Universidade de Aveiro

Departamento de Eletrónica,
Telecomunicações e Informática

Engenharia de Serviços (2014/2015)
Relatório Trabalho Prático

We Care Services (WCS)

Grupo:
Tiago Soares NºMec: 60456
Hugo Frade NºMec: 59399
André Correia NºMec: 60411
Tiago Lourenço NºMec: 59696

Docente:
Prof. Diogo Gomes

20 de Dezembro de 2014

Conteúdo

Introdução

Este trabalho prático incide sobre toda a matéria leccionada nas aulas de Engenharia de Serviços.

No início do semestre foi-nos proposto escolhermos um tema para o nosso projecto e desde a nossa primeira reunião surgiu-nos várias ideias, entre as quais desenvolver uma plataforma web que permita notificar os eventos, sejam eles musicais, desportivos, entre outros; plataforma web onde os pacientes possam efectuar a marcação de consultas médicas e dar a possibilidade de o paciente ser notificado sobre estas, entre muitas outras ideias. Mas desde cedo percebemos que a segunda hipótese era um tema muito mais interessante e mais enriquecedor/útil para a comunidade. Depois de ter sido escolhido o tema, atribuímos um nome ao nosso projecto: We Care Services (WCS), com o intuito de os nossos serviços serem uma mais valia para os pacientes que pretendam efectuar consultas médicas.

O objectivo deste trabalho consiste em desenvolver uma plataforma web que permita que os pacientes possam efectuar consultas médicas sem terem de sair das suas próprias casas. Pretendemos também alertar o utilizador da hora/dia da consulta médica através da implementação de serviços de notificação: envio de emails, de SMS para o paciente. Com a plataforma Web pretendemos mostrar ao paciente a lista de consultas disponíveis com base em diferentes especialidades médicas, efectuar o login a partir do Facebook ou até mesmo do Gmail e ainda visualizar um calendário de marcações médicas. Tudo isto vai ser abordado mais à frente neste relatório com mais detalhe.

Durante este relatório tentamos explicar pormenorizadamente tudo aquilo que foi feito durante todo o semestre por forma a que os objetivos fossem alcançados com sucesso e realizados da melhor maneira possível para que a integração entre serviços fosse bem sucedida, não esquecendo de otimizar todo o código necessário.

Serviços - Arquitectura

Dado que cada elemento de grupo teria de desenvolver um serviço diferente na aplicação, foram pensados quatro serviços distintos que possibilitassem a criação de um sistema de marcação de consultas e alerta de notificações sem ter de se deslocar ao centro de saúde ou hospital. Esses serviços são os seguintes: Web Site, Notificações, Autenticação do Utilizador e *Schedule*.

Vamos mostrar agora o diagrama de arquitectura simplificado do nosso sistema.

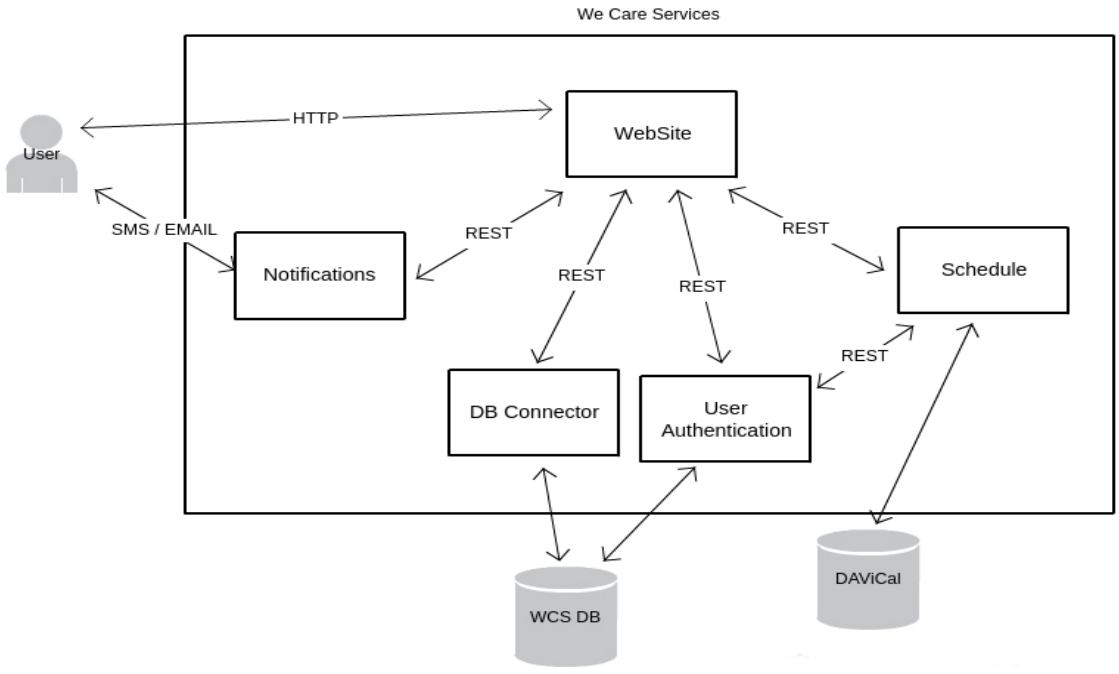

Figura 1: Arquitectura Sistema We Care Services

Todos os componentes desta arquitectura correspondem aos serviços previamente mencionados, à excepção do módulo auxiliar de conecção à base de dados que permite ao módulo `Web Site` efectuar algumas operações na base de dados, nomeadamente alterar a informação de perfil do utilizador e dados estatísticos de consultas.

Podemos constatar que a comunicação entre os diferentes serviços é efectuada através de *ReSTful Webservices*, sendo que as respectivas interfaces de cada módulo serão explicadas ao longo do relatório.

Agora vamos apresentar os vários diagramas de casos de uso referentes ao nosso sistema. Vamos começar por apresentar o *package* de autenticação (ver Fig.1) onde apresentamos as diversas ações em termos de autenticação que um simples utilizador pode executar no sistema. A autenticação pode ser feita de dois modos distintos: ou através do método de autenticação do Facebook ou através do método de autenticação do Gmail.

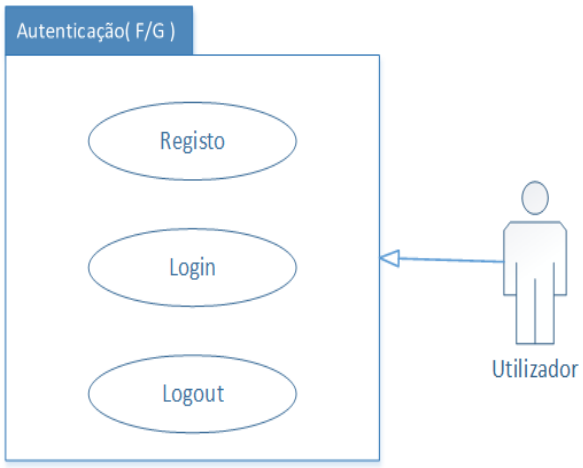

Figura 2: Módulo de Autenticação

Posteriormente apresentamos um *package* associado às diversas ações que um paciente pode tomar no nosso sistema (ver Fig.2). Um paciente pode efectuar uma marcação de uma consulta médica seleccionando um determinado médico associado a uma determinada especialidade, pode visualizar o calendário do doutor seleccionado e visualizar dados da consulta que marcou.

Depois de a consulta ter sido marcada com sucesso, este pode visualizar uma mensagem no seu telemóvel a indicar a hora/dia da consulta ou então recebe um email que contém um ficheiro *ics* que indica também a hora/dia da consulta. O paciente pode ainda desmarcar qualquer consulta.

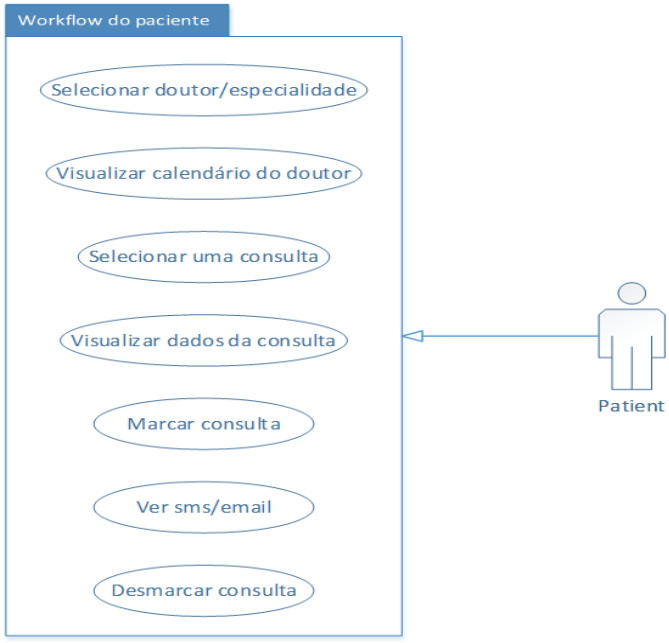

Figura 3: Módulo de Workflow do paciente

Na próxima figura apresentamos quais é que são as ações unicamente acessíveis a qualquer utilizador registado no nosso sistema. Este pode consultar os seus dados pessoais, alterá-los, visualizar o seu histórico de consultas e ainda visualizar o calendário de consultas onde pode verificar quais é que são as consultas marcadas ou as que ainda estão disponíveis.

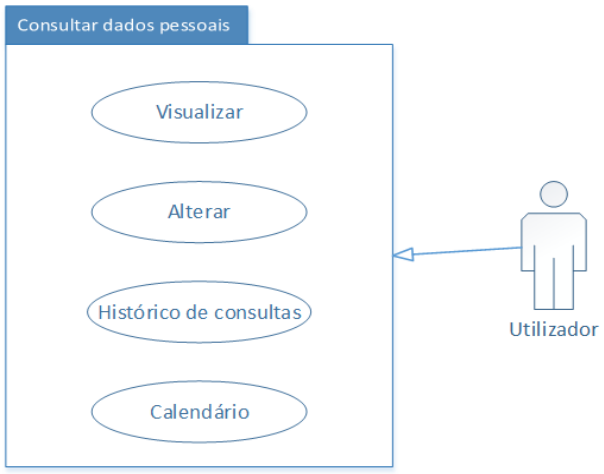

Figura 4: Módulo de visualização de Dados Pessoais

Por último, apresentamos mais dois *packages* em que o primeiro representa as ações que um doutor e um administrador podem desempenhar no nosso sistema e o outro *package* é unicamente acessível ao administrador (ver Fig.4).
Um doutor no nosso sistema pode visualizar a sua lista de pacientes, selecionar um paciente e visualizar o histórico de consultas do mesmo, visulizar o seu calendário, seleccionar uma determinada consulta, verificar os dados de uma consulta e tem ainda a possbilidade de cancelar a mesma.

Já o administrador pode efectuar as mesmas acções que foram referidas anteriormente, ou seja, todas as acções que um doutor no nosso sistema pode efectuar, e ainda pode visualizar todos os pacientes que existem na nossa aplicação, enviar emails/SMS's particulares para qualquer paciente (desejar boas festas, por exemplo), visualizar calendários dos pacientes, visualizar a lista de todos os doutores que estão registados no nosso sistema, visualizar os calendários destes e ainda pode consultar a *dashboard* onde pode visualizar quantas consultas foram marcadas num determinado mês, por exemplo.

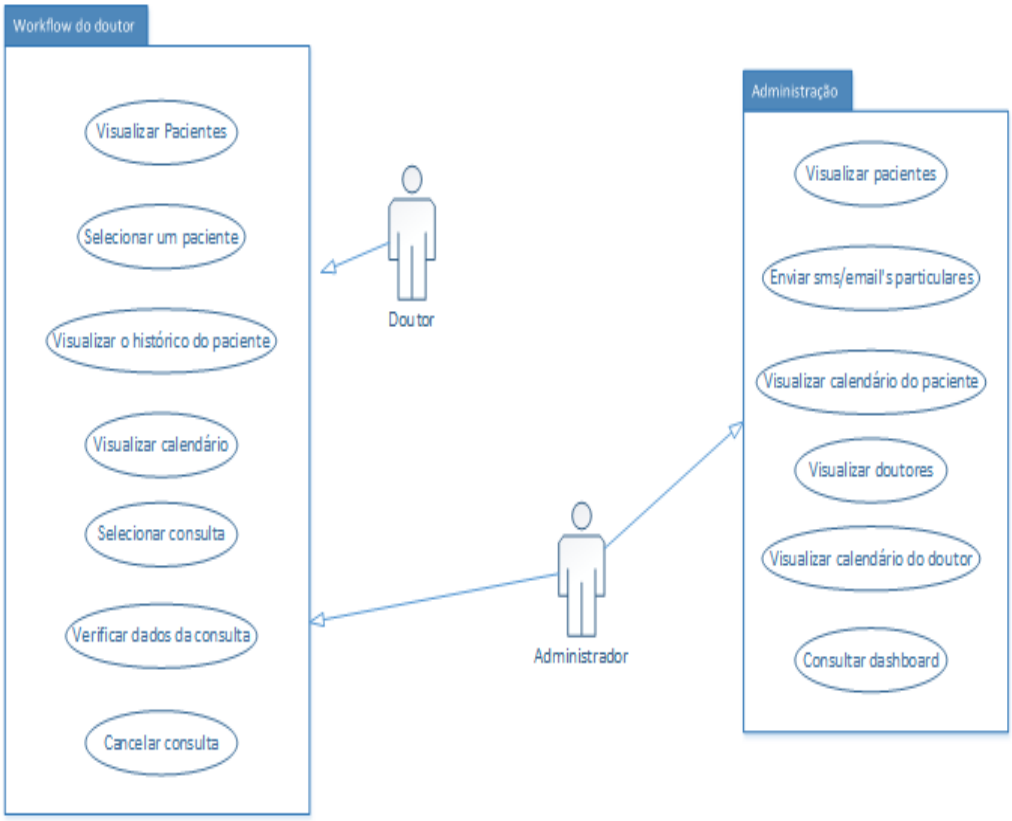

Figura 5: Módulo do workflow do doutor e do workflow da administração

Apresentamos de seguida os serviços que foram implementados, o tipo de linguagens que foi usado para cada um e a respectiva atribuição a cada um dos diferentes elementos de grupo:

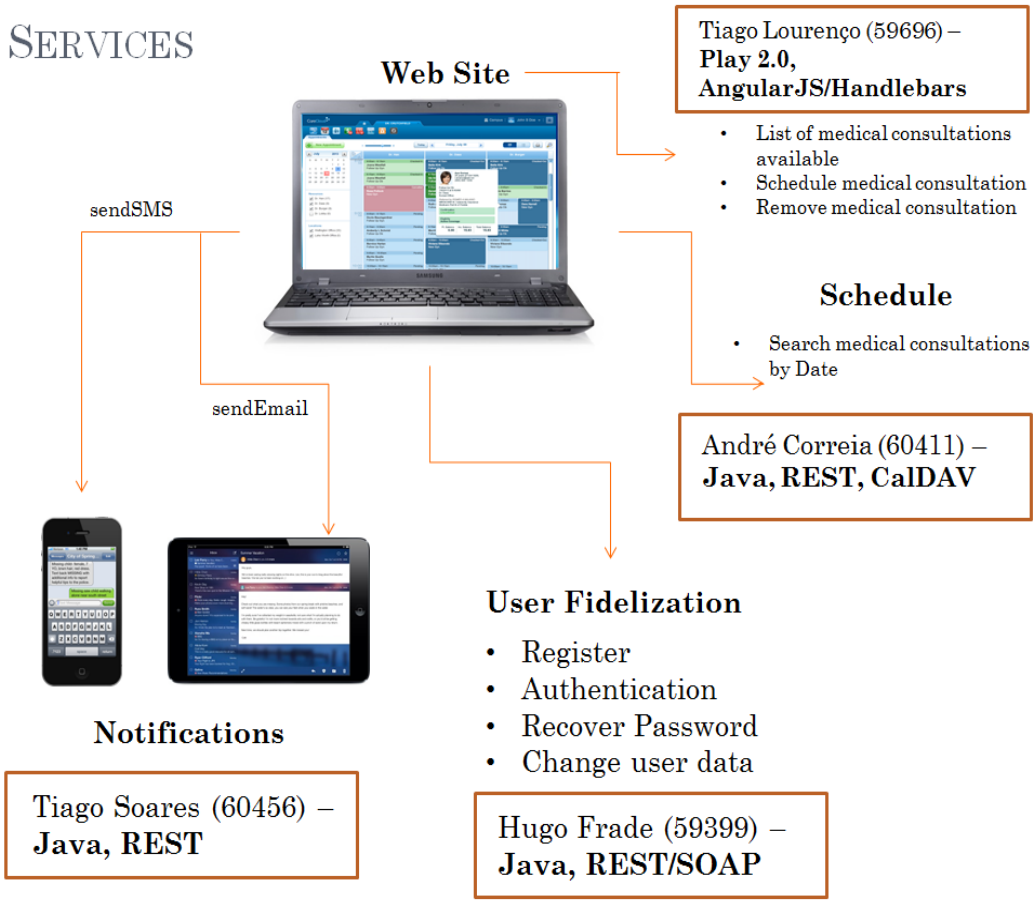

Figura 6: Serviços - Arquitectura

Web Site - Serviço de desenvolvimento das páginas Web para visualização de informação e de calendário por parte do paciente, para este efectuar o login via Facebook ou Gmail, para marcação de consultas, entre outras coisas. Não só é responsável pelo design/estética do conteúdo que é apresentado ao paciente e aos médicos mas também integra todos os restantes serviços, actuando assim como o serviço *Compositor*. Este serviço foi atribuído ao Tiago Lourenço e desenvolvido em Java usando para tal o Play Framework (IDE usado foi o IntelliJ).

Notificações - Serviço de notificação de eventos, neste caso o evento corresponde à marcação da consulta propriamente dita bem como avisar na véspera a mesma, assegurando assim o alerta eficaz ao paciente das consultas que marcou. Estas notifcações correspondem não só ao envio

de emails (através do email que é obtido após o login pelo Facebook/Gmail), mas também ao envio de SMS para o número do paciente e para este último foi proposta a criação de um SMS Gateway que irá ser abordado mais à frente com mais detalhe. Damos ainda a possibilidade de ser enviado um email com *attachment*, em que este é um simples ficheiro .ics que corresponde ao calendário. Este serviço foi atribuído ao Tiago Soares e desenvolvido em Java implementando *web services* do tipo RESTful (IDE usado foi o NetBeans 8.0).

Autenticação do Utilizador - Serviço responsável pela autenticação do paciente, mais concretamente o *login* por parte do utilizador na nossa plataforma via Facebook ou Gmail. É através deste serviço que é possível obtermos os dados do utilizador mal este se autentique por uma destas vias e com estes dados, nomeadamente o email, é possível depois enviar um email para este caso pretenda efectuar uma marcação de consulta médica. Este serviço foi atribuído ao Hugo Frade e desenvolvido em Java (IDE usado foi o NetBeans 8.0).

Schedule - Serviço de *scheduling* que permite a importação de calendários referentes ao dia/hora da marcação de consultas e desta forma é possível efectuar a gestão dos dias/horas disponíveis por parte dos médicos (com base num tipo de especialidade médica) para que, deste modo, o paciente possa visualizar a lista de dias/horas disponíveis para consulta na nossa plataforma. Para além disto, é com base neste serviço que juntamente com o das Notificações é possível enviar SMS de x em x tempo (dia/hora) para o paciente para relembrá-lo da consulta que se irá realizar no dia seguinte, e desta forma estamos a assegurar a segurança do mesmo. Este serviço foi atribuído ao André Correia e desenvolvido em Java implementando *web services* do tipo RESTful (IDE usado foi o NetBeans 7.4).

Nas seções seguintes, cada um destes serviços irá ser abordado com mais cuidado onde iremos explicar passo a passo o que foi feito para que cada um deles funcionasse corretamente para posterior integração com o Web Site.

Notificações

Para o serviço referente às Notificações, decidimos que seriam necessárias três funcionalidades de notificação na nossa plataforma: envio de SMS para o paciente, simples envio de um email com uma determinada mensagem e envio de emails com attachment (no nosso caso refere-se ao ficheiro .ics que corresponde ao calendário e que não é nada mais nada menos que uma imagem).

Depois de ter em conta estas funcionalidades, o passo seguinte consistiu em definir que tecnologia de Web Services iria usar para implementá-los e após uma breve reflexão foi decidido implementar os Web Services usando RESTful, esta decisão foi tomada apenas porque no ano passado já tinha sido desenvolvido algo do género para a cadeira de Engenharia de Software e trouxe um mais à vontade com esta decisão.

De seguida iremos abordar mais pormenorizadamente cada um dos Web Services que foram desenvolvidos ao longo deste projecto e que foram criados dentro de um projecto Maven Web Application no NetBeans 8.0:

Send SMS

```
@GET
@Produces("application/json")
@Path("smsmessaging/outbound")
Public Response sendSMS(@QueryParam("address") String number,
@QueryParam("message") String sms, @QueryParam("senderAddress") String senderAddress)
```

Figura 7: Cabeçalho do Web Service - Send SMS

Este Web Service permite o envio de uma SMS para o paciente no dia em que a consulta é efectivamente marcada e no dia anterior à própria consulta.

Para efeitos de teste de envio e recepção de SMS, foi criada uma conta no **RedOxygen** [6] e depois foi testado um código em Java que se baseia na API de envio de SMS do RedOxygen [7].

Porém para este serviço foi recomendada a configuração de um SMS Gateway usando para esse efeito o **Kannel SMS Gateway** (em Linux). Este é um SMS Gateway/WAP open source usado para fornecer SMS's e através deste foi possível o envio e recepção de mensagens. De seguida iremos explicar sucitamente os passos que foram necessários para que a configuração do Kannel fosse realizada com sucesso.

Passos para configuração do Kannel em Linux [1], [2] e [3]

1. **Instalar o kannel**
   - Executar o comando na consola: *sudo apt-get install kannel*
2. Depois deste comando ser executado com sucesso, o Kannel é instalado no /etc e dentro deste existe uma pasta Kannel que contém um ficheiro de configuração (kannel.conf) e o próximo passo consistiu em configurar o Kannel propriamente dito, colocando as seguintes linhas de configuração neste ficheiro.

3. **Configuração do core (bearebox core)**

   - group = core
     admin-port = 9109
     smsbox-port = 9110
     admin-password = bar
     status-password = foo
     log-file = ”/var/log/kannel/bearerbox.log”
     log-level = 0
     box-deny-ip = ”*.*.*.*”
     box-allow-ip = ”127.0.0.1”

4. **Configuração do SMSC (Short Message Service Center)**

   - group = core
     group = smsc
     smsc = at modemtype = auto
     device = /dev/ttyACM1 /*tty onde se encontra o Modem*/
     speed = 19200
     my-number = +351915100939 /*número do cartão que está inserido no Modem*/
     sms-center = +351911616161 /*número do centro de mensagens (Vodafone)*/
     validityperiod = 167

5. **Configuração do Modem (Vodafone ZTE)**

   - group = modems
     broken = true
     id = ZTE WCDMA MSM K3806-Z.0.03
     name = ”ZTE WCDMA MSM K3806-Z.0.03”
     detect-string = ”K3806-Z.0.03”
     init-string = ”AT Q0 V1 E1 S0=0 C1 D2 +FCLASS=0”
     speed = 115200
     enable-mms = true
     message-storage = sm

6. **Configuração do smsBox**

   - group = smsbox
     bearerbox-host = localhost
     sendsms-port = 9111

global-sender = +351915100939 /*número do cartão que está inserido no Modem*/
sendsms-chars = "0123456789 +-"
log-file = "/var/log/kannel/smsbox.log"
log-level = 0
access-log = "/var/log/kannel/access.log"

7. **Configuração do sendsms-user**

   - group = sendsms-user
     username = kannel
     password = kannel
     concatenation = true
     max-messages = 3
     max-messages = 10

8. **Configuração do sms-service**

   - group = sms-service
     keyword = default
     get-url="http://192.168.8.217/cgi-bin/receivesms?receiver=receiverNumber
     &text=message&send=senderNumber"
     accept-x-kannel-headers = true
     max-messages = 3
     concatenation = true
     catch-all = true
     catch-all = true

9. **Depois de configurar o ficheiro kannel.conf, o passo seguinte consistiu em iniciar o servidor executando para este efeito os seguintes comandos:**

   - *sudo bearerbox -v 0 /etc/kannel/kannel.conf*
   - *sudo smsbox -v 0 /etc/kannel/kannel.conf*

Para verificar qual era o tty associado ao modem, foi executado o seguinte comando: *dmesg — grep tty* onde foi verificado que o modem estava associado ao ttyACM1 e foi adicionado no kannel.conf para que o device pudesse ser correctamente reconhecido quando o servidor era iniciado.

De seguida foi testado o envio de uma SMS de teste dum telemóvel para o cartão e verificar se o servidor interpretou bem a mensagem, e como recebeu essa mensagem então a recepção de SMS está a funcionar correctamente.

O mesmo foi feito mas agora para testar o envio de SMS e para tal foi feito o uso do seguinte URL:

http://localhost:9111/cgibin/sendsms?username=kannel&password=kannel&to=%2B351911102702&text=ola&from=%2B351915100939

de modo a testar se era enviada uma SMS do cartão que estava no device para esse telemóvel e surge uma mensagem no Web Browser : "0: Accepted for delivery" pelo que o envio de SMS está a funcionar correctamente (foi verificado na mesma nesse telemóvel se tinha recebido a mensagem, por via das dúvidas).

Depois de o Kannel estar bem configurado o passo seguinte foi construir um Web Service que incluía o URL acima testado.

De referir que este Web Service respeita a One API do GSMA SMS Gateway [4] pelo que cumpre com os três parâmetros de input obrigatórios (nº telemóvel de destino, conteúdo da mensagem e nº telemóvel de origem). Além disso caso a mensagem contenha mais do que 160 caracteres, então esta é dividida em duas pois uma SMS suporta no máximo 160 caracteres. Segundo a GSMA SMS RESTful API , devem ser cumpridas (mais pormenorizadamente) as seguintes regras:

- Sending SMS URL: http://example.com/{apiversion}/smsmessaging/outbound/{senderAddress}/requests
- Parâmetros obrigatórios: **address** (is the URL-escaped end user ID; in this case their MSISDN including the 'tel:' protocol identifier and the country code preceded by '+'. i.e., tel:+16309700001), **Message** ((string) must be URL-escaped as per RFC 1738. Messages over 160 characters may end up being sent as two or more messages by the operator.) e **senderAddress** (is the address to whom a responding SMS may be sent).

Adicionalmente foi usado o **Cron** [8] para correr um script de acordo com uma determinada periodicidade (dia/hora/minuto/segundo) e desta forma podemos enviar a SMS dinamicamente, ou seja, não só damos a possibilidade de ser enviada quando o utilizador efectua a consulta mas também passa a receber a mensagem na véspera da mesma. Mas para isto também foi necessário executar um comando neste srcipt que é o seguinte: *curl -i -H*

"Accept: application/jsonURL do envio da SMS" (através do **cURL** [9]) e desta forma o URL do envio de SMS é passado neste comando e que se encontra no script que é corrido com base numa periodicidade, como já foi referido.

Este WS retorna um Response que indica se a mensagem foi enviada com sucesso.

URL do WS "Send SMS": `http://192.168.8.217:9106/WCS_Notifications-1.0-SNAPSHOT/webresources/Notifications/smsmessaging/outbound?address=911102702&message=ola&senderAddress=915100939`

Send Email

```
@GET
@Produces("application/json")
@Path("mail/sendEmail")
public Response sendEmail(@QueryParam("Email") String emailDest,
@QueryParam("Message") String content, @QueryParam("Subject") String subject)
```

Figura 8: Cabeçalho do Web Service - Send Email

Este WS é responsável pelo envio de um email para o paciente no dia em que a consulta é marcada. Os passos que consistiram na implementação deste WS foram os seguintes [5]:

1. Inicialmente fiz uso do servidor SMTP do Gmail e associei uma conta Gmail para teste:

   - private final static String host = "smtp.gmail.com";
     private final static String user = "tiagomsoares92@gmail.com";
     private final static String pwd = "**********";
     private final static String port = "465";
     private final static String from = "tiagomsoares92@gmail.com ";
     private final static String SMTP_TLS_SSL = "true";
     private final static String auth = "true";

private final static String debug = ”true”;
private final static String SOCKET_FACTORY = ”javax.net.ssl.SSLSocketFactory”;

2. Depois foram definidas as propriedades da mensagem com base na classe “Properties” onde foram adicionados os parâmetros acima representados.

3. Foi estabelecida uma sessão através do uso da classe “Session” onde foi retornada uma Password Authentication com base no user e pwd acima instanciados.

4. O passo seguinte consistiu na construção da mensagem e para isso fizémos uso da classe “MimeMessage” através da qual foi possível definir o corpo da mensagem, o endereço de email de destino e o assunto da mensagem.

5. A classe “Transport” foi usada para entregar correctamente a mensagem ao destinatário.

Este WS recebe como parâmetros o endereço de email de destino, o assunto da mensagem e ainda o conteúdo da mensagem e retorna um Response que indica se a mensagem de email foi enviada com sucesso.

Mas para que o email podesse ser enviado dentro da rede da UA (eduroam) então o host teve de ser alterado para “smtp.office365.com”, o user passava a ser uma das nossas contas na UA e a porta também seria diferente (587). Com estas alterações foi possível que o email também podesse ser enviado neste tipo de ligação de rede.

URL do WS “Send Email”: http://192.168.8.217:9106/WCS_Notifications-1.0-SNAPSHOT/webresources/Notifications/mail/sendEmail?Email=tiagomsoares@ua.pt&Message=ola&Subject=assunto

Send Email with Attachment

```
@GET

@Produces("application/json")

@Path("mail/sendEmail/Attachment")

public Response sendMailAttach(@QueryParam("Mail") String mail, @QueryParam("Content")
String content,@QueryParam("Subject") String subject, @QueryParam("Attachment") String
attachment, @QueryParam("Filename") String filename)
```

Figura 9: Cabeçalho do Web Service - Send Email with Attachment

Este WS é em tudo semelhante ao anterior mas com a possibilidade de ser anexado um ficheiro quando o email é enviado para o destinatário. A implementação deste é igual ao WS anterior mas com a possibilidade de dividir a mensagem em partes [5] e para isso fizémos uso da classe "BodyPart" e com a classe "Multipart" podemos criar uma multipar message cuja segunda parte corresponde ao ficheiro anexado.

Para o anexo de ficheiros foi usada a classe "DataSource" onde definimos qual o ficheiro a ser anexado (através da função "SetFileName") e que depois é adicionado ao corpo da mensagem através da função "AddBodyPart".

Para enviar a mensagem completa com ambas as partes então foi usada a função "setContent" que recebe como parâmetro a multipart message.

Este WS recebe como parâmetros o endereço de email de destino, o conteúdo da mensagem, o assunto da mensagem, o ficheiro em anexo e o nome do ficheiro que é anexado à mensagem e retorna um Response que indica se a mensagem de email com anexo foi enviado com sucesso.

URL do WS "Send Email with Attachment": `http://192.168.8.217:9106/WCS_Notifications-1.0-SNAPSHOT/webresources/Notifications/mail/sendEmail/Attachment?Mail=tiagomsoares@ua.pt&Content=ola&Subject=assunto&Attachment=/home/wcs/ola.txt&Filename=file`

Depois dos WS serem criados, o passo seguinte consistiu em fazer o *deploy* destes no GlassFish que se encontra no nosso servidor. Para tal, basta efectuar o "Clean and Build" do projecto Maven e colocar o *.war* gerado na consola de administração do GlassFish (192.168.8.217:9105).

Também foi necessário incluir os .jar 's necessários para que a esta aplicação fosse bem "deployed" e neste caso houve a necessidade de inserir o **javax.mail.jar** na pasta /etc/init.d/glassfish4/glassfish/domains/domain1/libs/applibs e assim o deploy deste projecto (WCS_Notifications) foi feito com sucesso e a partir daqui foi só testar os URLs acima referidos e verificar o envio de emails e de SMS.

Schedule

Para este serviço foi criada uma máquina virtual (192.168.215.201), com o sistema operativo Ubuntu 10.04, onde foi configurado um servidor CalDAV e um servidor Glassfish. O código foi desenvolvido na linguagem Java e está disponível no projecto NetBeans "calendarservice".

0.1 Servidor CalDAV

A implementacão do protocolo CalDAV escolhida foi DAViCal[1] (v1.0.2) e a sua instalação foi feita com base no guia[2] disponível na página oficial do projecto.
A página de administraçao pode ser acedida através do url `http://192.168.8.217:9102` (user:admin; password:NMppaux) e permite gerir os utilizadores e as suas relações.
Para um dado utilizador registado na base de dados do servidor DAViCal é possível aceder aos seu calendário utilizando um cliente CalDAV (e.g. Lightning, Kontact, Evolution) através do url:
http://192.168.8.217:9102/caldav.php/[USERNAME]/calendar

0.2 Web Services desenvolvidos

De forma a facilitar a comunicação entre os diferentes serviços do sistema foram criados web services (REST) que permitem gerir os utilizadores e os eventos na base de dados DAViCal.

[1] [DAViCal - `http://www.davical.org/`]
[2] [Guia de instalação - `http://davical.org/installation.php`]

0.2.1 Gestão de utilizadores

Para a criação de utilizadores foi necessário utilizar a ferramenta "davical command line utility"[3] que permite interagir com a base de dados DAViCal através da linha de comandos.

Criar utilizador
Adiciona um utilizador à base de dados DAViCal.

```
@GET
@Produces("application/json")
@Path("createUser")
public Response createUser(@QueryParam("uid") String uid, @QueryParam("email") String email)
```

Valor de retorno:

- "true", utilizador criado com sucesso;
- "false", caso ocorra erro;

0.2.2 Gestão de eventos

A gestão de eventos nos calendários de cada utilizador é feita utilizando a biblioteca Java "ical4j"[4]. Esta API permite aceder aos calendários armazenados no servidor DAViCal e oferece suporte para parsing e edição dos mesmos.

Criar evento
Adiciona um evento ao calendário de um utilizador.

```
@GET
@Produces("application/json")
@Path("createEvent")
public Response createEvent(@QueryParam("uid") String uid,
@QueryParam("dayStart") int dayStart,
@QueryParam("monthStart") int monthStart,
@QueryParam("yearStart") int yearStart,
@QueryParam("hStart") int hStart,
@QueryParam("mStart") int mStart,
@QueryParam("sStart") int sStart,
@QueryParam("dayEnd") int dayEnd,
```

[3][Davical command line utility - http://sourceforge.net/projects/davical-cmdlnut/]

[4][ical4j - https://github.com/ical4j/ical4j/wiki]

@QueryParam(”monthEnd”) int monthEnd,
@QueryParam(”yearEnd”) int yearEnd,
@QueryParam(”hEnd”) int hEnd,
@QueryParam(”mEnd”) int mEnd,
@QueryParam(”sEnd”) int sEnd,
@QueryParam(”eid”) String eid,
@QueryParam(”subject”) String subject)
Valor de retorno:

- id do evento, caso sucesso;
- ”null”, caso ocorra erro;

Remover evento
Remove um evento do calendário de um utilizador.

@GET
@Produces(”application/json”)
@Path(”removeEvent”)
public Response removeEvent(@QueryParam(”uid”) String uid, @QueryParam(”eid”) String eid)
Valor de retorno:

- ”true”, caso sucesso;
- ”false”, caso ocorra erro;

Subscrever evento
Adiciona um email à lista de participantes do evento.

@GET
@Produces(”application/json”)
@Path(”subscribeEvent”)
public Response subscribeEvent(@QueryParam(”uid”) String uid, @QueryParam(”eid”) String eid, @QueryParam(”email”) String email)
Valor de retorno:

- ”true”, caso sucesso;
- ”false”, caso ocorra erro;

Remover subscrição
Anula a subscrição do evento.

@GET
@Produces(”application/json”)
@Path(”removeSubscription”)
public Response removeSubscription(@QueryParam(”uid”) String uid, @Query-Param(”eid”) String eid)
Valor de retorno:

- ”true”, caso sucesso;
- ”false”, caso ocorra erro;

Obter calendário

@GET
@Produces(”application/json”)
@Path(”getCalendar”)
public Response getCalendar(@QueryParam(”uid”) String uid)
Valor de retorno:

- conteúdo do calendario, caso sucesso;
- ”null”, caso ocorra erro;

Reminder
Serviço utilizado em conjunto com a ferramente ”cron”(Unix, job scheduler). Todos os dias a uma dada hora, é executado no servidor, um script que chama este web service.
A lista de pacientes é adquirida através do web service http://192.168.215.201:8080/teste1-1.0-SNAPSHOT/webresources/generic/listPatients, e para cada paciente, é verificado no seu calendario se existem consultas agendadas nas próximas 24 horas. É enviado um email ,com o respectivo calendário, a cada utilizador com eventos.

@GET
@Produces(”application/json”)
@Path(”reminder”)
public String reminder()
Valor de retorno:

- lista de utilizadores ao qual enviado um email com o respectivo calendario;

Autenticação do Utilizador

Relativamente à autenticação de utilizadores, no âmbito deste trabalho desenvolvemos modos de autenticação recorrendo às APIs do Facebook e Google. Estas APIs usam o processo `OAuth 2`. Este módulo está implementado no projecto *WCSAuthenticationModule*.

Para tal recorremos a uma base de dados composta por duas tabelas, sendo que uma contém os dados pessoais sobre o utilizador *We Care Services* e a outra contém as associações entre as contas *We Care Services* e as contas *sociais*.

0.3 Configuração Facebook

Para configurarmos o login com o *Facebook* necessitámos de registar a aplicação em `https://developers.facebook.com`. O url de base do website corresponde a `http://192.168.8.217:9106/WCSAuthenticationModule`, enquanto que o url para o qual o browser é redireccionado quando o utilizador efectua o login com sucesso e fornece o seu consentimento para a nossa aplicação aceder aos seus dados pessoais corresponde a `http://192.168.8.217:9106/WCSAuthenticationModule/CallbackFacebook`. Neste url a nossa aplicação obtém o *access token*, fornecendo o *authentication code* proveniente da API do *Facebook*. Após receber o *access token* a aplicação redirecciona a aplicação para a home page, sendo todo este processo transparente ao utilizador.

0.4 Configuração Google

Para configurarmos o login com o *Google* necessitámos de registar a aplicação em `console.developers.google.com`; para tal precisámos de activar a *Google+ API* (para aceder à informação privada do utilizador) e ainda o processo de autenticação *OAuth 2.0*. Visto que o Google não permite que os

url's de base correspondam a endereços IP privados, necessitámos de criar um domínio público `http://authen.ddns.net` (em `noip.com`) e associámo-lo ao nosso endereço privado `192.168.8.217:9106`. Ao contrário da autenticação utilizando o *Facebook*, no caso do *Google* obtemos o *access token* no url base do módulo de autenticação.

0.5 Flow de execução

Nesta secção iremos descrever o funcionamento do módulo de autenticação *We Care Services* propriamente dito. Começamos por apresentar o diagrama de sequência:

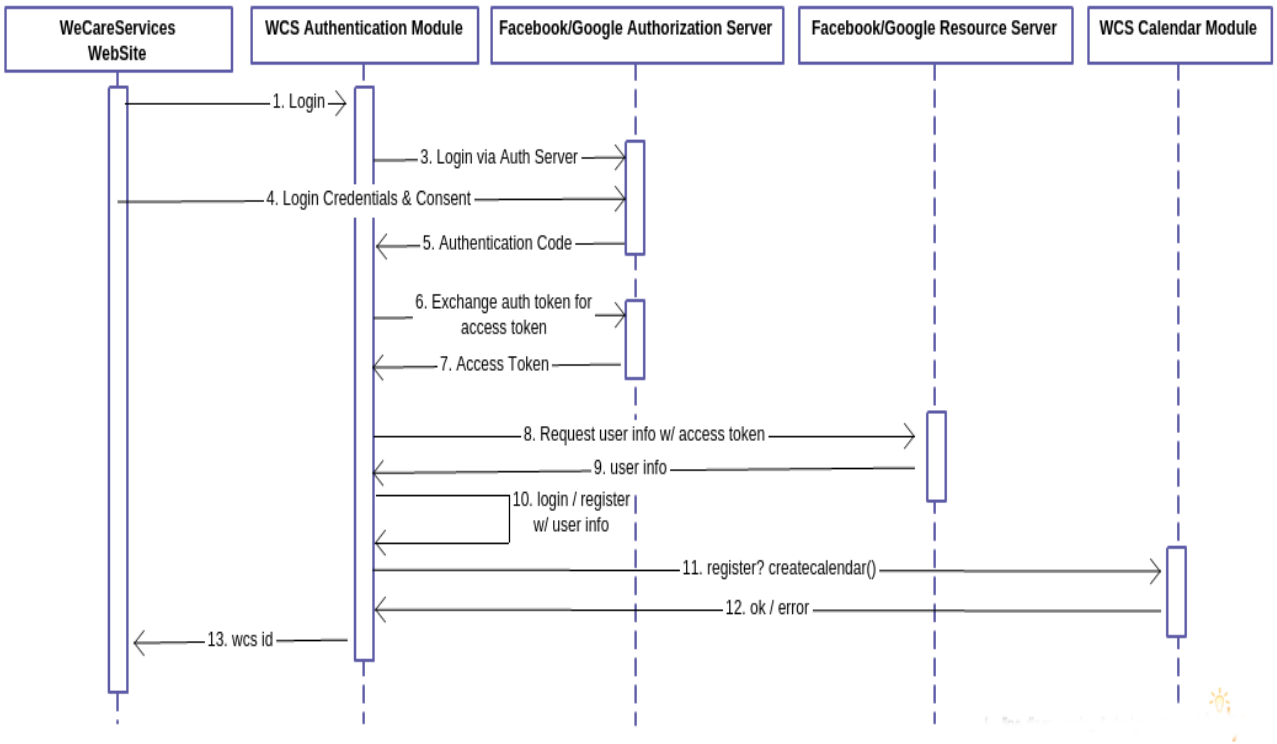

Figura 10: Authentication Process

Podemos constatar que as primeiras interacções correspondem ao processo *OAuth 2.0*, que não iremos explicar aqui em detalhe. Todas as interacções que existem com as APIs do Google e Facebook são suportadas pela classe

`AuthHelper`. Esta classe é responsável por construir os *url* de login, solicitar os *access token* e aceder à informação do perfil dos utilizadores. A partir do momento que obtemos a informação do utilizador é quando o nosso módulo de autenticação começa a realizar trabalho útil. A informação a que acedemos do perfil do utilizador corresponde ao primeiro e último nome, email e data de nascimento.

Relativamente ao flow de execução do nosso módulo de autenticação, implementado na classe `DBConnector`, detectamos qual o tipo de operação a ser realizada, de entre as seguintes:

- `login` quando o utilizador não se encontra autenticado e está a realizar login.
- `associação de contas` quando o utilizador se encontra autenticado, e à conta com que foi autenticado, quer associar outra conta *social*.

Relativamente ao primeiro caso, ao efectuar *login*, primeiramente verificamos se a conta que o utilizador está a utilizar para se autenticar já foi utilizada alguma vez no nosso sistema. Caso tenha sido, já existe uma instância dela no nosso sistema e apenas nos limitamos a actualizar os dados pessoais do utilizador e a retornar o identificador interno no nosso sistema *We Care Services*. A actualização dos dados pessoais do utilizador é efectuada quando detectamos que a informação proveniente da rede social é inconsistente com a que temos armazenada.

Caso detectemos que a conta que o utilizador está a utilizar para se autenticar nunca foi utilizada no nosso sistema criamos o registo da mesma e atribuimos-lhe um identificador interno, que no nosso caso corresponde a um número positivo incremental. Quando estamos a registar um utilizador, é necessário também criar calendário e associá-lo à respectiva conta *We Care Services*. Esta criação é efectuada efectuando uma chamada via *REST* ao módulo de Calendarização.

Para efectuar *login*, é necessário recorrer ao url de base do módulo de autenticação, utilizando o parâmetro `authmethod` e atribuindo-lhe o valor de `Facebook` ou `Google`, dependendo do modo como o utilizador se quiser autenticar. Posteriormente é retornado o identificador único do utilizador no sistema *We Care Services* utilizando o parâmetro `id_user` no url de resposta.

Relativamente ao segunto caso, ao efectuar associação de uma conta *social* a uma *We Care Services* já existente, podemos incorrer em duas situações distintas: associar uma conta *social* que ainda não tenha sido utilizada no nosso sistema ou uma que já tenha sido utilizada. Caso nunca tenha sido utilizada, limitamo-nos a associar o identificador interno da conta *social* (no nosso caso definimos este identificador como o *email*) ao nosso identificador interno e a actualizar eventuais dados pessoais que não tenham sido preenchidos anteriormente.

Quando queremos associar uma conta *social* que já está atribuída a outra conta *We Care Services*, fazemos a migração de informação caso seja necessário. Para tal, verificamos se tanto o número de telemóvel como a data de nascimento se encontram disponíveis na conta *nova*, ou seja, aquela à qual se pretende associar uma nova conta de uma rede social. Caso não estejam e na *antiga* estejam, actualizamos a informação. Neste caso pode ainda ocorrer a situação de uma conta *We Care Services* ficar sem nenhuma conta *social* associada a si; tendo então que ser apagada de forma a manter a informação na base de dados consistente.

Para efectuar *associação de contas*, é necessário recorrer ao url de base do módulo de autenticação utilizando os parâmetros `authmethod` e `id_user_wcs`, ao qual deve ser atribuído o valor do identificador único do utilizador. Posteriormente é retornado o identificador único do utilizador no sistema *We Care Services* utilizando o parâmetro `id_user` no url de resposta.

Web Site

O website é o ponto fulcral que permite mostrar aos diversos utilizadores o funcionamento em termos visuais de todos os serviços.

Para a construção do mesmo foi utilizada a framework *Play framework 2.0* que adota a arquitetura MVC. Em conjunto com esta framework foram utilizadas outras tools como o bootstrap, jQuery e o plugin fullCalendar fornecendo todas as características do sistema aos utilizadores.

O nosso website apresenta duas secções aos utilizadores, ou seja, uma secção pública e uma secção privada. A parte pública é utilizada para integrar o serviço de autenticação, permitindo a qualquer utilizador que esteja ou se queira registar no sistema tenha acesso ao mesmo. A partir do momento que um utilizador é autenticado tem acesso à parte interna do sistema. Na parte interna dividimos os vários utilizadores em 3 utilizadores diferentes, com diferentes permissões e funcionalidades acessíveis. Os tipos de utilizadores são o Paciente, o Doutor e o Administrador do sistema.

Vamos começar por descrever as funcionalidades disponíveis ao paciente. Este após ter realizado o processo de autenticação vai estar perante uma página onde lhe são apresentadas as diferentes especialidades existentes no hospital e os respetivos doutores associados. Após selecionar um doutor de uma respetiva especialidade, vai ser apresentado o calendário do doutor ao paciente, onde este pode marcar uma consulta, ou desmarcar a mesma. Tendo a possibilidade de mudar o dia ou hora da consulta ou mesmo cancelar a mesma. Após a marcação de uma consulta, o sistema automaticamente envia uma sms para o telemóvel do paciente a alertar que a consulta está registada, alertando o dia e hora da mesma. Algo semelhante ocorre com o envio de um email, onde o paciente recebe a mesma informação, contendo no entanto um ficheiro ics permitindo ao paciente anexar esse ficheiro no seu calendário virtual (ex: Outlook).

Outra funcionalidade de que o paciente dispõe e é comum a todos os tipos de utilizadores é o acesso aos dados pessoais, onde os mesmos podem alterar dados pessoais, consultar o seu histórico em termos de consultas médicas e têm também a possibilidade de consultar o seu calendário no sistema, que contém todas as suas consultas no hospital.

Passemos agora à descrição das funcionalidades acessíveis ao doutor. Um doutor ao aceder ao sistema é-lhe apresentada uma lista com os seus vários pacientes. Ao selecionar um dos pacientes o doutor pode visualizar o histórico de consultas do paciente no que diz respeito à sua especialidade como de outras especialidades que o paciente tenha frequentado no hospital. O doutor pode ainda consultar o seu calendário onde pode visualizar os detalhes das consultas e pode ainda, se necessário, cancelar uma determinada consulta. Ao realizar esta operação o que o sistema vai fazer é instantaneamente enviar uma sms ao paciente para o alertar de que a sua consulta foi cancelada. Permitindo ao mesmo fazer uma remarcação de uma consulta.

Por fim vamos descrever as funcionalidades disponíveis ao administrador. Este além de usufruir de todas as funcionalidades fornecidas ao doutor usufrui também de outras funcionalidades. A funcionalidade mais notória para o administrador é ter uma secção de "dashboard" onde lhe é possível visualizar dados estatísticos das consultas realizadas no hospital. Permitindo ao mesmo estar contextualizado acerca dos problemas mais tratados e trabalhados no hospital. Além desta funcionalidade o administrador tem a possibilidade de enviar sms's, emails simples e emails com anexos aos vários pacientes, individualmente. Podendo também, ao selecionar um paciente, visualizar o seu calendário. Por último o administrador pode ainda consultar a lista de doutores e visualizar o calendário respetivo de cada um.

Link para a nossa aplicação: 192.168.8.217:9104

Link para a nossa DEMO no Sapo Campus Videos: http://videos.ua.sapo.pt/p75PgL2HK00smNM6fMyY

Link para o nosso código no GitHub: https://github.com/TiagoVilaFlor/ES_WCS

Configurações no Servidor

Nas secções anteriores já referimos algumas das configurações que foram realizadas no nosso server, contudo vamos descrever com mais detalhe o que foi necessário configurar no nosso servidor de modo a que a nossa aplicação fosse bem *deployed*.

Inicialmente foi instalado o Ubuntu como sistema operativo e com o comando *sudo apt-get install* foram instaladas todas as dependências necessárias. Todas as configurações usadas para instalar o Kannel no servidor já foram explicadas na secção "Notificações". Foi instalado o GlassFish versão 4.0 como sendo o nosso Application Server de modo a que os nossos Web Services fossem bem deployed no servidor.

Foi também utilizada uma base de dados *mysql*, sendo que o script utilizado para a sua criação corresponde ao ficheiro `DatabaseWCS.sql` e está no projecto `WCSAuthenticationModule`.

Para fazer o deploy da nossa aplicação no servidor, foram usados os seguintes comandos: *play start* durante o desenvolvimento do projeto e *play run* quando já tinhamos a versão final do projeto pronta. De referir ainda que foi necessário incluir bibliotecas externas usadas para construir os Web Services na pasta applibs do GlassFish no nosso server, sem isto não era possível efectuar o deploy de forma correcta. Para correr o GlassFish foi usado o seguinte comando: *./asadmin start-domain domain1*.

Para que as sms pudessem ser entregues automaticamente aos pacientes, era necessário que o Kannel estivesse sempre a correr no nosso servidor e deste modo os comandos *sudo bearerbox -v 0 /etc/kannel/kannel.conf* e *sudo smsbox -v 0 /etc/kannel/kannel.conf* têm de estar a correr sempre na nossa máquina.

Trabalho Futuro

Para trabalho futuro gostaríamos que a nossa aplicação fosse ainda mais longe que a Web que foi implementada ao longo deste semestre, dando a possibilidade ao utilizador de efectuar a marcação de consultas através do seu telemóvel. Para tal teríamos de criar uma réplica da nossa aplicação em ambiente mobile. Infelizmente não tivémos tempo disponível para isso (e não fazia parte dos requisitos), mas seria algo interessante a ser desenvolvido.

Outras funcionalidades que gostaríamos de ver integradas com o nosso sistema consistiam na possibilidade de fornecermos aos pacientes a localização geográfica da farmácia mais próxima deste ou efetuar o respectivo pagamento da consulta via Paypal.

Desta forma, sem mais comentários a fazer, achámos que conseguímos realizar o projeto com sucesso, cumprindo com os objetivos iniciais e ainda outras funcionalidades que fomos desenvolvendo no desenrolar deste trabalho prático.

Referências

[1] *Sending SMS with SMPP, Kannel and Java*. URL: http://mobiforge.com/design-development/sending-sms-with-smpp-kannel-and-java.

[2] *Sample Kannel Configuration*. URL: https://gist.github.com/tobiasmcnulty/1710278.

[3] *Install and Configure Kannel (SMS Gateway) for Sending SMS*. URL: http://mahtabrasheed.wordpress.com/2013/01/10/how-to-installconfigure-kannelsms-gateway-for-sending-sms-from-your-computer/.

[4] *GSMA One API – SMS RESTful API*. URL: http://www.gsma.com/oneapi/sms-restful-api/.

[5] *Envio de Emails (e com Attachment) em Java (JavaMail API)*. URL: http://www.tutorialspoint.com/java/java_sending_email.htm.

[6] *RedOxygen – Global Bulk SMS Gateway*. URL: http://redoxygen.com/.

[7] *Sending SMS Text Message From Java using API of RedOxygen*. URL: http://www.luv2code.com/2012/07/03/sending-sms-text-message-from-java/.

[8] *How To: Automate Server Scripts With Cron*. URL: http://www.liquidweb.com/kb/how-to-automate-server-scripts-with-cron/.

[9] *REST-esting with cURL*. URL: http://blogs.plexibus.com/2009/01/15/rest-esting-with-curl/.