Faculdade de Engenharia da Universidade do Porto

OpenSocial: Desenvolvimento de um plugin para a Framework Symfony e de uma aplicação para o Palco Principal

Daniel Carlos Araújo Botelho

VERSÃO DEFINITIVA

Relatório de Projecto realizado no âmbito do Mestrado Integrado em Engenharia Informática

Orientador: João Canas Ferreira

Julho 2008

OpenSocial:
Desenvolvimento de um plugin para a Framework Symfony e de uma aplicação para o Palco Principal

Daniel Carlos Araújo Botelho

Relatório de Projecto
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Doutor Jorge Alves da Silva, Prof. Auxiliar da FEUP

_____

Arguente: Doutor António José Borba Ramires Fernandes, Prof. Auxiliar da Universidade do Minho

Vogal: Doutor João Paulo de Castro Canas Ferreira, Prof. Auxiliar da FEUP

16 de Julho de 2008

Resumo

Este documento enquadra-se no contexto da *Web 2.0*, onde os sites sociais são os seus principais impulsionadores.

Dado o crescente uso de redes sociais, a *Google* lançou o *OpenSocial* que visa permitir que sejam desenvolvidas aplicações que interajam com qualquer rede social, desde que estas suportem a sua *API*.

Dado esta ser abstracta, cada rede que a implemente, tem a possibilidade de adaptá-la às suas características e necessidades específicas.

Existem já várias plataformas sociais que suportam o *OpenSocial*, tais como *Hi5*, *Orkut*, *MySpace*, *Netlog*, *Ning*, *Imeem* e muitas outras que estão ainda em vias de o suportar.

Neste projecto é desenvolvida uma aplicação *OpenSocial* para o site de música Palco Principal, com a particularidade de esta ser implementada usando a *framework symfony*.

A *framework symfony* está optimizada para o desenvolvimento de *Websites*, não suportando por omissão o desenvolvimento de aplicações *OpenSocial*.

Com vista a resolver esta limitação, foi desenvolvido, no âmbito deste projecto, um *plugin* para a *framework symfony* que permite o desenvolvimento de aplicações *OpenSocial*, tirando proveito das funcionalidades que esta plataforma oferece.

Abstract

This document fits in the context of Web 2.0, where the social websites are their main stimulator.

Considering the increasing use of Social Networks, Google has introduced the OpenSocial API which intends to allow applications to interact with any Social Network, as long as they support its API.

Considering that the API is abstract, each container has the possibility to adapt it to its characteristics and specific needs.

There are already several social platforms which support OpenSocial, such as Hi5, Orkut, MySpace, Netlog, Ning, Imeem and many others will soon support it too.

In this project, an OpenSocial application for the music portal "*Palco Principal*" is developed, with the particularity of being implemented using the symfony framework.

The symfony framework is optimized for developing *Websites*, not supporting by default the development of OpenSocial applications.

In order to solve this limitation, a plugin for the symfony framework has been created in this project, allowing the development of OpenSocial applications, taking advantage of the features offered by this framework.

Agradecimentos

Gostaria de agradecer à minha família e amigos, pela disponibilidade e apoio durante todo período de realização deste projecto.

Gostaria também de agradecer aos colegas do Palco Principal por me terem fornecido os meios para a realização deste projecto, e ao Professor João Canas Ferreira pela sua orientação e disponibilidade.

Conteúdos

Índice de Tabelas

Índice de Figuras

Índice de listagens

Capítulo 1: Introdução

1.1 Enquadramento

Com a massificação da adopção do paradigma da *Web 2.0* na *Internet*, um novo "mundo" de oportunidades e novos modelos de negócio prosperam.

Podemos dizer que estamos na era da *Web social*, onde as pessoas podem interagir umas com as outras, tirando partido das vantagens que daí advêm.

Os casos mais evidentes do sucesso deste novo paradigma, são as redes sociais ou *sites* sociais.

Na figura abaixo podemos ver o "*Top Web 2.0 Sites*" gerado a 2008/06/27 pelo site *Movers 2.0* (http://movers20.esnips.com/) onde as 16 primeiras posições dos sites mais vistos (*ranking* fornecido pelo *site Alexa Rank*), estão entregues a 6 sites de redes sociais.

Rank	Site		Alexa Rank
1.	YouTube	YouTube Graphs | News | More...	3
2.	facebook	Facebook Graphs | News | More...	6
3.	myspace.com	MySpace Graphs | News | More...	7
4.	WIKIPEDIA	Wikipedia Graphs | News | More...	8
5.	orkut	Orkut Graphs | News | More...	11
6.	hi5	hi5 Graphs | News | More...	16

Figura 1.1: Top Web 2.0 Sites, disponível pelo site Movers 2.0 em 2008/06/27

Dado o crescente uso de *sites* sociais, novas áreas de negócio têm emergido, e uma delas consiste em sites sociais permitirem que terceiros desenvolvam aplicações que podem interagir com os utilizadores do seu site, e, aceder as funcionalidades que estes disponibilizam.

O facto de cada site social disponibilizar a sua própria API para desenvolver aplicações exclusivamente para a sua rede social, levava a que os programadorese tivessem de aprender várias APIs por forma a tornarem as suas aplicações distribuídas no maior número de sites sociais possíveis.

Em vista a resolver este problema, a Google lançou o OpenSocial que permite unificar o desenvolvimento de aplicações para sites sociais.

O *OpenSocial* é constituído por um conjunto de APIs desenhadas para desenvolver aplicações sociais na Web. O facto de existir um conjunto comum de APIs, significa que o programador apenas necessita de as aprender uma vez, para poder usa-las no desenvolvimento de aplicações sociais para múltiplos sites.

Qualquer site é livre de implementar as APIs do OpenSocial de acordo com as suas necessidades particulares, podendo assim alojar aplicações sociais desenvolvidas para estas APIs. Um site que implemente as APIs do OpenSocial é designado por "c*ontainer*" OpenSocial e actualmente existem já vários *containers*, sendo os mais importantes o MySpace, Hi5, Orkut, LinkedIn e Xing.

O Palco Principal é um portal de música, que segue o modelo Web 2.0. Actualmente possui uma das maiores comunidades de música de expressão Portuguesa na Web, agrupando ouvintes e artistas (mais de 6800 bandas, mais de 22000 utilizadores registados, mais de 15000 músicas disponíveis para audição e download e mais de 30000 visitantes regulares).

O principal objectivo do Palco Principal é fornecer a plataforma para que músicos e ouvintes interajam,

de forma a criar valor para ambos neste processo.

1.2 Motivação

Uma das grandes fontes de rendimento do Palco Principal como empresa, consiste na sua capacidade de fazer chegar publicidade aos seus visitantes.

Assim sendo, o OpenSocial é visto como uma oportunidade de expansão para a Palco Principal, permitindo que o seu site seja integrado em várias redes sociais permitindo uma maior divulgação, aumentando assim a sua fonte de rendimento.

Do ponto de vista do Palco Principal como portal de música, possuir uma aplicação OpenSocial traz grandes vantagens, pois vai permitir que os seus utilizadores (especialmente bandas) possam divulgar os seus projectos noutros ambientes usando as suas contas já existentes neste portal.

A Palco Principal como empresa que aposta no desenvolvimento tecnológico, especialmente no movimento *Open Source,* pretende também acompanhar e contribuir de forma activa para que melhor tecnologia seja desenvolvida nestas condições.

Uma vez que o Palco Principal assenta em *software* livre, nomeadamente na *framework symfony,* é, portanto, do interesse da Palco Principal que seja desenvolvido um *plugin* que expanda esta plataforma adaptando-a para o desenvolvimento de aplicações OpenSocial.

Este *plugin* será distribuído usando a licença MIT por forma a ser usado livremente por qualquer projecto que assente na *framework symfony*.

1.3 Objectivo deste projecto

O objectivo deste projecto consiste em realizar um estudo da *API* do *OpenSocial* e desenvolver uma aplicação *OpenSocial* experimental para o Palco Principal.

O Palco Principal usa a *framework symfony* para o desenvolvimento do seu site, e por isso, será realizado um *plugin* que permite adaptar esta plataforma para o desenvolvimento de aplicações OpenSocial.

Esta aplicação deverá também permitir a integração com o site do Palco Principal, fornecendo funcionalidades que gerem vantagens para os seus utilizadores e parceiros.

1.4 Organização do relatório

Este relatório encontra-se organizado em sete capítulos.

No **Capítulo 1**, é feita uma breve apresentação do trabalho e é sintetizado o seu conteúdo.

O **Capítulo 2** é dedicado ao *background* que é necessário para a realização deste projecto, incluindo as tecnologias utilizadas no mesmo.

O **Capítulo 3** está reservado para descrever a implementação realizada tanto do *plugin symfony* como da aplicação *OpenSocial*.

No **Capítulo 4**, é feita uma análise dos resultados obtidos com ambas as implementações.

Finalmente no **Capítulo 5** apresentam-se as principais conclusões e algumas ideias para futuros desenvolvimentos e expansões do projecto realizado.

O **Capítulo 6** incluiu as referências bibliográficas usadas neste documento e o **Capítulo 7**, está dedicado a anexos.

Capítulo 2: Background e tecnologias usadas

2.1 Palco Principal

A Palco Principal é uma empresa que surgiu em Junho de 2006 com o apoio do Programa NEOTEC.

Focada no mercado musical, actua no contexto da *Web 2.0*, oferecendo uma componente tecnológica e forma de negócio inovadora.

O Portal (www.palcoprincipal.com) alberga já uma das maiores comunidades musicais de expressão portuguesa presentes na Web, agrupando ouvintes e artistas (mais de 6800 bandas, mais de 22000 utilizadores registados, mais de 15000 músicas disponíveis para audição e download e mais de 30000 visitantes regulares).

Exemplo da sua consolidação enquanto projecto empresarial, foi a parceria comercial com o grupo Sonaecom (inclusão do site como site principal de música independente do portal Clix e gestão comercial das principais áreas de publicidade).

Durante o ano de 2007 pôde fechar um primeiro contrato de desenvolvimento tecnológico, reunindo conteúdos musicais, gestão de comunidades e o marketing de uma das maiores empresas portuguesas de venda de produtos de consumo doméstico: Worten - Equipamentos para o Lar.

A expansão no mercado tem-se feito de forma gradual tendo as audiências aumentado de forma considerável ao longo do último ano. A penetração no mercado brasileiro permitiu que no passado mês de Dezembro, em visita a São Paulo, Brasil, se tenham negociado um conjunto de questões que permitiram a constituição da empresa naquele Pais, e que vão garantir o estabelecimento, no 2º trimestre de 2008, de uma parceria com um dos mais importantes portais brasileiros: UOL.com.br.

2.1.1 Posicionamento

A empresa encara a Internet como um espaço de interacção e comunicação, associando novos canais de

venda, e que vai continuar a registar forte crescimento acompanhado de novos paradigmas de negócio.

Tem como objectivo primário que os músicos se promovam, interajam por novas formas com o seu público, e tudo isto criando valor no processo.

Foca-se na captação da atenção e confiança dos utilizadores, fidelizando-os, através de soluções personalizadas, agregando fortes comunidades de interesses.

Tem como compromisso o respeito pelos direitos de autor, o combate à pirataria e a qualquer sistema de fraude que ponha em causa os direitos dos artistas.

2.1.2 Utilizadores do Palco Principal

O Palco Principal permite dois tipos de utilizadores:

- *Banda*
- *Utilizador*

Para quem usa conta de *utilizador*, tem a oportunidade de criar uma página pessoal com fotografias, blog, músicas, bandas favoritas, etc. Depois podem adicionar amigos, tornarem-se fãs de bandas, comentar, serem comentados e interagir com os outros utilizadores com os quais têm maiores afinidades.

Um dos principais trunfos do Palco Principal é a plataforma que disponibiliza para quem quer mostrar e promover o seu trabalho. As bandas apenas têm de fazer o upload das suas músicas sem quaisquer custos, ficando as mesmas disponíveis na sua página do Palco Principal para audição gratuita.

No Palco Principal, cada música tem uma página própria. Em cada página, podem encontrar-se estatísticas, músicas relacionadas, comentários, letra, instrumentos envolvidos na sua criação, um player para audição e um código *embed* para inserção das músicas noutras páginas (Myspace, Hi5, blogspot, etc).

As músicas disponibilizadas poderão também ser enviadas a um amigo, ser alvo de *reviews*, e adicionadas à *playlist* de utilizadores do Palco Principal.

2.1.3 API RESTFul Beta

O Palco Principal está a desenvolver uma API RESTful, que permite que aplicações externas acedam ao site trocando mensagens por HTTP.

Esta é uma API ainda em desenvolvimento, mas que foi aproveitada neste projecto (ver secção 3.2.4 "*API da aplicação OpenSocial do Palco Principal*", página 57), e por consequente serão descritas as funcionalidades da mesma.

2.1.3.1 Login

Este comando permite que determinada aplicação externa faça login no Palco Principal. Quando alguém acede a este comando e faz *login* correctamente, é enviada uma *cookie* de sessão que permite ao utilizador

autenticar-se em futuros acessos.

A descrição deste comando pode ser encontrada em anexo na Tabela 7.1 "API RESTful do Palco Principal: login".

2.1.3.2 Playlist

Este comando permite obter a *playlist* de determinado *Utilizador* ou *Banda* do Palco Principal.

A descrição deste comando pode ser encontrada em anexo na Tabela 7.2 "API RESTful do Palco Principal: playlist".

2.1.3.3 Friends

Este comando permite obter os amigos de determinado *Utilizador* ou as bandas amigas de determinada *Banda* do Palco Principal.

A descrição deste comando pode ser encontrada em anexo na Tabela 7.1"API RESTful do Palco Principal: login".

2.1.3.4 Photos

Este comando permite obter a fotos de determinado *Utilizador* ou *Banda* que estejam registados no Palco Principal.

A descrição deste comando pode ser encontrada em anexo na Tabela 7.4 "API RESTful do Palco Principal: photos".

2.2 Symfony Web PHP Framework

Symfony é uma *framework* completa que é desenhada para optimizar o desenvolvimento de aplicações para a *Web.*

O *symfony* é escrito em *PHP5* que implementa o paradigma de programação orientada a objectos. Com a inclusão do *PHP5* podemos usar classes, objectos, métodos, herança e muitas outras funcionalidades.

2.2.1 PHP Extension and Application Repository (PEAR)

PEAR (http://pear.php.net/) é uma *framework* específica para componentes reutilizáveis em *PHP*. Esta *framework* permite fazer *download*, instalar, desinstalar, actualizar *scripts PHP*, sem nos preocuparmos onde os colocar ou em como torná-los disponíveis.

PEAR é um projecto *Open Source* desenvolvido pela comunidade e escrito em *PHP*.

2.2.2 YAML

O *symfony* usa por omissão ficheiros em *YAML*[1] para as suas configurações. O grande motivo para estes serem usados, prende-se pela simplicidade da sua sintaxe e facilidade de leitura.

2.2.3 Organização da estrutura de ficheiros

No *symfony* um projecto consiste num conjunto de serviços e operações disponíveis em determinado domínio.

Dentro de um projecto, as operações estão organizadas logicamente em aplicações.

O *symfony* permite a criação automática de projectos usando uma "*Pake Task*", que cria a estrutura base de um projecto *symfony*.

O comando *Pake* para iniciar um projecto é o seguinte:

```
> symfony init-project <nome_do_projecto>
```

Listagem 2.1: Como criar um projecto em *symfony*

Quando se executa este comando é gerada a seguinte estrutura de ficheiros:

Directoria	Descrição
apps/	Contém uma directoria para cada aplicação
batch/	*Scripts PHP* para serem executados
cache/	Contém a *cache* da aplicação
config/	Contém os ficheiros de configuração gerais do projecto
data/	Aqui pode conter por exemplo esquemas de bases de dados
doc/	Ficheiros de documentação do projecto
lib/	Contém classes ou bibliotecas externas
log/	Onde são guardados os ficheiros de *log*
plugins/	Os *plugins* que estão instalados no projecto
test/	Contém testes unitários ou funcionais
web/	A raiz do servidor *Web*
css/	Onde são guardados os ficheiros de *CSS*
images/	Onde são guardadas as imagens usadas nos *templates*
js/	Onde são guardados os ficheiros de *JavaSript*

Tabela 2.1: Estrutura da raiz de um projecto em symfony

Cada aplicação é constituída por um conjunto de módulos que normalmente representam um grupo de páginas similares.

O *symfony* disponibiliza também uma "*Pake Task*" para criar aplicações em determinado projecto.

Para isso basta executar o seguinte comando na raiz do projecto onde se pretende criar a aplicação:

```
> symfony init-app <nome_da_aplicação>
```

Listagem 2.2: Como criar uma aplicação em determinado projecto *symfony*

Após executar este comando deve ser gerado a seguinte estrutura de directorias, no caminho

"*meuProjecto/apps/minhaAplicação*":

Directoria	Descrição
config/	Contém ficheiros de configuração específicos da aplicação
il8n/	Ficheiros usados para internacionalizar a aplicação
lib/	Contém bibliotecas ou classes específicas para a aplicação
modules/	Contém os *modules* da aplicação
templates/	Contém os *templates* globais da aplicação

Tabela 2.2: Estrutura das directorias de uma aplicação em symfony

Também a criação de *modules* (ou módulos) é automatizada e para isso basta executar a seguinte linha numa consola ou terminal:

```
> symfony init-module <nome_da_aplicação> <nome_do_modulo>
```

Listagem 2.3: Como criar uma aplicação em determinado projecto *symfony*

Este comando irá gerar um determinado *module* em determinada aplicação e por omissão cria a seguinte estrutura de directorias:

Directoria	Descrição
actions/	Possui as acções e os *components* de determinado *module*
config/	Contém ficheiros locais de configuração
lib/	Contém bibliotecas ou classes específicas ao *module*
templates/	Contém os *templates* usados no *module*
validate/	Dedicada a ficheiros usados para validação de formulários

Tabela 2.3: Estrutura da directoria de um module em symfony

2.2.4 MVC no symfony

Symfony usa o clássico modelo para *web* conhecido como *MVC*, o qual consiste entre três níveis:

- O *model* representa a informação na qual a aplicação opera;
- A *view*, é a camada de visualização;
- O *controller*, responde às acções impostas pelo utilizador e invoca alterações no *model* ou na *view* conforme os casos.

2.2.4.1 Camada de controlo

No *symfony*, a *controller layer*, que contém o código que faz a ligação entre a lógica de negócio e a apresentação, está dividida em vários componentes que se podem usar para diferentes propósitos:

- O *front controller* é o ponto de entrada único para a aplicação. Este carrega as configurações e determina a acção a executar.
- As acções contém a lógica da aplicação. Estas verificam a integridade do pedido e geram a informação necessária para a camada de apresentação.
- Filtros são porções de código que são executados em cada pedido, antes ou depois da acção.

2.2.4.1.1 Acções

As acções ou *Actions*, são centrais para a aplicação, pois são elas que contém a lógica da aplicação. Estas usam o *model* e definem as variáveis a ser usadas na apresentação.

Quando é realizado algum pedido via *web* a uma aplicação *symfony*, o *URL* define a acção a ser executada.

2.2.4.1.2 Filtros

Quando uma acção é executada, esta pode ter de passar por diversos filtros. Os Filtros podem modificar todo o conteúdo de um pedido antes da acção ser executada ou podem alterar a resposta depois da mesma ser executada.

O *symfony* define uma classe chamada ***sfFilter*** e todos os filtros criados têm de estender esta classe.

Um filtro para ser executado tem de ser inserido na "*Filter Chain*" (corrente de filtros) de determinada acção. A corrente de filtros é executada sequencialmente e só quando todos os filtros que estão na corrente são executados é que a *view* é mostrada ao utilizador.

2.2.4.2 Camada de apresentação

A camada de apresentação (ou *View Layer*) é responsável por gerar uma página de acordo com uma acção.

No *symfony*, a camada de apresentação consiste em várias partes, as quais são desenhadas para serem facilmente modificadas pela pessoa que está responsável por trabalhar nela.

2.2.4.2.1 Helpers

Helpers são funções em *PHP* que retornam código *HTML* para ser usado em *templates*. Estes são usados apenas para agrupar conteúdo usado frequentemente em *templates*, e agrupando-o em apenas um método, tornando-se assim fácil manter essa secção de código.

Os ficheiros *symfony* que contém definições de *Helpers* não são carregados automaticamente pela *framework* (pois estes contém funções e não classes). Para se usar um *Helper* em determinado *template* é necessário primeiro declara-lo.

```
<?php use_helper('OpenSocial') ?>
...
<?php echo include_css('PalcoPrincipal.css') ?>
```

Listagem 2.4: Como declarar um *Helper*

2.2.4.2.2 Components

Components funcionam de forma bastante similar a uma *action*, com a particularidade de serem mais

rápidos. Tal como nas *actions*, *components* podem executar métodos e passar variáveis para a apresentação do mesmo.

Components não podem ser acedidos directamente por *URL*, pois para determinado *component* ser executado, este tem de ser chamado internamente pela aplicação (podendo ser chamado por outros *components*, pelo *layout* global ou por um *template* normalmente).

```
// dentro de um template fazer
<?php include_component('opensocial', 'requestPerson', array('type' =>
'VIEWER')) ?>
```

Listagem 2.5: Como incluir um *component* num *template*

Por exemplo, na Listagem 2.5, estamos a incluir o *component* "*requestPerson*" que se encontra no módulo "*opensocial*" e a passar-lhe a variável "*type*" com o valor "*VIEWER*".

2.2.4.2.3 Configuração do Layout

O *symfony* permite definir diferentes *layouts* de acordo com as necessidades de cada aplicação.

O *layout* definido por omissão encontra-se na raiz da aplicação, na pasta "*templates/*" (ver secção "Organização da estrutura de ficheiros" na página 8). *Layouts* adicionais podem ser adicionados à pasta "*templates/*", e, para os usarmos em determinada *view*, é necessário alterar o ficheiro de configurações "*view.yml*" que se encontra na pasta "*config/*", definindo o *layout* da seguinte forma:

```
indexSuccess:
  layout: my_layout
```

Listagem 2.6: Como definir novo *layout* no ficheiro "*view.yml*"

Desta forma estamos a dizer ao *symfony* que sempre que o "*index*" seja executado com sucesso, o *layout* a ser usado é o "*my_layout*".

2.2.4.3 Camada de modelo

A lógica de negócio de uma aplicação *web*, centra-se especialmente na sua modelação de dados (*Model Layer*).

O *symfony* usa o *Propel* [2] para gerir o mapeamento objecto-relacional. Assim sendo, podemos aceder a dados guardados numa Base de Dados relacional e modificá-los usando objectos.

Esta camada de abstracção introduzida pelo *Propel*, facilita ainda a reutilização de código e a fácil migração entre Bases de Dados.

2.2.4.3.1 Gerar o modelo

Para ser possível gerar o modelo da Base de Dados, o *symfony* necessita que seja configurado o ficheiro "*schema.xml*" que se encontra na pasta "*config/*" da raiz do projecto.

Este ficheiro contem a estrutura da base de dados, independente do fabricante, que permite gerar todas as classes para lidar com a mesma.

O *symfony* permite ainda importar a estrutura de uma base de dados já existente, gerando

automaticamente o ficheiro "schema.xml" de acordo com a essa estrutura.

Para que o symfony consiga aceder à estrutura da base de dados, é necessário configurar o ficheiro "*propel.ini*" com os dados de acesso à Base de Dados e depois executar a seguinte "*Pake Task*":

```
> symfony propel-build-schema
```

Listagem 2.7: Comando para gerar o "*schema.xml*" a partir de uma Base de Dados existente

Uma vez tendo o ficheiro "schema.xml" gerado, apenas precisamos de executar o seguinte comando para construir o modelo:

```
> symfony propel-build-model
```

Listagem 2.8: Comando para construir o modelo

Não esquecer que é necessário limpar a cache do *symfony* por forma que as novas classes criadas sejam carregadas pela *framework*.

2.2.5 Plugins

Usar *plugins* em *symfony* é a melhor forma de se reutilizar código entre diversos projectos. Em *plugins* podem-se agrupar classes, filtros, *mixins*, *helpers*, ficheiros de configuração, *tasks*, módulos, *schemas*, etc.

Plugins são simples de instalar, actualizar e desinstalar. Podem ser distribuídos como arquivos "*.tgz*", como pacotes *PEAR* ou simplesmente fazendo *checkout* de um repositório de controlo de versões.

A *framework symfony* tem em consideração os *plugins* e carrega-os automaticamente, como se fizessem parte da própria *framework*.

Um *plugin* é visto como uma extensão para determinado projecto *symfony*, que não só permite a reutilização de código por várias aplicações como também adicionar extensões ao núcleo do *symfony*.

2.2.5.1 Procurar plugins disponíveis para o symfony

A listagem oficial de *plugins* para o *symfony* pode ser encontrada no seguinte endereço *web*:

- http://trac.symfony-project.com/wiki/SymfonyPlugins

Aqui podem-se encontrar *plugins* desenvolvidos pelos programadores do núcleo do *symfony* mas também contribuições feitas pela comunidade.

Cada *plugin* tem a sua própria *wiki* onde contêm instruções de instalação e documentação.

2.2.5.2 Como instalar um Plug-in

O processo de instalação de um *plugin* pode varia de acordo com a forma como este foi criado.

Deve-se sempre consultar o ficheiro *README* (que se encontra por omissão na raiz do *plugin*) ou a página de instalação para se efectuar um correcta instalação do mesmo. Não esquecer que é necessário limpar a *cache* do *symfony* por forma a que este seja carregado pela *framework*.

Plugins são instalados por projecto, e basicamente consiste em colocar todos os ficheiros do *plugin* na

directoria "*meuProjecto/plugins/nomeDoPlugin/*".

2.2.5.2.1 Plugins PEAR

Os *plugins* que são listado na *wiki* do *symfony* são agrupados como um pacote *PEAR* e ficam anexados como um arquivo à *wiki* de cada *plugin*.

Para se instalar um plugin nestas condições, deve-se usar a "*Pake Task*" "*plugin-install*" com o endereço URL do mesmo, tal como é mostrado na listagem seguinte:

```
> cd meuProjecto
> symfony plugin-install http://plugins.symfony-project.com/pluginName
> php symfony cc
```

Listagem 2.9: Instalar plugin pelo track do symfony

2.2.5.2.2 Como activar um módulo de um *plugin*

Plugins podem conter módulos completos. A única diferença entre activar um módulo de um *plugin* ou módulo "clássico", é que este módulo não vai aparecer na directoria "*meuProjecto/apps/minhaAplicacao/modules*" (por forma a tornar mais simples futuras actualizações do *plugin*).

Estes precisam também de ser activados usando o ficheiro "*settings.yml*" tal como é mostrado na listagem seguinte:

```
all:
  .settings:
    enabled_modules:  [default, sfMyPluginModule]
```

Listagem 2.10: Activar um módulo de um *plugin*

Isto evita situações onde o módulo do *plugin* é erradamente tornado disponível à aplicação, quando esta não necessita dele, podendo conduzir a falhas de segurança.

Assim, podemos ter instalados os *plugins* que desejarmos, activando apenas os módulos que achamos convenientes para determinada aplicação.

2.2.5.3 Anatomia de um Plug-In

Plugins são escritos na mesma linguagem da *framework symfony* que é o *PHP*, e a sua estrutura é similar à estrutura da organização das aplicações.

2.2.5.3.1 Organização da estrutura de ficheiros

Uma directoria de um *plugin* é organizada praticamente da mesma forma como é organizada a estrutura de um projecto *symfony*. Os ficheiros têm de estar nas directorias correctas por forma a que o *symfony* carregue os ficheiros automaticamente quando necessite deles.

Abaixo encontra-se uma listagem a mostrar a estrutura de ficheiros de um *plugin* para a *framework*

symfony:

```
pluginName/
  config/
    *schema.yml        // Data schema
    *schema.xml
    config.php         // Specific plug-in configuration
  data/
    generator/
      sfPropelAdmin
        */             // Administration generator themes
          template/
          skeleton/
    fixtures/
      *.yml            // Fixtures files
    tasks/
      *.php            // Pake tasks
  lib/
    *.php              // Classes
    helper/
      *.php            // Helpers
    model/
      *.php            // Model classes
  modules/
    */                 // Modules
      actions/
        actions.class.php
      config/
        module.yml
        view.yml
        security.yml
      templates/
        *.php
      validate/
        *.yml
  web/
    *                  // Assets
```

Listagem 2.11: Estrutura de ficheiros de um plugin

2.2.5.4 Como criar um plugin

Para se criar um *plugin* apenas é necessário criar uma directoria com a estrutura que se encontra na Listagem 2.11, mas se se quiser que este seja instalado usando *PEAR* é necessário respeitar determinadas normas.

2.2.5.4.1 Criar o ficheiro package.xml

Por forma a que o *plugin* seja automaticamente reconhecido, é necessário adicionar na raiz do mesmo o ficheiro "*package.xml*". O ficheiro "*package.xml*" segue a sintaxe usada pelo *PEAR* e na Listagem 2.12 pode-se ver uma estrutura típica do mesmo:

```
<?xml version="1.0" encoding="UTF-8"?>
<package packagerversion="1.4.6" version="2.0"
xmlns="http://pear.php.net/dtd/package-2.0"
```

```
xmlns:tasks="http://pear.php.net/dtd/tasks-1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://pear.php.net/dtd/tasks-1.0
http://pear.php.net/dtd/tasks-1.0.xsd http://pear.php.net/dtd/package-2.0
http://pear.php.net/dtd/package-2.0.xsd">
 <name>sfSamplePlugin</name>
 <channel>pear.symfony-project.com</channel>
 <summary>symfony sample plugin</summary>
 <description>Just a sample plugin to illustrate PEAR
packaging</description>
 <lead>
  <name>Fabien POTENCIER</name>
  <user>fabpot</user>
  <email>fabien.potencier@symfony-project.com</email>
  <active>yes</active>
 </lead>
 <date>2006-01-18</date>
 <time>15:54:35</time>
 <version>
  <release>1.0.0</release>
  <api>1.0.0</api>
 </version>
 <stability>
  <release>stable</release>
  <api>stable</api>
 </stability>
 <license uri="http://www.symfony-project.org/license">MIT license</license>
 <notes>-</notes>
 <contents>
  <dir name="/">
   <file role="data" name="README" />
   <file role="data" name="LICENSE" />
   <dir name="config">
    <!-- model -->
    <file role="data" name="schema.yml" />
   </dir>
   <dir name="data">
    <dir name="fixtures">
     <!-- fixtures -->
     <file role="data" name="fixtures.yml" />
    </dir>
    <dir name="tasks">
     <!-- tasks -->
     <file role="data" name="sfSampleTask.php" />
    </dir>
   </dir>
   <dir name="lib">
    <dir name="model">
     <!-- model classes -->
     <file role="data" name="sfSampleFooBar.php" />
     <file role="data" name="sfSampleFooBarPeer.php" />
    </dir>
    <dir name="validator">
     <!-- validators ->
     <file role="data" name="sfSampleValidator.class.php" />
    </dir>
   </dir>
   <dir name="modules">
    <dir name="sfSampleModule">
```

```
     <file role="data" name="actions/actions.class.php" />
     <file role="data" name="config/security.yml" />
     <file role="data" name="lib/BasesfSampleModuleActions.class.php" />
     <file role="data" name="templates/indexSuccess.php" />
    </dir>
   </dir>
   <dir name="web">
    <dir name="css">
     <!-- stylesheets -->
     <file role="data" name="sfSampleStyle.css" />
    </dir>
    <dir name="images">
     <!-- images -->
     <file role="data" name="sfSampleImage.png" />
    </dir>
   </dir>
  </dir>
 </contents>
 <dependencies>
  <required>
   <php>
    <min>5.0.0</min>
   </php>
   <pearinstaller>
    <min>1.4.1</min>
   </pearinstaller>
   <package>
    <name>symfony</name>
    <channel>pear.symfony-project.com</channel>
    <min>1.0.0</min>
    <max>1.1.0</max>
    <exclude>1.1.0</exclude>
   </package>
  </required>
 </dependencies>
 <phprelease />
 <changelog />
</package>
```

Listagem 2.12: Exemplo de um ficheiro *package.xml*

O ficheiro "*package.xml*" vai servir para descrever o *plugin* e entre outras características permitir o controlo de versões.

Os elementos do *XML* que devem ser usados para personalizar o *plugin* são os seguintes:

- ***name*:** aqui fica o nome do *plugin*
- ***summary :*** é uma pequena descrição do que faz o *plugin*
- ***description:*** uma descrição mais completa do *plugin*
- ***lead:*** aqui encontra-se a informação de quem desenvolveu/mantém o *plugin*
- ***date:*** a data de lançamento desta versão
- ***version :*** a versão actual do *plugin* (isto é importante para se poder fazer *updates*)
- ***contents :*** os ficheiros que compõem este *plugin*

2.2.5.4.2 Criar o ficheiro README

Este é um ficheiro opcional, mas para que o *plugin* seja aprovado como *plugin symfony* deve existir na raiz do mesmo.

Este ficheiro não tem nenhuma sintaxe específica mas deve conter os seguinte conteúdo:

```
= sfSamplePlugin =
// nome do plugin
== Overview ==
// secção a explicar o que faz o plugin
== Installation ==
// secção a descrever como se instala o plugin
== Usage ==
// secção a explicar como se usa o plugin
== License ==
// a licença do plugin
```

Listagem 2.13: Exemplo de um ficheiro README

É aconselhado que o conteúdo do ficheiro *README* seja o mesmo conteúdo que se encontra na *wiki* referente ao *plugin*[3].

2.2.5.4.3 Usar o sfPluginManagerPlugin para gerar o plugin

Pode-se criar um *plugin* sem auxilio de ferramentas, mas o *symfony* possui um *plugin* que torna mais simples o empacotamento de *plugins* para o *symfony*.

Este plugin chama-se *sfPluginManager* e pode ser encontrado em http://trac.symfony-project.com/wiki/sfPluginManagerPlugin.

Uma vez tendo este *plugin* instalado deve-se seguir os seguintes passos:

1. Ir à pasta "*plugins*" dentro do projecto e criar uma pasta com o nome do *plugin* que queremos criar:

```
$ mkdir sfSamplePlugin
```

Listagem 2.14: Criar uma directoria com o nome do *plugin*

2. Colocar os ficheiros do *plugin*, seguindo a estrutura da Listagem 2.11 (página 14), dentro da directoria criada.
3. Copiar o ficheiro "*package-template.xml*" que se encontra na directoria "meuProject/sfPluginManagerPlugin/data/tasks" para a raiz do "sfSamplePlugin" criado
4. Executar a *Pake Task* do plugin na raiz do "meuProjecto":

```
$ symfony build-plugin-package sfSamplePlugin
```

Listagem 2.15: Criar o ficheiro "*package.xml*" usando o *sfPluginManagerPlugin*

5. Ir até à directoria do *sfSamplePlugin* e executar o comando:

```
$ symfony build-plugin-package sfSamplePlugin
```

Listagem 2.16: Criar o *PEAR package* usando o *sfPluginManagerPlugin*

Uma vez tendo seguido estes passos é gerado um ficheiro com o nome "*sfSamplePlugin-0.0.1.tgz*"[1] na

1 Na altura da criação deste documento existia um *bug* no *sfPluginManagerPlugin* que colocava no ficheiro "*package.xml*" na raiz do XML "*package/dependencies/package/name*" o nome do *plugin*. Desta forma o ficheiro

raiz do *plugin*, que pode ser incluído na *wiki* do *plugin* como um pacote *PEAR*.

2.2.5.5 Alojar plugins para o Symfony

O *symfony* disponibiliza gratuitamente um repositório com *subversion* (*SVN*[4]) para alojamento, uma secção de *issues* e uma *wiki* para descrever o *plugin* criado.

Na secção "Como criar um plugin" (página 14) é descrita de forma sucinta as características que um *plugin* deve possuir para ser aprovado como *plugin* para a plataforma de desenvolvimento *symfony*, no entanto existem dois endereços que devem ser consultados por forma a garantir que o *plugin* está de acordo com a especificação:

- http://trac.symfony-project.com/wiki/SymfonyPlugins#Howtocreateandcontributeasymfonyplugin
- http://trac.symfony-project.com/wiki/SymfonyPlugins#Hostingatsymfony-project.com

Após concluir todos os passos é então possível ter um *plugin* para a plataforma *symfony*.

2.3 API OpenSocial

O *OpenSocial* é o nome dado um conjunto comum de *APIs* desenhadas para redes sociais.

O objectivo desta *API* é permitir que terceiros desenvolvam aplicações usando a *API OpenSocial*, que por sua vez possam ser usadas em múltiplos *sites* desde que estes sejam *containers OpenSocial[5]*.

Programadores podem criar aplicações em *JavaSript* e *HTML*, que podem ser executadas em *sites* que implementem a *API OpenSocial*.

Esta *API* expõe métodos para aceder a informações relativas a pessoas, aos seus amigos, e a sua informação, dentro do contexto de um dado *container*.

Isto significa que uma aplicação que seja escrita para OpenSocial, se for executada dentro do Orkut, interage com os amigos no Orkut, enquanto que a mesma aplicação no MySpace permite interagir com os amigos do MySpace.

2.3.1 Especificação da API OpenSocial

2.3.1.1 O que é necessário para ser um container OpenSocial?

Ser um *container OpenSocial*, significa que determinado *site* implementa os métodos *JavaSript* da *API*

não poderia ser instalado correctamente usando as "*Pake Task*"s do *symfony*.
Caso a situação ainda se mantenha, deve-se manualmente editar o conteúdo do ficheiro "*package.xml*" substituindo "*<package>...<dependencies><package><name>sfSamplePlugin</name></package></dependencies></package>*" por "*<package>...<dependencies><package><name>symfony</name></package></dependencies></package>*" e voltar a repetir a Listagem 2.16.

OpenSocial[6].

Esta prevê também que o *container* possa não ter determinadas funcionalidades implementadas, mas para isso deve retornar "*opensocial.ResponseItem.NOT_IMPLEMENTED*" quando determinada funcionalidade seja requisitada.

Para além da *API OpenSocial*, um *container* tem também de implementar a *Gadgets API*[7], que permite usar funcionalidades extra.

2.3.1.2 Aspectos fundamentais

2.3.1.2.1 Pessoas

As redes sociais são feitas por pessoas e as pessoas são a parte fundamental da *API OpenSocial*.

O objecto *Person[8]* proporciona acesso à informação dos utilizadores. Parte desta informação é guardada no perfil do utilizador e, dependendo do site, pode incluir qualquer parâmetro específico à rede social onde se encontra.

Existe ainda a distinção entre *VIEWER* e *OWNER*. O *VIEWER* é uma pessoa que se encontra a ver determinada aplicação *OpenSocial*; enquanto que o *OWNER* é a pessoa que tem a aplicação instalada.

2.3.1.2.2 Relações

A possibilidade de criar relações é o que a torna numa aplicação social.

Existem duas colecções de objectos *Person* que podem ser pedidos directamente ao *container*:

- *VIEWER_FRIENDS*;
- *OWNER_FRIENDS*.

Fazer um pedido pelos *VIEWER_FRIENDS* retorna o conjunto de utilizadores que são amigos do utilizador que fez o pedido. Enquanto que um pedido pelos *OWNER_FRIENDS* retorna os amigos do utilizador de quem estamos a ver o perfil.

No caso de se fazer um pedido do tipo *VIEWER_FRIENDS* e *OWNER_FRIENDS* no próprio perfil, é retornado o mesmo conjunto de amigos.

Caso o *container* permita pesquisar perfis de forma anónima, não é possível obter os *VIEWER_FRIENDS*.

Notar também que o *OpenSocial* não faz qualquer assumpção acerca da relação entre *VIEWER* e *OWNER*.

2.3.1.2.3 Actividades

Como não é possível estar-se sempre *online*, é importante guardar-se um registo do que os amigos fazem, desde que estes queiram partilhar essa informação.

Ver a forma como as outras pessoas estão a interagir com a aplicação social permite depreender novas funcionalidades e usos para a mesma, por isso fluxos de actividades são muito importantes para o crescimento de aplicações orgânicas.

O *Opensocial* expõe fluxos de actividades (*activity streams*), que são colecções de acções de um utilizador realizadas em determinado contexto de um dado *container*.

Estas actividades podem incluir interacções com o próprio *container*, tais como actualizações do perfil ou instalação de um novo *Gadget*, ou interacções com uma aplicação *OpenSocial*.

Templates de actividades permitem aos criadores de aplicações definir mensagens com *placeholders* para partes da aplicação ou informação do utilizador. Esta separação de informação e apresentação permite que múltiplas actividades sejam combinadas em sumários de actividades.

2.3.1.2.4 Persistência

Aplicações podem proporcionar aos seus utilizadores uma experiência mais rica caso permitam guardar estados entre sessões.

O *OpenSocial* define uma área de dados que as aplicações podem usar para ler e escrever informação específica de cada utilizador. Esta informação pode ser lida por qualquer um que use a aplicação, mas apenas na perspectiva do *VIEWER* é que a informação pode ser escrita.

Obviamente esta área de armazenamento livre pode estar sujeita a abusos, por isso os *containers* podem impor quotas para preservar o espaço em disco. De qualquer maneira actualmente o *OpenSocial* não define este tipo de políticas.

2.3.1.2.5 Vistas

Os *containers* podem suportar várias localizações diferentes onde os *Gadgets* podem ser apresentados.

Essas localizações são formalmente chamadas de vistas. Todos os *Gadgets* (não só aplicações *OpenSocial*) tornaram-se conscientes da sua localização (*view-aware*).

Um *Gadget* pode saber em que vista esta a ser mostrado e quais são as vistas que o *container* suporta.

Os *containers* podem ainda definir as suas próprias vistas com características específicas. Os exemplos mais comuns de vistas são:

- **Perfil** (*Profile*) - Um *Gadget* na vista de Perfil é mostrado em conjunto com outras aplicações no perfil do utilizador, por isso o tamanho da sua janela é mais pequeno e não permite passar parâmetros *URL* do *container* para o *Gadget*.
- **Tela** (*Canvas*) **-** A vista de Tela, apenas mostra uma única aplicação, por isso suporta a passagem de parâmetros pelo *URL* e habitualmente o tamanho da janela é maior.
- **Pré-visualização** (*Preview*) - Esta vista já não é tão comum, mas é usada por alguns *containers* para mostrar uma Pré-visualização da aplicação.

Para além de ser possível saber em que vista se encontra determinado *Gadget*, é também possível solicitar ao *container* para redireccionar o *VIEWER* para uma determinada vista disponível.

2.3.1.3 Padrões da API

2.3.1.3.1 Fazer pedidos ao container

A *API OpenSocial* permite fazer pedidos ao *container* de forma assíncrona. Devido a este facto a maior parte dos métodos que fazem pedidos ao *container* não retornam informação directamente, especificando uma função de retorno (*callback*) que será executada quando a resposta do servidor esteja pronta.

Claro que fazer muitos pedidos assíncronos nem sempre é o ideal, por isso a *API* permite aglomerar vários pedidos de informação numa única chamada ao *container*.

Um programador pode assim criar um "*opensocial.DataRequest[9]*" e adicionar vários pedidos de objectos individuais. Aquando da recepção do *DataRequest*, o *container* pode processar cada um dos pedidos de forma optimizada, retornando o resultado de cada operação como um único objecto.

Os *containers* têm de preservar a ordem sequencial dos pedidos. Um pedido que contém uma escrita e depois uma leitura tem de retornar a nova informação escrita, enquanto um pedido que contenha uma leitura e depois uma escrita tem de retornar a informação que estava presente antes da escrita.

2.3.1.3.2 Encontrar funcionalidades específicas de determinado *container*

A especificação dos *Gadgets* e *OpenSocial*, determinam as *APIs* comuns que todos os *containers* têm de suportar, mas existem casos onde certos campos de perfil serão oferecidos como extensão pelos *containers*.

Os perfis de utilizador podem ser diferentes de *container* para *container* e o OpenSocial permite estender os objectos *Person* com campos arbitrários por forma a coincidir com o aspecto e uso de cada *container*.

Para se obter uma determinada propriedade de um campo do perfil, tem de se usar o método definido pela *API* "*opensocial.Person.getField()*"*[10]*.

Para ajudar os programadores a criar aplicações que tirem partido destas extensões, e saibam lidar na ausência das mesmas, a *API* inclui os métodos "*gadgets.util.hasFeature()*"[11] e o "*opensocial.Environment.supportsField()*"[12] que permitem saber em tempo de real se um *container* suporta determinada funcionalidade.

2.3.1.3.3 Efectuar pedido a servidor remoto

Aplicações OpenSocial podem usar o método "*gadgets.io.makeRequest()*"[13] para obter *HTML*, *JSON* e *ATOM data*, de servidores remotos. A chamada "*gadgets.io.makeRequest()*" pode também ser usada para passar dados dos G*adgets* para um servidor de aplicações de uma forma que não possa ser "*spoofed*"[14].

É esperado que o *container* faça a mediação da comunicação entre o *Gadget* e o servidor de aplicações.

Assim sendo obter conteúdo de forma segura envolve dois passos:

1. o *Gadget* contacta o *container*;
2. o *container* contacta o servidor de aplicações.

No **passo 1** (*gadget→container*), o *container* necessita de ter a possibilidade de validar qualquer parâmetro que conhece: o *ID* do *VIEWER*, o *ID* do *OWNER* (se for conhecido), e o *ID* da aplicação. A validação destes parâmetros depende do nível de detalhe que cada *container* usa na sua implementação.

Para o **passo 2** (*container→*servidor de aplicações), o servidor de aplicações necessita de ter a possibilidade de validar que os parâmetros obtidos do *container*, garantindo que não foram forjados por qualquer outra entidade. O *OpenSocial* usa o *OAuth* (http://oauth.net/) para fazer esta autenticação.

2.3.2 Containers OpenSocial

Actualmente existe já uma lista de *sites* que afirmam suportar a *API OpenSocial 0.7[15]*, mas na realidade poucos são os que a suportam na sua integridade.

Abaixo fica uma pequena descrição relativamente aos *containers* que foram testados, indicando o endereço do *container* (*URL*), o endereço da *sandbox* (ou seja endereço de testes *OpenSocial*), a versão que suportam e o tempo que levam a aprovar/rejeitar determinada aplicação.

2.3.2.1 Hi5

- ***URL***: http://www.hi5.com
- ***Sandbox***: http://sandbox.hi5.com/
- ***OpenSocial***: suporta a versão 0.7
- **Tempo de submeter aplicação**: 1 a 2 dias úteis

O Hi5 é actualmente o *site* que fornece um melhor nível de suporte a *API OpenSocial*. Actualmente suporta todas as funcionalidades da versão 0.7 e está já em vista o suporte à nova versão 0.8[16]

O Hi5 permite que os programadores testem as suas aplicações usando a *sandbox*, e tem um período muito curto de aprovação de aplicações.

Este permite ainda que as aplicações usem publicidade apenas no modo "*Canvas*" e apenas uma barra horizontal/vertical.

2.3.2.2 MySpace

- ***URL:*** http://www.myspace.com
- ***Sandbox***: http://developer.myspace.com/
- ***OpenSocial***: Pouco suporte à versão 0.7. Não suporta *Skins*, multi-linguas e muitas funcionalidades existentes na *Gadgets API*.

- **Tempo de submeter aplicação**: 3 semanas a um mês

O MySpace tem ainda um suporte muito limitado ao *OpenSocial* e o seu período de aprovação de aplicações é demasiado longo.

2.3.2.3 Orkut

- **url**: http://www.orkut.com
- **sandbox**: http://sandbox.orkut.com
- **OpenSocial**: suporta a versão 0.7
- **Tempo de submeter aplicação**: 3 semanas a um mês

O Orkut suporta a *API* v0.7 do *OpenSocial* e também permite testar as aplicações em tempo real, caso se coloque no endereço da aplicação o parâmetro "*&bpc=1*".

Este parâmetro é só para testes e faz com que o Orkut não use a versão da aplicação que se encontra em *cache*.

2.3.2.4 Netlog

- **url**: http://en.netlog.com/
- **sandbox**: http://en.netlog.com/go/developer/opensocial
- **OpenSocial**: suporta a versão 0.7
- **Tempo de submeter aplicação**: ainda não permite submeter aplicações

O Netlog ainda não permite submeter aplicações à sua directoria, mas permite que se teste a aplicação no seu *container*.

2.3.2.5 imeem

- **url**: http://www.imeem.com
- **sandbox**: http://www.imeem.com/developers
- **OpenSocial**: suporta a versão 0.7
- **Tempo de submeter aplicação**: ainda não permite submeter aplicações

O imeem à semelhança do Netlog ainda não permite adicionar aplicações à sua directoria, mas permite que se testem as aplicações.

2.4 Google Code

O *Google Code* (http://code.google.com) é um site que permite o alojamento de projectos *Open Source*.

Este site disponibiliza várias funcionalidades, mas as mais relevantes são:

- Alojamento gratuito
- Controlo de versões com *Subversion* (*SVN*)
- *Wiki* colaborativa
- Uma secção de *Issues*
- Uma cota de 100 MB para os projectos

2.5 Google Analytics

O *Google Analytics* (http://www.google.com/analytics) é uma ferramenta para *webmasters* que permite analisar o tráfego em determinado *site*.

2.6 Google Trends

Google Trends (http://www.google.com/trends) é uma ferramenta do *Google Labs* que mostra os termos que foram mais pesquisados usando o motor de busca da *Google*.

O *Google Trends* permite também saber quais foram os sites mais visitados, permitindo ordenar por regiões.[17]

2.7 JavaScript

JavaScript é um linguagem de *script* usada habitualmente no desenvolvimento para a *Web*, que tanto pode ser usada do lado do cliente como do lado do servidor.

Esta linguagem foi amplamente usada neste projecto uma vez que a API *OpenSocial* é implementada na mesma.

Diversos documentos e artigos sobre esta linguagem foram consultados no decurso deste projecto, e os mais importantes podem ser consultados na secção de Referências Bibliográficas[18][19][20][21][22].

2.8 HTML, CSS e XML

Uma vez que a aplicação desenvolvida neste projecto é uma aplicação que vai ser executada num *Web browser*, foi necessário que esta usasse HTML (*HyperText Markup Language)* para criar a sua estrutura e conteúdo, e CSS (Cascading Style Sheets) para lidar com a apresentação.

O XML (Extensible Markup Language**)** foi usado especificamente para enviar dados pela rede ou receber dados provenientes de fontes externas.

As principais fontes de documentação tanto para XML[23][24], HTML[25] e CSS[26], foram artigos e tutorias que estão disponíveis na *web*.

Capítulo 3: Implementação

Este capítulo é dedicado a descrever e justificar tanto a implementação realizada do *plugin* para o *symfony*, como da aplicação *OpenSocial* desenvolvida para o Palco Principal.

3.1 Implementação do sfOpenSocialPlugin

sfOpenSocialPlugin é nome dado ao *plugin* realizado neste projecto, que permite desenvolver aplicações *OpenSocial* usando a *framework symfony*.

Este *plugin* é distribuído com a licença *MIT*[27] e tanto o código fonte como a sua documentação estão alojadas no site *Google Code* (ver secção 2.4, página 23) com o endereço:

- **Documentação e SVN:**
  http://sfOpenSocialPlugin.googlecode.com/

Este *plugin* está em conformidade com as regras de implementação de *plugins* para *symfony* (ver secção 2.2.5.4 "*Como criar um plugin*" página 14), e, por isso possui também uma página *wiki* no *track* do *symfony* em:

- **Projecto no *track* do *symfony*:**
  http://trac.symfony-project.com/wiki/sfOpenSocialPlugin

3.1.1 Abstract

O *sfOpenSocialPlugin* permite usar a plataforma de desenvolvimento *symfony* para criar aplicações *OpenSocial* tirando proveito de todas as funcionalidades disponíveis na *API OpenSocial v0.7*.
Este *plugin* fornece uma implementação completa Orientada a Objectos e em *PHP5*, da versão 0.7 da *API JavaSript* do *OpenSocial* (incluindo a *API* de *Gadgets* do núcleo e *feature-specific*), o que torna aplicações desenvolvidas em cima dele, mais robustas, escaláveis e imunes a futuras alterações da *API OpenSocial*.
As funcionalidades mais relevantes disponíveis neste *plugin* são:

- Todas as funcionalidades da *API OpenSocial*: Pessoas, Relações, Actividades, Persistência e Vistas;
- Configurações *standard* usando o ficheiro "*app.yml*";
- Realizar *pedidos* transparentes dentro das *Acções* e aceder às *respostas* dentro da *Visualização*;
- Integração transparente das funcionalidades "*feature-specific*" dos *Gadgets* sem necessidade de qualquer configuração adicional;
- Documentação de acordo com o *PHPDoc*.

3.1.2 Motivação

O grande objectivo da *API OpenSocial* é tornar simples a criação de aplicações multi plataformas *web* e que esteja acessível a qualquer programador que tenha conhecimentos em *HTML* e *JavaSript*.

Para aplicações pequenas ou que não necessitem de contínua manutenção, a *API OpenSocial* é suficiente, pois muito rapidamente se consegue desenvolver uma aplicação sem uma grande curva de aprendizagem.

No entanto, para quem enverede num projecto de tamanho médio/grande e tenha intenções de conseguir manter o projecto, são necessárias mais funcionalidades que *JavaSript* e *HTML* por si só não disponibilizam.

No contexto de quem usa a *framework symfony*, poder desenvolver aplicações usando o mesmo *IDE* que usava e tirar proveito das funcionalidades do mesmo, é o ideal.

Por isso mesmo, o *sfOpenSocialPlugin* permite integrar-se na plataforma *symfony*, seguindo os mesmos padrões de configuração, e encapsula toda a *API JavaSript* em *PHP5*, tornando assim fácil efectuar *debug* e consequentemente permitindo desenvolver aplicações mais seguras e expansíveis.

3.1.3 Arquitectura

Este *plugin* implementa a *API OpenSocial* e por essa razão é necessário entender minimamente o funcionamento da mesma para se poder usar de forma consistente este *plugin*.

O *Opensocial* é uma extensão da *Gadgets API* (que também é desenvolvida pela *Google*) que consiste especificamente num ficheiro *XML* que é responsável por fornecer toda a informação ao *container* para que este seja capaz de gerar a aplicação.

O *plugin* desenvolvido permite criar aplicações *OpenSocial* simulando o uso do modelo *MVC* do *symfony* (ver secção 2.2.4 "*MVC no symfony*" página 9).

A ideia central deste *plugin* é usar apenas uma página de entrada e depois tirar proveito dos *components* do *symfony* para integrar as funcionalidades necessárias para cada aplicação.

Assim sendo, podemos desenhar a nossa página *frontend* da aplicação que por sua vez é responsável por incluir cada componente que achemos necessário.

Esta forma de actuar impulsiona e aumenta a capacidade de reutilização de *components*.

3.1.3.1 Camada de controlo e camada de apresentação

O *sfOpenSocialPlugin* não faz uma rígida separação de camadas e essa separação fica ao critério do programador de acordo com as suas necessidades.

De qualquer forma é aconselhado fazer a separação de *Pedidos* feitos ao *container* (ver secção 2.3.1.3.1 "*Fazer pedidos ao container*", página 21) e *Pedidos* externos (ver secção 2.3.1.3.3 "*Efectuar pedido a servidor remoto*", página 21), das *Respostas* obtidas a esses mesmos *Pedidos*.

Fazer uma boa separação entre os *Pedidos* e *Respostas* é boa prática para se conseguir uma aplicação escalável e aumentar a produtividade em casos de aplicações médias/grandes.

Esta separação é também boa para que se possa optimizar os recursos humanos em determinado projecto, pois podemos delegar os *Pedidos* a especialistas em *OpenSocial*, deixando as *Respostas* a cargo de responsáveis pelo *design* da aplicação.

3.1.3.2 Estrutura de ficheiros

Directoria	Descrição
..	Encontram-se os ficheiros *README* e *LICENSE*
data/tasks	Pasta que contém a *Pake Task* para gerar aplicações automaticamente
lib/	Onde se encontra todo o código fonte usado no *plugin*
misc/	Aqui encontram-se ficheiros que são usados aquando da geração de aplicações
modules/	Aqui podemos encontrar a aplicação exemplo disponível neste *plugin*

Tabela 3.1: Estrutura da raiz dos ficheiros no sfOpenSocialPlugin

Os ficheiros estão organizados em pastas de acordo com as suas funcionalidades. Quando o *plugin* é instalado em determinado projecto o programador apenas precisa de olhar para a pasta "*lib/*" que é onde se encontra realmente a implementação do *plugin*.

Nas secções seguintes serão descritas cada uma dessas pastas.

3.1.4 Biblioteca sfOpenSocialPlugin

A *API* do *sfOpenSocialPlugin* pode ser encontrada na pasta "*sfOpenSocial/lib/*". Nas secções seguintes vai ser descrita a biblioteca disponível no sfOpenSocialPlugin e em anexo (na secção "*Código fonte do sfOpenSocialPlugin*") encontram-se os cabeçalhos do código fonte de cada ficheiro aqui falado.

3.1.4.1 Classes base

3.1.4.1.1 OSConfig

Esta classe é responsável por fornecer os métodos para lidar com as configurações da aplicação *OpenSocial*.

Esta classe está intimamente ligada ao ficheiro "*app.yml*" e permite definir todas as características do *Gadget* tais como:

- Lidar e gerar os atributos dos *ModulePrefs*
- Lidar e gerar os *UserPref*
- Importar automaticamente as "*feature-specific*" no *Gadget*

Todas estas funcionalidades podem ser definidas no ficheiro "*app.yml*", mas a classe *OSConfig* permite alterar estas configurações em *Runtime*.

3.1.4.1.2 JSFunction

Esta classe representa uma função em *JavaSript[18][19]* que pode ser programada. É amplamente usada neste *plugin*, principalmente para gerar o código *JavaSript* dos *Requests* (pedidos) e das *Responses* (as respostas aos pedidos).

Com a classe *JSFunction*, é possível definir-se:

- o **nome** da função *JavaSript*;
- os **argumentos** ou parâmetros da função;
- o **conteúdo** da função *JavaSript*.

Possui ainda métodos estáticos que permitem gerar automaticamente as funções *JavaSript* que podem ser usadas directamente dentro dos *templates*.

3.1.4.1.3 StringUtil

Esta é uma classe que é usada internamente em quase todas as classes da biblioteca *OpenSocial* que necessitam de gerar conteúdo.

Possui funções estáticas para manipular *Strings*.

3.1.4.1.4 oSFilter

Este é o filtro que é responsável por gerar o conteúdo do *Gadget*. Este filtro é executado antes do conteúdo gerado pela *Action* seja mostrado ao utilizador.

3.1.4.2 OpenSocial namespace

3.1.4.2.1 Person

Na pasta "*lib/opensocial/person/*" encontram-se as classes que são responsáveis por representar *pessoas* na *API OpenSocial* (ver secção 2.3.1.2.1 "*Pessoas*", página 19).

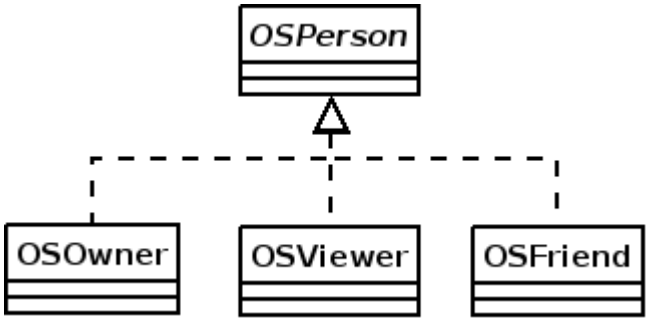


Figura 3.1: Modelo de classes da classe OSPerson

A classe *OSPerson* é uma classe abstracta, a qual implementa todos os métodos existentes na *API OpenSocial* v0.7.

Esta classe não pode ser instanciada, e as classes que o fazem são:

- ***OSOwner*** : representa uma pessoa sendo *OWNER* da aplicação
- ***OSViewer*** : representa uma pessoa como *VIEWER* em determinada aplicação
- ***OSFriend*** : quando se faz um pedido ao *container* pelos amigos de determinado *VIEWER* ou *OWNER*, este retorna uma determinada pessoa. A classe *OSFriend* serve para lidar com as pessoas retornadas pelo *container*.

3.1.4.2.2 Activity

A pasta "*lib/opensocial/activity/*" fornece as classes para lidar com as Actividades em determinado *container* (ver secção 2.3.1.2.3 "*Actividades*", na página 19).

As classes disponíveis nesta pasta são as seguintes:

Nome da classe	**Descrição**
OSActivity.**class**	Esta classe representa uma actividade
OSActivityPriority.**class**	Esta classe é transparente para o programador, mas serve para definir a prioridade das actividades
OSActivityField.**class**	Esta é uma classe que é transparente para o programador e é usada pela *OSActivity*.**class**
OSActivityMediaItem.**class**	Esta classe corresponde a um *MediaItem*
OSActivityMediaItemType.**class**	Esta classe define os tipos de *MediaItems* existentes, mas é transparente para o programador
JSFunctionErrorCallback.**class**	Esta é a função *JavaSript* criada por omissão quando é criada uma actividade. É transparente para o programador

Tabela 3.2: Classes para lidar com Actividades

A classe *OSActivity* representa uma actividade e sempre que seja necessário criar uma actividade é esta a classe que deve ser instanciada.

As actividades não implicam que seja realizado qualquer *DataRequest* ao *container* e por isso podem ser feitas a qualquer altura da aplicação.

Visto que não é necessário efectuar nenhum *request* específico, uma actividade tanto pode ser criada na *Action* como directamente na *View*.

Na *action* é possível criar-se um objecto do tipo *OSActivity* com o nome da variável como argumento (este nome de variável é necessário ser passado por parâmetro por forma a podermos definir vários tipos de actividades na mesma aplicação).

A classe *Activity* da *API OpenSocial v0.7* tem a possibilidade de definir *templates* para as actividades que são criadas, no entanto a classe *OSActivity* possui também um modo programático que permite definir um padrão para cada variável específica:

```
//Action
$this->mediaItem = new OSActivityMediaItem('logo');
$this->mediaItem->setImageItem();
$this->mediaItem-
>setURL('http://www.palcoprincipal.com/images/logo_opensocial_pp.png');

$this->activity->addMediaItem($this->mediaItem);
$this->activity->setPattern('title','O utilizador %s actualizou a musica %s
às %s!');
//View
function <?php $activity->getJSFunction->getName()?>() {
    <?php $activity->setTitle(sprintf($activity-
>getPattern('title'),'eu','lah lah','aa'));
          $activity->createActivity(false);
          echo $activity;

    ?>
  };
```

Listagem 3.1: Como usar o *setPattern* da classe *OSActivity*

Neste exemplo foi definido um padrão na própria actividade ainda na *Action*. Uma vez na *View* acedeu-se à actividade posteriormente criada na *Action*, e usando a função *sprintf* do *PHP[28]*, definiu-se o parâmetro *title* com os novos argumentos.

A função *setTitle()* também poderia ter sido chamada directamente a partir da *Action*, mas na action não tinha ainda acesso a variáveis obtidos pelo *container* (exemplo: *VIEWER*).

Quando se chama o método de instância *OSActivity::createActivity()*, é gerado todo o código *JavaSript* para criar esta actividade.

Quando é criado um objecto do tipo *OSActivity* é-lhe adicionado um *callback* para receber os erros que o *container* possa enviar. Isto é feito de forma transparente para o programador pois é criado no método construtor da classe *OSActivity* um objecto do tipo *JSFunctionErrorCallback* que é adicionado à actividade.

A classe *JSFunctionErrorCallback* é imprimida na aplicação final após ser chamado *OSActivity::createActivity* de forma transparente para o programador.

Opcionalmente o programador pode substituir a *callback* por uma sua ou até mesmo desactiva-la completamente.

3.1.4.2.3 DataRequest

A pasta "*lib/opensocial/datarequest/*" fornece as classes que permitem efectuar *DataRequests* a determinado *container* (ver secção 2.3.1.3.1 "*Fazer pedidos ao container*", na página 21).

A classe que realiza os *requests* chama-se *OSDataRequest*. Esta classe permite que lhe sejam adicionados vários tipos de *requests* antes de submeter os pedidos ao *conteiner*, e, para isso usa o método de instância *OSDataRequest::addRequest(OSDataRequestable)*.

Todas as classes que são adicionadas ao *OSDataRequest* têm de implementar a interface *OSDataRequestable*.

Por omissão existem cinco tipos de *requests* que podem ser adicionados ao *OSDataRequest*, antes deste ser encaminho ao *container*.

As cinco classes podem ser vistas na Figura 3.2:

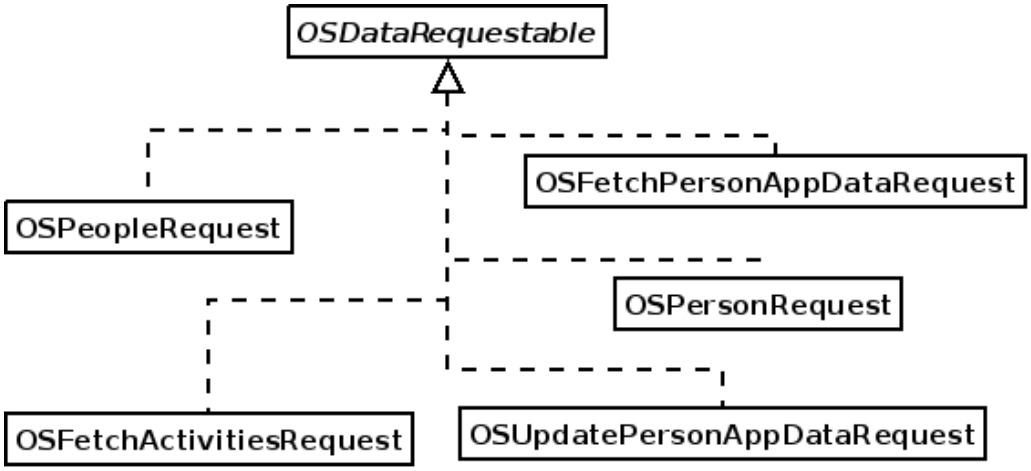


Figura 3.2: As quatro classes que implementam OSDataRequestable

Quando o *request* é submetido ao *container*, esse mesmo pedido só pode ser feito usando uma função *JavaSript* e caso seja necessário interpretar a resposta vinda do *container*, uma outra função *JavaSript* tem de existir para lidar com essa resposta (*callback*).

Essa função *JavaSript* de retorno está encapsulada na classe *OSDataResponse*, que para além de possuir a função *JavaSript* de retorno, disponibiliza também métodos para lidar com os dados recebidos do *container*.

Na Listagem 3.2 temos um exemplo de como realizar um *OSDataRequest* assumindo que estamos dentro de uma *Action*:

```
// Action
public function executeDataRequest()
{
  $this->viewer_friends = new OSViewerFriends('viewer_friends');

  $this->people_request = new OSPeopleRequest($this->viewer_friends);
  $this->people_request->addProfileDetails(OSPersonField::PROFILE_URL);
```

```
  $data_request = new OSDataRequest('requestInfo');
  $data_request->addRequest($this->people_request);

  $this->data_response = $data_request->send();
}
```

Listagem 3.2: Como realizar um *OSDataRequest* a partir de uma *Action*

Neste exemplo, inicialmente criamos um objecto do tipo *OSViwerFriends* que é passado como argumento para o *OSPeopleRequest*. Este *OSPeopleRequest* é um objecto que representa um pedido ao *container* do tipo *PeopleRequest*.

De seguida é criado um objecto do tipo *OSDataRequest* que recebe como argumento o seu identificador. Este identificador deve ser único na aplicação por forma a garantir que não há conflito de nomes aquando da geração da aplicação.

Na realidade este identificador serve para criar a função *JavaSript* de *request* e *response*, onde "***<identificador>**_request*" e "***<identificador>**_response*" correspondem aos nomes das funções *JavaSript*.

Posteriormente adicionamos o *OSPeopleRequest* ao *OSDataRequest* e executamos o método *OSDataRequest::send()*. Por forma a gerarmos um *DataResponse* é necessário chamar o método *OSDataRequest::send()* que vai gerar o código *JavaSript* do *request* que será automaticamente incluído na *view*.

O método *OSDataRequest::send()* retorna um objecto da classe *OSDataResponse*, objecto esse que será usado na *view*.

Este *DataResponse* vai ser inserido na *view* automaticamente, mas caso chamássemos antes *OSDataRequest::send(**false**)*, o código não seria injectado na *view* e poderíamos depois fazê-lo manualmente.

Na Listagem 3.3 podemos ver como poderíamos lidar com a resposta do *container* usando o objecto *OSDataResponse*:

```
// View
<div id="viewer_friends"> </div>

<?php echo JSFunction::addJSOpenTag($data_response->getJSFunction()); ?>
  var html = '';
  <?php echo $data_response->getData($people_request); ?>

  <?php $person = new OSFriend('person'); ?>
  <?php echo $viewer_friends->getVarName() ?>.each(function(<?php echo
$person->getVarName() ?>) {
    var thumb = <?php echo $person-
>getField(OSPersonField::THUMBNAIL_URL); ?>
    var profile = <?php echo $person-
>getField(OSPersonField::PROFILE_URL); ?>
    html += '<a href="' + profile + '" target="_top" style="float:left">'
+'<img src="' + thumb + '"/></a>';
    document.getElementById('viewer_friends').innerHTML = html;
  });
<?php echo JSFunction::addJSCloseTag($data_response->getJSFunction()) ?>
```

Listagem 3.3: Como lidar com o *OSDataResponse* na *View*

3.1.4.2.3.1 OSPeopleRequest

A classe *OSPeopleRequest* permite obter do *container* grupos de utilizadores. Um grupo de utilizadores é encapsulado na classe abstracta *OSFriends*.

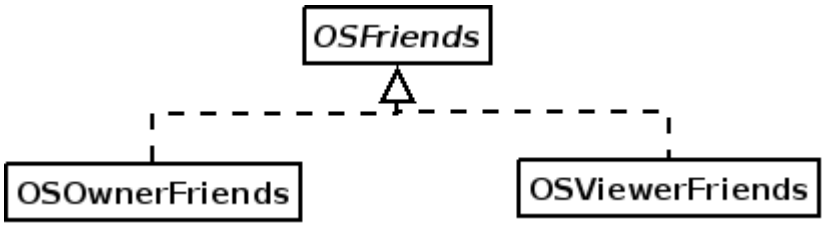


Figura 3.3: Classes que implementam a classe abstracta OSFriends

Como se pode ver na Figura 3.3 existem duas classes que implementam a classe abstracta *OSFriends* que são:

- ***OSOwnerFriends*** : que permite retornar o grupo de amigos do *OWNER*
- ***OSViewerFriends*** : que permite retornar o grupo de amigos do *VIEWER*

Aquando da criação de um objecto do tipo *OSFriends*, é necessário passar um *ID* (único no contexto onde seja usado este objecto) que irá servir para gerar o nome das variáveis.

Se voltarmos à Listagem 3.2 podemos verificar que foi criado um objecto do tipo *OSViewerFriends* ao qual passamos pelo método construtor o *ID* "*viewer_friends*". Ainda neste exemplo, foi efectuado um pedido ao *container* do tipo *OSPeopleRequest*.

3.1.4.2.3.2 OSPersonRequest

O *OSPersonRequest* é em tudo idêntico ao *OSPeopleRequest*, com a diferença que em vez de retornar um grupo de pessoas, retorna apenas uma pessoa.

Uma pessoa é encapsulada na classe abstracta *OSPerson* (ver secção 3.1.4.2.1 "*Person*", na página 30).

Na Listagem 3.4 podemos ver um exemplo de como fazer um *request* na *Action*, do *VIEWER* e do *OWNER*, usando o *OSPersonRequest*:

```
//action
$this->viewer = new OSViewer('viewer');
$this->owner = new OSOwner('owner');

$this->viewer_request = new OSPersonRequest($this->viewer);
$this->viewer_request->addProfileDetails(OSPersonField::PROFILE_URL);

$this->owner_request = new OSPersonRequest($this->owner);
$this->owner_request->addProfileDetails(OSPersonField::PROFILE_URL);

$data_request = new OSDataRequest('requestInfo');
$data_request->addPersonRequest($this->viewer_request);
$data_request->addPersonRequest($this->owner_request);

$this->data_response = $data_request->send();
```

Listagem 3.4: Como executar o *OSPersonRequest* na *Action*

3.1.4.2.3.3 OSFetchActivitiesRequest

A classe *OSFetchActivitiesRequest* permite obter do *container* actividades referentes ao *VIEWER*, *OWNER*, *VIEWER_FRIENDS* ou *OWNER_FRIENDS*.

Para se escolher de quem se pretende obter as Actividades, basta passar para método construtor da classe *OSFetchActivitiesRequest* um objecto que implemente *OSPerson* ou *OSFriends*.

3.1.4.2.3.4 OSUpdatePersonAppDataRequest

Para se guardar ou actualizar dados que estejam na camada de persistência do *container* (ver secção 2.3.1.2.4 "*Persistência*", página 20), deve-se usar a classe *OSUpdatePersonAppDataRequest*.

A classe *OSPersonAppData* é responsável por guardar os dados que serão escritos no *container*. Ao criar-se um objecto deste tipo, tem de se passar pelo método construtor um objecto da classe *OSViewer*, uma "chave" para identificar esta informação que se está a guardar e um valor para essa mesma chave.

Por sua vez a classe *OSFetchPersonAppDataRequest*, possui o método de instância *OSFetchPersonAppDataRequest::addPersonAppData()* que permite adicionar objectos *OSPersonAppData* que posteriormente serão escritos na camada de persistência do *container*.

3.1.4.2.3.5 OSFetchPersonAppDataRequest

Para se obter dados que estão guardados no *container* usando a camada de persistência (ver secção 2.3.1.2.4 "*Persistência*", página 20) é necessário fazer-se um *request* do tipo *OSFetchPersonAppDataRequest*.

Usando o método *OSFetchPersonAppDataRequest::fetchPersonAppData(OSPersonAppData)*, podemos pedir ao *container* os dados que pretendemos obter do mesmo.

3.1.4.2.4 Environment

A pasta "*lib/opensocial/environment/*" fornece as classes que permitem obter informação sobre o *container* onde a aplicação se encontra.

As classes disponíveis nesta pasta são as seguintes:

Nome da classe	Descrição
OSEnvironment.**class**	Esta é a classe que permite obter informações sobre o *container*
OSEnvironmentObjectType.**static.class**	Esta é uma classe transparente ao programador, pois é usada internamente pelo *OSEnvironment* e fornece um "enum" dos tipos de objectos de ambiente existentes no *container*.

Tabela 3.3: Classes para obter informações sobre o container

No exemplo abaixo estamos a obter o nome do *container* onde se encontra instalada a aplicação:

```
$env = new OSEnvironment('env');
$env->getDomainName();
```

Listagem 3.5: Como obter o nome do *container* usando a classe *OSEnvironment*

3.1.4.2.5 Message

A pasta "*lib/opensocial/message/*" fornece as classes que permitem enviar mensagens para determinados utilizadores de um *container*.

Existem quatro tipos de mensagens que podem ser enviadas:

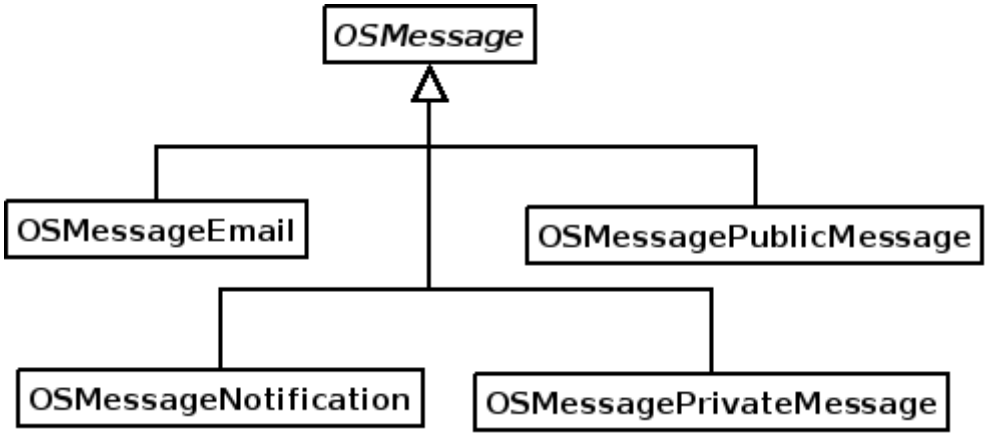


Figura 3.4: Classes que estendem OSMessage

As características de cada uma das classes é descrita na tabela seguinte:

Nome da classe	**Descrição**
OSMessageEmail.**class**	Esta classe permite enviar *emails* para determinados utilizadores de um *container*
OSMessageNotification.**class**	Permite enviar notificações para utilizadores do *container*
OSMessagePublicMessage.**class**	Permite enviar mensagens públicas para utilizadores do *container*
OSMessagePrivateMessage.**class**	Permite enviar mensagens privadas para utilizadores do *container*

Tabela 3.4: Classes para obter informações sobre o container

3.1.4.3 Gadgets namespace

3.1.4.3.1 GadgetRequest

A classe *GadgetRequest* permite efectuar pedidos a servidores remotos (ver secção 2.3.1.3.3 "*Efectuar pedido a servidor remoto*",página 21).

Existem mais quatro classes que estendem *GadgetRequest*, e são elas:

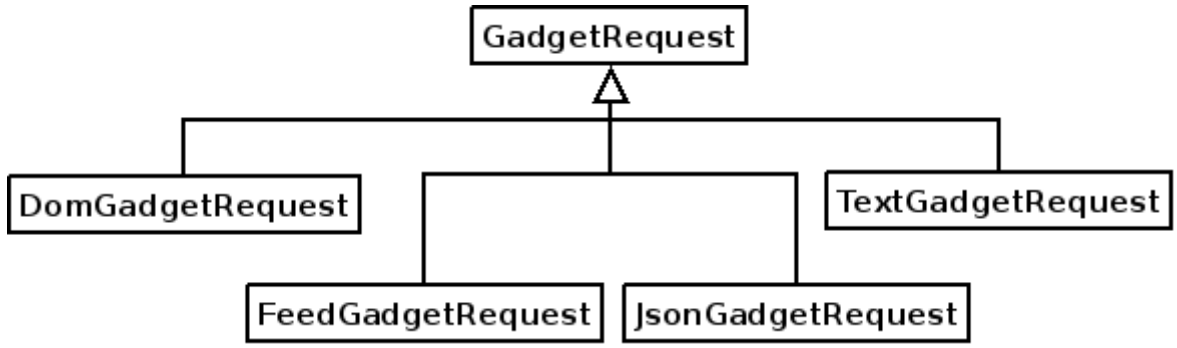


Figura 3.5: Classes que estendem GadgetRequest

A única diferença entre elas, é que cada uma é específica para um determinado tipo de conteúdo que se pretenda lidar.

Na Tabela 3.5 são mostradas as diferenças:

Nome da classe	**Descrição**
DomGadgetRequest.**class**	Esta classe é usada para lidar com *XML* e retorna um objecto *DOM*.
FeedGadgetRequest.**class**	Retorna a representação de uma *Feed* em formato *JSON*
JsonGadgetRequest.**class**	Retorna um objecto *JSON*
TextGadgetRequest.**class**	Usada para retornar texto simples.

Tabela 3.5: Classes que estendem GadgetRequest

3.1.4.3.2 GadgetsIO

Esta é a classe que torna possível obter conteúdo remoto usando a classe *GadgetRequest*.

Após termos definido qual vai ser o tipo de conteúdo que se vai obter do acesso remoto, é necessário efectuar realmente este pedido. É aí que entre a classe *GadgetsIO*, que serve exactamente para a efectuar essa chamada.

Abaixo fica um exemplo de como usar o *GadgetsIO*:

```
// action
$feed_request = new
FeedGadgetRequest('feedRequest','http://www.nytimes.com/services/xml/rss/nyt
/HomePage.xml');

$this->gresponse = GadgetsIO->makeRequest($feed_request);
// view
<?php echo JSFunction::addJSOpenTag($gresponse->getJSFunction()); ?>
  var text = <?php echo $gresponse->getText(); ?>
  document.getElementById('feed').innerHTML = text;
<?php echo JSFunction::addJSCloseTag($gresponse->getJSFunction()) ?>
```

Listagem 3.6: Como usar o *GadgetsIO*

Neste exemplo, podemos ver que primeiro foi criado um objecto do tipo *FeedGadgetRequest* na *Action*, passando-lhe como argumentos do construtor, o seu *identificador* e o *URL* do *feed*.

Após termos este objecto criado, chamámos então, o método de classe *GadgetsIO::makeRequest()*, por forma a que este gerasse automaticamente o código *JavaSript* deste pedido.

Á semelhança do método *DataRequest::send()* (ver secção 3.1.4.2.3 "*DataRequest*", página 32), o método

de classe *GadgetsIO::makeRequest()* retorna também uma classe de resposta, dada pelo nome *GadgetResponse* que é responsável por permitir o acesso aos dados retornados pelo servidor remoto.

3.1.4.3.3 GadgetsUtil

Esta é uma classe que disponibiliza métodos estáticos que ajudam a lidar com os *Gadgets*.

3.1.4.3.4 GadgetsPrefs

Esta é uma classe que incorpora todas as funcionalidades que a classe *Gadgets.Prefs* da *API OpenSocial v0.7*. Esta classe pode ser inicializada ainda na *action*, mas é na *view* que deve ser usada.

É necessário criar um objecto do tipo *GadgetsPrefs* e passar-lhe por argumento o nome que vai ser usado como variável na *view*.

Deve-se ter também a atenção de inicializar primeiro o objecto, ou chamando o método *initialize()* ou então forçar imprimir o objecto (tirando proveito do *__toString()*), por forma a que a variável interna *JavaSript* seja inicializada.

Após a inicialização do objecto pode-se usar normalmente as suas funcionalidades na *view* (ter em atenção que é necessário forçar a impressão do que é retornado por cada função). Aqui fica um exemplo:

```
// action
$this->gadget = new GadgetsPrefs('gadget');
// view
<?php echo $gadget->initialize();?>

function getLang()
{
  document.getElementById('getLang').innerHTML = <?php echo $gadget-
>getLang() ?>
}
function getCountry()
{
  document.getElementById('getCountry').innerHTML = <?php echo $gadget-
>getCountry() ?>
}
function set()
{
  var key = document.getElementById('set_key').value;
  var val = document.getElementById('set_val').value;
  <?php echo $gadget->set('key','val') ?>
}
function getString()
{
  var key = document.getElementById('getString_key').value;
  document.getElementById('getString').innerHTML = <?php echo $gadget-
>getString('key') ?>
}
...
```

Listagem 3.7: Como executar o *GadgetsPrefs*

3.1.4.3.5 ModulePrefs

Esta classe é responsável por permitir definir os <*ModulePrefs*> de uma aplicação *OpenSocial*, programaticamente.

Os <*ModulePrefs*> definidos usando esta classe, têm precedência sobre as configurações definidas no ficheiro "*app.yml*".

3.1.4.3.6 GadgetsJson

Esta classe define operações estáticas para fazer a tradução de e para objectos *JSON*.

3.1.4.4 Gadgets (feature-specific)

Para se poder usar *Gadgets feature-specific*, é necessário incluir no "<*ModulePrefs*>" o elemento "<*Require feature="nome_da_feature"/*>".

Uma das vantagens de usar o *sfOpenSocialPlugin* é que não é necessária a inserção manual no ModulePrefs do mesmo.

Na realidade o programador nem tem de saber que está a usar uma "*feature-specific*", pois isso é realizado automaticamente pelo *plugin* caso seja necessário incluir determinada *feature*.

3.1.4.4.1 GadgetsFlash

A classe *GadgetsFlash* permite incluir um objecto *Flash* na aplicação. O *sfOpenSocialPlugin* define uma classe chamada *FlashObject*, que permite definir todas as características do objecto *Flash* a incluir.

Uma vez tendo o objecto *FlashObject*, podemos então usar o método de classe *GadgetsFlash::embedFlash()* para adicionar o objecto à aplicação.

Abaixo fica um exemplo de como embeber um objecto *Flash* numa aplicação:

```
// Action
public function executeFlash()
{
  $palcoPrincipalFlash = new
FlashObject("http://www.palcoprincipal.com/widget/playerUsers/user/derbouka/
estilo/1");
  $palcoPrincipalFlash->addParam('id',"mp3player");
  $palcoPrincipalFlash->addParam('width',"200.3");
  $palcoPrincipalFlash->addParam('height',"196.9");
  $palcoPrincipalFlash->addParam('align',"center");
  $palcoPrincipalFlash->addParam('allowscriptaccess',"sameDomain");
  $palcoPrincipalFlash->addParam('wmode',"transparent");
  $palcoPrincipalFlash->addParam('quality',"high");

  $this->loadFlashFunction = new JSFunction('loadFlash');
  $flashLoad_variable = new JSVariable('flash_div',"'flash_load_div'");
  $this->loadFlashFunction->addVariable($flashLoad_variable);

  $this->loadFlashFunction-
```

```
>setContent(GadgetsFlash::embedFlash($flashLoad_variable,
$palcoPrincipalFlash));
      echo JSFunction::addJSTags($this->loadFlashFunction);
}

// View
<div id="flash_load_div"></div>
```

Listagem 3.8: Como usar a classe GadgetsFlash para carregar um objecto flash na Action

Como podemos ver neste exemplo, inicialmente criamos e configuramos o objecto *FlashObject*. Foi posteriormente criado um objecto do tipo *JSFunction*, que será a função *JavaSript* que vai ser responsável por executar o *Flash*.

Definimos também uma *JSVariable* que é a variável que vai indicar qual é a *DIV* que vai ser actualizado quando a *JSFunction* for executada.

Por fim, executamos o método de classe *GadgetsFlash::embedFlash* e o conteúdo gerado foi adicionado como sendo o conteúdo da *JSFunction* criada.

Uma vez gerada a função *JavaSript* que vai carregar o objecto *Flash*, na *view* apenas precisamos de ter uma *DIV* com o *ID* igual ao definido na *JSVariable*.

3.1.4.4.2 GadgetsMiniMessage

A classe *GadgetsMiniMessage* permite mostrar pequenas mensagens ao *VIEWER* da aplicação.

As *GadgetsMiniMessage*'s são mensagens que são adicionadas à aplicação em *Runtime* e estas podem posteriormente ser removidas programaticamente, ou auto-removíveis ou pode ser definido um determinado tempo até que a mensagem desapareça automaticamente (*dismissible)*.

Abaixo será exemplificado como usar a classe *GadgetsMiniMessage* no *sfOpenSocialPlugin*.

Visto as *GadgetsMiniMessages* não necessitarem de obter qualquer resposta por parte do *container*, estas podem ser chamadas a partir da *view* ou da *action*.

Neste exemplo, vai ser criado um *GadgetsMiniMessage* na *action*, definindo-o como "*dismissible*":

```
// Action
$dismissibleMessage = new GadgetsMiniMessage('dismissibleMessage');
$this->dismissibleMessageFunction = new
JSFunction('dismissibleMessageFunction');

$content = $dismissibleMessage->initialize();
$content .= $dismissibleMessage->createDismissibleMessage('<div>You can
dismiss this message!</div>');
$this->dismissibleMessageFunction->setContent($content);
echo JSFunction::addJSTags($this->dismissibleMessageFunction);

// View
<?php use_helper('OpenSocial'); ?>

<?php echo add_jsfunction_anchor('Create Dismissible Message',
$dismissibleMessageFunction) ?>
```

Listagem 3.9: Como usar a classe *GadgetsMiniMessage*

Como podemos ver, criamos um objecto do tipo *GadgetsMiniMessage* e um *JSFunction* por forma a que

toda a lógica fosse criada na *Action*. A primeira coisa a fazer quando se usa um objecto da classe *GadgetsMiniMessage* é inicializa-lo usando o método de instância *GadgetsMiniMessage::initialize()*.

Isto é necessário, porque a classe *GadgetsMiniMessage* implementa a interface *Instanciatable*, o que significa que se se quiser usar esta classe é necessário primeiro usar o método *Instanciatable::initialize()*. Caso não se inicialize esta classe antes de chamar qualquer outro método, é lançada uma *sfException*.

Na variável "*$content*" estamos a guardar o conteúdo da criação deste *GadgetsMiniMessage*, que posteriormente é definido como o conteúdo da *JSFunction* criada.

Uma vez na *View* é usada a função "*add_jsfunction_anchor()*" do *Helper Opensocial* (ver secção 3.1.4.5 "*OpenSocialHelper*", página 43), para criar um link para a *JSFunction* criada na *Action*.

3.1.4.4.3 GadgetsTab

Tabs são bastante úteis para organizar informação. O *sfOpenSocialPlugin* usa a classe *GadgetsTabSet* para permitir:

- Definir o alinhamento de tabs: direita, esquerda ou centro
- Mostrar ou esconder todas as tabs
- Adicionar ou remover tabs programaticamente
- Seleccionar determinada tab programaticamente
- Trocar posições entre tabs

A classe *GadgetsTabSet* implementa a interface *Instanciatable*, o que significa que é necessário chamar o método *Instanciatable::initialize()* antes de se poder usar qualquer outro método da classe *GadgetsTabSet*.

Abaixo fica um pequeno exemplo de como criar duas *GadgetsTabs* e adiciona-las ao *GadgetsTabSet*.

```
// Action
public function executeTabs2()
{
  $this->tabset = new GadgetsTabSet('tabset','tabs_div');
}

// View
<div id="tabs_div"></div>
<div id="tab1">Hi!</div>
<div id="tab2">Hey!</div>

<script type="text/JavaSript">

  <?php
    echo $tabset->getVarName().' = '.$tabset->initialize(false);
  ?>

  function onLoadTab()
  {
  <?php $selectedTab = $tabset->getSelectedTab();  ?>
  <?php echo $selectedTab->getTabContentContainer(true) ?>.innerHTML =
'selected tabName: '+<?php echo $selectedTab->getTabName(true); ?>+'
selected index:'+<?php echo $selectedTab->getTabIndex(true) ?>+' selected
name container:'+<?php echo $selectedTab->getTabNameContainer(true) ?>+'
```

```
selected callback:'+<?php echo $selectedTab->getTabCallback(true) ?>+'
selected content container:'+<?php echo $selectedTab-
>getTabContentContainer(true) ?>;
  }

  function init()
  {
  <?php
    $tab1 = new GadgetsTab("tab1");
    $tab1->setTooltip('My tab1');
    $tab1->setContentContainer(null);
    $tab1->setCallback(new JSFunction('onLoadTab'));
    echo $tabset->addTab($tab1);


    $tab2 = new GadgetsTab("tab2");
    $tab2->setTooltip('My tab2');
    $tab2->setContentContainer(null);
    $tab2->setCallback(new JSFunction('onLoadTab'));
    echo $tabset->addTab($tab2);
  ?>
  }
  <?php echo GadgetsUtil::registerOnLoadHandler(new JSFunction('init')); ?>
</script>
```

Listagem 3.10: Como usar a classe *GadgetsTabSet*

Começamos por criar um objecto do tipo *GadgetsTabSet* na *Action*, passando pelo método construtor o seu identificador único e o nome da *DIV*, que será a raiz de todas as *tabs* criadas.

Uma vez na *View*, foram criadas três *DIVs*:

- ***tabs_div*** : que é *DIV* onde vão estar todas as tabs
- ***tab1*** : que é a *DIV* onde vai ser escrito o conteúdo da *tab1*
- ***tabs2*** : que é a *DIV* onde vai ser escrito o conteúdo da *tab2*

Começamos então por inicializar o objecto *GadgetsTabSet* e é na função *JavaSript* "*init()*" onde são criadas cada uma das *GadgetsTab*'s e adicionadas ao *GadgetsTabSet* existente.

A função *JavaSript* "*onLoadTab()*" é a função que será chamada sempre que dada *GadgetsTab* seja seleccionada.

3.1.4.4.4 GadgetsViews

A classe *GadgetsViews* é a classe que permite lidar com as *Vistas* (ver secção 2.3.1.2.5 "*Vistas*", página 20). A classe *GadgetsView*, representa uma vista específica.

Abaixo fica um exemplo de como saber qual é a *Vista* em que se encontra a aplicação e como mostrar conteúdo de acordo com essa vista:

```
// View
<?php $currentView = GadgetsViews::getCurrentView(true); ?>

if(<?php echo $currentView->getName(true) ?> == "preview"){
  // mostra conteúdo para preview
}else if(<?php echo $currentView->getName(true) ?> == "canvas"){
  // mostra conteúdo para canvas
```

```
}else if(<?php echo $currentView->getName(true) ?> == "profile"){
  // mostra conteúdo para profile
}
```

Listagem 3.11: Como usar a classe *GadgetsViews*

3.1.4.4.5 GadgetsSkins

As aplicações *OpenSocial* podem-se "misturar" nas *Skins* onde se encontra instalada a aplicação.

No *sfOpenSocialPlugin* a classe *GadgetsSkins* é a classe responsável por aplicar a *Skin* usada no container, na aplicação.

Para isso, basta chamar dentro da *View*, o método de classe *GadgetsSkins::loadSkin()* e este aplica a *Skin* automaticamente à aplicação.

3.1.4.4.6 GadgetsWindow

A classe *GadgetsWindow* permite ao programador aceder às características da janela onde se encontra instalada a aplicação.

Esta é uma classe constituída apenas por métodos de classe e os mesmos são descritos na seguinte tabela:

Métodos estáticos	**Descrição**
adjustHeight	Permite ajustar a janela ao conteúdo da aplicação
setTitle	Permite definir o título da aplicação
getViewportDimensions	Permite saber quais são as dimensões da janela

Tabela 3.6: Métodos de classe disponíveis na classe GadgetsWindow

3.1.4.5 OpenSocialHelper

O *sfOpenSocialPlugin* disponibiliza um *Helper* (para saber como funcionam os *Helpers*, ver secção 2.2.4.2.1 "*Helpers*", página 10) chamado *OpenSocialHelper*.

O *OpenSocialHelper* existe para facilitar o desenvolvimento no *sfOpenSocialPlugin*. Com este *helper*, pode-se incluir secções de conteúdo no *Gadget*, ficheiros *JavaSript* e *CSS[29]*.

3.1.4.5.1 Incluir secção de conteúdo

O seguinte método permite incluir uma determinada secção de conteúdo num *Gadget*:

```
function include_content($content,$type=null,$href=null,$cdata=null){}
```

Listagem 3.12: Método para incluir secção de conteúdo usando o *OpenSocialHelper*

3.1.4.5.2 Incluir estilos

O *OpenSocialHelper* inclui dois métodos para incluir estilos numa aplicação:

```
function include_css($stylesheet){}
function include_remote_css($url,$nl=true){}
```

Listagem 3.13: Métodos para incluir estilos usando o *OpenSocialHelper*

O método "*include_css()*" vai incluir o ficheiro "*$stylesheet*", que se encontra na pasta "*web/css/*" na raiz do projecto;

O método "*include_remote_css()*" simplesmente inclui o "*$url*" que é passado como argumento.

3.1.4.5.3 Incluir JavaSript

O *OpenSocialHelper* inclui dois métodos para incluir *JavaSript* numa aplicação:

```
function include_js($js){}
function include_remote_js($url,$nl=true){}
```

Listagem 3.14: Métodos para incluir *JavaSript* usando o *OpenSocialHelper*

O método "*include_js()*" vai incluir o ficheiro "*$js*", que se encontra na pasta "*web/js/*" na raiz do projecto;

O método "*include_remote_js()*" simplesmente inclui o "*$url*" que é passado como argumento.

3.1.5 Como instalar o *plugin* no *symfony*

O *sfOpenSocialPlugin* é instalado da mesma forma que é instalado qualquer outro *plugin* para *symfony* (ver secção 2.2.5.2 "*Como instalar um Plug-in*", página 12).

Para proceder à instalação do *plugin* basta executar esta linha na consola ou terminal:

```
symfony plugin-install http://plugins.symfony-project.com/sfOpenSocialPlugin
```

Listagem 3.15: Como instalar o *sfOpenSocialPlugin*

Ao executar este comando a versão mais recente do *plugin*, vai ser colocada na pasta "*plugins/*" do projecto.

Não esquecer que é necessário limpar a *cache* do *symfony* por forma a que os mecanismos de "*auto-load"* do *symfony* carreguem as novas bibliotecas.

Para actualizar o *plugin* ou desinstala-lo, basta substituir o "*plugin-install*" por "*plugin-update*" ou "*plugin-uninstall*" respectivamente.

3.1.5.1 sfPakeOpenSocial

O *sfOpenSocialPlugin*, possui uma "*Pake Task*" que se encontra no *sfPakeOpenSocial*, que permite criar uma aplicação OpenSocial *standard* num projecto existente.

Para se criar uma aplicação OpenSocial, basta executar a seguinte linha na consola ou terminal:

```
symfony init-opensocial-app <applicationName> <moduleName>
```

Listagem 3.16: Como executar o *init-opensocial-app* usando o *sfPakeOpenSocial*

Ao ser executado o comando *init-opensocial-app*, vai ser criado na aplicação **<*applicationName*>**, o módulo **<*moduleName*>** e vai ser instalado nesse módulo, o módulo que vem com o *sfOpenSocialPlugin* na pasta "*sfOpenSocialPlugin/modules/*".

Caso não exista a aplicação **<*applicationName*>** ou o módulo **<*moduleName*>**, o *sfPakeOpenSocial*

inicializa-os, sendo este também responsável por configurar este módulo para funcionar correctamente.

3.1.6 Como configurar

A primeira coisa a fazer quando se cria uma aplicação *OpenSocial* é configura-la. Nesta secção vai ser descrito como configurar o *Layout* de dada aplicação e como personalizar a mesma.

3.1.6.1 Definir o Layout da aplicação

Visto que uma aplicação *OpenSocial* estende um *Gadget*, é necessário dizer ao *symfony* para usar o "*gadget_layout.php*" que se encontra na pasta "*sfOpenSocialPlugin/misc/app/templates/*", como *layout* do *template* "*indexSuccess.php*" (ver secção 2.2.4.2.3 "*Configuração do Layout*", na página 11).

Para se fazer isto, basta alterar o ficheiro "*view.yml*" da seguinte forma:

```
indexSuccess:
  http_metas:
    content-type: application/xml

  layout: gadget_layout
```

Listagem 3.17: Alterar o ficheiro "*view.yml*" para lidar com o "*gadget_layout*"

Este passo apenas é necessário, caso não se tenha usado a "*Pake Task*" "*init-opensocial-app*", pois esta já configura o *layout* por omissão.

3.1.6.2 Personalizar a aplicação

Agora que o *layout* já se encontra configurado, é necessário personalizar a aplicação.

Existem duas formas de o fazer no *sfOpenSocialPlugin*:

- usando o ficheiro "*app.yml*"
- usando a classe *ModulePrefs* para o fazer programaticamente.

Personalizar a aplicação, significa atribuir pares **chave-valor**, que depois serão adicionados ao *Gadget* de forma transparente ao programador.

3.1.6.2.1 Configurar o *app.xml*

Este ficheiro vai ser usado pela classe *OSConfig* para gerar a estrutura do *Gadget* (ver secção 3.1.4.1.1 "*OSConfig*", página 29)

Abaixo encontra-se um exemplo de configuração do "*app.yml*":

```
# default values
all:
  opensocial:
    base_url: http://www.palcoprincipal.com
    api_version: 0.7
```

```
    module_prefs:
      title: My First OpenSocial Application
#      directory_title
#      title_url
      description: This application was created by sfOpenSocialPlugin
      author: Daniel Botelho
      author_email: botelho.daniel@gmail.com
#      author_affiliation
#      author_location
#      screenshot
      thumbnail: http://www.palcoprincipal.com/images/logos/logo_palco_3.jpg
#      height
#      width
#      scaling
#      scrolling
#      singleton
#      author_photo
#      author_aboutme
#      author_link
#      author_quote

    locale:
      - lang: en
        country: us
#        messages
#        language_direction

    requires:
      - feature: opensocial-0.7
#      - feature: dynamic-height
#      - feature: setprefs
#      - feature: settitle
#      - feature: tabs
#      - feature: drag
#      - feature: grid
#      - feature: minimessage
#      - feature: analytics
#      - feature: flash
#      - feature: finance


    user_prefs:
#      - name: difficulty # the name of the User Preference
#        display_name: asd
#        urlparam: asd
#        datatype: asd
#        required: asd
#        default_value: asd
#        enum_value:
#          - {value: enum1, display_value: xx }
#          - {value: enum2, display_value: xx}

    content:
      type: html
#      href
#      cdata
```

Listagem 3.18: Exemplo de configuração do "*app.yml*"

3.1.6.2.1.1 all:opensocial:base_url

Este "***base_url***" é o endereço web onde se encontra instalada a aplicação. Este endereço vai ser usado pelo *sfOpenSocialHelper* para incluir automaticamente os ficheiros de estilo e ficheiros *JavaSript* (ver secção secção 3.1.4.5 "*OpenSocialHelper*", página 43).

3.1.6.2.1.2 all:opensocial:module_prefs

Todos os pares **chave-valor** presentes dentro deste nível vão ser escritos no *Gadget* como atributos dentro da entidade <*ModulePrefs*>[30].

3.1.6.2.1.3 all:opensocial:locale

Locales podem ser usados nos *Gadgets* para identificar quais as línguas suportados por determinada aplicação. Estes podem conter uma colecção[31] de valores tal como se pode ver no ficheiro *YAML* na Figura 3.18.

3.1.6.2.1.4 all:opensocial:requires

Os pares presentes em "*requires*" são bibliotecas específicas de *JavaSript* que podem ser importadas por determinado *Gadget* (ver secção 3.1.4.4 "*Gadgets (feature-specific)* ", página 39).

Estas são representadas como uma colecção de valores.

3.1.6.2.1.5 all:opensocial:user_prefs

Os *UserPref* podem ser representados, tal como está descrito em *Userprefs_Ref*[32]. Todos os valores são opcionais e os "*enum_value*" têm a particularidade de poderem conter *enums*.

3.1.6.2.1.6 all:opensocial:content

Esta secção apenas existe para aplicações que não fazem a distinção entre *Vistas* (ver secção 2.3.1.2.5 "*Vistas*", página 20). Caso se pretenda usar vários secções de conteúdo[33], deve-se comentar o "***content***" e usar o *sfOpenSocialHelper* para incluir as secções de conteúdo desejadas (ver secção secção 3.1.4.5 "*OpenSocialHelper*", página 43).

3.2 Implementação da aplicação OpenSocial

Esta secção é dedicada a descrever a implementação da aplicação *OpenSocial* desenvolvida para o Palco Principal.

3.2.1 Enquadramento e objectivo da aplicação

Esta aplicação visa permitir ao Palco Principal conseguir uma maior divulgação e exposição do seu site, de uma forma simples e quase sem custos para a mesma.

O Palco Principal é um site de música que assenta no paradigma de *Web2.0*, por isso os seus utilizadores são os principais responsáveis por gerar conteúdos e fazer o site crescer.

A vinda do *OpenSocial* e a promessa de aplicações multi-container que assentam numa *API* comum, vai de encontro às necessidades e ao público alvo do Palco Principal.

3.2.2 Arquitectura

A aplicação *OpenSocial* foi desenvolvida usando a *framework symfony* e utilizando o *sfOpenSocialPlugin* (ver secção 3.1 "*Implementação do sfOpenSocialPlugin*").

3.2.2.1 Estrutura de ficheiros

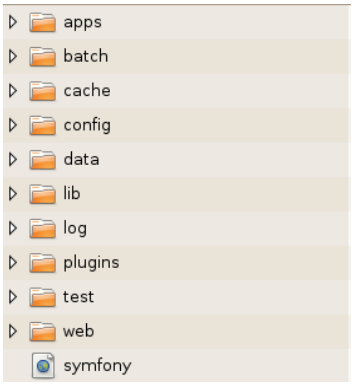


Figura 3.6: Raiz da aplicação

A estrutura de ficheiros da aplicação, segue a estrutura de ficheiros do *symfony*, onde em "*apps/*" vai estar a aplicação do Palco Principal; em "*config/*" vão estar as configurações da Base de Dados; em "*lib/*" o modelo da Base de Dados; em "*plugins/*" o "sfOpenSocialPlugin"; e finalmente em "*web/*" os ficheiros *CSS*, *JavaSript* e de Localização, que vão estar disponíveis para a aplicação.

O nome dado à aplicação foi "*palcoprincipal*" e pode ser encontrada em "*apps/palcoprincipal/*".

Esta é a base da aplicação e é aqui que se encontra o módulo que lida com a aplicação ("*apps/palcoprincipal/modules/palco/*") e o módulo que lida com a API ("*apps/palcoprincipal/api/*").

A raiz da estrutura da aplicação "*palcoprincipal*" pode ser vista na Figura 3.7.

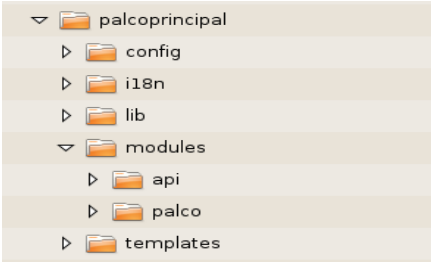


Figura 3.7: Raiz de "apps/palcoprincipal"

- **palcoprincipal/config/**

É na pasta "*config/*" onde estão as configurações base da aplicação. Para além dos ficheiros gerados automaticamente pelo *symfony* as únicas alterações que existem são, a inclusão do ficheiro "*app.yml*" que contém as definições do *Gadget*; e o "*filters.yml*", que indica quais os filtros a serem usados em cada módulo.

- **palcoprincipal/lib/**

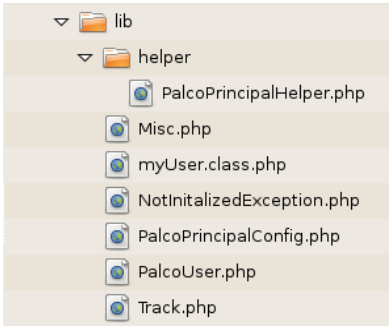


Figura 3.8: Estrutura da "lib"

A pasta "*lib/*" contém classes extra usadas neste projecto, tais como:

- ***Misc**.php* : contém o código de inclusão de um *Helper* dentro de uma *Action*;
- ***PalcoPrincipalConfig**.php* : permite aceder a configurações específicas da aplicação;
- ***PalcoUser**.php* : é responsável por lidar com *Owners* que são utilizadores do Palco Principal;
- ***PalcoPrincipalHelper**.php* : contém *helpers* para gerar o código usado nos templates.

3.2.2.1.1 Módulo: API (apps/palcoprincipal/modules/api)

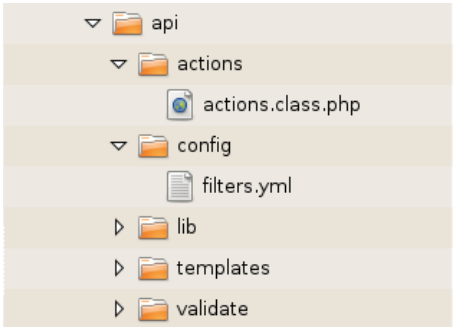


Figura 3.9: Estrutura do módulo API

Do módulo "*api*" apenas interessa saber do ficheiro "*actions.class.php*" que é onde se encontra a *API* implementada, e do ficheiro "*filters.yml*", que apenas indica que não são usados *templates* neste módulo.

3.2.2.1.2 Módulo: Palco (apps/palcoprincipal/modules/palco/)

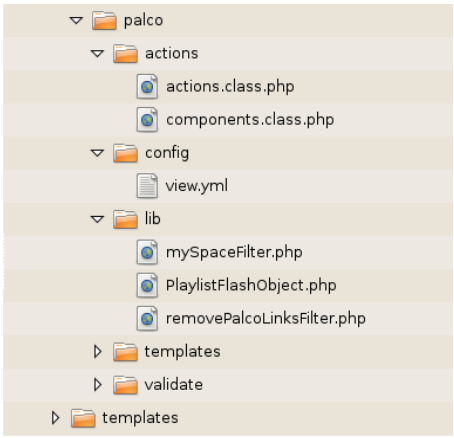


Figura 3.10: Estrutura do módulo Palco

Este módulo é responsável por gerar o conteúdo do ficheiro *Gadget* que é a aplicação do Palco Principal.

Aqui podemos identificar as seguintes pastas como sendo as mais importantes:

- **palco/actions/** : que é onde se encontram as *actions* e *actions* dos *components*;
- **palco/config/** : onde se encontra o ficheiro "*view.yml*" que é responsável por definir o *layout*;
- **palco/lib/ :** que é onde se encontram as classes específicas usadas neste módulo;
- **palco/templates/ :** onde se encontram definidos todos os *templates* incluindo os componentes.

Em anexo (na secção "Código fonte da aplicação OpenSocial do Palco Principal") encontram-se os cabeçalhos do código fonte de cada ficheiro aqui falado.

3.2.2.2 Modelo Cliente-Servidor

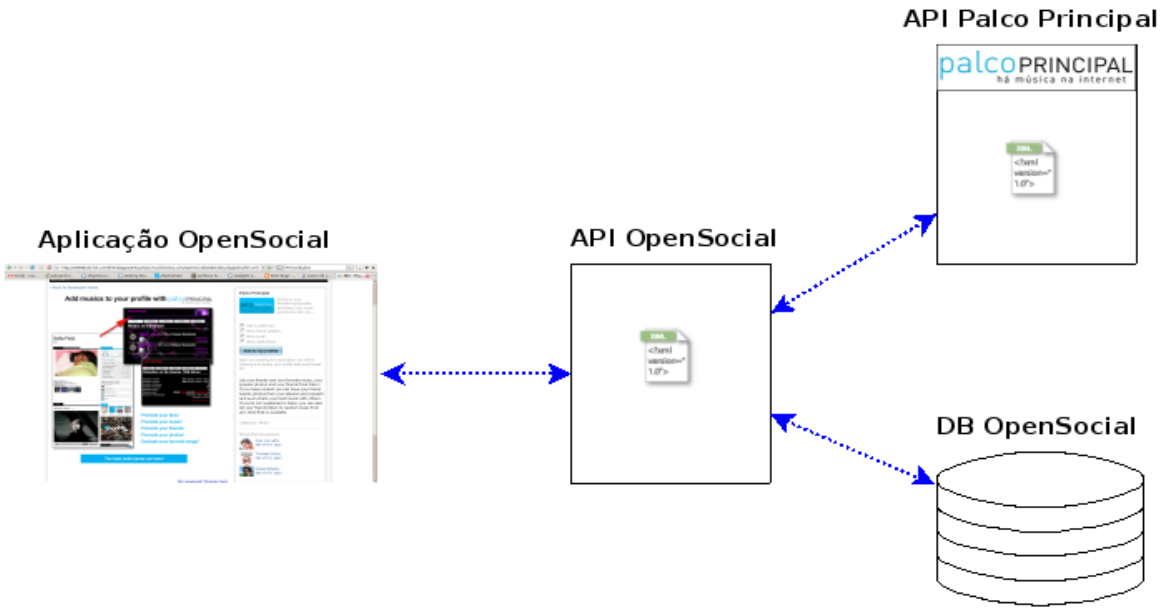


Figura 3.11: Modelo Cliente-Servidor

Esta aplicação assenta no modelo cliente servidor, onde o cliente é a aplicação instalada no *container* e o servidor é o módulo *API* desenvolvido que permite executar as operações requisitadas pela aplicação.

3.2.2.3 Usabilidade

A aplicação procura ser intuitiva, permitindo a um utilizador que esteja familiarizado com o *container* onde a aplicação se encontra instalada, seja capaz de facilmente conseguir realizar todas as funcionalidades disponíveis na mesma.

3.2.2.4 Aparência

Esta aplicação consegue adaptar a sua aparência com a aparência do *container* onde se encontra. Caso o utilizador que tenha a aplicação instalada use uma *Skin*, a aplicação vai usar o estilo dessa *Skin* (ver secção 3.1.4.4.5 "*GadgetsSkins*" página 43, para ver como funciona o mecanismo).

Figura 3.12: Aplicação usando a Skin Arcane

As *Skins* apenas estão disponíveis na vista *Profile* e na Figura 3.12 e Figura 3.13, encontram-se dois exemplos de como ver uma aplicação instalada utilizando duas *Skins* diferentes:

Figura 3.13: Aplicação usando a Skin Plasma Space

3.2.2.5 Segurança

Esta aplicação possui três níveis gerais de segurança:

- Proteger o núcleo do Palco Principal;
- Proteger a aplicação contra vulnerabilidades do tipo "*Cross-site Scripting*" (XSS)[34];

- Proteger a Base de Dados contra ataques do tipo "*SQL injection*".

A forma encontrada para proteger o núcleo do Palco Principal, foi desenvolver uma aplicação que funcionasse fora do "ambiente" do Palco Principal. Fora do "ambiente" deve ser entendido como, num servidor externo e sem usar a base de dados que o Palco Principal usa para gerar os conteúdos do seu site.

Para isso, foi desenvolvida uma *API* (ver secção 3.2.4 "*API da aplicação OpenSocial do Palco Principal*") em cima da *API RESTFul* que o Palco Principal está a desenvolver para permitir o acesso de aplicações externas (ver secção 2.1.3 "*API RESTFul Beta*", página 6).

Esta *API* desenvolvida permite independência entre a aplicação *OpenSocial* e o site do Palco Principal, permitindo que esta funcione quase de forma autónoma.

Quando as aplicações permitem que utilizadores inseriram conteúdos de texto, é necessário usar mecanismos para garantir que não seja injectado código malicioso na aplicação.

Este tipo de vulnerabilidades têm a designação de "*Cross-site Scripting*" (XSS), e na aplicação *OpenSocial* desenvolvida essa situação poderia suceder, pois esta permite que utilizadores façam dedicatórias a outros utilizadores.

Aqui houve então o cuidado de se usar mecanismos tanto do lado da aplicação *OpenSocial* como do lado do servidor, para garantir que a aplicação fosse resistente a este tipo de ataques.

O facto da aplicação ter sido desenvolvida em cima da *framework symfony*, deu-lhe também a possibilidade de usar o *Propel* para para gerir a base de dados. O *Propel* protege a Base de dados contra "*SQL injection*", por isso não foi necessário usar mais mecanismos de segurança adicionais[35].

3.2.2.6 Internacionalização

Esta aplicação está disponível em duas línguas:

- **Português**
- **Inglês**

A internacionalização da aplicação foi feita usando os mecanismos disponíveis pelo *sfOpenSocialPlugin* (ver secção 3.1.6.2 "*Personalizar a aplicação*", página 45).

3.2.3 Modelo da base de dados

Tal como foi dito a aplicação usa uma *Base de Dados* própria, permitindo que esta consiga manter persistência entre as acções realizados pelos utilizadores da aplicação.

Nesta secção, vai ser descrita a estrutura da mesma e na Figura 3.14 podemos ver o seu modelo Entidade-Relação.

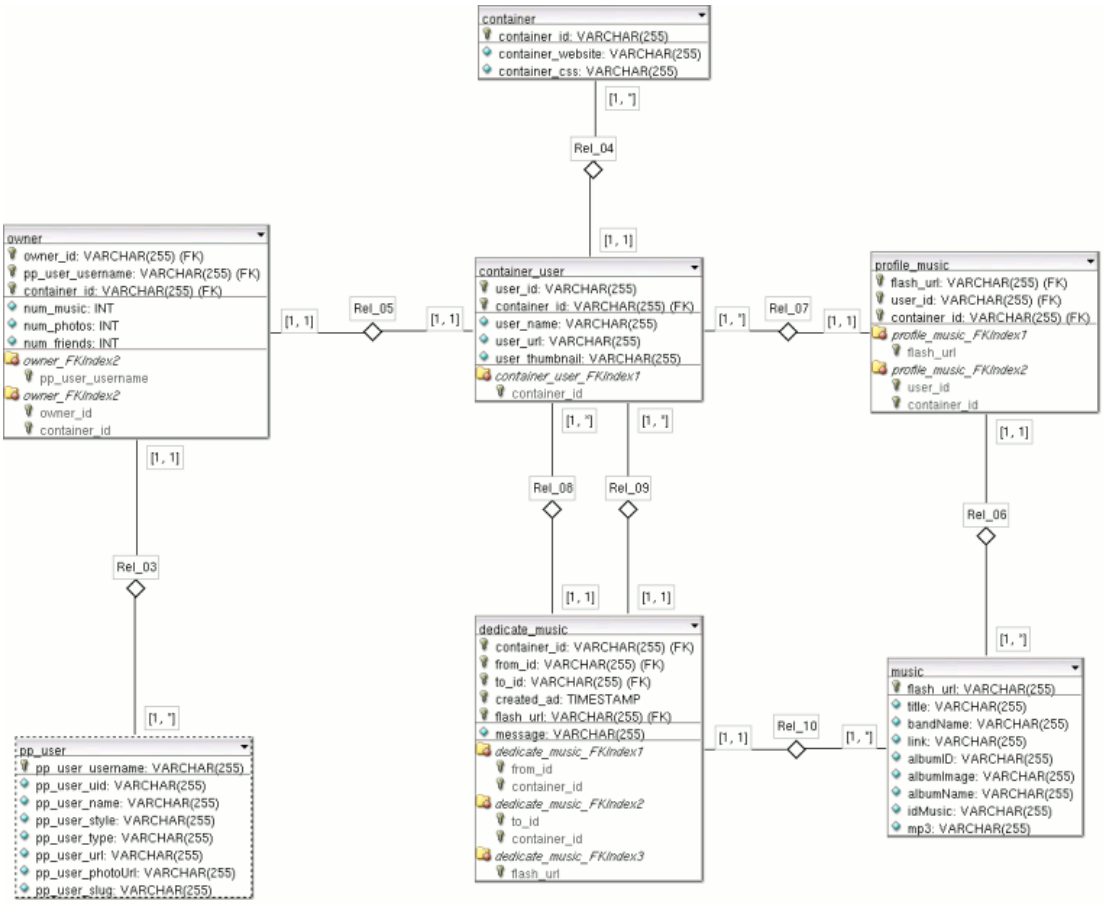


Figura 3.14: Modelo ER da base de dados

Código fonte para desenvolver esta Base de Dados encontra-se em anexo na Listagem 7.67.

3.2.3.1 Tabela: container

Figura 3.15: Tabela container

Esta tabela é responsável por guardar os dados referentes aos *containers* que usam a aplicação.

3.2.3.2 Tabela: container_user

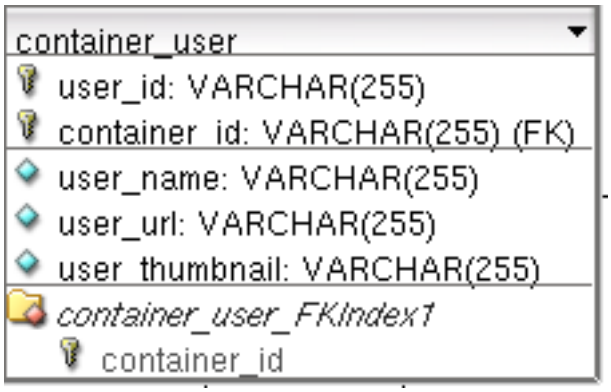


Figura 3.16: Tabela container_user

A tabela "***container_user***" é responsável por armazenar todos os utilizadores que estão em determinado *container*.

3.2.3.3 Tabela: owner

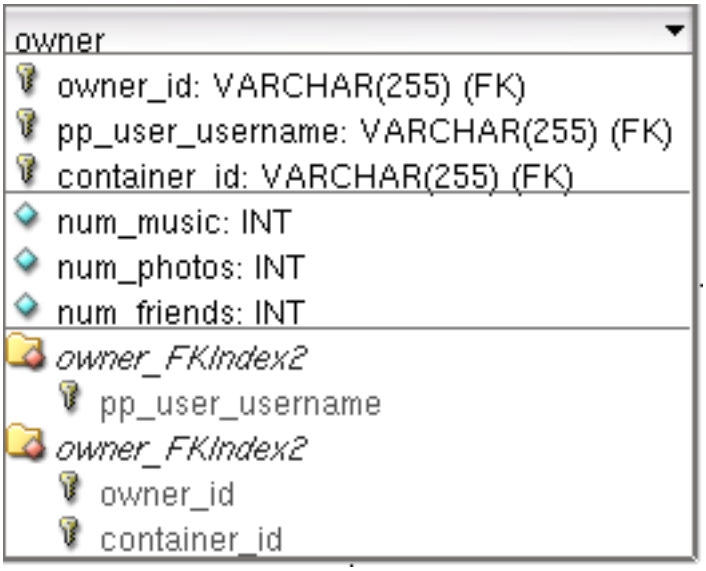


Figura 3.17: Tabela owner

Esta tabela guarda os dados de determinado *OWNER* que tem a aplicação instalada e está autenticado no Palco Principal.

3.2.3.4 Tabela: pp_user

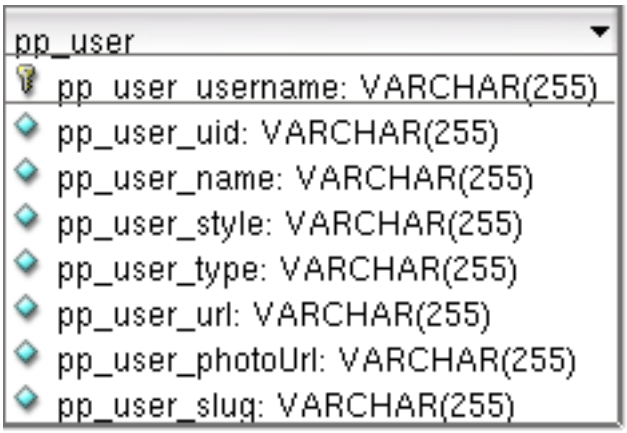


Figura 3.18: Tabela pp_user

Esta tabela armazena a informação de utilizadores do Palco Principal.

3.2.3.5 Tabela: music

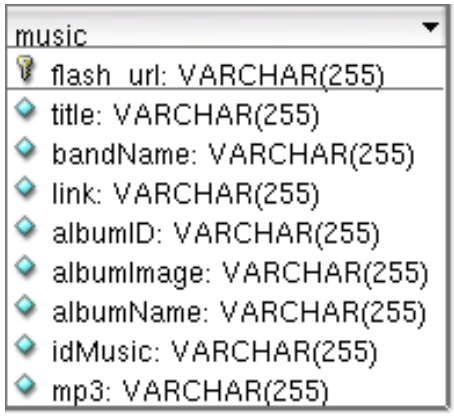


Figura 3.19: Tabela music

Esta tabela guarda a informação das músicas que já foram dedicadas ou adicionadas a perfis.

3.2.3.6 Tabela: profile_music

Figura 3.20: Tabela profile_music

Esta tabela representa as músicas que estão em determinado perfil.

3.2.3.7 Tabela: dedicate_music

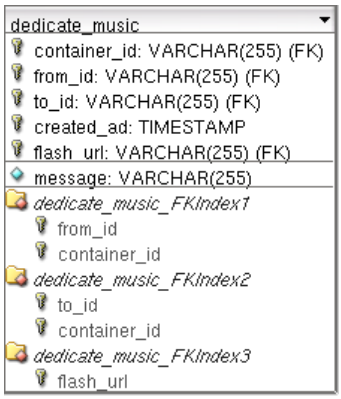


Figura 3.21: Tabela dedicate_music

A tabela "***dedicate_music***" é responsável por guardar as músicas dedicadas.

3.2.4 API da aplicação OpenSocial do Palco Principal

A aplicação *OpenSocial* do Palco Principal usa a sua própria Base de Dados e para isso foi necessário desenvolver uma API que permitisse à aplicação comunicar com essa mesma Base de Dados.

Nesta secção será descrita a *API* de comunicação entre a aplicação instalada no *container* e o servidor onde se encontra a Base de Dados.

Esta API disponibiliza comandos, que são endereços *URL*, e a possibilidade de se passar parâmetros, tanto por *GET* ou *POST*, por esses mesmos comandos.

Cada comando chamado, só tem duas respostas possíveis:

- uma mensagem de erro;
- ou o resultado esperado.

Nas subsecções seguintes serão descritos os comandos disponíveis nesta *API*.

3.2.4.1 Login

Este é primeiro comando que é chamado quando algum *VIEWER* acede a determinada aplicação instalada.

O *VIEWER* que acede à aplicação deve enviar tanto por *GET* ou *POST* o seu **nome**, **endereço do seu perfil** e o **thumbnail** que usa no *container*. É também necessário enviar o ***ID*** do *container* e do *OWNER* da aplicação.

Este método inicializa toda a informação que é necessária para o bom funcionamento da aplicação e retorna a informação do utilizador do Palco Principal a que o *OWNER* está associado.

Ver Tabela 7.5 que se encontra em anexo.

3.2.4.2 Register

Este comando é chamado quando determinado *OWNER* se pretende autenticar no Palco Principal.

Ver Tabela 7.6 que se encontra em anexo.

3.2.4.3 GetPlaylist

Este comando é chamado para retornar as músicas do Palco Principal de um *Utilizador* ou então as músicas de uma determinada *Banda*. A *API* identifica o utilizador do Palco Principal consultando o "***container_id***" e o "***owner_id***".

O atributo "***in_playlist***" pode possuir os valores:

- **1** Caso a música esteja no perfil do*VIEWER*
- **0** Caso a música não esteja no perfil do*VIEWER*
- **-1** Caso o*VIEWER* não tenha permissões para realizar acções na música

Ver Tabela 7.7 que se encontra em anexo.

3.2.4.4 GetFriends

Este comando retorna a lista de amigos de um utilizador, caso este possua uma conta de *Utilizador* no Palco Principal; ou então a lista de bandas amigas, caso a conta seja do tipo *Banda*.

Ver Tabela 7.8 que se encontra em anexo.

3.2.4.5 GetPhotos

Este comando retorna as fotos de um utilizador, caso este tenha uma conta de *Utilizador* no Palco Principal; ou então as fotos de uma banda, caso a conta seja do tipo *Banda*.

Ver Tabela 7.9 que se encontra em anexo.

3.2.4.6 GetAccountInfo

Este comando retorna as definições da aplicação instalada. Esta informação apenas é retornada, caso o *VIEWER* seja igual ao *OWNER* da aplicação.

Ver Tabela 7.10 que se encontra em anexo.

3.2.4.7 UpdateAccountInfo

Este comando permite alterar as configurações da aplicação instalada.

Ver Tabela 7.11 que se encontra em anexo.

3.2.4.8 GetBandFans

Este comando retorna os fãs de determinada *Banda* do Palco Principal.

Ver Tabela 7.12 que se encontra em anexo.

3.2.4.9 GetBandStats

Este comando retorna as estatísticas de determinada *Banda* do Palco Principal.

Ver Tabela 7.13 que se encontra em anexo.

3.2.4.10 AddProfileMusic

Este comando adiciona determinada música ao perfil de um *VIEWER*.

Ver Tabela 7.14 que se encontra em anexo.

3.2.4.11 DelProfileMusic

Este comando remove determinada música do perfil de um *OWNER*.

Ver Tabela 7.15 que se encontra em anexo.

3.2.4.12 GetProfileMusics

Este comando é chamado para retornar as músicas que estão no perfil de um *OWNER*. A *API* identifica o utilizador do Palco Principal consultando o "***container_id***" e o "***owner_id***".

O atributo "***in_playlist***" pode ter os seguintes valores:

- **1** Caso a música esteja no perfil do*VIEWER*
- **0** Caso a música não esteja no perfil do*VIEWER*
- **-1** Caso o*VIEWER* não tenha permissões para realizar acções na música

Ver Tabela 7.16 que se encontra em anexo.

3.2.4.13 AddDedicateMusic

Este comando permite dedicar determinada música a um dado amigo do *VIEWER*.

Ver Tabela 7.17 que se encontra em anexo.

3.2.4.14 DelDedicatedMusic

Este comando permite remover determinada música dedicada.

Ver Tabela 7.18 que se encontra em anexo.

3.2.4.15 GetDedicatedMusics

Este comando é chamado para retornar as músicas dedicadas tanto ao *VIEWER* como ao *OWNER*.

Ver Tabela 7.19 que se encontra em anexo.

3.2.4.16 GetPlaylistByStyle

Este comando retorna uma lista de músicas de determinado estilo musical.

Ver Tabela 7.20 que se encontra em anexo.

3.2.4.17 GetStyles

Este comando retorna uma lista de estilos musicais disponíveis.

Ver Tabela 7.21 que se encontra em anexo.

3.2.4.18 GetTopDedicatedMusicsByContainer

Este comando retorna as dez músicas mais dedicadas em determinado *container*.

Ver Tabela 7.22 que se encontra em anexo.

3.2.4.19 GetTopMusicsByContainer

Este comando retorna as dez músicas que estão em mais perfis de determinado *container*.

Ver Tabela 7.23 que se encontra em anexo.

3.2.4.20 GetContainerUsersByFlashURL

Este comando retorna os utilizadores de determinado *container* que têm determinada música no seu perfil.

Ver Tabela 7.24 que se encontra em anexo.

3.2.4.21 Códigos de erro

Todos os códigos de erro que podem ser retornados por esta *API* seguem o seguinte *template*:

```
<response>
  <error code="xxxx" msg="xxxx"></error>
</response>
```

Listagem 3.19: *Template* dos códigos de erro da *API OpenSocial* do Palco Principal

Abaixo ficam todos os códigos de erro disponíveis:

3.2.4.21.1 Erro 1

```
<response>
  <error code="1" msg="xxxx"></error>
</response>
```

Listagem 3.20: Formato do código de *erro 1*

Este erro é lançado quando faltam argumentos essenciais para a correcta execução determinado comando da *API*.

3.2.4.21.2 Erro 2

```
<response>
  <error code="2" msg="xxxx"></error>
</response>
```

Listagem 3.21: Formato do código de *erro 2*

Este erro é lançado quando não é possível aceder ao servidor do Palco Principal.

3.2.4.21.3 Erro 3

```
<response>
  <error code="3" msg="xxxx"></error>
</response>
```

Listagem 3.22: Formato do código de *erro 3*

Este erro é lançado quando o servidor do Palco Principal retorna um erro. Este erro vem em "*msg*."

3.2.4.21.4 Erro 4

```
<response>
  <error code="4" msg="xxxx"></error>
</response>
```

Listagem 3.23: Formato do código de *erro 4*

Este erro é lançado quando o *OWNER* já está autenticado no Palco Principal e tenta registar-se novamente.

3.2.4.21.5 Erro 5

```
<response>
  <error code="5" msg="xxxx"></error>
</response>
```

Listagem 3.24: Formato do código de erro 5

Este erro é lançado quando determinado *container* não tem autorização para usar a aplicação.

3.2.4.21.6 Erro 6

```
<response>
  <error code="6" msg="xxxx"></error>
</response>
```

Listagem 3.25: Formato do código de erro 6

Este erro é lançado quando determinado *container* não tem associado nenhum *CSS* específico.

3.2.4.21.7 Erro 7

```
<response>
  <error code="7" msg="xxxx"></error>
</response>
```

Listagem 3.26: Formato do código de erro 7

Este erro é lançado sempre que o *OWNER* da aplicação não está registado no Palco Principal.

3.2.4.21.8 Erro 8

```
<response>
  <error code="8" msg="xxxx"></error>
</response>
```

Listagem 3.27: Formato do código de erro 8

Este erro é lançado quando o *OWNER* da aplicação está registado no Palco Principal, mas ainda não activou a sua conta.

3.2.4.21.9 Erro 9

```
<response>
  <error code="9" msg="xxxx"></error>
</response>
```

Listagem 3.28: Formato do código de erro 9

Este erro é lançado quando determinada música já existe num perfil.

3.2.4.21.10 Erro 10

```
<response>
  <error code="10" msg="xxxx"></error>
</response>
```

Listagem 3.29: Formato do código de erro 10

Este erro é lançado quando determinada música não existe em determinado perfil.

3.2.4.21.11 Erro 11

```
<response>
  <error code="11" msg="xxxx"></error>
</response>
```

Listagem 3.30: Formato do código de erro 11

Este erro é lançado quando não existe determinada música dedicada.

3.2.4.21.12 Erro 12

```
<response>
  <error code="12" msg="xxxx"></error>
</response>
```

Listagem 3.31: Formato do código de erro 12

Este código de erro é lançado sempre que um *VIEWER* não tem permissões para executar determinado comando.

3.2.5 Funcionalidades

Esta aplicação foi desenvolvida de forma a tirar proveito das capacidades que o *OpenSocial* permite, mas também tirando proveito das funcionalidades que o Palco Principal fornece aos seus utilizadores.

A aplicação reage de forma diferente consoante o tipo de utilizador que está a usar a aplicação, e também consoante a Vista em que este se encontra.

A aplicação lida com os seguintes tipos de actores:

Actor 1 - *Utilizador não autenticado no Palco Principal* : Este é um utilizador que está autenticado no *container* onde a aplicação se encontra, mas que ainda não fez *login* no Palco Principal usando a aplicação;

Actor 2 - *Utilizador autenticado como Utilizador do Palco Principal* : Este é um utilizador que está autenticado na aplicação e tem uma conta de *Utilizador* no Palco Principal.

Actor 3 - ***Utilizador autenticado como Banda do Palco Principal*** : Este é um utilizador que está autenticado na aplicação e tem uma conta de *Banda* no Palco Principal.

***Actor 4 - Utilizador desconhecido* :** Este é um utilizador que não está autenticado no *container*, por isso não é reconhecido pela aplicação.

Tanto o ***Actor 1, Actor 2*** ou ***Actor 3***, podem dedicar músicas aos seus amigos no *container,* mas apenas o ***Actor 1*** e ***Actor 2***, podem adicionar músicas ao perfil da aplicação. Quanto a ouvir músicas, todos os actores, têm permissões para o fazer.

3.2.5.1 Casos de uso

Nesta secção serão descritos todos os casos de uso da aplicação. Para além disso serão também indicados em cada caso de uso quais são os comandos chamados da *API* para realizar esse caso de uso específico.

3.2.5.1.1 Ver pré-visualização da aplicação

Nome:	Ver pré-visualização da aplicação
Pré-condições:	*VIEWER* não tem a aplicação instalada
Vistas permitidas:	*Preview*
API:	-
Descrição:	Permite ao *VIEWER* ver a pré-visualização da aplicação

Tabela 3.7: Caso de uso: Ver pré-visualização da aplicação

Este caso de uso dá-se quando o utilizador que está a ver aplicação (*VIEWER*) não tem a aplicação

instalada e está na vista *Preview* da aplicação.

Nesta vista a aplicação não permite qualquer funcionalidade, mas apenas mostra uma imagem a descrever as funcionalidades da mesma.

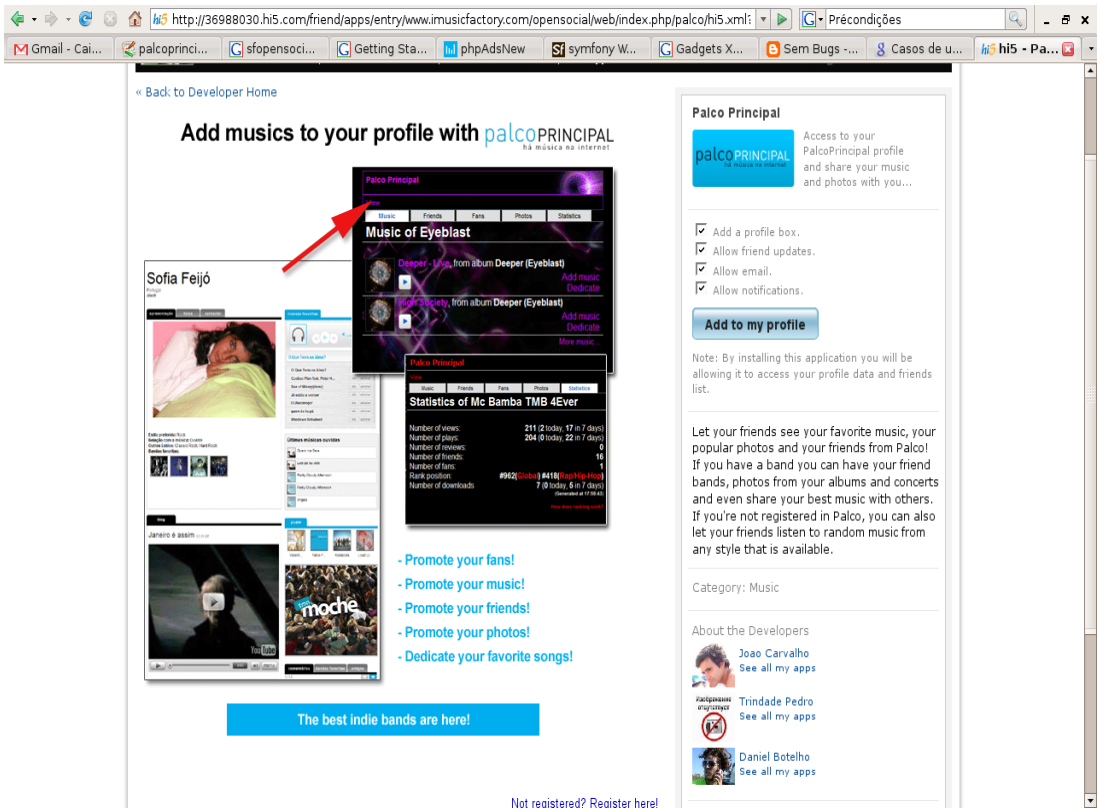


Figura 3.22: Aplicação na vista Preview

Esta imagem, tal como a aplicação, está internacionalizada tanto para **Português** como para **Inglês** e caso o utilizador queira experimentar a aplicação, basta usar o mecanismo do *container* para adicionar a aplicação ao seu perfil.

3.2.5.1.2 Ver Mais músicas

Nome:	Ver Mais músicas
Pré-condições:	*OWNER* tem a aplicação instalada e não está autenticado ou então está autenticado como *utilizador* no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetStyles API OpenSocial do Palco Principal: GetPlaylistByStyle API OpenSocial do Palco Principal: GetTopMusicsByContainer API OpenSocial do Palco Principal: GetTopDedicatedMusicsByContainer
Descrição:	Permite ver a secção mais músicas

Tabela 3.8: Caso de uso: Ver Mais musicas

Este caso de uso dá-se quando o utilizador que tem a aplicação instalada (*OWNER*), ou não está autenticado no Palco Principal ou está autenticado no Palco Principal como *utilizador*.

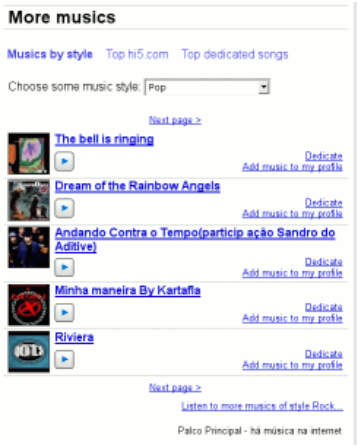


Figura 3.23: Ver secção Mais Músicas

Nestas condições quando alguém vir a aplicação (*VIEWER*), vai aparecer uma secção chamada "*Mais Músicas*" (se estiver em Português), a qual permite ver "*músicas por estilo*", o *TOP* de músicas instaladas em perfis no *container* e o *TOP* das músicas mais dedicados no *container*.

3.2.5.1.3 Adicionar músicas ao perfil

Nome:	Adicionar músicas ao perfil
Pré-condições:	*VIEWER* tem a aplicação instalada e não está autenticado como banda no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: AddProfileMusic
Descrição:	Permite a um *VIEWER* que tenha a aplicação instalada e não seja banda, adicionar músicas ao seu perfil

Tabela 3.9: Caso de uso: Adicionar músicas ao perfil

Para um utilizador poder adicionar músicas ao seu perfil, este tem que ter a aplicação instalada e não estar autenticado no Palco Principal como *Banda*.

Caso o utilizador esteja nestas condições, pode adicionar qualquer música que apareça nesta aplicação estando instalada no seu perfil, ou noutro qualquer perfil de utilizador no mesmo *container*.

Pode-se adicionar músicas quer se esteja na vista *Canvas* ou *Profile*, mas caso se esteja na vista *Canvas*, é adicionada uma nova actividade à lista de actividades do *VIEWER*, indicando que este adicionou a nova música.

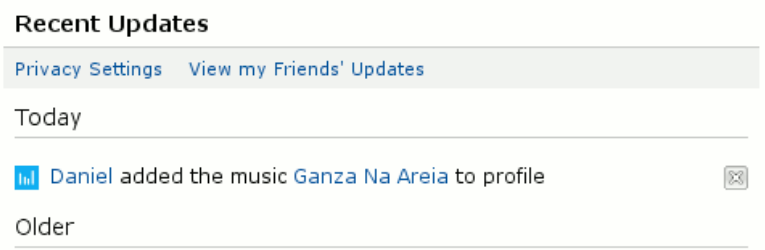


Figura 3.24: Actividade criada por adicionar música ao perfil

Na Figura 3.24 podemos ver um exemplo, de quando é adicionada uma música ao perfil.

3.2.5.1.4 Ver Músicas do perfil no container

Nome:	Ver Músicas do perfil no container
Pré-condições:	*OWNER* tem a aplicação instalada e não está autenticado como *banda* no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetProfileMusic
Descrição:	Permite ver as músicas favoritas de determinado *OWNER* com a aplicação instalada e que não está autenticado como *banda*

Tabela 3.10: Caso de uso: Ver Músicas do perfil no container

Este caso de uso dá-se quando o utilizador que tem a aplicação instalada (*OWNER*) não está autenticado no Palco Principal ou está autenticado como *utilizador*.

As "*músicas de perfil*" são músicas que são adicionadas usando o caso de uso: "*Adicionar músicas ao perfil* " (página 65).

Caso o *OWNER* não tenha ainda adicionado nenhuma música ao seu perfil, aparece uma mensagem de aviso. Esta mensagem de aviso é internacionalizada e faz a distinção caso quem esteja a ver a aplicação é o dono da aplicação ou não (*VIEWER != OWNER*).

Na Figura 3.25 podemos ver como seria vista a aplicação pelo *OWNER* no caso de não ter músicas no perfil e tendo a língua definida como **Inglês**:

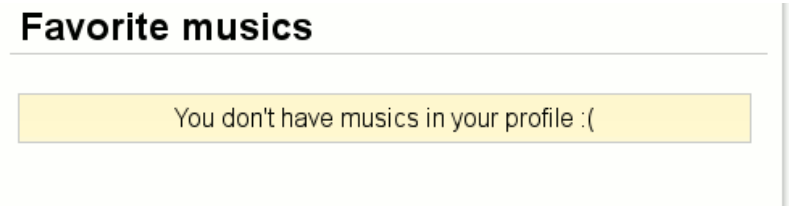


Figura 3.25: Aplicação sem músicas de perfil vista pelo OWNER

Na Figura 3.26 podemos ver como seria vista a aplicação por outro qualquer utilizador sem ser o *OWNER*:

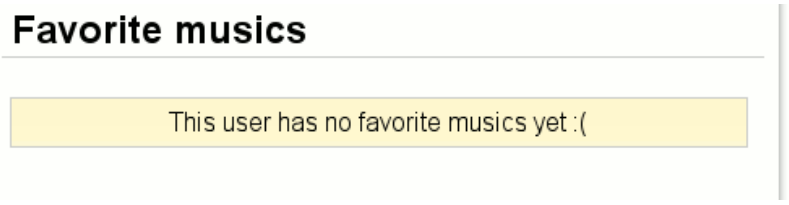


Figura 3.26: Aplicação sem músicas de perfil vista por outro qualquer utilizador

3.2.5.1.5 Ouvir musica

Nome:	Ouvir musica
Pré-condições:	*OWNER* tem a aplicação instalada
Vistas permitidas:	*Canvas, Profile*
API:	-
Descrição:	Permite ouvir qualquer música que esteja na aplicação

Tabela 3.11: Caso de uso: Ouvir musica

Para se usar este caso de uso, basta que se esteja a ver uma aplicação instalada.

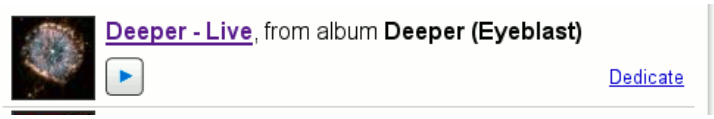


Figura 3.27: Como ouvir uma música

Para se ouvir a música basta clicar no botão de *play* e este será substituído por um objecto *Flash* que é responsável por tocar a música.

3.2.5.1.6 Dedicar musica

Nome:	Dedicar musica
Pré-condições:	*OWNER* tem a aplicação instalada e *VIEWER* está autenticado no container
Vistas permitidas:	*Canvas*
API:	API OpenSocial do Palco Principal: AddDedicateMusic
Descrição:	Permite dedicar músicas

Tabela 3.12: Caso de uso: Dedicar musica

Para se dedicar músicas, apenas é preciso que o utilizador que esta a ver a aplicação (*VIEWER*) esteja autenticado no *container*.

Figura 3.28: Dedicar música

Sempre que uma música é dedicada, é enviada uma notificação ao destinatário da dedicatória. Esta é a razão pela qual apenas em *Canvas* se podem fazer dedicatórias.

As dedicatórias apenas podem ser feitas aos amigos do *VIEWER* e estes não têm que ter a aplicação instalada.

3.2.5.1.7 Ver músicas dedicadas ao *OWNER*

Nome:	Ver músicas dedicadas ao *OWNER*
Pré-condições:	*OWNER* tem a aplicação instalada e não está autenticado ou então está autenticado como utilizador no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetDedicatedMusics
Descrição:	Permite ver as músicas dedicadas ao *OWNER*

Tabela 3.13: Caso de uso: Ver músicas dedicadas ao OWNER

Este caso de uso refere-se apenas a situações em que o *OWNER* ou não está autenticado no Palco Principal ou então está autenticado como *utilizador*.

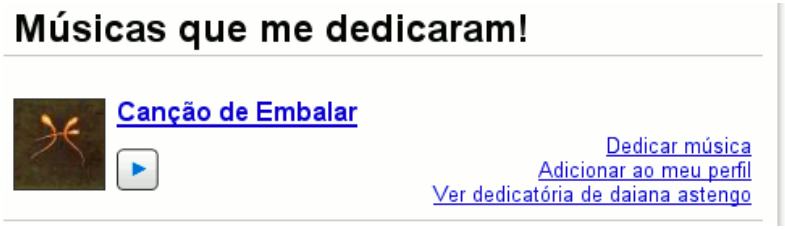


Figura 3.29: Ver músicas dedicadas ao OWNER

3.2.5.1.8 Ver Músicas dedicadas pelo OWNER

Nome:	Ver Músicas dedicadas pelo *OWNER*
Pré-condições:	*VIEWER!= OWNER* e *OWNER* tem a aplicação instalada e não está autenticado ou então está autenticado como utilizador no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetDedicatedMusics
Descrição:	Permite ao *VIEWER* ver as músicas que o *OWNER* lhe dedicou

Tabela 3.14: Caso de uso: Ver Músicas dedicadas pelo OWNER

Este caso de uso permite que um *VIEWER* veja as músicas que o *OWNER* da aplicação lhe dedicou.

3.2.5.1.9 Ver dedicatória

Nome:	Ver dedicatória
Pré-condições:	*OWNER* tem a aplicação instalada e não está autenticado ou então está autenticado como utilizador no Palco Principal, e tem pelo menos uma música dedicada
Vistas permitidas:	*Canvas*, *Profile*
API:	-
Descrição:	Permite ver as dedicatórias

Tabela 3.15: Caso de uso: Ver dedicatória

Figura 3.30: Ver dedicatória

Este caso de uso, permite ver a dedicatória feita a determinada música.

3.2.5.1.10 Rejeitar musica dedicada

Nome:	Rejeitar musica dedicada
Pré-condições:	*VIEWER* tem a aplicação instalada
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: DelDedicatedMusic
Descrição:	Permite rejeitar músicas dedicadas

Tabela 3.16: Caso de uso: Rejeitar musica dedicada

Este caso de uso permite ao *VIEWER* rejeitar determinada dedicatória caso este seja quem recebeu a mesma. A Figura 3.30 já engloba esta situação.

3.2.5.1.11 Efectuar login

Nome:	Efectuar login
Pré-condições:	*VIEWER* == *OWNER* e *OWNER* tem a aplicação instalada e não está autenticado no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetPlaylistByStyle
Descrição:	Permite ao utilizador fazer *login* no Palco Principal

Tabela 3.17: Caso de uso: Efectuar login

Um utilizador que tenha a aplicação instalada no seu perfil e não esteja autenticado no Palco Principal, pode a qualquer momento fazer *login* no Palco Principal e importar o seu perfil para a aplicação.

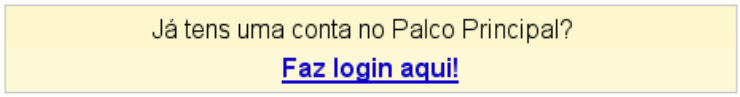


Figura 3.31: Fazer login

3.2.5.1.12 Configurar a aplicação

Nome:	Configurar a aplicação
Pré-condições:	*OWNER == VIEWER*, *OWNER* tem a aplicação instalada e está autenticado no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetAccountInfo API OpenSocial do Palco Principal: UpdateAccountInfo
Descrição:	Permite ao *OWNER* configurar a sua aplicação

Tabela 3.18: Caso de uso: Configurar a aplicação

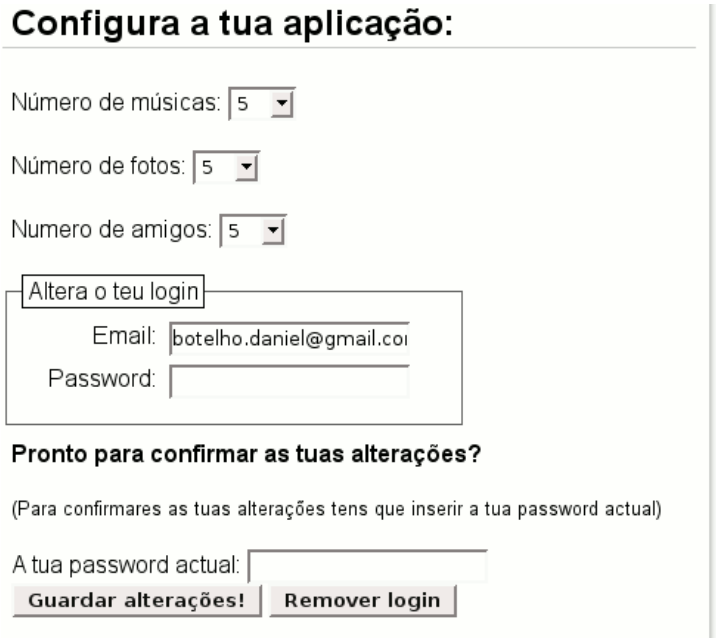


Figura 3.32: Configurar a aplicação

Este caso de uso apenas está disponível para utilizadores que têm a aplicação instalada e estão autenticados no Palco Principal.

Na Figura 3.32 temos todas as opções disponíveis.

3.2.5.1.13 Ver Musicas da Banda

Nome:	Ver Musicas da Banda
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como banda no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetPlaylist
Descrição:	Permite que alguém veja as músicas da Banda

Tabela 3.19: Caso de uso: Ver Musicas da Banda

Este caso de uso permite ver as músicas de determinada banda, caso o *OWNER* esteja autenticado como

Banda no Palco Principal.

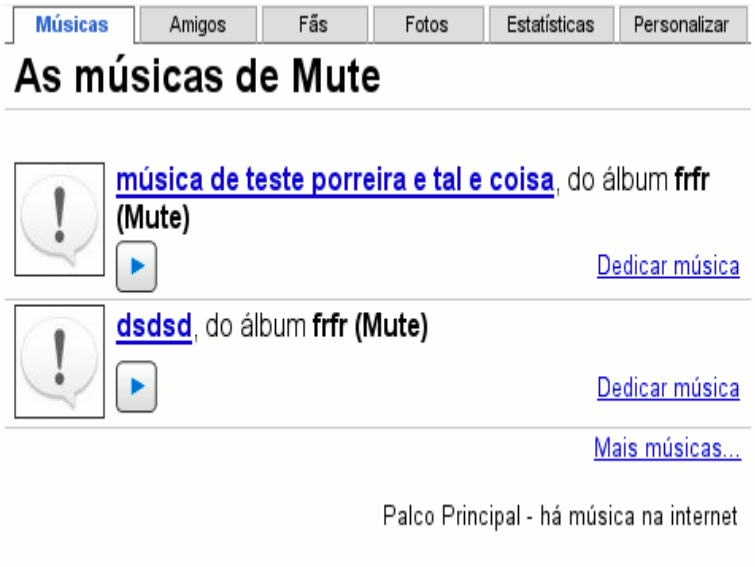


Figura 3.33: Ver músicas de uma Banda

3.2.5.1.14 Ver Amigos da Banda

Nome:	Ver Amigos da Banda
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como banda no Palco Principal
Vistas permitidas:	*Canvas*, *Profile*
API:	API OpenSocial do Palco Principal: GetFriends
Descrição:	Permite que alguém veja os Amigos da Banda

Tabela 3.20: Caso de uso: Ver Amigos da Banda

Figura 3.34: Ver Bandas amigas de determinada Banda

Este caso de uso permite ver as bandas amigas de determinada banda, caso o *OWNER* esteja autenticado como *Banda* no Palco Principal.

3.2.5.1.15 Ver Fãs da Banda

Nome:	Ver Fãs da Banda
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como banda no Palco Principal
Vistas permitidas:	*Canvas, Profile*
API:	API OpenSocial do Palco Principal: GetBandFans
Descrição:	Permite que alguém veja os Amigos da Banda

Tabela 3.21: Caso de uso: Ver Fãs da Banda

Este caso de uso permite ver os fãs de determinada banda, caso o *OWNER* esteja autenticado como *Banda* no Palco Principal.

Figura 3.35: Ver fãs de determinada banda

3.2.5.1.16 Ver Fotos da Banda

Nome:	Ver Fotos da Banda
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como banda no Palco Principal
Vistas permitidas:	*Canvas, Profile*
API:	API OpenSocial do Palco Principal: GetPhotos
Descrição:	Permite que alguém veja as Fotos da Banda

Tabela 3.22: Caso de uso: Ver Fotos da Banda

Este caso de uso permite ver as fotos de determinada banda, caso o *OWNER* esteja autenticado como *Banda* no Palco Principal.

Figura 3.36: Ver fotos de determinada Banda

3.2.5.1.17 Ver Estatísticas da Banda

Nome:	Ver Estatísticas da Banda
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como banda no Palco Principal
Vistas permitidas:	*Canvas, Profile*
API:	API OpenSocial do Palco Principal: GetBandStats
Descrição:	Permite que alguém veja as Estatísticas da Banda

Tabela 3.23: Caso de uso: Ver Estatísticas da Banda

Este caso de uso permite ver as estatísticas de determinada banda, caso o *OWNER* esteja autenticado como *Banda* no Palco Principal.

Figura 3.37: Ver estatísticas de determinada banda(no hi5)

3.2.5.1.18 Ver Musicas do Palco

Nome:	Ver Musicas do Palco
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como utilizador no Palco Principal
Vistas permitidas:	*Canvas, Profile*
API:	API OpenSocial do Palco Principal: GetPlaylist
Descrição:	Permite que alguém veja as Músicas no Palco Principal do utilizador

Tabela 3.24: Caso de uso: Ver Musicas do Palco

Este caso de uso permite ver as músicas de determinado utilizador, caso o *OWNER* esteja autenticado como *Utilizador* no Palco Principal.

Figura 3.38: Ver músicas de utilizador do Palco Principal

3.2.5.1.19 Ver Amigos do Palco

Nome:	Ver Amigos do Palco
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como utilizador no Palco Principal
Vistas permitidas:	*Canvas, Profile*
API:	API OpenSocial do Palco Principal: GetFriends
Descrição:	Permite que alguém veja os Amigos no Palco Principal do utilizador

Tabela 3.25: Caso de uso: Ver Amigos do Palco

Este caso de uso permite ver as Músicas de determinado utilizador, caso o *OWNER* esteja autenticado como *Utilizador* no Palco Principal.

Figura 3.39: Ver amigos de utilizador do Palco Principal

3.2.5.1.20 Ver Fotos do Palco

Nome:	Ver Fotos do Palco
Pré-condições:	*OWNER* tem a aplicação instalada e está autenticado como utilizador no Palco Principal
Vistas permitidas:	*Canvas, Profile*
API:	API OpenSocial do Palco Principal: GetPhotos
Descrição:	Permite que alguém veja as Fotos no Palco Principal do utilizador

Tabela 3.26: Caso de uso: Ver Fotos do Palco

Este caso de uso permite ver as Músicas de determinado utilizador, caso o *OWNER* esteja autenticado como *Utilizador* no Palco Principal.

Figura 3.40: Ver as fotos do utilizador do Palco Principal

Capítulo 4: Análise e validação dos resultados

4.1 sfOpenSocialPlugin

Tal como era proposto neste projecto, o *plugin* desenvolvido permite integrar o *OpenSocial* na plataforma de desenvolvimento *symfony*.

Este *plugin* está na base do desenvolvimento da aplicação *OpenSocial* do Palco Principal, e devido as características do mesmo, foi possível tornar a aplicação mais escalável e de configuração versátil.

A possibilidade de o *plugin* permitir usar o mesmo ficheiro de configurações para todas as aplicações e permitir a partir das *Actions* estender essas funcionalidades, permitiu a que a aplicação se tornasse mais robusta e permitiu também que a mesma aplicação pudesse ser adaptada a vários *containers* que possuíam implementações ligeiramente diferentes da *API OpenSocial*.

De qualquer forma não existe ainda *feedback* por parte da comunidade *symfony* em relação ao *sfOpenSocialPlugin*.

Enumeram-se de seguida algumas notícias em relação a este *plugin*:

- **(2008/03/23)** *Palco Principal @ OpenSocial*
  http://www.palcoprincipal.com/blog/2008/03/24/palco-principal-opensocial/
- **(2008/03/27)** *Palco Principal, Aberto e Social*
  http://pontosapo.com/2008/03/27/palco-principal-aberto-e-social/
- **(2008/04/18)** *Integrando OpenSocial con Symfony*
  http://www.symfony.es/2008/04/14/integrando-opensocial-con-symfony/

Os únicos dados que me permitem pelo menos saber as pessoas que têm tomado conhecimento da existência do *plugin*, são a partir da análise fornecida pelo *Google Analytics*.

Content Performance

Pageviews	Unique Pageviews	Time on Page	Bounce Rate	% Exit	$ Index
294 % of Site Total: 100.00%	229 % of Site Total: 100.00%	00:02:38 Site Avg: 00:02:38 (0.00%)	55.00% Site Avg: 55.00% (0.00%)	34.01% Site Avg: 34.01% (0.00%)	$0.00 Site Avg: $0.00 (0.00%)

URL	Pageviews	Unique Pageviews	Time on Page	Bounce Rate	% Exit	$ Index
/p/sfopensocialplugin/	87	70	00:03:22	57.81%	62.07%	$0.00
/p/sfopensocialplugin/w/list	32	14	00:00:22	33.33%	6.25%	$0.00
/p/sfopensocialplugin/wiki/HowToConfigure	21	17	00:03:09	44.44%	33.33%	$0.00
/p/sfopensocialplugin/wiki/SampleApplication	20	17	00:01:26	50.00%	15.00%	$0.00
/p/sfopensocialplugin/wiki/OpenSocialHelper	14	12	00:04:56	40.00%	35.71%	$0.00
/p/sfopensocialplugin/wiki/SampleActivity	12	10	00:05:29	0.00%	16.67%	$0.00
/p/sfopensocialplugin/wiki/SampleDataRequest	11	11	00:09:36	83.33%	81.82%	$0.00
/p/sfopensocialplugin/downloads/list	10	10	00:00:57	66.67%	40.00%	$0.00
/p/sfopensocialplugin/wiki/HowToInstall	8	6	00:02:41	0.00%	12.50%	$0.00
/p/sfopensocialplugin/wiki/SamplePersonRequest	8	7	00:00:30	0.00%	25.00%	$0.00

1 - 10 of 36

Figura 4.1: Relatório dos conteúdos mais vistos do sfOpenSocialPlugin (googlecode)

Da análise deste relatório do *Analytics*, dá para verificar que as páginas mais vistas, são referentes às configurações e exemplos de uso. Curiosamente a página "*HowToInstall*" apenas tem 10 *pageviews* o que significa que quem tem usado a *wiki* apenas está "curioso" e não tem reais intenções de usar o *plugin*.

4.2 Aplicação OpenSocial do Palco Principal

Aquando do desenvolvimento da aplicação, um dos desafios foi saber quais os *containers* que seriam mais vantajosos para o Palco Principal.

Foi utilizado o site *Google Trends*, e decidido definir as regiões **Portugal** e **Brasil**, dado ser este o público alvo do Palco Principal. Os *containers* usados para fazer o estudo foram: **MySpace**, **Hi5**, **Orkut**, **immem** e **Netlog**.

- **Localização:** Portugal

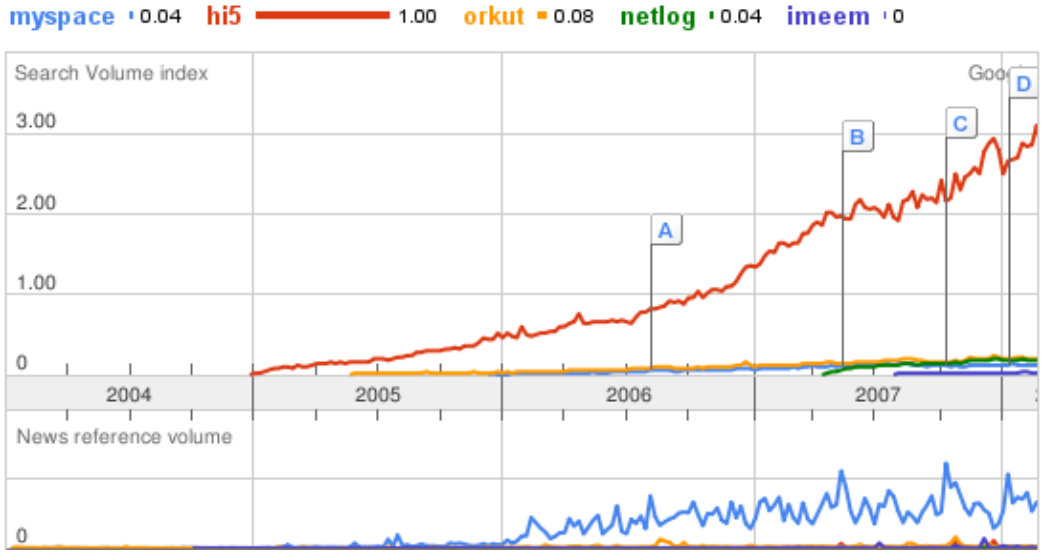


Figura 4.2: Google Trends: MySpace, Hi5, orkut, netlog, imeem (Localização Portugal)

Pela análise do gráfico gerado pelo *Google Trends*, podemos verificar que em Portugal o *container* que tem maior interesse para o Palco Principal é **Hi5**. Depois temos muito afastados o **Orkut** com um factor de **0.08**, aparecendo mais abaixo o **MySpace** e o **Netlog** que têm um factor de **0.04**. O **imeem** nem tem expressão comparado com estes *containers*.

- **Localização:** Brasil

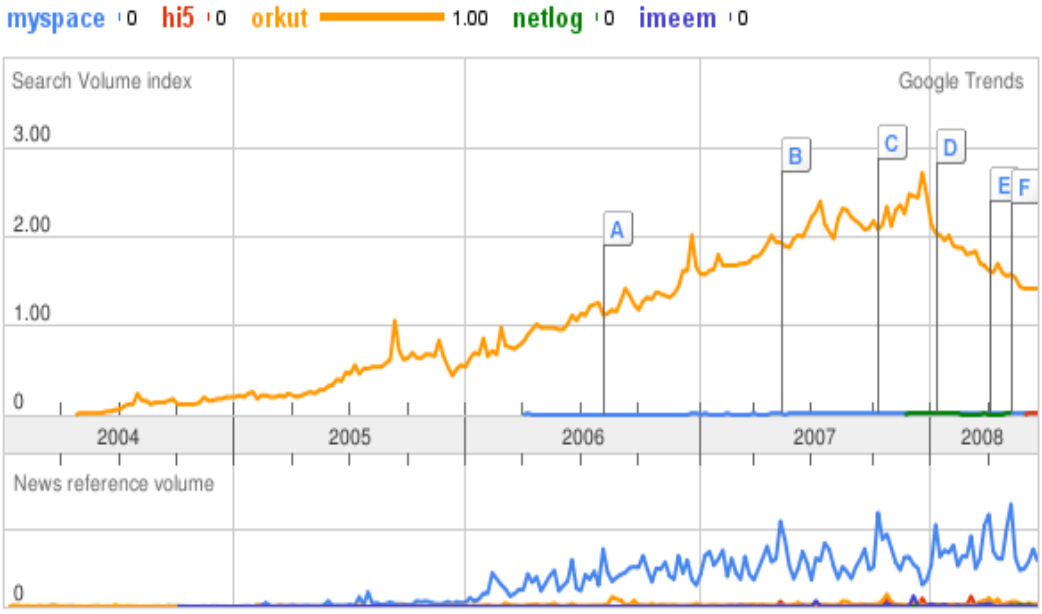


Figura 4.3: Google Trends: MySpace, Hi5, orkut, netlog, imeem (Localização Brasil)

Referente ao volume de pesquisas, no Brasil, o **Orkut** é líder destacado em relação a todos os outros. Muito abaixo do **Orkut**, aprecem todos os outros pela ordem **MySpace**, **Netlog**, **Hi5** e **imeem**.

Foram então escolhidos os *containers* **Hi5**, **MySpace** e **Orkut**, como prioritários para a aplicação.

4.2.1 Submissão da aplicação

A primeira aplicação a ser submetida, foi no **Hi5** a *14/05/2008* e demorou apenas três dias até que a aplicação fosse aprovada.

Para que esta fosse aprovada, foi necessário inserir os seguintes parâmetros nos *ModulePrefs[30]* do *Gadget*:

- **title** : O título da aplicação ou nome
- **summary** : uma pequena descrição da aplicação
- **description** : uma descrição mais completa da aplicação
- **icon** : o *icon* da aplicação
- **thumbnail** : uma pequena imagem da aplicação
- **author_email** : o *email* do autor da aplicação

Este parâmetros são obrigatórios para o correcto funcionamento da aplicação no **Hi5**[36].

O ***Orkut*** foi também submetido a *14/05/2008*, mas só foi aprovada a *12/06/2008*. O **Orkut**, não faz nenhuma especificação em relação ao que deve aparecer nos *ModulePrefs*, mas simplesmente leva períodos muito longos de aprovação das aplicações.

A aplicação adaptada para o **MySpace** só foi submetida a *23/05/2008*, sendo aprovada a *16/06/2008*.

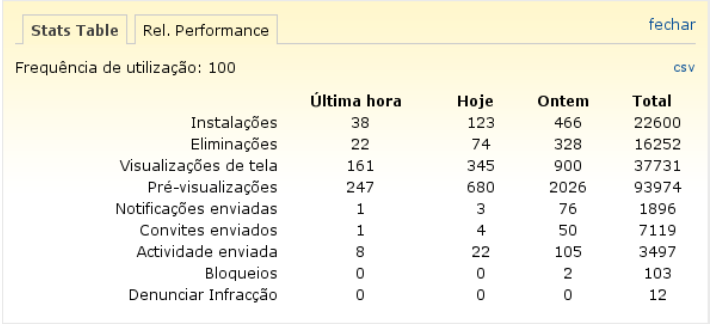
Na realidade o **MySpace** ainda não está completamente "afinado" para o *OpenSocial*, pois ainda não implementa várias funcionalidades da *API*.

Foi necessário retirar algumas funcionalidades da aplicação para que esta fosse aprovada no **MySpace**.

4.2.2 Análise de uso da aplicação

A aplicação tem sido mais usado por utilizadores do **Hi5**, dado esta estar a funcionar plenamente há cerca de dois meses.

As estatísticas fornecidas pelo **Hi5** referentes à aplicação são as seguintes:


Stats Table | Rel. Performance — fechar

Frequência de utilização: 100 — csv

	Última hora	Hoje	Ontem	Total
Instalações	38	123	466	22600
Eliminações	22	74	328	16252
Visualizações de tela	161	345	900	37731
Pré-visualizações	247	680	2026	93974
Notificações enviadas	1	3	76	1896
Convites enviados	1	4	50	7119
Actividade enviada	8	22	105	3497
Bloqueios	0	0	2	103
Denunciar Infracção	0	0	0	12

Figura 4.4: Estatísticas da aplicação instalado no Hi5 (03/07/2008)

Da análise destas estatísticas, podemos verificar que a aplicação está instalada em **6348** perfis do **Hi5** (diferença entre "Instalações" e "Eliminações"), que já foi Pré-visualizada (modo *Preview*) **93.974** vezes e

que foi vista em modo "*Canvas*" ("Visualizações de tela") **37.731** vezes.

Podemos também verificar que foram enviadas **1.896** notificações que corresponde a dedicar músicas a outros utilizadores do *container* (ver secção 3.2.5.1.6 "*Dedicar musica* ", página 67) e que foram adicionadas pelo menos **3.497** músicas ao perfil (ver secção 3.2.5.1.3 "*Adicionar músicas ao perfil* ", página 65).

Os utilizadores do **Hi5** enviaram também **7.119** convites aos seus amigos para usarem a aplicação, sendo que esta foi bloqueada por **103** utilizadores e **12** utilizadores denunciaram a aplicação como "infracção".

Os outros *containers* onde a aplicação se encontra (**Orkut** e **MySpace**) não fornecem quaisquer dados estatísticos referentes ao uso da aplicação, por isso será realizada uma análise especulativa tirando partido dos dados que se encontram na base de dados.

Serão realizadas algumas *queries* à base de dados de produção por forma a obter alguns dados mais concretos do uso da aplicação.

Table	Records	Size
container	~5	16.0 KiB
container_user	~89,983	27.6 MiB
dedicate_music	~2,753	2.0 MiB
lifecycle	0	1.0 KiB
music	~1,339	384.0 KiB
owner	~915	192.0 KiB
pp_user	~862	240.0 KiB
profile_music	~6,771	912.0 KiB
8 table(s)	~102,628	31.3 MiB

Figura 4.5: Dados na Base de dados do OpenSocial(30/06/2008)

A primeira análise a ser feita é referente ao número total de utilizadores que já viram a aplicação instalada ou que têm a aplicação instalada no seu perfil.

Para isso foi realizada uma pesquisa na tabela "***container_user***", e agrupados os resultados por *container*. O resultado desta pesquisa feita à base de dados pode ser consultada na Figura 4.6.

Container	Contagem
hi5.com	88868
orkut.com	2355
myspace.com	144
shindig	5
imeem.com	1

Figura 4.6: registos de container_users ordenados por container

Podemos verificar que o **Hi5**, neste momento possui perto de **97%** dos "***container_user***"s que se encontram nesta tabela o que dá para ter uma ideia do impacto que o **Hi5** está a ter no uso da aplicação. Estes resultados já seriam de esperar uma vez que o **Orkut** e o **MySpace** ainda só têm a aplicação disponível há coisa de quinze dias.

Referente à tabela "***owner***", que são os utilizadores que têm a aplicação instalada e possuem uma conta no Palco Principal, foi feita uma pesquisa à base de dados pelos *owners* de determinado *container* e desses *owners* identificados quais tinham conta de *Banda* ou de *Utilizador* no Palco Principal.

Os resultados aglomerados podem ser vistos na Figura 4.7.

Container	Contagem	Utilizadores	Bandas
hi5.com	819	502	317
myspace.com	42	8	34
orkut.com	61	36	25

Figura 4.7: owners ordenados pela contagem de registos em cada container

Da análise desta tabela, podemos verificar que o impacto do **Hi5** referente aos outros resultados decresce para **88.83%** e que destes, **38.7%** são *Bandas* e **61.3%** estão autenticados como *Utilizadores*. O **Orkut** possui **6.62%** dos resultados, onde **40.98%** são *Bandas* e **59.02%** são *Utilizadores*. Quanto ao **MySpace**, cuja fatia ocupa apenas **4.56%** dos resultados, **80.95%** dos *owners* são *Bandas* e **19.05%** têm conta de *Utilizador*.

4.2.3 Integração no palco

Da mesma maneira que a aplicação comunica com o Palco Principal usando a *API*, o mesmo acontece no caso inverso.

Ou seja, através da *API* que foi desenvolvida neste projecto, o Palco Principal criou uma nova secção na sua página inicial onde aparecem as *Top 3* músicas em cada um dos *containers*, tal como é mostrado na Figura 4.8.

Figura 4.8: Tops das músicas em mais perfis de cada container

Outra secção que foi criada no site do Palco Principal, foi na página de cada música onde agora aparece também onde essa música foi vista.

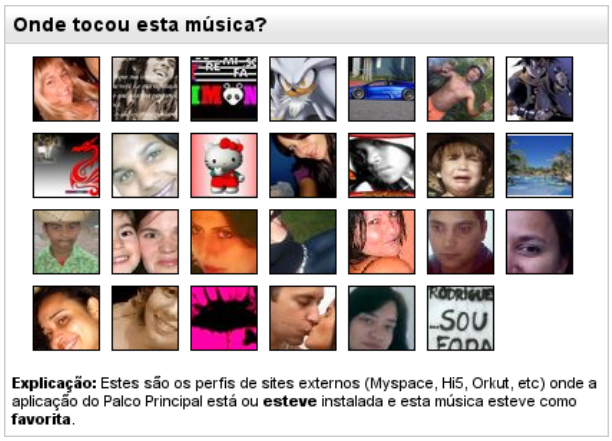


Figura 4.9: Onde tocou a música

Capítulo 5: Conclusões finais

5.1 Principais resultados obtidos

O objectivo deste projecto consistia em realizar um estudo profundo do *OpenSocial* e perceber como poderia ser vantajoso para o Palco Principal, dado o seu modelo de negócio.

A partir do estudo realizado, foi possível verificar que a *API OpenSocial* é já bastante completa, permitindo que um *site* possa realmente "*viver*" dentro de um dado *container*. Uma aplicação *OpenSocial* depende de terceiros para poder funcionar correctamente (i.e. *containers*) e a mesma apenas tem valor neste contexto.

Este é um factor de risco que pode levar empresas a não apostar no desenvolvimento deste tipo de aplicações, pois o seu sucesso é incerto e está completamente dependente de terceiros.

Outra questão é o facto do *OpenSocial* ser ainda muito recente, e os *containers* que o suportam não terem as suas implementações completamente funcionais. Este pode ser outro factor de desmotivação para se apostar no desenvolvimento de aplicações deste tipo.

Não existe propriamente um *debugger* para aplicações *OpenSocial* e muitas vezes, para se fazer *debug* ou experimentar determinadas funcionalidades, tem de se testar em aplicações que já se encontram em ambiente de produção.

Como exemplo, a maior parte dos *containers* não permite às aplicações criar a*ctividades* ou enviar *notificações* para utilizadores caso estas não estejam já aprovadas. Isto implica que tem de haver sempre uma fase de teste quando se implementam estas funcionalidades, que por uma razão ou outra, podem causar indisponibilidade da aplicação.

Apostar num *plugin* como o que foi realizado neste projecto, permite reduzir o risco na implementação e aumentar a segurança em relação à tecnologia. Este *plugin* permite que código criado por terceiros possa ser reutilizado em novos projectos que necessitem de funcionalidades idênticas, fornecendo assim módulos amplamente testados.

Para resolver estes problemas de teste e por forma a tornar as implementações mais maduras e de uma forma sustentada, existe também o projecto *shindig* (http://incubator.apache.org/shindig/) que é uma implementação *Open Source* para *containers OpenSocial*. Este projecto está ainda na incubadora do *Apache Software Foundation* (http://www.apache.org/), mas tem tido grandes desenvolvimentos desde o início deste ano.

O sucesso deste projecto pode levar a que mais sites sociais adiram ao *OpenSocial*, criando um efeito de "bola de neve" que só traz vantagens para todos os intervenientes.

5.1.1 sfOpenSocialPlugin

O sfOpenSocialPlugin é o nome dado ao *plugin* desenvolvido neste projecto, e que permite desenvolver aplicações OpenSocial em cima da *framework symfony*.

Este *plugin* foi aprovado pelo *symfony* no dia 9/04/2008, e actualmente encontra-se na versão *1.0.1*.

Apesar deste *plugin* cumprir os objectivos para o qual foi desenvolvido, não é conhecido até ao momento qualquer projecto que o use para desenvolver aplicações OpenSocial (com a excepção da aplicação realizada neste projecto).

Este panorama seria algo previsível, pois o nicho alvo deste *plugin* é muito reduzido.

Ou seja, para se usar este *plugin*, os programadores necessitam de saber usar as funcionalidades fornecidas pelo *symfony* (tais como saber lidar com *components*, definir *layouts*, etc...), compreender o funcionamento do OpenSocial e possuir algum conhecimento em JavaScript.

Um dos principais factores que contribui para o reduzido interesse no sfOpenSocialPlugin é de facto a acentuada curva de aprendizagem até se começar a obter produtividade com o uso do mesmo.

Por forma a simplificar o uso do *plugin*, foi criada a *task* "*init-opensocial-app*" (ver secção 3.1.5.1 "*sfPakeOpenSocial*", página 44) que gera automaticamente toda a estrutura de um projecto OpenSocial.

Uma vez tendo a estrutura gerada, o programador apenas necessita de trabalhar em cima dela, o que é já um bom ponto de orientação para quem se está a adaptar ao *plugin*.

O facto de o *plugin* implementar as APIs JavaScript do OpenSocial em PHP5, poderá ser interpretado como uma desvantagem pois o programador tem de aprender mais uma API, acentuando ainda mais a curva de aprendizagem. No entanto esta é uma falsa questão, pois as vantagens que a API em PHP5 traz, no sentido de fornecer uma melhor integração com o *symfony*, são numerosas.

Com esta implementação em PHP5 o programador pode tirar proveito dos mecanismos do *symfony* para passar objectos para os *templates* ou *components*, fazer uma melhor separação das camadas de modelação e visualização, estender a própria biblioteca disponível do sfOpenSocialPlugin, bem como tirar proveito de muitas outras funcionalidades fornecidas pela *framework symfony*.

Caso o programador não encontre necessidade de usar estas funcionalidades, pode apenas usar a *task* para criar a estrutura da aplicação, e usar a API JavaScript OpenSocial normalmente dentro dos *templates.*

5.1.2 Aplicação OpenSocial para o Palco Principal

A aplicação desenvolvida encontra-se já aprovada nos três principais *containers* OpenSocial: **Hi5**, **MySpace** e **Orkut**.

As principais funcionalidades da aplicação final são:

i. Ouvir músicas;
ii. Adicionar músicas ao seu perfil da aplicação;
iii. Dedicar músicas aos seus amigos;
iv. Permitir aos utilizadores do Palco Principal autenticarem-se, importando a sua conta para a aplicação instalada.

Qualquer utilizador que tenha a aplicação instalada pode adicionar músicas ao seu perfil. Esta funcionalidade é intuitiva, e permite que qualquer música que se encontre em aplicações instaladas no container possa ser adicionadas ao seu perfil.

Poder dedicar músicas aos amigos (quer tenham ou não a aplicação instalada) é sem dúvida a funcionalidade que melhor explora as capacidades virais que o OpenSocial disponibiliza.

Finalmente a possibilidade de importar os perfis do Palco Principal para a aplicação OpenSocial é também um dos factores considerados importantes para os utilizadores do Palco Principal, pois sabem que vão ter disponível nesse *container* os seus perfis (caso se autentiquem).

Esta aplicação foi útil para verificar o impacto das aplicações OpenSocial, pois mesmo tratando-se de uma aplicação experimental, possui já dezenas de milhares de utilizadores em apenas dois meses de actividade.

É também de salientar que a maior parte dos utilizadores desta aplicação são utilizadores do *container* Hi5, dado esta aplicação ter pelo menos mais um mês de "vida" em relação ao MySpace e ao Orkut.

O que acontece é que a aplicação, para poder ser usada em determinado *container*, deve primeiro ser aprovada pelo mesmo. Em relação ao Hi5 este processo foi muito breve (2 dias úteis), enquanto que no MySpace e Orkut se arrastou entre três a quatro semanas.

A grande diferença das aplicações instaladas nos três *containers* é que a aplicação que se encontra instalada no MySpace apenas está disponível em Português. Isto deve-se a uma limitação do MySpace que apenas permite que as aplicações estejam disponíveis numa única língua.

Excluindo alguns casos de indisponibilidade dos *containers*, a aplicação encontra-se a funcionar a 100%, permitindo que os utilizadores usem todas as suas funcionalidades independentemente do *container* em que a aplicação se encontre.

5.2 Desenvolvimentos futuros

Apesar de este projecto ter sido desenvolvido todo na versão *OpenSocial 0.7*, a versão actual é já a

0.8[37]. Esta versão foi lançada a 27 de Maio de 2008, mas até à altura ainda nenhum *container* a suporta e por isso mesmo não foi abordada neste documento.

O *OpenSocial* tem tudo para continuar por muito mais tempo, pois é já suportado por inúmeros *containers[15]* e existem já planos para futuras versões.

O facto do *OpenSocial* ser "acompanhado" pelo *Google* e de os maiores sites de redes sociais o suportarem (excluindo o *Facebook*) é uma razão para que se continue a apostar nesta nova tecnologia.

Em relação à aplicação do Palco Principal, está já a ser pensada uma nova versão, que tem o objectivo de ser mais integrada no *container*, tirando proveito das diversas vistas que os *containers* disponibilizam e permitir também a passagem de argumentos pelo *URL* quando se está no modo *Canvas* (algo que a versão actual não suporta).

Para além da aplicação OpenSocial para o Palco Principal, e em virtude do estudo realizado neste projecto ter verificado o impacto que uma aplicação OpenSocial possuiu, está também a ser pensada a melhor forma de adaptar o actual portal para que possa em breve tornar-se também ele num *container* OpenSocial.

Capítulo 6: Referências

Bibliography

1: , YAML, 2008, http://www.yaml.org/
2: , Propel, , http://propel.phpdb.org/trac/
3: Symfony, How to create and contribute a symfony plugin, 2008, http://trac.symfony-project.com/wiki/SymfonyPlugins#Howtocreateandcontributeasymfonyplugin
4: Ben Collins-Sussman,Brian W. Fitzpatrick & C. Michael Pilato, Version Control with Subversion, 2007, http://svnbook.red-bean.com/
5: OpenSocial, Hosting OpenSocial Apps, 2008, http://code.google.com/apis/opensocial/container.html
6: OpenSocial, OpenSocial API Reference (v0.7), 2008, http://code.google.com/apis/opensocial/docs/0.7/reference/
7: Google, Gadgets API Reference, 2008, http://code.google.com/apis/gadgets/docs/reference/
8: OpenSocial, opensocial.Person (v0.7), 2008, http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.Person.html
9: OpenSocial, opensocial.DataRequest (v0.7), 2008, http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.DataRequest.html
10: OpenSocial, opensocial.Person.getField(key) (v0.7), 2008, http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.Person.html#getField
11: OpenSocial, gadgets.util. hasFeature(feature), 2008, http://code.google.com/apis/opensocial/docs/0.7/reference/gadgets.util.html#hasFeature
12: OpenSocial, opensocial.Environment (v0.7), 2008, http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.Environment.html
13: OpenSocial, Static Class gadgets.io, 2008, http://code.google.com/apis/opensocial/docs/0.7/reference/gadgets.io.html
14: Wikipedia, Spoofing attack, 2008, http://en.wikipedia.org/wiki/Spoofing_attack
15: OpenSocial, OpenSocial: Getting Started Guide, 2008, http://code.google.com/apis/opensocial/gettingstarted.html
16: Paul Lindner, Back to the Sandbox, 2008, http://www.hi5networks.com/developer/2008/05/back-to-the-sandbox.html

17: Wikipedia, Google Trends, 2008, http://pt.wikipedia.org/wiki/Google_Trends
18: Christian Heilmann, Beginning JavaScript with DOM Scripting and Ajax: From Novice to Professional, 2006
19: Danny Goodman ,Michael Morrison, JavaScript Bible, 2006
20: Duffy, Scott, How to do Everything with JavaScript, 2003
21: Mozilla, Mozilla Developer Center, 2008, http://developer.mozilla.org/en/docs/JavaScript
22: McDuffie, Tina Spain, JavaScript Concepts & Techniques: Programming Interactive Web Sites, 2003
23: Harold, Elliotte Rusty, Effective XML, 2004
24: Harold, Elliotte Rusty, XML in a Nutshell: A Desktop Quick Reference, 2002
25: Chuck Musciano, Bill Kennedy, HTML: The Definitive Guide, Third Edition, 2007
26: Eric Meyer, Cascading Style Sheets: The Definitive Guide, 2006
27: Open Source Initiative OSI, The MIT License, , http://www.opensource.org/licenses/mit-license.php
28: Andi Gutmans, Stig Sæther Bakken, Derick Rethans, PHP 5 Power Programming, 2005
29: Eric Meyer, CSS: The Definitive Guide, 2006
30: Google, Gadgets API: ModulePrefs, 2008,
http://code.google.com/apis/gadgets/docs/reference.html#Moduleprefs_Ref
31: , YAML :: 2.1. Collections, 2008, http://www.yaml.org/spec/1.1/#id857181
32: , Gadgets - User Preferences, 2008,
http://code.google.com/apis/gadgets/docs/reference.html#Userprefs_Ref
33: Google, Gadgets API : Content, 2008,
http://code.google.com/apis/gadgets/docs/reference.html#Content_Ref
34: , Cross-site Scripting, 2008, http://en.wikipedia.org/wiki/Cross-site_scripting
35: Propel, Propel :: It makes for more secure applications., 2008,
http://propel.phpdb.org/trac/wiki/Users/Introduction/WhyPropel#Itmakesformoresecureapplications.
36: David McQueen, Quick refresher on required ModPrefs for gallery approval, 2008,
http://www.hi5networks.com/developer/2008/04/quick-refresher-on-required-mo.html
37: OpenSocial, OpenSocial API Reference (v0.8), 2008,
http://code.google.com/apis/opensocial/docs/0.8/reference/

Capítulo 7: Anexos

A - API Restful Beta do Palco Principal

A.1 - Login

URL:	http://www.palcoprincipal.com/api/login/Nome/:Nome/Password/:Password
Parâmetros:	**(1) Nome** : o login no Palco Principal **(2) Password** : a password no Palco Principal

Resposta:	`<response>`
	`  <login>`
	`  <username>xxxxxxxxx</username>`
	`  <name>xxxxxxxxxxxxx</name>`
	`  <style>xxxxxxxxxxx</style>`
	`  <type>xxxxxxxxxxxxx</type>`
	`  <url>xxxxxxxxxxxxxx</url>`
	`  <photoUrl>xxxxxxxxx</photoUrl>`
	`  <slug>xxxxxxxxxxxxxx</slug>`
	`  </login>`
	`</response>`
	Listagem 7.1: Resposta login

Erro:	`<response>`
	`  <error>`
	`  <Nome>Email ou password errados</Nome>`
	`  </error>`
	`</response>`
	Listagem 7.2: Erro no login

Tabela 7.1: API RESTful do Palco Principal: login

A.2 - Playlist

URL:	http://www.palcoprincipal.com/api/playlist[/:slug][/n/:n/page/:page]
Parâmetros:	**(1) slug** : o "slug" do utilizador no Palco Principal **(2) n** : (opcional) número de resultados máximo **(3) page** : (opcional) número da página que se quer aceder

Resposta:	`<response total="x">`
	`  <track>`
	`  <title>xxxxxxxxxxx</title>`
	`  <bandName>xxxxxxxxx</bandName>`
	`  <link>xxxxxxxxxxxxx</link>`
	`  <albumID>xx TODO xx</albumID>`
	`  <albumImage>xxxxxxx</albumImage>`
	`  <albumName>xxxxxxxx</albumName>`
	`  <idMusic>xx TODO xx</idMusic>`
	`  <mp3>xxxxxxxxxxxxx</mp3>`
	`  </track>`
	`</response>`
	Listagem 7.3: Resposta playlist

Erro:	`<response>`
	`  <error>`
	`  <Authentication>Not Authenticated</Authentication>`
	`  </error>`
	`</response>`
	Listagem 7.4: Erro no playlist

Tabela 7.2: API RESTful do Palco Principal: playlist

A.3 - Friends

URL:	http://www.palcoprincipal.com/api/friends[/:slug][/n/:n/page/:page]
Parâmetros:	**(1) slug** : o "slug" do utilizador no Palco Principal **(2) n** : (opcional) número de resultados máximo **(3) page** : (opcional) número da página que se quer aceder

Resposta:	`<response total="x">`
	`  <user>`
	`  <username>xxxxxxxxx</username>`
	`  <name>xxxxxxxxxxxx</name>`
	`  <style>xxxxxxxxxx</style>`
	`  <type>xxxxxxxxxxxx</type>`
	`  <url>xxxxxxxxxxxxx</url>`
	`  <photoUrl>xxxxxxxxx</photoUrl>`
	`  <slug>xxxxxxxxxxxxx</slug>`
	`  </user>`
	`  ...`
	`</response>`
	Listagem 7.5: Resposta friends

Erro:	`<response>`
	`  <error>`
	`  <Authentication>Not Authenticated</Authentication>`
	`  </error>`
	`</response>`
	Listagem 7.6: Erro no friends

Tabela 7.3: API RESTful do Palco Principal: friends

A.4 - Photos

URL:	http://www.palcoprincipal.com/api/photos[/:slug][/n/:n/page/:page]
Parâmetros:	**(1) slug** : o "slug" do utilizador no Palco Principal **(2) n** : (opcional) número de resultados máximo **(3) page** : (opcional) número da página que se quer aceder

Resposta:	`<response total="x">`
	`  <photo>`
	`  <photoID>xx TODO xx</photoID>`
	`  <smallUrl>xxxxxxxxxxxxxx</smallUrl>`
	`  <mediumUrl>xxxxxxxxxxxxx</mediumUrl>`
	`  <url>xxxxxxxxxxxxxxxxxx</url>`
	`  <permalink>xxxxxxxxxxxxx</permalink>`
	`  </photo>`
	`</response>`
	Listagem 7.7: Resposta photos

Erro:	`<response>`
	`  <error>`
	`  <Authentication>Not Authenticated</Authentication>`
	`  </error>`
	`</response>`
	Listagem 7.8: Erro no photos

Tabela 7.4: API RESTful do Palco Principal: photos

B - Código fonte do sfOpenSocialPlugin

B.1 - Classes Base

B.1.1 - OSConfig

```
<?php
class OSConfig
{
  public static function add($key,$value,$root=false) { ... }
  /**
   * This method returns the application base url
   * @return string The application base url
   */
  public static function getApplicationBaseUrl() { ... }

  private static function importFeature($feature) { ... }
  /**
   * Adiciona o DynamicHeight automaticamente ao gadget, quando alguma função o use.
   * Garantir que será inserido no ficheiro XML "<Require feature="dynamic-height" />
   * @return string The <Require feature="dynamic-height" /> to append in the moduleprefs
   */
  public static function importFeatureDynamicHeight() { ... }
  /**
   * Adds support for minimessages
   */
  public static function importMiniMessage() { ... }

  /**
   * Adds support for tabs
   */
  public static function importTabs() { ... }

  /**
   * Adds skins feature to the gadget.
   */
  public static function importFeatureSkins() { ... }

  /**
   * Includes <Require feature="views" />
   */
  public static function importViews() { ... }

  /**
   * Adiciona o settitle automaticamente ao gadget, quando alguma função o use.
   * Garantir que será inserido no ficheiro XML "<Require feature="settitle" />"
   */
  public static function importSetTitle() { ... }

  /**
   * Adiciona o flash automaticamente ao gadget, quando alguma função o use.
   * Garantir que será inserido no ficheiro XML "<Require feature="flash" />"
   */
  public static function importFlash() { ... }

  /**
   * Adiciona o SetPrefs automaticamente ao gadget, quando alguma função o use.
   * Garantir que será inserido no ficheiro XML "<Require feature="setprefs" />"
   */
  public static function importSetPrefs() { ... }

  /**
  * Esta função incluiu todos os módulos definidos no ficheiro YAML
  */
  public static function include_modules() { ... }

  /**
  * Esta função permite criar os modulos com valores definidos pelo utilizador( não consulta o
ficheiro YAML)
  */
  private static function include_modulePrefs($module_pram,$requires,$locales) { ... }
```

```
  /**
  * Esta função incluiu as UserPref definidas no ficheiro YAML
  */
  public static function include_userprefs() { ... }
}
```

Listagem 7.9: *Código fonte: OSConfig.php*

B.1.2 - JSFunction

```
<?php
class JSFunction
{
  public static function addJSTags($content) { ... }

  public static function addJSOpenTag(JSFunction $function=null) { ... }

  public static function addJSCloseTag(JSFunction $function=null) { ... }

  /**
   * @deprecated Usar antes o $func->__toString()
   */
  public static function renderFunction(JSFunction $func) { ... }

  function __construct($name=null,$args=null,$content=null) { ... }
  public function isClosure() { ... }

  public function getName() { ... }
  public function setName($name) { ... }

  public function getArgs() { ... }
  public function setArgs($args) { ... }

  /**
   * @return string This JavaSript Function content
   */
  public function getContent() { ... }

  /**
   * Appends content to this JavaSript function
   *
   * @param string $content
   */
  public function addContent($content) { ... }

  public function setContent($content) { ... }

  public function addVariable(JSVariable $variable) { ... }

  public function getVariables() { ... }
  public function setVariables($variables){ ... }

  private function variablesToString() { ... }

  public function __toString() { ... }
}
```

Listagem 7.10: Código fonte: OSConfig.php

B.1.3 - StringUtil

```
<?php
class StringUtil
{
  public static function appendableString($string, $appendable) { ... }

  public static function newLine($string=null) { ... }

  public static function normalizeString ($s = '') { ... }
}
```

Listagem 7.11: Código fonte: StringUtil.php

B.1.4 - oSFilter

```
<?php
class oSFilter extends sfFilter
{
  public function execute($filterChain){
    $filterChain->execute();
    $response = $this->getContext()->getResponse();

    // arranjar o JavaSript
    $response->setContent(str_ireplace('//<![CDATA[','',$response->getContent()));
    $response->setContent(str_ireplace('//]]>','',$response->getContent()));

    //adicionar os ModulePrefs
    $modulePrefs = OSConfig::include_modules();
    $modulePrefs .= OSConfig::include_userprefs();
    $response->setContent(str_ireplace('<Module>',"<Module>\n".$modulePrefs,$response-
>getContent()."\n"));
  }
}
```

Listagem 7.12: Código fonte: oSFilter.php

B.2 - OpenSocial namespace

B.2.1 - OSPerson

```
<?php
abstract class OSPerson
{
  /**
  * Esta classe cria um user em determinado contexto de um pedido ao container, com o seu nome
da variável.
  */
  function __construct($var) { ... }

  /**
   * @return string This OSPerson variable name
   */
  function getVarName() { ... }

  abstract function getType();

  function getName($appendable=false) { ... }
  function getId($appendable=false) { ... }
  function isOwner($appendable=false) { ... }
  function isViewer($appendable=false) { ... }
  function getField($field,$appendable=false) { ... }
  function getThumbnailURL($appendable=false) { ... }
  function getProfileURL($appendable=false) { ... }
}
```

Listagem 7.13: Código fonte: OSPerson.class.php

B.2.1.1 - OSOwner

```
<?php
class OSOwner extends OSPerson
{
  /**
  * This classe creates an OpenSocial Request with a $request_name and with $opt_params_name.
  */
  function __construct($var_name) { ... }
  function getType() { ... }
}
```

Listagem 7.14: Código fonte: OSOwner.class.php

B.2.1.2 - OSViewer

```
<?php
class OSViewer extends OSPerson
{
```

```
  /**
  * This classe creates an OpenSocial Request with a $request_name and with $opt_params_name.
  */
  function __construct($var_name) { ... }
  function getType() { ... }
}
```

Listagem 7.15: Código fonte: OSViewer.class.php

B.2.1.3 - OSFriend

```
<?php
class OSFriend extends OSPerson
{
  /**
  * This classe creates an OpenSocial Request with a $request_name and with $opt_params_name.
  */
  function __construct($var_name) { ... }
  function getType() { ... }
}
```

Listagem 7.16: Código fonte: OSFriend.class.php

B.2.2 - Activity

B.2.2.1 - OSActivity

```
<?php
/**
 * This class represents an Activity in the OpenSocial API.
 *
 * Using activities are very simple and to create an OSActivity is as simple as:
 * <code>
 * $this->activity = new OSActivity('update_musics');
 * </code>
 * You can create and send an OSActivity inside the Action like this:
 * <code>
 * //Action
 * $this->mediaItem = new OSActivityMediaItem('logo');
 * $this->mediaItem->setImageItem();
 * $this->mediaItem->setURL('http://www.palcoprincipal.com/images/logo_opensocial_pp.png');
 *
 * $this->activity->addMediaItem($this->mediaItem);
 * $this->activity->setTitle('New Music Added!');
 * $this->activity->createActivity(true);
 * </code>
 * Or if you like to setup the OSActivity object in the Action an execute it in the View you
can do like this:
 * <code>
 * //Action
 * $this->mediaItem = new OSActivityMediaItem('logo');
 * $this->mediaItem->setImageItem();
 * $this->mediaItem->setURL('http://www.palcoprincipal.com/images/logo_opensocial_pp.png');
 *
 * $this->activity->addMediaItem($this->mediaItem);
 * $this->activity->setPattern('title','O utilizador %s actualizou a musica %s às %s!');
 * //View
 * function <?php $activity->getJSFunction->getName()?>() {
 *     <?php $activity->setTitle(sprintf($activity->getPattern('title'),'eu','lah lah','aa'));
 *           $activity->createActivity(false);
 *           echo $activity;
 *
 *     ?>
 *   };
 * </code>
 *
 * @package    opensocial
 * @subpackage activity
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class OSActivity
{
  /**
```

```
  * This constructor creates an instance of OSActivity.
  * @param string The name used for this activity.
  */
  function __construct($activity_var) { ... }
  public function setHighPriority(){ ... }
  public function setLowPriority(){ ... }
  public function getPriority(){ ... }

  public function setErrorCallbackFunction(JSFunction $opt_callback) { ... }

  public function getErrorCallbackFunction() { ... }

  /**
   * This function creates an Activity
   * @param boolean If true, there will be injected the JSFunction in the response; else it
will just generate the JSFunction but will not inject in the response.
   * @param boolean $noCallback If it should inclue the callback function
   *
   */
  public function createActivity($inject =true,$noCallback=false) { ... }

  /**
   * This method returns the JSFunction used to create this activity
   * @return JSFunction The JSFunction used to create this activity
   */
  public function getJSFunction(){ ... }

  public function __toString() { ... }

  public function addMediaItem(OSActivityMediaItem $mediaItem) { ... }

  /**
  * This method allows to set the pattern for a variable
  * @param string $key The key value
  * @param string $value The value
  */
  public function setPattern($key,$value) { ... }

  /**
   * Get's the Value of a specific Key
   * @param string The key
   * @return string The value for this Key
   */
  public function getPattern($key) { ... }

  public function getActivityVarName(){ ... }
  public function getActivityParamsVarName(){ ... }
  public function getActivityParams(){ ... }
  public function getActivityMediaItemsVarName(){ ... }
  public function getActivityMediaItems(){ ... }

  /**
   * Enter description here...
   *
   * @param String $title The title of this activity
   * @param boolean $js_param If the title is a JavaSript variable
   */
  public function setTitle($title,$js_param=false) { ... }
  public function setTitleID($titleID) { ... }
  public function getField($field,$appendable){ ... }
}
```

Listagem 7.17: Código fonte: OSActivity.class.php

B.2.2.2 - OSActivityPriority

```
<?php
class OSActivityPriority
{
  const ACTIVITYPRIORITY_CLASS = 'opensocial.CreateActivityPriority';
  const HIGH = 'opensocial.CreateActivityPriority.HIGH';
  const LOW = 'opensocial.CreateActivityPriority.LOW';
}
```

Listagem 7.18: Código fonte: OSActivityPriority.class.php

B.2.2.3 - OSActivityField

```
<?php
class OSActivityField
{
  /** Opensocial class */
  const ACTIVITYFIELD_CLASS = 'opensocial.Activity.Field';

  /** A string specifying the primary text of an activity. */
  const TITLE = 'opensocial.Activity.Field.TITLE';
  /** A string specifying the title template message ID in the gadget spec.  */
  const TITLE_ID = 'opensocial.Activity.Field.TITLE_ID';
  /** A string specifying the application that this activity is associated with. */
  const APP_ID = 'opensocial.Activity.Field.APP_ID';
  /** A string specifying an optional expanded version of an activity. */
  const BODY = 'opensocial.Activity.Field.BODY';
  /** A string specifying the body template message ID in the gadget spec. */
  const BODY_ID = 'opensocial.Activity.Field.BODY_ID';
  /** An optional string ID generated by the posting application. */
  const EXTERNAL_ID = 'opensocial.Activity.Field.EXTERNAL_ID';
  /** A string ID that is permanently associated with this activity. */
  const ID = 'opensocial.Activity.Field.ID';
  /** Any photos, videos, or images that should be associated with the activity.*/
  const MEDIA_ITEMS = 'opensocial.Activity.Field.MEDIA_ITEMS';
  /** A string specifying the time at which this activity took place in milliseconds since the
epoch.*/
  const POSTED_TIME = 'opensocial.Activity.Field.POSTED_TIME';
  /** A number between 0 and 1 representing the relative priority of this activity in relation
to other activities from the same source */
  const PRIORITY = 'opensocial.Activity.Field.PRIORITY';
  /** A string specifying the URL for the stream's favicon. */
  const STREAM_FAVICON_URL = 'opensocial.Activity.Field.STREAM_FAVICON_URL';
  /** A string specifying the stream's source URL. */
  const STREAM_SOURCE_URL = 'opensocial.Activity.Field.STREAM_SOURCE_URL';
  /** A string specifing the title of the stream. */
  const STREAM_TITLE = 'opensocial.Activity.Field.STREAM_TITLE';
  /** A string specifying the stream's URL. */
  const STREAM_URL = 'opensocial.Activity.Field.STREAM_URL';
  /** A map of custom keys to values associated with this activity. */
  const TEMPLATE_PARAMS = 'opensocial.Activity.Field.TEMPLATE_PARAMS';
  /** A string specifying the URL that represents this activity. */
  const URL = 'opensocial.Activity.Field.URL';
  /** The string ID of the user who this activity is for. */
  const USER_ID = 'opensocial.Activity.Field.USER_ID';
  private function __construct() { ... }
  private function __clone() { ... }
}
```

Listagem 7.19: Código fonte: OSActivityField.class.php

B.2.2.4 - OSActivityMediaItem

```
<?php
class OSActivityMediaItem
{

  function __construct($activityMediaItem_var) { ... }

  public function getActivityMediaItemVarName(){ ... }

  public function setAudioItem() { ... }
  public function setVideoItem() { ... }
  public function setImageItem() { ... }

  public function setUrl($url) { ... }

  public function getType() { ... }
  public function getURL() { ... }
}
```

Listagem 7.20: Código fonte: OSActivityMediaItem.class.php

B.2.2.5 - OSActivityMediaItemType

```
<?php
class OSActivityMediaItemType
{
  const ACTIVITYMEDIAITEMTYPE_CLASS = 'opensocial.Activity.MediaItem.Type';

  const AUDIO = 'opensocial.Activity.MediaItem.Type.AUDIO';
  const IMAGE = 'opensocial.Activity.MediaItem.Type.IMAGE';
  const VIDEO = 'opensocial.Activity.MediaItem.Type.VIDEO';
}
```

Listagem 7.21: Código fonte: OSActivityMediaItemType.class.php

B.2.2.6 - JSFunctionErrorCallback

```
<?php
class JSFunctionErrorCallback extends JSFunction
{
  function __construct(OSActivity $activity) { ... }
}
```

Listagem 7.22: Código fonte: JSFunctionErrorCallback.class.php

B.2.3 - DataRequest

B.2.3.1 - OSDataRequest

```
<?php

/**
 * Used to request social information from the container. This includes data for friends,
profiles, app data, and activities. All apps that require access to people information should
send a DataRequest to the container. {@link
http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.DataRequest.html
opensocial.DataRequest (0.7) }
 *
 * Remember that this object should only be used in the Action, and in the View you should use
OSDataResponse instead. Anyway if you really need to access to this object in can pass it to
the view.
 * Using OSDataRequest is as simple as creating the object, appending request's to it, and
calling the "send()" function to generate the response. Here it's a sample code of how to get
the viewers friends from the container:
 *
 * <code>
 * // this code is in the Action
 * $this->viewer_friends = new OSViewerFriends('viewer_friends');

 * $this->people_request = new OSPeopleRequest($this->viewer_friends);
 * $this->people_request->addProfileDetails(OSPersonField::PROFILE_URL);

 * $data_request = new OSDataRequest('requestInfo');
 * $data_request->addRequest($this->people_request);

 * $this->data_response = $data_request->send(true);
 * // this code in the View
 * function <? echo $data_response->getJSFunction()->getName() ?>(<? echo $data_response-
>getJSFunction()->getArgs() ?>)
 * {
 *   <?php echo $data_response->getData($people_request); ?>
 *   <?php $person = new OSFriend('person'); ?>
 *   viewer_friends.each(function(<?php echo $person->getVarName() ?>) {
 *   var thumb = <?php echo $person->getField(OSPersonField::THUMBNAIL_URL); ?>
 *   var profile = <?php echo $person->getField(OSPersonField::PROFILE_URL); ?>
 *   html += '<a href="' + profile + '" target="_top" style="float:left">' +
 *     <img src="' + thumb + '"/></a>';
 * }
 * </code>
 * @package    opensocial
 * @subpackage datarequest
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class OSDataRequest
```

```
{
  /**
  * This classe creates an OpenSocial Data Request. The $id will be used to create variable
names.
  * @param string The name of this OSDataRequest
  */
  public function __construct($id) { ... }

  /**
   * This function should be called after all the request's have been appended to it. This
function will return an OSDataResponse that can be used to get in the view to access to the
containers response.
   * @param bool If true the generated OSDataRequest will be send to the container.
   * @return OSDataResponse The response object returned by the container.
   */
  public function send($inject = true) { ... }

  /**
   * This method allows to append multiple OSDataRequest's
   * @param OSDataRequest The OSDataRequest that will be appended to this one.
   */
  public function appendDataRequest(OSDataRequest $dataRequest) { ... }

  /**
   * Returns the DataRequest variable name.
   * @return string The DataRequest variable name
   */
  public function getDataRequestVarName() { ... }

  /**
   * Returns the Request JSFunction
   * @return JSFunction The Request JSFunction
   */
  public function getDataRequestFunction() { ... }

  /**
   * Returns the Response JSFunction
   * @return JSFunction The Response JSFunction
   */
  public function getDataResponseFunction() { ... }

  /**
   * Returns a list of classes that implements DataRequestable interface
   * @return DataRequestable[] A list of classes that implements DataRequestable
   */
  public function getDataRequestList() { ... }

  /**
   * Set the Request JSFunction.
   * @param JSFunction The new Request JSFunction.
   */
  public function setDataRequestFunction(JSFunction $request_func) { ... }

  /**
   * Add's a new OSDataRequestable object to the RequesList.
   * @param OSDataRequestable The new object to be added.
   */
  public function addRequest(OSDataRequestable $dataRequest) { ... }

  /**
   * This method will update the OSDataRequest Function
   */
  private function updateDataRequestFunction() { ... }

}
```

Listagem 7.23: Código fonte: OSDataRequest.class.php

B.2.3.2 - OSDataResponse

```
<?php
class OSDataResponse
{
  public function getJSFunction() { ... }
```

```
  /**
   * Get's the data request that has generate this response
   *
   * @return OSDataRequest The data request that has generate this response
   */
  public function getDataRequest(){  }

  function __construct(OSDataRequest  $container_request)  {  }
  /**
   * This method returns the data fecthed by the container. By default it initializes the
variable.
   * @param OSDataRequestable $data_request The requested data
   * @param boolean $initialize By default it initializes the variable
   */
  public function getData(OSDataRequestable $data_request,$initialize=true,$appendable=false,
$check_init=false){ ... }

  public function getErrorCode(OSDataRequestable $data_request,$appendable=false)  { ... }

  public function getErrorMessage(OSDataRequestable $data_request,$appendable=false){  }

  public function getOriginalDataRequest(OSDataRequestable $data_request,$appendable=false)  {
}

  public function hadError(OSDataRequestable $data_request = null,$appendable=false) {  }
}
```

Listagem 7.24: Código fonte: OSDataResponse.class.php

B.2.3.3 - OSPeopleRequest

```
<?php
/**
 * This class is intended to make People request's to the container. People request's are
OSViewerFriends or OSOwnerFriends, which allows to get viewer friends and owner friends
respectively.
 * Basically to use this class you need instantiate a class that implements the interface
OSFriends:
 * <code> $this->viewer_friends = new OSViewerFriends('viewer_friends');
 * //or
 * $this->owner_friends = new OSOwnerFriends('owner_friends');
 * </code>
 * Then you need to create an OSPeopleRequest instance, passing has argument one of the
variables previously created:
 * <code>
 * $this->viewer_friends_request = new OSPeopleRequest($this->viewer_friends);
 * //or
 * $this->owner_friends_request = new OSPeopleRequest($this->owner_friends);
 * </code>
 * Now that you have created the OSPeopleRequest object you need to create an OSDataRequest and
add that created object to it:
 * <code>
 * $data_request = new OSDataRequest('requestPeople');
 * $data_request->addRequest($this->viewer_friends_request);
 * //or
 * $data_request->addRequest($this->owner_friends_request);
 * </code>
 * After adding the OSPeopleRequest to the OSDataRequest you need to call the function "send()"
so that this OSDataRequest will be made to the container:
 * <code>
 * $this->data_response = $data_request->send(true);
 * </code>
 * It has been created the request to the container and now you can access to OSViewerFriends
or OSOwnerFriends from the view and iterate over those objects in the view:
 * <code>
 * // View
 * function <? echo $data_response->getJSFunction()->getName() ?>(<? echo $data_response-
>getJSFunction()->getArgs() ?>)
 * {
 *   <?php echo $data_response->getData($people_request); ?>
 *
 *   <?php $person = new OSFriend('person'); ?>
 *   <?php echo $viewer_friends->getVarName() ?>.each(function(<?php echo $person->getVarName()
```

```
?>) {
 *     var thumb = <?php echo $person->getField(OSPersonField::THUMBNAIL_URL); ?>
 *     var profile = <?php echo $person->getField(OSPersonField::PROFILE_URL); ?>
 *     html += '<a href="' + profile + '" target="_top" style="float:left">' +'<img src="' +
thumb + '"/></a>';
 *     document.getElementById('viewer_friends').innerHTML = html;
 *   });
 * }</code>
 *
 */
class OSPeopleRequest implements OSDataRequestable
{
 /**
  * OSPeopleRequest constructor
  * @param $people The OSFriends or OSPerson object
  */
  function __construct($people) { ... }

  public function generateDataRequest(OSDataRequest $dataRequest) { ... }

  public function getOptionalKey() { ... }

  /**
   * This method returns the people object
   * @return OSFriends|OSPerson Returns the people object
   */
  public function getPeople() { ... }

  /**
   * This method returns the var name for profileDetails
   * @return string Profile details var name.
   */
  public function getProfileDetailsVarName() { ... }
  /**
   * Gets the all the params requested
   * @return array The Request params.
   */
  public function getPeopleRequestParams() { ... }

  /**
   * Gets the profile details
   * @return array The profile details.
   */
  public function getProfileDetails() { ... }

  /**
   * Gets the people request variable name
   * @return string The people request variable name.
   */
  public function getPeopleRequestParamsVarName() { ... }

  /**
   * Set's the sort order. Look at OSDataRequestSortOrder to see the available params.
   * @see OSDataRequestSortOrder
   */
  public function setSortOrder($sortOrder) { ... }

  /**
   * Set's the filter type. Look at OSDataRequestFilterType to see the available params.
   * @see OSDataRequestFilterType
   */
  public function setFilterType(/*OSDataRequestFilterType*/ $filterType) { ... }

  /**
   * Set's the index of the first item to fetch.
   * @param int $first The index of the first item to fetch.
   * @link
http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.DataRequest.PeopleRequestF
ields.html#FIRST
   */
  public function setFirst(/*Number*/ $first) { ... }

  /**
   * Set's the maximum number of items to fetch;
```

```
   * defaults to 20. If set to a larger number, a container may honor the request, or may limit
the number to a container-specified limit of at least 20
   *
   * @param int $max The maximum number of items to fetch.
   * @link
http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.DataRequest.PeopleRequestF
ields.html#MAX
   */
  public function setMax(/*Number*/ $max) { ... }

  /**
   * Set's the profile details array.
   * @param array One array containing the profile details.
   */
  public function setProfileDetails(/*OSPersonField*/ $profileDetails) { ... }

  /**
   * Add's an request for profile detail. Look at OSPersonField to see the available params.
   * @see OSPersonField
   */
  public function addProfileDetails(/*OSPersonField*/ $profileDetail) { ... }
}
```

Listagem 7.25: Código fonte: OSPeopleRequest.class.php

B.2.3.4 - OSPersonRequest

```
<?php
/**
 * This class is intended to make Person request's to the container. Person request's are
classes that extends the abstract class OSPerson naming OSViewer or OSOwner, which allows to
get the viewer and the owner respectively.
 * Basically to use this class you need instantiate a class that extends the abstract class
OSPerson:
 * <code> $this->viewer = new OSViewer('viewer');
 * //or
 * $this->owner = new OSOwner('owner');
 * </code>
 * Then you need to create an OSPersonRequest instance, passing has argument one of the
variables previously created:
 * <code>
 * $this->viewer_request = new OSPersonRequest($this->viewer);
 * //or
 * $this->owner_request = new OSPersonRequest($this->owner);
 * </code>
 * Now that you have created the OSPersonRequest object you need to create an OSDataRequest and
add that created object to it:
 * <code>
 * $data_request = new OSDataRequest('requestPerson');
 * $data_request->addRequest($this->viewer_request);
 * //or
 * $data_request->addRequest($this->owner_request);
 * </code>
 * After adding the OSPersonRequest to the OSDataRequest you need to call the function "send()"
so that this OSDataRequest will be made to the container:
 * <code>
 * $this->data_response = $data_request->send(true);
 * </code>
 * It has been created the request to the container and now you can access to OSViewer or
OSOwner from the view and iterate over those objects in the view:
 * <code>
 * // View
 * <script type="text/JavaSript">
 * function <? echo $data_response->getJSFunction()->getName() ?>(<? echo $data_response-
>getJSFunction()->getArgs() ?>)
 * {
 *   var html = '';
 *   <?php echo $data_response->getData($owner_request); ?>
 *   var owner_name = <?php echo $owner->getName(); ?>
 *   var owner_thumb = <?php echo $owner->getField(OSPersonField::THUMBNAIL_URL); ?>
 *   var owner_profile = <?php echo $owner->getField(OSPersonField::PROFILE_URL); ?>
 *   var owner_nick = <?php echo $owner->getField(OSPersonField::NICKNAME); ?>
 *   html = '<a href="' + owner_profile + '" target="_top" style="float:left">' +'<img src="' +
owner_thumb + '"/></a>' +
```

```
 *     owner_name + ' - '+owner_nick+'</a>';
 *   document.getElementById("owner").innerHTML = html;
 *
 *   <?php echo $data_response->getData($viewer_request); ?>
 *   var viewer_name = <?php echo $viewer->getName(); ?>
 *   var viewer_thumb = <?php echo $viewer->getField(OSPersonField::THUMBNAIL_URL); ?>
 *   var viewer_profile = <?php echo $viewer->getField(OSPersonField::PROFILE_URL); ?>
 *   var viewer_nick = <?php echo $viewer->getField(OSPersonField::NICKNAME); ?>
 *   html = '<a href="' + viewer_profile + '" target="_top" style="float:left">' +'<img src="'
+ viewer_thumb + '"/></a>' +
 *     viewer_name + ' - '+viewer_nick+'</a>';
 *   document.getElementById("viewer").innerHTML = html
 *   <?php echo GadgetsWindow::adjustHeight(); ?>
 *
 * }
 * </script></code>
 *
 * @package    opensocial
 * @subpackage datarequest
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class OSPersonRequest implements OSDataRequestable
{
  /**
  * OSPersonRequest constructor
  * @param OSPerson The OSPerson object
  */
  function __construct(OSPerson $person) { ... }

  public function getPersonRequestParamsVarName() {  }

  /**
   * Get's the OSPerson Object
   * @return OSPerson The object associated to this OSPersonRequest.
   */
  public function getPerson() { ... }

  /**
   * Gets the all the params requested
   * @return array The Request params.
   */
  public function getPersonRequestParams() { ... }

  /**
   * This method returns the var name for profileDetails
   * @return string Profile details var name.
   */
  public function getProfileDetailsVarName()
  { ... }

  /**
   * Gets the profile details
   * @return array The profile details.
   */
  public function getProfileDetails() { ... }

  public function generateDataRequest(OSDataRequest $dataRequest) { ... }

  public function getOptionalKey() { ... }
  /**
   * Set's the sort order. Look at OSDataRequestSortOrder to see the available params.
   * @see OSDataRequestSortOrder
   */
  public function setSortOrder(/*OSDataRequestSortOrder*/ $sortOrder) { ... }
  /**
   * Set's the filter type. Look at OSDataRequestFilterType to see the available params.
   * @see OSDataRequestFilterType
   */
  public function setFilterType(/*OSDataRequestFilterType*/ $filterType) { ... }
  /**
   * Set's the index of the first item to fetch.
   * @param int $first The index of the first item to fetch.
   * @link
http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.DataRequest.PeopleRequestF
```

```
ields.html#FIRST
   */
  public function setFirst(/*Number*/ $first) { ... }
  /**
   * Set's the maximum number of items to fetch;
   * defaults to 20. If set to a larger number, a container may honor the request, or may limit
the number to a container-specified limit of at least 20
   *
   * @param int $max The maximum number of items to fetch.
   * @link
http://code.google.com/apis/opensocial/docs/0.7/reference/opensocial.DataRequest.PeopleRequestF
ields.html#MAX
   */
  public function setMax(/*OSDataRequestFilterType*/ $max) { ... }
  /**
   * Set's the profile details array.
   * @param array One array containing the profile details.
   */
  public function setProfileDetails(/*OSPersonField*/ $profileDetails) { ... }
  /**
   * Add's an request for profile detail. Look at OSPersonField to see the available params.
   * @see OSPersonField
   */
  public function addProfileDetails(/*OSPersonField*/ $profileDetail) { ... }

}
```

Listagem 7.26: Código fonte: OSPersonRequest.class.php

B.2.3.5 - OSFetchActivitiesRequest

```
<?php
class OSFetchActivitiesRequest implements OSDataRequestable
{
  public function __construct($idSpec) { ... }

  public function getActivitiesStreamVarName() { ... }

  public function getIdSpec(){ ... }

  public function generateDataRequest(OSDataRequest $dataRequest) { ... }
  public function getOptionalKey() { ... }
}
```

Listagem 7.27: Código fonte: OSFetchActivitiesRequest.class.php

B.2.3.6 - OSUpdateActivitiesRequest

```
<?php
class OSUpdatePersonAppDataRequest implements OSDataRequestable
{
  public function __construct() { ... }

  public function generateDataRequest(OSDataRequest $dataRequest) { ... }

  public function addPersonAppData(OSPersonAppData $appData) { ... }

  public function getOptionalKey() { ... }

  public function getUpdateAppDataList() {  }
}
```

Listagem 7.28: Código fonte: OSUpdatePersonAppDataRequest.class.php

B.2.3.7 - OSFetchActivitiesRequest

```
<?php
class OSFetchPersonAppDataRequest implements OSDataRequestable
{
  function __construct(OSViewer $person) { ... }

  public function getAppDataListVarName() { ... }

  public function generateDataRequest(OSDataRequest $dataRequest) { ... }
```

```
  public function getOptionalKey() { ... }

  function fetchPersonAppData(OSPersonAppData $appData) { ... }

  public function getPerson() { ... }
}
```

Listagem 7.29: Código fonte: OSFetchPersonAppDataRequest.class.php

B.2.4 - Environment

B.2.4.1 - OSEnvironment

```
<?php
class OSEnvironment implements Instanciatable
{
  /**
   * This is the constructor. To initalize this variable you need to call the method
"initialize()"
   * @param string The name of this OSEnvironment variable
   */
  public function __construct($env_var=null) { ... }

  public function initialized() { ... }

  /**
   * This method will init the environment variable so you can use this class
   * @return string HTML string that initializes this variable
   * @param boolean $initialize By default it initializes the variable
   */
  public function initialize($initialize=true,$appendable=false) { ... }

  /**
   * This method returns the variable name of this Environment class
   * @return string The variable used for this Environment
   */
  public function getVarName(){ ... }

  /**
   * Returns the current domain – for example, "orkut.com" or "myspace.com".
   * @param boolean If the command is appendable
   * @return string Returns the current domain – for example, "orkut.com" or "myspace.com".
   */
  public function getDomainName($appendable=false) { ... }

  /**
   * Returns true if this container supports some OSActivityField
   * @param string $fieldName Look at OSActivityField to see the allowed field types
   * @param boolean If the command is appendable
   * @return string If some Activity supports some $fieldName
   */
  public function supportsActivity(/*OSField*/ $fieldName,$appendable=false) { ... }

  /**
   * Returns true if this container supports some OSActivityMediaItem
   * @param string $fieldName Look at OSActivityField to see the allowed field types
   * @param boolean If the command is appendable
   * @return string If some Activity supports some $fieldName
   */
  public function supportsActivityMediaItem(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsAddress(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsBodyType(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsEmail(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsFilterType(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsMessage(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsMessageType(/*OSField*/ $fieldName,$appendable=false) { ... }
```

```
  public function supportsName(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsOrganization(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsPerson(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsPhone(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsSortOrder(/*OSField*/ $fieldName,$appendable=false) { ... }

  public function supportsURL(/*OSField*/ $fieldName,$appendable=false) { ... }

}
```

Listagem 7.30: Código fonte: OSEnvironment.class.php

B.2.4.2 - OSEnvironment

```
<?php
class OSEnvironmentObjectType
{
  const ENVIRONMENTOBJECTTYPE_CLASS = 'opensocial.Environment.ObjectType';

  const ACTIVITY = 'opensocial.Environment.ObjectType.ACTIVITY';
  const ACTIVITY_MEDIA_ITEM = 'opensocial.Environment.ObjectType.ACTIVITY_MEDIA_ITEM';
  const ADDRESS = 'opensocial.Environment.ObjectType.ADDRESS';
  const BODY_TYPE = 'opensocial.Environment.ObjectType.BODY_TYPE';
  const EMAIL = 'opensocial.Environment.ObjectType.EMAIL';
  const FILTER_TYPE = 'opensocial.Environment.ObjectType.FILTER_TYPE';
  const MESSAGE = 'opensocial.Environment.ObjectType.MESSAGE';
  const MESSAGE_TYPE = 'opensocial.Environment.ObjectType.MESSAGE_TYPE';
  const NAME = 'opensocial.Environment.ObjectType.NAME';
  const ORGANIZATION = 'opensocial.Environment.ObjectType.ORGANIZATION';
  const PERSON = 'opensocial.Environment.ObjectType.PERSON';
  const PHONE = 'opensocial.Environment.ObjectType.PHONE';
  const SORT_ORDER = 'opensocial.Environment.ObjectType.SORT_ORDER';
  const URL = 'opensocial.Environment.ObjectType.URL';
}
```

Listagem 7.31: Código fonte: OSEnvironmentObjectType.class.php

B.2.5 - Message

B.2.5.1 - OSMessage

```
<?php
abstract class OSMessage
{
  /**
  * Esta classe cria um user em determinado contexto de um pedido ao container, com o seu nome
da variável.
  */
  function __construct($var) { ... }

  function getVarName(){ ... }
  function getMessageParamsVarName(){ ... }
  function getMessageParams(){ ... }
  function getRecipientsParamsVarName(){ ... }
  function getRecipientsParams(){ ... }

  public function setTitle($title) { ... }

  public function setBody($body) { ... }

  abstract function getType();
}
```

Listagem 7.32: Código fonte: OSMessage.class.php

B.2.5.2 - OSMessageEmail

```
<?php
```

```
class OSMessageEmail extends OSMessage
{
  function __construct($var) { ... }

  public function getType(){ ... }
}
```

Listagem 7.33: Código fonte: OSMessageEmail.class.php

B.2.5.3 - OSMessageNotification

```
<?php
class OSMessageNotification extends OSMessage
{
  function __construct($var) { ... }

  public function getType(){ ... }
}
```

Listagem 7.34: Código fonte: OSMessageNotification.class.php

B.2.5.4 - OSMessagePublicMessage

```
<?php
class OSMessagePublicMessage extends OSMessage
{
  function __construct($var) { ... }

  public function getType(){ ... }
}
```

Listagem 7.35: Código fonte: OSMessagePublicMessage.class.php

B.2.5.5 - OSMessagePrivateMessage

```
<?php
class OSMessagePrivateMessage extends OSMessage
{
  function __construct($var) { ... }

  public function getType(){ ... }
}
```

Listagem 7.36: Código fonte: OSMessagePrivateMessage.class.php

B.3 - Gadgets namespace

B.3.1 - Classes base

B.3.1.1 - GadgetRequest

```
<?php
/**

 *  This class allows to fetches content from one provided URL and feeds that content into the
callback function.
 *
 *  To create an GadgetRequest we need to call the constructor like this:
 *  <code>
 *  $gadget_request = new
GadgetRequest('gadgetRequest','http://graargh.returnstrue.com/buh/fetchme.php');
 *  </code>
 *  Doing this we create an GadgetRequest Object identified by 'gadgetRequest' and that it will
be made to the 'http://graargh.returnstrue.com/buh/fetchme.php' url. After creating the object
we need to set it's content type like this:
 *  <code>
 *  $gadget_request->setMethod(GadgetMethodType::POST);
 *  </code>
 *  Here we are saying that the request will be as POST to the provided URL.
 *  Next we need to define what will be submited to the container and here we use something
like this:
 *  <code>$gadget_request->setPostData(GadgetsIO::encodeValues('{ data1 : "test", data2 :
```

```
1234566 }',true));
 *  </code>
 *  Now to actually perform the Request we need to call the GadgetsIO class like this:
 *  <code>
 *  $this->gadget_response = GadgetsIO::makeRequest($gadget_request);
 *  echo JSFunction::addJSTags(GadgetsUtil::registerOnLoadHandler($gadget_request->-
>getJSRequestFunction()));
 *  </code>
 *  That it will return an GadgetResponse object that will be used in the view and the last
line will say to the container to execute the this request when the gadget is loaded.
 *
 *  In the view we can use this code:
 *  <code>
 *  //view
 *  <div id="post">post</div>
 *
 *  <?php echo JSFunction::addJSOpenTag($gadget_response->getJSFunction()); ?>
 *    var text = <?php echo $gadget_response->getText(); ?>
 *    document.getElementById('post').innerHTML = text;
 *    <?php echo GadgetsWindow::adjustHeight(); ?>
 *  <?php echo JSFunction::addJSCloseTag($gadget_response->getJSFunction()) ?>
 *  </code>
 *  This code will handle the response and set the div "post" with the content received by the
container.
 * @package    gadgets
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class GadgetRequest
{

  /**
   * This constructor creates an GadgetRequest object with the $id and the $url
   *
   * @param string $id
   * @param string $url
   * @throws GadgetRequest must have an ID and an URL!
   */
  function __construct($id,$url) { ... }

  /**
   * Get's the request JavaSript function
   *
   * @return JSFunction
   */
  public function getJSRequestFunction() { ... }

  /**
   * Get's the response JavaSript function
   *
   * @return JSFunction
   */
  public function getJSResponseFunction() { ... }

  /**
   * Set's the response JavaSript function
   *
   * @return JSFunction
   */
  public function setJSResponseFunction(JSFunction $response_func) { ... }

  /**
   * This method generates this "html" reques
   *
   * @return string The HTML coded generated by this request
   */
  public function generateRequest() { ... }

  /**
   * The method returns the variable name that has the post data content
   *
   * @return string The variable name
   */
  public function getPostDataVar() { ... }
```

```
/**
 * Set's the authorization type of this request
 *
 * @param string $type One of the constant GadgetAuthorizationType values
 * @see GadgetAuthorizationType
 */
public function setAuthorization($type) { ... }

/**
 * Get's the authorization type of this request
 *
 * @return string One of the constant GadgetAuthorizationType values
 * @see GadgetAuthorizationType
 */
public function getAuthorization() { ... }

/**
 * Get's the content type of this request
 *
 * @return string One of the constant GadgetContentType values
 * @see GadgetContentType
 */
public function getContentType() { ... }

/**
 * Set's the content type of this request
 *
 * @param string $type One of the constant GadgetContentType values
 * @see GadgetContentType
 */
public function setContentType($type) { ... }

/**
 * If the content is a feed, whether to fetch summaries for that feed; defaults to false.
 *
 * @param boolean $bool
 */
public function setGetSummaries($bool) { ... }

/**
 * Get's the value of the GET_SUMMARIES object
 *
 * @return boolean
 */
public function getGetSummaries() { ... }
/**
 * The HTTP headers to send to the URL; defaults to null. Specified as a Map.<String,String>.
 *
 * @param string $headers
 */
public function setHeaders($headers) { ... }

/**
 * Get's the HEADERS value
 *
 * @return string
 */
public function getHeaders() { ... }
/**
 * The method to use when fetching content from the URL; defaults to GadgetMethodType::GET
 *
 * @param string $type One of the constant GadgetMethodType values
 * @see GadgetMethodType
 */
public function setMethod($type) { ... }
/**
 * Get's the Method type of this request
 *
 * @return string One of the constant GadgetMethodType values
 * @see GadgetMethodType
 */
public function getMethod() { ... }
/**
 * If the content is a feed, the number of entries to fetch; defaults to 3. Specified as a
```

```
Number.
    *
    * @param int $numEntries
    */
  public function setNumEntries($numEntries) { ... }

  /**
    * Get's the number of entries to fetch
    *
    * @return int
    */
  public function getNumEntries() { ... }
  /**
    * The data to send to the URL using the POST method; defaults to null. Specified as a
String.
    *
    * @param string $postData
    */
  public function setPostData($postData) { ... }

  /**
    * Get's the data that will be posted
    *
    * @return string
    */
  public function getPostData() { ... }

  /**
    * Get's the URL where the content is located
    *
    * @return string The URL where the content is located
    */
  public function getURL() { ... }
  /**
    * Set's the URL where the content is located
    *
    * @param string $url The URL where the content is located
    */
  public function setURL($url) { ... }

  /**
    * Get's the URL's variable name
    *
    * @return string URL varible name
    */
  public function getURLVarName() { ... }

  /**
    * Set's the URL's variable name
    *
    * @param string URL varible name
    */
  public function setURLVarName($url_var) { ... }

  /**
    * Get's the reques params variable name
    *
    * @return string Params variable name
    */
  public function getParamsVarName() { ... }

  /**
    * Set's the params variable name
    *
    * @param string Params varible name
    */
  public function setParamsVarName($params_var) { ... }


  protected function resetParams() { ... }
}
```

Listagem 7.37: Código fonte: GadgetRequest.class.php

B.3.1.2 - DomGadgetRequest

```
<?php
class DomGadgetRequest extends GadgetRequest
{
  public function __construct($id,$url) { ... }
}
```

Listagem 7.38: Código fonte: DomGadgetRequest.class.php

B.3.1.3 - FeedGadgetRequest

```
<?php
class FeedGadgetRequest extends GadgetRequest
{
  public function __construct($id,$url) { ... }
}
```

Listagem 7.39: Código fonte: FeedGadgetRequest.class.php

B.3.1.4 - JsonGadgetRequest

```
<?php
class JsonGadgetRequest extends GadgetRequest
{
  public function __construct($id,$url) { ... }
}
```

Listagem 7.40: Código fonte: JsonGadgetRequest.class.php

B.3.1.5 - TextGadgetRequest

```
<?php
class TextGadgetRequest extends GadgetRequest
{
  public function __construct($id,$url) { ... }
}
```

Listagem 7.41: Código fonte: TextGadgetRequest.class.php

B.3.1.6 - GadgetsIO

```
<?php
class GadgetsIO
{
  /**
   * This static method generates the gadget request and append it to the view if $inject=true
   * @param GadgetRequest $gadgetRequest The gadget request
   * @param boolean $inject True, the gadget will be appended to the view
   * @return GadgetResponse
   */
  public static function makeRequest(GadgetRequest $gadgetRequest,$inject = true) { ... }

  public static function encodeValues($content,$appendable=false) { ... }

  public static function getProxyUrl($url,$appendable=false) { ... }

}
```

Listagem 7.42: Código fonte: GadgetsIO.class.php

B.3.1.7 - GadgetsUtil

```
<?php
class GadgetsUtil
{
  /**
   * Escapes the input using html entities to make it safer.
   */
  public static function escapeString($text,$is_JavaSript=false,$appendable=false) { ... }

  /**
   * Gets the feature parameters.
   */
```

```
  public static function getFeatureParameters($feature,$appendable=false) { ... }

  /**
   * Gets the URL parameters.
   */
  public static function getUrlParameters($appendable=false) { ... }

  /**
   * Returns whether the current feature is supported
   */
  public static function hasFeature($feature,$appendable=false) { ... }

  /**
   * Reverses escapeString.
   */
  public static function unescapeString($str,$is_JavaSript=false,$appendable=false) { ... }

  /**
   * Registers an onload handler.
   */
  public static function registerOnLoadHandler(JSFunction $callback,$appendable=false) { ... }

  /**
   * Creates a closure that is suitable for passing as a callback. Any number of arguments may
be passed to the callback; they will be received in the order they are passed in.
   */
  public static function makeClosure($scope, $callback, $var_args,$appendable=false) { ... }

}
```

Listagem 7.43: Código fonte: GadgetsUtil.class.php

B.3.1.8 - GadgetsPrefs

```
<?php
class GadgetsPrefs
{
  public function __construct($var_name) { ... }

  public function initialize(){ ... }

  public function __toString(){ ... }

  public function getVarName(){ ... }

  /**
   *  Retrieves an array preference.
   */
  public function getArray($key,$appendable){  }

  /**
  *  Retrieves a boolean preference. Boolean getBool(key)
  */
  public function getBool($key,$appendable){ ... }

  /**
   * Gets the current country, returned as ISO 3166-1 alpha-2 code.
   */
  public function getCountry($appendable){ ... }
  /**
   * Retrieves a floating-point preference.
   */
  public function getFloat($key,$appendable){ ... }

  /**
   * Retrieves an integer preference.
   */
  public function getInt($key,$appendable){ ... }

  /**
   * Gets the current language the gadget should use when rendering, returned as a ISO 639-1
language code.
   */
  public function getLang($appendable){ ... }
```

```
  /**
   * Gets the module id for the current instance.
   */
  public function getModuleId($appendable){ ... }

  /**
   * Fetches an unformatted message.
   */
  public function getMsg($key,$appendable){ ... }

  /**
   * Returns a message value with the positional argument opt_subst in place if it is provided
or the provided example value if it is not, or the empty  public function  if the message is
not found.
   */
  public function getMsgFormatted($key, $opt_subst,$appendable){ ... }

  /**
   * Retrieves a  public function  preference.
   */
  public function getString($key,$appendable){ ... }

  /**
   * Stores a preference.
   */
  public function set($key, $val,$appendable){ ... }

  /**
   * Stores an array preference.
   */
  public function setArray($key, $val,$appendable){ ... }
}
```

Listagem 7.44: Código fonte: GadgetsPrefs.class.php

B.3.1.9 - ModulePrefs

```
<?php
/**
 * This class is used for setting ModulePrefs from code. You can also use the "app.yml" file to
do this.
 *
 * {@link http://code.google.com/apis/gadgets/docs/reference.html#Moduleprefs_Ref The
<ModulePrefs> section in the XML file specifies characteristics of the gadget, such as title,
author, preferred sizing, and so on. }
 * @package    gadgets
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class ModulePrefs
{
  /**
   * This Method allows you to set the aplication Title.
   * "Optional string that provides the title of the gadget. This title is displayed in the
gadget title bar on iGoogle. If this string contains user preference substitution variables and
you are planning to submit your gadget to the content directory, you should also provide a
directory_title for directory display."
   * @param string The title
   */
  public static function setTitle($title) { ... }

  /**
   * This Method allows you to change the thumbnail
   *
   * "ptional string that gives the URL of a gadget thumbnail. This must be an image on a
public web site that is not blocked by robots.txt. PNG is the preferred format, though GIF and
JPG are also acceptable. Gadget thumbnails should be 120x60 pixels. For more discussion of
thumbnail guidelines, see Publishing to the Content Directory."
   * @param string The thumbnail of the app
   */
  public static function setThumbnail($thumbnail) { ... }

  /**
   * This Method allows you to change the Authors email
```

```
   *
   * Optional string that provides the gadget author's email address. You can use any email
system, but you should not use a personal email address because of spam. One approach is to use
an email address of the form helensmith.feedback+coolgadget@gmail.com in your gadget spec.
   * @param string The author email of the app
   */
  public static function setAuthorEmail($author_email) { ... }

  /**
   * This Method allows you to set the aplications Author.
   *
   * "Optional string that lists the author of the gadget."
   * @param string The author of the app
   */
  public static function setAuthor($author) { ... }

  /**
   * This Method allows you to set the aplication Description.
   *
   * "Optional string that describes the gadget."
   * @param string The description
   */
  public static function setDescription($description) { ... }

  /**
   * This Method allows you to set any key value from ModulePrefs.
   *
   * @param string $key
   * @param string $value
   */
  public static function set($key,$value) { ... }
}
```

Listagem 7.45: Código fonte: ModulePrefs.class.php

B.3.1.10 - GadgetsJson

```
<?php
class GadgetsJson
{
  /**
   * Parses a JSON string, producing a JavaSript value.
   */
  public static function parse($text,$appendable=false) { ... }

  /**
   * Converts a JavaSript value to a JSON string.
   */
  public static function stringify($v,$appendable=false) { ... }

}
```

Listagem 7.46: Código fonte: GadgetsJson.class.php

B.3.2 - Feature-specific

B.3.2.1 - GadgetsFlash

```
<?php
class GadgetsFlash
{
  /**
   * Detects Flash Player and its major version.
   */
  public static function getMajorVersion($appendable=false) { ... }

  private static function executeFlash($type,JSVariable $swfContainer,FlashObject $flashObject,
$appendable) { ... }
  /**
   * Injects a Flash file into the DOM tree.
   *
   * @param JSVariable $swfContainer The JavaSript variable the will hold the div that will be
updated
```

```
   * @param FlashObject $flashObject The Flash object that will be loaded
   * @param boolean $appendable If this method will be appended or not
   * @return string HTML string
   */
  public static function embedFlash(JSVariable $swfContainer,FlashObject $flashObject,
$appendable=false) { ... }

  public static function embedCachedFlash(JSVariable $swfContainer,FlashObject $flashObject,
$appendable=false) { ... }
}
```

Listagem 7.47: Código fonte: GadgetsFlash.class.php

B.3.2.2 - GadgetsMiniMessage

```
<?php
/**
 * To use MiniMessages you should look at GadgetsMiniMessage.class. You can find OpenSocial
documentation here
[http://code.google.com/apis/opensocial/docs/0.7/reference/gadgets.MiniMessage.html Class
gadgets.MiniMessage]. This allows you to send mini messages to the user that is using the
application. You can create messages, remove them pragmatically or you can create dismissible
messages.
 *
 * Here I'm going to explain how you could use MiniMessages in the sfOpenSocialPlugin.
 * You can choose to use this class in the view or in the action. In this example I'm going to
create a dismissible MiniMessage Object in the action and render it in the view. You can find
the complete code under "sfOpenSocialPlugin/modules/application"
 * <code>
 * //action
 * $dismissibleMessage = new GadgetsMiniMessage('dismissibleMessage');
 * $this->dismissibleMessageFunction = new JSFunction('dismissibleMessageFunction');
 *
 * $content = $dismissibleMessage->initialize();
 * $content .= $dismissibleMessage->createDismissibleMessage('<div>You can dismiss this
message!</div>');
 * $this->dismissibleMessageFunction->setContent($content);
 * echo JSFunction::addJSTags($this->dismissibleMessageFunction);
 * </code>
 * Here I've created an GadgetsMiniMessage object and also an object representing an JavaSript
function. This was made, because I wanted to do all the logic in the action.
 * The first thing to do when using GadgetsMiniMessage class is to initialize the object. If
you don't do this, you will receive an sfException when using any other method. The $content
variable is used to gather the content for the JavaSript function, but if we wanted to do this
part in the view, just doing "echo" it would suffice.
 * To make use of this messages, I've created the following view:
 * <code>
 * <?php use_helper('OpenSocial'); ?>
 *
 * <div id="MiniMessage"></div>
 * <?php echo add_jsfunction_anchor('Create Dismissible Message',$dismissibleMessageFunction) ?
>
 * <?php echo add_jsfunction_anchor('Create Static Message',$staticMessageFunction) ?>
 * <?php echo add_jsfunction_anchor('Create Timer Message',$timerMessageFunction) ?>
 * <?php echo add_jsfunction_anchor('Dismiss Static Timer Message',
$dismissStaticMessageFunction) ?>
 * </code>
 * In the view I've created an anchor any of the JavaSript functions that were created. This
code will also include the "<Require feature="minimessage"/>" in the gadget
 *
 * @package    gadgets
 * @subpackage feature
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class GadgetsMiniMessage implements Instanciatable
{
  public function __construct($var,$opt_moduleId=null, $opt_container=null) { ... }

  public function initialized() { ... }

  public function initialize($initialize=true,$appendable=false) { ... }

  /**
   * This method returns the variable name of this MiniMessage class
```

```
   * @return string The variable used for this MiniMessage
   */
  public function getVarName(){ ... }

  /**
   * Creates a dismissible message with an [x] icon that allows users to dismiss the message.
When the message is dismissed, it is removed from the DOM and the optional callback function,
if defined, is called.
   *
   * @param string $message The message as an HTML string or DOM element
   * @param JSFunction $opt_callback Optional callback function to be called when the message
is dismissed
   * @param boolean $appendable
   * @return string HTML element of the created message
   */
  public function createDismissibleMessage($message,$opt_callback=null,$appendable=false) { ...
}

  /**
   * Creates a static message that can only be dismissed programmatically (by calling
dismissMessage()).
   *
   * @param string $message The message as an HTML string or DOM element
   * @param boolean $appendable
   * @return string HTML element of the created message
   */
  public function createStaticMessage($message,$element = null, $appendable=false) { ... }

  /**
   * Creates a message that displays for the specified number of seconds. When the timer
expires, the message is dismissed and the optional callback function is executed.
   *
   * @param string $message The message as an HTML string or DOM element
   * @param int $seconds Number of seconds to wait before dismissing the message
   * @param JSFunction $opt_callback Optional callback function to be called when the message
is dismissed
   * @param boolean $appendable
   * @return string HTML element of the created message
   */
  public function createTimerMessage($message, $seconds, $opt_callback=null, $appendable=false)
{ ... }

  /**
   * Dismisses the specified message.
   *
   * @param string $div Element message  - HTML element of the message to remove
   * @param boolean $appendable
   * @return string
   */
  public function dismissMessage($div, $appendable=false) { ... }
}
```

Listagem 7.48: Código fonte: GadgetsMiniMessage.class.php

B.3.2.3 - GadgetsTab

```
<?php
/**
 * This class correspond to the class
[http://code.google.com/apis/opensocial/docs/0.7/reference/gadgets.Tab.html gadgets.Tab] from
the OpenSocial API, with the difference that this one has more specific information.
 */
class GadgetsTab
{
  /**
   * By default the name corresponds to the contentContainer.
   *
   * @param string $name
   * @param boolean $fetch_mode If true this means that this variable was fetched by the
container
   */
  public function __construct($name,$fetch_mode=false) { ... }

  /**
```

```
   * Throws sfException if not in Fetch Mode
   * @throws sfException If not in Fetch Mode
   */
  private function assertFetchMode() { ... }

  /**
   * Throws sfException if in Fetch Mode
   * @throws sfException If in Fetch Mode
   */
  private function assertNotFetchMode() { ... }

  public function getVarName() { ... }

  /**
   * Returns the defined name of this Tab
   * @return string Returns the defined name of this Tab
   */
  public function getName() {  }

  /**
   * The name of the varible that holds the params
   *
   * @return string The name of the varible that holds the params
   */
  public function getParamsVar() { ... }

  /**
   * Returns the parameters of this Tab
   * @return string The params
   */
  public function getParams() { ... }

  public function setContentContainerElement($elementId) { ... }

  public function setContentContainer($contentContainer) { ... }
  public function setCallback(JSFunction $callback) { ... }
  public function setTooltip($tooltip) { ... }
  public function setIndex($index) { ... }
  /**
   * Returns the HTML element where the tab content is rendered.
   * @return Element Returns the HTML element where the tab content is rendered.
   */
  public function getTabContentContainer($appendable=false) { ... }
  /**
   * Returns the callback function that is executed when the tab is selected.
   *
   * @return Function Returns the callback function that is executed when the tab is selected.
   */
  public function getTabCallback($appendable=false) { ... }
  /**
   * Returns the label of the tab as a string (may contain HTML).
   *
   * @return String Returns the label of the tab as a string (may contain HTML).
   */
  public function getTabName($appendable=false) { ... }
  /**
   * Returns the tab's index.
   *
   * @return Number Returns the tab's index.
   */
  public function getTabIndex($appendable=false) { ... }
  /**
   * Returns the HTML element that contains the tab's label.
   *
   * @return Element Returns the HTML element that contains the tab's label.
   */
  public function getTabNameContainer($appendable=false) { ... }
}
```

Listagem 7.49: Código fonte: GadgetsTab.class.php

B.3.2.4 - GadgetsTabSet

```
<?php
```

```
/**
 * @package    gadgets
 * @subpackage feature
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class GadgetsTabSet implements Instanciatable
{
  public function __construct($var, $div=null) { ... }

  public function setOptModuleId($opt_moduleId) { ... }
  public function setOptDefaultTab($opt_defaultTab) { ... }
  public function setOptContainer($opt_container) { ... }

  public function getVarName() { ... }

  /**
   * This method will only initialize the variable used by this TabSet. This method is useful
when we want to make the variable global
   *
   */
  public function initializeVariable($appendable=false) { ... }

  public function initialize($initialize=true,$appendable=false) { ... }

  public function initialized() { ... }

  /**
   * Throws sfException if it's not initialized
   * @throws sfException If it's not initialized
   */
  private function assertInitialized() { ... }

  /**
   * Adds a new tab based on the name-value pairs specified in opt_params.
   * @param boolean $jsTabName This should be set to true if the tabs name should be handled as
a JavaSript variable
   * @return string HTML string
   */
  public function addTab(GadgetsTab $tab,$jsTabName=false) { ... }

  /**
   * Returns the currently selected tab object.
   *
   * @return GadgetsTab Returns the currently selected tab object.
   */
  public function getSelectedTab($initialize=true,$appendable=false) { ... }

  /**
   * Sets the alignment of tabs. Tabs are center-aligned by default.
   *
   * @param string $align
   * @param integer $opt_offset
   */
  public function alignTabs($align, $opt_offset=null,$appendable=false) { ... }

  /**
   * Sets the alignment of tabs to ALIGN_LEFT
   *
   * @param integer $opt_offset
   */
  public function alignTabsLeft($opt_offset=null,$appendable=false) { ... }

  /**
   * Sets the alignment of tabs to ALIGN_RIGHT
   *
   * @param integer $opt_offset
   */
  public function alignTabsRight($opt_offset=null,$appendable=false) { ... }

  /**
   * Sets the alignment of tabs to ALIGN_CENTER
   *
   * @param integer $opt_offset
   */
```

```
  public function alignTabsCenter($opt_offset=null,$appendable=false) { ... }

  /**
   * Shows or hides tabs and all associated content.
   *
   * @param boolean $display true to show tabs; false to hide tabs.
   */
  public function displayTabs($display=null) { ... }

  /**
   * This method is meant to be used inside a for loop. To use this method you should do
something like this inside a JavaSript function in the view:
   * <code>
   *  for(<?php $tab = $tabset->getEachTab(); ?>){
   *    <?php echo $tab->getTabName(); ?>
   * </code>
   * This method will generate the necessary content for the for loop and will return an
GadgetsTab object that you can use inside the for loop.
   * This method can only be used in the view, since it echoes the for loop content to it.
   * @param integer $tabs_length This value is useful when removing all tabs
   * @return GadgetsTab
   */
  public function getEachTab($start_idx=0,$tabs_length=null,$inc_idx=true) { ... }

  /**
   * Removes a tab at tabIndex and all of its associated content.
   *
   * @param Integer $tabIndex
   * @return HTML string
   */
  public function removeTabByIndex($tabIndex) { ... }

  /**
   * Removes a tab and all of its associated content.
   *
   * @param Integer $tabIndex
   * @return HTML string
   */
  public function removeTab(GadgetsTab $tab) { ... }

  /**
   * Returns the tab headers container element.
   *
   * @param unknown_type $appendable
   * @return Returns the tab headers container element.
   */
  public function getHeaderContainer($appendable=false) { ... }

  /**
   * Selects the tab at tabIndex and fires the tab's callback function if it exists. If the tab
is already selected, the callback is not fired.
   *
   * @param unknown_type $tabIndex
   * @return unknown
   */
  public function setSelectedTabByIndex($tabIndex) { ... }

  /**
   * Selects the tab and fires the tab's callback function if it exists. If the tab is already
selected, the callback is not fired.
   *
   * @param unknown_type $tabIndex
   * @return unknown
   */
  public function setSelectedTab(GadgetsTab $tab) { ... }

  public function swapTabsByIndex($tabIndex1,$tabIndex2) { ... }
}
```

Listagem 7.50: Código fonte: GadgetsTabSet.class.php

B.3.2.5 - GadgetsViews

```
<?php
```

```
/**
 * This is the static class that is responsible to handle views in the OpenSocial container
 */
class GadgetsViews
{
  /**
   * Returns the current view.
   * @param boolean $initialize if the variable should be initialized
   * @return GadgetsView Returns the current view.
   */
  public static function getCurrentView($initialize=false) { ... }

  /**
   * Attempts to navigate to this gadget in a different view.
   * @param GadgetsView $view
   * @return string HTML code
   */
  public static function requestNavigateTo(GadgetsView $view,$opt_params=null,
$appendable=false) { ... }

  /**
   * Returns a map of all the supported views.
   *
   * @param boolean $initialize
   * @param boolean $appendable
   * @return GadgetsSupportedViews
   */
  public static function getSupportedViews($initialize=false,$appendable=false) { ... }

  /**
   * Returns the parameters passed into this gadget for this view.
   *
   * @param boolean $appendable
   * @return Map.<String, String>
   */
  public static function getParams($appendable=false)
  { ... }
}
```

Listagem 7.51: Código fonte: GadgetsViews.class.php

B.3.2.6 - GadgetsView

```
<?php
class GadgetsView
{
  public function __construct(JSVariable $variable) { ... }

  public function isInit(){ ... }

  public function init() { ... }

  /**
   * return the variable used
   *
   * @return JSVariable variable used
   */
  public function getVariable() { ... }

  /**
   * Returns the name of this view.
   *
   * @param boolean $appendable
   * @return string HTML content
   */
  public function getName($appendable=false) { ... }

  /**
   * Returns true if the gadget is the only visible gadget in this view. On a canvas page or in
maximize mode this is most likely true; on a profile page or in dashboard mode, it is most
likely false.
   *
   * @param boolean $appendable
   * @return boolean True if the gadget is the only visible gadget; otherwise, false
```

```
   */
  public function isOnlyVisibleGadget($appendable=false) { ... }
}
```

Listagem 7.52: Código fonte: GadgetsView.class.php

B.3.2.7 - GadgetsSkins

```
<?php
/**
 * To use OpenSocial skins you should look at GadgetsSkins.class. You can find OpenSocial
documentation here
[http://code.google.com/apis/opensocial/docs/0.7/reference/gadgets.skins.html Static Class
gadgets.skins]. This class provides operations for getting display information about the
currently shown skin.
 *
 * Here I'm going to explain how you could use Skins in the sfOpenSocialPlugin. You can choose
to use this class in the view or in the action, but it's more straight forward to use it in the
view. So in this example I'm going to use just the view to setup the gadget skin. You can find
the complete code under "sfOpenSocialPlugin/modules/application"
 * <code>
 * //view
 *
 * <script type="text/JavaSript">
 * function setSkin() {
 *   <?php echo GadgetsSkins::loadSkin(); ?>
 * }
 * </script>
 * <?php echo JSFunction::addJSTags(GadgetsUtil::registerOnLoadHandler(new
JSFunction('setSkin'))); ?>
 * </code>
 * Doing this will generate all the code to import the skin from the container. This code will
also include the "<Require feature="skins"/>" in the gadget
 *
 * @package    gadgets
 * @subpackage feature
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */
class GadgetsSkins
{
  private static function getProperty($property,$appendable=false) { ... }

  public static function loadSkin() { ... }

  /**
   * Get the color that anchor tags should use.
   *
   * @param boolean $appendable If it's appendable or not
   * @return string HTML string that will be appended in the view
   */
  public static function getAnchorColor($appendable=false) { ... }

  /**
   * Get the color of the background of the gadget.
   *
   * @param boolean $appendable If it's appendable or not
   * @return string HTML string that will be appended in the view
   */
  public static function getBackgroundColor($appendable=false) { ... }

  /**
   * Get an image to use in the background of the gadget.
   *
   * @param boolean $appendable If it's appendable or not
   * @return string HTML string that will be appended in the view
   */
  public static function getBackgroundImage($appendable=false) { ... }

  public static function getBackgroundRepeat($appendable=false) { ... }

  /**
   * Get the color that the main font should use.
   *
   * @param boolean $appendable If it's appendable or not
```

```
   * @return string HTML string that will be appended in the view
   */
  public static function getFontColor($appendable=false) { ... }
}
```

Listagem 7.53: Código fonte: GadgetsSkins.class.php

B.3.2.8 - GadgetsWindow

```
<?php
class GadgetsWindow
{
  /**
   * Adjusts the gadget height
   */
  public static function adjustHeight($opt_height=null,$appendable=null) { ... }

  /**
   * Sets the gadget title.
   */
  public static function setTitle($title,$appendable) { ... }

  /**
   * Detects the inner dimensions of a frame. See:
http://www.quirksmode.org/viewport/compatibility.html for more information.
   */
  public static function getViewportDimensions($appendable) { ... }
}
```

Listagem 7.54: Código fonte: GadgetsWindow.class.php

B.4 - OpenSocialHelper

```
<?php
/**
 * OpenSocialHelper is intended to ease the sfOpenSocialPlugin use. With this helper you can
include the gadget content's, JavaSript and CSS. Usually the symfony framework allows to
include css and JavaSript from the file "view.yml", in the head HTML section. Or application
don't use the default html layout, but an gadget layout( look at HowToConfigure for more
information) and so we need to include them in the content. This should be fixed in a future
version, but for now to include a custom stylesheet or external JavaSript file you should use
our method's in the OpenSocialHelper.
 *
 * @package    helper
 * @author     Daniel Botelho <botelho.daniel@gmail.com>
 */

/**
 * This method allow to include a Gadget content into the layout
 *
 * @param string $content The content
 * @return string HTML Content code
 */
function include_content($content,$type=null,$href=null,$cdata=null) { ... }

/**
 * This method is used to include JavaSript code located at "web/js"
 *
 * @param string $url The name of the JavaSript file located at "web/js"
 * @return string HTML code
 */
function include_js($src) { ... }
/**
 * This method is used to include remote JavaSript content
 *
 * @param string $url The JavaSript URL
 * @param boolean $nl If it should append a newline or not
 * @return string HTML code
 */
function include_remote_js($src,$nl=true) { ... }

/**
 * This method is used to include CSS code located at "web/css"
 *
```

```
 * @param string $url The name of the CSS file located at "web/css"
 * @return string HTML code
 */
function include_css($stylesheet) { ... }
/**
 * This method is used to include remote CSS content
 *
 * @param string $url The Content Style Sheet URL
 * @param boolean $nl If it should append a newline or not
 * @return string HTML code
 */
function include_remote_css($url,$nl=true) { ... }

/**
 * This method creates a link to an specified JSFunction.
 *
 * @param string $link The description of the link
 * @param JSFuntion $jsf The JavaSript function
 * @param string $args Arguments represented as a string example: "arg1,agr2"
 * @return string HTML code
 */
function add_jsfunction_anchor($link,JSFunction $jsf,$args=null) { ... }
```

Listagem 7.55: Código fonte: OpenSocialHelper.php

B.5 - sfPakeOpenSocial

```
<?php
pake_desc('Create an "Hello world" OpenSocial application');
pake_task('init-opensocial-app', 'project_exists');

function loadOpensocialAppDirs($rootDir) { ... }

function loadOpensocialAppModuleDirs($rootDir) { ... }

function writeToFile($file, $data) { ... }

function fixModuleNames($moduleDir,$moduleName) { ... }

function enable_sampleapp($applicationDir,$sampleAppName) { ... }

function run_init_opensocial_app($task, $args) { ... }

function os_copy_app_data($osTemplatesDir,$appTemplatesDir) { ... }

function os_copy_app_module_data($osTemplatesDir,$appTemplatesDir) { ... }


/**
 * This code adapted from "sfPakeManagerPluginPlugin"
 *
 * @param unknown_type $srcdir
 * @param unknown_type $dstdir
 * @return unknown
 */
function os_copy_dir($srcdir, $dstdir) { ... }
?>
```

Listagem 7.56: Código fonte: sfPakeOpenSocial.php

C - Código fonte da aplicação OpenSocial do Palco Principal

C.1 - palcoprincipal/config/

```
all:
  opensocial:

    base_url: http://www.imusicfactory.com/opensocial/web/
    api_url: http://www.imusicfactory.com/opensocial/web/index.php/api/

    api_version: 0.7

    module_prefs: #ref: http://code.google.com/apis/gadgets/docs/reference.html#Moduleprefs_Ref
      title: Palco Principal
      description: "Let your friends see your favorite music, your popular photos and your
friends from Palco! If you have a band you can have your friend bands, photos from your albums
and concerts and even share your best music with others. If you're not registered in Palco, you
can also let your friends listen to random music from any style that is available."
      author: Daniel Botelho
      author_email: botelho.daniel@gmail.com
      thumbnail: http://www.palcoprincipal.com/images/logo_opensocial_pp.png

    locale:
      - lang: pt
        country: ALL
        messages: http://www.imusicfactory.com/opensocial/web/locale/pt_ALL.xml
      - #lang: all
        #country: ALL
        messages: http://www.imusicfactory.com/opensocial/web/locale/all_ALL.xml

    requires:
      - feature: opensocial-0.7


    user_prefs:

    content:
      type: html
```

Listagem 7.57: Código fonte: palcoprincipal/config/app.yml

```
rendering: ~
web_debug: ~
security:  ~

# generally, you will want to insert your own filters here
oS:
  class: oSFilter

mySpace:
  class: mySpaceFilter

netlog:
  class: removePalcoLinksFilter

cache:     ~
common:    ~
flash:     ~
execution: ~
```

Listagem 7.58: Código fonte: palcoprincipal/config/filters.yml

C.2 - palcoprincipal/lib/

```
<?php

class Misc
{
  public static function use_helper($helperName) { ... }
}
```

Listagem 7.59: Código fonte: palcoprincipal/lib/Misc.php

```
<?php
```

```
/**
 * Esta class corresponde a utilizador do palco principal.
 *
 */
class PalcoUser
{

  public function __construct(OSOwner $owner) { ... }
  /**
   * @return OSOwner O OSOwner
   */
  public function getOwner() { ... }

  /**
   * inicia todas as variáveis
   */
  public function initAll() { ... }

  public function unsetAll() { ... }

  /**
   * Inicia a variavel
   * @return string A string para inicializar a variavel
   * @param string $value Se quiser atribuir um valor ao inicializar a variavel
   */
  public function initFlashPlaylistURL($value=null) { ... }
  public function initUsername($value=null) { ... }
  public function initName($value=null) { ... }
  public function initStyle($value=null) { ... }
  public function initType($value=null) { ... }
  public function initURL($value=null) { ... }
  public function initPhotoURL($value=null) { ... }
  public function initSlug($value=null) { ... }

  /**
   *
   * @return string Retorna a FlashPlaylistURL
   * @throws NotInitalizedException Caso a variavel não tenha sido inicalizada
   */
  public function getFlashPlaylistURL($appendable=false) { ... }
  public function getUsername($appendable=false) { ... }
  public function getName($appendable=false) { ... }
  public function getStyle($appendable=false) { ... }
  public function getType($appendable=false) { ... }
  public function getURL($appendable=false) { ... }
  public function getPhotoURL($appendable=false) { ... }
  public function getSlug($appendable=false) { ... }
  /**
   * @param string $url O endereço da Playlist
   * @param boolean $init Se deve ou não inicializar a variavel
   * @return string Retorna a FlashPlaylistURL
   */
  public function setFlashPlaylistURL($value, $init=false) { ... }
  public function setUsername($value, $init=false) { ... }
  public function setName($value, $init=false) { ... }
  public function setStyle($value, $init=false) { ... }
  public function setType($value, $init=false) { ... }
  public function setURL($value, $init=false) { ... }
  public function setPhotoURL($value, $init=false) { ... }
  public function setSlug($value, $init=false) { ... }
}
```

Listagem 7.60: Código fonte: palcoprincipal/lib/PalcoUser.php

```
<?php
function loading_content($message) { ... }
function refresh_content($method,$message) { ... }
function content_loaded() { ... }
function set_content($update_div,$message_txt,$js_div=false,$js_message=false,
$appendable=false) { ... }
function append_content($update_div,$message,$js_var=false) { ... }
function error_content($update_div,$message,$js_var=false) { ... }
function empty_content($update_div,$message,$js_var=false) { ... }
?>
```

Listagem 7.61: Código fonte: palcoprincipal/lib/helper/PalcoPrincipalHelper.php

C.3 - Módulo API

```
<?php
Misc::use_helper('Text');

class apiActions extends sfActions
{
  public function preExecute() { ... }
 public function executeIndex() { ... }
 public function executeGetContainerUsersByFlashURL() { ... }
 public function executeGetTopDedicatedMusicsByContainer() { ... }
 public function executeGetTopMusicsByContainer() { ... }
 public function executeGetAccountInfo() { ... }
 public function executeUpdateAccountInfo() { ... }
  public function executeGetStyles() { ... }
  public function executeError404() { ... }
  public function executeGetPlaylistByStyle() { ... }
 public function executeGetBandStats() { ... }
  public function executeGetBandFans() { ... }
 public function executeGetFriends() { ... }
 public function executeGetPhotos() { ... }
  public function executeGetProfileMusics() { ... }
 public function executeDelProfileMusic() { ... }
 public function executeDelDedicatedMusic() { ... }
 public function executeStrip() { ... }
  public function executeAddDedicateMusic() { ... }
 public function executeGetDedicatedMusics() { ... }
 public function executeAddProfileMusic() { ... }
 public function executeGetPlaylist() { ... }
  public function executeRegister() { ... }
  public function executeLogin() { ... }
}
```

Listagem 7.62: Código fonte: palcoprincipal/modules/api/actions/actions.class.php

```
rendering: ~
web_debug: ~
security:  ~

# generally, you will want to insert your own filters here
oS:
  enabled: off

mySpace:
  enabled: off

netlog:
  enabled: off

cache:     ~
common:    ~
flash:     ~
execution: ~
```

Listagem 7.63: Código fonte: palcoprincipal/modules/api/config/filters.yml

C.4 - Módulo Palco

```
<?php

class palcoActions extends sfActions
{
  public function executeIndex() { ... }
  public function executeHi5() { ... }
  public function executeOrkut() { ... }
  public function executeNetlog() { ... }
  public function executeMySpace() { ... }
}
```

Listagem 7.64: Código fonte: palcoprincipal/modules/palco/actions/actions.class.php

```
<?php
class palcoComponents extends sfComponents{
  public function executeInit() { ... }
  public function executeMoreMusics() { ... }
  public function executeDedicateMusic() { ... }
```

```
  public function executeAuthenticate() { ... }
  public function executeRegister() { ... }
  public function executeConfigure() { ... }
  public function executeViewStylePaylist() { ... }
  public function executeAddMyPlayList() { ... }
  public function executeAddMyFriends() { ... }
  public function executeBandFans() { ... }
  public function executeBandStatistics() { ... }
  public function executeTopMusicsByContainer() { ... }
  public function executeTopDedicatedMusicsByContainer() { ... }
  public function executeProfileMusics() { ... }
  public function executeAddMyPhotos() { ... }
  public function executeNotifyOwner() { ... }
  public function executeMySpaceAuthenticate() { ... }
  public function executeMySpaceInit() { ... }
  public function executeMySpaceAddMyPlayList() { ... }
  public function executeMySpaceDedicateMusic() { ... }
  public function executeMySpaceProfileMusics() { ... }
}
```

Listagem 7.65: Código fonte: palcoprincipal/modules/palco/actions/components.class.php

```
indexSuccess:
  http_metas:
    content-type: application/xml
  stylesheets:    [PalcoPrincipal]
  layout: gadget_layout

mySpaceSuccess:
  http_metas:
    content-type: application/xml
  stylesheets:    [PalcoPrincipal]
  layout: gadget_layout
```

Listagem 7.66: Código fonte: palcoprincipal/modules/palco/config/view.yml

D - Código fonte da Base de Dados da aplicação OpenSocial

```
--
-- Estrutura da tabela `container`
--

CREATE TABLE IF NOT EXISTS `container` (
  `container_id` varchar(255) NOT NULL default '' COMMENT 'Este é o identificador do
container',
  `container_website` varchar(255) default NULL COMMENT 'O site a que esta associado o
container',
  `container_css` varchar(255) default NULL COMMENT 'CSS especifico para este container',
  PRIMARY KEY  (`container_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Table para lidar com a informação dos
containers';

--
-- Estrutura da tabela `container_user`
--

CREATE TABLE IF NOT EXISTS `container_user` (
  `container_id` varchar(255) NOT NULL default '',
  `user_id` varchar(255) NOT NULL default '',
  `user_name` varchar(255) default NULL,
  `user_url` varchar(255) default NULL,
  `user_thumbnail` varchar(255) default NULL,
  PRIMARY KEY  (`container_id`,`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Corresponde a um utilizador em determinado
container';

--
-- Estrutura da tabela `dedicate_music`
--

CREATE TABLE IF NOT EXISTS `dedicate_music` (
  `container_id` varchar(255) NOT NULL default '',
  `from_id` varchar(255) NOT NULL default '',
  `to_id` varchar(255) NOT NULL default '',
  `created_at` timestamp NOT NULL default '0000-00-00 00:00:00',
  `message` varchar(255) NOT NULL default '',
  `flash_url` varchar(255) NOT NULL default '',
  PRIMARY KEY  (`container_id`,`from_id`,`to_id`,`created_at`,`flash_url`),
  KEY `dedicate_music_ibfk_1` (`flash_url`),
  KEY `dedicate_music_ibfk_3` (`container_id`,`to_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabela de músicas dedicadas';

--
-- Estrutura da tabela `lifecycle`
--

CREATE TABLE IF NOT EXISTS `lifecycle` (
  `container_id` varchar(255) NOT NULL default '',
  `user_id` varchar(255) NOT NULL default '',
  `created_at` timestamp NOT NULL default '0000-00-00 00:00:00',
  `event_type` varchar(255) NOT NULL default '',
  `event_desc` varchar(255) default NULL,
  PRIMARY KEY  (`container_id`,`user_id`,`created_at`,`event_type`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='Tabela que guarda informação sobre actividades
da aplicaç';

--
-- Estrutura da tabela `music`
--

CREATE TABLE IF NOT EXISTS `music` (
  `flash_url` varchar(255) NOT NULL default '',
  `title` varchar(255) NOT NULL default '',
  `bandName` varchar(255) default NULL,
  `link` varchar(255) NOT NULL default '',
  `albumID` varchar(255) default NULL,
  `albumImage` varchar(255) NOT NULL default '',
  `albumName` varchar(255) default NULL,
```

```
  `idMusic` varchar(255) default NULL,
  `mp3` varchar(255) default NULL,
  PRIMARY KEY  (`flash_url`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Corresponde a uma entrada de uma música';

--
-- Estrutura da tabela `owner`
--

CREATE TABLE IF NOT EXISTS `owner` (
  `container_id` varchar(255) NOT NULL default '',
  `owner_id` varchar(255) NOT NULL default '',
  `pp_user_username` varchar(255) NOT NULL default '',
  `num_music` int(11) default '5',
  `num_photos` int(11) default '5',
  `num_friends` int(11) default '5',
  PRIMARY KEY  (`container_id`,`owner_id`),
  KEY `pp_user_username` (`pp_user_username`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabela que liga um owner de uma aplicacao a um
user do palco';

--
-- Estrutura da tabela `pp_user`
--

CREATE TABLE IF NOT EXISTS `pp_user` (
  `pp_user_username` varchar(255) NOT NULL default '',
  `pp_user_uid` varchar(255) NOT NULL default '',
  `pp_user_name` varchar(255) NOT NULL default '',
  `pp_user_style` varchar(255) default NULL,
  `pp_user_type` varchar(255) default NULL,
  `pp_user_url` varchar(255) default NULL,
  `pp_user_photoUrl` varchar(255) default NULL,
  `pp_user_slug` varchar(255) default NULL,
  PRIMARY KEY  (`pp_user_username`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Tabela que corresponde a um utilizador do palco
principal';

--
-- Estrutura da tabela `profile_music`
--

CREATE TABLE IF NOT EXISTS `profile_music` (
  `flash_url` varchar(255) NOT NULL default '',
  `user_id` varchar(255) NOT NULL default '',
  `container_id` varchar(255) NOT NULL default '',
  PRIMARY KEY  (`flash_url`,`user_id`,`container_id`),
  KEY `profile_music_ibfk_2` (`container_id`,`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='Musicas adicionadas a aperfis';

--
-- Restrições na tabela `container_user`
--
ALTER TABLE `container_user`
  ADD CONSTRAINT `container_user_ibfk_1` FOREIGN KEY (`container_id`) REFERENCES `container`
(`container_id`);

--
-- Restrições na tabela`dedicate_music`
--
ALTER TABLE `dedicate_music`
  ADD CONSTRAINT `dedicate_music_ibfk_1` FOREIGN KEY (`flash_url`) REFERENCES `music`
(`flash_url`),
  ADD CONSTRAINT `dedicate_music_ibfk_2` FOREIGN KEY (`container_id`, `from_id`) REFERENCES
`container_user` (`container_id`, `user_id`),
  ADD CONSTRAINT `dedicate_music_ibfk_3` FOREIGN KEY (`container_id`, `to_id`) REFERENCES
`container_user` (`container_id`, `user_id`);

--
-- Restrições na tabela `owner`
--
ALTER TABLE `owner`
  ADD CONSTRAINT `owner_ibfk_1` FOREIGN KEY (`pp_user_username`) REFERENCES `pp_user`
```

```
(`pp_user_username`),
  ADD CONSTRAINT `owner_ibfk_2` FOREIGN KEY (`container_id`, `owner_id`) REFERENCES
`container_user` (`container_id`, `user_id`);

--
-- Restrições na tabela `profile_music`
--
ALTER TABLE `profile_music`
  ADD CONSTRAINT `profile_music_ibfk_1` FOREIGN KEY (`flash_url`) REFERENCES `music`
(`flash_url`),
  ADD CONSTRAINT `profile_music_ibfk_2` FOREIGN KEY (`container_id`, `user_id`) REFERENCES
`container_user` (`container_id`, `user_id`);
```

Listagem 7.67: Código fonte da Base de Dados da aplicação OpenSocial

E - API da aplicação OpenSocial do Palco Principal

E.1 - Login

URL:	/api/login.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container* **(4) viewer_name** : (opcional) nome do *VIEWER* no *container* **(5) viewer_url** : (opcional) *URL* do perfil do *VIEWER* no *container* **(6) viewer_thumbnail** : (opcional) *Thumbnail* do *VIEWER* no *container*

Resposta:	`<response>`
	`  <login>`
	`  <username>xxxxxxxxx</username>`
	`  <name>xxxxxxxxxxxx</name>`
	`  <style>xxxxxxxxxx</style>`
	`  <type>utilizador|banda</type>`
	`  <url>xxxxxxxxxxxxx</url>`
	`  <photoUrl>xxxxxxxxx</photoUrl>`
	`  <slug>xxxxxxxxxxxxx</slug>`
	`  </login>`
	`</response>`
	Listagem 7.68: Resposta login.xml

Erros:	Erro 1 Erro 7 Erro 8

Tabela 7.5: API OpenSocial do Palco Principal: Login

E.2 - Photos

URL:	/api/register.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container* **(4) palco_email** : O *email* da conta do Palco Principal **(5) palco_passwd** : A *password* da conta no Palco Principal
Resposta:	Resposta igual à Listagem 3.11 Como usar a classe GadgetsViews
Erros:	Erro 1 Erro 2 Erro 3 Erro 4 Erro 8

Tabela 7.6: API OpenSocial do Palco Principal: Register

E.3 - GetPlaylist

URL:	/api/getPlaylist.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container*
Resposta:	<response total="x"> <track in_playlist="-1|0|1"> <title>xxxxxxxxxxx</title> <bandName>xxxxxxxxx</bandName> <link>xxxxxxxxxxxx</link> <albumID>xx TODO xx</albumID> <albumImage>xxxxxxx</albumImage> <albumName>xxxxxxxx</albumName> <idMusic>xx TODO xx</idMusic> <mp3>xxxxxxx</mp3> <flash_url>xxxxxxx</flash_url> </track> ... </response> Listagem 7.69: Resposta getPlaylist.xml
Erros:	Erro 1 Erro 2 Erro 7 Erro 8

Tabela 7.7: API OpenSocial do Palco Principal: GetPlaylist

E.4 - GetFriends

URL:	/api/getFriends.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*
Resposta:	<response total="x"> <user> <slug>xxxxxxxxx</slug> <name>xxxxxxxxxxxx</name> <style>xxxxxxxxxx</style> <type>xxxxxxxxxxxx</type> <url>xxxxxxxxxxxxx</url> <photoUrl>xxxxxxxxx</photoUrl> <name_truncated>xxxxxxxxx</name_truncated> </user> ... </response> Listagem 7.70: Resposta getFriends.xml
Erros:	Erro 1 Erro 2 Erro 7 Erro 8

Tabela 7.8: API OpenSocial do Palco Principal: GetFriends

E.5 - GetPhotos

URL:	/api/getPhotos.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*
Resposta:	<response total="x"> <photo> <photoID>xxxxxxxxx</photoID> <smallUrl>xxxxxxxxx</smallUrl> <mediumUrl>xxxxxxxxx</mediumUrl> <url>xxxxxxxxx</url> <permalink>xxxxxxxxx</permalink> </photo> ... </response> Listagem 7.71: Resposta getPhotos.xml
Erros:	Erro 1 Erro 2 Erro 7 Erro 8

Tabela 7.9: API OpenSocial do Palco Principal: GetPhotos

E.6 - GetAccountInfo

URL:	/api/getAccountInfo.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container*
Resposta:	<response> <accountInfo> <username>xxx</username> <numMusics>xx</numMusics> <numFriends>xx</numFriends> <numPhotos>xx</numPhotos> </accountInfo> </response> Listagem 7.72: Resposta getAccountInfo.xml
Erros:	Erro 1 Erro 7 Erro 12

Tabela 7.10: API OpenSocial do Palco Principal: GetAccountInfo

E.7 - UpdateAccountInfo

URL:	/api/updateAccountInfo.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container* **(4) username** : O *email* da conta do Palco Principal **(5) passwd** : A *password* da conta no Palco Principal **(6) remove** : (opcional) Caso o utilizador pretenda remover o *login* **(7) n_musics** : (opcional) Número máximo de músicas a mostrar **(8) n_friends** : (opcional) Número máximo de amigos a mostrar **(9) n_photos** : (opcional) Número máximo de fotos a mostrar **(10) new_palco_email** : (opcional) Novo *email* da conta **(11) new_palco_passwd** : (opcional) *Password* da nova conta **(12) viewer_name** : (opcional) nome do *VIEWER* no *container* **(13) viewer_url** : (opcional) *URL* do perfil do *VIEWER* no *container* **(14) viewer_thumbnail** : (opcional) *Thumbnail* do *VIEWER* no *container*
Resposta:	Resposta igual à Listagem 3.11 Como usar a classe GadgetsViews
Erros:	Erro 1 Erro 2 Erro 3 Erro 7

Tabela 7.11: API OpenSocial do Palco Principal: *UpdateAccountInfo*

E.8 - GetBandFans

URL:	/api/getBandFans.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*

Resposta:	`<response count="x">`
	`  <bandFan>`
	`  <name>xxxxxxxxxxxxx</name>`
	`  <url>xxxxxxxxxxxxxx</url>`
	`  <photoUrl>xxxxxxxxx</photoUrl>`
	`  </bandFan>`
	`  ...`
	`</response>`
	Listagem 7.73: Resposta getBandFans.xml

Erros:	Erro 1 Erro 2 Erro 7 Erro 8

Tabela 7.12: API OpenSocial do Palco Principal: GetBandFans

E.9 - GetBandStats

URL:	/api/getBandStats.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*
Resposta:	<response> <stats> <nome>xxxxxxxxxxxxx</nome> <link>xxxxxxxxxxxxx</link> <estilo>xxxxxxxxxxxxx</estilo> <imagem>xxxxxxxxxxxxx</imagem> <visitas>xxxxxxxxxxxxx</visitas> <visitas24>xxxxxxxxxxxxx</visitas24> <visitas7>xxxxxxxxxxxxx</visitas7> <plays>xxxxxxxxxxxxx</plays> <plays24>xxxxxxxxxxxxx</plays24> <plays7>xxxxxxxxxxxxx</plays7> <reviews>xxxxxxxxxxxxx</reviews> <amigos>xxxxxxxxxxxxx</amigos> <fas>xxxxxxxxxxxxx</fas> <downloads>xxxxxxxxxxxxx</downloads> <downloadsHoje>xxxxxxxxxxxxx</downloadsHoje> <downloads30>xxxxxxxxxxxxx</downloads30> <posicaoGeral>xxxxxxxxxxxxx</posicaoGeral> <posicaoEstilo>xxxxxxxxxxxxx</posicaoEstilo> <graf>xxxxxxxxxxxxx</graf> <data>xxxxxxxxxxxxx</data> </stats> </response> Listagem 7.74: Resposta getBandStats.xml
Erros:	Erro 1 Erro 2 Erro 7 Erro 8

Tabela 7.13: API OpenSocial do Palco Principal: GetBandStats

E.10 - AddProfileMusic

URL:	/api/addProfileMusic.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container**_id : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container* **(4) flash_url** : O endereço *flash* da música **(5) title** : O *título* da música **(6) link** : O *URL* da música **(7) albumImage** : A imagem do álbum
Resposta:	<response> <track in_playlist="1"> <title>xxxx</title/> <albumImage>xxxx</albumImage> <link>xxxx</link> <flash_url>xxxx</flash_url> </track> </response> Listagem 7.75: Resposta addProfileMusic.xml
Erros:	Erro 1 Erro 7 Erro 9 Erro 12

Tabela 7.14: API OpenSocial do Palco Principal: AddProfileMusic

E.11 - DelProfileMusic

URL:	/api/delProfileMusic.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container* **(4) flash_url** : O endereço *flash* da música
Resposta:	<response> <track in_playlist="0"> <title>xxxx</title/> <albumImage>xxxx</albumImage> <link>xxxx</link> <flash_url>xxxx</flash_url> </track> </response> Listagem 7.76: Resposta delProfileMusic.xml
Erros:	Erro 10

Tabela 7.15: API OpenSocial do Palco Principal: DelProfileMusic

E.12 - GetProfileMusics

URL:	/api/getProfileMusics.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container**_id : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*
Resposta:	<response total="x"> <track in_playlist="-1|0|1"> <title>xxxx</title/> <albumImage>xxxx</albumImage> <link>xxxx</link> <flash_url>xxxx</flash_url> </track> ... </response> Listagem 7.77: Resposta getProfileMusic.xml
Erros:	Erro 10

Tabela 7.16: API OpenSocial do Palco Principal: GetProfileMusic

E.13 - AddDedicateMusic

URL:	/api/addDedicateMusic.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container* **(4) flash_url** : O endereço *flash* da música **(5) title** : O título da música **(6) link** : O *URL* da música **(7) albumImage** : A imagem do álbum
Resposta:	Resposta igual à Listagem 3.18 Exemplo de configuração do "app.yml"
Erros:	Erro 1 Erro 7 Erro 9 Erro 12

Tabela 7.17: API OpenSocial do Palco Principal: AddDedicateMusic

E.14 - DelDedicatedMusic

URL:	/api/delDedicatedMusic.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : *ID* do *VIEWER* no *container* **(4) flash_url** : O endereço *flash* da música **(5) from_id** : Quem enviou a dedicatória **(6) to_id** : O destinatário da dedicatória **(7) created_at** : A data de criação da dedicatória
Resposta:	<response> <dedicatedMusic> <containerUser type="from"> <id>xxxxxxxxxxxxx</id> <name>xxxxxxxxxxxxx</name> <url>xxxxxxxxxxxxxx</url> <thumbnail>xxxxxxxxx</thumbnail> </containerUser> <containerUser type="to"> <id>xxxxxxxxxxxxx</id> <name>xxxxxxxxxxxxx</name> <url>xxxxxxxxxxxxxx</url> <thumbnail>xxxxxxxxx</thumbnail> </containerUser> <track> <title>xxxx</title/> <albumImage>xxxx</albumImage> <link>xxxx</link> <flash_url>xxxx</flash_url> </track> <message> <date>xxxx</date/> <content>xxxx</content/> </message> </dedicatedMusic> </response> Listagem 7.78: Resposta delDedicatedMusic.xml
Erros:	Erro 1 Erro 11

Tabela 7.18: API OpenSocial do Palco Principal: DelDedicatedMusic

E.15 - GetDedicatedMusics

URL:	/api/getDedicatedMusics.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container* **(4) from_id** : (opcional) Quem enviou a dedicatória **(5) to_id** : (opcional) O destinatário da dedicatória
Resposta:	<response total="xxxx"> <dedicatedMusic> <containerUser type="from"> <id>xxxxxxxxxxxx</id> <name>xxxxxxxxxxxx</name> <url>xxxxxxxxxxxx</url> <thumbnail>xxxxxxxxx</thumbnail> </containerUser> <containerUser type="to"> <id>xxxxxxxxxxxx</id> <name>xxxxxxxxxxxx</name> <url>xxxxxxxxxxxx</url> <thumbnail>xxxxxxxxx</thumbnail> </containerUser> <track in_playlist="-1|0|1"> <title>xxxx</title/> <albumImage>xxxx</albumImage> <link>xxxx</link> <flash_url>xxxx</flash_url> </track> <message> <date>xxxx</date/> <content>xxxx</content/> </message> </dedicatedMusic> ... </response> Listagem 7.79: Resposta getDedicatedMusics.xml
Erros:	Erro 1

Tabela 7.19: API OpenSocial do Palco Principal: GetDedicatedMusics

E.16 - GetPlaylistByStyle

URL:	/api/getPlaylistByStyle.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : (opcional) *ID* do *container* **(2) owner_id** : (opcional) *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container* **(4) style** : O *ID* do estilo músical

Resposta:	`<response total="x">`
	`  <style>xxxx</style>`
	`  <track in_playlist="-1|0|1">`
	`  <title>xxxx</title/>`
	`  <albumImage>xxxx</albumImage>`
	`  <link>xxxx</link>`
	`  <flash_url>xxxx</flash_url>`
	`  </track>`
	`  ...`
	`</response>`
	Listagem 7.80: Resposta getPlaylistByStyle.xml

Erros:	Erro 1 Erro 2

Tabela 7.20: API OpenSocial do Palco Principal: GetPlaylistByStyle

E.17 - GetStyles

URL:	/api/getStyles.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : (opcional) *ID* do *container* **(2) owner_id** : (opcional) *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*

Resposta:	`<response>`
	`  <style>`
	`  <key>xxxx</key>`
	`  <value>xxxx</value>`
	`  </style>`
	`  ...`
	`</response>`
	Listagem 7.81: Resposta getStyles.xml

Erros:	-

Tabela 7.21: API OpenSocial do Palco Principal: GetStyles

E.18 - GetTopDedicatedMusicsByContainer

URL:	/api/getTopDedicatedMusicsByContainer.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : (opcional) *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*
Resposta:	Resposta igual à Listagem 7.79 Resposta getDedicatedMusics.xml
Erros:	Erro 1

Tabela 7.22: API OpenSocial do Palco Principal: GetTopDedicatedMusicsByContainer

E.19 - GetTopMusicsByContainer

URL:	/api/getTopMusicsByContainer.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container_id** : *ID* do *container* **(2) owner_id** : (opcional) *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container*
Resposta:	Resposta igual à Listagem 7.77 Resposta getProfileMusic.xml
Erros:	Erro 1

Tabela 7.23: API OpenSocial do Palco Principal: GetTopMusicsByContainer

E.20 - GetContainerUsersByFlashURL

URL:	/api/getContainerUsersByFlashURL.xml
Tipo:	GET ou POST
Parâmetros:	**(1) container**_id : (opcional) *ID* do *container* **(2) owner_id** : (opcional) *ID* do *OWNER* no *container* **(3) viewer_id** : (opcional) *ID* do *VIEWER* no *container* **(4) flash_url** : O endereço *flash* da música

Resposta:	`<response total="xxxx">`
	`  <containerUser container_id="xxxx">`
	`  <id>xxxx</id>`
	`  <name>xxxx</name>`
	`  <url>xxxx</url>`
	`  <thumbnail>xxxx</thumbnail>`
	`  </containerUser>`
	`  ...`
	`</response>`
	Listagem 7.82: Resposta getContainerUsersByFlashURL.xml

Erros:	Erro 1

Tabela 7.24: API OpenSocial do Palco Principal: GetContainerUsersByFlashURL