado para este estágio foi o desenvolvimento de uma nova aplicação g.Point de forma a substituir a que já existia.

Com o desenvolvimento das novas funcionalidades o g.Core terá novos argumentos em relação à concorrência, oferecendo aos seus utilizadores novas funcionalidades em termos de mobilidade e inovação, fatores que são decisivos para adquirir novos clientes.

1.4 Estrutura do documento

Esta seção descre a estrutura do documento. Os restantes capítulos são constituídos por:

- Capítulo 2: Neste capítulo é explicada a importância das aplicações de gestão industrial, particularmente de gestão de produção. Também é descrito o estado do *software* g.Core antes do estágio.
- Capítulo 3: Aqui é descrita a metodologia utilizada para o desenvolvimento dos vários projetos, assim como a forma como foi planeado.
- Capitulo 4: Este capítulo mostra as alterações realizadas na aplicação g.Core já no contexto deste estágio.
- Capítulo 5: Neste capítulo é explicado todo o processo de desenvolvimento da aplicação g.Webservice, desde justificações tecnológicas à implementação.
- Capítulo 6: Este capítulo descreve o desenvolvimento da aplicação Relógio de Ponto bem como o seu funcionamento.
- Capítulo 7: Neste capítulo é descrito todo o processo de desenvolvimento das aplicações móveis, g.Core Mobile e g.Produção.
- Capítulo 8: Por fim, neste capítulo são resumidos os resultados obtidos neste estágio, bem como uma análise de possíveis trabalhos futuros a desenvolver no g.Core.

Estado da Arte

Este capitulo tem como objetivo dar a conhecer as áreas abrangidas neste projeto principalmente dar a entender a importância do *software* de gestão na área da indústria. Segue-se uma análise da utilização de *software* para a gestão de processos industriais e da importância da mobilidade neste tipo de processos. Por último, pretende-se explicar o que é o controlo de assiduidade bem como várias soluções existentes.

2.1 Gestão de Produção Industrial

No final do século XVIII deu-se início à revolução industrial [2] [3], trazendo novos métodos de produção mais automatizados e eficientes. A revolução industrial começou devido à enorme concorrência e ambição em ter maiores lucros por parte das empresas do setor têxtil, que começaram a utilizar o motor a vapor e mais tarde motores elétricos e de explosão.

Na segunda metade do seculo XX iniciou-se uma nova era, a era computacional com a invenção do computador [4]. Conseguindo realizar enorme quantidade de cálculos e capacidade de executar aplicações, o computador começou a ser utilizado pelas indústrias para otimização de custos, conseguindo prever lucros e prejuízos de forma fiável. A chegada do computador permitiu às empresas calcular, por exemplo, custos de eletricidade, água e de materiais de forma quase exata, fator decisivo.

Nos dias de hoje a maioria das empresas utilizam *software* de gestão, como é o caso de *software* de faturação. Muitos dos *produtos* destinados à gestão da faturação têm inúmeras funcionalidades (como gestão de stocks, gestão de clientes, etc.) e muitos deles (ex.: Primavera [5], PHC [6], etc.), permitem o desenvolvimento de funcionalidades extra, dando ao cliente a possibilidade de adaptar o *software* às suas necessidades.

Embora os produtos de gestão de faturação sejam adaptáveis, por vezes não são suficientes para a exigências de algumas empresas, com é o caso das empresas dos setor industrial. Frequentemente, este setor tem necessidades muito específicas, principalmente quando existem processos de produção muito complexos, sendo necessária a utilização de *software* mais específico para a gestão de produção como é o caso do Organimold e o g.Core, este último desenvolvido pela Grifin.

2.1.1 *Organimold*

O Organimold é um *software* produzido pela empresa GrandSoft, sediada na Marinha Grande, que tem como objetivo a gestão da produção de moldes. Este *software* é o principal concorrente do g.Core, estando no mercado desde 2003 e com funcionalidades muito avançadas e específicas para a indústria dos moldes.

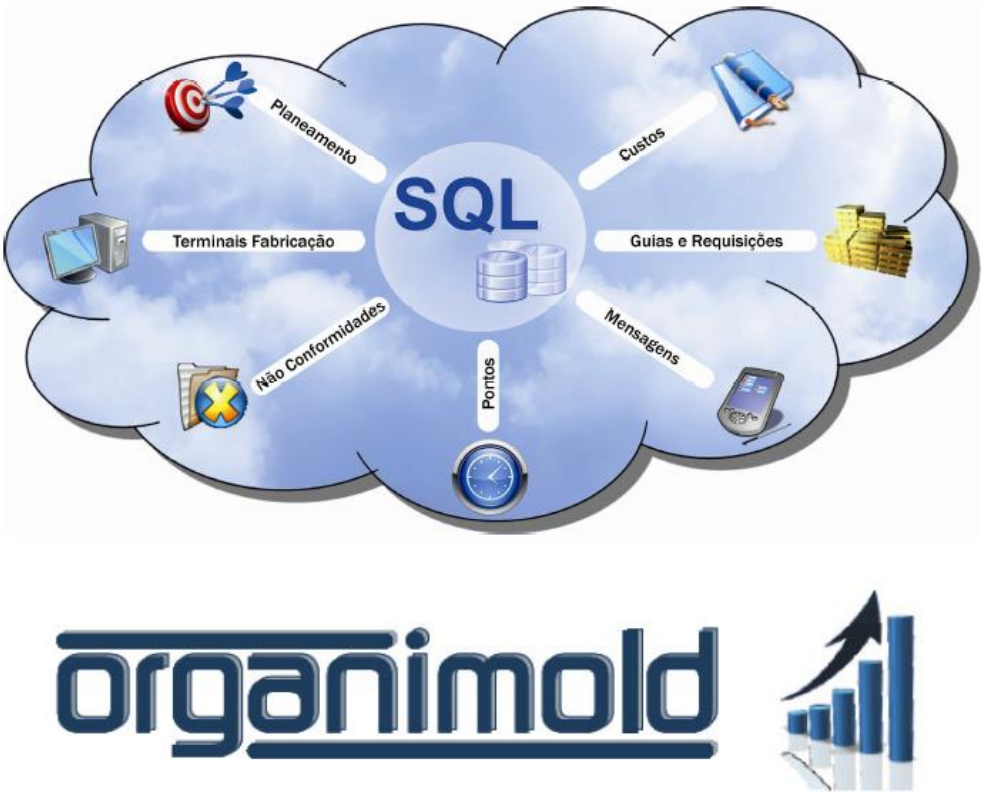

Figura 1 Esquema de funcionamento e funcionalidades do Organimold. [7]

O Organimold é um *software* [7] *standalone* que está ligado a uma base de dados presente na rede onde são armazenados todos os dados. Esta aplicação disponibiliza as seguintes funcionalidades:

- Gestão de Obras/Moldes e Orçamentos;
- Planeamento Geral da Produção;
- Planeamento por Secção;
- Terminal Fabricação (PC);
- Orçamentos;
- Análise de Custos;

- Gestão de Não Conformidades;
- SMS – Serviço de Mensagens Internas.

Por si só, o Organimold apenas dispõe de funcionalidades de gestão de produção de moldes, não dispondo de funcionalidades de faturação. A empresa WinSIG® [8] desenvolveu uma aplicação, de nome SIG Moldes [9], que permite interligar o Organimold com o software de faturação PHC. O conjunto destes dois sistemas permite a uma empresa de produção de moldes gerir todo o seu negócio, deste a orçamentação e faturação até à produção dos mesmos.

2.1.2 g.Moulds

O g.Moulds é um módulo pertencente ao g.Core que oferece funcionalidades de gestão de produção de moldes. Este módulo está em constante desenvolvimento, sendo desenvolvido em parceria com algumas empresas de produção de moldes da região de Leiria.

Ao início do estágio o g.Moulds apresentava as seguintes funcionalidades gerais:

- Gestão de Obras/Moldes e Orçamentos;
- Orçamentos;
- Gestão de Não Conformidades;

As funcionalidades completas do sofware g.Core estão disponíveis no Anexo I, onde estão tambem listadas as funcionalidades de cada módulo, incluído o g.Moulds.

2.2 Controlo de Assiduidade e Pontualidade

No mundo empresarial, onde normalmente existe exigência de horários, é muito importante haver um controlo da assiduidade e pontualidade dos funcionários. Existem muitos sistemas que permitem fazer este tipo de controlo de forma o mais otimizada possível, recorrendo a terminais de registos, onde o funcionário regista as suas entradas e saídas. Os terminais de registos normalmente permitem registos com recurso às seguintes tecnologias de identificação:

- Tag RFid;
- Tag ou dispositivo com *Near Field Communication* (NFC);
- Credenciais de acesso;
- Reconhecimento biométrico como, por exemplo, facial ou impressão digital.

Associada aos terminais existe normalmente uma aplicação proprietária que permite gerir todos os funcionários em relação à sua assiduidade e pontualidade, permitindo algumas também gerir férias, feriados, bem como visualizações de relatórios. Existem várias soluções no mercado que prestam este tipo de soluções, como por exemplo: Idonic® [10], Netponto® [11], Safetech® [12].

2.2.1 g.Point

Como já referido na secção 1.2, o g.Core disponibiliza um módulo de gestão de assiduidade e pontualidade. Este módulo trabalha em conjunto com uma aplicação terminal também desenvolvida pela Grifin. O terminal, o g.Point, permite registo biométrico através de impressões digitais. No entanto, este terminal tem um pequeno passo extra no registo que podia ser eliminado: para fazer o registo, para além da impressão digital, é necessário inserir o número de funcionário. Este espeto podia ser melhorado, pedindo apenas a impressão digital, algo que a nova versão desenvolvida no âmbito deste estágio já corrige.

A aplicação terminal envia os registos periodicamente para a base de dados do g.Core, podendo o g.Point ter acesso aos registos efetuados. O módulo g.Point permite:

- Gestão de horários e de planos de trabalhos, sendo possível ainda configurar limites e tolerâncias;
- Gestão de férias, feriados;
- Visualização e exportação das listagens dos registos dos utilizadores.

As principais características do g.Point estão descritas no Anexo I.

2.3 Aplicação g.Core

O g.Core é uma aplicação web de gestão empresarial cujos módulos foram descritos de forma genérica no Capítulo 1. Na secção 2.1.2 e 2.2.1 descrevemos também com mais algum pormenor as funcionalidades dos módulos g.Moulds e g.Point. Esta aplicação está desenvolvida na linguagem de programação PHP [13] e dispõe de uma típica arquitetura cliente-servidor onde os clientes acedem a todas as funcionalidades da aplicação via HTTPS [14] através de um *browser* com suporte a HTML5. Sendo o g.Core focado nas empresas, existem alguns aspetos que a Grifin quis preservar na realização deste projeto como, por exemplo, a segurança e a integridade dos dados. Estes aspetos são essenciais porque muitas das empresas que usam este *software* dedicam-se ao mesmo setor de atividade, sendo concorrentes entre elas. Nesse sentido, cada cliente tem um domínio único onde está instalada e configurada a aplicação. Cada aplicação apenas diz respeito a um cliente particular, tendo uma base de dados única que só contém dados do mesmo. Desta forma, caso haja problemas numa base de dados as outras não são afetadas.

Tal como mostra a figura seguinte, o g.Core pode ser dividido verticalmente, em dois componentes principais: o g.Core propriamente dito e a base de dados. Ou seja, no g.Core não existe uma separação clara entre as camadas usuais em aplicações empresariais, nomeadamente, as camadas de apresentação, de negócio e de acesso a dados. Na prática, isto significa que cada vista inclui a apresentação, a lógica de negócio e o acesso a dados.

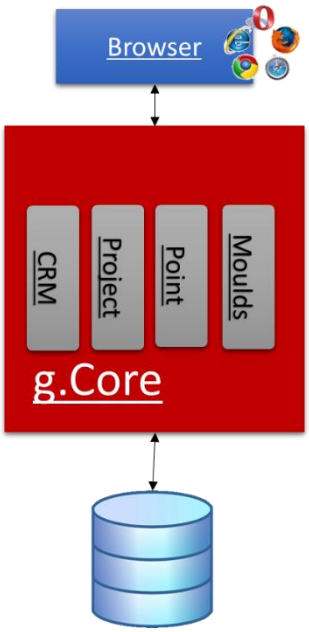

Figura 2 Arquitetura do g.Core antes do estágio.

Como foi já referido, o g.Core é uma aplicação web que tem como objetivo a gestão empresarial e, horizontalmente, é constituída por vários módulos que disponibilizam funcionalidades distintas.

A base de dados contém todos os dados envolvidos na aplicação e é comum a todos os módulos. Desenvolvida sob o sistema relacional de base de dados MySQL, todas as tabelas são armazenadas usando o motor MyISAM [15], disponível neste sistema. O motor MyISAM, em relação ao motor InnoDB [16], permite uma maior flexibilidade e facilidade na alteração da base de dados devido à forma como as tabelas são guardadas em disco. Enquanto o InnoDB guarda todas as tabelas dentro de um único ficheiro, o MyISAM divide em vários ficheiros, ocupando também menos espaço. Como todos os clientes estão alojados no mesmo servidor o espaço é determinante, pois é partilhado entre eles. No entanto o MyISAM tem algumas limitações em relação ao InnoDB, nomeadamente, o facto de não suportar relacionamentos entre tabelas e transações [17] [18] [19].

Até ao início do estágio não existia no g.Core nenhuma divisão entre lógica de negócio e acesso a dados. O acesso a dados no g.Core era totalmente realizado através de *queries* SQL criadas pelo programador no contexto do código. Isto faz com que, no mínimo, exista uma grande repetição de *queries* SQL pois nenhuma delas é reaproveitada o que torna a manutenção da aplicação muito mais difícil pois os acessos à base de dados estão divididos pelas múltiplas páginas da aplicação. Para a execução das *queries*, era, e em alguns casos é ainda usado, o interface PHP Database Objects (PDO) [20] [21] [22] que disponibiliza funcionalidades de segurança contra SQL Injection.

2.3.1 Gestão de caminhos (routes)

A aplicação g.Core dispõe de uma área de administração, já desenvolvida aquando do início do estágio. Esta área, que apenas está acessível à equipa de desenvolvimento, permite gerir

caminhos para páginas do *software*. Quando a equipa de desenvolvimento cria uma página nova, tem de acrescentar o caminho para essa página na área de administração. Para a criação de um novo caminho, é necessário o caminho para o ficheiro da nova página, um nome para a página e o módulo ou subfuncionalidade do módulo a que pertence.

Com os caminhos inseridos na área de administração, os menus respetivos são gerados automaticamente, podendo os utilizadores aceder às respetivas funcionalidades a partir desse momento.

2.3.2 Gestão de permissões

A gestão de permissões permite definir as funcionalidades a que cada utilizador ou conjunto de utilizadores registados podem ou não aceder.

No g.Core as permissões são dadas a departamentos. Departamentos são conjuntos de utilizadores que têm necessidades idênticas na utilização da aplicação, sendo assim configurados para pertencer ao mesmo departamento. Por exemplo, um departamento comercial é constituído por utilizadores que necessitam maioritariamente de funcionalidades disponibilizadas no módulo de g.CRM enquanto um departamento de gestão de projeto necessita mais de funcionalidades presentes no g.Project. Por isto, pode haver necessidade de limitar o acesso a determinadas funcionalidades. Esta limitação pode ser definida pelo administrador do sistema na página de gestão de cada departamento. Para cada caminho criado pode ou não ser dado o acesso a um determinado departamento.

2.3.3 Segurança

É necessária uma licença de uso, emitida pela empresa Grifin, para que o g.Core possa ser utilizado. Depois disso, apenas os utilizadores com conta, criada previamente pelo administrador, podem utilizar o *software*. Para entrar, os utilizadores usam o *username* e *password* fornecidos pelo administrador do sistema. Para garantir a autenticidade em cada caminho, o *software* usa o sistema de sessões. Este sistema é utilizado pela maioria das plataformas web.

Para garantir uma maior segurança, o g.Core permite ainda o acesso restrito a um determinado IP. Este método é muito utilizado por empresas que pretendam que os seus colaboradores tenham acesso à aplicação apenas quando estão na rede da empresa.

Finalmente, a Grifin disponibiliza também a possibilidade do uso de HTTPS, aumentando o nível de segurança da aplicação. O serviço HTTPS é considerado com uma funcionalidade extra pelo qual o cliente tem de pagar para usufruir.

Metodologia e Gestão de Projeto

Este capítulo descreve a metodologia utilizada para o desenvolvimento dos vários projetos desenvolvidos no âmbito deste estágio.

3.1 Metodologia

As metodologias [23] de desenvolvimento têm grande importância no desenvolvimento de *software*, auxiliando o processo de produção de forma a obter um produto de qualidade e de custos reduzidos. Existem vários tipos de metodologias de desenvolvimento, entre metodologias ágeis e outras mais clássicas, tendo todas o objetivo definir um conjunto de regras a seguir na realização de projetos.

A entidade de acolhimento, Grifin, já tinha muita experiência no desenvolvimento de *software*, pelo que já possuía uma metodologia de desenvolvimento definida adaptada à realidade e recursos da empresa. A metodologia de desenvolvimento utilizada é adaptada à constante necessidade que os clientes têm de obter novas funcionalidades e correções o mais rápido possível e aos poucos recursos humanos da empresa, sendo o processo de desenvolvimento composto pelas seguintes fases:

- **Análise dos Requisitos:** Após a recolha dos requisitos junto do cliente, a equipa de desenvolvimento (programadores e gestor de projeto) reúne-se para realizar a análise de requisitos. Nesta etapa a equipa de programação dá a sua opinião sobre os requisitos, discutindo o nível de dificuldade bem com o tempo de desenvolvimento para cada requisito.

- **Planeamento:** Nesta fase o gestor de projeto define as diferentes etapas de produção do projeto, normalmente com duração de 30 dias, bem como os requisitos para cada uma. Cada etapa significa uma versão pronta para entrega do projeto final, sendo as funcionalidades desenvolvidas de forma incremental, até que estejam cumpridos todos os requisitos.

- **Desenvolvimento:** Após o gestor de projeto definir os requisitos para a próxima a etapa, a equipa de programação inicia o desenvolvimento das funcionalidades definidas para a etapa. Durante esta fase são realizadas reuniões semanais já que o tamanho da empresa e da equipa de trabalho permite um acompanhamento constante de toda a equipa.

- **Validação:** A etapa de validação passa por uma fase inicial de testes de aceitação interna. Estes testes são realizados pela equipa do projeto com o objetivo de encontrar

erros introduzidos na etapa de desenvolvimento. Só após essa primeira validação é que o *software* desenvolvido é colocado em produção, sendo designados apenas alguns utilizadores para testarem/utilizarem as novas funcionalidades. Esta etapa é muito importante já que permite obter *feedback* do cliente em relação ao trabalho desenvolvido podendo levar a alterações adicionais no produto.

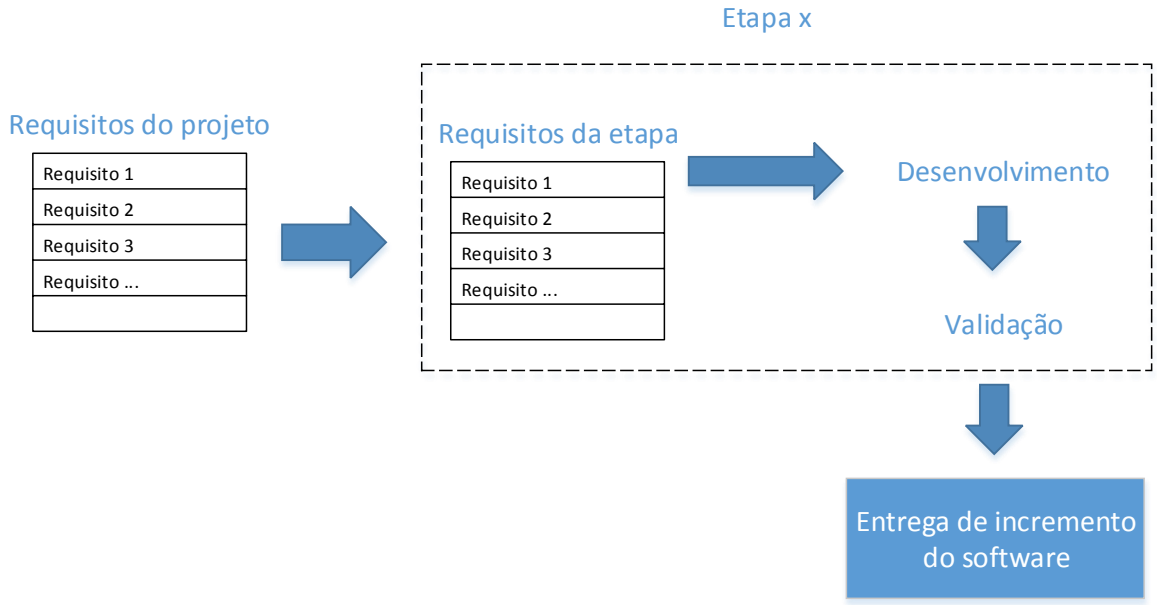

Figura 3 Metodologia de desenvolvimento utilizada.

Na metodologia de desenvolvimento utilizada neste estágio encontramos algumas características associadas a metodologias de desenvolvimento SCRUM [24] como é o caso da divisão dos requisitos por várias etapas e a consequente entrega de incrementos do projeto.

O Anexo II mostrar a lista de requisitos para cada projeto desenvolvido no âmbito deste estágio.

3.2 Gestão de Projeto

O estágio teve início a 3 de setembro de 2013, prolongando-se até dia 30 de junho de 2014, o que corresponde uma duração de 10 meses. Este estágio decorreu num contexto empresarial, exigente, com a necessidade de adaptação à empresa e à sua forma de trabalhar.

As primeiras semanas foram de formação sobre o *software* de gestão g.Core, desde as suas funcionalidades até à forma como fora desenvolvido até então. Como o g.Core já se encontrava numa fase avançada de desenvolvimento, já com muitas funcionalidades desenvolvidas, foi necessário algum tempo até entender toda a lógica de negócio, modelo de dados e estrutura do código do projeto. Embora, no âmbito deste estágio, tenham sido

desenvolvidos projetos totalmente novos, todos estão relacionados com o g.Core, dependendo dos dados e da sua camada de lógica de negócio.

Para este estágio estavam definidos, de forma generalizada, a seguinte ordem de trabalhos:

- Análise e estudo do *software* g.Core;
- Desenvolvimento do Relógio de Ponto 2.0;
- Desenvolvimento da aplicação móvel g.Core Mobile;
- Desenvolvimento da aplicação móvel g.Produção.

Durante a fase de análise do g.Core descobrimos alguns problemas no desenvolvimento já existente: o g.Core não tinha sido desenvolvido de forma a possibilitar a integração com outras aplicações. Este aspeto, explicado no capítulo 2.4, levou-nos a adicionar uma nova tarefa na ordem de trabalhos, o g.Webservice, ficando a ordem de trabalhos da seguinte forma:

- Análise e estudo do *software* g.Core;
- Desenvolvimento do g.Webservice;
- Desenvolvimento do Relógio de Ponto 2.0;
- Desenvolvimento da aplicação móvel g.Core Mobile;
- Desenvolvimento da aplicação móvel g.Produção.

Com esta nova ordem de trabalhos aumentámos a quantidade de trabalho com o desenvolvimento do g.Webservice. _No entanto, considerámos que podíamos dessa forma realizar um projeto mais bem estruturado, trazendo novas ideias e valor acrescentado à empresa.

O g.Core

Como referimos na secção 2.3, no g.Core não havia uma divisão entre a lógica de negócio e a camada de acesso a dados antes do estágio ter início. Esse problema foi detetado após algum tempo de desenvolvimento na aplicação. A repetição de *queries* e a correção de *queries* erradas foi o principal motivo para propormos a criação de uma camada de acesso a dados.

Uma possível solução poderia passar pela adoção de uma plataforma que implemente o padrão de desenho MVC [25] [26]. Este padrão prevê a divisão de uma aplicação em três camadas: modelos, vistas e controladores. O uso de um padrão deste tipo solucionaria à partida os problemas estruturais do g.Core: a inexistência de divisão entre camada de acesso a dados, lógica de negócio e vistas. No entanto, transpor todo o código existente para uma arquitetura deste tipo representaria uma tarefa muito onerosa para a empresa.

Uma segunda solução consistiria em criar apenas a camada de acesso a dados. Começámos por estudar várias soluções disponibilizadas que facilitam a criação desta camada. Após uma pesquisa detalhada, analisámos várias soluções, entre elas, ferramentas de Object-Relational Mapping (ORM) [27]. O ORM é uma técnica usada para acesso a bases de dados que permite definir e gerir o mapeamento entre as tabelas da base de dados e classes do modelo de domínio. Ao utilizar uma ferramenta deste tipo, o programador trabalha apenas com classes, ficando o módulo de ORM responsável por fazer de interface entre os objetos e as tabelas da base de dados. No entanto, uma ferramenta de ORM não tem como objetivo substituir completamente as tradicionais *queries* SQL, já que pode haver perda de desempenho em consultas com muitos cálculos. Estas ferramentas facilitam sobretudo nas tarefas de CRUD (Create, Read, Update e Delete), evitando a criação de *queries* SQL para realizar este tipo de operações.

Foram analisadas várias ferramentas de ORM como por exemplo a Doctrine 2 [28], Eloquent e Propel. Todas estas ferramentas são muito completas no que se refere a operações de base de dados, apenas diferindo na forma com foram desenvolvidas. A Doctrine 2 baseia-se no padrão de desenho Data Mapper [29] enquanto o Eloquent [30] e o Propel [31] implementam o padrão Active Record [32].

4.1 Data Mapper vs Active Record

Existem vários padrões de desenho para a criação de ferramentas ORM, sendo o Data Mapper e o Active Record os mais utilizados [33].

O Data Mapper é um padrão de desenho que segue o princípio Single Responsibility Principle (SRP) [33], que prevê que as classes correspondentes às entidades do modelo de domínio não

tenham qualquer funcionalidade inerente à ferramenta de ORM, sendo utilizados outros objetos para realizar as operações de persistência das entidades. Este padrão prevê uma camada que relaciona as entidades com a base de dados sem que estas necessitem de implementar uma interface ou de herdar de uma classe.

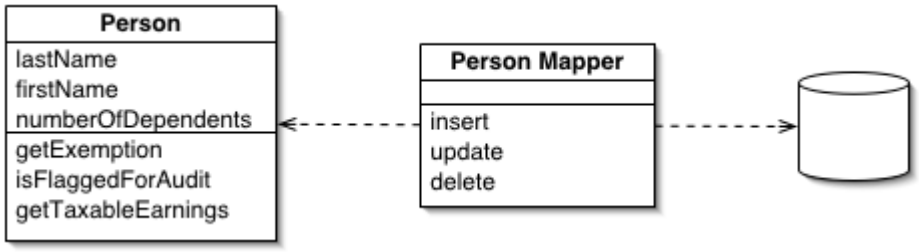

Figura 4 Padrão de desenho Data Mapper. [34]

Por seu lado, o Active Record é um padrão de desenho que prevê que a própria entidade trata das operações de persistência, nomeadamente as operações de CRUD.

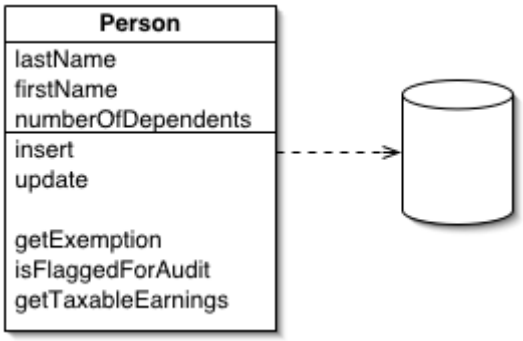

Figura 5 Padrão de desenho Active Record. [35]

Ambos os padrões têm as suas vantagens e desvantagens. O Data Mapper é mais indicado para aplicações de grande dimensão, onde é necessário ter um modelo de domínio simples e independente da ferramenta de acesso a dados; O Active Record é mais indicado para aplicações mais simples e com muitas ações de CRUD a desempenhar.

4.2 Eloquent

Depois de uma análise realizada entre várias ferramentas de ORM, o Eloquent foi a escolhida. Embora o Active Record também fosse uma boa opção houve um fator determinante que tornou o Eloquent a escolha ideal: o uso deste ORM pelo Laravel. Já a pensar numa solução de futuro para a restruturação do g.Core, e apos um estudo de várias *frameworks* PHP, concluímos que o Laravel é uma *framework* popular e de muito potencial para utilizar numa

próxima versão do g.Core. Com a escolha do Eloquent preparamos o g.Core para trabalhar à partida com essa *framework*, embora o Eloquent não seja apenas passível de ser usado com o Laravel.

Uma vez tomada a decisão, procedeu-se à integração do Eloquent no g.Core. Este processo passou por criar o modelo de domínio em que, em geral, cada classe representa uma tabela. O Eloquent prevê que, para cada tabela, seja criada uma classe associada que estenda da classe Eloquent. A figura seguinte representa a implementação de uma classe de forma a ser utilizada com o Eloquent:

```
class Utilizador extends Eloquent {

    protected $table = 'core_base_utilizadores';
    protected $updated_at = false;
    protected $created_at = false;
    protected $primaryKey = "id_utilizador"

    public function departamento()
    {
        return $this->belongsTo('Departamento',"id_departamento");
    }


    public function actividades()
    {
        return $this->hasMany('Actividade',"id_utilizador");
    }

    public function contacto()
    {
        return $this->hasOne('Contacto',"id_utilizador");
    }

    public function projectos()
    {
        return $this->belongsToMany('Projectos',"id_utilizador");
    }
}
```

Figura 6 Classe herda da classe Eloquent.

Esta classe, correspondente à entidade Utilizador, representa a tabela *core*_base_utilizadores e tem como chave primária id_utilizador. Para definir estes dois parâmetros, o nome da tabela e a chave primária, utiliza-se a declaração das variáveis table e primaryKey com os respetivos valores. Na figura podemos ver ainda as variáveis booleanas created_at e updated_at. Estas variáveis definem se a tabela deve conter os campos created_at e timestamp updated_at. Estes campos são utilizados, respetivamente, para guardar datas de criação ou alteração de uma linha.

O Eloquent permite definir relações entre entidades, de 1 para 1, de 1 para N e de N para N, facilitando a obtenção de todos os objetos relacionados com uma determinada instância de uma entidade. Na figura acima podemos verificar que existem quatro métodos, cada um deles permitindo que a entidade Utilizador se relacione com outras entidades. O quadro seguinte explica a funcionalidade de cada um desses métodos:

Método	**Tipo de Relação**
departamento	Um utilizador pertence a um departamento; Um departamento é composto por vários utilizadores. (N para 1)
actividades	Um utilizador tem várias atividades mas uma atividade só pertence a um utilizador. (1 para N)
contacto	Cada utilizador tem um contacto e cada contacto só pertence a um utilizador. (1 para 1)
projetos	Um utilizador pode pertencer a vários projetos e um projeto pode ter vários utilizadores. (N para N)

Tabela 1 Exemplos de relacionamentos entre entidades.

Com estes métodos definidos, é possível obter os objetos relacionados. A próxima figura mostra como podemos utilizar um Modelo para obter dados da base de dados:

```
$contacto = Utilizador::find(1)->contacto; //Devolve Contacto associado
$actividades = Utilizador::find(1)->actividades; //Devolve um array de Actividade
```

Figura 7 Exemplo de utilização do Eloquent.

Embora, as funcionalidades relativas a relacionamentos que o Eloquent disponibiliza sejam muito úteis, existem alguns cuidados a ter quando as utilizamos. Um dos principais cuidados é a utilização de métodos de relacionamento dentro de ciclos. Nesse caso, pode existir um elevado número de *queries* à base de dados. O Eloquent disponibiliza funções que permitem contornar este problema, cuja utilização é ilustrada na figura seguinte.

```
//Bom Exemplo
$utilizadores = Utilizador::all(); // SELECT * FROM core_base_utilizadores
$utilizadores->load("departamento"); //SELECT * FROM authors where id_utilizador in (1, 2, 3, 4, 5, ...)
//Queries executadas antes do ciclo. Apenas duas queries necessárias.
foreach ($utilizadores as $utilizador) {
    echo $utilizador->departamento->nome;
}

//Mau Exemplo
$utilizadores = Utilizador::all(); // SELECT * FROM core_base_utilizadores
foreach ($utilizadores as $utilizador) {
    echo $utilizador->departamento->nome; //Queries executadas dentro do loop. Uma querie por cada utilizador
}
```

Figura 8 Bom e Mau exemplo do uso do Eloquent.

O código da figura contém dois ciclos foreach que permitem mostrar no ecrã o nome do departamento a que cada utilizador pertence. No primeiro caso, após obter todos os utilizadores, é utilizado o método load. Este método faz com que os departamentos de todos os utilizadores sejam obtidos com um único acesso à base de dados. Como o Eloquent disponibiliza um sistema de cache, não é necessário nenhum acesso à base de dados dentro do ciclo. No segundo caso, que ilustra uma má prática, não é utilizado o método load antes do foreach. Isto faz com que seja feito um acesso à base de dados em cada iteração do ciclo de modo a obter o departamento de cada utilizador, diminuindo o desempenho da execução do código.

A integração do Eloquent no g.Core tem vindo a ser realizada de forma progressiva e vai decorrer ainda algum tempo até que o método anterior de acesso à base de dados seja totalmente substituído. A base de dados é constituída por centenas de tabelas e será necessário criar todas as entidades envolvidas. Outro aspeto que tem atrasado a integração do Eloquent prende-se com o facto de o projeto (g.Core) estar já numa fase muito avançada, implicando um grande investimento de tempo neste processo.

g.Webservice

Um dos principais objetivos delineados para este estágio foi o de tornar o g.Core capaz de comunicar com aplicações exteriores, ou seja, com outros tipos de clientes para além do cliente web. O g.Core não foi projetado para suportar integração, não tendo uma camada de lógica de negócio bem definida. Este problema faz com que seja difícil desenvolver apenas uma camada de serviços que utilize como base a lógica de negócio já existente.

Analisando toda a arquitetura e estruturação do g.Core, optámos por desenvolver uma aplicação empresarial que disponibiliza as funcionalidades do g.Core em forma de webservice, de nome g.Webservice. De início, não foram definidas todas as funcionalidades e recursos que o g.Webservice iria disponibilizar, sendo apenas definidas as funcionalidades que iam ser necessárias para o desenvolvimento das aplicações referentes a este estágio.

O g.Webservice é uma nova aplicação, com uma nova lógica de negócio e apresentação mas que utiliza a mesma base de dados que a aplicação g.Core. Como a base de dados é partilhada por estas duas aplicações, optou-se pela reutilização da camada de acesso a dados e modelo de domínio do g.Core. Esta solução só se tornou possível devido à utilização da mesma linguagem. Dessa forma, é possível a partilha de uma biblioteca com funcionalidades de acesso a dados, evitando-se a duplicação do modelo de domínio e das suas respetivas classes.

Ao desenvolver uma aplicação empresarial, um dos aspetos mais importantes a ter em conta é a forma de comunicação entre esta e os seus clientes. Na aplicação g.Webservice esse aspeto era muito importante, já que esta deveria suportar diversos tipos de aplicações cliente. Estas aplicações podem ser do tipo *standalone* presentes em equipamentos com "muito" poder de processamento, ou aplicações móveis, normalmente presentes em dispositivos com pouco poder de processamento e com tráfego de internet limitado. Este requisito fez com que se procurasse adotar uma solução que fornecesse um protocolo de comunicação com pouco overhead de sintaxe, escalável e de rápido processamento.

Existem dois tipos principais de webservices: os que são baseados em REpresentational State Transfer (REST) e os que são baseados em Simple Object Access Protocol (SOAP) [36], cada um com as suas particularidades. Após estudo e análise de cada uma destas abordagens, optou-se pelo uso de webservices REST. Esta escolha deve-se à sua maior portabilidade e ao facto de suportar vários formatos de resposta, ao contrário do SOAP que apenas suporta XML. Apesar de o REST permitir vários formatos de resposta, existem dois que são os mais utilizados e suportados, o JavaScript Object Notation (JSON) e o eXtensible Markup Language (XML). O JSON foi o formato escolhido para a troca de informação entre o g.Webservice e os seus clientes. As razões que levaram a essa escolha foram o baixo overhead de sintaxe e rápido processamento quando comparado com o XML, fatores que são importantes se os clientes forem aplicações móveis [37].

5.1 Arquitetura

Para o desenvolvimento do g.Webservice propusemos a utilização de uma framework de forma a facilitar o desenvolvimento e estruturação da aplicação. A proposta foi bem recebida pelo grupo de trabalho. O Laravel [38] foi a framework escolhida para ser utilizada como base de desenvolvimento do g.Webservice. Esta framework é uma das mais utilizadas e está em constante desenvolvimento. Conta ainda com boa documentação e uma comunidade vasta de utilizadores. O Laravel prevê o uso do padrão MVC, permitindo criar aplicações com uma arquitetura bem definida. As funcionalidades fornecidas pelo uso desta framework são inúmeras. No contexto de desenvolvimento de um webservice REST, o Laravel suporta a maioria das funcionalidades necessárias, entre elas a fácil criação de caminhos e controladores, o suporte aos métodos HTTP e possibilidade do uso de filtros (ou Middlewares). O Laravel inclui ainda uma biblioteca de ORM, o Eloquent, a mesma biblioteca de ORM que foi adicionada ao projeto g.Core.

O g.Webservice é uma aplicação que usa como base a framework PHP Laravel. Usando esta *framework*, o g.Webservice foi construído tendo como base o padrão de desenho MVC seguindo, assim, a estrutura que este padrão sugere.

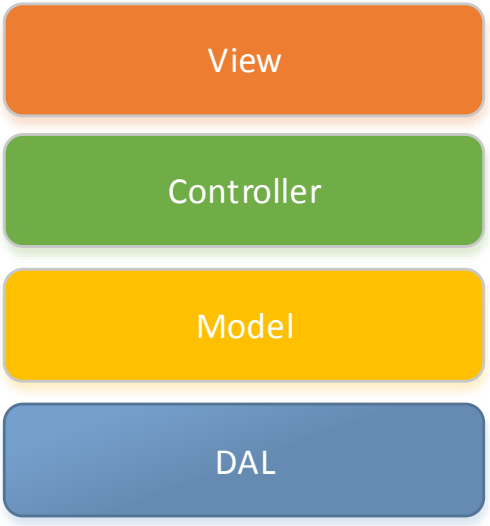

Figura 9 Arquitetura do g.Webservice.

Na figura acima está representado o modelo MVC onde se podem identificar as camadas de apresentação (View), controlo (Controller) e modelo de dados (Model). A estas, foi ainda adicionada uma camada de acesso a dados (Data Access Layer - DAL).

A camada de apresentação, no caso do g.Webservice, é responsável pela entrada e saída de informação. São suportados os métodos HTTP GET, POST, PUT e DELETE para satisfazer os diversos tipos de pedidos. Como se trata de um webservice, não são disponibilizadas páginas web, com o tradicional HTML, mas sim mensagens em formato JSON, podendo estas ser consumidas por aplicações externas.

A camada de controlo é composta por controladores que tratam de toda a lógica de negócio da aplicação. Existem vários controladores, com funcionalidades distintas. Cada controlador é composto por um conjunto de métodos que são executados mediante o caminho e método HTTP utilizados pelo cliente. Estes métodos contêm a lógica de negócio da aplicação e são responsáveis pela interação com as classes do modelo de domínio.

A camada de modelo é constituída pelo modelo de domínio que tem como objetivo interagir com a base de dados. A aplicação g.Webservice utiliza o mesmo modelo de domínio e respetivas classes já implementadas no projeto g.Core.

Por último a camada DAL, uma camada adicionada pela equipa de trabalho de forma a facilitar a interação com a base de dados. Esta camada é constituída por um ORM, o Eloquent, que faz com que o modelo de domínio abstraia a base de dados, facilitando assim o acesso. Tal como vimos no capítulo anterior, esta camada também já se encontra implementada no projeto g.Core.

O esquema seguinte mostra de uma forma mais detalhada o funcionamento da aplicação usando o Laravel:

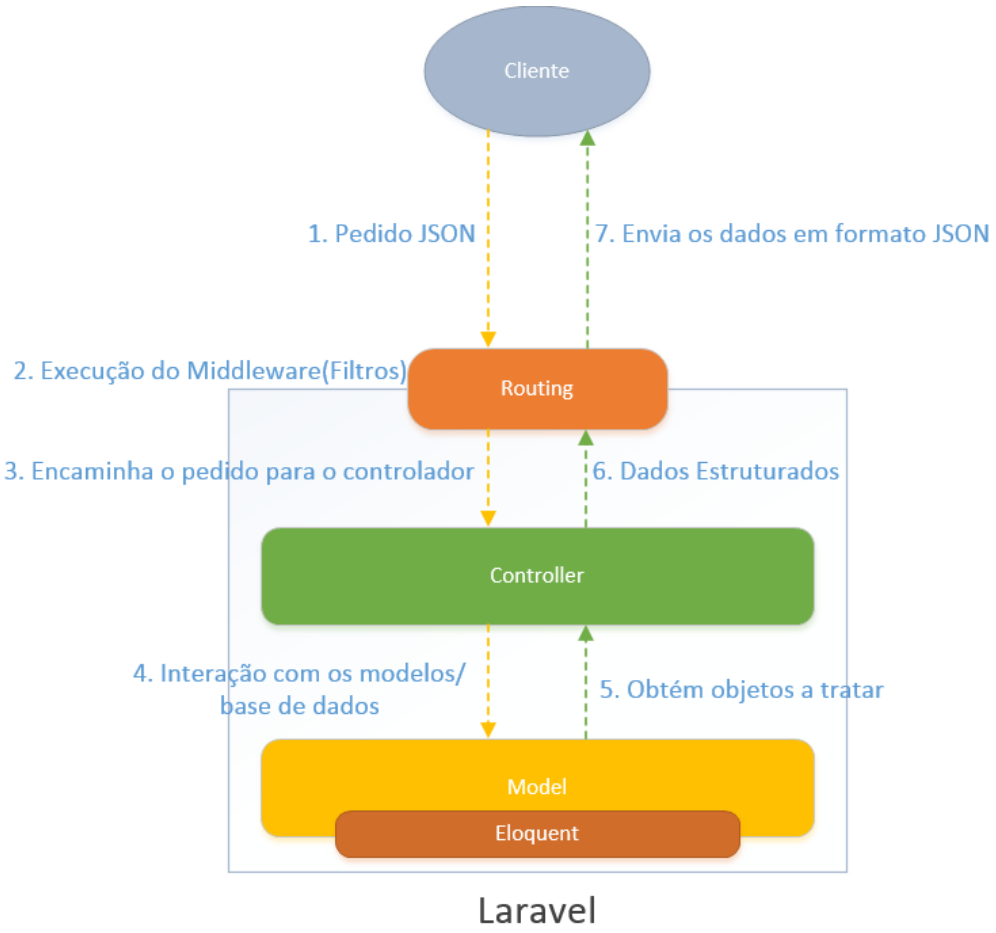

Figura 10 Fluxo de um pedido ao g.Webservice.

O cliente faz um pedido JSON ao serviço g.Webservicse através de um endereço (ex: api.grifincore.com/crm/activities). A camada de routing/view, que representa a camada de apresentação, ao receber o pedido, analisa o endereço e o método HTTP e associa-o a um controlador. Caso esse controlador tenha um filtro (ou Middleware) [39] associado, executa-o antes de encaminhar o pedido para o respetivo controlador. Os filtros são uma funcionalidade do Laravel que permite executar funções antes ou depois da execução do controlador. Exemplos de utilização de filtros são a verificação de sessão ou a verificação de permissões para aceder a determinado método. Após a execução do filtro, o controlador realiza toda a lógica de negócio correspondente ao pedido, interagindo com as classes do modelo. Quando necessário, o ORM Eloquent acede à base de dados devolvendo instâncias das classes pertencentes ao modelo que podem ser manipuladas pelo controlador. De seguida, a camada de apresentação, trata de transformar essas instâncias em formato JSON e envia-as como resposta ao cliente.

5.2 HTTP, Routing e Controllers

Um webservice tem como principal objetivo responder a pedidos de aplicações cliente. No caso de serviços HTTP, tem de responder aos pedidos consoante o caminho e o método http solicitado pelo cliente. O Laravel dispõe de um sistema de routing e controladores que são responsáveis por satisfazer os pedidos, facilitando assim a associação do pedido com o código a executar. O g.Webservice tem como objetivo fornecer, em forma de Web API, algumas funcionalidades presentes na aplicação g.Core. Nesse sentido, foi necessária a criação de uma estrutura de caminhos (routes) que permitissem uma divisão lógica entre os vários módulos e funcionalidades. Seguindo como exemplo o pedido de todas as Atividades, uma vez que estas pertencem ao módulo CRM, o endereço para fazer este pedido tem a estrutura [endereço do servidor]/api/CRM/activities. Esta estrutura de caminhos, [endereço do servidor]/api/[modulo]/[recurso], faz com que os caminhos estejam logicamente bem divididos.

Metodo HTTP	Caminho (URL)	Ação (Método)
GET	/ activities	índex
POST	/ activities	Store
PUT/PATCH	/ activities /{id}	update
DELETE	/ activities /{id}	destroy

Tabela 2 Exemplo de métodos disponibilizados por um Resource Controller.

Quando um cliente faz um pedido, o sistema de routing decide o controlador ao qual o pedido vai ser encaminhado. Este encaminhamento depende do método HTTP e do caminho do endereço associado a esse pedido. No Laravel é possível associar caminhos a controladores de modo a que a *framework* mapeie automaticamente os métodos índex, store, update e destroy presentes nesses controladores. A esses controladores dá-se o nome de Resource Controllers [40]. Um Resource Controller é um tipo de controlador que apenas mapeia automaticamente os métodos de Create, Read, Update e Delete (CRUD). É possível criar outros métodos no controlador, mas o mapeamento tem de ser realizado manualmente.

A Framework Laravel permite ainda criar controladores associados a funcionalidades específicas definidas pelo programador, os RESTful Controllers. Os RESTful Controllers mapeiam todos os métodos declarados no controlador, desde que o seu nome siga uma determinada estrutura. O nome deve começar com a designação do método HTTP seguido do nome pretendido para o método.

Metodo HTTP	**Caminho (URL)**	**Ação (Método)**
POST	/reports/close	postCloseReport
GET	/ reports /finished	getFinished

Tabela 3 Exemplo de métodos disponibilizados por um Restfull Controller.

Este tipo de controlador é mais flexível podendo ser usado preferencialmente em controladores que não necessitam de métodos CRUD.

O g.Webservice utiliza ambos os tipos de controladores, *Resource Controllers* e *RESTful Controllers*. Se o objetivo for disponibilizar operações de CRUD, opta-se pelos *Resource Controllers*. Caso o objetivo seja executar apenas funcionalidades não associadas a ações CRUD, utilizam-se os *RESTful Controllers*.

Ainda no que diz respeito ao *routing*, o Laravel permite aplicar filtros a caminhos. Os filtros podem ser usados por exemplo para a implementação de um sistema de segurança, como aconteceu na implementação do g.Webservice, para o desenvolvimento do OAuth 2.0.

5.3 Segurança – OAuth 2.0

A segurança é um dos fatores mais importantes a ter em conta na criação de aplicações web. Quando a aplicação web consiste num serviço exposto, como o g.Webservice, esse fator torna-se ainda mais importante. De forma a assegurar a segurança dos acessos ao g.Webservice, foi desenvolvido um método de autenticação para que só os utilizadores autenticados possam aceder aos conteúdos disponibilizados. Foi nesse sentido que foi implementado o protocolo de autenticação *OAuth 2.0*, descrito na RFC 6749 [41], utilizado

por muitos serviços. Para melhor compreensão segue-se uma listagem de alguns termos importantes utilizados no contexto do OAuth 2.0:

- ***Access token*:** Token utilizado para aceder aos conteúdos autorizados**.**
- ***Resource Owner* (Proprietário dos dados):** Entidade capaz de obter acesso aos conteúdos protegidos. Esta entidade pode ser um utilizador ou até uma aplicação.
- ***Client* (Cliente):** Aplicação que pretende ter acesso a dados pertencentes a um determinado proprietário.
- ***Authorization code*:** Token intermediário que permite ao cliente obter o *Access Token.*
- ***Authorization Grant*:** Refere-se ao processo que permite obter um *Access Token*.

O OAuth 2.0 é a segunda versão do protocolo OAuth que prevê um conjunto de regras para que uma aplicação *third-party* obtenha acesso limitado a um serviço HTTP. O OAuth 2.0 usa um sistema de *tokens* que expiram passado algum tempo. A aplicação cliente pede autorização de acesso à aplicação empresarial enviando um *authorization grant* e esta devolve-lhe um *Access Token* associado. Um *authorization grant* consiste num conjunto de informações que o cliente tem de enviar ao servidor de autenticação para que este lhe dê acesso ao conteúdo. O *authorization grant* enviado pelo cliente pode ser obtido de várias formas como por exemplo:

- ***Resource Owner Password Credentials*:** Este método utiliza as credenciais do utilizador (*username* + *password*) para obter um *token* que permite o acesso aos recursos. Este método só deve ser utilizado quando existe um grande grau de confiança entre o proprietário do recurso e o cliente que lhe tenta aceder. Este método é utilizado pelas aplicações móveis desenvolvidas no âmbito deste estágio.
- ***Client Credentials*:** A aplicação cliente tem acesso a conteúdos protegidos enviando por exemplo um *Client Id* e um *Client Secret*. Este método pode ser utilizado em casos em que a aplicação cliente necessita de aceder a conteúdos protegidos que apenas dizem respeito à aplicação. Este método é utilizado pela aplicação relógio de Ponto, desenvolvida no âmbito deste estágio e descrita no próximo capítulo.
- ***Refresh Token:*** *Token* fornecido pelo servidor de autenticação quando o *Access Token* é pedido. Quando o *Access Token* expira, o *Refresh Token* pode ser utilizado para obter um novo *Access Token*.

Os três métodos de autorização referidos acima são os disponibilizados pelo g.Webservice para que aplicações externas possam aceder a dados protegidos. O esquema seguinte ilustra a forma como o protocolo OAuth 2.0 foi implementado no g.Webservice.

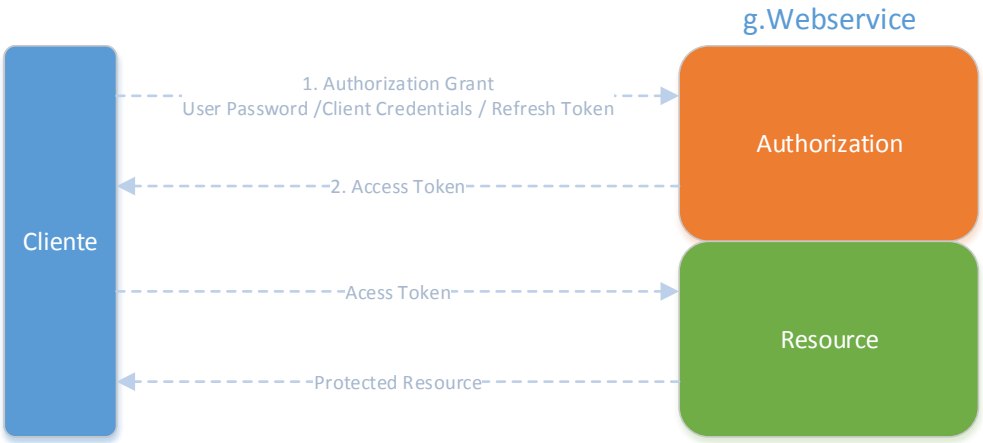

Tabela 4 Processo de autenticação de um cliente.

A segurança do g.Webservice está dividida em duas etapas: *Authorization* e *Resource*. A *Authorization* é responsável pela gestão dos pedidos de acesso. A etapa *Resource* trata de toda a lógica de negócio da aplicação, só disponibilizando o acesso aos utilizadores com *tokens* válidos.

O processo de autenticação no servidor começa com o envio do *Authorization Grant* por parte da aplicação cliente. O g.Webservice verifica se os dados enviados pelo cliente são válidos. Caso sejam válidos o g.Webservice envia ao cliente um *token* e o respetivo *refresh token*.

A imagem seguinte é um exemplo de uma resposta de um pedido de acesso bem-sucedido em que o serviço envia como resposta um objeto JSON que contém todas as credenciais de acesso.

```
HTTP/1.1 200 OK
     Content-Type: application/json;charset=UTF-8
     Cache-Control: no-store
     Pragma: no-cache

     {
       "access_token":"2YotnFZFEjr1zCsicMWpAA",
       "expires_in":3600,
       "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
     }
```

Tabela 5 Exemplo de um token gerado pelo OAuth 2.0.

O objeto JSON é constituído por:

- ***access_token*:** chave usada para realizar pedidos de conteúdos ao serviço;
- ***expires_in*:** duração do *token*;
- ***refresh_token*:** chave que pode ser ou não usada para pedir um novo *access token* ao serviço.

Podem ser definidos atributos extras nesta resposta de forma a aumentar a complexidade da autenticação.

Depois de o cliente obter *o access token*, pode então pedir conteúdos ao g.Webservice enviando *o access token* em cada pedido realizado. Quando *o access tok*en já não for válido, o serviço envia uma resposta a indicar que este expirou. Depois, o cliente só voltará a conseguir fazer pedidos após obter um nov*o access token*.

O protocolo OAuth2.0 tem como principal objetivo proteger a informação ao nível da autorização/autenticação, só utilizadores autorizados e autenticados podem aceder à informação. Para este protocolo funcionar de forma mais segura e confidencial possível, deverá ser utilizado em conjunto com outro protocolo HTTPS. Com o HTTPS protegemos a nossa informação de ataques *men in the middle*, garantindo a confidencialidade da informação. Por omissão, o g.Core, e consequentemente o g.Webservice, não utiliza o protocolo HTTPS mas, caso o cliente assim o pretenda, esse protocolo é ativado, sendo o custo dos certificados por si suportados.

5.4 Modelo de Domínio

O g.Webservice utiliza o modelo de domínio que já existia definido no projeto g.Core. No entanto, foi necessário proceder a uma alteração nesse modelo no âmbito do desenvolvimento do g.Webservice devido à implementação do OAuth 2.0. Foi necessário adicionar novas tabelas para gerir as aplicações clientes, os utilizadores e os *tokens* associados a cada pedido de acesso. A imagem seguinte ilustra uma pequena porção do modelo de domínio do g.Webservice, correspondente às entidades criadas para a implementação do OAuth 2.0.

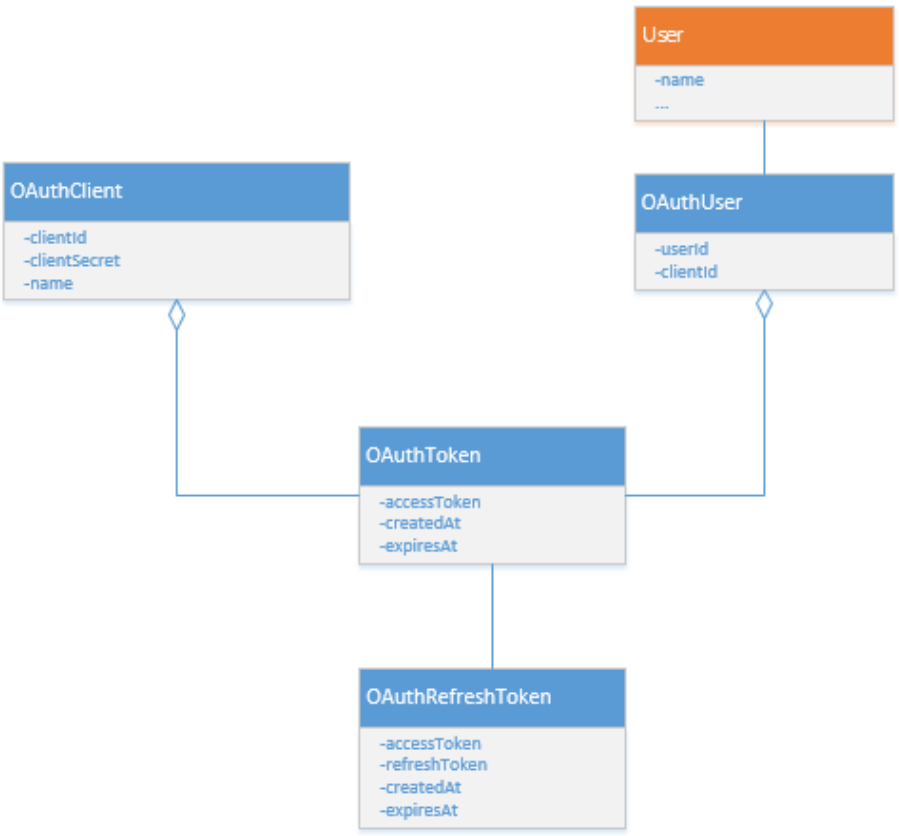

Figura 11 Modelo de Domínio OAuth 2.0 – g.Webservice.

O modelo de domínio do OAuth 2.0 implementado no g.Webservice representado acima é composto pelas seguintes entidades:

- **OAuthClient:** Entidade que representa uma aplicação cliente com autorização de acesso ao *webservice*.
- **OAuthUser:** Entidade que representa um utilizador que se pode autenticar no *webservice*. Esta entidade referencia a entidade User que representa um utilizador do g.Core. O AuthUser também referencia a entidade OAuthClient já que um utilizador tem sempre que estar associado a uma aplicação cliente para se poder autenticar na aplicação.
- **OAuthToken:** Entidade que representa um *token* gerado para um utilizador ou cliente.
- **OAuthRefreshToken:** Entidade que representa um *refresh token* associado a um *access token*.

5.5 Tecnologias

O g.Webservice foi desenvolvido com a linguagem de *scripting* PHP, já utilizada no desenvolvimento da aplicação g.Core. A utilização desta linguagem foi definida com um dos requisitos de desenvolvimento do projeto. Este requisito deve-se ao facto de o servidor suportar apenas a linguagem PHP e também porque a utilização da mesma linguagem utilizada no desenvolvimento do g.Core traz benefícios no que toca à reutilização de código. O NetBeans [42] foi o Integrated Development Environment (IDE) escolhido uma vez que tínhamos já experiência no desenvolvimento de aplicações com esta ferramenta e que a Grifin deixou esta escolha ao nosso critério. Este IDE suporta multilinguagens e inclui várias ferramentas que facilitam o desenvolvimento de projetos.

Relógio de Ponto

A aplicação "Relógio de Ponto" (RP) é uma aplicação *stantalone* de assiduidade e pontualidade que permite o registo de movimentos de entrada e saída dos utilizadores de uma empresa. Esta aplicação faz parte do módulo g.Point, módulo de gestão de assiduidade e pontualidade. Ao início deste estágio já existia uma versão do RP em produção, no entanto pouco funcional, pelo que a empresa Grifin delineou para este estágio a criação de uma nova versão.

A versão já existente deste *software* permitia o registo de movimentos através de um leitor biométrico. No entanto, tinha algumas limitações, nomeadamente, o facto de o utilizador ter de inserir o seu número de identificação do sistema (g.Core), através de um NumPad, e só posteriormente, poder fazer o registo biométrico. Este método tornava o produto pouco funcional e produtivo já que cada registo demorava algum tempo. Outro aspeto menos positivo desta versão do RP estava relacionado com o facto de a ligação à base de dados do g.Core para o envio da informação sobre os movimentos ser realizada remotamente via Internet. O RP periodicamente verificava se existiam registos por enviar e, caso existissem, enviava diretamente para a base de dados do g.Core fazendo uma ligação pela Internet à base de dados do g.Core.

Dadas as limitações da versão existente, foi proposto pela empresa o desenvolvimento de uma nova versão do "Relógio de Ponto" que resolvesse os problemas existentes e acrescentasse novas funcionalidades. Como a primeira versão não foi desenvolvida pela Grifin, não foi possível ter acesso ao seu código fonte.

Para o novo relógio de ponto definiram-se os seguintes requisitos:

- Registo de movimentos (entradas/saídas) através do leitor biométrico *Digital Persona®*;
- Guardar todos os movimentos localmente;
- Sincronizar movimentos com o g.Core através do g.Webservice;
- Permitir a associação do utilizador a uma chave biométrica;
- Sincronização das chaves biométricas entre diversos RP;
- Registo de movimento com a inserção do identificador do utilizador e *password*;
- Sincronização de utilizadores g.Core;
- Criação de novo utilizador do Relógio de ponto;
- Fácil instalação.

Desenvolvida no âmbito do estágio, o novo "Relógio de Ponto" é uma aplicação onde os utilizadores registam os seus movimentos na empresa através da leitura da sua impressão digital ou de inserção de credenciais. Posteriormente, todos os registos são enviados para o g.Core através do g.Webservice, sendo este a fazer a análise dos movimentos efetuados pelos utilizadores, classificando-os como entrada ou saída. A imagem seguinte esquematiza a arquitetura de alto nível da aplicação:

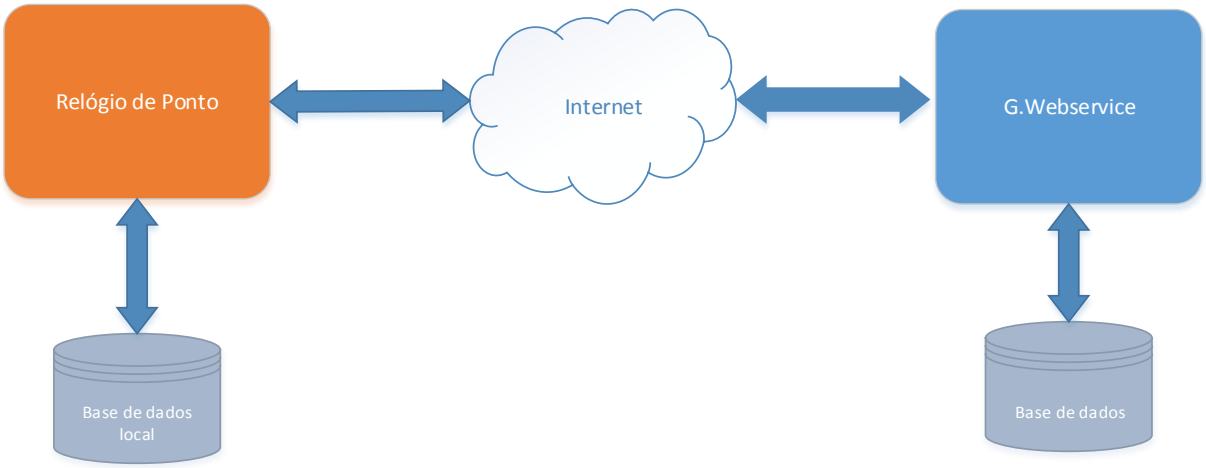

Figura 12 Arquitetura de comunicação Relógio de ponto.

Como se pode ver na figura acima, esta nova versão do RP utiliza uma Base de Dados local para guardar todos os registos dos movimentos efetuados, não ficando assim dependente da ligação *ao* webservice para funcionar.

6.1 Arquitetura

A Aplicação Relógio de Ponto tem uma arquitetura lógica de multicamada orientada a serviços, tendo o g.Webservice como principal fonte de dados. A arquitetura do RP define-se por três camadas diferentes, Apresentação, Lógica de Negócio e Dados.

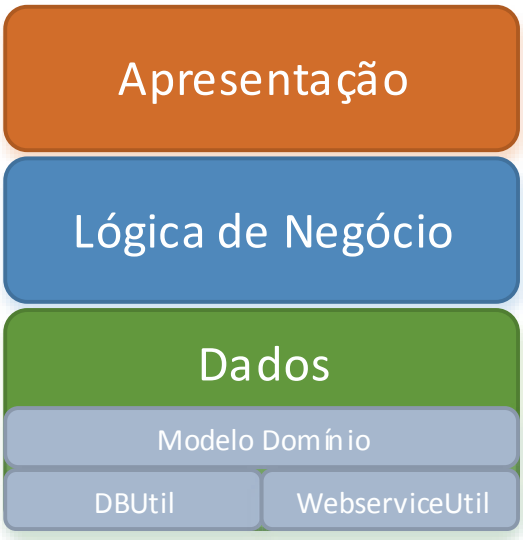

Figura 13 Arquitetura de Software do Relógio de Ponto.

A camada de apresentação é responsável pelo conjunto de componentes visuais que permite que os utilizadores dos relógios de ponto possam interagir com o g.Point. Desenvolvida utilizando a ferramenta SWING [43], biblioteca java para desenhar interfaces, a camada de apresentação é responsável por comunicar com a camada de lógica de negócio. Nesta aplicação toda a Lógica de Negócio executada através da camada de apresentação está implementada no *event handler* do controlador. No entanto, nem todas as ações de lógica de negócio são executadas pela camada de apresentação. O RP está preparado para suportar um leitor biométrico Digital Persona® que funciona como interface de entrada do sistema, ou seja, a lógica de negócio de verificação da leitura biométrica é executada quando o utilizador coloca o dedo no leitor biométrico.

A camada de dados é responsável pelo acesso a dados e é composta pelo Modelo de Domínio, e dois utilitários de acesso a dados, o DBUtil e o WebserviceUtil. O DBUtils, explicado na secção 6.6.1, é responsável pelos acessos à base de dados local enquanto o WebserviceUtil, explicado na secção 6.6.2, contém um conjunto de métodos para o acesso a*o* webservice.

6.1.1 Base de Dados

Na aplicação Relógio de Ponto há a necessidade de guardar a informação dos utilizadores e das leituras localmente, o que permite que a aplicação trabalhe *offline*, não ficando

dependente do webservice. Ao contrário da versão anterior, o novo RP não necessita de um *software* de gestão de base de dados (SGBD), como o MYSQL, para que a base de dados esteja acessível à aplicação. Isto deve-se à utilização de uma base de dados SQLITE que é composta por um ficheiro que contém todas as tabelas e registos, sendo possível acedê-los utilizando apenas uma biblioteca SQLITE. Essa biblioteca permite aceder à base de dados diretamente, podendo realizar transações sem a necessidade de um SGBD instalado no dispositivo. Embora o SQLITE tenha menos desempenho e funcionalidades que outras bases de dados (ex: MYSQL) é muito mais portável, o que para esta aplicação é muito importante já que facilita muito a sua instalação e configuração.

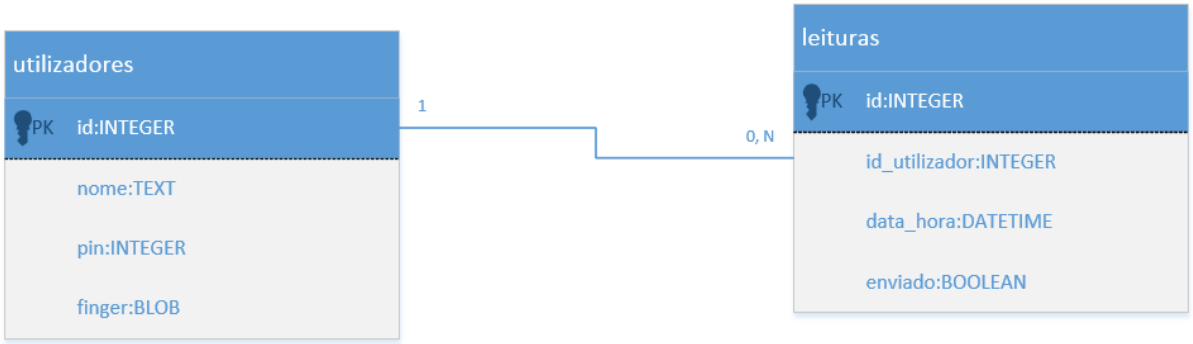

Figura 14 Modelo de base de dados do Relógio de Ponto.

A figura anterior representa o diagrama da base de dados da aplicação Relógio de Ponto. A base de dados desta aplicação é composta por duas tabelas, utilizadores e leituras. A tabela utilizadores guarda a informação de todos os utilizadores que registam entrada no relógio de ponto. É na tabela utilizadores que é guardado em forma de BLOB o registo biométrico de forma a ser utilizado posteriormente pela aplicação. A tabela utilizadores relaciona-se com a tabela leituras de modo a que um utilizador possa ter várias leituras associadas. Na tabela leituras são guardados todos os movimentos registados por um utilizador através da aplicação RP. Cada registo contém a data e hora do movimento, o id do utilizador bem como o campo enviado que indica se a leitura já foi enviada para o *webservice*.

Tal como referimos acima, foi também desenvolvida a classe DBUtils que contém um conjunto de métodos estáticos que permitem interagir com as tabelas. A DBUtil utiliza a biblioteca SQLite JDBC para aceder à base de dados. Esta biblioteca tem um conjunto de funcionalidades, como o suporte a *preparement* statement, permitindo aceder à base de dados de forma fácil e segura.

6.1.2 *Webservice*

O RP é uma aplicação que integra com a aplicação g.Core, mais concretamente, com o módulo g.Point. Sem a aplicação RP o módulo g.Point não tem a informação necessária que permita fazer o controlo de assiduidade e pontualidade dos funcionários de uma empresa. Na versão anterior do RP, a integração era feita através de uma ligação direta à base de dados do g.Core e, sendo esse método pouco seguro, decidiu-se utilizar o g.Webservice para a nova implementação do RP.

Para uma fácil comunicação entre o RP e o g.Webservice foi desenvolvida uma biblioteca em JAVA denominada de WebserviceUtil. Esta biblioteca tem um conjunto de funcionalidades que facilitam a realização de pedidos HTTP ao g.Webservice, tratando da resposta e de possíveis erros.

A WebserviceUtil utiliza a biblioteca Apache *HttpComponents* e a biblioteca GSON [44]. A biblioteca Apache *HttpComponents,* desenvolvida pela *Apache Software Foundation*, fornece um conjunto de funcionalidades que facilitam a realização de pedidos HTTP como, por exemplo, o suporte a todos os métodos HTTP, definição de *Headers, etc*. A biblioteca GSON, desenvolvida pela Google, é uma biblioteca JAVA que permite converter JSON em objetos JAVA e vice-versa.

Utilizando as bibliotecas *HttpComponents* e *GSON*, a WebserviceUtil disponibiliza métodos que permitem fazer pedidos HTTP a um servidor. Para cada método HTTP é disponibilizada uma função que devolve um tipo genérico, convertendo a resposta JSON para o objeto definido.

```
private static <T> T postRequestType( String url, List<NameValuePair> nameValuePairs, Type listType)
{
    HttpClient httpclient = WebserviceUtils.getNewHttpClient();
    HttpPost httppost = new HttpPost(url);

    HttpParams httpParameters = new BasicHttpParams();
    int timeoutConnection = 3000;
    HttpConnectionParams.setConnectionTimeout(httpParameters, timeoutConnection);
    int timeoutSocket = 5000;
    HttpConnectionParams.setSoTimeout(httpParameters, timeoutSocket);
    httppost.setParams(httpParameters);
    String json  ;
    httppost.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    HttpResponse response = httpclient.execute(httppost);
    HttpEntity entity = response.getEntity();
    InputStream instream = entity.getContent();
    json = WebserviceUtils.convertStreamToString(instream);



    T data = GsonUtils.getInstace().fromJson(json, listType);

    return data;
}
```

Figura 15 Método postRequestType do WebserviceUtil.

Na Figura 15 mostra-se um exemplo de um método disponibilizado pelo *WebserviceUtils* que permite fazer pedidos POST a um servidor. Este método recebe como parâmetros o endereço

do servidor, uma *List<NameValuePair>* de parâmetros a enviar e uma variável *Type* que permite à biblioteca *GSON* converter o *JSON* recebido para o objeto correto. Como se pode observar, o método representado na Figura 15 devolve um objeto genérico. Dessa forma é possível utilizar o mesmo método para diversos pedidos. O *WebserviceUtils* disponibiliza também métodos que permitem fazer pedidos GET, PUT e DELETE, funcionando de maneira semelhante ao representado na Figura 15.

Todos os pedidos são realizados utilizando o serviço de autenticação OAuth 2.0 que o g.Webservice disponibiliza. A aplicação relógio de ponto utiliza o método de autenticação *Client Credentials* onde é necessário enviar um *client id* e um *client secret* válidos para obter o *token* acesso ao g.Webservice. O *client id* e um *client secret* são configurados na aplicação na área de administração.

6.2 Modelo de Domínio

O Relógio de Ponto é composto por um modelo de domínio simples de apenas duas entidades: Utilizador e Leituras. O seguinte diagrama de classes demonstra como estas duas entidades se relacionam.

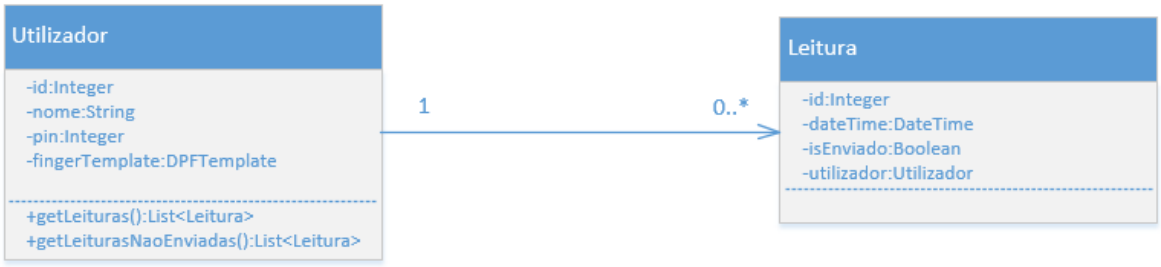

Figura 16 Modelo de domínio do Relógio de Ponto

A entidade Utilizador representa um determinado utilizador do sistema que é constituída pelas seguintes campos:

- id: Número inteiro, único, que identifica o utilizador no sistema g.Core;
- nome: Nome do utilizador;
- pin: Código utilizado para fazer login, caso o leitor biométrico esteja indisponível;
- fingerTemplate: Objeto que contém a informação biométrica do utilizador.

A entidade utilizador relaciona-se com a entidade Leitura, onde cada utilizador pode ter zero ou mais Leituras associadas. A entidade Leitura contém os dados referentes a um movimento, entrada ou saída, registados no RP. Esta entidade é constituída pelos seguintes campos:

- id: Número único que identifica uma determinada Leitura;
- dateTime: A data e hora em que foi efetuado o movimento;

- isEnviado: Campo booleano que indica se a leitura já foi enviada para o servidor;
- utilizador: Instância do Utilizador a que pertence a Leitura.

6.3 Tecnologias Utilizadas

O projeto relógio de ponto foi desenvolvido na linguagem Java utilizando a plataforma de desenvolvimento Java Development Kit (JDK) 7 desenvolvida pela ORACLE em conjunto com o IDE Netbeans. O facto de a aplicação ser desenvolvida em JAVA utilizando o JDK permite que a aplicação seja cross-plataform, podendo ser executada em todos os sistemas operativos que suportem a Java Virtual Machine (JVM). Desta forma, a instalação do Relógio de Ponto não fica confinada a um sistema operativo podendo esta ser instalada num sistema low-cost, como por exemplo um Raspberry Pi com um sistema operativo Linux instalado. A escolha do NetBeans deveu-se principalmente ao facto de este já conter bibliotecas necessárias para desenvolver uma aplicação *desktop*, entre elas, uma de desenvolvimento de interfaces gráficas em Java, como é o exemplo da biblioteca SWING.

A utilização do dispositivo leitor biométrico Digital Persona obrigou ao estudo da API associada. Este dispositivo tem uma API bem documentada e com bons exemplos. A utilização como sucesso desta tecnologia foi decisiva para o desenvolvimento deste trabalho.

6.4 Funcionamento

O funcionamento da aplicação Relógio de Ponto pode ser dividido em três fases: a configuração, registo e verificação. A configuração são todas as ações iniciais ou de suporte desempenhadas pela Grifin para que a aplicação funcione corretamente. Todas essas configurações são realizadas através da área de administração da aplicação. Após a configuração inicial, pode haver a necessidade dos utilizadores se registarem no relógio de ponto, permitindo que os utilizadores posteriormente possam efetuar o registo de movimentos através da verificação dos dados de utilizador.

Para permitir o registo biométrico é necessário *hardware* específico. Nesse sentido, a Grifin forneceu-nos um equipamento que permite fazer este tipo de registo. Esse equipamento é o leitor biométrico de impressões digitais de nome *U.are.U 4500* [45], produzido pela Digital Persona, empresa que tem várias soluções deste género.

Figura 17 Leitor biométrico U.are.U 4500.

Para este equipamento, a Digital Persona, disponibiliza um SDK para várias linguagens de programação entre elas a linguagem JAVA, utilizada para desenvolver a aplicação Relógio de Ponto.

6.4.1 Configuração

Após a instalação da aplicação Relógio de Ponto é necessário proceder à configuração deste *software* para que este consiga comunicar com o g.Webservice. Este passo é realizado por um técnico da empresa Grifin através de uma área de administração que permite a configuração bem como a gestão dos utilizadores registados.

Para entrar na área de administração do *software* basta iniciar a aplicação e fazer duplo clique no logotipo da Grifin, presente no canto inferior direito.

Figura 18 Ecrã de autenticação para a área de administração.

A área de administração é protegida por uma credencial de *login* e *password* para que apenas pessoas autorizadas a ela tenham acesso.

Após a inserção de credenciais válidas, a aplicação mostra-nos o ecrã principal da área de administração. No menu superior existe um sub-menu de opções onde é possível definir as configurações de acesso ao g.Webservice. No ecrã de configuração é necessário colocar o endereço para o g.Core do cliente, bem como as credenciais que permitem que a aplicação se autentique no g.Webservice, o *client id* e o *client secret*.

Figura 19 Ecrã de principal de configuração.

6.4.2 Registo

Com o acesso ao g.Webservice configurado, o RP já consegue comunicar com aquele conseguindo assim obter todos os utilizadores registados no g.Core do cliente. Caso os utilizadores ainda não tenham sido registados, é necessário então proceder a esse registo, registando os seus dados biométricos.

Figura 20 Registo de impressão digital.

Para iniciar o processo de registo é necessário selecionar o utilizador que se pretende registar e depois clicar no botão “Registo Biométrico”, surgindo depois um ecrã para proceder à captura da impressão digital.

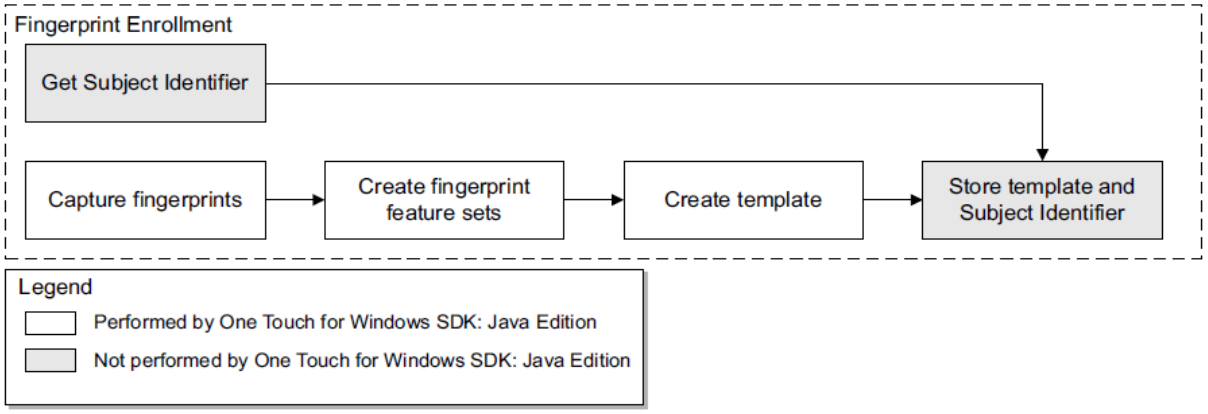

Figura 21 Esquema de registo da impressão digital [46].

O processo de captura da impressão digital é uma funcionalidade disponibilizada pelo SDK Digital Persona®, tratando da captura das impressões digitais, transformando-as em modelos, passiveis de serem utilizados no contexto aplicacional e serem guardadas em base de dados.

O SDK Digital Persona® fornece uma classe denominada DPFPEnrollmentControl para o Swing que trata de todo o processo de captura de uma nova impressão digital. A DPFPEnrollmentControl é uma componente gráfica que estende de um JPanel podendo ser utilizada como um componente visual Swing. Este componente permite ao utilizador escolher que dedos pretende registar pedindo posteriormente para o utilizador passar o dedo no *scanner*, obtendo assim a impressão digital. A classe DPFPEnrollmentControl tem definido um método *addEnrollmentListener* que recebe um objeto que implemente a interface *DPFPEnrollmentListener* que, com a implementação do método *fingerEnrolled*, permite tratar da impressão digital. Este método recebe como parâmetro um objeto do tipo *DPFPTemplate*, objeto esse que representa o modelo da impressão digital.

Já com o modelo de impressão digital, *DPFPTemplate*, a aplicação RP associa então o modelo da impressão digital ao utilizador que foi selecionado para realizar o registo. Posteriormente, e para finalizar este processo, é guardado o modelo da impressão digital na tabela utilizadores da base de dados no campo firgerTemplate associado ao utilizador selecionado. De notar que para facilitar a gravação e leitura de objetos *DPFPTemplate* em base de dados, este objeto disponibiliza o método *serialize()* que permite transformar o modelo em *byte[]* e o método *deserialize()* que permite transformar *byte[]* num modelo.

6.4.3 Verificação

A verificação é a ação que permite a um determinado utilizador registar uma entrada/saída, comparando se um dado biométrico inserido corresponde a algum já inserido na aplicação RP. O *software* Relógio de Ponto permite realizar esse registo de duas formas: por registo biométrico ou por introdução de credenciais do utilizador.

Figura 22 Ecrã principal - Registo biométrico.

Quando o leitor biométrico está configurado e a funcionar, a aplicação apresenta o ecrã apenas informativo, ficando à escuta de um novo registo biométrico. O SDK disponibiliza a interface DPFPDataListener que através do método *dataAcquired(DPFPTemplate)*, sempre que um utilizador tenta efetuar o registo de entrada através do leitor de impressão digital, recebe o *DPFPTemplate* gerado nessa tentativa de registo. Posteriormente a aplicação compara o objeto *DPFPTemplate* com os vários já registados no sistema. Todas as impressões guardadas em base de dados são carregadas para um *ArrayList<DPFPTemplate>* no inicio da aplicação, sendo dessa forma possível compará-las com as impressões digitais que se pretendem autenticar. Para comparar duas impressões o SDK do leitor biométrico disponibiliza a classe *DPFPVerification* que fornece o método *match*, método que recebe como parâmetro dois objetos *DPFPTemplate* devolvendo verdadeiro caso as impressões sejam iguais.

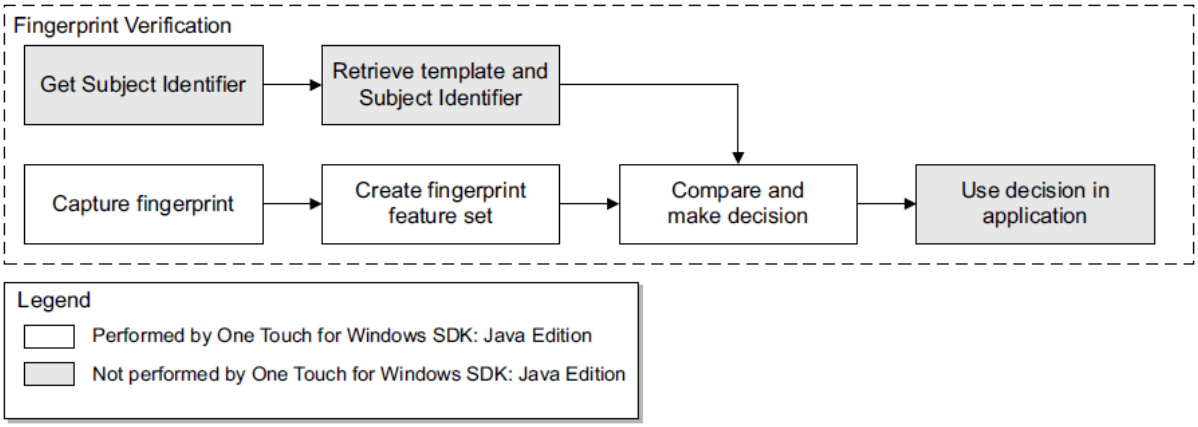

Figura 23 Esquema de registo da impressão digital [46].

Para além do registo biométrico, a aplicação Relógio de Ponto, permite, também, o registo através da inserção do número de identificação e de um pin, configurados anteriormente na aplicação g.Core. Esta opção apenas surge quando não existe nenhum leitor biométrico no sistema, mantendo assim o funcionamento da aplicação.

Figura 24 Registo com as credenciais número e pin.

Para registar o movimento, o utilizador, tem de introduzir o seu identificador do g.Core (id utilizador) e o pin que é configurado na aplicação g.Core. Os pins e os ids de cada utilizador são sincronizados para a base de dados local do RP de forma que seja possível realizar a autenticação *offline*.

Aplicações Móveis

O g.Core é uma aplicação *web* sendo, assim, acessível em todos os dispositivos com um *browser* com suporte HTML5 e ligação à Internet. Porém, não está otimizado para plataformas móveis como *smartphones* e *tablets*. Para resolver esse problema, a Grifin optou por desenvolver aplicações móveis que permitissem utilizar funcionalidades do g.Core num contexto de mobilidade. No âmbito deste projeto definiu-se então o desenvolvimento da aplicação g.Core Mobile, que agrega todas a funcionalidades básicas, pertencentes ao g.Core como, por exemplo, a gestão do calendário pessoal e gestão de clientes.

Para outras funcionalidades, usadas em contextos muito concretos, definiu-se o desenvolvimento de aplicações específicas, como é o exemplo da aplicação g.Produção desenvolvida também no âmbito deste estágio.

Antes de iniciar o desenvolvimento das aplicações móveis, g.Core Mobile e g.Produção, foi necessário decidir qual o tipo desenvolvimento que seria mais adequado a estas novas aplicações. No conceito de mobilidade existem três tipos de desenvolvimento: nativo, *web* e híbrido.

O desenvolvimento nativo é específico de uma plataforma (ex.: iOS [47] ou Android [48]) utilizando as ferramentas de desenvolvimento que a mesma disponibiliza. Com este tipo de desenvolvimento consegue-se o desempenho máximo da aplicação, ficando no entanto confinado a uma plataforma específica.

As aplicações web são desenvolvidas geralmente utilizando HTML5, JavaScript e CSS que apenas necessitam de um *browser* para serem utilizadas, podendo ser usadas em múltiplas plataformas com o desenvolvimento de apenas uma aplicação. No entanto, este método tem limitações como o baixo desempenho, não ter suporte a funcionalidades *offline* e a impossibilidade de utilizar funcionalidades nativas (câmara, calendários, geo-localização, etc.).

O desenvolvimento híbrido permite tirar partido da maioria das funcionalidades nativas, no entanto, com perda de desempenho. Este método geralmente utiliza HTML5 embebido numa aplicação nativa combinando o desenvolvimento web com o nativo. Este tipo de desenvolvimento permite criar uma aplicação multiplataforma podendo esta tirar partido das funcionalidades específicas de cada plataforma.

Após a análise dos diferentes tipos de desenvolvimento, decidimo-nos pelo desenvolvimento de aplicações nativas. Esta escolha deve-se principalmente à necessidade de um bom desempenho e utilização de funcionalidades nativas, nomeadamente, funcionalidades de geo-localização. O desenvolvimento de uma aplicação híbrida chegou a parecer a opção mais

correta, no entanto, devido a inexperiência no desenvolvimento desse tipo de aplicações, optámos pelo desenvolvimento nativo.

Tomada a opção do desenvolvimento nativo, foi necessário escolher qual a plataforma alvo. Após um estudo elaborado pela Grifin aos seus clientes, concluiu-se que a maioria utiliza dispositivos Android. Sendo assim, definiu-se que as aplicações g.Core Mobile e g.Produção fossem desenvolvidas para a plataforma Android. Existem várias versões Android distribuídas no mercado, no entanto, as versões mais comuns são as 4.03 (API 15) e superiores já presentes em mais de 75% dos dispositivos Android tal como mostra a Figura 24. Com base nestes dados, definimos que as aplicações móveis a desenvolver devem suportar versões de Android iguais ou superiores à versão 4.03.

Version	Codename	API	Distribution
2.2	Froyo	8	1.3%
2.3.3 - 2.3.7	Gingerbread	10	21.2%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	16.9%
4.1.x	Jelly Bean	16	35.9%
4.2.x		17	15.4%
4.3		18	7.8%
4.4	KitKat	19	1.4%

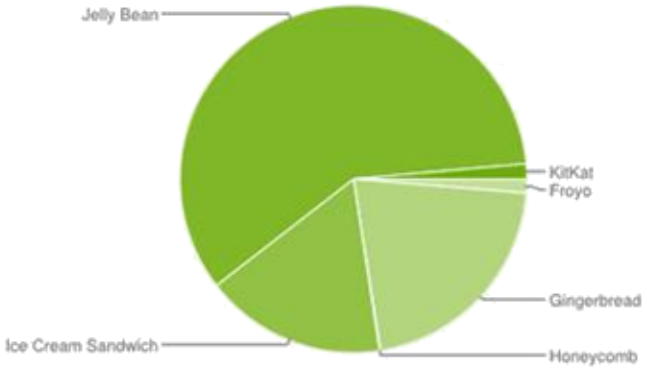

Figura 25 Percentagem de utilização das várias versões de Android. [49] Dados recolhidos a 12 de Janeiro de 2014.

7.1 g.Library

A criação de duas aplicações móveis, g.Core Mobile e g.Produção, e a possível criação de mais aplicações Android com acesso ao g.Webservice justificou a criação de uma biblioteca de nome g.Library que tem todas as funcionalidades de acesso à webAPI do g.Core. Esta biblioteca acumula várias funcionalidades que permitem que aplicações desenvolvidas nativamente para Android consigam facilmente aceder às funcionalidades disponibilizadas pelo g.Webservice. O g.Library disponibiliza funcionalidades de comunicação com a WebAPI (gestão de acessos e obtenção de recursos), persistência de dados locais e um conjunto de componentes visuais que facilitam a autenticação de um utilizador na aplicação.

7.1.1 Autenticação

O acesso aos conteúdos do *g.Webservice* estão protegidos, podendo apenas ser acedidos por utilizadores autorizados e autenticados, utilizando o método *OAuth 2.0*, protocolo de autenticação disponibilizado pelo *g.Webservice*. Todo o processo necessário para a autenticação de clientes perante o g.Webservice é semelhante. Nesse sentido, a biblioteca g.Library disponibiliza um conjunto de funcionalidades que permitem facilitar e uniformizar todo o processo de autenticação por parte dos clientes Android. A g.Library suporta apenas o método de autenticação *Resource Owner Password Credentials* que permite a autenticação utilizando as credenciais do g.Core.

De forma a tornar fácil a implementação do método de autenticação, a g.Library disponibiliza um conjunto de componentes visuais, e não só, que facilitam o processo de autenticação, como a inserção das credenciais e a verificação e validação das credenciais.

Foi criada uma classe *Activity* de nome *SplashScreenBase* e duas classes *Fragment*, *LoginFragment* e *VerifyLoginFragment*. A classe *SplashScreenBase* é responsável por gerir os fragmentos a mostrar, enquanto as classes *LoginFragment* e *VerifyLoginFragment* contêm a lógica de autenticação propriamente dita. O ecrã da *SplashScreenBase* é composto por uma área (*Container*) onde são mostrados os *Fragments.*

Figura 26 Ecrã inicial disponibilizado pela g.Library.

Numa primeira fase, quando a *Activity SplashScreenBase* é iniciada, verifica se a aplicação tem alguma conta de utilizador configurada. Caso ainda não tenha nenhuma conta confirmada, mostra o *LoginFragment* que permite ao utilizador introduzir as suas credenciais do g.Core.

Após o utilizador preencher todos os campos e carregar no botão Entrar, o *LoginFragment* é substituído pelo *VerifyLoginFragment* (Figura 26). A classe *VerifyLoginFragment* é responsável pelo envio das credenciais do utilizador para o *g.Webservice*. Caso as credenciais estejam corretas o *access-token* e respetivo *refresh-token* são guardados na aplicação de forma

persistente utilizando o mecanismo *SharedPreferences* em modo *Private*. O *VerifyLoginFragment* é também responsável pela verificação do *access-token* e utilização do *refresh-token*. Caso já exista uma conta quando a aplicação é iniciada, a *SplashScreenBase* mostra o *VerifyLoginFragment* que verifica se o *access-token* da conta está válido. Caso não esteja, utiliza o *refresh-token* para obter novas credencias de acesso.

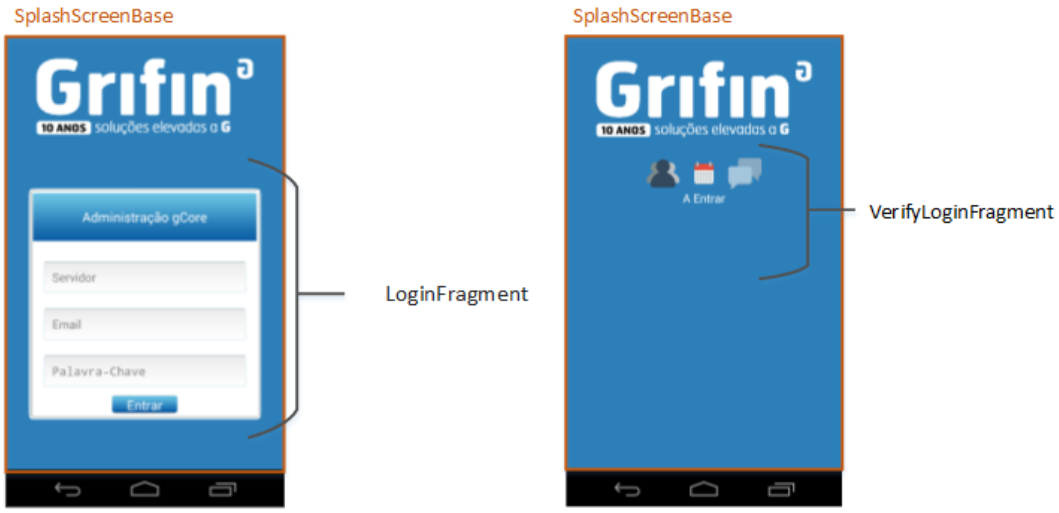

Figura 27 Fragments que permitem autenticação.

Para que uma aplicação que inclua o g.Library possa utilizar a funcionalidade de autenticação, basta criar uma classe que estenda da SplashScreenBase e implementar o método getMainClass que devolve um objeto Class<?>. Esse objeto devolvido deve ser a Activity a que se pretende iniciar após a autenticação.

7.1.2 WebAPI

O g.Library disponibiliza um conjunto de métodos que permite o acesso aos conteúdos do g.Webservice, facilitando a integração de aplicações Android. Este conjunto de métodos no formam uma camada que permite comunicar com o g.Webservice utilizando os métodos HTTP.

A camada de comunicação foi desenvolvida utilizando a biblioteca HTTP já disponibilizada pelo SDK Android e a biblioteca externa de processamento de formato JSON, GSON. Esta camada é constituída por uma função para cada método HTTP: GET, POST, PUT e DELETE. Todos estes métodos tratam automaticamente da autenticação do pedido no webservice, bem como do tratamento de possíveis erros no pedido. Estas funções tratam automaticamente da resposta devolvendo logo o objeto referente ao pedido. Como estes métodos devolvem objetos do tipo T, objeto genérico, podem ser usados para todos os pedidos independentemente do objeto que o pedido devolva.

```
Type listType = new TypeToken<ArrayList<Evento>>() {}.getType();

ArrayList<NameValuePair> nvp = new ArrayList<NameValuePair>();
nvp.add(new BasicNameValuePair("data", sdf.format(date)));
nvp.add(new BasicNameValuePair("terminadas", ""+isTerminado()));

ArrayList<Evento> eventos = Webservice.getRequestType(getActivity()
    , "utilizadores/"+AccountManager.getUserAccount(getActivity()).getUtilizador().getID()+"/eventos/dia"
    , listType, nvp);
```

Figura 28 Exemplo de utilização da webAPI disponibilizada pela g.Library.

A imagem acima é um exemplo de um pedido GET utilizando a camada de acesso webAPI presente na g.Library. Nesta imagem podemos ver que é invocado o método getRequestType. Este método recebe quatro parâmetros: o Context, que permite internamente obter os *tokens* de acesso do utilizador e o endereço do servidor que estão guardados nas *SharedPreferences* [50] da aplicação; o método a executar pelo webservice; a lista de variáveis a enviar e uma variável Type que indica o tipo da mensagem a receber, podendo assim corresponder a resposta a uma entidade na altura em que o *parsing* da resposta JSON é realizado.

A lógica de funcionamento bem como os parâmetros recebidos são semelhantes nos restantes métodos de acesso, diferenciando-se só no que toca à implementação interna, pois cada método corresponde a um método HTTP diferente.

7.1.3 AndroidAnnotations

AndroidAnnotations [51] [52] é uma ferramenta Android que tem como objetivo principal simplificar o desenvolvimento e manutenção de aplicações Android. Esta ferramenta funciona com base em anotações Java que permitem diminuir o número de linhas de código necessárias para realizar algumas ações, gerando o código na altura da compilação da aplicação. Desta forma, não existe perda de desempenho durante a execução da aplicação.

```java
@EActivity(R.layout.translate) // Sets content view to R.layout.translate
public class TranslateActivity extends Activity {

    @ViewById // Injects R.id.textInput
    EditText textInput;

    @ViewById(R.id.myTextView) // Injects R.id.myTextView
    TextView result;

    @AnimationRes // Injects android.R.anim.fade_in
    Animation fadeIn;

    @Click // When R.id.doTranslate button is clicked
    void doTranslate() {
        translateInBackground(textInput.getText().toString());
    }

    @Background // Executed in a background thread
    void translateInBackground(String textToTranslate) {
        String translatedText = callGoogleTranslate(textToTranslate);
        showResult(translatedText);
    }

    @UiThread // Executed in the ui thread
    void showResult(String translatedText) {
        result.setText(translatedText);
        result.startAnimation(fadeIn);
    }

    // [...]
}
```

Figura 29 Exemplo de Activity a utilizar o AndroidAnnotation [51].

Ao incluir esta ferramenta na g.Library, todas as aplicações que usam esta biblioteca ficam aptas a utilizar este sistema de anotações, tendo apenas de configurar o novo projeto para que resolva as anotações no momento da compilação. A utilização de uma ferramenta deste género permite facilitar a escrita e leitura do código, otimizando o desenvolvimento e suporte do mesmo.

7.2 g.Core Mobile

O g.Core, como referido anteriormente, é uma aplicação web que pode ser acedida através de *web browsers*, independentemente do sistema operativo ou dispositivo. No entanto, esta aplicação não está otimizada para ser acedida por dispositivos móveis, como *tablets* e *smartphones*. Devido a essa limitação e a pedido de alguns clientes, a Grifin decidiu criar uma aplicação móvel que permita o acesso a algumas funcionalidades do g.Core a partir destes dispositivos de forma fácil e rápida. Assim, a Grifin propôs, para este estágio, a criação de uma aplicação cliente para Android, o g.Core Mobile, que permita aceder a algumas das

funcionalidades do g.Core. Repare-se que, na prática, o g.Core Mobile deve consumir serviços da aplicação g.Webservice que, por sua vez, replica alguma da lógica de negócio originalmente desenvolvida para o g.Core.

*O g.*Core permite muitas funcionalidades, mas nem todas são aplicadas em situações de mobilidade. Por outras palavras, existem funcionalidades no g.Core que só fazem sentido num contexto *desktop*. Assim, o *g.Core Mobile* foi projetado para suportar, numa fase inicial, as seguintes funcionalidades: gestão de entidades externas, gestão de atividades e gestão de tarefas.

7.2.1 Gestão de Entidades Externas

No g.Core Mobile é possível fazer a gestão de entidades externas, isto é, criar, visualizar, editar e eliminar uma entidade. Além disso, é possível editar as coordenadas GPS de uma entidade, recolhendo a localização momentânea do dispositivo móvel. Para realizar essa tarefa foi utilizado o *LocationManager* [53], uma API que permite obter a localização de dispositivos *Android*, através da *wifi*, redes móveis e satélite GPS. Desta forma, quando um funcionário visita um cliente pela primeira vez, pode definir a coordenada GPS desse cliente de forma fácil.

Na aplicação é possível ver a listagem de todas as entidades, ordenada pelas mais consultadas e pesquisar por nome, contacto ou morada sendo, dessa forma, rápida a procura de um determinado contacto.

Quando o utilizador seleciona uma entidade, a aplicação permite fazer a realização de chamadas, o envio de *emails* e iniciar um navegador GPS que indique o caminho para essa entidade. Estas funcionalidades são disponibilizadas com pedidos específicos ao sistema *Android*, utilizando assim aplicações já presentes nos dispositivos para realizar estas tarefas. Por exemplo, ao tocar no número de contacto de uma entidade, a aplicação g.Core mobile pede ao sistema Android para mostrar todas as aplicações instaladas no sistema que permitam realizar chamadas, podendo depois o utilizador escolher que aplicação pretende utilizar. Na Figura 30 é mostrado um exemplo de código que permite iniciar uma chamada utilizando uma aplicação já existente no sistema.

```
Intent callIntent = new Intent(Intent.ACTION_CALL);
callIntent.setData(Uri.parse("tel:"+getItem(position).getTelefone()));
callIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
getContext().startActivity(callIntent);
```

Figura 30 Código utilizado para iniciar uma chamada utilizando o sistema Android.

Esta funcionalidade permite que a aplicação forneça funcionalidades adicionais sem que seja necessário implementá-las, utilizando assim recursos já existentes presentes em outras aplicações.

7.2.2 Gestão de Tarefas e Atividades

A aplicação móvel g.Core Mobile permite fazer a gestão de tarefas e atividades semelhante a um calendário diário, listando tarefas e atividades numa listagem diária.

Tarefas são notas, ou pequenos "afazeres" que podem ser criados para uma determinada data e atribuídos a um utilizador. Uma tarefa pode ainda estar relacionada com uma entidade terceira. À semelhança do g.Core, o g.Mobile permite também a criação de tarefas bem como a gestão das mesmas.

A gestão de atividades é também suportada pela aplicação móvel g.Mobile. As atividade são eventos que têm definidas datas de início e de fim e estão sempre associadas a uma entidade externa. Uma vez concluída a atividade, é necessário preencher um relatório referente à mesma. O g.Mobile permite a gestão de atividades, criação e edição, mas não permite a sua conclusão. Ao concluir uma atividade, é sempre necessário o preenchimento de um relatório. Como essa tarefa implica a escrita de texto, com toda a descrição da atividade, não faz sentido realizá-la num contexto de mobilidade.

7.2.3 Design e Conceção / Navegação

No desenvolvimento de uma aplicação móvel nativa deve-se tentar seguir ao máximo as *guidelines* recomendadas pela plataforma alvo. A plataforma Android não foge a essa regra, disponibilizando um conjunto de regras de *design* e experiência de utilização para que os utilizadores da aplicação tenham a melhor experiência possível.

Durante o desenvolvimento da aplicação g.Mobile tivemos em conta as *guidelines* [54] da plataforma Android para que a sua utilização fosse de fácil aprendizagem por parte dos clientes.

O g.Mobile é constituído por uma zona principal onde são adicionados Fragments referentes a funcionalidades específicas. É possível alterar de funcionalidade utilizando o menu lateral, sendo este mostrado ao fazer *swipe* ou carregar no ícone da aplicação, como é ilustrado na Figura 31.

Figura 31 *Workflow* do menu lateral.

Esta forma de navegação é recomendada, segundo as *guidelines Android*, para aplicações com mais de três funcionalidades principais quando é necessário navegar entre elas. O SDK *Android* disponibiliza a componente *NavigationDrawer* que permite implementar este sistema de navegação. O *NavigationDrawer* é composto por um *Fragment* que é mostrado sempre que o utilizador faz *swipe* a partir da borda lateral esquerda do ecrã.

Ainda no que toca à navegação da aplicação, só apenas as principais funcionalidades são mostradas na Activity principal sendo as outras funcionalidades abertas em novas atividades. Este tipo de navegação faz com que, enquanto o utilizar tenha uma navegação rápida entre as principais funcionalidades e quando está a utilizar uma funcionalidade mais específica, fique apenas contextualizado nessa funcionalidade.

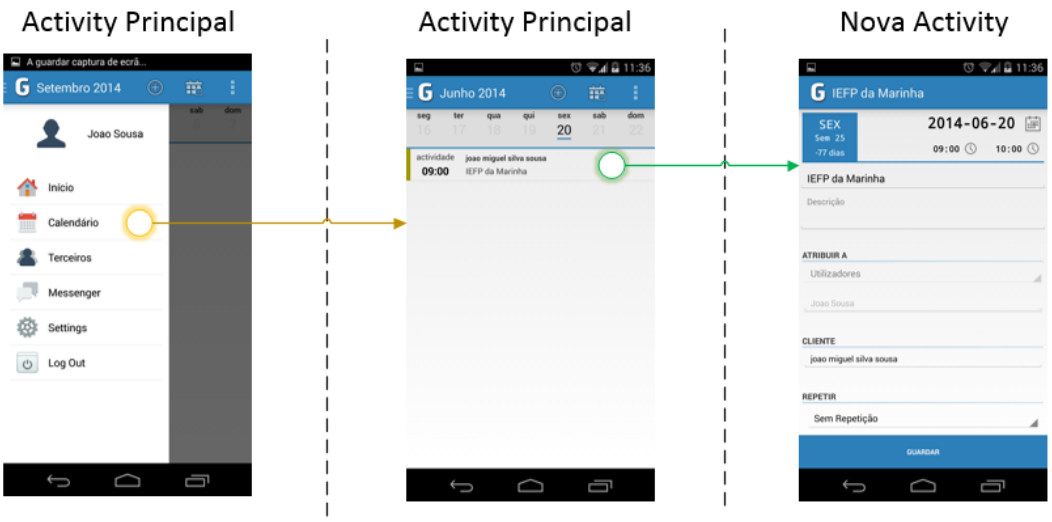

Figura 32 Navegação entre funcionalidades principais e funcionalidades mais específicas.

A imagem anterior simboliza as interações necessárias por parte de um utilizador para editar uma tarefa/atividade do dia atual. No primeiro passo o utilizador seleciona a opção Calendário presente no menu principal. Neste caso a *Activity* principal mantém-se, sendo apenas atualizada a área central com um novo *Fragment* relativo à funcionalidade de calendário. No segundo passo o utilizador carrega na tarefa que pretende editar e dessa forma é aberta uma nova *Activity* referente à edição de uma tarefa/atividade. Esta nova *Activity* não tem menu lateral nem nenhuma outra funcionalidade que não pertence ao contexto de edição de uma tarefa. Neste sentido, funcionalidades como criação e edição são realizadas preferencialmente em novas *Activity*.

7.2.4 Sistema de Notificações

O sistema operativo *Android* permite que as aplicações criem notificações. O g.Core Mobile usa essa possibilidade para notificar o utilizador de que uma atividade ou tarefa foi criada no *g.Core* e também para alertar quando uma atividade ou tarefa está quase a decorrer. Para que estas notificações sejam possíveis, é necessária a comunicação e sincronização entre *o g.Core* e o g.Core Mobile. Existem duas implementações possíveis que permitem que o servidor envie um aviso a um dispositivo *Android*: *POP* e *PUSH*. A implementação *POP* consiste em pedidos constantes ao servidor, por parte do dispositivo móvel, para verificar se uma determinada condição ocorreu. Este método implica maior consumo de energia, algo a evitar quando se trata de dispositivos móveis. O método *PUSH* é muito mais "económico" do que o método *POP*. O *PUSH* permite que o servidor envie informação para os seus clientes. Embora este método tenha uma implementação mais complexa, a Google disponibiliza uma biblioteca de nome *Google Play Services*, que contém a funcionalidade *Google Cloud Message (GCM)* [55] que permite o envio de mensagens *PUSH* para dispositivos *Android* (Figura 33).

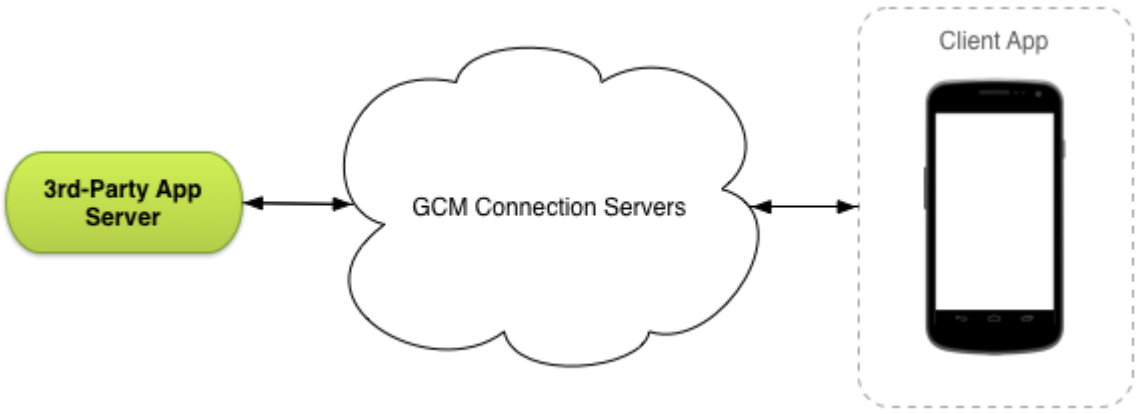

Figura 33 Arquitetura de funcionamento do Google Cloud Message [55].

O Sistema de GCM é constituído por três componentes:

GCM Connection Servers – Serviço disponibilizado pela Google que faz a ligação entre uma aplicação *3rd-Party* e uma aplicação móvel *Android* registada no serviço *GCM*.

3rd-Party Application Server – Aplicação que envia mensagens para o serviço *GCM*. O *g.Core Mobile* e o *g.Webservice* são as aplicações que desempenham este papel neste projeto.

Client App – É uma aplicação presente num dispositivo *Android* e que está registada no serviço GCM, como é o caso, da aplicação *g.Core Mobile*.

Numa primeira fase, os dispositivos da aplicação *Android* registam-se no serviço *GCM* obtendo um ID correspondente a esse registo. Posteriormente, a aplicação envia esse ID para o servidor *3rd-Party*. Com os IDs das aplicações móveis, o servidor 3rd-Party já consegue enviar mensagens para os clientes. Para isso é necessário recorrer ao serviço GCM. Envia-se via HTTP uma mensagem JSON que contém, por exemplo, um vetor como os IDs de todos os dispositivos para os quais se pretende enviar a mensagem e um campo com a mensagem a enviar. No cabeçalho desta mensagem tem de constar ainda a chave de autorização. Esta chave é obtida fazendo o registo do servidor 3rd-Party na API GCM.

Uma vez recebida a mensagem pelo serviço GCM, este envia-a para os dispositivos utilizando os IDs recebidos. Caso algum dipositivo esteja *offline*, só lhe é enviada a mensagem quando este voltar a ficar *online*.

A aplicação *Android*, através de um serviço associado, recebe a mensagem podendo assim despoletar ações como, por exemplo, a criação de notificações.

O g.Core e o g.Webservice utilizam este serviço para enviar uma mensagem aos clientes g.Core Mobile para que estes tenham sempre a informação atualizada e recebam notificações acerca de eventos realizados no g.Core.

Para a aplicação g.Core Mobile receber as notificações do GCM foi necessária a criação de um serviço associado a essa aplicação. Um serviço, no contexto do Android, é um processo que corre em *background* associado a uma determinada aplicação, tendo as mesmas permissões. Um serviço é idealmente usado para realizar operações em *background* que perdurem no tempo ou que estejam sempre a ser repetidas. Dessa forma o serviço fica a ser executado mesmo que a aplicação seja fechada. No contexto da implementação do GCM, foi criado um serviço para ficar à escuta por novas mensagens GCM.

7.2.5 Modelo de Domínio

O g.Core Mobile é uma aplicação cliente que tem como servidor o g.Webservice. Consequentemente, é normal que parte do modelo de domínio do cliente seja semelhante ao do servidor. Como esta aplicação inclui a biblioteca g.Library, tem por omissão todas as classes

referentes ao modelo de domínio implementado no g.Webservice, no entanto, só utiliza uma parte desse modelo de domínio.

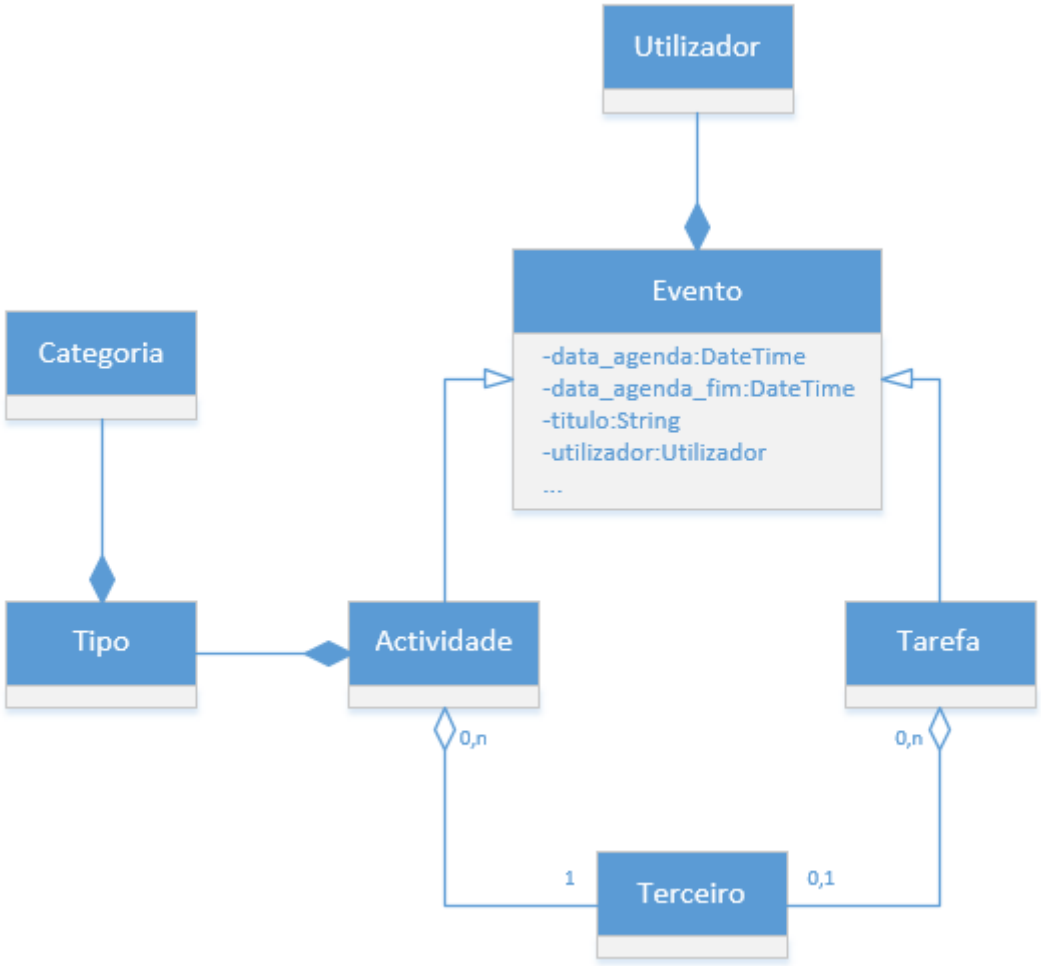

Figura 34 Modelo de domínio utilizado no g.Core Mobile.

A Figura 34 representa o modelo de domínio utilizado na aplicação g.Core Mobile. Este modelo é constituído pelas seguintes entidades:

- **Utilizador:** A entidade Utilizador representa um determinado utilizador do g.Core, tendo campos para as informações gerais como, por exemplo: nome, contacto, departamento, etc.
- **Terceiro:** A entidade Terceiro representa uma entidade externa, ou seja um cliente da empresa. É com base nesta entidade que é realizada a gestão de terceiros da aplicação g.Core Mobile.
- **Evento:** A entidade Evento representa um evento que pode ser calendarizado. Em suma, esta entidade foi criada de forma a agrupar as características comuns das tarefas e dos eventos, como por exemplo: *utilizador*, *data_agenda*, *data_agenda_fim*, etc.

- **Tarefa:** Esta entidade representa uma tarefa no contexto da aplicação g.Core. Uma tarefa tem de estar sempre associada ao Utilizador que a vai realizar. Esta entidade também possibilita a associação a um Terceiro. No entanto, esta associação é facultativa.

- **Atividade:** A entidade Atividade representa uma atividade do contexto da aplicação g.Core. Esta entidade, ao contrário da tarefa, tem obrigatoriamente de estar relacionada a um Terceiro. Uma atividade ainda pertence a um determinado tipo e categoria.

- **Tipo:** A entidade Tipo compõe a entidade Atividade e é composta por uma categoria.

- **Categoria:** Compõe a entidade Tipo, para que um tipo pertença a uma categoria.

7.3 g.Produção

O controlo de custos numa empresa é cada vez mais importante. Num setor tão competitivo como o da indústria dos moldes o controlo de custos é um fator decisivo. O g.Core já disponibilizava ferramentas de gestão de projetos que permitiam calcular custos. No entanto, não tinha nenhuma ferramenta específica para o setor da indústria dos moldes, que permitisse cálculos específicos de gastos de produção. Nesse sentido, surgiu a necessidade desenvolver todo um processo que permitisse aos gestores e responsáveis de projeto saber de forma mais detalhada quais os custos e tempo gasto num determinado molde. A g.Produção é uma aplicação desenvolvida para a plataforma Android que tem o objetivo de permitir aos trabalhadores do setor produtivo dos moldes iniciar e terminar trabalhos sendo possível posteriormente analisar o tempos gasto nos vários trabalhos.

Para esta aplicação foram definindo os seguintes requisitos:

- Iniciar trabalhos atribuídos;

- Pausar e finalizar trabalhos;

- Visualizar trabalhos a decorrer;

- Apenas utilizadores com permissão para aceder ao g.Core podem usar a aplicação.

O utilizador autenticado na aplicação apenas pode realizar operações relativamente aos trabalhos que lhe foram atribuídos.

7.3.1 Funcionamento

Esta aplicação, desenvolvida para tablets, disponibiliza um conjunto de funcionalidades desenvolvidas no *software* g.Core de forma a complementá-lo. Na aplicação g.Core é possível atribuir trabalhos a utilizadores, trabalhos esses que pertencem a uma obra, mais

concretamente, um molde. Um determinado trabalho pode ter associado ou não um equipamento.

O ecrã principal (Figura 35) da aplicação g.Produção é composto por um menu lateral sempre presente com as principais opções, uma área principal onde são mostrados os *Fragments* relativos a cada funcionalidade e uma barra de informações presente no fundo do ecrã.

Figura 35 Ecrã principal da aplicação g.Produção.

Após a atribuição dos trabalhos, os utilizadores ao entrarem na aplicação podem ver uma listagem com todos os moldes a que têm trabalhos atribuídos. Após escolher o molde e a obra em que vai trabalhar é mostrado o ecrã de gestão de trabalhos de uma obra. Esse ecrã lista os trabalhos disponíveis, a decorrer e terminados referentes ao utilizador nessa obra.

O utilizador escolhe o trabalho a iniciar e caso o trabalho tenha um equipamento associado escolhe o equipamento a utilizar. Existem equipamentos que não podem ser utilizados simultaneamente em vários trabalhos. Neste caso o trabalho não pode ser iniciado até que o equipamento fique livre.

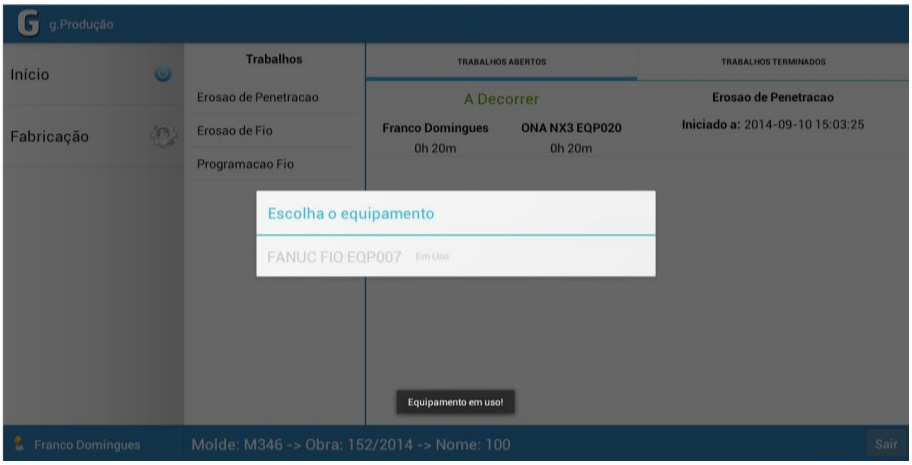

Figura 36 Escolha do equipamento para iniciar trabalho, no entanto não existe nenhum equipamento disponível.

Um trabalho já iniciado pode ser colocado em dois estados, “pausa” e “terminado”. O estado de “pausa” permite que o utilizador possa indicar que fez uma pausa na realização do trabalho e podendo retomá-lo quando assim o entender. O estado “terminado” indica que o trabalho já foi realizado e pode ser então dado por terminado.

7.3.2 Integração com o g.Webservice

A aplicação g.Produção foi desenvolvida como complemento de funcionalidades existentes no módulo g.Moulds da aplicação g.Core. De forma a integrar a aplicação móvel g.Produção foi utilizado o g.Webservice.

Método	**URL**	**Funcionalidade**
GET	utilizadores/{id }/moldes	Devolve uma lista de moldes de um utilizador.
GET	utilizadores/{id }/moldes/{id }/obras	Devolve uma lista de obras pertencentes a um determinado utilizador e molde.
GET	utilizadores/{id}/obras/{id}/trabalhos	Devolve a lista de trabalhos pertencentes a um utilizador e obra.
GET	utilizadores/{id}/obras/{id}/producao	Devolve lista de trabalhos em produção, o seja, que estão a ser realizados ou que já foram concluídos.
POST	utilizadores/{id}/obras/{id}/producao	Cria uma uma produção realtiva a um trabalho. Os dados do trabalho são enviados por POST.
POST	utilizadores/{id}/obras/{id}/producao/{id}/pausa	Coloca a produção em pausa.
POST	utilizadores/{id}/obras/{id}/producao/{id}/terminar	Termina a produção.
POST	utilizadores/{id}/obras/{id}/producao/{id}/iniciar	Inicia a produção

Tabela 6 Métodos do g.Webservice utilizados pela aplicação g.Produção.

Todas as comunicações com o g.Webservice por parte da aplicação g.Producao são realizadas utilizando as funcionalidades disponibilizadas pela biblioteca g.Library. A aplicação g.Produção

funciona em grande parte sem "saber" o modelo de negócio já que essa parte é executada maioritariamente pelo g.Webservice.

7.3.3 Modelo de domínio

À semelhança do g.Core Mobile, a aplicação g.Producao também é uma aplicação cliente do g.Webservice logo, parte do modelo de domínio do cliente é semelhante ao servidor. Esta aplicação inclui a biblioteca g.Library utilizando algumas entidades, como mostra na Figura 37.

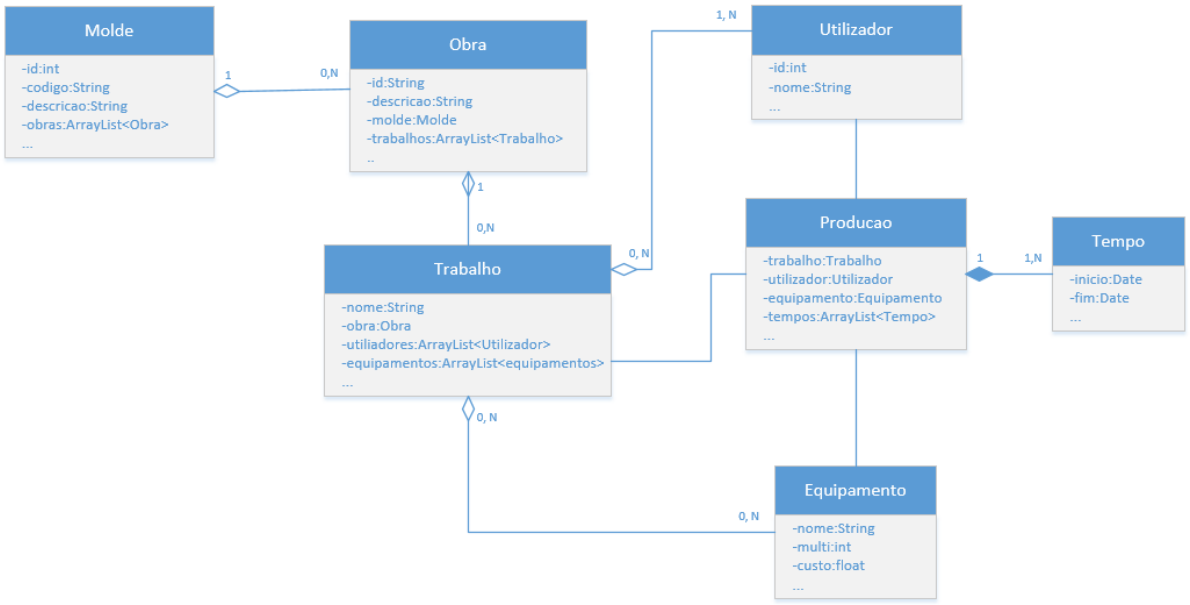

Figura 37 Modelo de domínio da aplicação g.Produção.

O modelo de domínio da aplicação é constituído pelas seguintes entidades:

- **Molde:** Representa um determinado molde. Um molde é composto por várias obras.
- **Obra:** A entidade Obra representa uma determinada obra de um molde. Uma obra tem de estar sempre associada a um molde. Uma obra entende-se por uma parte de um molde que está em produção.
- **Trabalho:** A entidade Trabalho representa um determinado trabalho de uma obra. Uma obra tem vários trabalhos associados, trabalhos esses que podem ser executados por vários utilizadores, daí a relação com a entidade Utilizador. Um trabalho pode ainda ter associados vários equipamentos necessários para a realização do trabalho.
- **Utilizador:** A entidade Utilizador representa um determinado utilizador.
- **Equipamento:** A entidade Equipamento representa um equipamento que pode ser usado para a realização de um trabalho. Um equipamento pode ser ou não multitarefa, ou seja, ser utilizado em vários trabalhos em simultâneo. Para identificar se um

equipamento é multitarefa usamos o campo multi, que indica quantos trabalhos em simultâneo esse equipamento disponibiliza.

- **Produção:** A entidade produção representa trabalhos que foram ou estão a ser realizados. Para iniciar uma nova produção é necessário um Utilizador e um Trabalho. Caso o trabalho necessite de equipamento, então esse equipamento tem de ser associado a essa produção. Uma produção é ainda composta por um ou mais tempos. Dessa forma, é possível fazer o seguimento da produção, conseguindo-se saber se houve pausas e calcular o tempo exato que demorou essa produção.

- **Tempo:** A entidade Tempo compõe a entidade Produção, permitindo marcar o início e o fim de um determinado tempo de trabalho.

7.4 Tecnologias Utilizadas

Ambas as aplicações móveis foram desenvolvidas para dispositivos móveis Android, sendo utilizadas as mesmas tecnologias no desenvolvimento em ambas. As aplicações foram realizadas utilizando a *Android Development Tools plugin*, ferramenta para o IDE eclipse disponibilizada pela Android. A linguagem de programação utilizada no desenvolvimento destas aplicações foi o JAVA.

Conclusão

Para o estágio na empresa Grifin foi proposta a realização de aplicações, *desktop* e móveis, que acrescentassem novas funcionalidades e melhorassem a aplicação g.Core. Numa fase inicial foi proposta a realização de uma nova versão da aplicação Relógio de Ponto e duas aplicações móveis para Android, g.Core Mobile e g.Producao. Todas estas aplicações tinham em comum a necessidade de comunicar com o g.Core. No entanto, este não estava preparado para integrar com aplicações externas. Nesse sentido, para evitar acessos diretos à base de dados via internet, como acontecia com a primeira versão da aplicação Relógio de Ponto, sugerimos a criação de um webservice chamado g.Webservice. Esta sugestão implicou um desafio para o grupo de trabalho, já que a empresa não estava enquadrada neste tipo de conceitos de serviços *web* e integração de sistemas.

Com o g.Webservice foi possibilitada à aplicação g.Core um crescimento e expansibilidade, pois o g.Webservice replicou parte da lógica de negócio do g.Core, implementando-a de forma estruturada, seguindo o padrão de desenho MVC. O g.Webservice desempenha um papel importantíssimo neste projeto, tornando-se o ponto central de todo o trabalho, permitindo às aplicações g.Core Mobile, g.Produção e Relógio de Ponto interagir de forma transparente com os recursos do g.Webservice, recursos esses obtidos da base de dados do g.Core.

A aplicação Relógio de Ponto foi o primeiro cliente do g.Webservice a ser desenvolvido. Com um primeira versão deste *software* já presente em alguns clientes, foi um grande desafio conseguirmos migrar todos os clientes para o novo sistema, desafio esse conseguido com sucesso. Até ao fim deste estágio esta aplicação já se encontrava presente em cerca de dez clientes, mais sete que a primeira versão.

No que toca às aplicações móveis desenvolvidas neste projeto, ambas cumpriram os objetivos e requisitos delineados. A aplicação g.Produção chegou a um produto final, sendo utilizada por vários clientes do g.Core. A aplicação g.Core Mobile, ao contrário das restantes, ainda se encontra em desenvolvimento estando previstas novas funcionalidades.

Durante o desenrolar deste estágio na empresa Grifin muitos trabalhos e sugestões foram realizados e muitas delas executadas com sucesso. Um desses exemplos foi a implementação de uma camada de acesso a dados na aplicação g.Core utilizando um ORM.

8.1 Trabalho Futuro

A aplicação g.Core tem muitas funcionalidades para implementar, mas pensamos que o futuro do desenvolvimento do g.Core deveria passar pela criação de uma nova versão baseada no padrão MVC. Esta nova versão, na nossa opinião, poderia utilizar a *framework* Laravel como base, aproveitando assim algum do trabalho já desenvolvido neste estágio. O g.Core está preparado para cada cliente ter o seu próprio g.Core, com aplicação e dados separados. Este facto faz com que o crescimento do número de clientes torne muito difícil manter as várias aplicações. Pensamos então que no futuro o g.Core pode passar por ser composto apenas por uma aplicação para vários clientes, mas várias bases de dados. Consoante o utilizador, o g.Core iria utilizar a base de dados do cliente correspondente.

Bibliografia

[1] Grifin – Soluções de Informática, “Grifin,” [Online]. Available: http://www.grifin.pt/.

[2] History®, “Industrial Revolution,” [Online]. Available: http://www.history.com/topics/industrial-revolution.

[3] Revolução Industrial, “Só Historia,” [Online]. Available: http://www.sohistoria.com.br/resumos/revolucaoindustrial.php.

[4] J. Kopplin, “An Illustrated History of Computers,” 2002. [Online]. Available: http://www.computersciencelab.com/ComputerHistory/History.htm.

[5] Primavera, [Online]. Available: http://pt.primaverabss.com/pt/.

[6] PHC. [Online]. Available: https://www.phc.pt/portal/programs/cindex.aspx.

[7] G. -. Organimold, “Grandsoft,” [Online]. Available: http://www.grandesoft.pt/organimold.htm.

[8] WinSig, “WinSig,” [Online]. Available: http://www.winsig.pt/.

[9] SIG, “SIG MOLDES,” [Online]. Available: http://www.winsig.pt/pt-mz/sig-moldes.

[10] Idonic, “Idonic:Gest,” [Online]. Available: http://www.relogios-ponto.com.pt/.

[11] Netponto, “Netponto,” [Online]. Available: http://www.netponto.com/.

[12] Safetech, “Safetech,” [Online]. Available: http://www.safetech.pt/Biometria.

[13] PHP, [Online]. Available: http://php.net/.

[14] J. Dotson, “HTTP vs. HTTPS: What's the Difference?,” [Online]. Available: http://www.biztechmagazine.com/article/2007/07/http-vs-https.

[15] MySQL, “The MyISAM Storage Engine,” [Online]. Available: https://dev.mysql.com/doc/refman/5.0/en/myisam-storage-engine.html.

[16] MySQL, “The InnoDB Storage Engine,” [Online]. Available: http://dev.mysql.com/doc/refman/5.0/en/innodb-storage-engine.html.

[17] R. Support, “MySQL Engines - MyISAM vs Innodb,” Rackspace, 30 Maio 2013. [Online]. Available: http://www.rackspace.com/knowledge_center/article/mysql-engines-myisam-vs-innodb.

[18] MySQL, “Storage Engines,” [Online]. Available: https://dev.mysql.com/doc/refman/5.1/en/storage-engines.html.

[19] T. G. Stuff, “MyIsam Vs Innodb Vs Memory – Mysql Storage Engine Comparison,” [Online]. Available: http://www.thegeekstuff.com/2014/02/myisam-innodb-memory/.

[20] PHP, “MySQL Functions (PDO_MYSQL),” [Online]. Available: http://php.net/manual/en/ref.pdo-mysql.php#ref.pdo-mysql.

[21] PHP, “PDO::prepare,” [Online]. Available: http://php.net/manual/en/pdo.prepare.php.

[22] P. Brown, “Prevent PHP SQL Injection with PDO Prepared Statements,” [Online]. Available: http://culttt.com/2012/09/24/prevent-php-sql-injection-with-pdo-prepared-statements/.

[23] Agile Methodology, “Agile Methodology,” [Online]. Available: http://agilemethodology.org/.

[24] Scrum Methodology, “Scrum Methodology,” [Online]. Available: http://scrummethodology.com/.

[25] T. Marston, “The Model-View-Controller (MVC) Design Pattern for PHP,” [Online]. Available: http://www.tonymarston.net/php-mysql/model-view-controller.html.

[26] Tom Dalling, “Model View Controller Explained,” [Online]. Available: http://www.tomdalling.com/blog/software-design/model-view-controller-explained/.

[27] M. Fowler, Patterns of enterprise application architecture, Addison-Wesley, 2003.

[28] Doctrine, “Doctrine,” [Online]. Available: http://www.doctrine-project.org/.

[29] D. Mapper. [Online]. Available: http://martinfowler.com/eaaCatalog/dataMapper.html.

[30] Illuminate Database, “Illuminate Database,” [Online]. Available: https://github.com/illuminate/database.

[31] Propel, “Propel,” [Online]. Available: http://propelorm.org/.

[32] A. Record. [Online]. Available: http://martinfowler.com/eaaCatalog/activeRecord.html.

[33] R. Walker, “Active Record vs Data Mapper for Persistence,” [Online]. Available: http://russellscottwalker.blogspot.pt/2013/10/active-record-vs-data-mapper.html.

[34] M. Fowler, “Data Mapper,” [Online]. Available: http://martinfowler.com/eaaCatalog/dataMapper.html.

[35] M. Fowler, “Active Record,” [Online]. Available: http://www.martinfowler.com/eaaCatalog/activeRecord.html.

[36] M. Rouse, “SOAP (Simple Object Access Protocol),” [Online]. Available: http://searchsoa.techtarget.com/definition/SOAP.

[37] Nordic APIs, “SOAP vs. REST,” [Online]. Available: http://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/.

[38] Laravel, “Laravel,” [Online]. Available: http://laravel.com/.

[39] Laravel, “Controller Filters,” [Online]. Available: http://laravel.com/docs/4.2/controllers#controller-filters.

[40] Laravel, “RESTful Resource Controllers,” [Online]. Available: http://laravel.com/docs/4.2/controllers#restful-resource-controllers.

[41] E. D. Hardt, “The OAuth 2.0 Authorization Framework,” [Online]. Available: https://tools.ietf.org/html/rfc6749.

[42] Netbeans, “Netbeans,” [Online]. Available: https://netbeans.org/.

[43] Oracle, “Oracle,” [Online]. Available: http://docs.oracle.com/javase/tutorial/uiswing/.

[44] google-gson, “google-gson,” [Online]. Available: https://code.google.com/p/google-gson/.

[45] Crossmatch, “U.are.U 4500 Reader,” [Online]. Available: http://www.crossmatch.com/UareU4500Reader/.

[46] Digital Persona, One Touch® for Windows SDK, Developer Guide.

[47] Apple, “Develop Apps for IOS,” [Online]. Available: https://developer.apple.com/technologies/ios/.

[48] Android, “Android Developers,” [Online]. Available: http://developer.android.com/index.html.

[49] Android, “Android Dashboards,” [Online]. Available: https://developer.android.com/about/dashboards/index.html. [Acedido em 12 Janeiro 2014].

[50] Android, “SharedPreferences,” [Online]. Available: http://developer.android.com/reference/android/content/SharedPreferences.html.

[51] Android Annotations, “Android Annotations,” [Online]. Available: https://github.com/excilys/androidannotations/wiki.

[52] J. Orshalick, “Speed up your app development with AndroidAnnotations,” [Online]. Available: http://www.techrepublic.com/blog/software-engineer/speed-up-your-app-development-with-androidannotations/.

[53] Android, “LocationManager,” [Online]. Available: http://developer.android.com/reference/android/location/LocationManager.html.

[54] Android, “Android Design Principles,” [Online]. Available: http://developer.android.com/design/get-started/principles.html.

[55] Android, “Google Cloud Message,” [Online]. Available: https://developer.android.com/google/gcm/gcm.html.

[56] Grandsoft, “O que é o OrganiMold,” [Online]. Available: http://www.grandesoft.pt/organimold/download/What_is_Organimold-2011-Port.pdf.

Anexo I – Funcionalidades g.Core

Este anexo mostra todas as funcionalidades que o g.Core disponibiliza ou pretende disponibilizar em cada módulo.

g.CRM – Gestão de Relacionamento com clientes

- Gestão de Clientes, Fornecedores e Fabricantes;
    - Gestão de Contactos (CMS – Contact Managment System);*
- Gestão de Atividades Técnicas e criação de Relatórios;
    - Sistema de Ticketing Support;*
- Gestão de Atividades Comerciais (SFA – Sales Force Automation);
    - Gestão de SLT – Sales Lead Tracking;*
    - Prospeção e Marketing;*
    - Orçamentação e Oportunidades de Negócios (Leads);
    - Gestão de Objetivos, Planeamento e KPI (Indicadores de Controlo de Progresso);*
- Integração com Google Calendar para sincronização imediata e automática;
- Atribuição de Atividades por Utilizador ou Departamentos;
- Atribuição de Atividades a Projetos – g.Project – para controlo direto de horas e custos;
- Mapa de Planeamento Diário sobre Atividades individual e Geral;
- Calendário Pessoal e Departamental;
- Gestão de Contratos, com envio de Alertas por SMS e PDF por correio eletrónico;
- Listagens personalizáveis e exportáveis;
- Personalização de permissões de Utilizadores e Departamentos;

g.Project – Gestão de Projetos e Custos

- Gestão de Projetos;
    - Controlo de valores de Adjudicação, Despesas e Documentos;
    - Gestão de Horas adjudicadas / consumidas;
- Gestão de Objetivos e Metas;*
    - Criação e Controlo de WBS (Work BreakDown Struture);*
    - Criação automática de Atividades - g.CRM – através das WP (Work Packages);*
    - Calendário e Planeamento GANTT;*
- Atribuição de Equipas, Fases e Equipamentos;
- Personalização de Fases e Movimentos;
- Análise de EVM – Earned Value Managment;*
- Gráficos em tempo real sobre Horas, Utilizadores e Valores;
- IDR Individual - Inserção de Despesas Rápidas para utilizadores;
- Listagem e Análise de Projetos;
- Interligação direta com Gestão de Moldes - g.Moulds;
- Listagens personalizáveis e exportáveis;
- Personalização de permissões de Utilizadores e Departamentos;

g.Moldes – Gestão de Moldes e Obras

- Gestão de Orçamentação
    - Impressão Multilingue;
    - Duplicação e Revisões de Orçamentos;
    - Criação automática de Projeto / Molde;
    - Gestão de Orçamentação a Fornecedores (Revenda);*
    - Interligação com Gestão de Atividades Comerciais SFA – g.CRM;*
- Gestão de Moldes
    - Ficha técnica do Molde, Dados Financeiros, Estados, Prazos, Materiais, etc...
    - Interligação direta com Gestão de Projetos - g.Project – para controlo de custos;
    - Interligação direta com Gestão de Obras - g.Obras;

- Interligação direta com Atividades e Relatórios - g.CRM;*
- Gestão de Alterações ao Molde;
- Gestão de Qualidade e Processos do Molde;
- Gestão de Prazos de Testes e Produções com Cálculo de Custos automáticos;
- Depósito Documental;
- Gestão de Materiais e Stocks por cliente / molde;
- Gestão de Textos Multilingue;
- Gestão de Lista de Materiais, Revisões e Alertas de modificações;*
- Gestão de Departamento de Logística (Transportes);*
- Gestão de Compras - Produtos e Serviços;*
- Gestão de Subcontratações;*
- Gestão de Acessos para Clientes, Fornecedores e Fabricantes;*
- Calendário e Planeamento GANTT com Relatórios;*
- Gestão de ficheiros do molde direto ao servidor local;*

- Gestão de Obras
  - Interligação direta com Gestão de Projetos - g.Project – para controlo de custos;
  - Tipos de Obras: Produção; Controlo Dimensional; Elétrodos, etc…;
- Gestão de Produção
  - Inserção automática através de terminais / posto por IDR;
  - Interligação direta com Gestão de Acessos - g.Point – Recursos Humanos;
  - Informação em tempo real sobre Prazos de Moldes / Obras / Trabalhos;
  - Cálculos de produção por Molde / Obra / Funcionário / Secção / Máquina;*
  - Calendário e Planeamento GANTT por Funcionário / Secção / Máquina;*
  - Priorização automática e manual de Molde / Obra / Trabalhos;*
  - Gestão de Não-Conformidades (problema, responsável, acções a tomar, custos, etc…);*
- Listagens personalizáveis e exportáveis;
- Personalização de permissões de Utilizadores e Departamentos;

g.Point – Gestão de Acessos e Assiduidade

- Gestão de Funcionários e Departamentos;
- Gestão de Horários e Planos de Trabalho, configurável em limites e tolerâncias;
- Gestão de Férias, Feriados, Faltas, configurável em limites legais;*
- Visualização, verificação e alterações de Marcações;
- Acessos Administrativos, de Chefias e Pessoais;*
- Alertas automáticos de faltas;*
- Listagens personalizáveis e exportáveis;
- Personalização de permissões de Utilizadores e Departamentos;

Anexo II – Análise de Requisitos

De seguida é apresentada a análise de requisitos para as várias aplicações desenvolvidas no âmbito deste estágio. Este documento encontra-se dividido em quatro secções, cada uma dedicada a uma das aplicações implementados.

g.Webservice

Requisitos Funcionais	**Prioridade**
O sistema deverá utilizar a os modelos de acessos a dados disponibilizados pelo g.Core.	Media
O sistema deve realizar autenticação com credenciais de utilizadores g.Core	Alta
O sistema deve realizar autenticação de aplicações autorizadas	Alta
O sistema aceita pedidos de modo RestFull (HTTP verbs)	Media
O sistema permite CRUD sobre atividades	Alta
O sistema permite CRUD sobre tarefas	Alta
O sistema deve permitir o envio dos tempos do Relógio de Ponto	Alta
O sistema deve permitir obter a lista de todos os utilizadores	Alta
O sistema deve permitir atribuir registos biométricos a utilizadores	Alta
O sistema permite CRUD sobre trabalhos	Alta
O sistema permite CRUD sobre produções	Alta
O sistema permite obter uma lista de todos os trabalhos de um utilizador	Alta
O sistema permite obter uma lista de todos os moldes de um utilizador	Alta
O sistema permite obter uma lista de todas as obras de um utilizador	Alta

Requisitos Tecnológico	**Prioridade**
O sistema permite operar por HTTP e por HTTPS	Alta
O sistema é desenvolvido em PHP	Alta
O sistema devolve dados no formato JSON	Alta

Relógio de Ponto

Requisitos Funcionais	**Prioridade**
O sistema deverá permitir fazer o registo utilizando a impressão digital do utilizador.	Alta
O sistema deve utilizar o g.Webservice para comunicar com o g.Core.	Alta
O sistema deve listar no ecrã inicial os últimos registos.	Baixa
O sistema deve guardar todos os registos localmente, continuando o seu normal funcionamento mesmo quando não consegue comunicar com o g.Webservice.	Alta
O sistema permite fazer o registo através a inserção do número de utilizador e pin, mas apenas e só quando registo biométrico não está disponível.	Alta
O sistema deve permitir o registo de impressões digitais no sistema.	Alta
O sistema deverá sincronizar todos os dados para o g.Webservice.	Baixa

O sistema deve mostrar a hora atua no ecrã principal.	Baixa
O sistema deve dar feedback de som quando é efetuado um novo registo de movimento.	Media
O sistema deve proteger a área administrativa com credenciais de acesso.	Media

Requisitos Tecnológico	**Prioridade**
O sistema deve utilizar o leitor biométrico Digital Persona	Alta
O sistema é desenvolvido em Java	Baixa
O sistema tem de ser multiplataforma (Windows ou Linux)	Baixa

g.Core Mobile

Requisitos Funcionais	**Prioridade**
O sistema deverá permitir fazer login com as credenciais do g.Core.	Alta
Apenas devem ser permitidos utilizadores do g.Core.	Alta
O sistema deve listar no ecrã inicial as atividades e tarefas para hoje.	Alta
O sistema deve permitir pesquisar tarefas e atividades.	Alta
O sistema deve permitir listar as atividades e tarefas numa determinada data.	Alta
O sistema deve permitir criar, editar e eliminar tarefas e atividades	Alta
O sistema deve permitir fazer a gestão de entidades externas.	Alta
O sistema deve permitir iniciar uma aplicação de GPS com destino á morada da entidade externa	Media
O sistema deve permitir iniciar uma aplicação de *email* de forma a enviar uma email a uma entidade externa.	Media
O sistema deve permitir iniciar uma chamada para uma entidade externa.	Media

Requisitos Tecnológico	**Prioridade**
O sistema é desenvolvido em Java para a plataforma Android	Alta
O sistema deve correr em dispositivos com a versão de Android 4.03 ou superior	Alta
O sistema deve funcionar dependente do g.Webservice	Alta

g.Produção

Requisitos Funcionais	**Prioridade**
O sistema deverá permitir fazer login com as credenciais do g.Core.	Alta
Apenas devem ser permitidos utilizadores do g.Core.	Alta
O sistema deve listar no ecrã inicial todos os trabalhos a decorrer.	Alta
O sistema deve permitir listar todos os moldes do utilizador.	Alta
O sistema deve permitir listar as obras de um molde do utilizador	Alta
O sistema deve permitir visualizar todos os trabalhos atribuídos ao utilizador numa determinada obra	Alta
O sistema deve permitir iniciar novos trabalhos.	Alta

O sistema apenas deve deixar iniciar um trabalho caso este tenha alguma máquina livre para realizar o trabalho.	Alta
O sistema deve permitir pausar um trabalho.	Alta
O sistema deve permitir visualizar todos os trabalhos terminados pertencentes ao utilizador de uma determinada obra.	Baixa
O sistema deve permitir visualizar todos os trabalhos a decorrer pertencentes ao utilizador de uma determinada obra.	Media
O sistema deve permitir terminar um trabalho.	Alta

Requisitos Tecnológico	**Prioridade**
O sistema é desenvolvido em Java para a plataforma Android	Alta
O sistema deve correr em dispositivos Tablet com a versão de Android 4.03 ou superior	Alta
O sistema deve funcionar de forma dependente do g.Webservice	Alta