FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mobile Check-in

João Nuno Santos de Gusmão Guedes

DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Orientador: António Pimenta Monteiro

17 de Fevereiro de 2014

Mobile Check-in

João Nuno Santos de Gusmão Guedes

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Prof. Jorge Alves da Silva

Arguente: Prof. Helena Rodrigues

Vogal: Prof. António Pimenta Monteiro

17 de Fevereiro de 2014

Resumo

Atualmente, a ausência de automação nos processos de validação de utentes em serviços influencia negativamente o setor: os tempos de espera dos utentes pioram a sua satisfação e percepção da qualidade dos serviços; recursos humanos e materiais são gastos anualmente para manter estes processos ativos.

No entanto, cada vez mais utentes transportam consigo dispositivos móveis de alto desempenho, facto que pode ser aproveitado para trazer uma mudança tecnológica a este setor.

Pretende-se desenvolver um sistema nos serviços de saúde que responda a este problema. Este sistema deverá correr nas plataformas mais utilizadas atualmente como o *Android* e *iOS*, e deverá utilizar códigos digitais, a tecnologia NFC e outros meios considerados pertinentes e que tornem expedito o processo de validação numa consulta.

As instituições de saúde que pretendam utilizar este sistema deverão disponibilizar um quiosque nas suas instalações que permita a interação com os dispositivos móveis dos utentes.

Agradecimentos

À minha família, em especial aos meus pais e irmã por toda a paciência, apoio e condições dadas durante o meu percurso académico.

Ao Pedro Rocha, Rui Ferreira, Ricardo Canastro e Nuno Ribeiro pelo auxílio que prestaram na Glintt HS para que este projeto tivesse sucesso.

Ao Diogo Ramalho pelas pequenas (mas úteis) dicas.

Ao Ricardo Teixeira pelos vários anos de companheirismo ao longo do curso.

Aos meus colegas da Glintt HS pelo apoio e companhia durante o tempo em que lá estive.

Ao Professor António Pimenta Monteiro pela orientação durante o processo de desenvolvimento e escrita.

João Guedes

"Never before in history has innovation offered promise of so much to so many in so short a time."

William Henry "Bill" Gates III

Conteúdo

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

Abreviaturas e Símbolos

AJAX	Assynchronous Javascript and XML
APNS	Apple Push Notification Service
CBC	Cipher-block Chaining
CSS	Cascading StyleSheets
EMV	Express Mastercard Visa
GCM	Google Cloud Messaging
HTML	HyperText Markup Language
HTTPS	Secure Hypertext Transfer Protocol
ISO	International Organization for Standardization
JSON	Javascript Object Notation
NFC	Near Field Communication
NFC tag	microchip NFC presente em autocolantes ou posters
NIST	National Institute of Standards and Technology
PKCS	Public-Key Cryptography Standards
QR	Quick Response
REST	Representational State Transfer
SDK	Software Development Kit
SHA-1	US Secure Hash Algorithm 1
Smartcard	cartÃ£o com circuÃto integrado
SOAP	Simple Object Access Protocol
TLS	Transport Layer Security
WCF	Windows Communication Foundation

Capítulo 1

Introdução

1.1 Contexto e motivação

Atualmente, a ausência de automação nos processos de validação de utentes em serviços influencia negativamente o setor: os tempos de espera dos utentes pioram a sua satisfação e percepção da qualidade dos serviços;[Yed12, AA11] recursos humanos e materiais são gastos anualmente para manter estes processos ativos.

Como consequência, os utentes de transportes públicos necessitam de aguardar em filas de espera para poderem carregar os seus títulos de transporte; administrativas de instituições de saúde, para admitirem novos pacientes, interrompem funções da sua competência como recebimentos, atendimentos telefónicos e acompanhamento de pacientes. Os pacientes têm também a desvantagem de ter de aguardar em filas de espera.

Segundo Alrubaiee et al., pesquisas reveleram que um serviço hospitalar de boa qualidade atrai novos clientes e retém os já existentes, reduz custos, melhora a imagem corporativa e melhora a rentabilidade do serviço.[AA11]

Surge então a necessidade de automatizar os processos de validação de utentes nos serviços, com vista a otimizar recursos humanos e materiais e aproveitar benefícios financeiros.

A disseminação das tecnologias móveis é um facto que pode ajudar a resolver este problema; neste contexto, pretende-se desenvolver uma aplicação móvel como prova de conceito no setor da saúde, que permita aos utentes validarem-se numa consulta.

1.2 Projeto

O projeto Mobile Check-In consiste no desenvolvimento de um sistema de validação de utentes na área da saúde. Este projeto integrar-se-á num sistema em desenvolvimento na empresa Glintt HS, que desenvolve soluções informáticas para instituições de saúde tais como clínicas e hospitais privados.

Pretende-se desenvolver uma solução que permita aos utentes usar dispositivos móveis para se validarem numa consulta previamente marcada, contendo uma referência gerada que será validada num terminal a ser implementado nas instituições de saúde.

O sistema será composto pelos seguintes módulos:

- **Aplicação Cliente**: aplicação a ser utilizada num dispositivo móvel *Android* ou *iOS* pelos utentes das instituições de saúde.
- **Servidor da Instituição**: servidor presente numa instituição de saúde, que opera conjuntamente com o **Quiosque** para validar um utente numa consulta.
- **Quiosque**: aplicação a correr num Quiosque presente nas instituições de saúde, munido de sensores NFC e leitor de código de barras. Interage diretamente com o **Servidor da Instituição**.
- **Servidor Central**: trata de toda a comunicação entre os utentes e as instituições de saúde — é um intermediário entre a Aplicação Cliente e o Servidor da Instituição.

O uso de dispositivos móveis na área da saúde limita-se ao uso na prática clínica como a coleta de informações relacionada com pacientes [MPL04], não havendo soluções na área de gestão hospitalar de utentes.

1.3 Objetivos

Nesta dissertação pretende-se cumprir os seguintes objetivos:

- Desenvolver uma aplicação móvel capaz de validar utentes utilizando vários modelos de *smartphones — Android*, *iOS*, *Windows Phone*.
- Desenvolver o *software* que irá correr num Quiosque presente numa instituição de saúde, onde ocorrerá a validação dos utentes.
- Desenvolver os componentes intermédios que garantam um correto funcionamento e comunicação da aplicação móvel com as instituições de saúde.
- Garantir que a comunicação entre os diversos componentes ocorre de forma segura e a confidencialidade da informação é assegurada.
- Estudar a tecnologia *NFC* nos dispositivos que a suportem

1.4 Estrutura da Dissertação

Para além da introdução, este relatório contém mais 6 capítulos.

No capítulo 2, é descrito o estado da arte, a escolha das tecnologias e são presentados trabalhos relacionados.

No capítulo 3 são descritos os requisitos funcionais e não funcionais levantados na fase de desenho do projeto, assim como os casos de utilização do sistema.

No capítulo 4 é descrita a arquitetura do sistema a desenvolver, com uma perspetiva de alto nível.

No capítulo 5 são referidos detalhes de implementação de mais baixo nível, sendo feita a decomposição horizontal de cada um dos módulos descritos anteriormente.

No capítulo 6 são descritos testes efetuados em alguns componentes e as conclusões retiradas.

No capítulo 7 é analisado o estado final do projeto, a satisfação dos objetivos e trabalho futuro.

Capítulo 2

Estado da arte

2.1 Introdução

O estudo do Estado da Arte no âmbito deste projeto compreende as técnicas a usar na autenticação de um utente, a codificação da informação necessária para que seja garantida a confidencialidade da informação do utente e os meios que poderão ser utilizados numa aplicação móvel para autenticar um utente num Quiosque.

Serão descritos alguns projetos semelhantes existentes atualmente no setor dos serviços.

2.2 Autenticação

A autenticação de um utilizador da Aplicação Cliente no Servidor Central pode ser realizada usando algumas das técnicas enunciadas nesta secção.

2.2.1 *Basic Authentication*

Basic Authentication é o tipo de autenticação mais simples e consiste no envio de um *username* e palavra-passe em cada pedido feito pelo cliente. O cliente poderá guardar as credenciais para evitar que o utilizador as tenha que inserir repetidamente.

Segundo Berners-Lee et. al[BLFF96], um cliente que queira autenticar-se num servidor usando o protocolo HTTP — geralmente, mas não necessariamente, após receber uma resposta com o código 401 — poderá fazê-lo incluindo um campo próprio (*request-header*) designado *Authorization* que contêm as credenciais de acesso do utilizador, da seguinte forma:

1. O *username* e palavra-passe são combinados numa única *string* com o formato "*username*:*palavra-passe*".

2. Esta *string* é convertida para o formato *Base64* e é enviada para o servidor com o prefixo *Basic*.

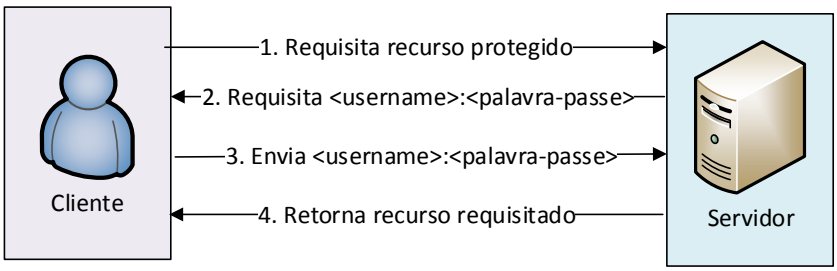


Figura 2.1: Funcionamento do *Basic Authentication*

2.2.2 Baseada em *token*

O uso de palavras-passe e PIN's na autenticação possui várias anomalias:

- A autenticação baseada em conhecimento depende da memória do cliente, i.e se o cliente não se conseguir recordar exatamente da palavra-passe que definiu, a autenticação irá falhar. Não sendo um ponto forte dos humanos a capacidade de memorização de palavras-passe fortes e robustas devido ao seu tamanho e complexidade, esta abordagem aparenta ser vulnerável.[Per]
- As palavras-passe são fáceis de anotar e partilhar. Alguns clientes não vêm qualquer problema em revelar as suas palavras-passe a outros; vêem-no como uma funcionalidade e não um risco.[Per]
- Não é possível revogar acesso a um cliente sem revogar o acesso a todos os outros, sendo necessário mudar a palavra-passe de um cliente para o conseguir.[Har12]
- Os clientes ganham acesso a todos os recursos disponibilizados pelo servidor, sendo impossível restringir a duração e a extensão dos mesmos.[Har12]

Hardt sugere como método alternativo o *OAuth* 2.0[Har12]. Segundo o autor, o *OAuth* resolve os problemas enunciados anteriormente introduzindo uma camada de autorização e separando o papel do cliente do dono dos recursos. No *OAuth*, o cliente requisita acesso aos recursos controlados pelo dono, que se encontram hospedados no servidor, e é emitido um conjunto de credenciais diferente das do dono dos recursos.

A Figura 2.2 ilustra o funcionamento deste método.

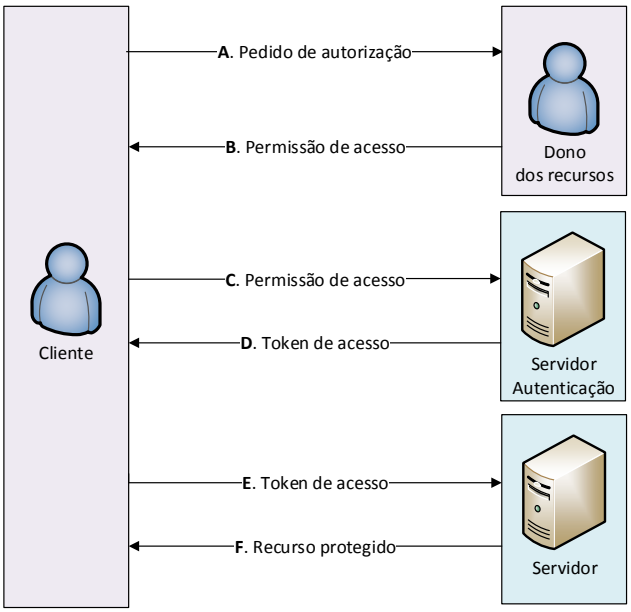


Figura 2.2: Funcionamento do *OAuth*

A O cliente requisita autorização ao dono dono dos recursos. O pedido de autorização pode ser feito diretamente ao dono, ou indiretamente via servidor de autenticação como intermediário.

B O cliente recebe a permissão de acesso — a credencial que representa a autorização do dono.

C O cliente requisita um *token* de acesso autenticando-se no servidor de autenticação e apresentando a sua permissão de acesso.

D O servidor de autenticação autentica o cliente e valida a permissão de acesso, e, se válida, emite um *token* de acesso.

E O cliente requisita o recurso protegido do servidor e autentica-se apresentando o *token* de acesso

F O servidor valida o *token* e, se válido, fornece o recurso.

2.3 Codificação de informação

Toda a informação gerada no decurso do sistema pelos vários Servidores e que seja necessária reter na Aplicação Cliente deverá estar encriptada de forma a que, se terceiros tiverem acesso indevido à Aplicação, os dados não sejam comprometidos. Uma vez que a encriptação/desencriptação de dados depende geralmente de uma palavra-chave, que terá de ser suficientemente segura e robusta, foram estudadas algumas das soluções mais utilizadas atualmente no domínio da geração de palavras-chave, assim como algoritmos de encriptação de dados.

2.3.1 US Secure Hash Algorithm 1

O SHA-1 foi introduzido pelo NIST em 1995 como um standard de processamento de informação federal. Desde a sua publicação que tem sido adotado por praticamente todas as soluções comerciais de segurança [WYY05]

De acordo com o RFC 3174, quando uma mensagem inferior a 2^{64} bits é codificada com o SHA-1, produz um *message digest* de 160 bits. O SHA-1 é seguro uma vez que é computacionalmente inviável encontrar uma mensagem que corresponda ao seu *digest*.[EJ97]

A literatura mais recente sugere que é possível produzir um ataque de colisão que torne este algoritmo inviável em 2^{69} operações.[WYY05]

2.3.2 PBKDF 2

O PBKDF2 (*Password Based Key Derivation Function* 2) é uma função que produz uma chave derivada a partir de uma palavra-chave original. Esta função é atualmente usada pela indústria em protocolos de segurança como o *WPA* e em várias aplicações.

O processo de transformação consiste em aplicar à palavra-chave original uma função criptográfica (*SHA1-HMAC*) e um valor de *salt* — que pode ser visto como um índice para um conjunto de chaves derivadas de uma palavra-chave original. Este processo pode ser repetido várias vezes.[Kal00]

Usando um valor de *salt*, um ataque usando um dicionário de palavras-chave torna-se muito complexo, uma vez que para cada palavra-chave irá existir um número elevado de chaves derivadas.

```
1 DK = PBKDF2 (P, S, c, dkLen)
2
3 Input:      P          palavra-chave, string
4             S          salt, string
5             c          numero de iteracoes, inteiro positivo
6             dkLen      comprimento da chave pretendida, inteiro positivo
7
8
9 Output:     DK         chave derivada, com comprimento dkLen
```

Código-fonte 2.1: Cabeçalho da função *PBKDF2*

2.3.3 AES

Em 2001, após um período de seleção de 4 anos, o NIST elegeu a cifra de *Rijndael* como o *standard* de encriptação de informação governamental Norte-Americano. Esta cifra destaca-se pela sua estrutura simétrica e paralela, que torna a implementação flexível e resistente a ataques, e por se adaptar aos processadores modernos — *Pentium*, *RISC* e processadores paralelos.[JD00]

Este algoritmo usa blocos de cifra e chave de cifra de 128, 192 ou 256 bits. Um bloco de cifra é uma primitiva capaz de converter texto simples em texto cifrado e vice-versa, usando uma chave de cifra.[JD02] O tamanho de uma chave de cifra especifica o número de transformações que um bloco de texto simples está sujeito até se tornar num bloco cifrado:

- 10 ciclos para chaves de 128 bits
- 12 ciclos para chaves de 192 bits
- 14 ciclos para chaves de 256 bits

Alguns ataques foram levados a cabo e documentados. O ataque mais conhecido, descrito pelos próprios autores do algoritmo é o *Square attack*.[Luc00] No entanto, estes ataques não comprometeram o uso do algoritmo dada a sua alta complexidade, mesmo quando a quantidade de informação a codificar é pequena.[Luc00, HG00]

2.4 Meios de transmissão de informação

A validação de um utente numa consulta envolve dois intervenientes — o Quiosque e o utente através da Aplicação Cliente. Foram estudados os meios como esta comunicação poderá ocorrer na prática, de encontro aos objetivos desta dissertação.

2.4.1 *Near Field Communication* (NFC)

Segundo o *NFC Forum* — grupo responsável pelo desenvolvimento da especificação desta tecnologia — o NFC consiste na comunicação bidireccional e sem contacto de dois dispositivos a uma proximidade de no máximo 10 centímetros, seguindo as normas ISO 18092, ISO 14443-2,3,4 e JIS X6319-4. [For]

Os dispositivos NFC podem operar em três modos distintos:

***Peer to peer* (par a par)**
: Comunicações entre dois dispositivos NFC, por exemplo a troca de informações entre dois telemóveis.

Emulação de cartão *smartcard*
: Um dos dispositivos comporta-se como um *smartcard*, e o outro — o leitor — não diferencia este entre um *smartcard* e um dispositivo NFC.

escrita/leitura NFC
: Permite a escrita e leitura de conteúdo em *tags* NFC. Este modo é usado na fidelização de clientes e localização geográfica, colocando *tags* NFC em pontos específicos do percurso dos clientes.

Na literatura encontram-se várias propostas para o uso desta tecnologia no setor dos serviços, nomeadamente dos transportes[OoPT12, FT11, For11], saúde[PHAC12], serviços financeiros[Ltd10] e lazer[JLC12].

Vários projetos piloto têm sido postos em prática usando esta tecnologia em países como Alemanha, Inglaterra, França e Espanha.[OoPT12, For11]

Em 2007 foi feita uma prova no terreno de uma solução de validação de passageiros recorrendo a dispositivos móveis com a tecnologia NFC nos transportes de Londres, como alternativa ao tradicional cartão de transporte *Oyster*, envolvendo 500 clientes de uma operadora de telecomunicações.[OoPT12] Os investigadores concluíram que "resultados chave desta investigação mostram que clientes mantiveram um alto nível de interesse e satisfação durante a prova, e os principais benefícios para o utilizador são a conveniência, facilidade de uso e *status*."[OoPT12]

A empresa alemã de transportes ferroviários Deutsche Bahn oferece, desde 2011, a possibilidade dos seus clientes validarem títulos de transporte recorrendo a dispositivos móveis com a tecnologia NFC, através de *touchpoints* — terminais que permitem a comunicação via NFC, onde os passageiros efetuam o *check-in* na estação de entrada e *check-out* na estação de saída. Após a viagem, o custo é debitado numa conta associada ao utilizador.[Bah, OoPT12]

Puma et al. propõe o uso do NFC para o controlo de acesso a áreas específicas de uma instituição de saúde, por exemplo controlando o acesso de visitantes à sala de um paciente [PHAC12]

As três maiores emissoras de cartões de crédito — Mastercard, Visa e American Express — oferecem, respetivamente, o *PayPass*[Mas], *payWave*[Vis] e *ExpressPay* [Com]. Estes produtos utilizam o NFC para permitir que os seus clientes realizem pagamentos em modo *online*, em que o terminal possui uma ligação a um servidor de pagamentos, ou em modo *offline* diretamente no terminal. O protocolo EMV desenvolvido por estas três entidades reforça a segurança através do uso de algoritmos de encriptação que certificam a legitimidade do cartão.[LRL10]

2.4.2 Códigos digitais

Em alternativa à tecnologia NFC, propõe-se o uso de códigos digitais — códigos de barras tradicionais e/ou bidimensionais, como o *QR* — para transmitir a informação de uma consulta.

Códigos unidimensionais (1D)

Os códigos unidimensionais encontram-se presentes em todo o tipo de artigos comercializados no mundo inteiro, sendo os mais comuns os do tipo *UPC* e *EAN*. A entidade responsável pela definição do *standard* relativo a estas simbologias descreve-as da seguinte forma[US12]:

EAN-8

O EAN-8 é usado em embalagens pequenas onde o EAN-13 seria demasiado grande.

É capaz de codificar o *Global Trade Item Number* 8, usado para identificar produtos comerciais conhecidos, e é também usado para identificação interna por qualquer estabelecimento ou entidade. Codifica no máximo 8 dígitos.

EAN-13

O EAN-13 é uma extensão do EAN-8 capaz de codificar 13 dígitos. É usado globalmente em livros, jornais e revistas.

UPC-A

O UPC-A identifica unicamente artigos de venda em retalho. É capaz de codificar o *Global Trade Item Number* 12 e, à semelhança do EAN, pode ser usado para identificação interna. Codifica no máximo 12 dígitos.

UPC-E

Compressão do formato UPC-A para embalagens pequenas, é capaz de comprimir a mesma informação em apenas 6 dígitos.

Códigos bidimensionais (2D)

QR (*Quick Response*)

O código QR é um símbolo bidimensional criado em 1994 pela Denso, e aprovado como standard internacional ISO em Junho de 2000. Oferece várias vantagens como ocupar pouca área, ter alta densidade e capacidade, e a capacidade de detetar e recuperar de erros. [LL06, Soo08]

Tabela 2.1: Especificação do código QR versão 40 [Soo08]

Tamanho do símbolo	Min. 21x21 células Máx. 177x177 células (4 células de intervalo)
Tipo de informação e volume	Carateres numéricos: máx. 7,089 Alfabeto: máx. 4,296 Binário (8 bit): máx. 2,953 Kanji: máx. 1,817
Eficácia de conversão	Modo numérico: 3.3 células/carater Modo alfanumérico: 5.5 células/carater Modo binário (8 bit): 8 células/carater Modo Kanji (13 bit): 13 células/carater
Correção de erro	Nível L: aprox. 7% de área do símbolo recuperada Nível M: aprox. 15% de área do símbolo recuperada Nível Q: aprox. 25% de área do símbolo recuperada Nível H: aprox. 30% da área do símbolo recuperada

Lagoa et al. propõe um modelo compatível com um sistema de acesso de controlo simples, em que o terminal de validação apenas possui uma câmera.[LMCC10]

1. O utilizador adquire um *token* de validação via Web através de um canal seguro usando TLS

2. O servidor gera o *token* e envia ao utilizador
3. O utilizador escolhe uma *password*, e a aplicação cliente cifra o *token* usando o modo de cifra CBC e o algoritmo de Rijndael; o código QR é então gerado
4. O utilizador apresenta o código QR no terminal de validação e introduz a *password* que escolheu.
5. O terminal de validação contacta um serviço Web para validar a informação contida no código QR.

A leitura do código QR no terminal de validação está dependente de vários fatores como a qualidade da câmera, ângulo de leitura do código, luminosidade do ecrã e área do símbolo. O tamanho do dispositivo móvel irá também condicionar a área máxima possível do símbolo.[LMCC10]

Li et al. propõe um método de binarização — transformação de uma imagem em valores binários de branco e preto — em diversas condições de iluminação e em diferentes dispositivos, com uma taxa de sucesso considerável. A Tabela 2.2 apresenta os resultados de diferentes métodos de binarização.

Tabela 2.2: Métodos de binarização [LL06]

Método de binarização	**Taxa de reconhecimento**	**Tempo de execução (ms)**
Algoritmo de referência do standard QR	80%	36
Método de Otsu	90%	50
Método de Niblack alterado	93%	1080
Método de Ohbuchi	85%	25
Método de Liu e Liu	99.8%	35 (81.4%); 38 (15.7%); 52 (0.028%)

PDF417

O PDF417 foi desenvolvido pela *Symbol Technologies* em 1995, após dois anos de investigação. Segundo Marriott[Mar95], este formato surgiu pela necessidade de transportar os dados das transações no próprio código de barras, sem depender de uma base de dados externa tipicamente usada na altura. O objetivo principal era o de desenvolver uma tecnologia semelhante à dos códigos de barras unidimensionais — de impressão acessível e leitura fácil — mas capaz de transportar uma quantidade de informação centenas de vezes superior.

Um único símbolo PDF417 pode conter mais de um 1KB de dados binário, 2,000 carateres alfanúmericos ou 3,000 dígitos. Adicionalmente, usando a funcionalidade Macro PDF, múltiplos símbolos podem ser ligados de forma a aumentar a capacidade do símbolo original.

Uma vantagem em relação aos códigos *QR* é o deste símbolo ser baseado em códigos de barras, i.e sistemas capaz de ler códigos de barras lineares tradicionais (UPC, EAN, etc.) são inerentemente capazes de ler esta simbologia.

2.4.3 *Web Services* e notificações *Push*

Outra possibilidade para a validação de um utente é o uso de *Web Services* se este possuir ligação de dados no seu dispositivo, ou se a instituição disponibilizar acesso via *Wi-Fi*. O pedido de validação e toda a informação da consulta é enviada pela *Web* e o mesmo meio é usado para enviar a resposta, através de notificações assíncronas — notificações *Push*.

***Representational State Transfer* — REST**

A arquitetura *REST* nasceu da tese de doutoramento de Roy Fielding no ano 2000, e é amplamente utilizada atualmente. Esta arquitetura baseia-se nos métodos (*verbs*) fornecidos pelo protocolo *HTTP* para a criação (*POST*), atualização (*PUT*), leitura (*GET*) e remoção (*DELETE*) de informação. É *stateless* — i.e não é preservada qualquer informação sobre o estado do cliente ou servidor — por natureza, podendo ser feita *cache* no cliente das respostas do servidor, se este o permitir.

O uso desta arquitetura é apropriado para comunicações entre dispositivos móveis e servidores por não necessitar de bibliotecas próprias, necessitando apenas de suportar comunicação por HTTP. O tráfego gerado por este tipo de arquitetura é bastante reduzido — o *payload* dos pedidos e respostas é menor do que na arquitetura *SOAP*. Relativamente à segurança, a arquitetura *REST* assume que o transporte será seguro e não fornece medidas adicionais de segurança.[KW12]

Mulligan et al. conclui por uma série de testes que "a implementação *REST* do modelo de transmissão de dados provou ser mais eficiente em termos de largura de banda utilizada aquando da transmissão de pedidos pela Internet e da latência por eles gerada".[GM09]

***Simple Object Access Protocol* — SOAP**

O W3C [W3C] define o protocolo *SOAP* como um mecanismo leve de troca de informação estruturada entre pontos distribuídos usando *XML*. O *SOAP* não define por si nenhuma semântica, mas oferece um mecanismo simples para que cada aplicação expresse a sua semântica. Este protocolo define as seguintes partes:

- *Envelope*: define a *framework* para expressar o conteúdo de uma mensagem; quem deverá lidar com ela, e se é opcional ou obrigatória.
- *Regras de codificação*: define um mecanismo de serialização que pode ser usado para trocar instâncias de tipos de dados definidos pelas aplicações.
- *SOAP RPC*: define a convenção que pode ser usada para realizar chamadas e receber respostas.

Este protocolo, ao contrário da arquitetura *REST*, permite definir os tipos de dados e métodos usados pelo servidor num ficheiro *WSDL*, acessível através de um *URL*. Isto torna-se vantajoso quando se pretende "espelhar"a estrutura do servidor num cliente que pretenda

aceder ao serviço quando existir uma estrutura complexa de dados. Plataformas como o *Microsoft Visual Studio* e *Netbeans* permitem rapidamente localizar e recriar estas definições no cliente.

Em termos de segurança, o *SOAP* oferece suporte a *SSL* (*HTTPS*) e possui uma extensão designada *WS-Security* — mecanismo para associar *tokens* de segurança a mensagens, garantindo a sua proteção através de integridade, confidencialidade e autenticação.[IBM]

Notificações *Push*

Google Cloud Messaging — GCM

Em 2012 a *Google* anunciou o serviço *GCM* — Google Cloud Messaging. Este serviço é definido por "permitir o envio de dados de qualquer servidor para um dispositivo *Android*. O serviço *GCM* trata de todos os aspetos relativamente ao enfileiramento e entrega de mensagens às aplicações *Android*. É um serviço gratuíto e sem quotas, independentemente do tamanho das mensagens".

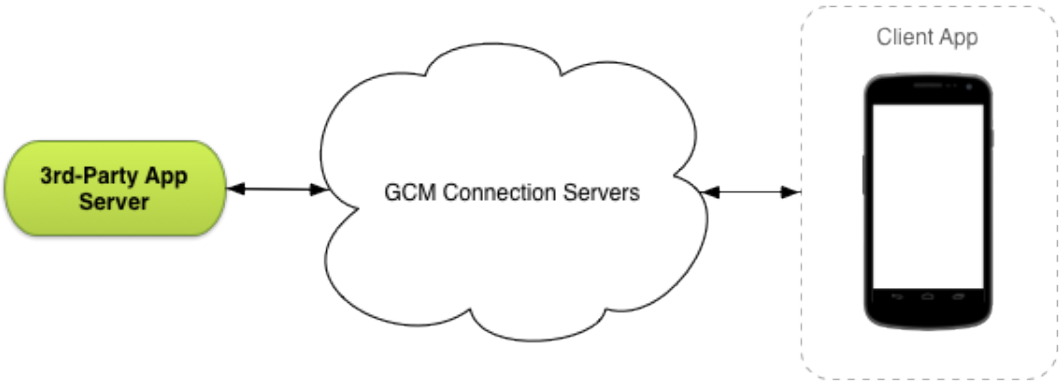


Figura 2.3: Funcionamento do *GCM*

Segundo a documentação oficial, o funcionamento do sistema ocorre da seguinte forma:

- Os servidores de ligação *GCM* fornecidos pela *Google* (*GCM connection servers*) recebem mensagens de servidores pessoais (*3rd-party application servers*) e enviam-nas para as aplicações *Android* que correm nos dispositivos. Atualmente são fornecidos servidores *HTTP* e *XMPP*.
- O servidor pessoal é implementado por quem desejar comunicar com os servidores da *Google*. Os servidores pessoais enviam as mensagens aos servidores *GCM* da *Google*; estes servidores enfileiram e guardam a mensagem, e enviam-na para o dispositivo quando este tiver ligação de dados.
- A aplicação a correr no dispositivo regista-se previamente no servidor *GCM* e recebe um *registration ID*.

***Apple Push Notification Service* - APNS**

A *Apple* fornece um sistema semelhante ao da *Google* para dispositivos *iOS*.

Segundo a documentação oficial, "cada dispositivo estabelece uma ligação *IP* creditada e encriptada com o serviço e recebe notificações através desta ligação persistente. Se uma notificação para uma aplicação chegar ao serviço enquanto esta aplicação não está a correr, o dispositivo alerta o utilizador que a aplicação tem dados à espera de serem consumidos. Os *developers* criam as notificações nos seus servidores. Ligam-se posteriormente aos *APNs* (servidores de notificações da *Apple*) através de um canal seguro. Quando é necessário enviar novos dados aos utilizadores, estem preparam a notificação e passam-na aos *APNs*, que enviam para os dispositivos dos utilizadores."

2.5 Tecnologias

2.5.1 Codificação e descodificação de códigos digitais

Listam-se de seguida algumas bibliotecas disponíveis para a codificação de códigos digitais:

- **Libqrencode**: biblioteca da linguagem C que permite a codificação de códigos QR modelo 2, que não requer ficheiros adicionais em *run-time*, codifica os símbolos de forma rápida e otimiza automaticamente os dados de entrada. Esta biblioteca é disponibilizada segundo a licença GNU Lesser General Public License[Ken]

- **ZXing**: biblioteca *open-source* multi-formato implementada em *Java*, que suporta a codificação e descodificação de vários formatos de códigos bidimensionais, incluindo códigos QR. Oferece integração nativa para *Android*, mas foram desenvolvidos *ports* para *iOS*, *.NET*, *C++* e *Ruby*[ZXi]

- **ZBar bar code reader**: *software suite*, *open-source* e multiplataforma, que permite a descodificação de vários tipos de símbolos 2D. Suporta a licença GNU LGPL 2.1. [Bro]

- **BWIP-JS**: biblioteca Javascript capaz de gerar códigos digitais 1D e 2D (UPC, EAN, PDF417, QR, etc.) em qualquer *browser* ou plataforma que suporte o elemento *Canvas* de *HTML*.

2.5.2 Ferramentas de desenvolvimento multiplataforma de aplicações móveis

Foram realizados vários testes a plataformas de desenvolvimento de aplicações móveis antes de se iniciar a implementação. Durante este estudo foram executadas provas de conceito em algumas das plataformas de forma a avaliar se cumpririam os requisitos necessários.

Appcelerator Titanium

Análise geral

- **Interface**: Usa controlos nativos
- **Plataformas suportadas**: Android, iOS, Blackberry, Tizen, Web e Windows Phone (final 2013)
- **Extensibilidade**: Permite adicionar novas funcionalidades por módulos, disponibilizados num Marketplace.
- **Preço**: Gratuíto
- **Linguagem**: Javascript, XML
- **Reutilização de código**: As interfaces das diferentes plataformas são geradas a partir do mesmo código, assim como a lógica de negócio; grande reutilização do código.
- **IDE**: Titanium Studio (baseado em Eclipse). Não possui code completion nos ficheiros ".tss". (Alloy) e não possui designer de interface.
- **Documentação**: A referência da plataforma é algo confusa inicialmente.
  Os developers mantêm uma relação próxima com a comunidade. Para além da documentação oficial existe bastante informação disponível na internet, como resposta a dúvidas.
- **Outras caraterísticas**: Possui suporte MVC.

Funcionalidades críticas para o projeto

- **Chamadas a Webservices**: Suporta REST e SOAP. O uso do SOAP é no entanto desaconselhado.
- **Suporte NFC em Android**: Escrita e leitura.
- **Calendário**: Não existem um módulo de calendário nativo. Pode ser desenvolvido de raíz usando controlos fornecidos pelo SDK. É possível aceder à API para a criação e leitura de eventos em Android e iOS.
- **API de mapas**: É possível usar API em Android, iOS e Web.
- **Segurança**: É possível usar bibliotecas de JS externas que encriptem os dados (RSA, AES, ...) tendo que ser compatíveis com o standard CommonJS. AES 256-bit disponível no Marketplace (pago).
- **Armazenamento**: SQLite e localStorage.

Observações

- O desenvolvimento usando esta plataforma aparenta ser acessível.
- Permite criar controlos novos baseados em grelhas e labels.
- Bastante lento a fazer deploy no emulador, torna-se impraticável usá-lo.
- O SDK do Mobile Web tem algumas limitações. Não funciona corretamente em todos os browsers.
- Tamanho do binário gerado é considerável (cerca de 15 MB)

Phonegap

Análise geral

- **Interface**: Não usa controlos nativos. Pode recorrer a bibliotecas de terceiros (ex: jQuery mobile)
- **Plataformas suportadas**: Android, iOS, Windows Phone, Symbian, Bada, Tizen.
- **Extensibilidade**: Possui bindings às API nativas, é possível extender as funcionalidades através de plugins.
- **Preço**: Gratuíto
- **Linguagem**: Javascript, HTML, CSS
- **Reutilização de código**: A code-base é a mesma para as várias plataformas; as interfaces são iguais e a reutilização pode chegar aos 100%.
- **Outras caraterísticas**: É possível fazer deploy de uma aplicação na cloud para as várias plataformas.

Observações

- Não é possível desenvolver para Web.
- Não é possível desenvolver interfaces nativas.

Xamarin

Análise geral

- **Interface**: Usa controlos nativos
- **Plataformas suportadas**: Android, iOS, Windows Phone.

- **Extensibilidade**: Integrado com a plataforma; suporte para praticamente toda a API de Android e iOS.
- **Preço**: ($999/ano)(número de developeres)(número de plataformas)
- **Linguagem**: C#
- **Reutilização de código**: As interfaces das diferentes plataformas são geradas independentemente umas das outras e apenas a lógica de negócio é partilhada; menor reutilização do código.
- **IDE**: Xamarin Studio. Possui integração com Visual Studio e designer de interface para iOS, Android e Windows Phone.
- **Documentação**: Bem organizada e de fácil consulta. Muitos snippets de código disponíveis que facilitam a entrada na plataforma.

Funcionalidades críticas para o projeto

- **Chamadas a Webservices**: Suporta REST, SOAP e WCF.
- **Suporte NFC em Android**: Escrita e leitura.
- **Criação de calendários**: Acesso às APIs de iOS e Android para criar e ler eventos.
- **API de mapas**: É possível usar a API em Android e iOS.
- **Segurança**: SQLCipher para encriptar conteúdos da base de dados (pago).
- **Armazenamento**: SQLite.

Observações

- Mais complexo de programar, especialmente a parte das interfaces; o C# faz uma aproximação das linguagens nativas de cada plataforma, sendo necessário conhecer algumas especificidades.
- Não é possível desenvolver para Web.
- Overhead grande para desenvolver as várias interfaces individualmente.

DevExtreme (PhoneJS)

Análise geral

- **Interface**: Usa controlos nativos
- **Plataformas suportadas**: Android, iOS, Windows Phone, Web.
- **Extensibilidade**: Permite incluir plugins do Phonegap

- **Preço**: $199
- **Linguagem**: HTML, jQuery, CSS
- **Reutilização de código**: Grande reutilização do código, as interfaces podem ser desenvolvidas para as várias plataformas.
- **IDE**: Possui integração com Visual Studio e designer de interface.
- **Documentação**: Bastante completa, com manual de referência, how-to's, exemplos...
- **Outras caraterísticas**: Possui integração com o ChartJS, que permite incluir vários tipos de gráficos

Funcionalidades críticas para o projeto

- **Chamadas a Webservices**: Suporta REST.
- **Suporte NFC em Android**: Escrita e leitura usando um plugin.
- **Armazenamento**: SQLite (plugin Cordova incluído).

Observações

- Deploy bastante rápido nos dispositivos móveis.
- Desempenho muito fraco. Só se obtiveram bons resultados no Nexus 4.
- Não é possível desenvolver para Web.
- Não é possível desenvolver interfaces nativas.

Icenium

Análise geral

- **Interface**: Não usa controlos nativos. Recorre às bibliotecas Kendo UI e jQuery Mobile.
- **Plataformas suportadas**: Android, iOS.
- **Extensibilidade**: Permite o uso de plugins para Apache Cordova (Plugman compatible). Não existe um Marketplace da plataforma.
- **Preço**: $16-$79
- **Linguagem**: Javascript, HTML5, CSS
- **Reutilização de código**: A code-base é a mesma para as várias plataformas; as interfaces são iguais e a reutilização pode chegar aos 100%.

- **IDE**: Possui integração com Visual Studio e designer de interface. Existe tambem uma versao Web e um IDE para Windows.
- **Documentação**: Bastantes exemplos disponíveis. Não existe manual de referência.

Funcionalidades críticas para o projeto

- **Chamadas a Webservices**: Suporta REST e SOAP.
- **Suporte NFC em Android**: Escrita e leitura usando um plugin.
- **Criação de calendários**: É possível criar calendários usando plugins de jQuery.
- **API de mapas**: Suporte para API da Microsoft (Bing) e Google.

Observações

- Não permite fazer deploy diretamente para o dispositivo, apenas para o emulador.
- O emulador é bastante rápido, as WebViews funcionam bem mas são lentas nos smartphones. Só teve boa prestação no Nexus 4. A verossimilhança do emulador deixa algumas dúvidas.
- Plataforma ideal para usar se não forem necessários muitos controlos personalizados.
- Não existe IDE para Mac.

Conclusões

Não foi possível desenvolver interfaces rápidos para vários dispositivos diferentes que incluam exclusivamente WebViews, em qualquer das plataformas estudadas.

A plataforma Appcelerator Titanium é a única que oferece *bindings* nativos suficientes para criar calendários, grelhas e outros tipos de controlos sem recorrer a WebViews. Também é possível incluir código HTML que interaja com os restantes módulos do sistema, através de eventos. Esta deverá ser a plataforma de eleição para a realização do projeto, uma vez que cumpre todos os requisitos.

A plataforma DevExpress deve ser considerada se houver grande necessidade de adaptar código HTML já existente para criar aplicações nativas com controlos standard.

2.5.3 Justificação de escolha

A escolha para a biblioteca de codificação e descodificação de códigos digitais deverá pender para a *ZXing* se o código for gerado no servidor, pelo facto de existirem implementações para várias plataformas e por suportar vários tipos de simbologias. Se for necessário gerar o código no cliente, a opção deverá recair na biblioteca *BWIP-JS*, por suportar vários tipos de simbologias.

A escolha da ferramenta de desenvolvimento de aplicações móveis multiplataforma deverá recair sobre a *Appcelerator Titanium*, pois cumpre os requisitos das plataformas móveis pretendidas para a Aplicação Cliente e está a ser utilizada no sistema em desenvolvimento na Glintt, na qual este projeto se integrará.

2.6 Outros projetos

FlyTAP Mobile

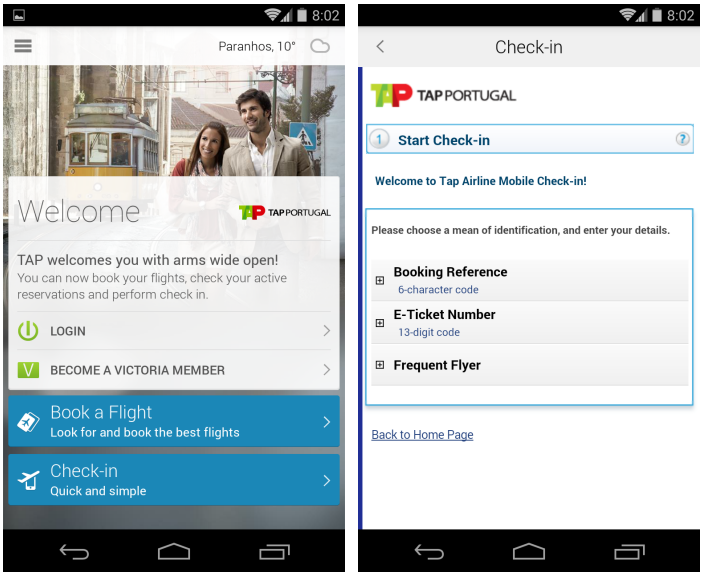


Figura 2.4: Aplicação *FlyTAP*

A companhia aérea portuguesa *TAP Portugal* disponibiliza para os seus clientes a aplicação *FlyTAP Mobile*, que permite aos seus clientes consultar informações sobre reservas, inscrição no programa Victoria e respectiva consulta do extracto de milhas, reclamação de milhas em falta e consulta de horários de voos, partidas e chegadas.

Esta aplicação inclui um módulo de Check-in de passageiros, que permite fazer o Check-in para voos através do telemovel, não sendo necessário passar pelo balcão de check-in no dia da viagem. Após finalizar o processo de Mobile Check-in, os clientes da TAP recebem no seu telemóvel o cartão de embarque. Este serviço encontra-se disponível em 30 escalas e em 32 aeroportos, nas plataformas Android e iOS.

Marriott International

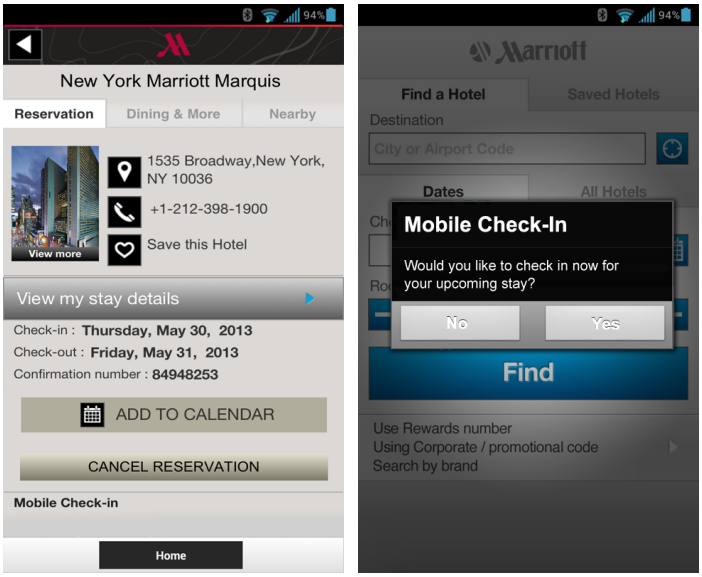


Figura 2.5: Aplicação *Marriott International*

A cadeia de hóteis *Marriott International* disponibiliza aos seus clientes uma aplicação que permite gerir reservas de quartos e realizar o check-in até duas horas antes da chegada, usando um sistema de notificações *Push* para avisar os clientes quando o quarto estiver pronto. À chegada ao hotel, o cliente dirige-se à receção e faz o levantamento da chave do quarto.

A aplicação permite também fazer a requisição de serviços como entrega de toalhas e serviço de despertar. Encontra-se disponível nas plataformas Android e iOS.

2.7 Conclusões

Foram analisadas soluções para a validação de utentes em serviços. Algumas das soluções encontradas exigem que o utente possua um *smartphone* com a tecnologia NFC; outras apenas permitem que seja realizada a validação online e fora do local onde o serviço será prestado. Não foi encontrada uma solução unificada que permita que várias técnicas possam ser utilizadas.

Desenvolver este sistema poderá ser uma oportunidade de forçar um avanço no setor dos serviços recorrendo às novas tecnologias emergentes.

Capítulo 3

Requisitos

3.1 Introdução

Concetualmente, o sistema foi idealizado para ser decomponível em três módulos principais:

- **Módulo Cliente**: corresponde à aplicação móvel a ser desenvolvida. Esta aplicação deverá permitir ao utilizador autenticar-se no serviço; listar, ver detalhes e obter direções de todas as suas consultas; validar-se numa consulta através de NFC, códigos digitais e *Web Services*.
- **Módulo Servidor Central**: um servidor intermédio que trata das comunicações entre a aplicação cliente e a instituição. Deverá funcionar bidireccionalmente, permitindo que a aplicação cliente contacte uma instituição de saúde e vice-versa.
- **Módulo Servidor Instituição**: representa o servidor duma instituição de saúde, a ser operado por um funcionário local. Este servidor deverá estabelecer contacto com um utente através do Servidor Central, sendo-lhe disponibilizada uma interface para notificar o utente quando este tem novas senhas de consulta atribuídas, enviar-lhe mensagens de Marketing ou outras que entenda serem pertinentes. É também responsável por gerar as senhas de atendimento de uma consulta e enviá-las ao utente.

No início do desenvolvimento do projeto foi elaborado um relatório de especificação de Requisitos, que foi sendo atualizado de forma a incluir todas as funcionalidades pretendidas.

São apresentadas de seguida as transcrições dos requisitos levantados, bem como os casos de utilização em notação *UML*.

3.2 Módulo Cliente

Tabela 3.1: Requisitos funcionais (módulo cliente)

Identificador	Descrição
FC1	O utente deverá poder autenticar-se no sistema através de um nome de utilizador e uma palavra-chave.
FC2	O utente deverá poder listar todas as suas consultas por data e hora.
FC3	Ao selecionar uma consulta, o utente deverá poder ver os seguintes detalhes: Data, especialidade, local (visto num mapa com o ponto assinalado) e médico.
FC4	O utente deverá poder obter direções num mapa para uma instituição de saúde onde se irá realizar uma determinada consulta.
FC5	O utente deverá poder validar-se numa consulta manualmente através de um código 1D e/ou 2D.
FC6	O utente deverá poder validar-se numa consulta através de NFC.
FC7	O utente deverá poder validar-se numa consulta usando dispositivo de forma automática através de um *Web Service*, quando este se encontrar dentro do instituição.
FC8	Após validar-se numa consulta, o utente deverá poder consultar a sua senha de atendimento e atualizá-la.

Tabela 3.2: Requisitos não-funcionais (módulo cliente)

Identificador	Descrição
NC1	Deverá existir uma aplicação cliente para a plataforma Android.
NC2	Deverá existir uma aplicação cliente para a plataforma iOS.
NC3	Deverá existir uma aplicação cliente para a plataforma Windows Phone.
NC4	Deverá existir uma aplicação cliente para a Web que se possa utilizar através de um *Browser*.
NC5	As interfaces das aplicações clientes deverão ser desenvolvidas com elementos gráficos nativos da plataforma em questão
NC6	Toda a informação deverá ser guardada de forma encriptada.

3.3 Módulo Servidor Central

Tabela 3.3: Requisitos funcionais (módulo servidor central)

Identificador	Descrição
FS1	O servidor deverá receber as credenciais de um utente e validar junto da instituição.
FS2	O servidor deverá poder consultar as instituições de saúde e, fornecendo as credenciais de um utente, deverá poder listar todas as suas consultas. Os dados de uma consulta deverão conter: Data, especialidade, local (coordenadas GPS), médico, código check-in (token).
FS3	O servidor deverá poder enviar notificações para todos os dispositivos de um utente.
FS4	O servidor deverá poder fazer pedidos de check-in automáticos a uma instituição de saúde
FS5	O servidor deverá expôr métodos que permitam a uma Instituição de Saúde enviar mensagens — senhas de atendimento, *tokens* de consulta e mensagens genéricas.

Tabela 3.4: Requisitos não funcionais (módulo servidor central)

Identificador	Descrição
NS1	O servidor deverá expor uma interface de acesso REST para a aplicação cliente
NS2	O servidor deverá expor uma interface de acesso às instituições de saúde.
NS3	O servidor deverá poder enviar notificações para todos os dispositivos de um utente.
NS4	Toda a comunicação entre o Servidor Central e os restantes componentes deverá ocorrer de forma segura, através de um canal *HTTPS*.

3.4 Módulo Servidor Instituição

Tabela 3.5: Requisitos funcionais (módulo servidor instituição)

Identificador	Descrição
FI1	Deverá ser desenvolvida uma interface que permita a uma instituição de saúde enviar mensagens de notificação e de marketing aos seus clientes.
FI2	Deverá existir um quiosque associado a uma instituição que permita a um utilizador validar-se numa consulta.
FI3	O quiosque associado deverá permitir a um utente realizar o check-in com o seu dispositivo móvel através de NFC e códigos digitais 1D/2D.

3.5 Casos de utilização

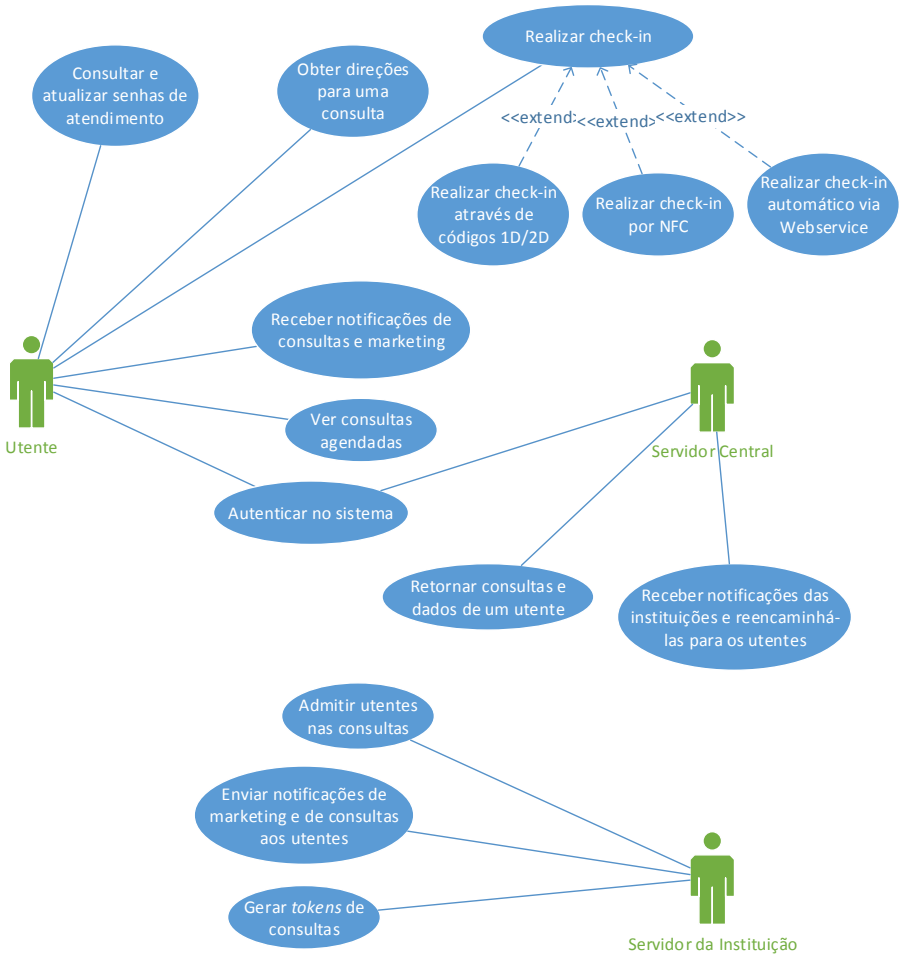


Figura 3.1: Casos de utilização

Capítulo 4

Arquitetura

Neste capítulo será feita uma primeira abordagem à arquitetura escolhida para o projeto vista sobre diferentes perspetivas, mantendo o enfoque na abstração de forma a manter uma visão clara e tentando reduzir ao mínimo os detalhes de implementação.

A decomposição horizontal e uma explicação mais detalhada de todos os módulos será feita posteriormente no Capítulo 5.

4.1 Vista lógica

O projeto é composto por 5 módulos que comunicam entre si assíncronamente através de interfaces REST e SOAP. Os módulos encontram-se organizados por domínios distribuídos fisicamente.

Utente

Representa o utilizador final da aplicação e compreende a aplicação móvel existente para as 3 plataformas.

Intermediário

Representa o serviço central prestado pela Glintt HS que trata de todas as comunicações entre uma instituição de saúde e um utente. Esta camada intermédia é necessária dado que nem todas as instituições de saúde possuem a capacidade de suportar o tráfego gerado pelas comunicações, ou não têm interesse em expor uma interface às aplicações desenvolvidas na Glintt HS por questões de segurança.

Instituição de saúde

Representa o serviço de *check-in* prestado por uma instituição de saúde, incluíndo o quiosque onde um utente se pode validar. Este quiosque comunica diretamente com o servidor da instituição.

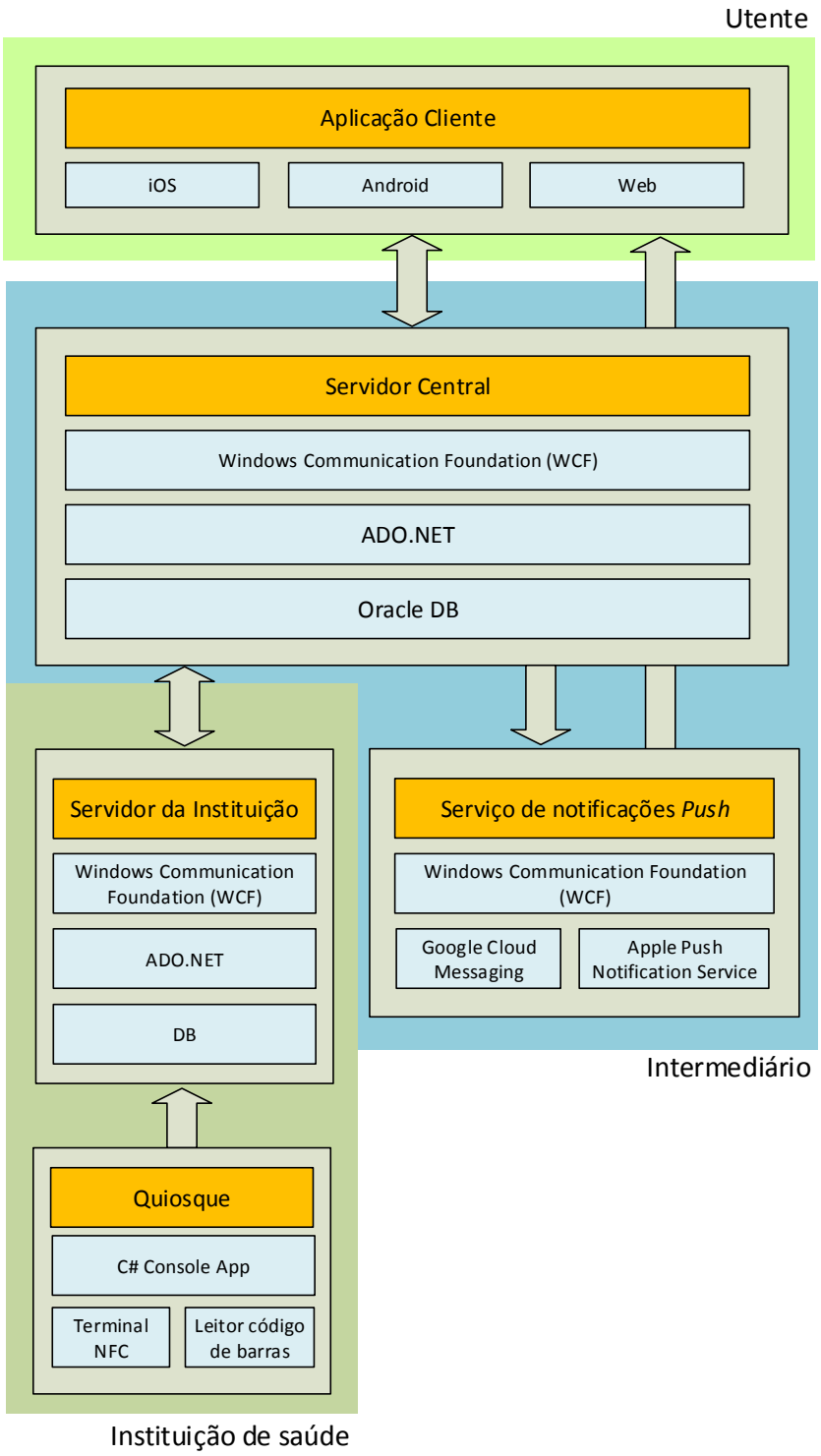


Figura 4.1: Arquitetura de alto nível

4.2 Vista de desenvolvimento

Nesta vista é possível observar a decomposição em componentes e respetiva descrição das interfaces de acesso dos servidores central e da instituição.

São também indicados os protocolos implementados por cada servidor.

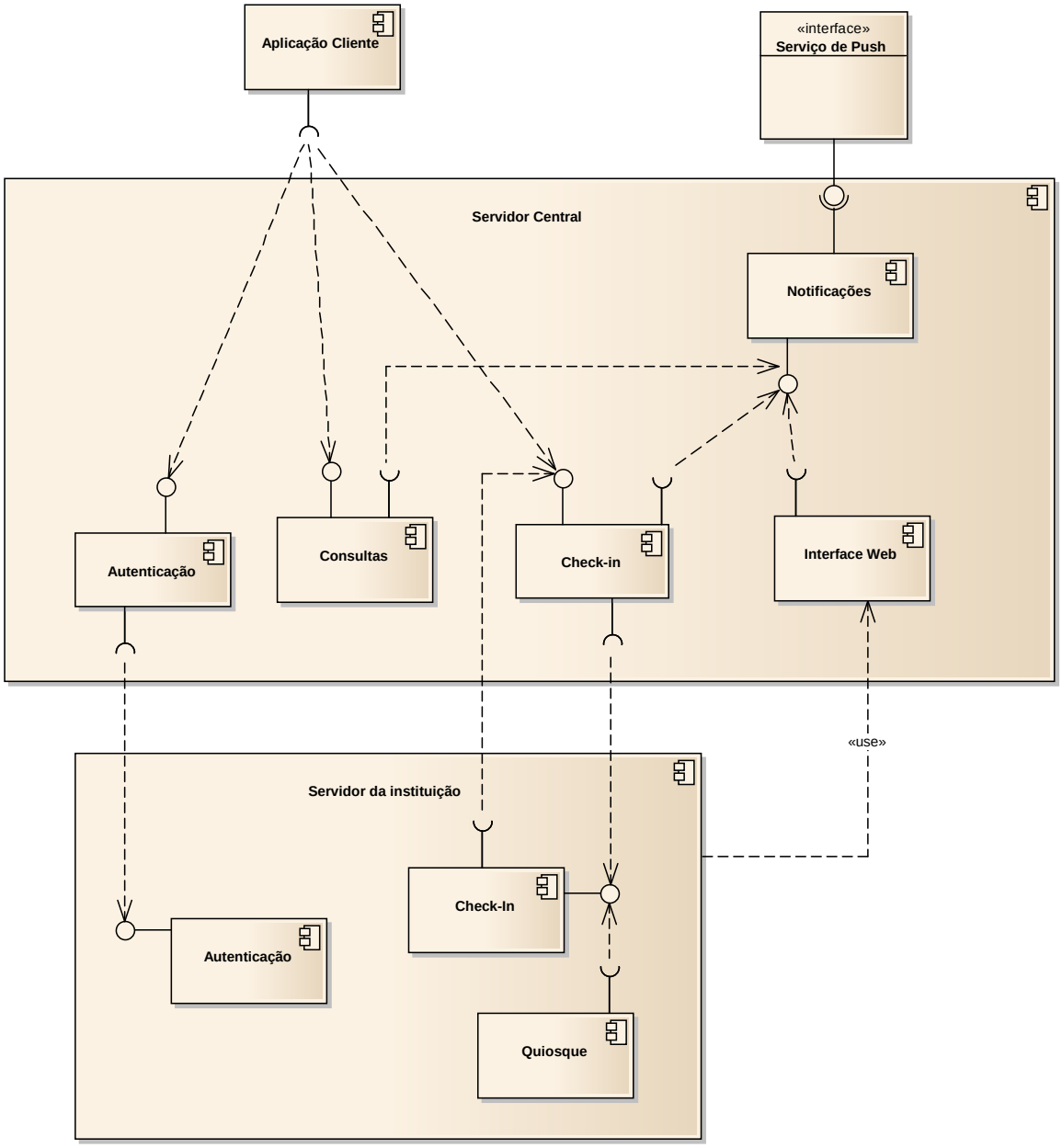


Figura 4.2: Diagrama de Componentes UML

Aplicação Cliente

Consome os métodos expostos pelos componentes de Autenticação, Consultas e Check-in da interface do **Servidor Central**.

Servidor Central

Expõe a interface **IMainService**, que fornece dois terminais de acesso(*endpoints*):

- **RESTful** para que a **Aplicação Cliente** possa realizar pedidos HTTP, i.e receber e enviar dados por *AJAX* recorrendo à notação *JSON*.
- **SOAP** uma vez que os serviços *WCF* suportam nativamente este protocolo e tratam de todo o encapsulamento de informação, tornando-se prático para a comunicação entre este servidor e o **Servidor da Instituição**.

Esta interface decompõe-se logicamente nos seguintes componentes:

Autenticação

Permite receber as credenciais de acesso de um utilizador da **Aplicação Cliente** e passá-las à Instituição de Saúde de forma a autenticar um utente.

Consultas

Agrega todas as consultas de utentes da base de dados e é responsável por criar notificações de novas consultas, recorrendo para isso ao componente de **Notificações**

Check-in

Permite realizar a validação de um utente de forma automática através de um serviço Web.

O **Servidor Central** passa o pedido de validação ao componente **Check-in** do **Servidor da Instituição**, que por sua vez invoca o método de validação do **Servidor Central**. Este método depende do componente de **Notificações** para notificar o utente dos dados da senha da consulta.

Notificações

Componente responsável por direccionar notificações de consultas, senhas de atendimento, mensagens genéricas e de Marketing ao utente, recorrendo para isso ao **Serviço de Push**.

Interface Web

Interface Web disponibilizada ao **Servidor da Instituição** para este emitir novos *tokens* de consulta e criar mensagens genéricas e de Marketing para um determinado utente. Recorre ao componente de **Notificações** para executar a entrega.

Servidor da Instituição

Expõe a interface **IInstitutionServer**, que fornece um terminal de acesso *SOAP* para satisfazer os pedidos e envio de dados do **Servidor Central**.

Esta interface decompõe-se logicamente nos seguintes componentes:

Autenticação

Recebe as credenciais do **Servidor Central**, confirma se estas são válidas e retorna o resultado.

Check-in

Recebe pedidos de validação do **Servidor Central**, tal como indicado anteriormente. Recebe também pedidos de validação do Quiosque, quando o utente se valida de forma manual — usando NFC ou códigos de barras. Se um pedido de validação for aceite é invocado um método do **Servidor Central** que envia ao utente os dados da senha de atendimento da consulta.

Quiosque

Serviço que corre num quiosque instalado na instituição de saúde e aguarda o *input* de códigos de barras ou o contacto por NFC. Após receber o *input*, descodifica os dados recebidos e envia o pedido de validação do utente ao componente *Check-in*.

Serviço de Push

Expõe a interface **IPushNotificationManagement**, que permite o registo de novos dispositivos e o envio de notificações assíncronas aos utentes. Consoante o tipo de dispositivo do utente —*Android* e *iOS* — o serviço delega a entrega das notificações para o *GCM* (*Google Cloud Messaging*) ou para o *APNS* (*Apple Push Notification Service*).

4.3 Vista de processo

A perspetiva apresentada nesta secção permite observar o fluxo de acções e eventos que ocorrem durante a utilização do sistema.

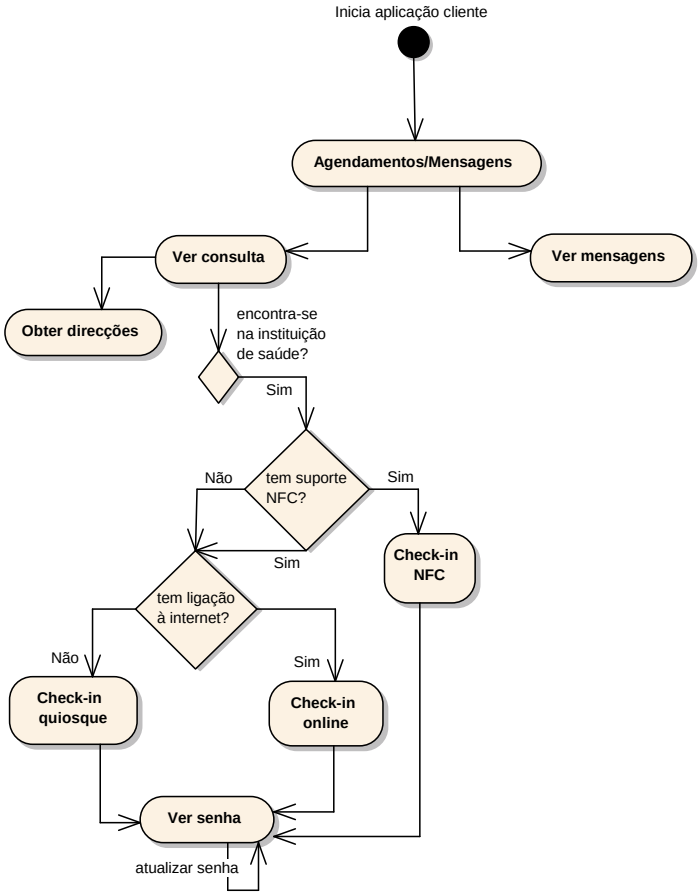


Figura 4.3: Diagrama de Atividade UML para a Aplicação Cliente

A Figura 4.3 coloca em evidência os cenários de validação de uma consulta. Estando o utente dentro da instituição de saúde onde irá realizar a consulta, dispõe de três maneiras de realizar a validação:

1. **Quiosque (manual)**: O utente não possui ligação de dados nem tecnologia *NFC* no dispositivo móvel. Neste caso a Aplicação Cliente gera um código de barras que permite validar a consulta junto do leitor digital presente no Quiosque da instituição de saúde.

   Após a validação, o Quiosque imprime a senha de atendimento; posteriormente o utente poderá consultar novamente o Quiosque e visualizar no ecrã digital o tempo de espera, o número do utente a ser atendido no momento e o número que lhe foi atribuído.

   No caso de o utente possuir ligação de dados mas ter ficado temporariamente sem ligação, é-lhe enviada uma notificação assíncrona pelo Serviço de Push.

2. **NFC**: Se o utente possuir um dispositivo móvel que suporte a comunicação por *NFC*, poderá validar-se junto do Quiosque escolhendo a opção de Check-in NFC dentro de uma consulta. À data da realização deste projeto, apenas os dispositivos *Android* possuíam esta tecnologia. Este modo não exige que o utente possua ligação de dados, e a emissão da senha de atendimento é feita à semelhança do modo Quiosque (manual).

3. **Online**: O utente possui ligação de dados — 3G ou WiFi disponibilizado na instituição — e pode realizar um pedido de Check-in via Servidor Central. Após a validação com sucesso, é enviada ao utente uma notificação assíncrona com os dados da senha de atendimento.

As Figuras 4.4 e 4.5 ilustram o fluxo de acções desencadeadas pelos intervenientes do sistema, onde ocorrem os seguintes eventos:

Autenticação (Figura 4.4)

O utente insere as suas credenciais de acesso — *username* e *password* — que são transmitidas à instituição de saúde. Uma vez validadas, é dada autorização ao utente para realizar operações.

Listar consultas (Figura 4.4)

Após a autenticação com sucesso, o Servidor Central retorna as consultas do utente. Por questões de otimização, os pedidos de listagem de consultas são feitos no Servidor Central e não no Servidor da Instituição, de forma a minimizar os pedidos e tornar o processo mais célere. Isto é possível uma vez que existe um mecanismo de sincronização das bases de dados das instituições com o Servidor Central.

Notificação de consultas e mensagens (Figura 4.4)

Através da interface Web disponibilizada pelo Servidor Central, a Instituição pode enviar mensagens genéricas e de Marketing e emitir novos *tokens* — identificadores de consultas — que são enviados assincronamente para o utente através do Serviço de Push.

Notificação de senhas de atendimento (Figura 4.5)

Via online

O utente realiza o check-in dentro da instituição, o Servidor Central recebe o pedido e transmite-o ao Servidor da Instituição. O pedido é processado e validado, sendo transmitida uma resposta ao Servidor Central, e em caso de check-in válido é enviada uma notificação assíncrona ao utente com a senha de atendimento, através do Serviço de Push.

Via quiosque

O utente realiza o check-in no Quiosque via código de barras ou NFC, e o seu pedido é feito diretamente ao Servidor da Instituição. O pedido é processado e validado, e é

enviada uma notificação assíncrona ao utente com a senha de atendimento. Adicionalmente, o Quiosque recebe o *feedback* da validação e mostra os dados da senha de atendimento no ecrã.

Atualização de senhas de atendimento (Figura 4.5)

Sempre que o utente queira consultar o estado da sua senha de atendimento — i.e. ver o tempo estimado de espera, ver o número do utente atualmente a ser atendido e o seu próprio número — poderá fazê-lo através da Aplicação Cliente, necessitando de ligação de dados para atualizar a senha de atendimento.

Poderá também fazê-lo no Quiosque, bastando para isso realizar novo check-in por código de barras ou NFC. O sistema reconhece que o utente já fez check-in, e em vez de emitir uma nova senha de atendimento atualiza a já existente e mostra-a no ecrã digital presente no Quiosque.

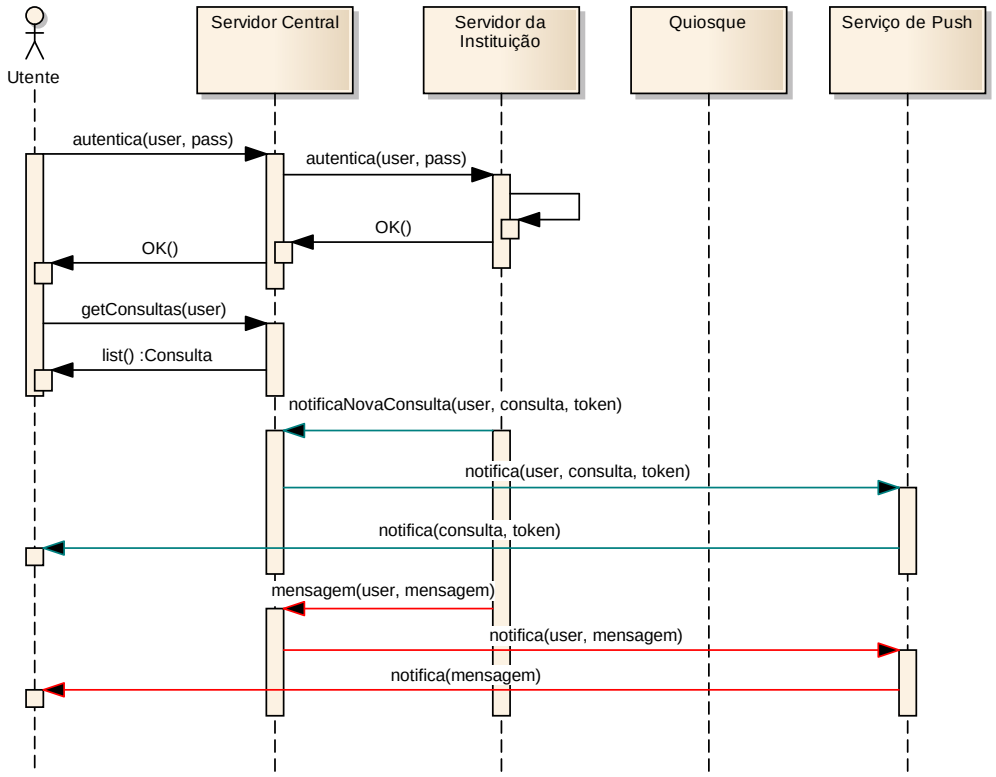


Figura 4.4: Diagrama de Sequência UML — Autenticação, Consultas e Notificações

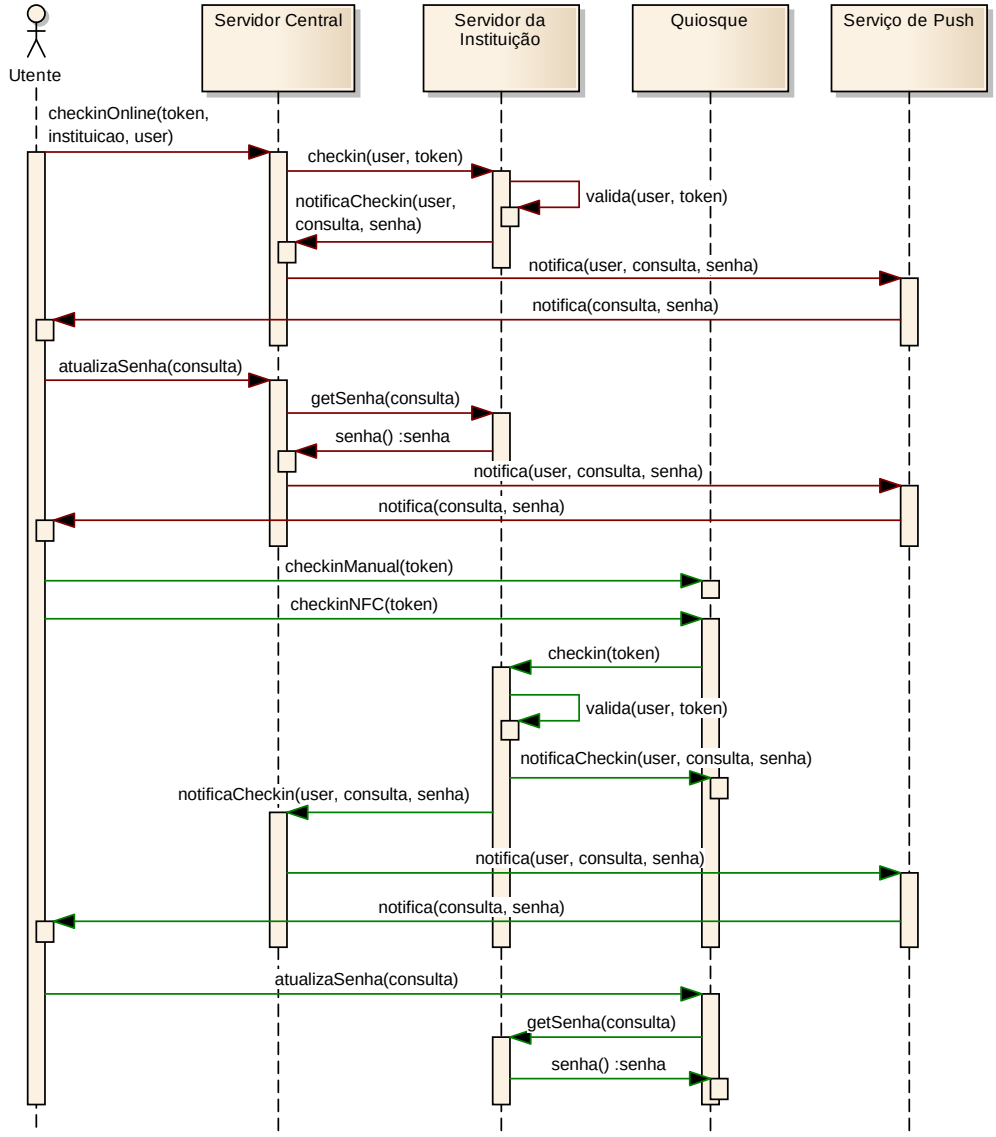


Figura 4.5: Diagrama de Sequência UML — Check-in

Capítulo 5

Implementação

Este capítulo apresentará uma visão mais profunda e de baixo nível dos componentes pertencentes ao sistema. Incluirá a decomposição horizontal de cada componente referido no Capítulo 4 e de todo o *hardware* testado e utilizado ao longo do projeto.

5.1 Aplicação Cliente

Este componente do sistema foi desenvolvido utilizando a plataforma de desenvolvimento móvel multi-plataforma *Appcelerator Titanium 3*, descrito no Capítulo 2. A aplicação para a plataforma *Android* e Web foi desenvolvida em ambiente *Windows* e a aplicação para *iOS* (*iPad* e *iPhone*) desenvolvida em ambiente *Mac OS X*, uma vez que não é possível fazer *deploy* para estes dispositivos sem ser neste sistema operativo.

5.1.1 Decomposição Horizontal

Esta plataforma disponibiliza a *framework* Alloy que opera segundo o paradigma MVC — Model View Controller — que facilita o desacoplamento da camada de dados, lógica e visual da aplicação.

Camada de dados

O modelo de dados da *framework* Alloy é baseada na biblioteca *Backbone.js*. Esta biblioteca fornece estrutura a aplicações baseadas em *Javascript*, oferecendo um modelo chave-valor (*key-value*) com eventos próprios e coleções com várias funções de acesso e iteração. As operações de escrita e leitura de dados são feitas através de operações *RESTful*. As plataformas *Android* e *iOS* permitem a criação de bases de dados *SQLite*, facto que é aproveitado pela plataforma para realizar a persistência de dados.

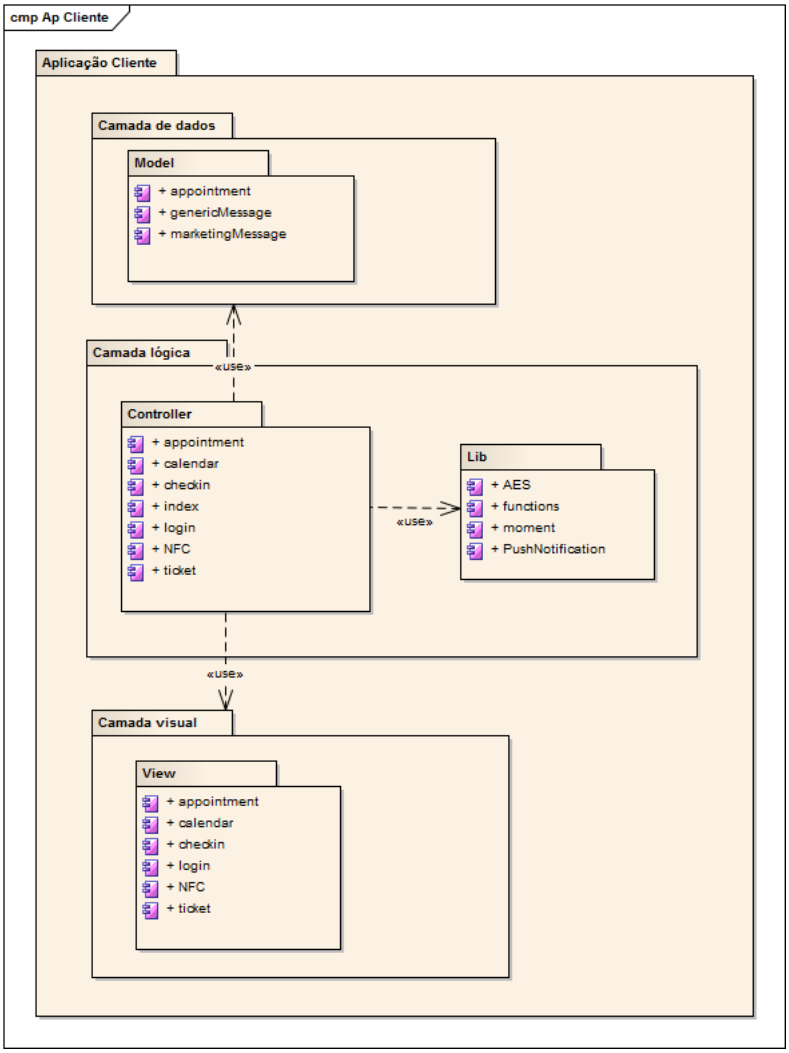


Figura 5.1: Diagrama de Componentes UML — Decomposição horizontal da Aplicação Cliente

Os dados relativos às consultas de um utente retornadas do Servidor Central, incluindo os *tokens* de acesso à consulta, são armazenadas na aplicação de forma a tornar possível o seu acesso quando não existe ligação de dados.

Todas as mensagens de marketing e de notificação enviadas para o utente são também armazenadas. O Código-fonte 5.1 ilustra a estrutura do modelo de dados das mensagens.

O objeto-modelo criado segue a implementação *CommonJS* adotada pelo *Appcelerator Titanium* para a criação de módulos, uma vez que o *Javascript* não possui suporte nativo de *packages*.

O *standard CommonJS* permite a criação de módulos *Javascript* contidos e que não poluem o espaço global. Os métodos e variáveis adicionadas ao objeto *exports* são expostos e acessíveis a partir de outros módulos, sendo todos os outros métodos e variáveis privados. Desta forma é possível a modularização e manutenção mais fácil do código produzido.[Mik]

```
exports.definition = {
  config: {
    columns: {
        "institutionId": "integer",
        "title": "string",
        "body": "string",
        "date": "datetime",
      "read": "integer"
    },
    adapter: {
      type: "sql",
      collection_name: "genericMessage"
    }
  },
  extendModel: function(Model) {
    _.extend(Model.prototype, {
      // extended functions and properties go here
    });

    return Model;
  },
  extendCollection: function(Collection) {
    _.extend(Collection.prototype, {
      // extended functions and properties go here
    });

    return Collection;
  }
};
```

Código-fonte 5.1: Modelo de dados das mensagens

O objeto *config* inclui a definição das colunas e tipos de dados a usar no modelo; o objeto *adapter* define em que formato (*SQL* ou *localStorage*) serão persistidos os dados. É também possível definir funções do modelo e das coleções (conjunto de objetos) adicionando métodos ao objecto *extendModel* e *extendCollection*.

Camada lógica e visual

A Aplicação Cliente adapta-se visualmente às dimensões da plataforma em que corre, tendo sido desenvolvida para dispositivos *Android* de alta, média e baixa densidade e de várias resoluções. No caso do *iPhone* e *iPad*, o aspeto visual é idêntico com a exceção de uma barra de navegação superior.

Os controladores definidos são:

index

Ao iniciar a aplicação, este controlador é corrido. A sua função é a de inicializar variáveis que serão necessárias durante a existência da aplicação. Lança o controlador **login** automaticamente.

login

Exibe o ecrã de autenticação ao utilizador. Envia as credenciais para o Servidor Central e, caso o utente seja validado, regista o utilizador no serviço de notificações *Push* e retorna as consultas do utente. Caso não haja ligação de dados, são carregadas as consultas previamente guardadas. É chamado o controlador **calendar**.

calendar (Figura 5.2)

Mostra uma vista com *tabs* onde o utilizador pode escolher ver um calendário com as suas consultas agendadas e as suas mensagens guardadas. Ao escolher uma consulta, o utilizador é redireccionado para o controlador **appointment**.

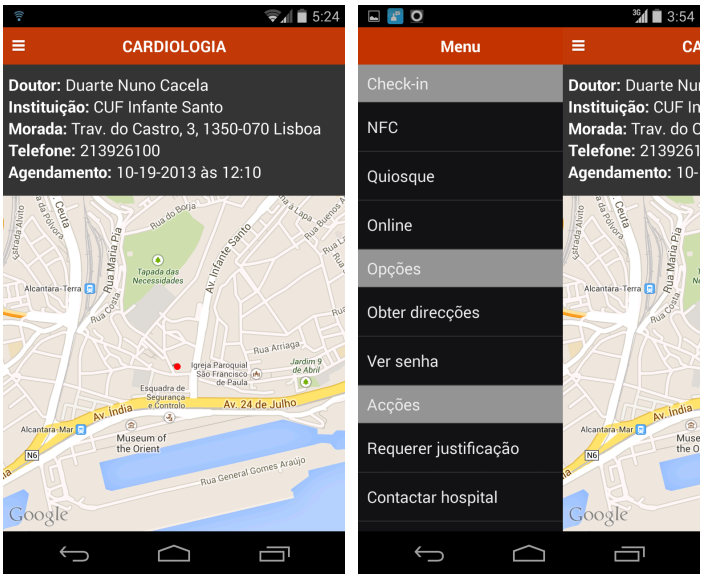
Figura 5.2: Vista do menu **calendar**

appointment (Figura 5.3)

O utilizador pode observar os detalhes da sua consulta e da instituição de saúde onde esta se irá realizar. É também nesta vista que o utilizador se pode validar numa consulta, ver a senha de atendimento, obter direcções para a instituição de saúde e executar acções sobre uma consulta (a implementar futuramente).

checkin (Figura 5.4)

Quando o utilizador opta por realizar a validação numa consulta através do leitor de código de barras do Quiosque, este controlador é invocado, mostrando uma janela com o código de barras gerado a partir do *token*. O tipo de código de barras gerado é o PDF417, sendo feita uma explicação mais rigorosa da razão da sua utilização em 5.4. É de salientar que a geração do código de barras é feito do lado da aplicação cliente por questões de eficiência. Uma análise mais detalhada pode ser observada no Capítulo 6.



(a) Consulta (b) Menu lateral

Figura 5.3: Fluxo de visualização de uma consulta

Figura 5.4: Vista da validação por código de barras

NFC

Este controlador é invocado quando o utilizador opta por realizar a validação numa consulta através do leitor NFC presente no Quiosque. O dispositivo é reconhecido como um *Smartcard* pelo leitor, e lê o *token* contido nele.

ticket (Figura 5.5)

Este controlador é invocado quando o utilizador escolhe a opção **Ver senha** no menu de opções de uma consulta, ou quando recebe uma notificação *Push* após realizar

a validação. Numa senha de atendimento pode ser consultado o número de senha do utente atualmente a ser atendido ("Senha atual"), a senha do próprio utente ("A sua senha") e uma estimativa do tempo de espera. É também possível atualizar esta informação ao longo do tempo. É necessária ligação de dados para ter acesso a uma senha de atendimento, já que os dados nela contidos resultam de uma chamada a um serviço Web.

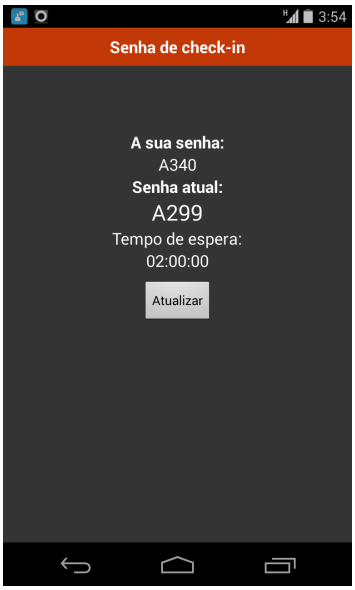


Figura 5.5: Vista de uma senha de atendimento

5.1.2 Geração de símbolos 2D

Apesar de inicialmente a geração de códigos de barras ser feita do lado servidor, concluiu-se que esta solução não era eficiente e comprometia o correto funcionamento do sistema (ver Capítulo 6). Optou-se então por gerar os códigos de barras na Aplicação Cliente. Uma vez que não existiam módulos gratuítos para a plataforma *Appcelerator Titanium* que usassem controlos nativos e adaptar bibliotecas *Javascript* genéricas ao padrão *CommonJS* é complexo e demorado, a solução encontrada passou pela inclusão de uma vista contendo uma *WebView*, capaz de interpretar e exibir código *HTML* e *Javascript*.

Desta forma, é possível incluir bibliotecas *Javascript* que funcionem em qualquer *browser*. Foi utilizada a biblioteca *bwip-js*[War], que permite codificar texto em praticamente todas as simbologias 1D e 2D existentes. Esta versatilidade é da maior importância, já que a decisão sobre qual o leitor de códigos de barras a utilizar no quiosque não é definitiva e poderá mudar. Assim, apesar da versão final deste projeto utilizar a simbologia PDF417, esta poderá facilmente ser alterada no futuro.

5.1.3 NFC

A versão *KitKat* (4.4) do sistema operativo *Android* inclui uma nova funcionalidade designada *HCE — Host-based Card Emulation*. Segundo a documentação oficial, o *HCE* foi implementado para que qualquer aplicação possa simular um *smartcard* NFC, permitindo aos utilizadores realizar transações com leitores NFC. Sendo esta uma funcionalidade muito recente, só alguns dispositivos com chip NFC a podem utilizar, tal como o *Nexus* 4 e 5. Como prova de conceito foi desenvolvido um pequeno módulo *Android* que implementa um serviço para ler e escrever mensagens NFC. A comunicação funciona da seguinte forma:

- Estando o dispositivo móvel em contacto com o leitor NFC, o leitor emite o comando *APDU SELECT <identificador>*.
- O sistema operativo *Android* decide qual o serviço a ser executado baseando-se nesse identificador recebido. Esse identificador — *F0010203040506* — foi definido como sendo da Aplicação Cliente (ver Código-Fonte 5.3).
- O serviço chamado (Código-Fonte 5.2) retorna *null*, já que é necessário que o utilizador confirme qual das consultas pretende validar.
- Quando o utilizador selecciona a consulta e prime o botão **Validar**, é enviado o *token* da consulta para o leitor NFC, que processa a informação recebida.

```
11 public class MyHostApduService extends HostApduService {
12     @Override
13     public byte[] processCommandApdu(byte[] apdu, Bundle extras) {
14        return null;
15     }
16     @Override
17     public void onDeactivated(int reason) {
18     }
19 }
```

Código-fonte 5.2: Serviço Android recetor de pedidos NFC

```
20 <host-apdu-service xmlns:android="http://schemas.android.com/apk/res/android"
21            android:description="@string/servicedesc"
22            android:requireDeviceUnlock="false">
23     <aid-group android:description="@string/aiddescription"
24                android:category="other">
25         <aid-filter android:name="F0010203040506"/>
26     </aid-group>
27 </host-apdu-service>
```

Código-fonte 5.3: apduservice.xml

5.1.4 Algoritmos

Os algoritmos mais relevantes utilizados na Aplicação Cliente são descritos nesta secção.

Advanced Encryption Standard

De forma a impedir o acesso de terceiros à informação de um utente gerada e usada pela Aplicação Cliente, foi usada uma implementação pública da cifra **AES** (*Advanced Encryption Standard*) baseada no algoritmo de *Rijndael*.[Vena] O código foi adaptado de forma a ser compatível com o standard *CommonJS* utilizado pelo *Appcelerator Titanium*. Esta implementação usa o modo de operação *counter* para poder encriptar texto.

Esta cifra exige a presença de uma palavra-passe para a encriptação/desencriptação dos dados, sendo para isso gerada uma *hash* de 256-*bit* conforme sugerido pelo RFC 2898 (*Password-Based Key Derivation Function 2*)[Kal00]. Para tal foi usada uma biblioteca disponível publicamente[Ana] que utiliza uma função de *salt* de 64-*bit* para gerar a chave, conforme recomendado pelo RFC 2898. Segundo o RFC 2898, é difícil pré-computar todas as chaves possíveis correspondentes a um dicionário de palavras-chave. Se o tamanho da função de *salt* for de 64-*bit*, existirão 2^{64} chaves possíveis para cada palavra-chave, tornando-se impraticável percorrer todas as chaves possíveis[Kal00]. O número de iterações (máximo de 10,000 suportadas) também influencia a robustez da chave gerada, sendo recomendado um mínimo de 1000 iterações. De forma a tornar o processo suficientemente célere, foram usadas 2000 iterações.

Haversine

Quando o utilizador pretende fazer a validação Online, é necessário averiguar se este se encontra nas imediações da instituição de saúde. Considera-se para o efeito que um utente se encontra dentro de uma instituição de saúde quando a distância linear entre a sua posição atual e a posição da instituição de saúde é inferior ou igual a 1000 metros. Para calcular esta distância foi utilizada uma biblioteca *Javascript* pública[Venb].

Esta funcionalidade foi testada com sucesso simulando diversas latitudes e longitudes, contribuindo para isso o facto de a tolerância definida ser alta e o algoritmo ser bastante preciso — o erro é geralmente inferior a 0.3%[Venb].

5.2 Servidor Central

O Servidor Central é uma peça fundamental do sistema, que estabelece a ligação entre os vários componentes do sistema (ver 4.2).

Tabela 5.1: Interface do Servidor Central

Nome	Método	Descrição
GetAppointmentsByUser	GET	Recebe o ID de um utente e retorna todos os pares (**data**, **consulta**) desse utente.
GetTicket	GET	Recebe o ID de um episódio de consulta e pede à respetiva instituição de saúde os detalhes de uma senha de atendimento. Estes detalhes incluem o número da senha do utente desse episódio, o número da senha do utente atualmente a ser atendido e o tempo estimado de espera.
AskForCheckin	POST	Função invocada pela Aplicação Cliente aquando da validação Online de uma consulta. Recebe o ID de uma instituição e o *token* de uma consulta e requisita o check-in nessa instituição. Retorna uma mensagem de sucesso ou falha.
NotifyCheckin	POST	Função invocada por uma instituição de saúde quando pretende que um utente receba uma notificação *Push* com a sua senha de atendimento, após ter feito check-in.
NotifyUserOfNewAppointment	POST	Função invocada por uma instituição de saúde quando pretende notificar um utente com o *token* de uma consulta. Recebe um *token* gerado pela instituição e um episódio de consulta e envia uma notificação *Push* ao utente.
RegisterForPushNotification	POST	Função invocada pela Aplicação Cliente para registar um dispositivo de um utente no serviço de notificações *Push*.
SendGenericMessage	POST	Função invocada por uma instituição de saúde quando pretende enviar uma mensagem com conteúdo genérico — alertas de atrasos de pagamento ou outras - a um utente.
SendMarketingMessage	POST	Função invocada por uma instituição de saúde quando pretende enviar uma mensagem de Marketing, mais concretamente um URL que pode ser visto na Aplicação Cliente por um utente.

5.2.1 Decomposição Horizontal

A interface disponibilizada pelo Servidor Central foi desenvolvida na *framework Windows Communication Foundation* e desenvolvida usando o ambiente *Microsoft Visual Studio Professional 2012*, usando a linguagem *C#*. O sistema operativo utilizado foi o *Microsoft Windows 7*. Esta interface expõe os métodos constantes da Tabela 5.1.

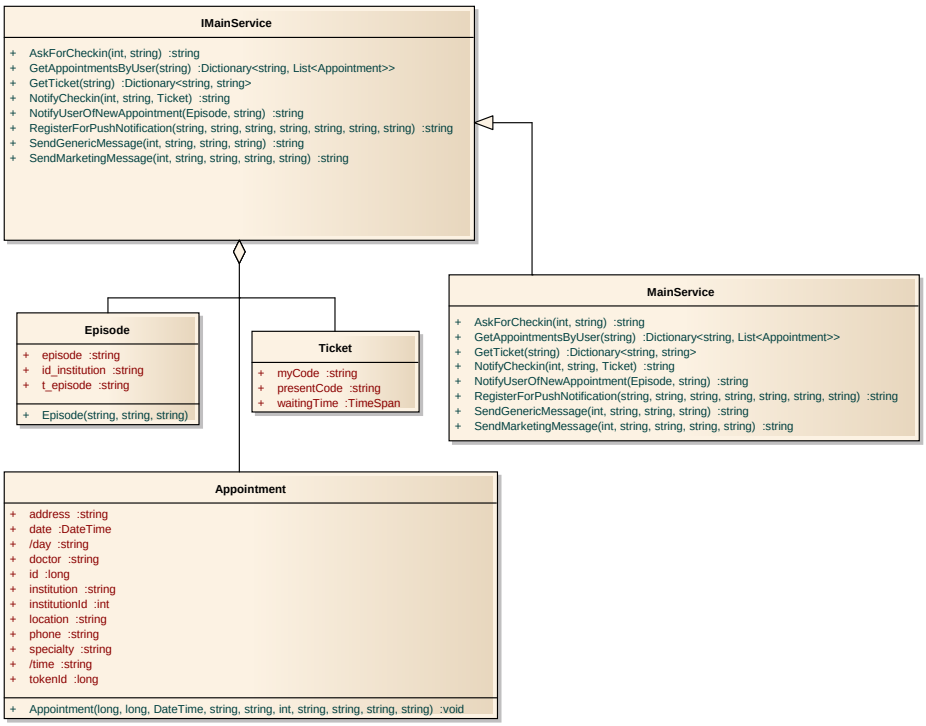


Figura 5.6: Diagrama de classes do Servidor Central

A interface descrita na Tabela 5.1 compreende a camada de Aplicação do Servidor Central, invocando camadas inferiores para aceder aos dados que necessita. O mapeamento relacional foi implementado recorrendo à *ADO.NET Entity Framework*, resultando nas entidades descritas na Figura 5.7.

As classes **MOBCHECKIN_VIEW** e **MOBCHECKIN_EPISODIO_VALUES** foram geradas automaticamente a partir de *Views* definidas em *PL/SQL* numa base de dados *Oracle*.

A classe **MOBCHECKIN_EPISODIO_VALUES** foi projetada para guardar pares {**chave**, **valor**} de uma consulta, sendo atualmente utilizada apenas para guardar *tokens* de uma consulta. Futuramente irá guardar outros dados que sejam considerados relevantes.

A classe **MOBCHECKIN_VIEW** agrega todas as consultas existentes de todos os utentes, incluindo todos os detalhes relevantes para a Aplicação Cliente incluíndo o *token* das consultas, caso existam. Apesar de numa primeira observação este modelo aparentar ser algo simplista, a agregação dos dados de uma consulta é um procedimento complexo com várias operações de *JOIN* (Código-Fonte A.1).

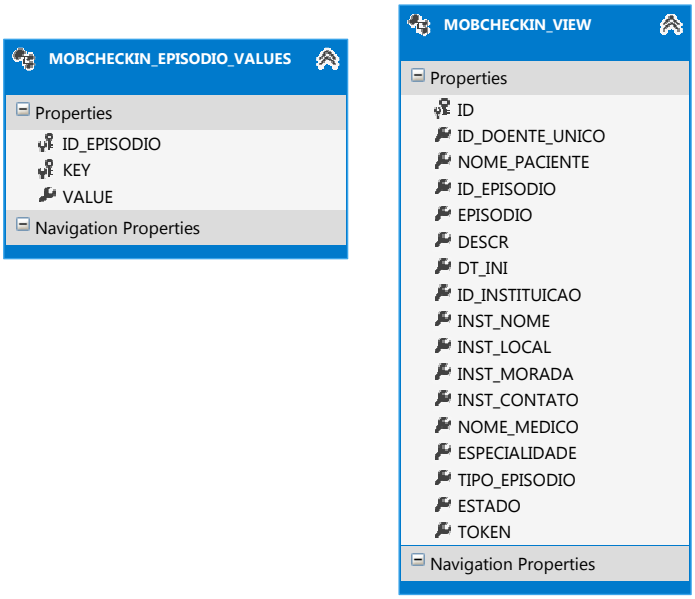


Figura 5.7: Modelo de dados usado pelo Servidor Central

5.2.2 Comunicação

O Código-Fonte 5.4 ilustra os *endpoints* definidos para o acesso ao Servidor Central. Esta configuração inclui o acesso para a Aplicação Cliente através de um *endpoint RESTful* usando o binding *webHttpBinding*, que aceita pedidos HTTP no endereço **http(s)://<endereço>:<porta>/MainService/REST/**. A comunicação entre o Servidor da Instituição e o Servidor Central é feito através de um *endpoint* baseado em *SOAP* usando o *binding basicHttpBinding*, que aceita pedidos no endereço **http(s)://<endereço>:<porta>/MainService/**.

De forma a garantir uma comunicação segura, ambos os *endpoints* permitem a comunicação através de um canal *HTTPS*.

```
28    <services>
29      <service name="Server.MainService">
30        <endpoint address="" binding="basicHttpBinding" name="
              BasicHttpBinding_IMainService" contract="Server.IMainService" />
31        <endpoint address="REST" behaviorConfiguration="restfulBehavior" binding="
              webHttpBinding" contract="Server.IMainService" />
32      </service>
33    </services>
34    <behaviors>
35      <endpointBehaviors>
36        <behavior name="restfulBehavior">
37          <webHttp />
38        </behavior>
39      </endpointBehaviors>
```

```
40      <serviceBehaviors>
41        <behavior>
42          <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
43          <serviceDebug includeExceptionDetailInFaults="true" />
44        </behavior>
45      </serviceBehaviors>
46    </behaviors>
```

Código-fonte 5.4: Configuração de *endpoints*

5.2.3 Interface de acesso Web

Foi desenvolvida uma interface Web *ASPX* usando a biblioteca de *CSS Bootstrap*, que expõe as funcionalidades de notificação de *tokens* de consulta e o envio de mensagens genéricas e de Marketing às instituições de saúde. Esta interface foi desenvolvida por uma questão de conveniência aquando do desenvolvimento do projeto, de forma a poder testar as funcionalidades expostas duma forma rápida. Por outro lado, permite ter uma perspetiva da utilização do sistema por parte do responsável da instituição de saúde.

5.2.4 Serviço de notificações *Push*

Para garantir uma comunicação eficaz entre as instituições de saúde e os utentes foi melhorado um módulo de notificações *Push* usado na *Glintt HS*. Este módulo usa internamente a biblioteca *PushSharp*[Dic], que trata da comunicação com os servidores de notificações da *Google* (*GCM*) e da *Apple* (*APNS*). Desta forma, mesmo não tendo a Aplicação Cliente aberta ou não possuindo ligação de dados na altura, o utente receberá as notificações. De notar que o servidor *APNS* não garante a entrega de mensagens e deve ser utilizado apenas para notificar o cliente de que existem novos dados disponíveis. [App]

Quando um utente instala a Aplicação Cliente num dispositivo e se autentica pela primeira vez, o Servidor Central chama o método **RegisterForPushNotification** do serviço de *Push*, que regista numa base de dados os seguintes atributos:

userId
: ID único de um utente. Se este parâmetro não existir, a mensagem é enviada em *Broadcast*, i.e é enviada para todos os utentes.

deviceId
: *token* atribuído ao dispositivo pelo serviço **GCM** no caso de o dispositivo ser *Android*, ou pelo **APNS** no caso de ser *iOS*.

deviceOS
: Sistema operativo do dispositivo — *Android* ou *iOS*

deviceModel
: Modelo do dispositivo.

deviceVersion

Versão do Sistema Operativo do dispositivo.

appName

Nome do *package* da Aplicação Cliente (com.glintths.mobile.checkin)

appVersion

Versão da Aplicação Cliente

Todas as autenticações subsequentes do utente chamam novamente este método, e se o dispositivo já estiver registado não há quaisquer alterações na base de dados.

Originalmente este módulo permitia apenas o envio de notificações com um título e um corpo de mensagem, tendo sido acrescentada a funcionalidade de incluir pârametros conforme conveniente. O Código-fonte 5.5 indica a estrutura duma notificação *Push* de uma mensagem no formato *JSON*, onde se podem observar alguns dos parâmetros enviados.

```
47 {
48     "message": "Informamos que possui o valor de 1,200 euros em atraso.",
49     "title": "Pagamentos em atraso",
50     "typeOfNotification": "genericMessage",
51     "date": "13-01-2014 15:01:28"
52 }
```

Código-fonte 5.5: Estrutura de notificação *Push* de uma mensagem em *Android*

5.3 Servidor da Instituição

À semelhança do Servidor Central, o servidor de uma instituição (Servidor da Instituição) foi codificado no ambiente *Microsoft Visual Studio Professional 2012* usando a linguagem *C#*. A interface do Servidor da Instituição expõe os métodos constantes da Tabela 5.2

Tabela 5.2: Interface do Servidor da Instituição

Nome	Método	Descrição
GetTicket	GET	Recebe o ID de um episódio de consulta do Servidor Central e retorna os detalhes de uma senha de atendimento. Estes detalhes incluem o número da senha do utente desse episódio, o número da senha do utente atualmente a ser atendido e o tempo estimado de espera.
DoCheckin	POST	Recebe o *token* de uma consulta e realiza o *check-in* desse *token*. O Servidor Central é notificado em caso de sucesso ou falha e é enviada uma notificação *Push* ao utente.

5.3.1 Comunicação

Na configuração dos *endpoints* do Servidor da Instituição apenas existe um *endpoint* que usa o *binding basicHttpBinding*, não existindo um ponto de acesso *REST* já que ele não é necessário. A configuração do Servidor pode ser vista no excerto Código-Fonte 5.6.

```
53      <services>
54       <service name="InstitutionServer.InstitutionService"
55         <endpoint address="" binding="basicHttpBinding" contract="InstitutionServer
               .IInstitutionService" />
56       </service>
57     </services>
58     <behaviors>
59       <serviceBehaviors>
60         <behavior>
61           <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true"/>
62           <serviceDebug includeExceptionDetailInFaults="true"/>
63         </behavior>
64       </serviceBehaviors>
```

Código-fonte 5.6: Configuração de *endpoints*

5.4 Quiosque

O Quiosque existe em virtude do Servidor da Instituição, simulando o *firmware* que irá correr dentro de um Quiosque. Para atingir este objetivo foi desenvolvida uma aplicação de console em *C#*, que recorre à biblioteca **Microsoft Point of Service for .NET v1.12** para estabelecer a ligação entre os sensores de validação — leitor de código de barras e terminal NFC.

5.4.1 Microsoft Point of Service for .NET

Segundo a documentação, esta biblioteca simplifica o desenvolvimento de aplicações fornecendo uma *API* implementada em cima da arquitetura *.NET*. o *Point of Service for .NET* simplifica o desenvolvimento de serviços para periféricos fornecendo um conjunto de classes *.NET* para os dispositivos mais comuns.[Mic].

O excerto Código-Fonte 5.7 ilustra o funcionamento desta biblioteca para o leitor de códigos de barras.

O dispositivo é inicializado e reclamado pelo sistema por tempo indefinido. Sempre que é recebido um novo código de barras, o método **activeScanner_DataEvent** é corrido. Este método processa o código recebido e remete-o para a Instituição de Saúde à qual pertence.

```
65     try
66     {
```

```
67          activeScanner = (Scanner)explorer.CreateInstance(selectedScanner);
68          activeScanner.Open();
69          activeScanner.Claim(0);
70          activeScanner.DeviceEnabled = true;
71          activeScanner.DataEvent += new DataEventHandler(activeScanner_DataEvent);
72          activeScanner.ErrorEvent += new DeviceErrorEventHandler(
                activeScanner_ErrorEvent);
73          activeScanner.DecodeData = true;
74          activeScanner.DataEventEnabled = true;
75      }
76      catch (PosControlException)
77      {
78          // Log error and set the active scanner to none
79          Console.WriteLine(string.Format("Activation Failed: {0}", selectedScanner.
                ServiceObjectName));
80          activeScanner = null;
81      }
```

Código-fonte 5.7: *Handlers* subscritos pelo leitor de código de barras

5.4.2 Leitores de símbolos

Para a leitura dos códigos de barras foram estudados dois dispositivos fornecidos pela *Glintt*, ambos capazes de ler símbolos em superfícies altamente refletoras como é o caso dos *smartphones* e *tablets*:

Datalogic Magellan 1100i

Leitor de símbolos 1D e 2D capaz de interpretar as simbologias 2D PDF417, Datamatrix, QR Code, Maxicode e Aztec. É capaz de interpretar as simbologias 1D mais correntes, tal como UPC, EAN e Code 128. O leitor foi configurado de forma a ler todas as simbologias e a funcionar em fracas condições de iluminação.

Pelos testes realizados com este dispositivo concluiu-se que tem alguma dificuldade em interpretar códigos QR, falhando bastantes vezes ou demorando mais do que o esperado; os problemas de *timeout* foram frequentes.

No entanto, é capaz de ler símbolos lineares — i.e que são lidos por varrimento ao invés de serem interpretados — muito rapidamente e corretamente.

A opção de usar este leitor foi entretanto descartada, uma vez que, dado o valor da unidade, a sua integração nos quiosques das instituições representava um custo elevado para a *Glintt*.

Motorola ds457

Leitor de símbolos 1D e 2D mais compacto e acessível que o Magellan 1100i. As caraterísticas deste dispositivo são semelhantes às do Magellan 1100i, apresentando também falhas na leitura de códigos QR.

Por este dispositivo ser compacto, a área de projeção de infravermelhos é inferior à do Magellan 1100i, o que implica que, para ler códigos de maior dimensão, se tenha que afastar o dispositivo móvel do leitor, comprometendo a qualidade e tempo de leitura. Como tal, a taxa de sucesso na leitura de códigos PDF417 foi inferior ao Magellan 1100i, já que este tipo de símbolo varia de tamanho conforme a informação que contém.

Usando códigos de barras tradicionais como o UPC-A foram obtidos bons resultados, no entanto este tipo de símbolo não deverá ser uma opção devido à sua limitação de tamanho — codifica no máximo 12 carateres.

5.4.3 Leitor de NFC

Para a comunicação NFC entre o dispositivo móvel e o quiosque foi estudado o seguinte leitor, disponibilizado pela *Glintt*:

HID Omnikey 5321

Leitor/escritor de *smartcards* (conhecidas como *tags*) NFC, capaz de comunicar com os modelos mais comuns como o MiFare, DESFire, Infineon My-d, entre outros.

As limitações deste dispositivo para tudo o que não seja escrita ou leitura de *smartcards* são consideráveis, não possuindo suporte para o modo *P2P* usado pelo *Android Beam* para emparelhar dois dispositivos NFC.

O modo de emulação de cartão — o leitor age como se fosse um *smartcard* estando por isso recetivo à escrita por qualquer dispositivo — também não é suportado por este leitor.

As opções restantes são a de usar uma *tag* NFC auxiliar, que seria escrita a partir da Aplicação Cliente e apresentada ao leitor. Esta opção não foi considerada viável, uma vez que obrigava o utente a trazer consigo uma *tag* sempre que quisesse validar o dispositivo.

Para finalizar, resta a opção de modo de emulação de cartão no cliente, suportado em alguns dispositivos *Android* a partir da versão *KitKat* (4.4.x), em que o dispositivo móvel simula uma *tag* NFC que pode ser lida por qualquer dispositivo.

Capítulo 6

Testes

Durante a fase de implementação foram testados alguns pontos críticos relativos ao funcionamento do sistema.

6.1 Testes no Servidor Central

Foram realizados testes de carga no servidor para aferir o seu tempo de resposta a pedidos feitos pela Aplicação Cliente.

Tabela 6.1: Testes de carga

Teste	Frequência de pedidos	Nº pedidos	Completados	Descartados	Taxa de sucesso
#1	10/segundo	851	603	248	70%
#2	30/segundo	310	221	89	71.2%
#3	10/segundo	692	692	0	100%
#4	30/segundo	1993	1993	0	100%

O método invocado pela Aplicação Cliente nestes testes foi o **GetAppointmentsByUser**, que retorna todas as consultas de um utente. A escolha deste método para teste deveu-se ao facto de ser o mais intensivo em termos de recursos, já que envolve retornar uma grande quantidade de dados e transmiti-los pela rede.

Os testes #1 e #2 incluíam a geração do código digital (QR) da consulta, enviando um *bitmap* de resolução 256 por 256 como uma *string* de *base64*. Os testes #3 e #4 não incluíam a geração de código.

Estes testes permitiram concluir que gerar os códigos digitais no Servidor Central, independentemente da frequência dos pedidos, não seria uma opção viável e seria necessário gerá-los do lado da Aplicação Cliente.

6.2 Testes na Aplicação Cliente

6.2.1 Testes de memória

A geração dos meses do calendário na Aplicação Cliente é feita de forma estática — i.e são vistas carregadas em memória antes da atividade iniciar — de forma a tornar a aplicação mais usável. A criação dinâmica dos meses revelou-se demasiado lenta nos dispositivos com menor *performance* e foi descartada.

Tornou-se então importante ter os recursos de memória em atenção e procurar carregar um número útil de meses para um utilizador conseguir ter uma noção do seu calendário de consultas, sem comprometer a *performance* da aplicação.

Tabela 6.2: Testes de memória

Número de meses	Número de eventos	Memória alocada	Memória utilizada
6	71	4,4 MB	60,55%
12	71	5,2 MB	64,42%
48	8	10,9 MB	69,52%
48	71	12,14 MB	75,87%

Pode-se assumir que raramente acontecerá o caso de um utente ter mais de 70 eventos. Assim sendo, carregando 12 meses no calendário conseguir-se-á obter um valor de memória alocada inferior a 6 MB, obtendo uma *performance* aceitável na maioria dos dispositivos.

6.2.2 Dispositivos testados

No decurso do projeto, a aplicação cliente foi testada nos seguintes dispositivos *Android*:

Nexus 4

Processador: Quad-core 1.5 GHz Krait
Memória: 2 GB RAM
Ecrã: 768 x 1280 pixels, 4.7 polegadas
Sistema Operativo: KitKat 4.4

Nexus 7

Processador: Quad-core 1.2 GHz Cortex-A9
Memória: 1 GB RAM
Ecrã: 800 x 1280 pixels, 7 polegadas
Sistema Operativo: KitKat 4.4

Huawei G300

Processador: Qualcomm MSM7227A Snapdragon
Memória: 512 MB RAM
Ecrã: 480 x 800 pixels, 4.0 polegadas
Sistema Operativo: Ice Cream Sandwich 4.0

Optimus LG-P500

Processador: Qualcomm MSM7227
Memória: 512 MB RAM
Ecrã: 320 x 480 pixels, 3.2 polegadas
Sistema Operativo: Gingerbread 2.3.7

O desenvolvimento para iOS foi testado nas seguintes plataformas/dispositivos:

Emulador iOS 7

Este emulador foi testado em ambiente *Mac OS X*, simulando o sistema operativo **iOS 7** e os dispositivos *iPhone* 5 e *iPad* 4.

iPad 2

Processador: Dual-core A5
Memória: 512 MB RAM
Ecrã: 1024 x 768 pixels, 9.7 polegadas
Sistema Operativo: iOS 6

A aplicação obteve um bom desempenho em todos os dispositivos utilizados. Como seria expectável, a utilização é mais rápida e agradável em dispositivos com melhores caraterísticas; no entanto, é perfeitamente usável em dispositivos de baixa gama como o *Optimus LG-P500*.

Capítulo 7

Conclusões e Trabalho Futuro

7.1 Satisfação dos Objetivos

Pode-se concluir que foram satisfeitos os principais objetivos propostos, tendo sido desenvolvido um sistema testado e funcional.

O objetivo principal de usar os dispositivos móveis como meio de validação em consultas foi atingido em pleno, sendo incluidas funcionalidades que não estavam previstas inicialmente como o caso da validação online via *Web Services* e o sistema de notificações *Push*.

Através da implementação de um canal seguro de comunicação foi possível cumprir o objetivo de assegurar a confidencialidade dos dados transmitidos pelos diversos intervenientes do sistema.

Conseguiu-se desenvolver uma Aplicação Cliente funcional para as principais plataformas pretendidas — *Android* e *iOS*. Para além destas foi desenvolvida uma versão *Web*. O facto de existir uma variedade de dispositivos *Android* com *performances* e resoluções distintas exigiu que fosse dedicado tempo adicional a esta plataforma, tendo-se conseguido um resultado funcional em dispositivos de baixa, média e alta gama.

A validação por *NFC* foi implementada parcialmente, sendo produzida uma implementação simples mas capaz de provar que esta opção é viável como meio de validação.

Um dos problemas da solução inicial consistia na grande quantidade de tráfego gerado entre a Aplicação Cliente e o Servidor Central. De forma a superar este problema, optou-se por gerar os códigos digitais na Aplicação Cliente ao invés de os gerar no Servidor Central. Esta alteração retirou uma carga considerável do Servidor Central, reduzindo o tráfego gerado de forma a permitir que todo o sistema se mantenha funcional mesmo com um número elevado de clientes.

A arquitetura optada para o desenvolvimento dos servidores — Central e da Instituição — é flexível e escalável; futuramente poderão ser adicionadas funcionalidades caso a empresa sinta tal necessidade.

Pelas várias funcionalidades implementadas e pelo nível de cumprimento dos requisitos, o protótipo produzido demonstrou ir ao encontro das necessidades de uma aplicação comercial. Por este motivo, a Glintt demonstrou interesse em integrar esta solução num projeto maior.

7.2 Trabalho Futuro

Pelo facto de o projeto desenvolvido ser um protótipo, existem melhoramentos que poderão ser introduzidos.

Não foi possível implementar o módulo de autenticação de utentes, tendo sido dada prioridade a outros requisitos considerados mais importantes. Será algo a incluir em iterações futuras do projeto.

Por se ter optado por desenvolver a Aplicação Cliente usando a *framework Appcelerator Titanium*, o *deploy* para a plataforma *Windows Phone* foi adiado até esta plataforma ser suportada, algo que é previsto pela empresa responsável. No entanto, esta plataforma foi considerada de menor prioridade para a *Glintt* em relação a *Android*, *iOS* e *Web*.

Seria importante estudar um dispositivo NFC mais habilitado à comunicação com dispositivos móveis, levando assim ao desenvolvimento de um módulo de validação mais eficaz.

O sistema de notificações *Push* poderá ser melhorado notificando o cliente de que existem novos dados disponíveis ao invés de os enviar diretamente, eliminando a possibilidade de poder haver alguma perda de informação.

Bibliografia

[AA11] Laith Alrubaiee and Feras Alkaa'ida. The mediating effect of patient satisfaction in the patients' perceptions of healthcare quality – patient trust relationship. *International Journal of Marketing Studies*, 3(1), 2011.

[Ana] Parvez Anandam. Password-based key derivation function 2 (pbkdf2). a javascript implementation (version 1.5). Disponível em `http://http://anandam.name/pbkdf2/`, acedido a última vez em 8 de Janeiro de 2014.

[App] Apple. Apns. Disponível em `https://developer.apple.com/library/mac/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html`, acedido a última vez em 13 de Janeiro de 2014.

[Bah] Deutsche Bahn. Startseite - touch and travel. Disponível em `http://www.touchandtravel.de`, acedido a última vez em 18 de Julho de 2013.

[BLFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945 (Informational), May 1996.

[Bro] Jeff Brown. Zbar bar code reader. Disponível em `http://zbar.sourceforge.net`, acedido a última vez em 22 de Julho de 2013.

[Com] American Express Company. What is expresspay? Disponível em `http://www.americanexpress.com/expresspay`, acedido a última vez em 18 de Julho de 2013.

[Dic] Jon Dick. Pushsharp. Disponível em `https://github.com/Redth/PushSharp`, acedido a última vez em 13 de Janeiro de 2014.

[EJ97] D. Eastlake and P. Jones. Rfc 3174. Technical report, Network Working Group, 1997.

[For] NFC Forum. Nfc forum: home. Disponível em `http://www.nfc-forum.org/`, acedido a última vez em 18 de Julho de 2013.

[For11] NFC Forum. Nfc in public transport. Technical report, January 2011.

[FT11] Luka Finzgar and Mira Trebar. Use of nfc and qr code identification in an electronic ticket system for public transport. pages 1–6. Software, Telecommunications and Computer Networks (SoftCOM), 19th International Conference on, September 2011.

[GM09] Denis Gracanin Gavin Mulligan. A comparison of soap and rest implementations of a service based interaction independence middleware framework. Winter Simulation Conference, 2009.

[Har12] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), October 2012.

[HG00] Marine Minier Henri Gilbert. A collisions attack on the 7-rounds rijndael. Centre National d'Etudes des Télécommunications, 2000.

[IBM] Verisign IBM, Microsoft. Web services security (ws-security). Disponível em `http://people.cs.vt.edu/~kafura/cs6204/Readings/WebServices/WS-security.pdf`, acedido a última vez em 18 de Janeiro de 2014, Abril.

[JD00] Vincent Rijmen Joan Daemen. Rijndael. NISSC, 2000.

[JD02] Vincent Rijmen Joan Daemen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[JLC12] Glenn Ergeerts Rud Beyers Karel Renckens Luc Wante Jens Luz, Frederik Schrooyen and Marc Ceulemans. A mobile nfc payment terminal for the event-wallet on an android smartphone. 2012.

[Kal00] B. Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. RFC 2898 (Informational), September 2000.

[Ken] Fukuchi Kentaro. libqrencode. Disponível em `http://fukuchi.org/works/qrencode/`, acedido a última vez em 22 de Julho de 2013.

[KW12] Ravindra Thool Kishor Wagh. A comparative study of soap vs rest web services provisioning techniques for mobile host. *Journal of Information Engineering and Applications*, 2(5):12–17, 2012.

[LL06] Yue Liu and Mingjun Liu. Automatic recognition algorithm of quick response code based on embedded system. Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06), 2006.

[LMCC10] David Conde Lagoa, Enrique Costa Montenegro, Francisco J. González Castaño, and Felipe Gil Castiñeira. Secure etickets based on qr-codes with user-encrypted content. pages 257–258. Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on, 2010.

[LRL10] Izabela Lacmanovic, Biljana Radulovic, and Dejan Lacmanovic. Contactless payment systems based on rfid technology. MIPRO, Maio 2010.

[Ltd10] Mobey Forum Mobile Financial Services Ltd. Alternatives for banks to offer secure mobile payments. Technical report, 2010.

[Luc00] Stefan Lucks. Attacking seven rounds of rijndael under 192-bit and 256-bit keys. AES Candidate Conference, 2000.

[Mar95] Mark Marriott. Pdf417 portable data files a new dimension in barcodes. *Sensor Review*, 15(1):33–35, 1995.

[Mas] Mastercard. What is paypass nfc? Disponível em `http://www.mastercard.com/us/paypass/phonetrial/whatispaypass.html`, acedido a última vez em 18 de Julho de 2013.

[Mic] Microsoft. Microsoft pos for .net overview. Disponível em `http://msdn.microsoft.com/en-us/library/ms828083(v=winembedded.10).aspx`, acedido a última vez em 14 de Janeiro de 2014.

[Mik] Zach Carter Hannesw Tobie Ondřej Žára Charles Jolley KrisKowal Chris Zumbrunn Gmsox Mikeal, Gozala. Commonjs api specification - modules/1.1.1. Disponível em `http://wiki.commonjs.org/wiki/Modules/1.1.1`, acedido a última vez em 02 de Janeiro de 2014.

[MPL04] D. A. Moraes, I. T. Pisa, and P. R. L. Lopes. Protótipo para coleta de informações em saúde utilizando dispositivos móveis. 2004.

[OoPT12] GSMA London Office and International Association of Public Transport. Mobile nfc in transport. Technical report, September 2012.

[Per] Adrian Perrig. Shortcomings of password-based authentication. Disponível em `https://www.usenix.org/legacy/events/sec2000/full_papers/dhamija/dhamija_html/node2.html`, acedido a última vez em 15 de Janeiro de 2014, Junho.

[PHAC12] José Pirrone Puma, Mónica Huerta, Rodolfo Alvizu, and Roger Clotet. Mobile identification: Nfc in the healthcare sector. Andean Region International Conference, 2012.

[Soo08] Tan Jin Soon. Qr code. *Synthesis*, 2008.

[US12] GS1 US. Gs1 barcode chart, Abril 2012.

[Vena] Chris Veness. Movable type scripts — aes advanced encryption standard. Disponível em `http://www.movable-type.co.uk/scripts/aes.html`, acedido a última vez em 8 de Janeiro de 2014.

[Venb] Chris Veness. Movable type scripts — calculate distance, bearing and more between latitude/longitude points. Disponível em http://www.movable-type.co.uk/scripts/latlong.html, acedido a última vez em 8 de Janeiro de 2014.

[Vis] Visa. Visa paywave. Disponível em http://www.visaeurope.com/en/cardholders/visa_paywave.aspx, acedido a última vez em 18 de Julho de 2013.

[W3C] W3C. Soap version 1.2. Disponível em http://www.w3.org/TR/2007/REC-soap12-part1-20070427/, acedido a última vez em 17 de Janeiro de 2014, Abril.

[War] Mark Warren. bwip-js - barcode writer in pure javascript. Disponível em https://code.google.com/p/bwip-js/, acedido a última vez em 14 de Janeiro de 2014.

[WYY05] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. *Advances in Cryptology - CRYPTO 2005*, pages 17–36. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005.

[Yed12] Vineeth Yeddula. Healthcare quality: Waiting room issues. Master's thesis, University of Nebraska, Agosto 2012.

[ZXi] ZXing. zxing - multi-format 1d/2d barcode image processing library with clients for android, java. Disponível em http://code.google.com/p/zxing/, acedido a última vez em 22 de Julho de 2013.

Anexo A

Anexos

```
select distinct
    doentes.id_doente_unico,
    doentes.nome,
    vista.uniqueepisidepisodio,
    vista.uniqueepisepisodio,
    vista.uniqueepisdescr,
    vista.uniqueepisdtini,
    vista.uniqueepisidinstituicao,
    instituicoes.nome,
    instituicoes.localizacao,
    instituicoes.morada,
    instituicoes.contato,
    medicos.nome,
    vista.medicalspecialitydescr,
    vista.uniqueepistepisodio,
    vista.uniqueepisflgestado,
    valores.value
from
    MOBCHECKIN_EPISODIO_VALUES valores full outer join
        SD_DOENTE_UNICO doentes inner join
            GR_MEDICOS medicos inner join
                    GRV_EPISOD_SYNC vista inner join
                        MOBCHECKIN_INSTITUICAO instituicoes
                    on vista.uniqueepisidinstituicao=instituicoes.id_instituicao
            on vista.MAINDOCTOR=medicos.ID_MEDICO
        on vista.uniqueepisiddoenteunico=doentes.id_doente_unico
    on valores.id_episodio=vista.uniqueepisidepisodio WHERE valores.key='TOKEN' OR
        valores.key IS NULL) order by vista.uniqueepisdtini asc
```

Código-fonte A.1: Vista MOBCHECKIN_VIEW