Mestrado em Engenharia Informática

Estágio

Relatório Final

SoFly:
from the foundations to the roof

Pedro Miguel Nunes Martins Pinto
pmpinto@student.dei.uc.pt

Orientador da BroadScope:
Virgílio Esteves

Orientador do DEI:
Jorge Henriques

Data: 2 de Setembro de 2015

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Mestrado em Engenharia Informática

Estágio

Relatório Final

SoFly:
from the foundations to the roof

Pedro Miguel Nunes Martins Pinto
pmpinto@student.dei.uc.pt

Júri Vogal:
Paulo Carvalho

Júri Arguente:
Filipe Araújo

Orientador do DEI:
Jorge Henriques

Orientador da BroadScope:
Virgílio Esteves

Data: 2 de Setembro de 2015

Resumo

O impacto das redes sociais na sociedade em geral tem vindo a aumentar de dia para dia, em parte devido à liberdade de expressão que as mesmas permitem. Assim sendo, as empresas de uma forma geral, mas os grupos de media e os canais de televisão em particular, cada vez mais olham para estas redes sociais a fim de perceberem melhor o seu público-alvo, bem como os seus gostos e tendências. Estes factos evidenciam que as redes de media e os canais de televisão necessitam de meios para analisarem e perceberem esses mesmos acontecimentos de uma forma eficiente e rápida, para poderem responder e actuar em tempo útil. Este estágio de Mestrado em Engenharia Informática visa colmatar essa evidente carência através do desenvolvimento de uma plataforma de análise de redes sociais, plataforma essa que visa permitir reunir dados das diversas redes sociais e analisá-los posteriormente, bem como providenciar formas de visualização dessa mesma análise. A plataforma visa permitir aos seus clientes compreenderem melhor o seu público através das análises que a dita plataforma permite e providenciar-lhes a capacidade de trazerem a experiência social aos seus conteúdos de media.

Abstract

The impact of social networks in society has been increasing everyday, partly due to the freedom of speech allowed by them. Thus enterprise businesses in general, but specially media groups and TV stations in particular, increasingly look for the social networks to better understand their audiences and drive engagement. These demonstrates that the media networks and television channels need means to analyze these events quickly and efficiently, so that they can respond in a timely manner. This Master's internship in Computer Science aims to develop a social network analysis platform, which will gather data from several social networks, analyze it and allow the visualization of that analysis. The platform aims to enable its clients to better understand their audience through the analytics it provides and to give them the ability to bring social experience into their media content.

Agradecimentos

Queria agradecer a toda a equipa da BroadScope pela forma como me apoiaram e ajudaram desde o primeiro dia.

A todos os meus amigos pelo apoio, ajuda e paciência demonstrados ao longo deste percurso da minha vida.

À minha família pelo apoio e motivação, sem os quais nada disto seria possível.

À minha namorada Soraia pela paciência, motivação e incentivos dados quando necessário.

A ambos os meus orientadores pelo tempo despendido e paciência demonstrada durante esta importante fase da minha vida.

Um grande obrigado a todos por tudo.

Índice

Termos e Acrónimos

ADO	*ActiveX® Data Objects*
API	*Application Programming Interface* (Interface de Programação de Aplicações)
ASP	*Active Server Pages* (Páginas Activas de Servidor)
BLOB	*Binary Large Object*
CDN	*Content Delivery Network* (Rede de Fornecimento de Conteúdo)
COM	*Component Object Model*
CRUD	*Create*, *Read*, *Update and Delete* (Criar, Ler, Actualizar e Remover)
CTO	*Chief Technology Officer* (Director-Chefe Tecnológico)
DOM	*Document Object Model* (Modelo de Objecto de Documentos)
Framework	Plataforma de desenvolvimento
Forever Frame	Técnica na qual são utilizados pedaços de dados para manter uma conexão de longa duração viva numa *iframe* escondida.
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
HTTPS	*Hypertext Transfer Protocol Secure* (Protocolo Seguro de Transferência de Hipertexto)
IaaS	*Infrastructure-as-a-Service* (Infra-estrutura como um Serviço)
IDE	*Integrated Development Environment* (Ambiente Integrado de Desenvolvimento)
JSON	*JavaScript Object Notation*
LINQ	*Language-Integrated Query*
Load Balancing	Distribuir o trabalho por múltiplos recursos de computação
Long Polling	Variante da técnica *polling* (ver *Polling*)
MVC	*Model-View-Controller* (Modelo-Vista-Controlador)
OAuth	*Open Standard Authorization* (Standard Aberto de Autenticação)
ODATA	*Open Data Protocol* (Protocolo Aberto de Dados)
On-premises	No local ou nas instalações.
ORM	*Object-Relational Mapping* (Mapeamento Objecto-Relacional)
OWIN	*Open Web Interface* (Interface Aberto de Web)
PaaS	*Platform-as-a-Service* (Plataforma como um Serviço)
Polling	Processo de espera para verificação de estado ou prontidão

REST	*Representational State Transfer*
SaaS	*Software-as-a-Service* (Software como um Serviço)
Second Screening	Proporciona uma experiencia de visualização melhorada e aumentada através do uso de um segundo dispositivo de visualização
SQL	*Structured Query Language* (Linguagem de Consulta Estruturada)
Server-sent events	Tecnologia na qual um *browser* recebe actualizações automáticas de um servidor através de uma conexão HTTP (ver HTTP)
SSL	*Secure Sockets Layer* (Camada Segura de Sockets)
TLS	*Transport Layer Security* (Camada de Transporte de Segurança)
XML	*Extensible Markup Language*

1. Introdução

1.1. Motivação

As redes sociais tem um impacto importantíssimo nos dias de hoje na sociedade em geral, pois permitem a pessoas ou comunidades, independentemente do local onde se encontrem, expressarem as suas opiniões. Por permitirem essa mesma liberdade, entre outras vantagens, existem cada vez mais redes sociais e de maior dimensão, tornando o seu impacto na sociedade cada vez mais significativo. Assim sendo, as empresas, e em especial as redes de media e canais de televisão, cada vez mais se focam nas redes sociais para melhor perceberem as suas audiências, bem como para compreender as suas tendências, gostos, vontades e necessidades.

Tendo por base os pontos anteriormente mencionados, torna-se evidente que as redes de media e os canais de televisão necessitam de estar dotados de meios eficientes e rápidos para responderem convenientemente e em tempo útil às necessidades, gostos e vontades do seu público-alvo ou às suas mudanças. Como tal, a plataforma SoFly visa colmatar essa dita carência.

1.2. Enquadramento

O presente projecto encontra-se inserido no âmbito do estágio académico anual para a obtenção do grau de Mestre em Engenharia Informática, a decorrer nas instalações da empresa *BroadScope Lda*, localizada em Coimbra no Instituto Pedro Nunes (IPN).

Este mesmo projecto visa colmatar a necessidade evidente, das redes de media e canais de televisão, em melhor entender/compreender as suas audiências, bem como perceber quem são. Com vista aos suprimentos destas necessidades, propõe-se especificar e desenvolver de raiz uma plataforma que vise permitir reunir dados de várias redes sociais, analisá-los posteriormente e disponibilizar visualizações das análises efectuadas. A partir da visualização das análises conduzidas, pretende-se extrair informação útil e valiosa de negócio que permita validar as opções de negócio tomadas pelos clientes, bem como auxiliar as decisões futuras dos mesmos.

1.3. Objectivos do Estágio

É objectivo do estágio analisar, propor e desenvolver módulos para uma plataforma, o *SoFly Analytics*, plataforma essa que irá ser construída integralmente de raiz durante o estágio. A plataforma e os seus módulos não serão, no entanto, desenvolvidos inteiramente pelo estagiário, cabendo ao mesmo desenvolver técnicas e mecanismos que cumpram os requisitos e assim permitir à equipa, estagiário incluído, desenvolver posteriormente os módulos da plataforma com base nesse mesmo trabalho efectuado anteriormente.

Numa primeira fase do projecto, os objectivos do estagiário visam: desenvolver técnicas tendo por base tecnologias que permitam desenvolver os módulos, desenhar a arquitectura interna desses mesmos módulos, identificar os padrões de *design* de *software* a utilizar e as tecnologias que resolvam os problemas encontrados, mas que cumpram inteiramente os requisitos dos ditos módulos.

Na segunda fase do projecto, é objectivo do estagiário desenvolver a plataforma juntamente com a restante equipa de desenvolvimento com base no trabalho efectuado na primeira fase, ou seja, com base nas técnicas desenvolvidas, os padrões de *design* de *software* identificados, respeitar a arquitectura delineada, utilizar as tecnologias referênciadas, bem como respeitar os requisitos de cada módulo.

É contudo importante realçar que o objectivo deste estágio é a construção da plataforma (fundações da plataforma *SoFly*) e não do produto em si, por isso mesmo, o projecto e o estágio são orientados à construção da dita plataforma e não orientado ao produto nem ao negócio. Considera-se ainda importante referir que os algoritmos desenvolvidos para a análise dos dados não fazem parte do âmbito do estágio em si e foram desenvolvidos em paralelo com mesmo, não existindo em nenhuma altura qualquer envolvimento do estagiário no seu desenvolvimento ou implementação.

A plataforma *SoFly Analytics*, ou *SoFly* como diminutivo, visa reunir informações das redes sociais e analisá-las em "tempo real", permitindo assim obter feedback através da análise das mesmas em tempo útil. A plataforma permitirá a canais de televisão e a grupos de media compreenderem melhor as suas audiências através da análise das informações nas redes sociais, de forma a conduzirem o conteúdo das mesmas, ou seja, permitir-lhes adaptar e responder de uma forma ágil e rápida, bem como orientar as temáticas disponíveis e discutidas nas redes sociais, possibilitando-lhes também a adição de novas temáticas. No entanto, sendo que esses conteúdos se encontram espalhados por várias redes sociais, não é

práctico, nem fiável e muito menos eficiente, controlá-los, organizá-los e analisá-los manualmente. O *SoFly* procura assim possibilitar aos seus clientes uma melhor compreensão do seu público-alvo, sendo que também permite, através desta análise de uma forma totalmente automática, a tomada de decisões de negócio melhor fundamentadas.

É importante mencionar que apenas são reunidos os dados marcados como públicos das redes sociais, pois os marcados como privados pelos utilizadores não são fornecidos através das API das respectivas redes sociais sem o consentimento desses mesmos utilizadores. Para além desta restrição, as redes sociais obrigam ao cumprimento integral dos seus termos de serviço, os quais variam de acordo com a rede social em causa, facto este que o sistema cumpre na íntegra.

O *SoFly* foi desenvolvido totalmente utilizando a *Framework.NET* na linguagem *C#* (*C Sharp*), estando por isso condicionado o desenvolvimento dos respectivos módulos a essa mesma linguagem e *framework*. A plataforma possui vários módulos [1] os quais são:

1. Camada de Dados
2. API de negócio
3. *Web Front-End*
4. Notificações
5. Mensagens
6. Trabalhadores

1. Camada de dados: É o único módulo a ser totalmente desenvolvido pelo estagiário, sendo que nos restantes o estagiário irá dividir o desenvolvimento dos mesmos com a restante equipa. A camada de dados é o módulo no qual é efectuado todo o armazenamento de dados da plataforma. No entanto, deseja-se que os dados que os clientes recolham das redes sociais se encontrem segregados e como tal, não deverão ser guardados juntamente com os restantes dados da plataforma, bem como deve ser possível a um cliente utilizar qualquer plataforma na nuvem ou *on-premesis* para guardar esses mesmos dados. Este módulo será exclusivamente desenvolvido pelo estagiário, como já foi referido anteriormente.

2. API de negócio: A plataforma deverá permitir aos clientes devidamente autenticados acederem aos dados coleccionados das redes sociais através de uma API, a qual corresponde ao módulo denominado API de negócio, que implementará o padrão arquitectural REST respeitando e seguindo todas as normas desse padrão. Pretende-se que a dita API permita aceder aos dados no módulo da camada de dados em regime de exclusividade, ou seja, mais

nenhum módulo deverá aceder directamente ao módulo da camada de dados. Contudo, e ao contrário do módulo anterior, toda a equipa de desenvolvimento irá trabalhar no desenvolvimento deste módulo, estagiário incluído.

3. Web Front-End: Deverá existir um *front-end* na plataforma, o qual será construído com base em tecnologia *ASP.NET* e com recurso ao padrão de design de software Modelo-Vista-Controlador (MVC) [2][3], ao qual corresponde o módulo *Web Front-End*. Pretende-se que o *front-end* também faça uso da API de negócio para aceder aos dados armazenados sem qualquer acesso directo à camada de dados, assegurando a premissa já referida anteriormente. Para além do *front-end,* também se pretende que os clientes possuam a opção de usar *plugins*, desenvolvidos pela empresa ou pelos clientes, para se ligarem à API de negócio e acederem aos dados. Este módulo, à semelhança do anterior, não será desenvolvido apenas pelo estagiário, mas sim por toda a equipa de desenvolvimento.

4. Notificações: Pretende-se que a plataforma possua um sistema de notificações devido ao facto de um dos requisitos ser a possibilidade de os clientes poderem aceder e/ou visualizar os dados reunidos em tempo real. O dito sistema deverá permitir enviar os dados, em tempo real, para todos aqueles que de eles necessitarem. Neste módulo, e mais uma vez à semelhança dos dois imediatamente anteriores, o desenvolvimento do módulo será efectuado por toda a equipa de desenvolvimento.

5. Mensagens: Deverá existir um sistema de mensagens na plataforma, como tal este módulo será responsável pela distribuição de todas as mensagens entre os módulos da plataforma que necessitem desse dito serviço. À semelhança dos módulos anteriores, o desenvolvimento será efectuado por toda a equipa de desenvolvimento.

6. Trabalhadores: Por último, mas não menos importante, o módulo Trabalhadores é responsável por todo o tipo de trabalho necessário, desde a captura dos dados nas redes sociais à análise posterior desses mesmos dados. Este é o módulo no qual o estagiário tem menos responsabilidades e menor participação, pois o desenvolvimento e envolvência no mesmo será mínimo, com ênfase apenas nos mecanismos de integração e na integração deste módulo com os restantes.

1.4. Estrutura do Documento

O presente Documento é estruturado contento os seguintes sete capítulos:

1. Introdução
2. Gestão do Projecto
3. O Projecto
4. Desenvolvimento
5. Conclusões e Comentários finais
6. Referências
7. Anexos

O primeiro capítulo, no qual nos encontramos, contém a introdução ao projecto bem como as informações gerais do estágio.

No segundo capítulo é descrita a equipa no qual o presente projecto é enquadrado, expondo também a coordenação do mesmo, bem como reveladas as metodologias e o planeamento adoptado.

Em seguida, o terceiro capítulo aborda os requisitos, o estado da arte, os desafios e as soluções encontradas para os mesmos.

O desenvolvimento do projecto é descrito no quarto capítulo, mais concretamente os estudos efectuados, as ferramentas e tecnologias utilizadas, exposta a arquitectura do sistema e os testes executados na sua validação.

No quinto capítulo são revistas as conclusões obtidas no presente estágio curricular.

O sexto capítulo contêm todas as referências consultadas.

Finalmente o sétimo capítulo possui todos os anexos relevantes à contextualização do estágio.

2. Gestão do Projecto

2.1. Equipa

A equipa do presente projecto é composta por três membros, sendo um deles o estagiário e pelo CTO da empresa onde decorre o estágio.

2.2. Coordenação e Reuniões

Na gestão do projecto foram coordenadas reuniões quinzenais entre o estagiário e os coordenadores, tanto da empresa como do DEI (Departamento de Engenharia Informática).

Para além das ditas reuniões também são entregues relatórios com uma frequência quinzenal para se manter um registo preciso tanto do desenvolvimento do projecto como dos problemas e soluções encontradas. Estes relatórios são do formato 15/5 (quinze minutos de escrita e cinco de leitura) para permitirem uma rápida leitura e escrita dos mesmos, bem como uma efectiva e útil transmissão do trabalho realizado e a realizar.

Apesar das reuniões anteriormente referidas também são efectuadas reuniões diárias com toda a equipa de desenvolvimento, nas quais o orientador da empresa está presente. É sempre discutido o progresso do projecto, bem como as dificuldades encontradas e as dependências que possam existir. As reuniões são realizadas de acordo com a metodologia ágil adoptada pela empresa, sendo esta descrita mais em pormenor no subcapítulo seguinte.

2.3. Metodologia

No presente projecto foi seguida uma metodologia ágil [4][5], mais concretamente o *Scrum* [5], mas não de uma forma rígida.

Na metodologia *Scrum*, de acordo com o seu manifesto [4], são seguidos os seguintes princípios [4]:

- Satisfação do cliente através da entrega rápida de *software* útil
- Aceitar alterações nos requisitos, mesmo em fases tardias de desenvolvimento
- O *software* funcional é entregue frequentemente
- Cooperação diária entre as equipas de desenvolvimento e as de negócio

- Os projectos são contruídos por indivíduos motivados
- A comunicação deve ser feita de forma directa e pessoalmente
- A entrega do *software* funcional é a principal forma de medição do progresso
- O desenvolvimento deve ser sustentado num ritmo constante
- Atenção constante à excelência técnica e bom *design*
- Simplicidade
- Equipas auto-organizadas
- Adaptações regulares a alterações no projecto

Foram e são realizadas reuniões diárias, como referido anteriormente, com todos os membros da equipa presentes sem excepção. As reuniões têm sempre lugar antes do inicio do trabalho diário e seguem o princípio das reuniões diárias [5] de *Scrum*, sendo por isso efectuadas de pé e nas quais cada membro da equipa deve responder às seguintes três perguntas [5]:

1. O que foi feito ontem?
2. O que vai fazer hoje?
3. Que problemas/impedimentos/desafios encontrou que não deixaram o trabalho ser feito?

Seguindo esta metodologia no final de cada dia é guardado o progresso diário feito pela equipa de desenvolvimento na *Sprint* [5] para análise futura.

No decorrer do presente projecto foram sendo fornecidas constantes actualizações ao cliente de forma regular e faseada. Neste caso o cliente é um portal desportivo na Bulgária, o qual gere páginas de clubes, bem como de desportistas, em várias redes sociais.

Os requisitos foram também alterados durante o processo de desenvolvimento conforme o feedback dado pelo cliente. Foram para além de adicionadas novas funcionalidades ao produto, alteradas algumas das funcionalidades existentes. Algumas dessas alterações, a título de exemplo, foram a simplificação de tarefas, bem como a adição de atalhos para facilitar o acesso a algumas funcionalidades.

Os objectivos podiam ser ajustados, apesar de bem definidos desde o início, mas não ser alterados inteiramente.

2.4. Planeamento

Neste subcapítulo irá ser analisado e discutido o planeamento do presente estágio. Após análise com o responsável do projecto e orientador do estágio concluiu-se que o melhor planeamento para um bom andamento do mesmo seria a estratégia evidenciada seguidamente em duas fases distintas.

Na primeira fase do estágio, que correspondeu ao primeiro semestre, existiriam cinco tarefas principais, apresentadas em seguida:

- Análise de requisitos
- Desenho da arquitectura e validação
- Provas de conceito
- Escrita do relatório
- Defesa

No entanto a tarefa Provas de conceito dividir-se-ia em subtarefas devido à complexidade da mesma, as quais referenciamos seguidamente:

- Afinação da arquitectura (todos os módulos)
- Segregação da informação das bases de dados (módulo Camada de dados)
- ORM - Entity Framework (módulos Camada de dados e API de negócio)
- Autenticação – Oauth (módulo API de negócio)
- Concorrência optimística (módulos camada de dados e API de negócio)
- Distribuição de mensagens (módulos Notificações, Mensagens, Trabalhadores e API de negócio)
- Sistema de plugins gráficos (módulo Web Front-End)

Na segunda fase do estágio, a realizar-se durante o segundo semestre, apenas se encontram três tarefas principais, que são:

- Desenvolvimento e testes
- Escrita do relatório
- Defesa

A tarefa Desenvolvimento e testes, à semelhança da tarefa Provas de conceito, foi dividida em múltiplas subtarefas devido à sua complexidade, neste caso foram criadas

subtarefas para cada módulo da plataforma. No entanto algumas destas subtarefas, devido novamente à complexidade das mesmas, tiveram novamente de ser subdivididas. É importante referir que todas as tarefas já tem incluídos os testes efectuados. As subtarefas criadas foram:

- Módulo Camada de Dados
- Módulo API de negócio
    - Mapeamento das entidades
    - Implementação de acessos para manipulação e localização dos objectos de negócio
    - Criação dos endpoints da Web API com autenticação *OAuth*
- Módulo Mensagens
- Módulo Notificações
- Módulo Trabalhadores
    - Criação do *worker SocialGrabber* – trabalhador que visa reunir os dados das redes sociais
    - Criação do *worker Analyser* – trabalhador que visa analisar os dados recolhidos das redes sociais
- Módulo *Web Front-End*
    - Criação do *website* com autenticação
    - Adição de gestão de streams
    - Adição de gestão temas para plugins
    - Adição de gestão de plugins
    - Adição de gestão de subscrições
    - Adição de gestão do utilizador
    - Adição de gestão da audiência
    - Desenvolvimento de plugins

Seguidamente irão ser mostrados os diagramas de *Gantt* que representam o planeamento acima descrito, para uma melhor visualização do mesmo. É importante mencionar que o mesmo inclui os *sprints* efectuados durante o desenvolvimento da plataforma, razão pela qual não foram incluídos no relatório.

Nome da Tarefa	Duração	Início	Conclusão
Estágio	**216 days**	**Mon 15/09/14**	**Mon 13/07/15**
Primeiro Semestre	**105 days**	**Mon 15/09/14**	**Fri 06/02/15**
Análise requisitos	20 days	Mon 15/09/14	Fri 10/10/14
Desenho arquitetura e validação	20 days	Mon 29/09/14	Fri 24/10/14
Provas de conceito	**55 days**	**Mon 27/10/14**	**Fri 09/01/15**
Afinação arquitetura	50 days	Mon 27/10/14	Fri 02/01/15
Segregação informação BD's	10 days	Mon 27/10/14	Fri 07/11/14
Entity Framework	10 days	Mon 27/10/14	Fri 07/11/14
OAuth	15 days	Mon 10/11/14	Fri 28/11/14
Concorrência optimística	10 days	Mon 10/11/14	Fri 21/11/14
Distribuição de mensagens	15 days	Mon 01/12/14	Fri 19/12/14
Sistema de plugins gráficos	15 days	Mon 22/12/14	Fri 09/01/15
Escrita do relatório	82 days	Mon 06/10/14	Tue 27/01/15
Defesa	5 days	Mon 02/02/15	Fri 06/02/15
Segundo Semestre	**111 days**	**Mon 09/02/15**	**Mon 13/07/15**

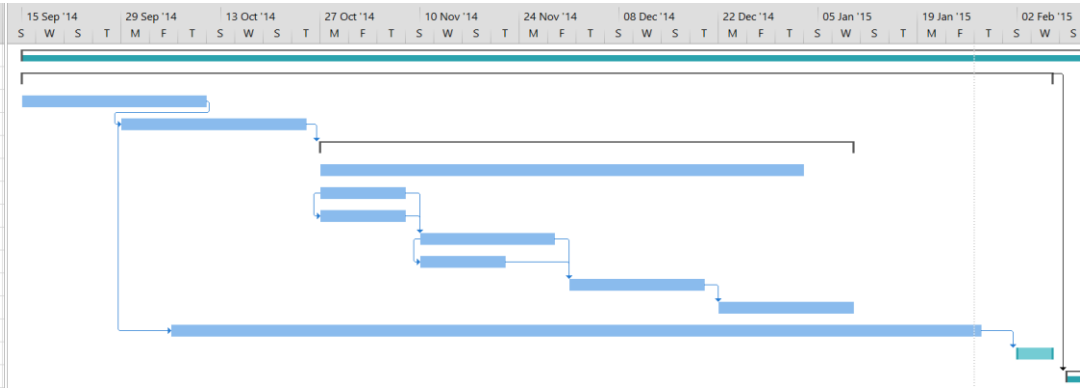


Figura 1 – Diagrama de Gantt do primeiro semestre

Nome da Tarefa	Duração	Início	Conclusão
Estágio	**216 days**	**Mon 15/09/14**	**Mon 13/07/15**
Primeiro Semestre	**105 days**	**Mon 15/09/14**	**Fri 06/02/15**
Segundo Semestre	**111 days**	**Mon 09/02/15**	**Mon 13/07/15**
Desenvolvimento e testes	**105 days**	**Mon 09/02/15**	**Fri 03/07/15**
Módulo Camada de Dados	15 days	Mon 09/02/15	Fri 27/02/15
Módulo API de negócio	**40 days**	**Mon 02/03/15**	**Fri 24/04/15**
Mapeamento das entidades	10 days	Mon 02/03/15	Fri 13/03/15
Implementação de acessos para manipulação e localização dos objectos de negócio	20 days	Mon 02/03/15	Fri 27/03/15
Criação dos endpoints da Web API com autenticação Oauth	20 days	Mon 30/03/15	Fri 24/04/15
Módulo Mensagens	15 days	Mon 30/03/15	Fri 17/04/15
Módulo Notificações	15 days	Mon 27/04/15	Fri 15/05/15
Módulo Trabalhadores	**35 days**	**Mon 20/04/15**	**Fri 05/06/15**
Criação do SocialGrabber	20 days	Mon 20/04/15	Fri 15/05/15
Criação do Analyser	15 days	Mon 18/05/15	Fri 05/06/15
Módulo Web Front-End	**50 days**	**Mon 27/04/15**	**Fri 03/07/15**
Criação do website com autenticação	10 days	Mon 27/04/15	Fri 08/05/15
Adição de gestão de streams	20 days	Mon 11/05/15	Fri 05/06/15
Adição de gestão de temas para plugins	10 days	Mon 11/05/15	Fri 22/05/15
Adição de gestão de plugins	15 days	Mon 08/06/15	Fri 26/06/15
Adição de gestão de subscrições	10 days	Mon 11/05/15	Fri 22/05/15
Adição de gestão de utilizador	10 days	Mon 11/05/15	Fri 22/05/15
Adição de gestão da audiência	20 days	Mon 08/06/15	Fri 03/07/15
Desenvolvimento de plugins	15 days	Mon 18/05/15	Fri 05/06/15
Escrita do relatório	76 days	Mon 23/03/15	Mon 06/07/15
Defesa	1 day	Mon 13/07/15	Mon 13/07/15

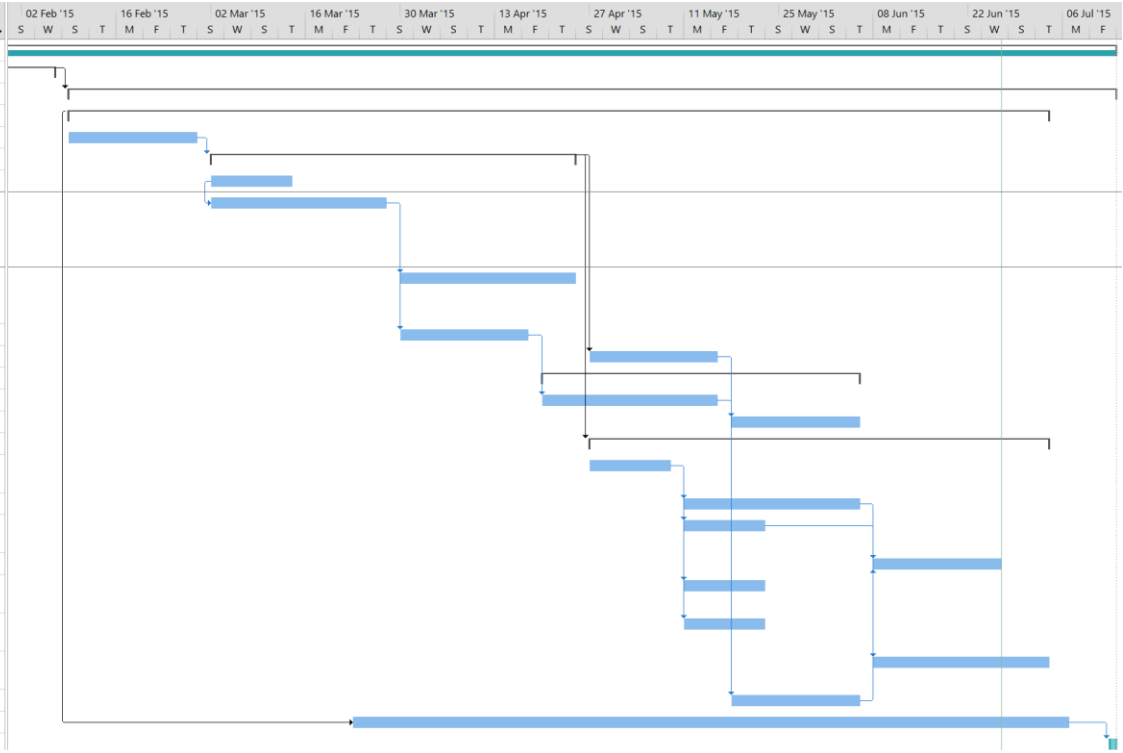


Figura 2 – Diagrama de Gantt do segundo semestre

3. O Projecto

3.1. Cenários de Uso

Nesta secção serão analisados os cenários de uso que a plataforma visa resolver. Estes cenários surgem como substituto da documentação de requisitos devido à sua não utilização por parte da empresa nos seus projectos. A empresa prefere empregar a discussão de cenários de uso com os clientes para determinar o que deve ser implementado e como. Estes denominados cenários de uso são semelhantes aos casos de uso, partilhando algumas características, mas não são casos de uso nem seguem a rigidez dos mesmos. Em seguida, são apresentadas as relações entre os principais cenários de uso e os módulos da plataforma.

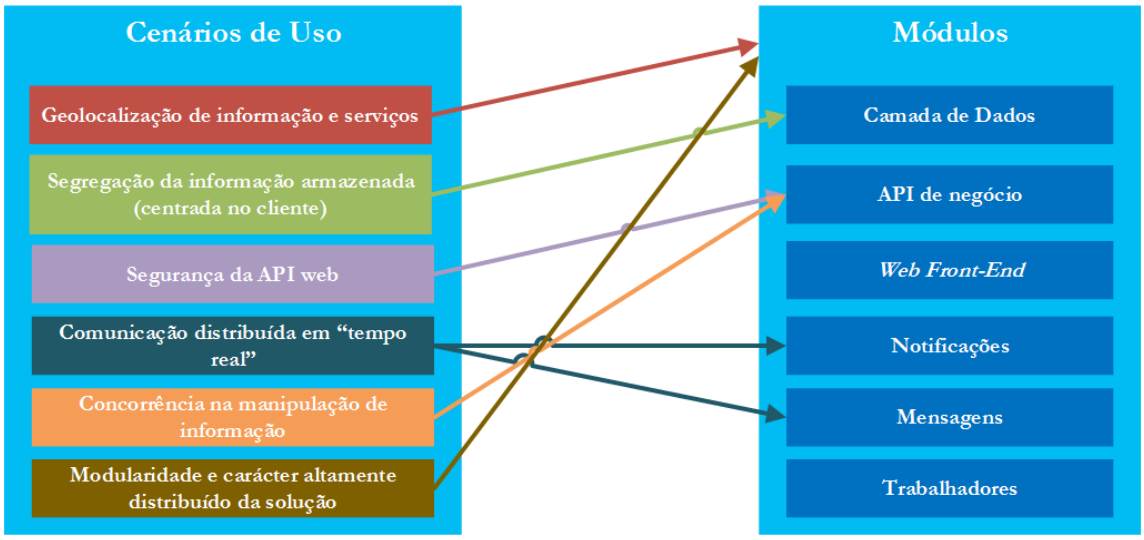


Figura 3 – Relações entre Cenários de uso e Módulos da plataforma

Os dois cenários de uso que não se ligam a módulos individualmente (Geolocalização de informação e serviços e Modularidade e carácter altamente distribuído da solução), mas sim à tabela Módulos como um todo, tem um significado especial, pois ambos os cenários se aplicam a todos os módulos. Os cenários anteriormente mencionados permitem revelar com que princípios se pretende construir a plataforma, sendo em seguida feita uma breve descrição de cada um.

Na geolocalização da informação e serviços pretende-se que tanto a informação como os serviços da própria plataforma sejam distribuídos por vários *data centers* em regiões distintas. Por esta razão a plataforma será planeada e preparada para lidar com tal cenário.

Pretende-se com a segregação da informação armazenada, centrada no cliente, que a plataforma permita separar os dados adquiridos nas redes sociais por cliente. Desta forma permite também aos clientes definir onde guardar a informação, ou seja, se pretendem guardar esses dados em outra plataforma de computação na nuvem que não a escolhida para alojar a plataforma ou mesmo nas suas próprias instalações (*on-premises*).

Na segurança da API web pretende-se utilizar tecnologias de ponta ao nível da segurança, sendo no entanto crucial usar uma norma já existente com garantias dadas no mercado.

No cenário da comunicação distribuída em tempo real pretende-se que a plataforma possua mecanismos que permitam distribuir mensagens entre os vários módulos existentes na plataforma para além da interacção destes com componentes externos, bem como garantir que essas comunicações sejam em "tempo real".

Pretende-se que relativamente à concorrência na manipulação de informação a plataforma possua meios para permitir e garantir a integridade dos dados em acessos simultâneos de cariz possivelmente conflituoso tanto à informação guardada como à qual a plataforma possua acesso.

Na modularidade e carácter altamente distribuído da solução pretende-se garantir que a plataforma seja desenhada e construída de uma forma modular, ou seja, que a plataforma seja dividida em partes distintas, bem como seja possível distribuir o próprio sistema.

3.2. Estado da Arte

3.2.1. Plataformas de Análise de Redes Sociais

É sempre imperativo fazer uma análise das soluções existentes no mercado, tanto a nível nacional como internacional. Sendo assim neste capítulo irá ser feita uma análise das plataformas analíticas semelhantes à que se pretende desenvolver e por conseguinte, concorrentes à mesma. A análise seguidamente descrita já tinha sido efectuada, ao nível do negócio, por membros da empresa e originou precisamente este estágio.

Ao nível nacional não existem propostas semelhantes na análise de redes sociais em tempo real ou não, orientadas à televisão, mais especificamente a programas televisivos como: séries, *reality shows*, concursos, jornais televisivos ou outros. Pretende-se assim que o produto seja pioneiro nessa área em Portugal e desta forma conseguir obter uma posição de referência num mercado emergente.

Em relação ao panorama europeu a realidade é diferente, pois já existem plataformas de análise com características semelhantes, sendo estas no entanto apenas orientadas a marcas e não à televisão. Por marcas entenda-se marcas comercias de produtos, bem como a marca de uma dada empresa.

No cenário americano é diferente, pois existem já plataformas de análise de redes sociais orientadas à televisão. Assim sendo, iremos analisar essas plataformas seguidamente.

As principais soluções comerciais existentes na análise das redes sociais orientadas à televisão, no presente momento, são:

- *Spredfast*™ [6]
- *Netbase*™ [7]
- *Telescope*™ [8]
- *Wayin*™ [9]

Em seguida iremos fazer uma descrição de cada uma destas plataformas mencionadas, bem como mencionar quem são os seus principais parceiros e as fontes de dados.

3.2.1.1. Spredfast™

É uma plataforma de marketing social que visa envolver as audiências das redes sociais com as marcas.

Fornece visualizações web para os dados agregados das várias redes sociais. Algumas das suas principais fontes de dados encontram-se: *Twitter*™, *Facebook*™, *Instagram*™, *Vine*™, blogues, entre outros. Também fornece alguns tipos de análise sobre os dados recolhidos, tais como: volume, sentimento (positivo, negativo ou neutro), demografia, influenciadores, conteúdo popular, geolocalização e tendências. É parceiro certificado tanto pelo *Twitter*™ como pelo *Facebook*™.

3.2.1.2. Netbase™

Providencia análises nas redes sociais em "tempo real", rápido e preciso. Foca maioritariamente o seu negócio na análise dos dados que recolhe e não providencia visualizações de qualquer tipo.

As suas fontes de dados sociais incluem: *Twitter*™, *Facebook*™, *LinkedIn*™, *TripAdvisor*™, fóruns, blogues, entre outros. Na análise dos dados utiliza processamento de linguagem natural para determinação dos sentimentos, ou seja, verificar se nas redes sociais algo originou um sentimento positivo ou negativo nas suas audiências. É parceiro do *Netflix*®.

3.2.1.3. Telescope™

Desenvolve experiências de participação em tempo real que permitem às empresas de media, marcas e agências analisarem o entusiasmo das suas audiências e entregar os conteúdos mais relevantes e as soluções de compromissos sociais em qualquer ecrã.

As suas principais fontes são: *Twitter*™, *Facebook*™ e *Instagram*™. É parceiro certificado do *Twitter*™ e *Facebook*™, assim como é parceiro de grandes empresas na *Fortune*™ *500*, a qual é uma revista conceituada que publica anualmente um ranking com as quinhentas empresas americanas mais importantes com base nas suas receitas anuais.

3.2.1.4. *Wayin™*

Considera-se uma empresa de inteligência social que integra conteúdo social em novas experiências para os consumidores e providência valor acrescido e controlo às marcas. A *Wayin*™ oferece a habilidade de agregar, integrar e medir os conteúdos e dados sociais em "tempo real", possibilitando assim às empresas, de todas as dimensões, entenderem e extraírem novos valores sobre o marketing social.

Os parceiros da *Wayin*™ são marcas internacionais, equipas desportivas, canais de televisão e agências de publicidade.

3.2.2. Comparação

Neste subcapítulo irá ser feita uma análise da comparação com as plataformas anteriormente descritas, com base nas características mais importantes que a plataforma *SoFly* pretende implementar.

As cinco características no modelo de negócio que a plataforma visa implementar são:

1. *Web Plugins*
2. *Plugins* de Emissão

3. Análises e Monitorização
4. API
5. *Second Screening*

Com base nestas cinco características, foi feita uma tabela de comparação com as quatro plataformas de análise a qual se pode consultar seguidamente.

	Web Plugins	*Plugins* de Emissão	Análises e Monitorização	API	*Second Screening*
Spredfast™	✓	✓	✓	✓	✗
Netbase™	✗	✗	✓	✓	✗
Telescope™	✓	✓	✗	✗	✓
Wayin™	✓	✓	✗	✓	✗

Analisando os resultados demonstrados pela tabela anterior torna-se evidente que nenhuma das plataformas abrange as cinco características que a plataforma *SoFly* visa cobrir. Uma delas, a *Netbase*™, apenas abrange duas características das cinco possíveis. Duas plataformas cobrem três características, existindo no entanto uma diferença entre estas duas plataformas (*Telescope*™ e *Wayin*™), pois apesar de ambas cobrirem os dois tipos diferentes de *plugins* (web *plugins* e *plugins* de emissão), umas delas cobre a característica *Second Screening* enquanto a outra cobre a API, ou seja, uma delas possui *Second Screening* e a outra possui uma API. A única que está perto de cobrir as cinco é a *Spredfast*™, mas mesmo essa não cobre o *Second Screening*.

Com base nesta análise conclui-se que o mercado europeu não possui nenhuma plataforma semelhante orientada à televisão e consequentemente não existe concorrência directa no mercado. Sabendo que existem clientes interessados neste tipo de análises e que os mesmos necessitam dessas mesmas análises para fundamentarem melhor as suas opções de negócio, pois encontram-se neste momento a conduzi-las manualmente apesar de não serem tão abrangentes nem minimamente eficientes na condução das mesmas, a *BroadScope* visa colmatar essa necessidade através da plataforma *SoFly*, tanto a nível nacional como internacional, apesar do segundo apenas a nível europeu numa fase inicial.

4. Desenvolvimento

4.1. Arquitectura Geral

Os estudos efectuados foram essencialmente para ultrapassar problemas encontrados tanto na análise dos requisitos, como na tomada de decisões arquitecturais [1][2][3]. Também foram conduzidos estudos no desenvolvimento de provas de conceito que visavam validar essas mesmas decisões arquitecturais, assim como a sua concordância com os requisitos que lhes deram origem.

Como uma das restrições era a construção dos módulos utilizando a *Framework.NET* e a linguagem de programação *C#*, todos os estudos conduzidos tiveram em conta esse requisito.

Seguidamente será discutida a arquitectura da plataforma *SoFly* no entanto sem entrar em detalhes tecnológicos, sendo que esses mesmos detalhes serão descritos em pormenor no subcapítulo posterior. Os vários módulos existentes na plataforma são:

1. Camada da Dados (*Data Layer*)
2. API de Negócio (*Business API*)
3. *Web Front-End*
4. Mensagens (*Messaging*)
5. Notificações (*Notifications*)
6. Trabalhadores (*Workers*)

A arquitectura da plataforma é visível no esquema imediatamente abaixo, sendo de seguida feita uma descrição dos vários módulos da plataforma nas subsecções sequentes.

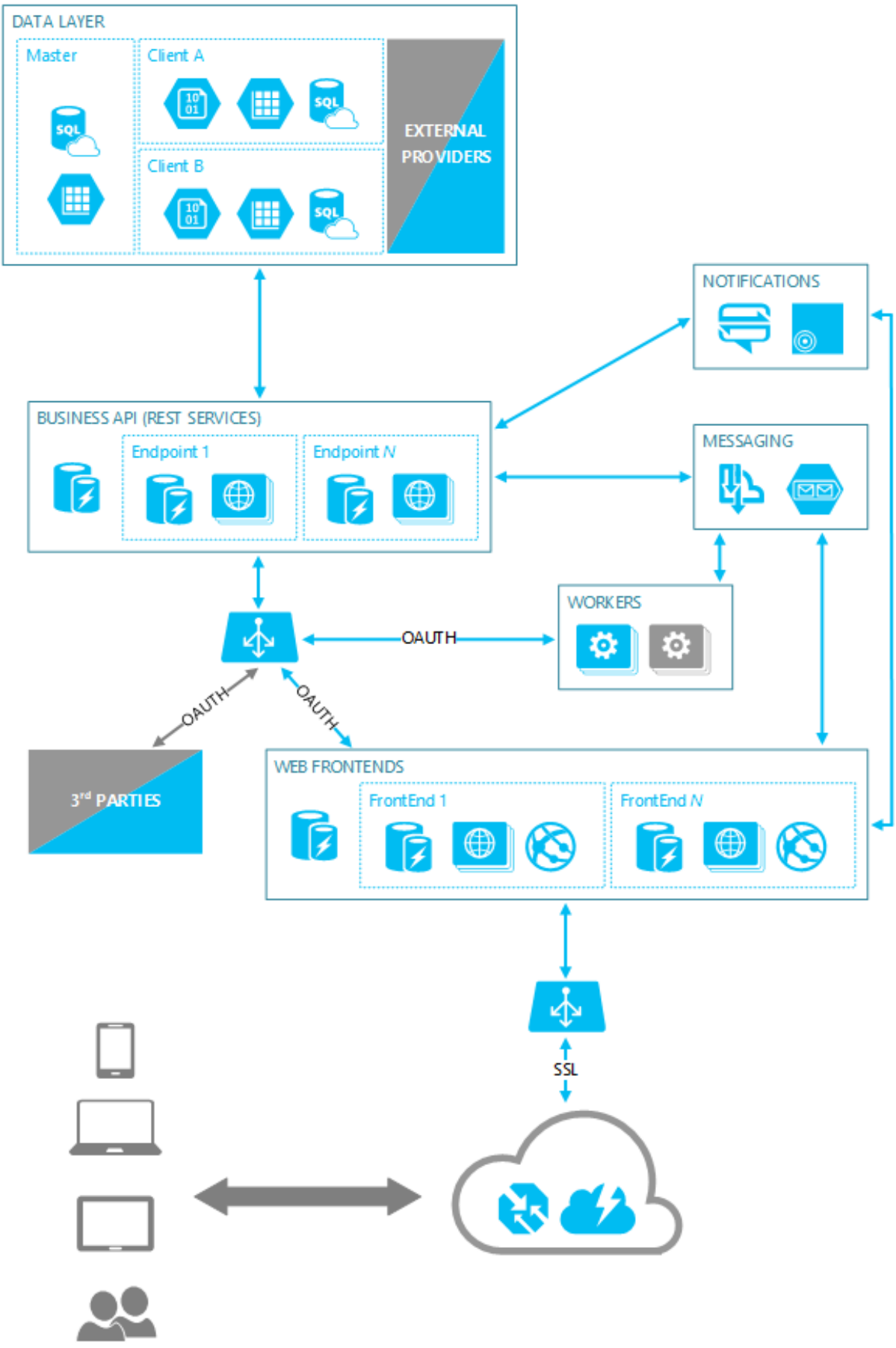


Figura 4 - Arquitectura da plataforma SoFly Analytics

4.1.1. Camada de Dados (*Data Layer*)

Este módulo representa o armazenamento dos dados da plataforma. Foi subdividido em três módulos que são: *Master*, *Client* e *External Providers*, cada um com diferentes funções. O primeiro (*Master*) representa onde os dados gerais, ou comuns, da plataforma são guardados. O segundo (*Client*) é onde os dados dos vários clientes irão ser guardados, garantindo assim a segregação dos dados dos clientes. O terceiro e final sub-módulo representa a possibilidade de escolha pelo cliente, de onde armazenar os dados, seja noutra plataforma na nuvem ou *on-premesis*.

4.1.2. API de Negócio (*Business API*)

O módulo API de negócio representa os serviços expostos e as operações CRUD efectuadas sobre o módulo da camada de dados. Sendo por isso o módulo com acesso directo exclusivo ao módulo da camada de dados. Este módulo é assim responsável por toda a camada de negócio e por todas as instâncias da API.

4.1.3. *Web Front-End*

O módulo *Web Front-End* representa o *front-end* da plataforma e é responsável pelas instâncias das páginas web, bem como pelos *plugins*. As páginas web são utilizadas pelos clientes para acederem ao *front-end* da plataforma *SoFly* e interagirem com a mesma.

4.1.4. Mensagens (*Messaging*)

Este módulo é responsável pela distribuição das mensagens entre os módulos da API de Negócio, *Web Front-End* e Trabalhadores, sendo que o sistema de distribuição das mensagens é bidireccional e assíncrono. Este módulo utiliza para as comunicações dois serviços, um para o envio das mensagens e outro para as guardar até serem entregues ao destinatário, estes dois serviços serão posteriormente analisados com mais detalhe.

4.1.5. Notificações (*Notifications*)

O módulo das notificações é responsável por entregar as notificações entre os módulos *Web Front-End* e API de negócio, estas comunicações funcionam de forma assíncrona e bidireccional, semelhante ao tipo de comunicações no módulo Mensagens. Para este tipo de comunicações existem dois serviços, um serviço web e um serviço para dispositivos móveis, ambos os serviços serão analisados mais à frente em detalhe.

4.1.6. Trabalhadores (*Workers*)

Este módulo representa todos os serviços, ou máquinas, que necessitem de ser chamadas, ou criadas, para efectuarem qualquer tipo de trabalho que seja necessário com os dados armazenados e não só. Alguns desses trabalhos incluem, mas não se limitam a analisar o seu conteúdo, à produção de relatórios com os resultados dessa análise, assim como activar os mecanismos de captura dos dados nas diferentes redes sociais.

4.2. Arquitectura do Sistema

Neste capítulo irá ser descrita a arquitectura da plataforma mais detalhadamente, assim como os módulos desenvolvidos para a mesma. Foram efectuados estudos, já acima referidos, para se validarem as decisões arquitecturais tomadas e a sua concordância com os requisitos que lhes deram origem.

No esquema da arquitectura da plataforma, visível na secção anterior, evidencia-se a cor azul os serviços que correm na plataforma *Microsoft® Azure* ™ [10] e a cinza os que podem correr noutras plataformas na nuvem ou *on-premises*.

4.2.1. Camada de Dados (*Data Layer*)

O módulo Camada de Dados foi dividido em três módulos que são: *Master*, *Client* e *External Providers*. É importante referir que o armazenamento não é feito *on-premises*, mas sim no *Microsoft® Azure™*, que é a plataforma de computação na nuvem da *Microsoft®*.

No sub-módulo *Master* são guardados todos os dados de acesso à plataforma assim como as relações de todas as entidades comuns na base de dado. Estes mesmos dados podem ser guardados tanto num *Microsoft® Azure™ SQL Database* [10], como numa *Microsoft® Azure™ Table Storage* [10] (*Microsoft® structured NoSQL based data*).

O sub-módulo *Client*, no esquema evidenciado como *Client A* e *Client B*, representa o armazenamento e segregação dos dados recolhidos nas redes sociais para cada cliente, o que significa que cada cliente tem a sua própria área de armazenamento isolada. Desta forma evita-se que clientes que tenham muitos dados armazenados influenciem os tempos de acesso aos dados de outros clientes, ou seja, se um cliente tiver muitos dados guardados numa dada base de dados, digamos a título de exemplo, 20 Gigabytes, o tempo de acesso desse mesmo cliente numa dada pesquisa não influencia os tempos de acesso de outro cliente que apenas possui dados reunidos na ordem dos 2 Gigabytes, pois ambos têm a sua própria base de

dados e não uma que possua todos os dados conjuntos, independentemente do cliente que os mandou angariar. Com este tipo de isolamento é permito a cada cliente gerir o tipo de armazenamento e a capacidade de acordo com as suas necessidades singulares. No entanto, este tipo de isolamento tem uma singularidade que importa referir, a duplicação de dados. O que acontecerá é que os dados recolhidos nas redes sociais para cada cliente irão seguramente, algures no tempo, serem guardados em duplicado nas bases de dados dos vários clientes, pois estes não podem aceder aos dados guardados por outros, devido ao isolamento forçado, o que significa que se um cliente guardar um certo *Tweet*, ou publicação no *Facebook*™, outro cliente necessitará também de guardar o mesmo *Tweet* ou publicação no *Facebook*™. Ao garantir esta segregação da informação por cliente, a plataforma diferencia-se do existente no mercado. Neste preciso momento existem três tipos de armazenamento possíveis para o cliente no *Microsoft® Azure*™ disponíveis na plataforma *SoFly*, que são: *Microsoft® Azure*™ *SQL Database*, *SQL Server*™ [10][11], *Microsoft® Azure*™ *Table Storage* e BLOB [10].

O último sub-módulo, *External Providers*, representa a possibilidade de o cliente poder escolher como forma de armazenamento uma base de dados *on-premises* sua, ou qualquer outro tipo de serviço de armazenamento que o cliente use ou pretenda usar, seja na nuvem ou não. Através desta variedade pretende-se possibilitar a liberdade ao cliente de escolher o que melhor se adequa ao seu negócio, sendo esta flexibilidade oferecida ao cliente mais um motivo de diferenciação.

4.2.2. API de Negócio (*Business API*)

A API de negócio expõe os serviços e as operações CRUD sobre as bases de dados assim como toda a parte de negócio da plataforma, permitindo assim o acesso aos dados armazenados, ou vedando-os, dependendo sempre das permissões do utilizador ou serviço. Não existe mais nenhuma maneira de aceder aos dados armazenados sem ser através desta API, que foi pensada para ser possível escalá-la tanto horizontalmente como verticalmente, ou seja, permitir correr em várias instâncias ou simplesmente aumentar a potência das máquinas que actualmente correm a API. Como a API estará a correr no *Microsoft® Azure*™, será bastante simples e rápido escalá-la.

Futuramente irá ser adicionado um sistema de cache à API, para se tentar evitar aceder constantemente às bases de dados sem que seja absolutamente necessário. Para além desta funcionalidade, também se pretende adicionar um sistema de *Load Balancing* no acesso à mesma, quando for necessário escalar para múltiplas instâncias. Este sistema terá por base

o *Microsoft® Azure™ Load Balancer* [10], que permitirá distribuir o tráfico aleatoriamente pelas várias instâncias a correrem a API simultaneamente.

A API, como já foi referido anteriormente, irá fazer uso da *Framework ASP.NET Web API 2* [11], que permite a criação fácil e rápida de serviços HTTP. Neste caso os serviços são totalmente REST e apenas utilizam JSON, podendo se necessário utilizar XML.

A autenticação na API irá ser feita através do *OAuth* [11], versão 2.0, permitindo assim a aplicações cliente acederem aos recursos expostos por ela. Através deste tipo de autenticação é possível fornecer a parceiros o acesso à API, proporcionando assim aos clientes, liberdade para eles próprios desenvolverem aplicações clientes para consumirem os ditos recursos expostos. Um exemplo práctico é o possível acesso de *plugins* aos recursos tornados visíveis para o exterior pela API, *plugins* esses que podem ser desenvolvidos por parceiros ou pela própria empresa.

4.2.3. *Web Front-End*

Este módulo é responsável pela página web disponível para os clientes acederem à sua área e dependendo das permissões de cada utilizador, permitir todas as acções disponíveis ao mesmo. O *front-end* será desenvolvido, como referido anteriormente, fazendo uso da *Framework ASP.NET MVC 5* [11] e publicado no *Microsoft® Azure™ Websites* [10], o que permite facilmente desenvolver e publicar páginas web na plataforma de computação na nuvem da *Microsoft®*. Outra das razões é a facilidade da integração no *Visual Studio®* [11] nesta mesma plataforma, o que permite sem ter de sair da IDE, publicar o *front-end* na plataforma de computação na nuvem da *Microsoft®*, podendo-se assim escalar tanto verticalmente como horizontalmente rapidamente nas configurações anteriormente à ordem de publicação na IDE.

Irá existir uma cache para se evitar fazer pedidos desnecessários à API de negócio, evitando-se assim consumir recursos aquando de pedidos redundantes e sobrecarregar sem necessidade a mesma, semelhante ao sistema de cache descrito anteriormente na API de negócio.

O acesso ao *front-end* irá sempre ocorrer através do protocolo SSL/TLS, utilizando-se para isso o protocolo de comunicações HTTPS, ou seja, irá utilizar a camada HTTP em cima do protocolo SSL/TLS, para assim fazer uso das capacidades de segurança desse protocolo.

Futuramente poderão existir várias instâncias do *front-end*, bastando para tal apenas utilizar a facilidade de escalabilidade proporcionada pelo serviço *Microsoft® Azure™ Websites*, como referido num dos parágrafos anteriores. Com a possível introdução de várias instâncias a correrem simultaneamente irá ser utilizado um *Load Balancer*, para decidir a qual instância a conexão será feita, que irá distribuir aleatoriamente o tráfico pelas múltiplas instâncias a correrem nas máquinas virtuais.

O acesso ao *front-end* pelo exterior irá passar pelo *Microsoft® Azure™ Traffic Manager* [10] assim como pelo *Microsoft® Azure™ Content Delivery Network* (CDN) [11], para se melhor controlar o acesso e o tráfego ao mesmo, apenas no contexto de fluxo de acessos e não de segurança. Em baixo irá ser feita uma descrição do uso de cada um dos serviços.

O *Microsoft® Azure™ CDN* permite entregar globalmente conteúdos de uma forma rápida através da cache de conteúdos estáticos e de *BLOBs* em nós físicos espalhados pelo mundo inteiro, alguns exemplos são: América do Norte, Europa, Ásia e América do Sul. Deste modo garante-se a entrega destes mesmos conteúdos aos utilizadores do *front-end* através do *content delivery network* mais perto dos utilizadores.

Em relação ao *Microsoft® Azure™ Traffic Manager*, este permite o controlo da distribuição do tráfego dos utilizadores para os *endpoints* especificados. Este sistema funciona aplicando políticas nas *queries* no motor de DNS, permitindo também, entre outras coisas, a possibilidade de se remover um *endpoint* para manutenção ou actualização sem necessidade de desligar o serviço totalmente, bastando somente voltar a adicionar o *endpoint* assim que a manutenção ou actualização tiver terminado.

4.2.4. Mensagens (*Messaging*)

O módulo Mensagens é responsável pela gestão e entrega de mensagens na plataforma. Este módulo possui um sistema de fila de mensagens do tipo publicador/subscritor a correr num *Service Bus* no *Microsoft® Azure™*. Em seguida será feita uma descrição destes dois serviços mais detalhadamente.

O *Service Bus* é o sistema de distribuição de mensagens entre as várias instâncias da API e do *front-end*, bem como do módulo Trabalhadores, garantindo-se assim que todas as mensagens são entregues às várias instâncias dos serviços. Este serviço corre no *Microsoft® Azure™*, como foi referido anteriormente e denomina-se *Microsoft® Azure™ Service Bus* [10]. O sistema de filas de mensagens é do tipo publicador/subscritor, sendo este um serviço

bidireccional e assíncrono, não é preciso por isso que o publicador e o subscritor interajam ao mesmo tempo.

O *Microsoft® Azure™ Storage Queue* [10] é um sistema de armazenamento de mensagens, o qual para além do armazenamento fornece um interface baseado em REST com os seguintes comandos: *Get/Put/Peek*. O comando *Get* permite ir buscar a mensagem, o comando *Put* possibilita alterar a mensagem sem a retirar da pilha e o *Peek* permite verificar o conteúdo da mensagem sem a remover da fila. O *Microsoft® Azure™ Storage Queue* providencia assim um serviço de mensagens fiável e persistente.

4.2.5. Notificações (*Notifications*)

Este módulo é responsável por entregar todo o tipo de notificações geradas nos módulos API de negócio e *Web Front-End*. Estas notificações podem ser entregues por dois métodos distintos: *Microsoft® Azure™ Notifications Hub* [10] e *ASP.NET SignalR* [11]. Ambos os métodos serão descritos mais em pormenor seguidamente.

O *Microsoft® Azure™ Notification Hub* fornece uma infra-estrutura fácil de usar, que permite enviar notificações do tipo *push* para quaisquer dispositivos móveis compatíveis com o serviço, alguns exemplos dos dispositivos móveis compatíveis são: *iOS™*, *Android™* e *Windows Phone™*. Esta infra-estrutura é disponibilizada através da plataforma de computação na nuvem *Microsoft® Azure™*.

O *ASP.NET SignalR* é uma biblioteca para a *Framework ASP.NET* [11] que permite adicionar funcionalidades em tempo real a clientes web. O que isto significa é que possibilita o envio de mensagens do lado do servidor, fazendo *push* das mensagens em tempo real para os clientes conectados e registados. Esta biblioteca utiliza *web sockets*, quando disponíveis, para as comunicações, comunicações essas que são bidireccionais entre o servidor e o *browser*. Outra das funcionalidades permitidas é chamar funções *JavaScript* no *browser* no lado do servidor, assim como permitir criar grupos. Estas funcionalidades aqui descritas permitem à plataforma o envio de mensagens para o *front-end*, bem como para os *plugins* registados dos parceiros. A possibilidade de existirem grupos é uma funcionalidade importante, pois permite subscrever vários clientes a um dado grupo, que por sua vez permite filtrar as mensagens do publicador a grupos, assim uma notificação é enviada para um determinado grupo e todos os clientes pertencentes a esse grupo irão receber a notificação, sem haver por isso

necessidade de se implementar mecanismos para manter listas de clientes, ligados ou não, pois o *ASP.NET SignalR* permite gerir isto internamente.

4.2.6. Trabalhadores (*Workers*)

O módulo *Workers* representa todo o tipo de trabalho que seja necessário efectuar, seja com os dados armazenados ou não. Alguns dos exemplos desses trabalhos são: análise semântica sobre o conteúdo recebido das redes sociais, produção de relatórios das análises efectuadas sobre esses mesmos dados, activação dos mecanismos de captura dos dados pedidos pelo cliente numa ou mais redes sociais, entre outros trabalhos que sejam precisos. Estas tarefas podem ser executadas nos *Microsoft® Azure™ Cloud Services* [10], ou noutros serviços, dependendo sempre do tipo de tarefas que sejam necessárias.

No caso de serem executas em *Microsoft® Azure™ Cloud Services*, então irão correr em máquinas ou instâncias na plataforma de computação na nuvem da *Microsoft®*, o *Microsoft® Azure™*. Máquinas, ou instâncias, que apenas existirão pela duração da tarefa, ou seja, são automaticamente criadas e apagadas assim que o trabalho seja concluído, evitando-se assim consumir recursos desnecessariamente e reduzir os custos operacionais da plataforma.

4.3. Implementação

Nesta secção será explicado todo o desenvolvimento efectuado durante a segunda fase do projecto, o qual corresponde à implementação dos estudos efectuados na primeira fase e descritos nos capítulos anteriores. Os diagramas de classes de cada módulo encontram-se nos anexos, separados por módulos.

4.3.1. Camada de Dados (*Data Layer*)

A camada de dados, como mencionado anteriormente, é o módulo responsável por todo o armazenamento da plataforma. Devido a esse facto, é onde se encontram as configurações e mecanismos de acesso às bases de dados, bem como os objectos representados nessas mesmas bases de dados, os quais serão doravante designados de entidades.

Como os dados foram separados em dois grupos, um contendo os dados gerais da plataforma e o outro os dados dos clientes, todos os nomes das classes neste módulo, exceptuando as entidades e as enumerações (*Enums*), contem no nome uma referência ao grupo do qual pertencem, sendo denominados de *Master* e *Client* respectivamente.

Os diagramas das duas bases de dados *Client* e *Master* foram construídas com utilização de uma ferramenta, o *SQL Server 2014 Management Studio*, encontrando-se estes à parte no projecto da camada de dados, pois assim permite alterá-lo independentemente dos projectos dos módulos. Essa independência permite-nos gerar scripts das bases de dados, de actualização por exemplo, quando necessário, sem alterar previamente o código fonte dos módulos se o mesmo não for requerido.

Foram criadas duas classes que gerem os acessos a esses dois grupos (*Master* e *Client*), no entanto existem também duas outras classes que providenciam os respectivos contextos para cada um dos grupos, ou seja, uma contem o contexto para o *Master* e outra para o *Client*. Por contexto, entenda-se que providenciam programaticamente o conhecimento da estrutura das bases de dados, bem como os métodos necessários para manipular esses ditos dados. Foi também construído o contexto para o acesso à *Storage Account* da *Microsoft*, o qual permite aceder a dois tipos de armazenamento: o *Table Storage* e o BLOB.

Para além dos contextos e configurações criadas, foi também criada uma entidade por tabela.

É também importante referir que, para além das classes anteriormente mencionadas, foram também criadas excepções, extensões e enumeradores. As extensões são classes estáticas (*static*) que extendem outras classes, ou seja, permitem "adicionar" métodos a tipos de objectos existentes sem criar derivações ou alterar esses tipos originais.

4.3.2. API de negócio (*Business API*)

O módulo API de negócio é responsável por gerir todos os acessos ao módulo camada de dados, bem como o único com acesso directo ao dito. Devido a esse facto, é onde se encontra todo o negócio e os pontos de acesso do exterior à plataforma.

Como neste módulo é onde está todo o negócio, todos os objectos de negócio (*Business Objects*) se encontram no mesmo. Para cada objecto de negócio foram sempre criadas três classes, classes essas que possuem todos os métodos necessários para trabalhar com os objectos de negócio e são criadas com base numa nomenclatura bem definida. Essa nomenclatura define que os nomes dessas classes sejam geradas da seguinte forma: *[BusinessObject]Locator*, *[BusinessObject]Manipulator* e *[BusinessObject]ConversionExtensions*, sendo que *[BusinessObject]* se refere ao nome do objecto de negócio ao qual as classes dizem respeito. Para melhor compreensão da nomenclatura utilizada no nome das três classes, será dado um exemplo: para o objecto de negócio *SocialMessage* foram criadas as classes *SocialMessageLocator*,

SocialMessageManipulator e *SocialMessageConversionExtensions*, sendo essas classes genéricamente denominadas de *locator*, *manipulator* e *conversion*, respectivamente. A classe *locator* é onde se encontram todos os métodos para a localização (*read*) de dados sobre o seu respectivo objecto de negócio, o que significa que todas as pesquisas existentes sobre este objecto de negócio estão nesta classe. A nomenclatura para os métodos nos *locators* especifica que os mesmos devem sempre se iniciar com o prefixo *Find*. Os metódos para manipulação dos objectos de negócio encontram-se todos na classe *manipulator*, no entanto como referido previamente, cada objecto de negócio tem o seu próprio *manipulator*. Os métodos de manipulação centram-se sempre na criação (*create*), actualização (*update*) e remoção (*remove*). A nomenclatura dos métodos nos *manipulators* estipula que na criação, remoção e actualização de dados os métodos se devem designar sempre por *Add, Remove* e *Update*, respectivamente. Com a utilização das classes *locators* e *manipulators* obtém-se todos os métodos CRUD para um dado objecto de negócio. A extensão *conversion* é unicamente responsável por transformar as entidades de armazenamento de dados em objectos de negócio e vice-versa. Para uma melhor compreensão, será apresentado na imagem seguinte um fluxo simplificado decorrente de um pedido para a localização de um objecto de negócio num *locator*.

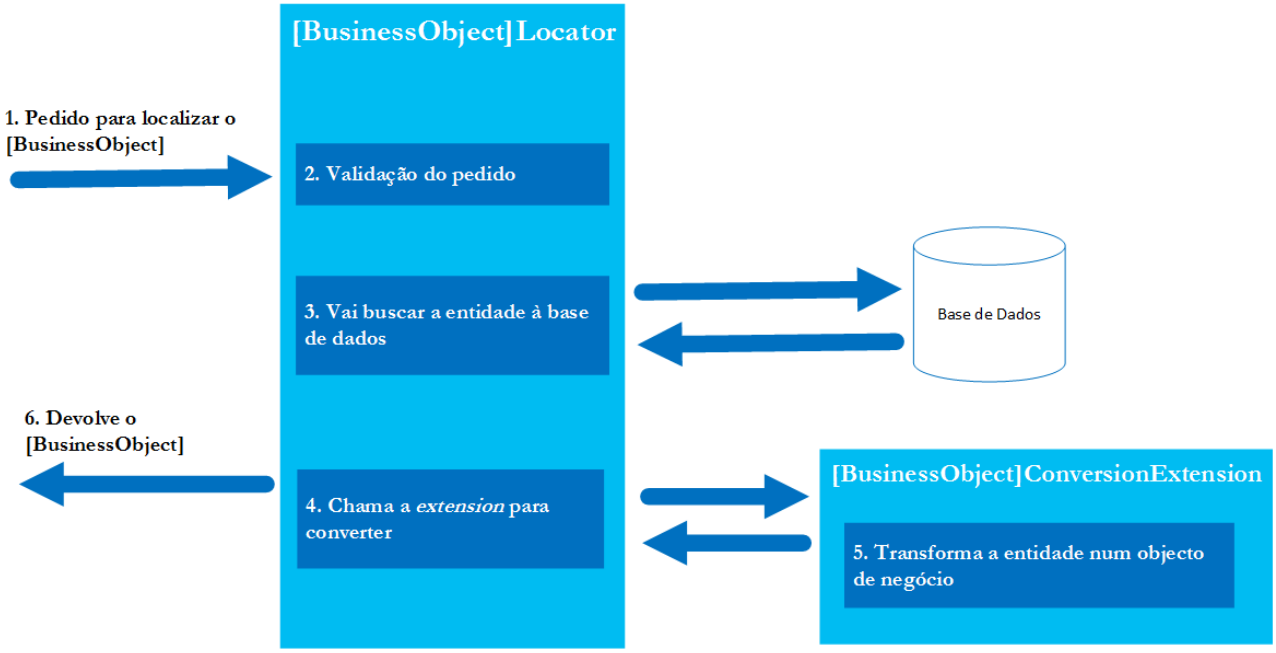


Figura 5 - Exemplo do fluxo num locator

Todos os objectos de negócio derivam obrigatoriamente de uma classe abstracta (*abstract class*) denominada *CommonBase*. Esta classe possui todas as propriedades obrigatórias e comuns aos objectos de negócio, os quais são: *ETag*, *HasBeenLoaded* e *IsValid*. O primeiro guarda o valor da *eTag* da entidade correspondente, ou seja, a versão da entidade na base de

dados quando foi feita a conversão de entidade para objecto de negócio, o segundo possui apenas dois valores possíveis: *true* ou *false*, apenas possuindo o valor de *true* se o objecto estiver com todas as propriedades carregadas da sua correspondente entidade, no entanto se estiver a *false* apenas possui o valor do identificador carregado, ou seja, o *Identifier* (Id) da tabela correspondente; o terceiro e último corresponde a uma validação de se os dados carregados são válidos ou não, retornando *true* ou *false* conforme o resultado dessa dita validação. Esta validação varia consoante o objecto de negócio, sendo que cada um possui propriedades obrigatórias diferentes, contento no entanto algumas semelhantes.

É também importante referenciar que existem classes denominadas de *relators* que servem para gerir relacionamentos entre objectos de negócio. A nomenclatura especifica que o nome da classe é resultante da concatenação dos nomes dos dois objectos de negócio envolvidos, acrescentado o sufixo *Relator*, ou seja, *[BusinessObject1][BusinessObject2]Relator*. Para uma melhor compreensão da nomenclatura é dado em seguida um exemplo: para relacionar os objectos de negócio *Subscription* e *SocialConnector*, foi criado o seguinte *relator* (tendo por base a nomenclatura anteriormente descrita) *SubscriptionSocialConnectorRelator*. A nomenclatura definida para os métodos nas classes *relators* especifica que os métodos para criação de associações entre dois objectos de negócio seja designado de *Associate*, o de actualização de associações de *UpdateAssociation* e o de remoção de associações de *Dissociate*. Para clarificar os motivos da nomenclatura utilizada nos metodos e a relevância destas classes, será dado em seguida um exemplo de um *relator* existente na plataforma: se se pretender associar uma subscrição (objecto de negócio *Subscription*) a um conector social (objecto de negócio *SocialConnector*), é invocado um *relator* para o realizar, neste caso específico o *SubscriptionSocialConnectorRelator* e executado o método *Associate*, sendo que para se desfazer essa relação é utilizado o método *Dissociate*. Por último, para actualizar é executado o método *UpdateAssociation*.

Para além da parte de negócio se encontrar toda situada neste módulo, também os acessos do exterior se encontram aqui, ou seja, todos os *endpoints* disponibilizados pela API. Cada objecto de negócio possui o seu controlador de API, controlador esse onde se ecnontram disponíveis todos os possíveis pontos de acesso ao exterior desse recurso. Como controlador entenda-se a classe que permite adicionar e configurar os acessos externos a um dado recurso, recurso esse que é um objecto de negócio. Todos os pontos de acesso, criados nestes ditos controladores, são totalmente RESTfull e fazem uso de JSON para a transmissão e recepção dos dados. Estes ditos controladores, dependendo da natureza do pedido,

chamam os *locators* e/ou os *manipulators* para as operações CRUD e apenas aceitam pedidos HTTP dos seguintes tipos:

- GET (para localizações)
- POST (para criações)
- PUT (para actualizações)
- DELETE (para remoções)

4.3.3. *Web Front-End*

O *Web Front-End* é o módulo no qual os clientes podem visualizar e interagir com a plataforma num contexto visual. Este módulo faz uso do módulo API de negócio para aceder aos dados, o que significa que utiliza exactamente os mesmos *endpoints* externos que a API providencia para o exterior, não existindo por isso qualquer diferenciação entre este módulo ou outra aplicação desenvolvida por terceiros. É também importante referir que todo o *front-end* foi desenhado com base no padrão MVC.

Este módulo foi subdividido nos seguintes módulos visuais:

1. Autenticação dos utilizadores
2. Gestão das streams
3. Gestão de temas para plugins
4. Gestão de plugins
5. Gestão de subscrições
6. Gestão do utilizador
7. Gestão da audiência

1. **Autenticação dos utilizadores:** Este módulo centrou-se na autenticação dos utilizadores da plataforma e subsequentemente na API de negócio, para assim o utilizador não só poder aceder à plataforma como aos dados necessários para a correcto funcionamento da mesma. Utilizando a tecnologia *ASP.NET Identity*, a autenticação foi baseada em *claims* e *cookies* de sessão, conforme estudado na primeira fase do projecto. O uso dessa tecnologia é explicado mais em pormenor no próximo capítulo.
2. **Gestão das *streams*:** O módulo de gestão de *streams* visa possibilitar aos utilizadores poder gerir *streams*. As *streams* não são mais que fontes de informação social o que no caso da rede social Facebook significa que são páginas dessa

mesma rede. Quando um *stream* é criado, é enviado um pedido para a API de negócio, a qual notifica um trabalhador que é necessário recolher dados numa determinada rede social (mais uma vez, no caso do *Facebook* é uma página nessa rede social), o que significa que o trabalhador vai recolher todos os *posts*, bem como todos os comentários existentes nessa dita página, para além dos *likes* e *shares* desses *posts* e comentários.

3. **Gestão de temas para plugins:** Neste módulo os utilizadores gerem temas para os *plugins* existentes, ou seja, criam, actualizam e apagam temas para qualquer tipo de *plugin* existente. Existe também uma pré-visualização na página de criação e edição dos temas, possibilitanto assim aos utilizadores verem as alterações a ocorrerem no momento em que as realizam.
4. **Gestão de plugins:** Possibilitam aos utilizadores da plataforma criar *plugins*, bem como editá-los ou mesmo apagá-los. Para ser possível crial plugins tem de existir pelo menos um tema, pois o plugin não pode ser criado sem um tema, assim como já tem de existir pelo menos um *stream* criado, pois o utilizador irá ter de seleccionar um dado *post* numa das *streams* existentes. À semelhança do módulo anterior também existe uma pré-visualização, permitindo assim ao utilizador verificar qual o tema que pretende e se se adequa ao *post*.
5. **Gestão de subscrições:** Neste módulo os utilizadores gerem as subscrições da plataforma, ou seja, criam, actualizam e apagam projectos na plataforma, pois cada subscrição pode ter vários projectos, bem como gerem os conectores sociais da mesma. Os conectores sociais são o que a plataforma utiliza para fazer os pedidos nas redes sociais, pois necessita que sejam criadas *apps* sociais nessas mesmas redes, para assim obter as permissões necessárias para obter dados dessas mesmas redes. No entanto, existe uma *app* de sistema, mas apenas é utiliza em último recurso devido à possibilidade da plataforma não conseguir obter todos os dados relevantes, pois o utilizador social não deu consentimento na utilização dessa *app* para a obtenção de dados do mesmo na rede social.
6. **Gestão do utilizador:** Possibilita aos utilizadores gerirem os seus dados na plataforma, nos quais se inclui alterar os seguintes dados próprios: imagem de perfil, primeiro nome, último nome e palavra-passe.
7. **Gestão da audiência:** Este módulo permite aos utilizadores da plataforma visualizar os utilizadores sociais de uma determinada rede social que interagiram num dado *stream*, *stream* esse criado no módulo de gestão das *streams*. É importante

mencionar que é neste módulo que os utilizadores visualizam todas as análises efectuadas sobre os utilizadores sociais.

Como foi referido anteriormente, o *front-end* foi desenvolvido com base no padrão MVC. As *views* foram construídas tendo como base HTML, *Razor syntax* e CSS para a estilização, sendo que as tecnologias referidas serão explicadas mais em pormenor no capítulo seguinte. Para além das *views*, foram também utilizadas as denominadas *shared views* (Vistas Partilhadas), as quais são, de uma forma simplista, *views* que podem ser utilizadas dentro de quaisquer outras *views*, minimizando-se assim o HTML duplicado. Os *models* são utilizados para passarem os dados dos controladores para as *views*. Foi também utilizado JavaScript nas *views* sempre que necessário, por exemplo: na geração de gráficos, no *upload* de ficheiros, bem como na geração de *templates*. Para a geração de gráficos foi utilizada uma biblioteca denominada Kendo UI.

Os *plugins web* foram criados inteiramente utilizando JavaScript, HTML e CSS. Estes *plugins* fazem uso de algumas bibliotecas, das quais se destacam: *jQuery* e *Signal*R. O *jQuery* para facilidade de manipulação do DOM dos *plugins* e o *Signal*R para as comunicações em "tempo real".

4.3.4. Mensagens (*Messaging*)

O módulo mensagens é onde são geridas as mensagens provenientes dos restantes módulos. Estas mensagens servem para os módulos comunicarem entre si, quando necessário.

Foram desenvolvidos métodos para a utilização do *Microsoft® Azure™ Service Bus* e o *Microsoft® Azure™ Storage Queue* para a gestão das mensagens, bem como foram desenvolvidos objectos que são as mensagens enviadas.

Foi desenvolvida uma classe para gerir os acessos à *queue* e onde foram implementados todos os métodos que providenciam interacções com a *Storage Queue*. Também foram implementadas as classes utilizadas na *queue* como mensagens para a transmissão de informação entre módulos.

Para além das classes referidas acima, também foram criadas extensões para facilitar o desenvolvimento, a clareza e a simplicidade do código, para assim melhorar drasticamente a manutenção do mesmo.

4.3.5. Notificações (*Notifications*)

Este módulo providencia comunicação em "tempo real" entre os módulos API de negócio e o módulo *Web Front-End*, conforme mencionado anteriormente.

As ditas comunicações em "tempo real" são geridas com base em duas tecnologias: *ASP.NET SignalR* e *Microsoft® Azure™ Notification Hub*. A primeira para clientes *web* e a segunda para cliente móveis.

Foi criada uma classe a qual gere todas as notificações, a *NotificationClient*. Esta classe também gere as ligações existentes dos clientes, bem como efectua *push* dos dados.

Para as comunicações, foi criada uma classe abstracta denominada *NotifiableHub*, a qual todos os *hubs* obrigatoriamente implementam. Esta classe foi implementada para se fazer override aos métodos de gestão das ligações dos *hubs*, os quais são herdados da classe abstracta *Hub* da biblioteca *ASP.NET SignalR*, bem como possuem os métodos implementados de gestão de grupos, sendo que o funcionamento da dita biblioteca será explicado no próximo capítulo. Um *hub* é uma classe que possuí métodos expostos para o exterior, métodos esses que os clientes registados podem evocar. Entenda-se por clientes, os clientes *web* que se registaram (*plugins* e o *front-end*), sendo a classe *NotificationClient* responsável pela gestão dos clientes registados, bem como pelo envio dos dados para estes por *push*. Todos os *plugins* se registam para receberem os dados em "tempo real", no entanto é importante mencionar que os *plugins* se registam em grupos, ou seja, os *plugins* registam-se num determinado grupo no *NotificationClient*, grupo esse que identifica inequivocamente a *stream* e neste caso um *post* específico no Facebook, pois no presente momento apenas foram criados *plugins* para *posts* derivando esse facto unicamente da necessidade dos clientes.

4.3.6. Trabalhadores (*Workers*)

O módulo trabalhadores é responsável por todo o trabalho efectuado na plataforma. Esse trabalho vai desde a recolha de dados nas redes sociais à análise dos mesmos, como já foi referido inúmeras vezes anteriormente. Para os diferentes trabalhos necessários, foram criados diversos processos (trabalhadores) para os realizar.

Para a recolha de dados nas redes sociais foi criado um processo central denominado de *SocialGraber*, o qual recebe mensagens oriundas da API de negócio, quando a mesma necessita de recolher dados nas redes sociais. Este processo assim que recebe esta mensagem, levanta um processo à parte, processo esse que irá recolher esses ditos dados na rede social pretendida e reenviar os dados processados à API de negócio para a mesma os processar e

guardar na camada de dados. Quando esse trabalho estiver terminado o processo irá notificar o processo central e cessará as suas actividades.

Para a análise dos dados recolhidos nas redes sociais, existe um processo que analisa periodicamente os novos dados recolhidos das redes sociais e os processa de acordo com a análise pretendida.

É importante referir no entanto, que nenhum dos trabalhadores teve intervenção directa do estagiário, sendo o mesmo apenas responsável pela integração dos vários trabalhadores na plataforma e não no desenvolvimento dos ditos trabalhadores.

4.4. Ferramentas e Tecnologias

Um dos requisitos era o uso da *Framework* ASP.NET e a linguagem C#. Todas as soluções encontradas tiveram essa restrição em conta. Para todos os desafios encontrados foram desenvolvidas provas de conceito que visavam garantir que as tecnologias e opções tomadas garantiam a concordância com os requisitos, requisitos esses descritos anteriormente.

A plataforma, *SoFly*, foi construída tendo por base tecnologias *Microsoft*®, mais especificamente a *Framework* .NET [11]. Devido a isso a IDE escolhida para o desenvolvimento foi o *Microsoft*® *Visual Studio*® *Ultimate Edition*, mais concretamente a versão 2013, que é a mais recente. A plataforma foi desenvolvida para tirar partido da infra-estrutura fornecida pelo serviço de computação na nuvem da *Microsoft*®, o *Microsoft*® *Azure*™, escolha essa evidenciada no subcapítulo posterior, no qual se compara e justifica a escolha dessa mesma plataforma. A utilização do *Visual Studio*®, mais uma vez facilita o acesso aos recursos e serviços fornecidos pelo *Microsoft*® *Azure*™, facilitando, por exemplo, a publicação de *websites* na nuvem assim como de outros serviços, que são vários, fornecidos pela plataforma de computação na nuvem da *Microsoft*®. Na apresentação da primeira versão da arquitectura da plataforma *SoFly*, procedeu-se ao debate da proposta com o responsável técnico, neste caso era o orientador do estágio na empresa, e realizaram-se alguns ajustes e melhoramentos. Sendo então iniciada a esquematização da versão final, versão essa descrita e esquematizada em pormenor numa secção anterior.

Para os testes unitários foi utilizada uma *framework* denominada NUnit devido à sua curva de aprendizagem e fácil integração na IDE *Visual Studio*®. Para os testes funcionais

foram empregadas duas aplicações: o *Fiddler*™ [12] e o *Wireshark*™ [13], dependendo sempre da singularidade do próprio teste.

Para o desenvolvimento do *front-end* da aplicação, foi escolhida a *Framework* ASP.NET MVC *5*, devido à familiarização e imposição da tecnologia da *Microsoft*®, a *Framework* ASP.NET, bem como da sua fácil integração no *Microsoft*® *Azure*™ através dos *Microsoft*® *Azure*™ *Websites*. Utilizando o *Visual Studio*® facilmente se publica um *website* no *Microsoft*® Azure™ e o torna-mos disponível publicamente. Para a autenticação no *front-end* é utilizada a tecnologia ASP.NET *Identity 2* [11] em conjunto com a *Framework* OWIN [11], que permite controlo total sobre os esquemas na base de dados da entidade Utilizador e da informação pretendida nos perfis dessa entidade, guardando esses mesmos dados numa base de dados, seja uma base de dados relacional ou não, para se garantir assim a persistência dos mesmos. Permite fornecer também perfis aos utilizadores, por exemplo digamos um perfil chamado "Admin" sem restrições no acesso e outro denominado de "Normal" o qual apenas terá acesso à leitura de dados e não de escrita nem de alteração dos mesmos, facilitando assim atribuir restrições de acesso com base nesses mesmos perfis. Para além da possibilidade de atribuição de perfis, também é permitido usar a tecnologia de autenticação com base em *Claims*, na qual a identidade do utilizador também possui representado quem o utilizador é ou não é, para além das suas permissões para fazer algo ou não. Desde a ASP.NET *Identity 2* que a autenticação tem por base o *middleware* OWIN, o qual é *open-source*, o que significa que utiliza a *Framework* OWIN para geração de *cookies* de autenticação, cookies essas que possuem as já designadas *Claims* para alem das restantes informações de segurança. Para acelerar o acesso ao *front-end* irá ser utilizado o *Microsoft*® *Azure*™ CDN e o *Microsoft*® *Azure*™ *Traffic Manager*, o primeiro permite aumentar a rapidez de acesso através da cache de conteúdo estático, o segundo através do controlo do tráfego dos utilizadores. Ambos os serviços permitem através da geolocalização dos utilizadores gerir os acessos aos conteúdos, redireccionados para os *datacenters* do *Microsoft*® *Azure*™ mais pertos. Foi também utilizado o *Kendo* UI para a geração de gráficos no front-end, o qual é desenvolvido inteiramente em JavaScript.

Ao nível da API o maior desafio era encontrar uma forma de garantir a segurança desta, mas ao mesmo tempo esse mecanismo não poderia vedar o acesso dos dados aos parceiros e nem aos *plugins* web, pois não é viável inserir o nome de utilizador e a palavra-passe neste tipo de *plugin,* bem como não os expusesse a acessos não autorizados. Na segurança da API, escolheu-se utilizar autenticação com recurso a *OAuth 2*, pois permite a autenticação com recurso a *access tokens* e *refresh tokens*, evitando-se assim o constante pedido

de nome de utilizador e palava-passe ao utilizador, o que tornava inviável, a título de exemplo, o uso de *plugins* webs, em *JavaScript*. Os *access tokens* e os *refresh tokens* são *tokens* criados na base de dados pela primeira vez quando um utilizador se autentica com nome de utilizador e palavra-passe. No entanto os *refresh tokens* servem apenas para a aplicação pedir um *access token* novo, quando este expira ou foi revogado, daí advém o nome *refresh token*. Os *access tokens* são enviados através dos pedidos, no cabeçalho do pedido, como forma de garantir a autenticidade dos mesmos, tendo contudo uma curta longevidade. Na API de negócio foi utilizada a *Framework ASP.NET Web API 2,* juntamente com autenticação *OAuth* 2 e *ASP.NET Identity 2*, à semelhança do *front-end* que também empregou ambas as tecnologias de autenticação. A API de negócio foi totalmente construída para respeitar e usar a arquitectura REST, sendo por isso totalmente *Restful* e somente utilizando JSON. Na API o acesso à base de dados é feita com a ajuda da *Entity Framework 6* [11] para todas as operações do tipo CRUD, pois permite uma abstracção sobre a camada de dados. A *Entity Framework* é uma ORM *open-source* para *ADO.NET* [11] que é um mecanismo COM criado pela *Microsoft*®.

No módulo das bases de dados, foi escolhido utilizar o *Microsoft® Azure™ SQL Database* (anteriormente denominado *SQL Azure™*) na base de dados *Master*, sendo que o armazenamento dos dados recolhidos nas redes sociais ficará segregado para cada cliente, cujo tipo de armazenamento poderá ser escolhido pelo mesmo. Os diferentes tipos de base de dados permitidos presentemente são os seguintes: *Microsoft® Azure™ SQL Database*, *SQL Server®*, *Microsoft® Azure™ Table Storage* (*Microsoft® structured NoSQL based data*) e BLOB. Um dos desafios encontrados foi proporcionar suporte para o controle de concorrência optimística dos dados no qual se optou por empregar *ETags,* devido à facilidade e simplicidade das *ETags* em garantir o controlo concorrencial sobre os dados armazenados. As *Etags* consistem em um identificador associado a uma versão de um recurso, esta muda sempre que o recurso mudar.

Um dos desafios, este ao nível do sistema de envio de mensagens, visava garantir que o envio de mensagens entre as diversas partes do sistema, garantia desacoplar as várias componentes do dito sistema, bem como a escolha das técnicas a utilizar no mesmo. Foi então escolhido utilizar um sistema de fila de mensagens, do tipo publicador/subscritor, sobre um *service bus*, funcionalidade importante quando for necessário escalar a API ou o *front-end*. Assim foi utilizado o *Microsoft® Azure™ Service Bus* e o *Microsoft® Azure™ Storage Queue*, que possibilitam o uso de mecanismos distribuídos de envio de mensagens, que permitem então garantir o desacoplamento entre as diversas partes do sistema. Fornecendo também a eventualidade de escolher a melhor técnica a utilizar neste mesmo sistema, entre

os quais: publicador/subscritor, filas de mensagens, garantias de entrega "pelo menos uma vez" e/ou "no máximo uma vez".

Para a entrega de notificações, tanto no *front-end* como nos *plugins* em tempo real, foi escolhida a biblioteca *ASP.NET SignalR*, a qual faz uso da tecnologia *push*. Permitindo assim enviar os dados aos clientes web sem estes terem de os requisitar, ou seja, quando existirem dados novos o servidor toma a iniciativa de enviar esses mesmos dados aos clientes registados, ao invés da tradicional tenologia *pull*, no qual os clientes é que tem a obrigação de periodicamente fazerem a verificação da existência de dados novos e na eventualidade de existirem, os irem buscar. Com esta abordagem evita-se, para além de não ser necessário criar qualquer lógica para ir periodicamente verificar a existência de dados novos, gastar largura de banda e recursos no lado do cliente, ao não se criar pedidos que são perfeitamente escusáveis e redundantes a maioria das vezes. Esta tecnologia faz uso primeiramente de *web sockets,* o que permite para além de outras funcionalidades, a de comunicações bidireccionais, apenas quando estas não estiverem disponíveis reverte para os tipos antigos de transporte de dados, mais especificamente *forever frame*, *server-sent events* e *long polling*, sendo este tipo de gestão interno à biblioteca.

No entanto para a entrega de notificações, neste caso nos dispositivos móveis, foi escolhido utilizar o *Microsoft® Azure™ Notification Hub*, que como o ASP.NET *SignalR* permite enviar notificações usando a tecnologia *push,* ou *server push*, mas ao contrário do ASP.NET *SignalR* apenas tem como cliente final os dispositivos móveis e não os clientes web. Alguns desses dispositivos incluem, mas não se restringem, aos que correm os sistemas operativos móveis: *Android™*, *iOS™* e *Windows Phone®*. Esta infra-estrutura, mais uma vez, é disponibilizada através do *Microsoft® Azure™*.

Nos subcapítulos seguintes proceder-se-á a uma análise que visa esclarecer algumas das escolhas tomadas, mais concretamente sobre a escolha da plataforma de computação na nuvem, o tipo de armazenamento de informação utilizado, o uso e tipo de intermediário de mensagens e as web *application frameworks* eleitas para o desenvolvimento dos módulos web.

4.4.1 Plataformas de Computação na Nuvem

Neste subcapítulo proceder-se-á à comparação dos fornecedores de serviços/plataformas na nuvem. Foram apenas considerados os fornecedores que possuam serviços/plataformas em múltiplos locais a nível internacional, um dos requisitos, razão pela qual nenhum fornecedor nacional foi tido em consideração.

Foram considerados três fornecedores como passiveis de serem utilizados, nomeadamente:

- *Amazon® Web Services* [14]
- *Microsoft® Azure™*
- *Google® Cloud Platform* [15]

Em seguida será feita uma descrição dos serviços/plataformas suportados por cada um dos três fornecedores considerados.

4.4.1.1. *Amazon® Web Services*

A *Amazon®* é uma empresa de comércio electrónico, iniciou-se a vender apenas livros, mas agora vende de tudo um pouco, sendo generalista. Em 2006 iniciou-se na computação na nuvem, sendo correntemente um dos maiores provedores de serviços na nuvem.

A *Amazon® Web Services* possui vários serviços na nuvem, espalhados por várias regiões no globo, chamadas de *Availability Zones,* que são centros de processamento distintos e isolados uns dos outros. Alguns do serviços incluem: Infra-estrutura como um Serviço, Plataforma como um Serviço e Software como um Serviço. É fornecido um portal para controlo e visualização dos recursos alocados assim como uma previsão dos custos mensais. Possui um período grátis de demonstração com duração de 12 meses, mas com limites na utilização dos seus serviços.

4.4.1.2. *Microsoft® Azure™*

A *Microsoft®* é um dos gigantes mundiais na área das tecnologias, começou no desenvolvimento de sistemas operativos e a partir daí estendeu o seu leque para as mais variadas áreas, desde as consolas de jogos (*Xbox®*), a *browsers* (*Internet Explorer®*), a *tablets* (*Microsoft® Surface™*), entre outros. O *Microsoft® Azure™* é a resposta da *Microsoft®* ao emergente mercado do *cloud computing*.

Alguns dos serviços fornecidos pela nuvem da *Microsoft®* abrangem: Infra-estrutura como um Serviço, Plataforma como um Serviço e Software como um Serviço. A plataforma também fornece integração com o *Microsoft® Visual Studio®*, bem como com o *Eclipse™*. Semelhante à plataforma anterior, os centros de processamento estão divididos e isolados em várias regiões. A *Microsoft®* providencia um período de demonstração gratuito durante

30 dias, no qual o cliente possui $200 para gastar em qualquer combinação de serviços. Também existe um portal onde o cliente pode visualizar e controlar os gastos assim como obter uma previsão, para além disso possui uma interface simples de usar onde se consegue controlar os recursos utilizados e alocar novos ou desalocar.

4.4.1.3 *Google® Cloud Platform*

A *Google*® nasceu da criação de um motor de busca, desde essa altura cresceu e tornou-se num dos gigantes da indústria, contribuindo para isso em grande parte as parcerias feitas e o desenvolvimento de novos produtos e serviços, nomeadamente o sistema operativo para *smartphones Android*™. A *Google*® entrou mais tarde na corrida aos serviços na nuvem, em comparação com os outros dois fornecedores.

Alguns dos serviços que a *Google*® *Cloud Platform* presta são: Infra-estrutura como um Serviço, Plataforma como um Serviço e *Software* como um Serviço. Também possui um portal onde o cliente tem a possibilidade de visualizar os gastos efectuados, bem como as previsões dos mesmos, à semelhança do portal descrito anteriormente na outra plataforma de computação na nuvem. Mais uma vez também possui os centros de processamento espalhados e isolados por regiões. Existe a possibilidade de um período grátis de demonstração com a duração de 60 dias, no qual são creditados $300 para serem gastos nos serviços pretendidos.

4.4.1.4. Comparação

Neste subcapítulo será feita uma análise comparativa entre os fornecedores mencionados e descritos anteriormente, a qual culminará na escolha do fornecedor a ser utilizado, sendo para isso necessário cumprir os vários requisitos existentes.

Seguidamente verifica-se o cumprimento dos vários requisitos exigidos às plataformas na nuvem, evidenciado na tabela seguinte para melhor visualização do cumprimento dos mesmos com uma análise descritiva dessa mesma tabela posteriormente.

	Amazon® Web Services	Google® Cloud Platform	Microsoft® Azure™
Possui *IaaS*	✓	✓	✓
Possui *PaaS*	✓	✓	✓
Suporta *ASP.NET*	✓	✗	✓
NoSQL e *BLOB*	✓	✓	✓
CDN e *Cache*	✓	✓	✓
Escalabilidade/Elasticidade	✓	✓	✓
Load Balacing	✓	✓	✓
Push Notification	✓	✓	✓
Service Bus	✓	✓	✓

Olhando para a tabela constata-se que apenas o *Google® Cloud Platform* não cumpre os requisitos totalmente, mais especificamente o de suportar *ASP.NET*, por essa razão ficou automaticamente excluído, passando assim a escolha a recair sobre uma das outras duas plataformas passiveis ainda de escolha. Qualquer uma das outras duas plataformas cumpre todos os requisitos, assim sendo foi necessário avaliar com base em outros critérios, para além dos requisitos, qual das duas seria a plataforma eleita.

A equipa ponderou e debateu sobre o assunto e acabou por definir novos critérios para o desempate na escolha entre as duas plataformas, sendo esses critérios os seguintes:

- Facilidade de utilização
- Experiência/conhecimento da equipa
- Custo

Com base nestes novos critérios, procedeu-se a uma nova análise das duas plataformas ainda candidatas, que se pode visualizar na seguinte tabela. Nesta análise considera-se que o valores variam segundo uma escala quantitativa de: Muito Mau(á), Mau(á), Razoável, Bom(a) e Muito Bom(a).

	Amazon® Web Services	Microsoft® Azure™
Facilidade de utilização	Razoável	Boa
Experiência/conhecimento da equipa	Má/Mau	Boa/Bom
Custo	Muito Bom	Muito Bom

Foi novamente efectuada uma análise pela equipa de desenvolvimento com base nestes novos critérios, a qual atribuiu valores com base na escala anteriormente descrita.

A equipa decidiu atribuir Razoável na facilidade de utilização ao *Amazon® Web Services* e Boa ao *Microsoft® Azure™* devido em parte a já terem alguma experiência na utilização do portal do *Microsoft® Azure™*, mas também porque o portal da *Amazon® Web Services* é um pouco complexo no início. No entanto o que fez verdadeiramente pender a balança para um dos lados foi a complexidade de utilização do PassS da *Amazon® Web Services* em contraste com a facilidade de uso do PaaS do *Microsoft® Azure™*.

Em relação à experiência/conhecimento da equipa, a plataforma da *Microsoft®* ficou claramente à frente devido à equipa já ter adquirido uma vasta experiência na sua utilização em projectos passados, tendo utilizado no passado um vasto leque dos serviços disponíveis por essa plataforma de computação na nuvem e por isso a curva de aprendizagem na plataforma da *Microsoft®* seria sempre inferior em relação à da plataforma da *Amazon®*, pois tudo teria de ser aprendido com base a recursos *online* (vídeos, blogues, fóruns ou plataformas de *e-learning*) ou a livros sobre a matéria.

Na última categoria, Custo, a equipa atribuio a ambos a mesma nota, Muito Bom, pois os preços são muito semelhantes em ambas as plataformas. Esse facto deve-se muito provavelmente não a uma coincidência, mas sim fruto da grande competitividade entre ambas as plataformas pela liderança do mercado. Apesar disso existem algumas pequenas diferenças que valem apena ser referenciadas, das quais se destaca na plataforma da *Amazon®* a componente muito forte na customização dos custos de utilização, podendo esse factor ser uma vantagem para empresas com conhecimentos fortes na área, como uma possível desvantagem para empresas que não possuam esse mesmo tipo de conhecimento, em contraste a plataforma da *Microsoft®* possibilita grandes descontos a empresas que necessitem de elevados consumos de recursos na sua plataforma de computação na nuvem.

Sendo assim, devido aos argumentos evidenciados anteriormente, a escolha recaiu sobre a plataforma *Microsoft® Azure™*, em deterimento da *Amazon® Web Services*.

4.4.2. Armazenamento de informação

Existem várias formas de se guardar informação, sendo as bases de dados regularmente as mais empregadas para esse fim, principalmente quando se necessita de um acesso simples e rápido à mesma. As bases de dados existem em várias formas e para vários feitios, sendo as relacionais as mais comuns e uma das mais antigas, fazendo uso da linguagem *SQL,* enquanto as denominadas *NoSQL*, bem mais recentes, recorrem a vários mecanismos para leitura e escrita dos dados, mecanismos esses que elas próprias fornecem, variando conforme a implementação do fabricante das mesmas. Para além destes dois existem mais tipos, no entanto estes são os mais utilizados e por isso os mais comuns. Na plataforma SoFly pretende-se utilizar ambos os tipos de armazenamento, ou seja, *SQL* e *NoSQL*.

Existem várias fabricantes de bases de dados *SQL* no entanto os mais comuns, ou mais conhecidos, são a *Oracle*® e a *Microsoft*®. A *Oracle*® através das bases de dados *Oracle™* e *MySQL™*, a *Microsoft*® através do *SQL Server™*, *Access*® e do mais recente *Microsoft® Azure™ SQL Database*.

Em relação ao *NoSQL* os mais conhecidos, ou usados, são o *MongoDB™* [16] e *Cassandra™* [17] sendo ambos *open-source*. A *Cassandra™* pertence à *Apache Software Foundation™* e o *MongoDB™* pertence à *MongoDB, Inc*®. A plataforma visa suportar qualquer tipo de base de dados *NoSQL* que o cliente pretenda utilizar, no entanto o *Microsoft® Azure™* suporta um nativamente que é o *Microsoft® Azure™* Table Storage.

O *MongoDB™* é uma base de dados não relacional tendo por base objectos. Cada entrada é considerada um objecto, que é uma estrutura de dados composta por campos e pares de valores, similar aos objectos JSON. Cada um destes campos pode incluir outros objectos, *arrays* e *arrays* de objectos. Os principais benefícios de ter como base objectos são: correspondem a tipos de dados nativos em muitas linguagens de programação, objectos embebidos e *arrays* reduzem a necessidade de *queries* com *joins* dispendiosos e um esquema dinâmico suporta polimorfismo fluente.

A *Cassandra™* também é uma base de dados não relacional, no entanto é um cruzamento entre dois tipos distintos: pares chave-valor (*key-value*) e orientada a colunas. O que significa que a cada chave corresponde um valor (objecto) e cada chave tem valores que

são colunas. As colunas são agrupadas em *sets* denominadas de *column families*. Estes *sets* podem assim ser considerados de tabelas.

O *Microsoft® Azure™ Table Storage* visa permitir armazenar grandes quantidades de dados estruturados. É uma base de dados não relacional, como as anteriores, e é ideal para:

- Guardar dados não estruturados
- Armazenar *datasets* que não necessitem de *joins* complexos, chaves estrangeiras ou *stored procedures* e que permitam ser desnormalizados para acessos rápidos
- Criar rapidamente *queries* para aceder aos dados através da indexação de clusters e aceder aos dados utilizando protocolos ODATA [11] e *queries* LINQ [11] com bibliotecas .NET

Seguidamente é apresentado um diagrama que permite visualizar facilmente o paradigma interno da *Table Storage*.

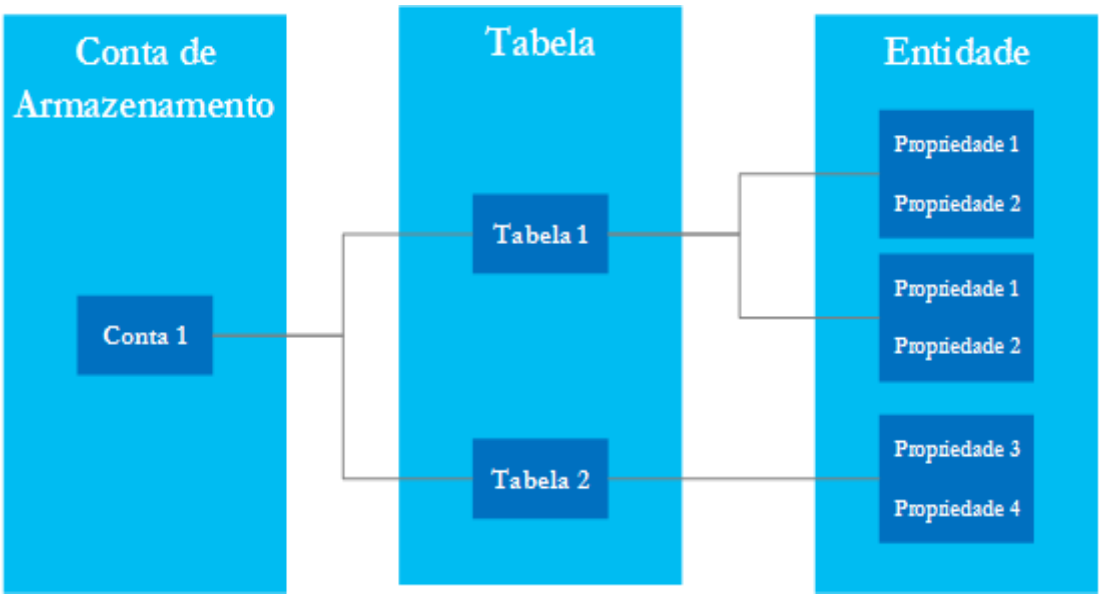


Figura 6 - Diagrama interno da Microsoft® Azure™ Table Storage

No diagrama anterior a Conta de Armazenamento (*Storage Account*) é por onde todos os acessos são conduzidos, podendo existir várias. A Tabela (*Table*) é uma colecção de entidades, não existindo um *schema* forçado sobre as mesmas, o que na práctica significa que as entidades não necessitam de possuir as mesmas propriedades, sendo que uma propriedade é um par valor-chave. O número de tabelas está apenas dependente do limite de capacidade contratado. Uma Entidade (*Entity*) é um conjunto de propriedades, semelhante a uma *row* de uma base de dados relacional, contudo o limite máximo por entidade é de 1 Megabyte. Uma

propriedade, como referido anteriormente, é um par valor-chave e cada entidade pode possuir até 252 propriedades. Cada entidade possui três propriedades de sistema denominadas: *partition key*, *row key* e *timestamp*, que são obrigatórias e por isso comuns a todas as entidades. Entidades com a mesma *partition key* possuem *queries* mais rápidas e são manipuladas em operações atómicas.

4.4.3. Intermediário de Mensagens

Um intermediário de mensagens [11] é um padrão arquitectural [1][2][3] que visa mediar as comunicações entre aplicações através de mecanismos distribuídos de envio de mensagens. Estes mecanismos tendem a ser bidireccionais, mas não é obrigatório. O uso destes intermediários oferece uma plataforma escalável, melhora a acessibilidade e a confiança dos dados e concentra todo o fluxo de mensagens num único sítio. Como os intermediários de mensagens utilizam sistemas de envio de mensagens para mover a informação entre sistemas, muitos suportam meios de transporte assíncronos assim como variadas técnicas a utilizar nestes mesmos sistemas, entre os quais: publicador/subscritor, filas de mensagens, garantias de entrega "pelo menos uma vez" e/ou "no máximo uma vez". Na figura seguinte pode-se ver um diagrama que exemplifica o uso de um intermediário de mensagens na comunicação entre várias aplicações.

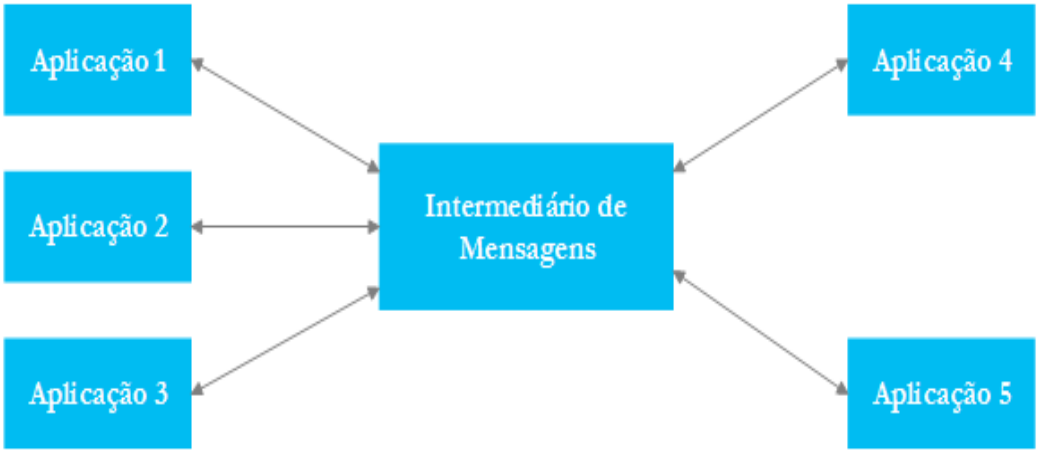


Figura 7 - Intermediário de Mensagens a mediar o envio de mensagens entre várias aplicações

As aplicações devem-se registar previamente com o intermediário de mensagens para assim poderem receber e/ou enviar mensagens. O intermediário pode providenciar os seus próprios métodos de registo ou fazer uso de serviços externos para o efeito.

As obrigações [11] de um intermediário consistem em:

1. Receber a mensagem
2. Determinar o receptor da mensagem e efectuar o *routing* para o mesmo
3. Controlar e manipular diferenças ao nível do interface nas aplicações
4. Enviar a mensagem para o destinatário

Existem vários intermediários de mensagens no mercado e de vários fornecedores e, por vezes, um único fornecedor possui vários intermediários. Essa variedade mesmo dentro do leque de produtos de um só fornecedor provêm da especialização de cada um desses intermediários de mensagens.

Um dos intermediários de mensagens mais comum e mais utilizado é o *Service Bus*. Este intermediário é um modelo de *software* arquitectural desenhado e utilizado primariamente como forma de comunicação bidireccional entre as aplicações. Ambas as três plataformas de computação na nuvem previamente discutidas, analisadas e descritas anteriormente fornecem precisamente este tipo de serviço, em seguida irá ser descrito o *Microsoft® Azure™ Service Bus* que pertence precisamente à plataforma escolhida de computação na nuvem.

O *Microsoft® Azure™ Service Bus* providencia uma infra-estrutura segura e amplamente disponível na nuvem para todo o tipo de comunicações, distribuição de eventos em larga escala e publicação de serviços. Também fornece opções de conectividade para serviços, inclusive serviços *REST*, assim como garante a capacidade de envio de mensagens através dos padrões *relayed* [2][3][11] e *brokered* [2][3][11]. No envio de mensagens através do padrão *relayed* o serviço suporta o envio directo de mensagens num sentido (*one-way*), pedido/resposta e *peer-to-peer*. O *brokered* permite componentes duráveis e assíncronos como as *Queues* [11] e os Tópicos e Subscrições [11], com técnicas de publicador/subscritor e desacoplamento temporal, ou seja, quem envia ou recebe não precisa de estar *online* ao mesmo tempo, pois a infra-estrutura de mensagens guarda as mensagens de uma forma fidedigna até ao receptor estar *online* ou pronto para as receber.

Uma *Queue* é um *buffer* sequencial e persistente no qual um ou mais produtores enviam mensagens, que são inseridas na cauda, e um ou mais consumidores competem por receber as mensagens que são retiradas pela cabeça. Possui também um ponteiro, partilhado por todos os consumidores, que aponta para a mensagem que deve ser consumida. As *Service Bus*

Queues [11] providenciam várias metodologias que possibilitam alterar esse comportamento, nomeadamente o tempo de visibilidade das mensagens na *queue*, o deferimento de processamento das mensagens e a possibilidade das mensagens retornarem à *queue*.

Tópicos são um *buffer* sequencial e persistente no qual um ou mais produtores inserem mensagens na cauda desse mesmo buffer, à semelhança da *queue*, no entanto tem múltiplas cabeças, as quais são denominadas de subscrições e cada uma recebe uma cópia da mensagem. Cada subscrição tem o seu próprio ponteiro que é movido cada vez que uma mensagem é consumida. As subscrições podem ser filtradas para que apenas um subconjunto de mensagens esteja disponível para as mesmas.

4.4.4. *Web Application Frameworks*

As *Web application frameworks* são *frameworks* criadas e desenhadas para permitir o desenvolvimento de *websites* dinâmicos, aplicações, serviços e recursos *web*. O objectivo da mesma é retirar o peso das actividades e funções mais comuns no desenvolvimento *web*, providenciando normalmente bibliotecas e/ou *templates* e incentivando muitas das vezes a reutilização do código. Estas *frameworks* baseiam-se maioritariamente em padrões arquitecturais e um dos mais comuns no desenvolvimento web é o Modelo-Vista-Controlador.

O padrão Modelo-Vista-Controlador visa separar a parte lógica do interface do utilizador, sendo que para isso divide as aplicações web em três camadas lógicas: modelo, vista e controlador. Na figura seguinte está representado o diagrama desta arquitectura.

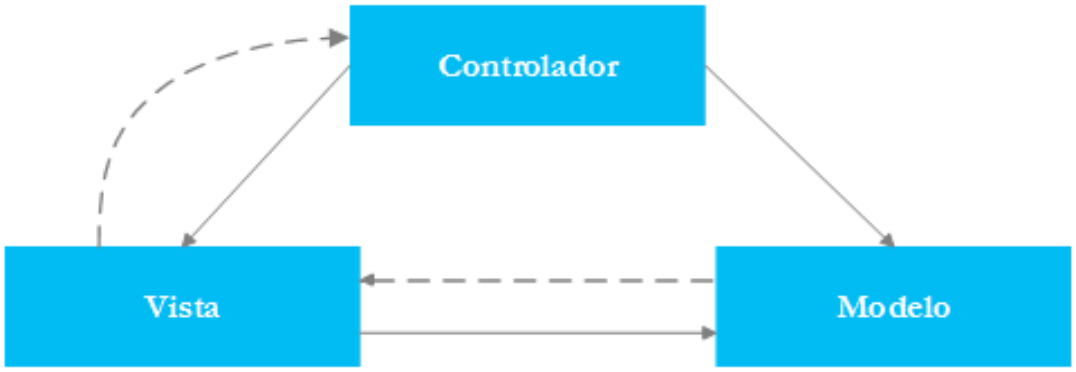


Figura 8 - Diagrama da arquitectura MVC

No diagrama anterior podemos ver três módulos: Modelo [2][3], Vista [2][3] e Controlador [2][3]. O módulo Controlador interage com o módulo Modelo actualizando o seu estado, também interage com o módulo Vista alterando a apresentação do Modelo que

é representado na Vista. O Modelo notifica as Vistas e os Controladores associados quando existe uma alteração no seu estado, denominada de arquitectura *push-based*, no entanto em algumas variações o módulo Modelo é passivo obrigando por isso aos outros módulos a fazerem pedidos para saberem quando este mudou, designada de arquitectura *pull-based*. Por fim, o módulo Vista apenas permite ao utilizador visualizar a representação do Modelo no estado em que se encontra.

Existem muitas *frameworks* para o desenvolvimento web, no entanto dado um dos requisitos do projecto ser o desenvolvimento da plataforma utilizando a *Framework ASP.NET,* foi escolhida para o desenvolvimento do *front-end* a *Framework ASP.NET MVC 5* e para o desenvolvimento da API a *Framework ASP.NET Web API 2*.

A *Framework ASP.NET MVC 5* vai na quinta versão, estando já disponível a sexta versão, no entanto está apenas disponível para testes como versão de acesso antecipado e não de produção. Utiliza o padrão Modelo-Vista-Controlador, descrito anteriormente, assim como outros padrões [2][3]. Neste projecto foi utilizada a linguagem de programação *C#*, a qual é uma das linguagens compatíveis com a dita *framework*, no entanto para além dessa linguagem também foi utilizado *JavaScript* para o desenvolvimento do *front-end*, bem como a biblioteca *jQuery*™ [18]. Esta *Framework* utiliza um motor de renderização (*view engine*) denominado *Razor* [11]. Este permite o uso de *HTML*, *CSS* e *JavaScript*, mas também a integração de código de servidor nas vistas permitindo assim a geração dinâmica de código nas mesmas.

A *Framework ASP.NET Web API 2* visa permitir criar web APIs de maneira fácil e rápida, bem como possibilita criar aplicações que respeitem totalmente os princípios da arquitectura REST, fazendo uso dos serviços HTTP para esse efeito. A *framework* aceita pedidos quer em JSON quer em XML por omissão e as respostas tanto podem ser em JSON como em XML, a mesma viabiliza expor de forma segura os serviços e recursos à internet.

4.5. Riscos

Num projecto complexo e ambicioso como este, para o desenvolvimento de uma plataforma de análise de redes sociais, existem vários riscos inerentes. Contudo esses riscos foram drasticamente reduzidos graças à criação de protótipos funcionais, protótipos esses criados anteriormente ao ínicio deste mesmo estágio. Sendo assim possível minimizar bastante esses ditos riscos e por vezes eliminá-los, tanto ao nível do desenvolvimento como da arquitectura e integração dos seus vários módulos.

Apesar da criação dos protótipos ter tido um grande impacto na redução dos riscos não os eliminou totalmente, no entanto, devido à criação dos mesmos e sua posterior utilização num caso de teste com um cliente num cenário real, foi possível identificar e resolver alguns problemas, tanto ao nível do desenvolvimento como da arquitectura do sistema, bem como validar algumas das opções tomadas e quando necessário fazer reajustes.

Contudo apesar de todas estas precauções e opções tomadas, os riscos nunca são totalmente eliminados, principalmente quando não se está apenas dependente de si próprio, mas também de terceiros, nomeadamente de quem fornece os dados crus, neste caso específico as redes sociais e consequentemente as suas próprias APIs. A esta dependência pouco ou nada se poderia fazer para diminuir os riscos, pois a qualquer momento uma simples alteração na API numa das redes sociais poderia significar grandes e profundas alterações no desenvolvimento da plataforma. Outro risco inerente, mais uma vez da necessidade de utilização das redes sociais, prende-se com o facto de se ter de respeitar sempre os termos de utilização das respectivas redes sociais para o uso das suas APIs, bem como dos dados que estas providenciam, pois qualquer alteração às mesmas poderá implicar a necessidade de adição ou actualização de funcionalidades durante o processo de desenvolvimento ou após o término do mesmo. Para mitigar os riscos derivados da utilização de APIs e dados externos, neste caso das redes sociais, são conduzidos testes para validar os pedidos feitos a essas mesmas APIs, possibilitando assim saber quando as mesmas são alteradas. Infelizmente não possibilita saber antecipadamente quando as mesmas irão ocorrer. A única maneira de se mitigar completamente este risco é tornar-nos parceiros dessas mesmas redes sociais, possibilitando assim saber antecipadamente as alterações, bem como quando irão ocorrer.

Por estas razões, os riscos inerentes a um projecto complexo e ambicioso como o delineado para este mesmo estágio, foram mitigados sempre que tal foi possível.

4.6. Testes

Como em todos os projectos desenvolvidos na empresa, não sendo este a excepção à regra, são conduzidos testes durante todo o desenvolvimento do produto de forma a garantir o correcto funcionamento e a integridade dos dados.

São sempre feitos testes unitários ao código desenvolvido, por regra são sempre efectuados no final do desenvolvimento de cada módulo para sua validação ou quando um módulo era alterado. Na construção destes testes é utilizada uma *framework*, de seu nome

NUnit [19], a qual corre em todas as linguagens da *Framework ASP.NET*. Uma das razões da sua escolha, para além da curva de aprendizagem no seu uso, foi a facilidade de integração com a IDE utilizada no desenvolvimento dos módulos da plataforma, o *Visual Studio*®. Estes testes irão ser feitos aquando do início do desenvolvimento dos módulos.

Para os testes funcionais, de integração e de segurança estes inicialmente foram feitos manualmente, sendo para isso utilizadas ferramentas no seu auxílio, nomeadamente o *Fiddler*™ ou o *Wireshark*™, dependendo sempre da singularidade do teste a correr.

5. Conclusões e Comentários Finais

Irá ser feita em seguida uma análise ao estágio decorrido, sendo a mesma dividida em duas fases.

Na primeira fase, correspondente ao primeiro semestre, foi despendido bastante tempo em investigação e análise dos requisitos técnicos e funcionais. Esta investigação centrou-se em provas de conceito que visavam validar as decisões tomadas, com base nos requisitos que lhes deram origem. Estas mesmas provas de conceito, para além das validações, permitiram ganhar conhecimentos e competências novas que irão ser uma mais-valia tanto na segunda fase do projecto como no futuro profissional do estagiário.

A segunda fase, correspondente ao segundo semestre, foi centrado no desenvolvimento dos módulos com a restante equipa de desenvolvimento, bem como na validação do correcto funcionamento dos mesmos graças aos testes efectuados durante o seu desenvolvimento. Para o bom funcionamento desta segunda parte do estágio, os conhecimentos adquiridos na primeira fase do estágio contribuíram de uma forma decisiva.

Os objectivos propostos no início do estágio eram analisar, propor e desenvolver módulos para uma plataforma de análise de redes sociais. Pretendia-se no entanto que o desenvolvimento fosse efectuado por toda a equipa, estagiário incluído, e não apenas pelo estagiário. Contudo cabia ao estagiário desenvolver as técnicas e os mecanismos necessários para o desenvolvimento da dita plataforma. Todos os objectivos enunciados anteriormente foram escrupulosamente cumpridos, estanto a plataforma neste momento funcional e *online*, podendo ser utilizada pelos clientes.

Nesta recta final, estando ambas as fases do estágio já concluídas, posso afirmar que o mesmo contribuiu de uma forma enriquecedora para o meu futuro profissional através do aumento das minhas capacidades técnicas, bem como no trabalho em equipa.

No futuro pretende-se continuar a fazer crescer a plataforma, visando torná-la a plataforma de referência na área. Para isso pretende-se adicionar suporte a mais redes sociais, existentes ou emergentes, bem como adicionar novas funcionalidades que os clientes possam necessitar. Pretende-se também melhorar a análise de toda a informação reunida nas redes sociais, adicionando previsões com base nos sentimentos (positivos ou negativos), bem como tornar-nos parceiros dessas mesmas redes sociais, sempre que o mesmo seja possível. O Facebook, a título de exemplo, providência uma API apenas aos seus parceiros o que em

termos prácticos dá uma enorme vantagem aos mesmos, pois essa API fornece acesso a dados que de outra maneira não estão disponíveis. Foi também referenciado anteriormente que se pretende adicionar funcionalidades novas, uma das funcionalidades será a adição de *Insights* à palataforma, que permitirá visualizar o impacto dos recursos existentes nos programas de televisão (apresentadores, actores ou outros), através da medição da sua popularidade, bem como o seu impacto nas redes sociais. Outra funcionalidade que poderá ser adicionada, será a incorporação de mensagens instantâneas na plataforma entre os utilizadores de uma dada subscrição, permitindo assim comunicarem entre eles sem recurso a *software* externo.

6. Referências

6.1. Livros

[1] Microsoft® .NET - Architecting Applications for the Enterprise (2nd Edition) (Developer Reference), Dino Esposito e Andrea Saltarello, Microsoft® Corporation, Setembro 2014

[2] Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries (2nd Edition), Krzysztof Cwalina e Brad Abrams, Addison-Wesley, Novembro 2008

[3] Microsoft® Application Architecture Guide (Patterns & Practices), Microsoft® Patterns & Practices Team, Microsoft® Corporation, Novembro 2009

6.2. Páginas Web

[4] Manifesto for Agile Software Development, http://www.agilemanifesto.org

[5] Agile Development: Lessons Learned from the First Scrum, http://www.scrumalliance.org/resources/35

[6] Spredfast™, https://www.spredfast.com

[7] Netbase™, http://www.netbase.com

[8] Telescope™, http://telescope.tv

[9] Wayin™, http://wayin.com

[10] Microsoft® Azure™, http://azure.microsoft.com

[11] Microsoft® Software Developer Network (MSDN™), http://msdn.microsoft.com

[12] Fiddler™, http://www.telerik.com/fiddler

[13] Wireshark™, https://www.wireshark.org

[14] Amazon® Web Services, http://aws.amazon.com

[15] Google® Cloud Platform, https://cloud.google.com

[16] MongoDB™, http://www.mongodb.org

[17] Cassandra™, http://cassandra.apache.org

[18] jQuery™, http://jquery.com

[19] NUnit™, http://www.nunit.org

7. Anexos

[1] SoFly – Overview, Outubro 2013

[2] Case Study – SIC Factor X, Outubro 2013

[3] Camada de Dados – Diagrama de Classes

[4] API de Negócio – Diagrama de Classes

[5] Web Front-End – Diagram de Classes

[6] Mensagens (Messaging) – Diagrama de Classes

[7] Notificações (Notifications) – Diagrama de Classes

Anexo 1

SoFly - Overview

SoFly (HK) Ltd.

Outubro 2013

SOFLY ANALYTICS

Engage with your audience &
Get meaningful insights from social conversations

@SoFlyHQ | contact@sofly.tv | www.analytics.sofly.tv | SoFly (HK) Ltd.

#Major challenges of Social Television

With the rise of Social Media the importance of engaging your audience across these platforms is critical to a successful brand story. When it comes to television, the social aspect of the experience can dictate the success of your production. With all the activity surrounding televised broadcasts over these platforms, producing companies can have access to meaningful insights and crucial feedback. Relationships with your customers can be achieved and maintained over social platforms even though most of the users prefer to remain anonymous. This relationships are as important for your band or company as they are critical in today's digital world for your consumers and effectively drive audience's choices.

Reach your audience and drive conversations

Considering the information flow over Social Media, it's hard to get your audience to notice your message. An even harder job is to generate meaningful conversations that would propagate your brand over these platforms. Personal relationships are the key aspect to brand propagation and this can be achieved by actively interacting with your audience on a daily basis and constantly improving and adjusting your storytelling according to your audience taste.

Amplify your message and retain followers

By engaging with your audience over Social Media platforms you amplify their engagement directly with your production or broadcast, as well as, with the brands that are being advertised. This engagement alongside with the direct communication and personal relationship built over time are crucial factors to audience retention and the relationship between your social impact and your direct business results.

Create customer experience across all platforms

Currently over 50% of your audience is actively searching the World Wide Web for related content, which means that this behaviour makes it not only important but imperative to your business to track, measure and actively contribute to this process. Your audience is looking for an augmented and meaningful experience across every platform. Accounting for all this activity across various platforms and devices requires a simple and intuitive tool that could measure, analyse and help your team react in real-time to ongoing trends and events.

#SoFly Analytics

SoFly Analytics helps your team identify what topics are predominant in social conversations around your content and act upon this insights in order to reach new audience and enhance your engagement with existing viewers. Our platform empowers your team to collaborate and place their focus on aspects of social activity that require attention. The real-time analyses capability will provide an essential tool for adjusting your storytelling even during live broadcasts. The resulting improvement of your audience's viewing experience over diverse platforms and devices will benefit your general business results as well as provide you with new communication channels and advertisement opportunities.

Engage socially with a single solution

SoFly Analytics allows you to follow, measure, engage and analyse all of your social activity within one simple and intuitive tool letting your team focus their effort on what matters – your audience.

Listen and Publish

By tracking social conversations you'll stay up-to-date with current trends and topics that are being talked about your brand. Engaging with your audience is as important as understanding what drives them and for that matter SoFly Analytics provides you with all the required tools to schedule and publish your content, as well as understand when this content should be published in order to amplify your reach.

Collaborate and Moderate

Marketing is not a one man job and SoFly Analytics is ready to support your team by providing them a tool to communicate as well as collaborate and moderate the content that is being published. Scheduling and planning of upcoming broadcasts will save your marketing team precious time that can be used to engage with your audience during the broadcast.

Integrate social experience within your digital content

SoFly Analytics has a wide range of web plugins that can be used on your website to integrate social experience and provide your audience with augmented viewing experience across every platform and device.

Integrate social experience into your live broadcasts

SoFly Analytics allows you to exports moderated and filtered social feeds and integrate social experience into your live broadcasts providing your audience with real-time social experience while keeping the content profanity free and moderated.

Measure and analyse your impact

Knowing who your key influencers are and engaging with them will greatly contribute to overall propagation of your content over Social Media channels. SoFly Analytics provides you with easy and intuitive tools to achieve that in real-time. Alongside, we provide you with tools to analyse historical data in detail allowing you to drill down to specific metrics and post performances. Gained insights will greatly improve your understanding of your audience's taste and as a result allow you to adjust your storytelling and to stand out from the crowd.

Anexo 2

Case Study - SIC FactorX

SoFly (HK) Ltd

Outubro 2013

SOFLY ANALYTICS

@Case Study

FACTOR X

Factor X engages social audience during live broadcasts and
Becomes the most trending show on Twitter in Portugal in 2013.

Case Study was developed in partnership with SIC – Sociedade Independente de Comunicação.

#Overview

SIC - Factor X

SOFLY ANALYTICS

Organization	SIC
Project	Factor X
Audience	TV viewers
Country	Portugal
Industry	Media
Sector	Live Broadcast
Products	SoFly Analytics
	SoFly Social Plugins

#KeyObjectives

Increase social activity around the real-time TV show in order to attract new fans, improve engagement and increase viewership.

#Strategy

Incorporate elements of real-time interaction of the audience with the show by using **SoFly Social Polls**; use of **SoFly Analytics** for Twitter stream analyses and data visualization.

#KeyResults

Factor X became the most trending Portuguese TV show on Twitter in 2013. Twitter activity around the show jumped 665% with the general increase in Twitter audience of 432%. Overall, potential impressions on Twitter had reached 5.3 million prints per broadcast (equivalent to 49% of the population of Portugal).

#Background

Gain social traction in emerging Twitter community

Social Television and the technological evolution seen around it during the past few years has changed and moulded in a whole new way the TV-viewing experience. Factor X, which aired in the fall of 2013, is a live TV Show focused on finding emerging talented singers around the country. Factor X production and marketing teams wanted to explore in-depth the social media opportunities in the emerging Twitter community in Portugal. As a result they wanted a platform capable to analyse, in real-time, Twitter interaction of the audience and empower communication and social integration during the show broadcast.

Involvement of SoFly Analytics and SoFly Social Plugins allowed to establish an easy and comprehensive communication channel between Twitter audience of the show and the production teams. Involvement of an interactive real-time Twitter content, active audience polls and social data visualization in the studio allowed the show to take the first place between most trended TV shows in Portugal on Twitter in 2013.

#Objectives

Establish a two-way communication channel with the audience

In order to capture the attention of the audience and to create an emerging experience, SoFly team had three goals in mind:

- Attract new fans and increase Twitter audience of the show in order to increase viewership.
- Create a real-time platform that would allow an easy understanding of social conversations.
- Empower social engagement over Twitter by integrating live data in the studio and on-air.

#Strategy

Comprehensive real-time social data delivered with Simplicity

In order to activate and engage the audience, production team encouraged TV-viewers to Tweet with a show-centric hashtag. This strategy was eagerly adopted once the social content was displayed on stage during a live broadcast.

SoFly Analytics provided the marketing team with real-time Twitter activity analyses, Twitter streams, audience activation metrics, most active and influent user lists, counters for contestant and judge mentions as well as respective sentimental analyses.

On-set, SoFly Social Plugins supplied the production team with the leader-boards of most active users on Twitter in order to display their photos on-air. An online poll was made to identify most popular judge and the results were displayed once the poll time was over.

Social integration and its adoption by the audience quickly made Factor X the most trending topic on Twitter in Portugal during the broadcasts.

#Results

The "Social Era" of live Television

During the period of time that social interaction was integrated into Factor X, Twitter activity jumped to a stunning 665% and the active audience has increased 432% once data visualisations were periodically shown on-air.

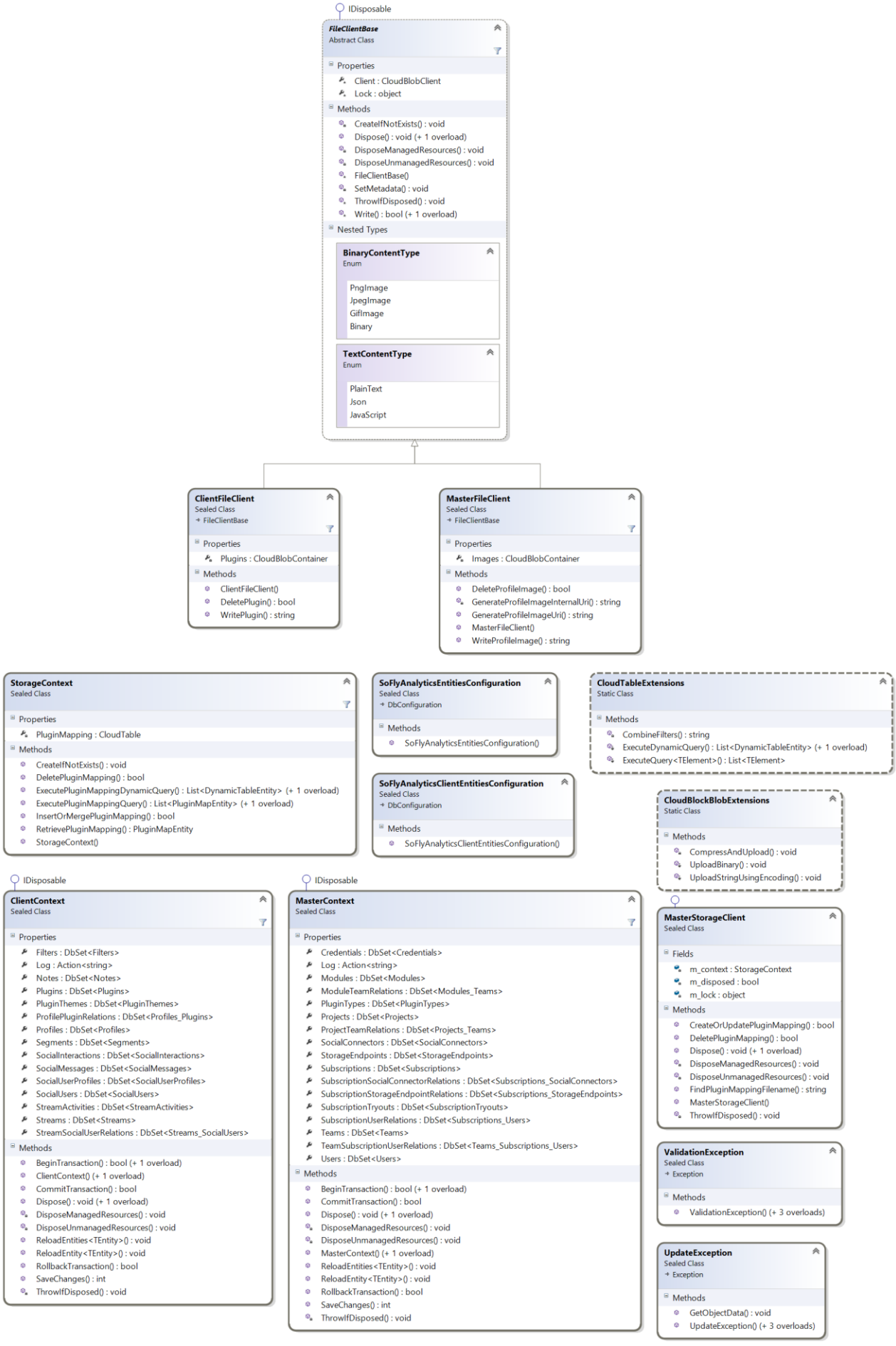
Average amount of tweets per broadcast rounded 37,000 tweets with a major spike of 1191 tweets per minute related to a key moment of a show. Number of potential impressions on Twitter also increased by 274% reaching 5,264,474 potential views (equivalent to 49% of population of Portugal).

The grows of interest and interaction could also be tracked by following the interval of time during the show that the amount of tweets per minute were larger than their average value and that systematically grew by 6.5 minutes per episode. This indicated a substantial and constant increase in time that the active audience remained engaged with the show.

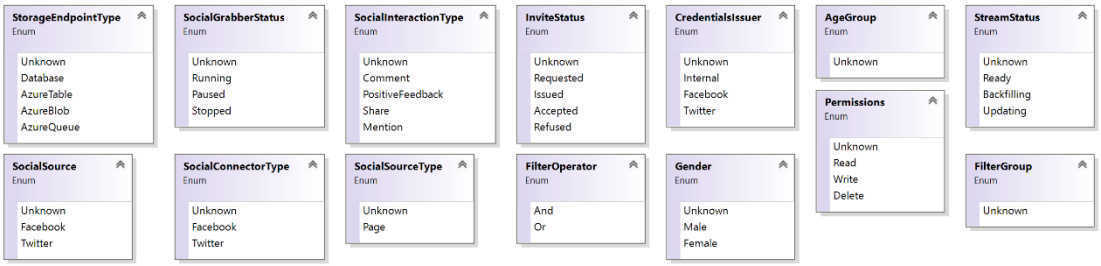
Anexo 3

Camada de Dados - Diagrama de Classes

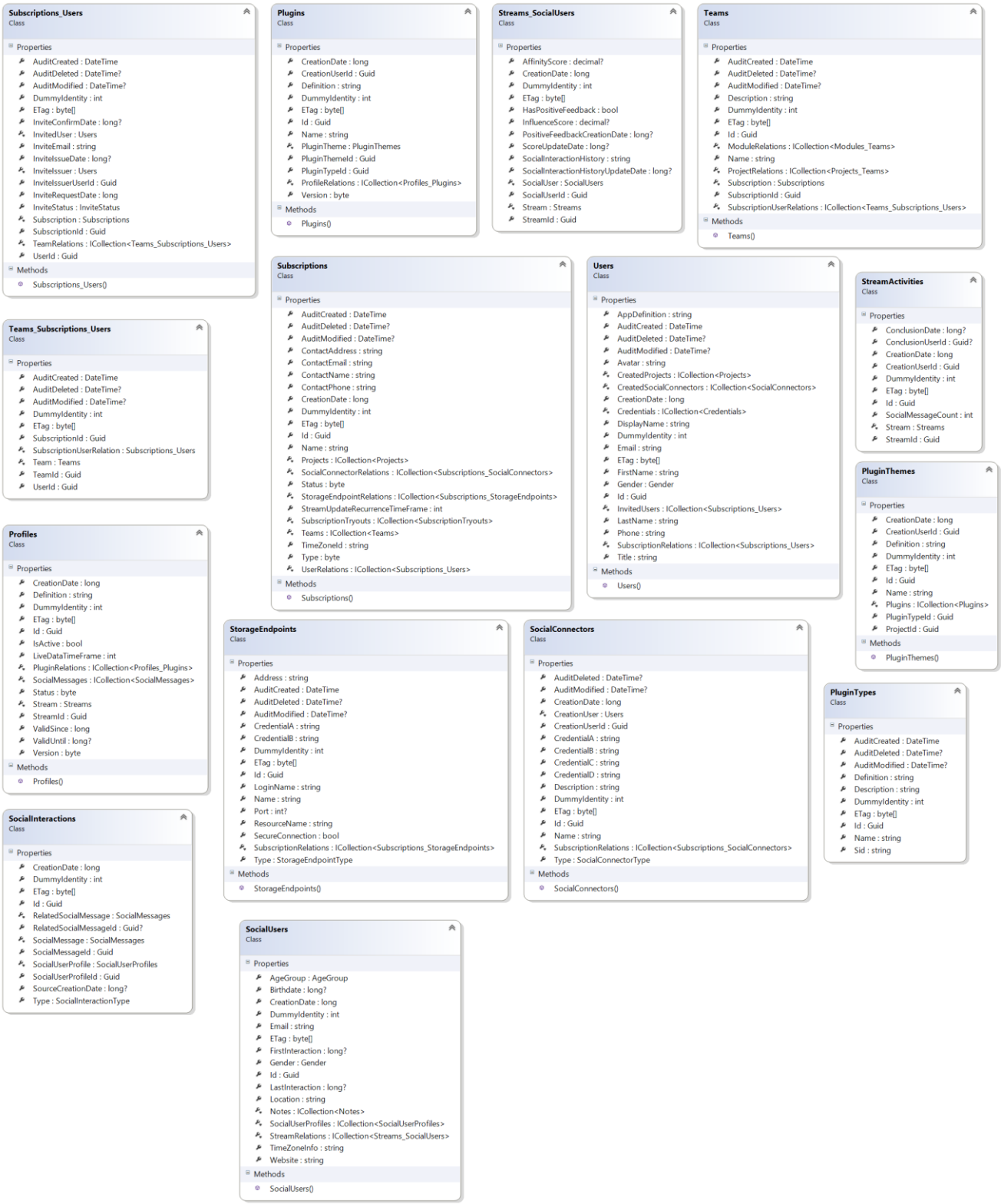
Client & Contexts



Enums



Entities

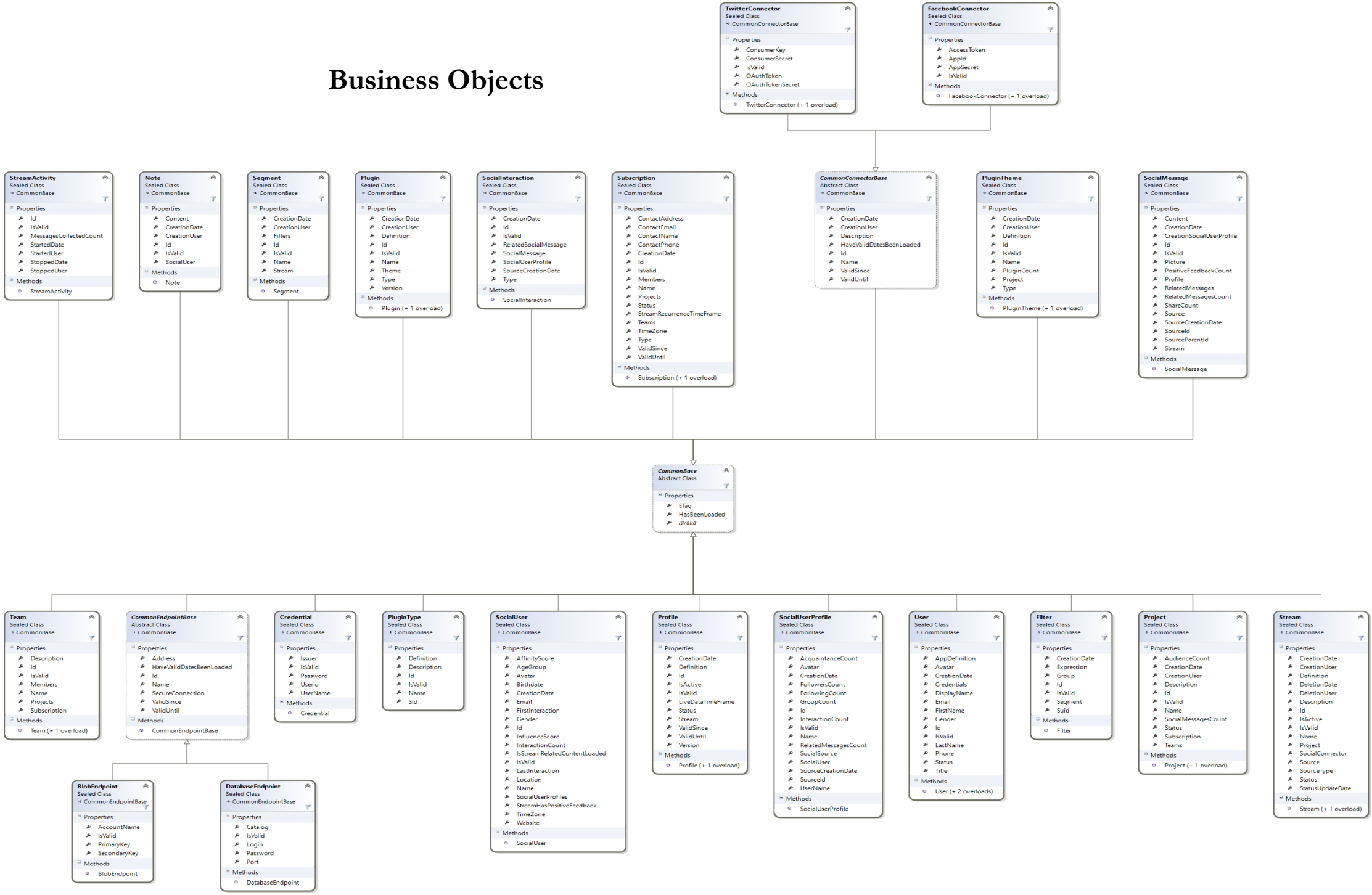



SoFlyAnalyticsEntities
Class
DbContext
Properties
Credentials : DbSet<Credentials>
Modules : DbSet<Modules>
Modules_Teams : DbSet<Modules_Teams>
PluginTypes : DbSet<PluginTypes>
Projects : DbSet<Projects>
Projects_Teams : DbSet<Projects_Teams>
SocialConnectors : DbSet<SocialConnectors>
StorageEndpoints : DbSet<StorageEndpoints>
Subscriptions : DbSet<Subscriptions>
Subscriptions_SocialConnectors : DbSet<Subscriptions_SocialConnectors>
Subscriptions_StorageEndpoints : DbSet<Subscriptions_StorageEndpoints>
Subscriptions_Users : DbSet<Subscriptions_Users>
SubscriptionTryouts : DbSet<SubscriptionTryouts>
Teams : DbSet<Teams>
Teams_Subscriptions_Users : DbSet<Teams_Subscriptions_Users>
Users : DbSet<Users>
Methods
OnModelCreating() : void
SoFlyAnalyticsEntities() (+ 1 overload)
Subscriptions_StorageEndpoints
Class
Properties
AuditCreated : DateTime
AuditDeleted : DateTime?
AuditModified : DateTime?
DummyIdentity : int
ETag : byte[]
Id : Guid
StorageEndpoint : StorageEndpoints
StorageEndpointId : Guid
Subscription : Subscriptions
SubscriptionId : Guid
ValidSince : long
ValidUntil : long?
SoFlyAnalyticsClientEntities
Class
DbContext
Properties
Filters : DbSet<Filters>
Notes : DbSet<Notes>
Plugins : DbSet<Plugins>
PluginThemes : DbSet<PluginThemes>
Profiles : DbSet<Profiles>
Profiles_Plugins : DbSet<Profiles_Plugins>
Segments : DbSet<Segments>
SocialInteractions : DbSet<SocialInteractions>
SocialMessages : DbSet<SocialMessages>
SocialUserProfiles : DbSet<SocialUserProfiles>
SocialUsers : DbSet<SocialUsers>
StreamActivities : DbSet<StreamActivities>
Streams : DbSet<Streams>
Streams_SocialUsers : DbSet<Streams_SocialUsers>
Methods
OnModelCreating() : void
SoFlyAnalyticsClientEntities() (+ 1 overload)
Subscriptions_SocialConnectors
Class
Properties
AuditCreated : DateTime
AuditDeleted : DateTime?
AuditModified : DateTime?
DummyIdentity : int
ETag : byte[]
Id : Guid
SocialConnector : SocialConnectors
SocialConnectorId : Guid
Subscription : Subscriptions
SubscriptionId : Guid
ValidSince : long
ValidUntil : long?
SocialUserProfiles
Class
Properties
AcquaintanceCount : int?
Avatar : string
CreationDate : long
DummyIdentity : int
ETag : byte[]
FollowersCount : int?
FollowingCount : int?
GroupCount : int?
Id : Guid
Name : string
RelatedMessagesCount : int?
SocialInteractions : ICollection<SocialInteractions>
SocialMessages : ICollection<SocialMessages>
SocialUser : SocialUsers
SocialUserId : Guid
Source : SocialSource
SourceCreationDate : long?
SourceId : string
UserName : string
Methods
SocialUserProfiles()
SocialMessages
Class
Properties
Content : string
CreationDate : long
CreationSocialUserProfileId : Guid
DummyIdentity : int
ETag : byte[]
Id : Guid
Picture : string
PositiveFeedbackCount : int?
Profile : Profiles
ProfileId : Guid?
RelatedMessagesCount : int?
ShareCount : int?
SocialInteractions : ICollection<SocialInteractions>
SocialInteractions1 : ICollection<SocialInteractions>
SocialUserProfile : SocialUserProfiles
Source : SocialSource
SourceCreationDate : long?
SourceId : string
SourceParentId : string
Stream : Streams
StreamId : Guid
Methods
SocialMessages()
Segments
Class
Properties
CreationDate : long
CreationUserId : Guid
DummyIdentity : int
ETag : byte[]
Filters : ICollection<Filters>
Id : Guid
Name : string
Stream : Streams
StreamId : Guid
Methods
Segments()
Filters
Class
Properties
CreationDate : long
DummyIdentity : int
ETag : byte[]
Expression : string
Group : FilterGroup
Id : Guid
Segment : Segments
SegmentId : Guid
Suid : string
Credentials
Class
Properties
AuditCreated : DateTime
AuditDeleted : DateTime?
AuditModified : DateTime?
DummyIdentity : int
ETag : byte[]
Issuer : CredentialsIssuer
Password : string
User : Users
UserId : Guid
UserName : string
Streams
Class
Properties
CreationDate : long
CreationUserId : Guid
Definition : string
DeletionDate : long?
DeletionUserId : Guid?
Description : string
DummyIdentity : int
ETag : byte[]
Id : Guid
IsActive : bool
Name : string
Profiles : ICollection<Profiles>
ProjectId : Guid
Segments : ICollection<Segments>
SocialMessages : ICollection<SocialMessages>
SocialUserRelations : ICollection<Streams_SocialUsers>
Source : SocialSource
Status : StreamStatus
StatusUpdateDate : long
StreamActivities : ICollection<StreamActivities>
Subscriptions_SocialConnectorsId : Guid
Type : SocialSourceType
Methods
Streams()
Projects
Class
Properties
AudienceCount : int
AuditDeleted : DateTime?
AuditModified : DateTime?
CreationDate : long
CreationUser : Users
CreationUserId : Guid
Description : string
DummyIdentity : int
ETag : byte[]
Id : Guid
Name : string
SocialMessagesCount : int
Status : SocialGrabberStatus
Subscription : Subscriptions
SubscriptionId : Guid
TeamRelations : ICollection<Projects_Teams>
Methods
Projects()
Modules_Teams
Class
Properties
AuditCreated : DateTime
AuditDeleted : DateTime?
AuditModified : DateTime?
DummyIdentity : int
ETag : byte[]
Module : Modules
ModuleId : Guid
Permission : Permissions
Team : Teams
TeamId : Guid
Projects_Teams
Class
Properties
AuditCreated : DateTime
AuditDeleted : DateTime?
AuditModified : DateTime?
DummyIdentity : int
ETag : byte[]
Project : Projects
ProjectId : Guid
Team : Teams
TeamId : Guid
Modules
Class
Properties
AuditCreated : DateTime
AuditDeleted : DateTime?
AuditModified : DateTime?
Description : string
DummyIdentity : int
ETag : byte[]
Id : Guid
Name : string
TeamRelations : ICollection<Modules_Teams>
Methods
Modules()
PluginMapEntity
Sealed Class
TableEntity
Properties
Filename : string
PluginId : Guid
PluginTypeId : Guid
Methods
PluginMapEntity()
SubscriptionTryouts
Class
Properties
AuditCreated : DateTime
AuditDeleted : DateTime?
AuditModified : DateTime?
DummyIdentity : int
EndDate : long?
ETag : byte[]
Id : Guid
StartDate : long
Subscription : Subscriptions
SubscriptionId : Guid
Type : byte
Profiles_Plugins
Class
Properties
CreationDate : long
DummyIdentity : int
ETag : byte[]
Plugin : Plugins
PluginId : Guid
Profile : Profiles
ProfileId : Guid
Notes
Class
Properties
Content : string
CreationDate : long
CreationUserId : Guid
DummyIdentity : int
ETag : byte[]
Id : Guid
SocialUser : SocialUsers
SocialUserId : Guid

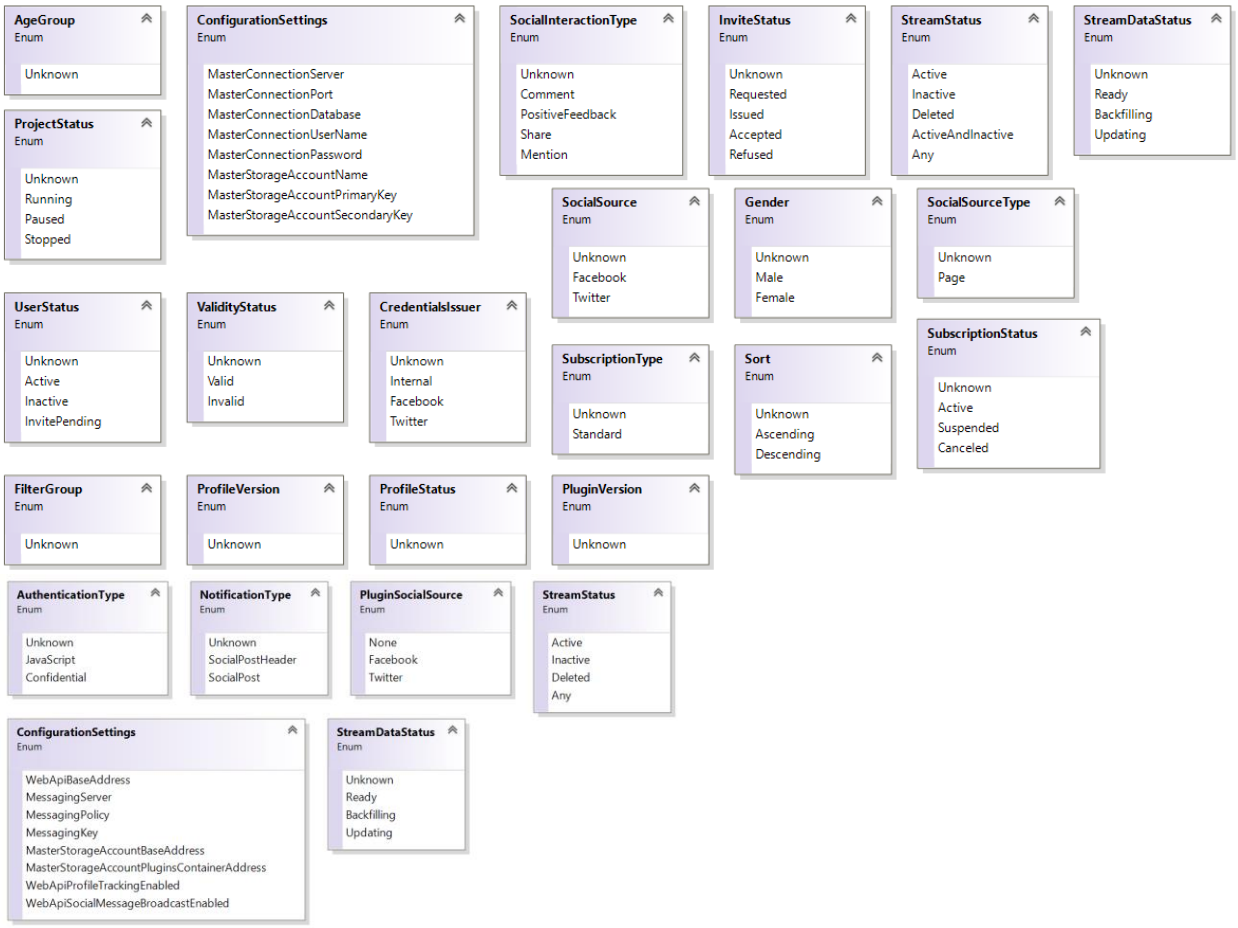
Anexo 4

API de Negócio - Diagrama de Classes

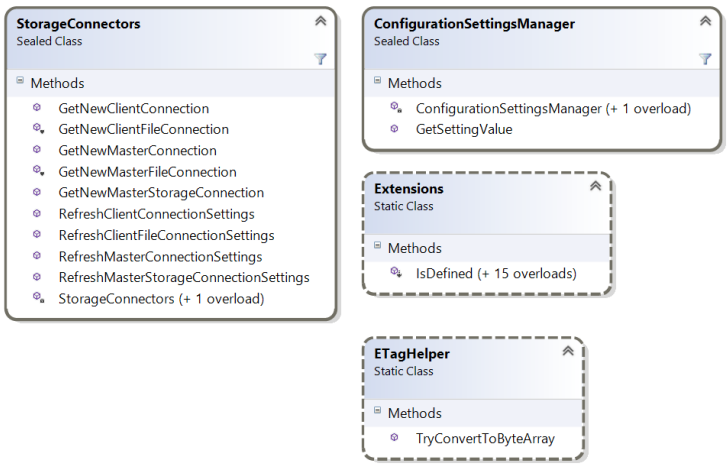
Business Objects



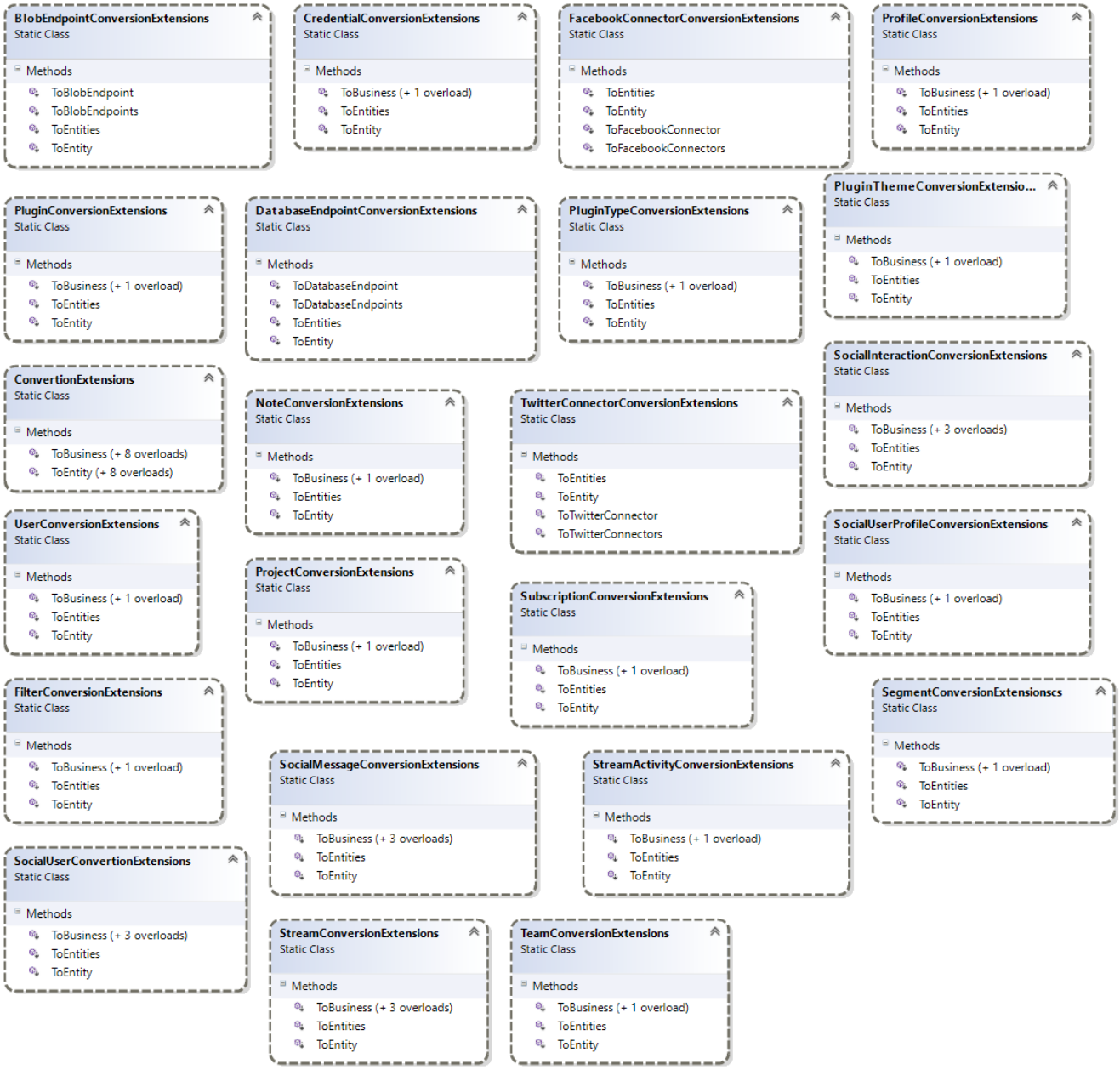
Enums



Configuration

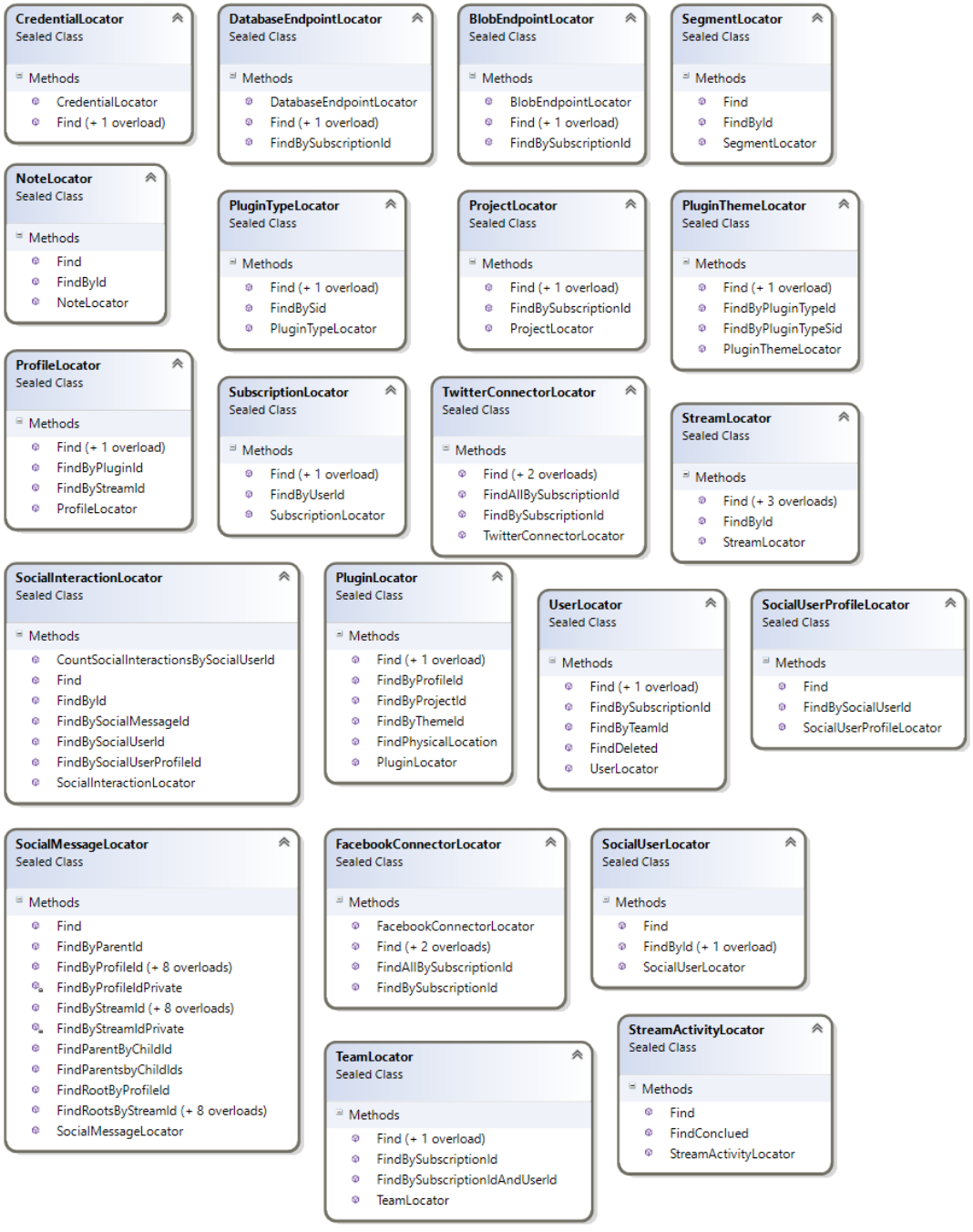


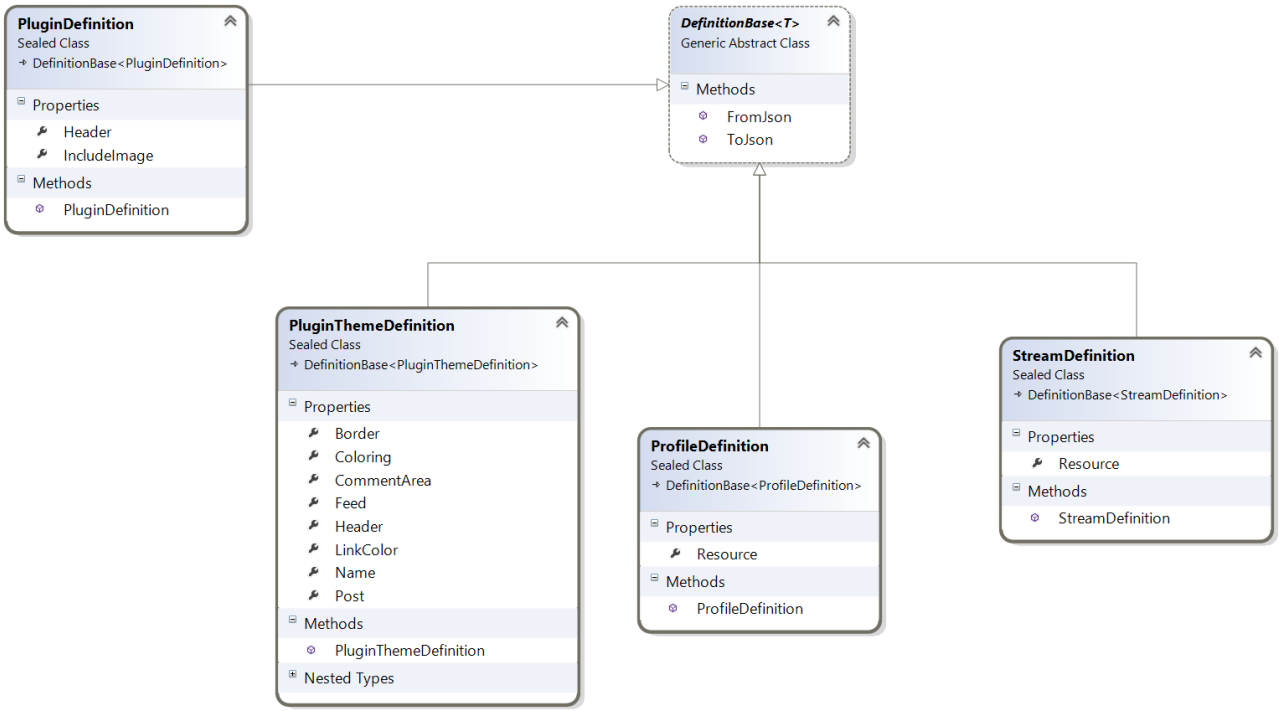
ConversionsExtensions

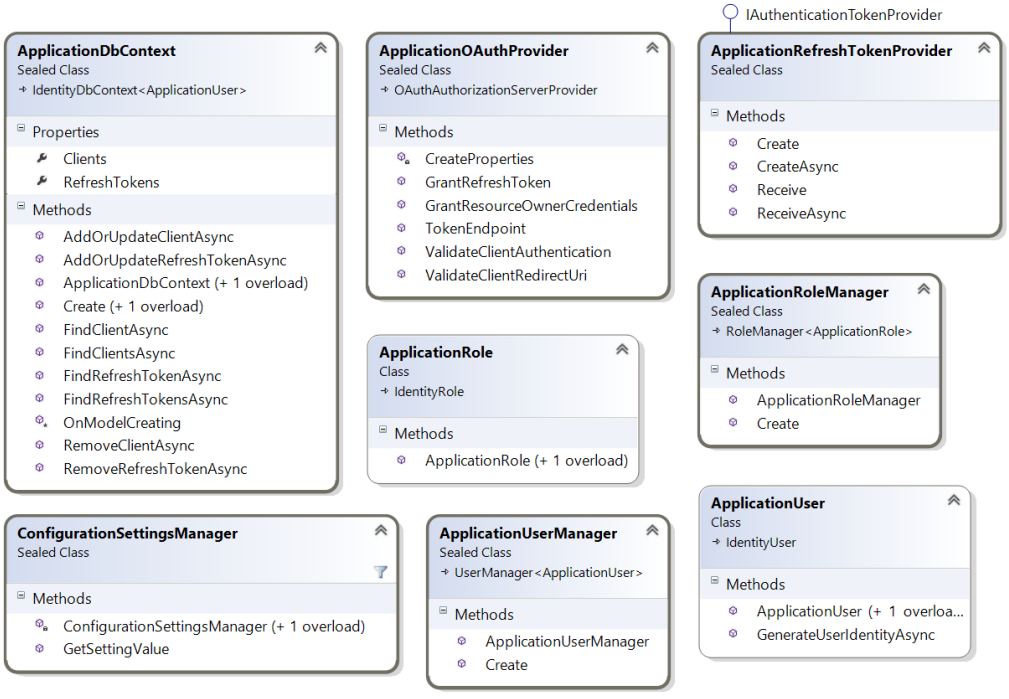


Manipulators & Relators

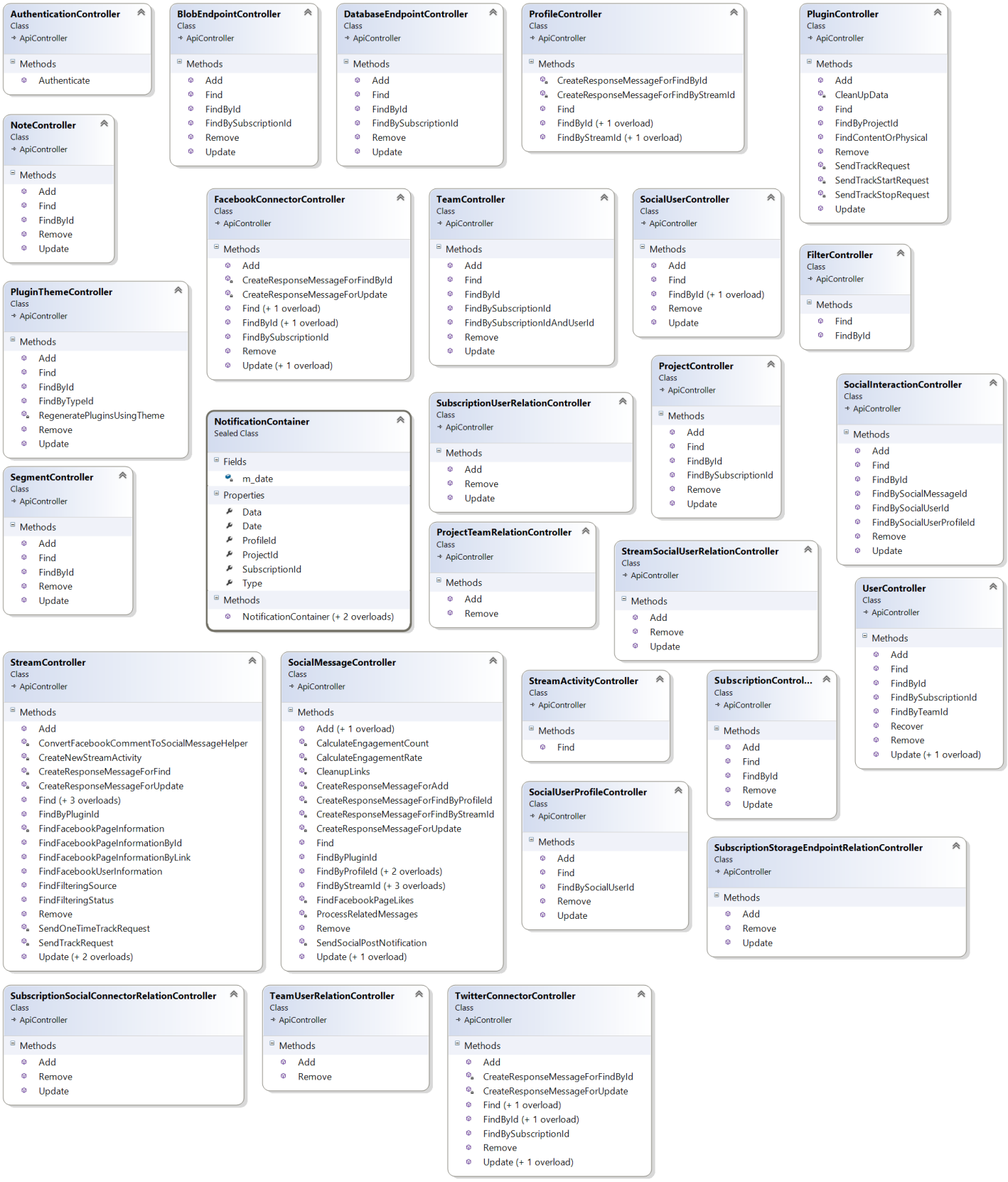
Locators


CredentialLocator
Sealed Class
Methods
CredentialLocator
Find (+ 1 overload)
DatabaseEndpointLocator
Sealed Class
Methods
DatabaseEndpointLocator
Find (+ 1 overload)
FindBySubscriptionId
BlobEndpointLocator
Sealed Class
Methods
BlobEndpointLocator
Find (+ 1 overload)
FindBySubscriptionId
SegmentLocator
Sealed Class
Methods
Find
FindById
SegmentLocator
NoteLocator
Sealed Class
Methods
Find
FindById
NoteLocator
PluginTypeLocator
Sealed Class
Methods
Find (+ 1 overload)
FindBySid
PluginTypeLocator
ProjectLocator
Sealed Class
Methods
Find (+ 1 overload)
FindBySubscriptionId
ProjectLocator
PluginThemeLocator
Sealed Class
Methods
Find (+ 1 overload)
FindByPluginTypeId
FindByPluginTypeSid
PluginThemeLocator
ProfileLocator
Sealed Class
Methods
Find (+ 1 overload)
FindByPluginId
FindByStreamId
ProfileLocator
SubscriptionLocator
Sealed Class
Methods
Find (+ 1 overload)
FindByUserId
SubscriptionLocator
TwitterConnectorLocator
Sealed Class
Methods
Find (+ 2 overloads)
FindAllBySubscriptionId
FindBySubscriptionId
TwitterConnectorLocator
StreamLocator
Sealed Class
Methods
Find (+ 3 overloads)
FindById
StreamLocator
SocialInteractionLocator
Sealed Class
Methods
CountSocialInteractionsBySocialUserId
Find
FindById
FindBySocialMessageId
FindBySocialUserId
FindBySocialUserProfileId
SocialInteractionLocator
PluginLocator
Sealed Class
Methods
Find (+ 1 overload)
FindByProfileId
FindByProjectId
FindByThemeId
FindPhysicalLocation
PluginLocator
UserLocator
Sealed Class
Methods
Find (+ 1 overload)
FindBySubscriptionId
FindByTeamId
FindDeleted
UserLocator
SocialUserProfileLocator
Sealed Class
Methods
Find
FindBySocialUserId
SocialUserProfileLocator
SocialMessageLocator
Sealed Class
Methods
Find
FindByParentId
FindByProfileId (+ 8 overloads)
FindByProfileIdPrivate
FindByStreamId (+ 8 overloads)
FindByStreamIdPrivate
FindParentByChildId
FindParentsbyChildIds
FindRootByProfileId
FindRootsByStreamId (+ 8 overloads)
SocialMessageLocator
FacebookConnectorLocator
Sealed Class
Methods
FacebookConnectorLocator
Find (+ 2 overloads)
FindAllBySubscriptionId
FindBySubscriptionId
SocialUserLocator
Sealed Class
Methods
Find
FindById (+ 1 overload)
SocialUserLocator
TeamLocator
Sealed Class
Methods
Find (+ 1 overload)
FindBySubscriptionId
FindBySubscriptionIdAndUserId
TeamLocator
StreamActivityLocator
Sealed Class
Methods
Find
FindConclued
StreamActivityLocator

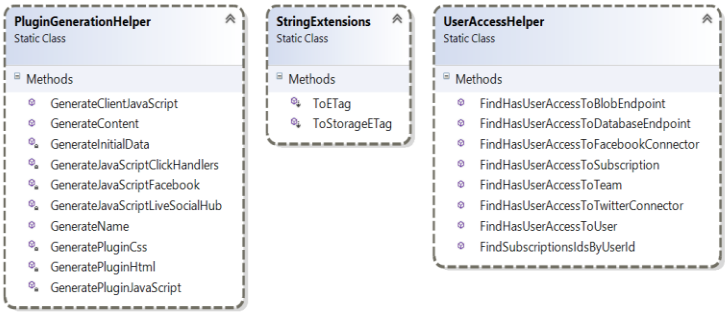

PluginDefinition
Sealed Class
DefinitionBase<PluginDefinition>
Properties
Header
IncludeImage
Methods
PluginDefinition
DefinitionBase<T>
Generic Abstract Class
Methods
FromJson
ToJson
PluginThemeDefinition
Sealed Class
DefinitionBase<PluginThemeDefinition>
Properties
Border
Coloring
CommentArea
Feed
Header
LinkColor
Name
Post
Methods
PluginThemeDefinition
Nested Types
ProfileDefinition
Sealed Class
DefinitionBase<ProfileDefinition>
Properties
Resource
Methods
ProfileDefinition
StreamDefinition
Sealed Class
DefinitionBase<StreamDefinition>
Properties
Resource
Methods
StreamDefinition


ApplicationDbContext
Sealed Class
IdentityDbContext<ApplicationUser>
Properties
Clients
RefreshTokens
Methods
AddOrUpdateClientAsync
AddOrUpdateRefreshTokenAsync
ApplicationDbContext (+ 1 overload)
Create (+ 1 overload)
FindClientAsync
FindClientsAsync
FindRefreshTokenAsync
FindRefreshTokensAsync
OnModelCreating
RemoveClientAsync
RemoveRefreshTokenAsync
ApplicationOAuthProvider
Sealed Class
OAuthAuthorizationServerProvider
Methods
CreateProperties
GrantRefreshToken
GrantResourceOwnerCredentials
TokenEndpoint
ValidateClientAuthentication
ValidateClientRedirectUri
IAuthenticationTokenProvider
ApplicationRefreshTokenProvider
Sealed Class
Methods
Create
CreateAsync
Receive
ReceiveAsync
ApplicationRoleManager
Sealed Class
RoleManager<ApplicationRole>
Methods
ApplicationRoleManager
Create
ApplicationRole
Class
IdentityRole
Methods
ApplicationRole (+ 1 overload)
ConfigurationSettingsManager
Sealed Class
Methods
ConfigurationSettingsManager (+ 1 overload)
GetSettingValue
ApplicationUserManager
Sealed Class
UserManager<ApplicationUser>
Methods
ApplicationUserManager
Create
ApplicationUser
Class
IdentityUser
Methods
ApplicationUser (+ 1 overloa...
GenerateUserIdentityAsync

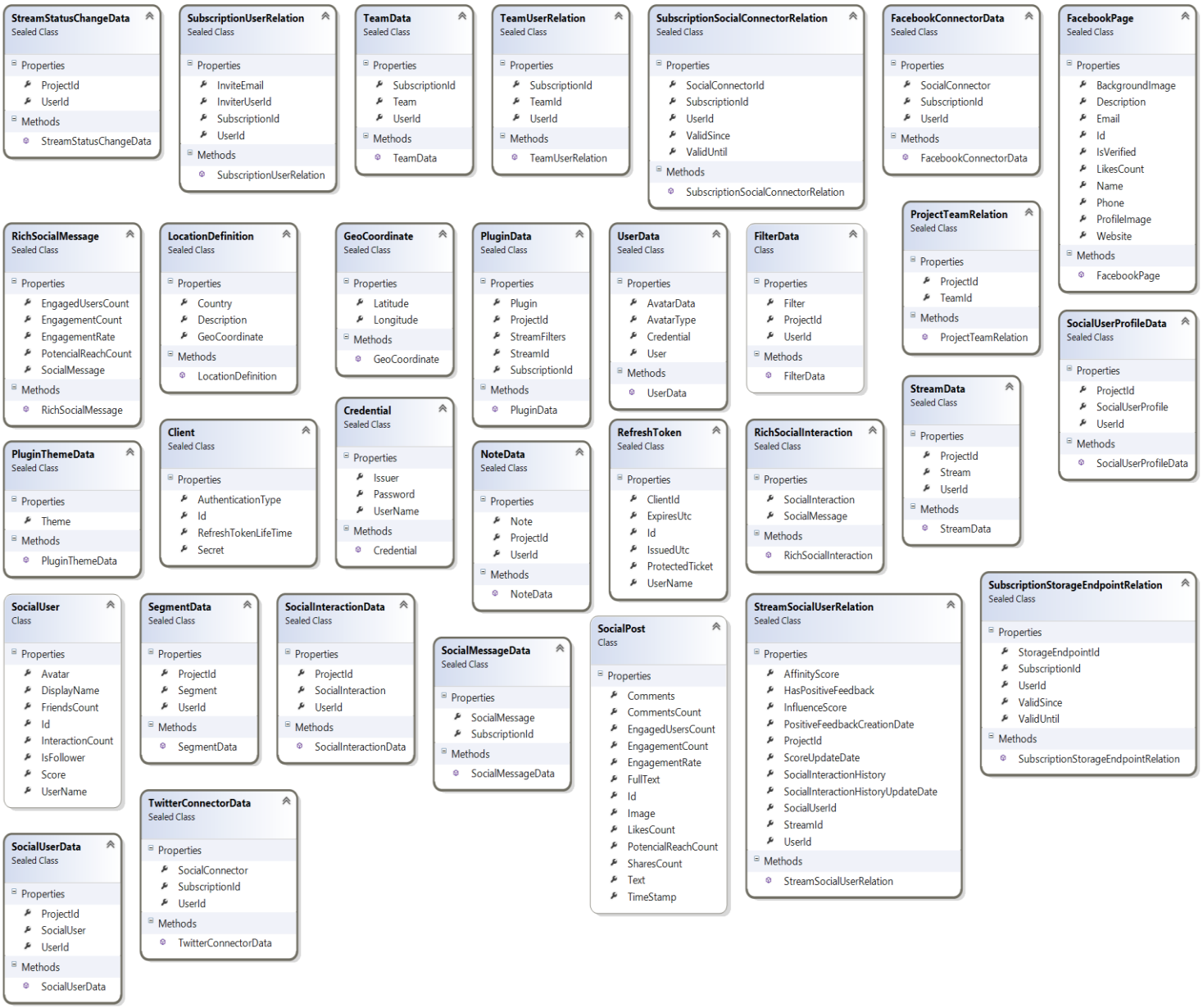
Controllers



Helpers & Extensions



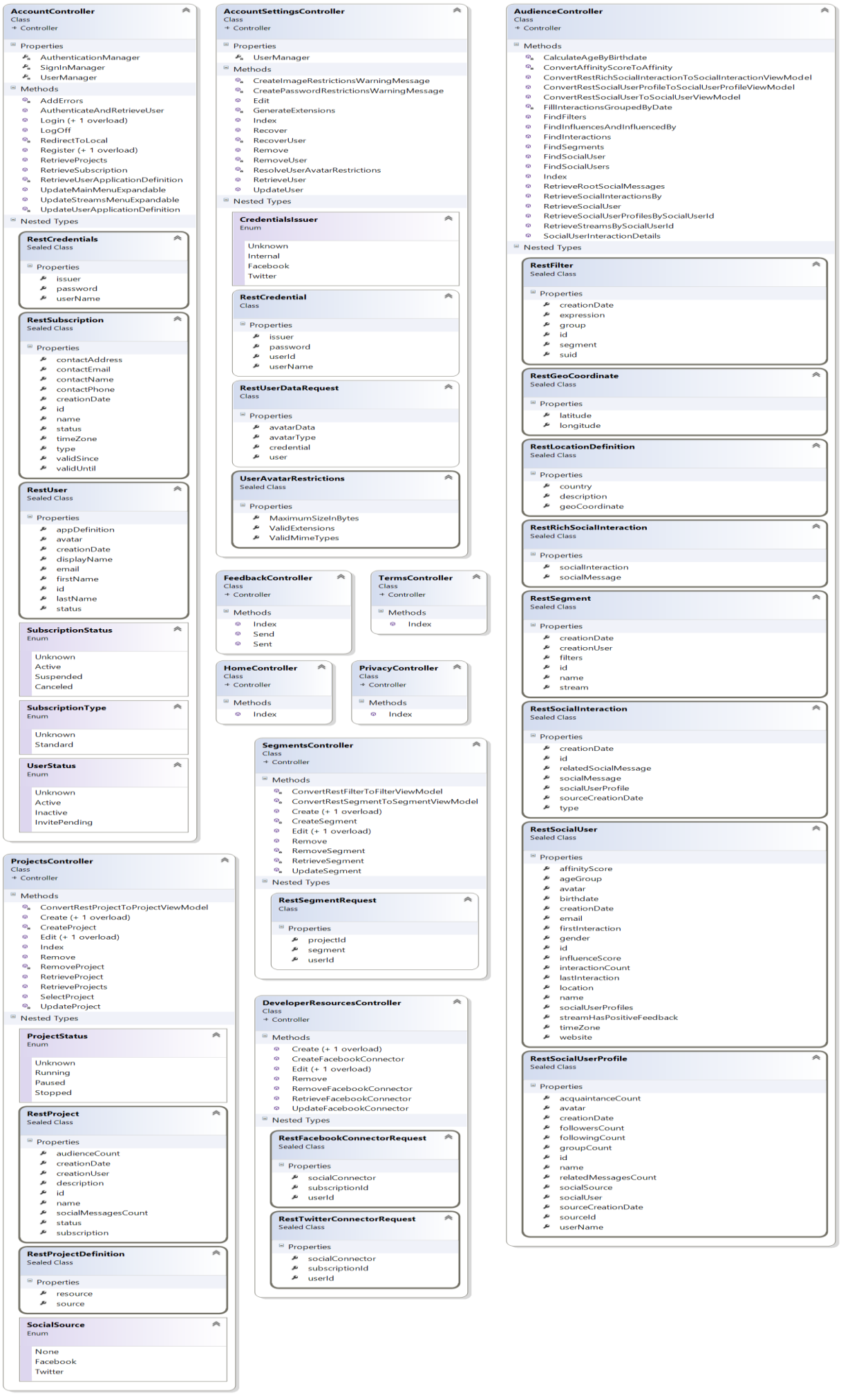
Data

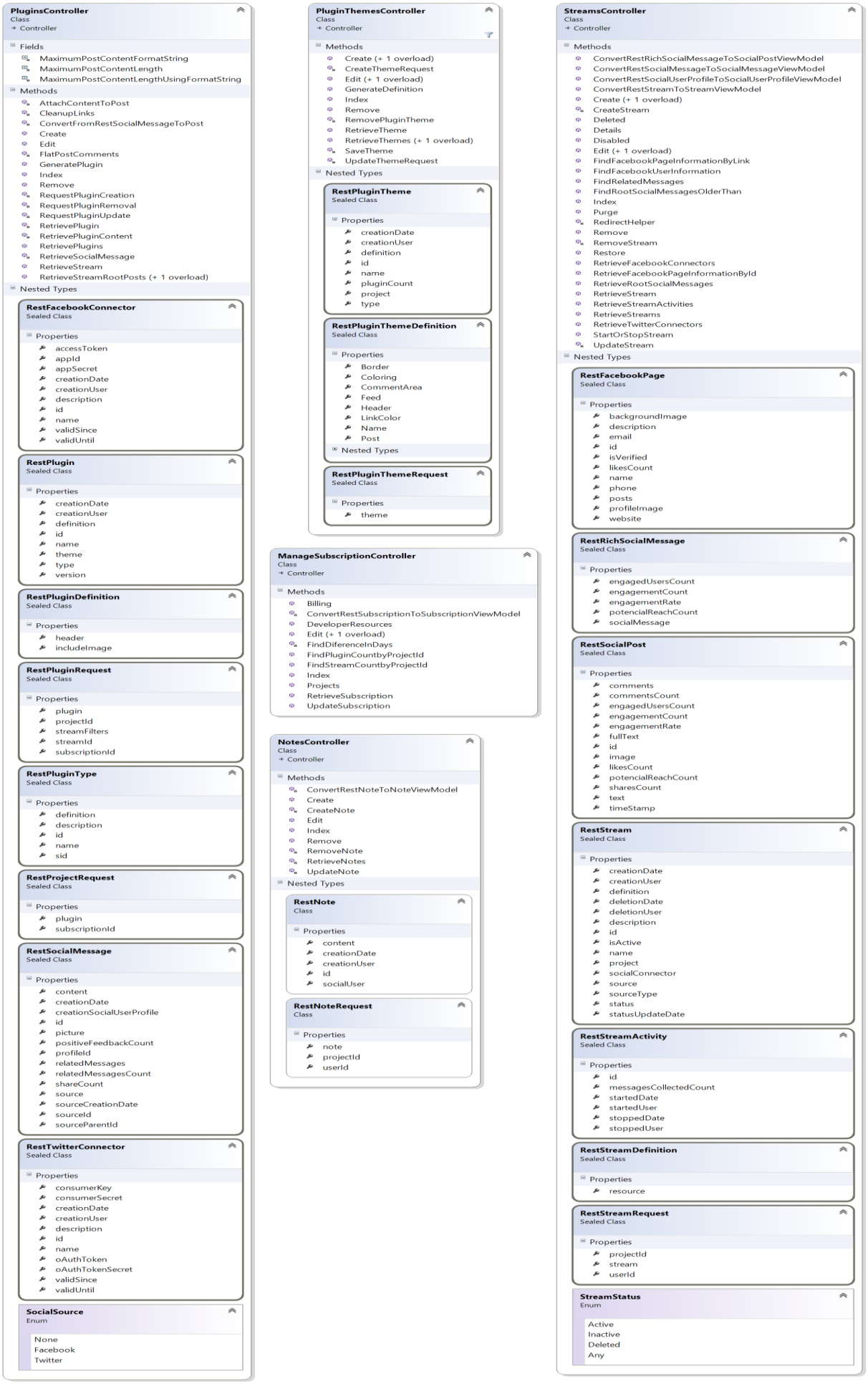


Anexo 5

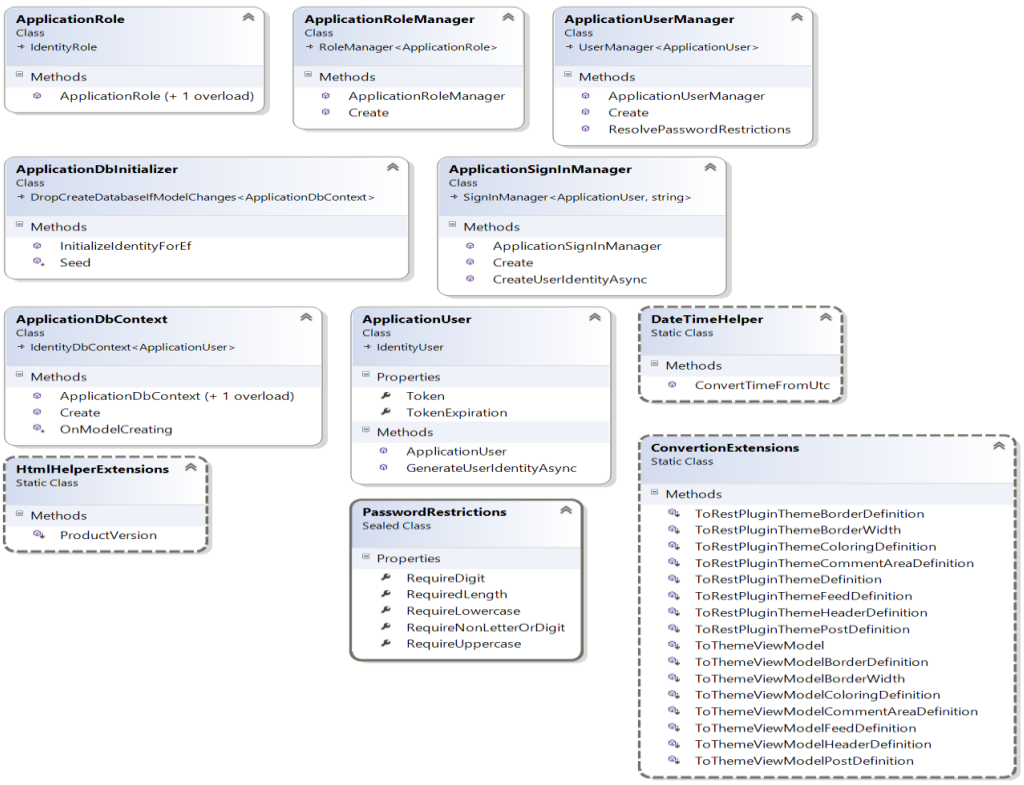
Web Front-End - Diagrama de Classes

Controllers

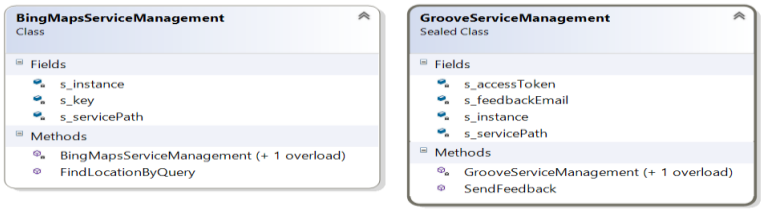



PluginsController
Class
Controller
Fields
MaximumPostContentFormatString
MaximumPostContentLength
MaximumPostContentLengthUsingFormatString
Methods
AttachContentToPost
CleanupLinks
ConvertFromRestSocialMessageToPost
Create
Edit
FlatPostComments
GeneratePlugin
Index
Remove
RequestPluginCreation
RequestPluginRemoval
RequestPluginUpdate
RetrievePlugin
RetrievePluginContent
RetrievePlugins
RetrieveSocialMessage
RetrieveStream
RetrieveStreamRootPosts (+ 1 overload)
Nested Types
RestFacebookConnector
Sealed Class
Properties
accessToken
appId
appSecret
creationDate
creationUser
description
id
name
validSince
validUntil
RestPlugin
Sealed Class
Properties
creationDate
creationUser
definition
id
name
theme
type
version
RestPluginDefinition
Sealed Class
Properties
header
includeImage
RestPluginRequest
Sealed Class
Properties
plugin
projectId
streamFilters
streamId
subscriptionId
RestPluginType
Sealed Class
Properties
definition
description
id
name
sid
RestProjectRequest
Sealed Class
Properties
plugin
subscriptionId
RestSocialMessage
Sealed Class
Properties
content
creationDate
creationSocialUserProfile
id
picture
positiveFeedbackCount
profileId
relatedMessages
relatedMessagesCount
shareCount
source
sourceCreationDate
sourceId
sourceParentId
RestTwitterConnector
Sealed Class
Properties
consumerKey
consumerSecret
creationDate
creationUser
description
id
name
oAuthToken
oAuthTokenSecret
validSince
validUntil
SocialSource
Enum
None
Facebook
Twitter
PluginThemesController
Class
Controller
Methods
Create (+ 1 overload)
CreateThemeRequest
Edit (+ 1 overload)
GenerateDefinition
Index
Remove
RemovePluginTheme
RetrieveTheme
RetrieveThemes (+ 1 overload)
SaveTheme
UpdateThemeRequest
Nested Types
RestPluginTheme
Sealed Class
Properties
creationDate
creationUser
definition
id
name
pluginCount
project
type
RestPluginThemeDefinition
Sealed Class
Properties
Border
Coloring
CommentArea
Feed
Header
LinkColor
Name
Post
Nested Types
RestPluginThemeRequest
Sealed Class
Properties
theme
ManageSubscriptionController
Class
Controller
Methods
Billing
ConvertRestSubscriptionToSubscriptionViewModel
DeveloperResources
Edit (+ 1 overload)
FindDiferenceInDays
FindPluginCountbyProjectId
FindStreamCountbyProjectId
Index
Projects
RetrieveSubscription
UpdateSubscription
NotesController
Class
Controller
Methods
ConvertRestNoteToNoteViewModel
Create
CreateNote
Edit
Index
Remove
RemoveNote
RetrieveNotes
UpdateNote
Nested Types
RestNote
Class
Properties
content
creationDate
creationUser
id
socialUser
RestNoteRequest
Class
Properties
note
projectId
userId
StreamsController
Class
Controller
Methods
ConvertRestRichSocialMessageToSocialPostViewModel
ConvertRestSocialMessageToSocialMessageViewModel
ConvertRestSocialUserProfileToSocialUserProfileViewModel
ConvertRestStreamToStreamViewModel
Create (+ 1 overload)
CreateStream
Deleted
Details
Disabled
Edit (+ 1 overload)
FindFacebookPageInformationByLink
FindFacebookUserInformation
FindRelatedMessages
FindRootSocialMessagesOlderThan
Index
Purge
RedirectHelper
Remove
RemoveStream
Restore
RetrieveFacebookConnectors
RetrieveFacebookPageInformationById
RetrieveRootSocialMessages
RetrieveStream
RetrieveStreamActivities
RetrieveStreams
RetrieveTwitterConnectors
StartOrStopStream
UpdateStream
Nested Types
RestFacebookPage
Sealed Class
Properties
backgroundImage
description
email
id
isVerified
likesCount
name
phone
posts
profileImage
website
RestRichSocialMessage
Sealed Class
Properties
engagedUsersCount
engagementCount
engagementRate
potencialReachCount
socialMessage
RestSocialPost
Sealed Class
Properties
comments
commentsCount
engagedUsersCount
engagementCount
engagementRate
fullText
id
image
likesCount
potencialReachCount
sharesCount
text
timeStamp
RestStream
Sealed Class
Properties
creationDate
creationUser
definition
deletionDate
deletionUser
description
id
isActive
name
project
socialConnector
source
sourceType
status
statusUpdateDate
RestStreamActivity
Sealed Class
Properties
id
messagesCollectedCount
startedDate
startedUser
stoppedDate
stoppedUser
RestStreamDefinition
Sealed Class
Properties
resource
RestStreamRequest
Sealed Class
Properties
projectId
stream
userId
StreamStatus
Enum
Active
Inactive
Deleted
Any

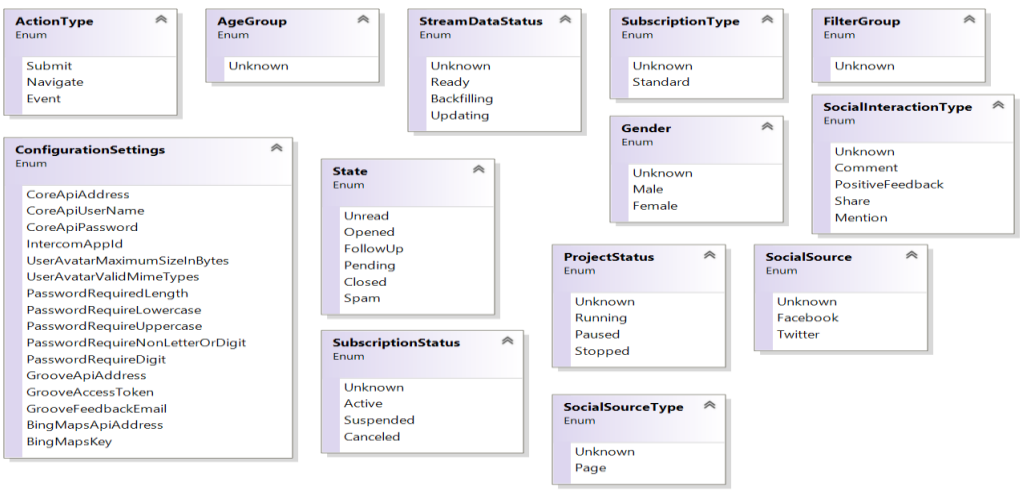
Configuration



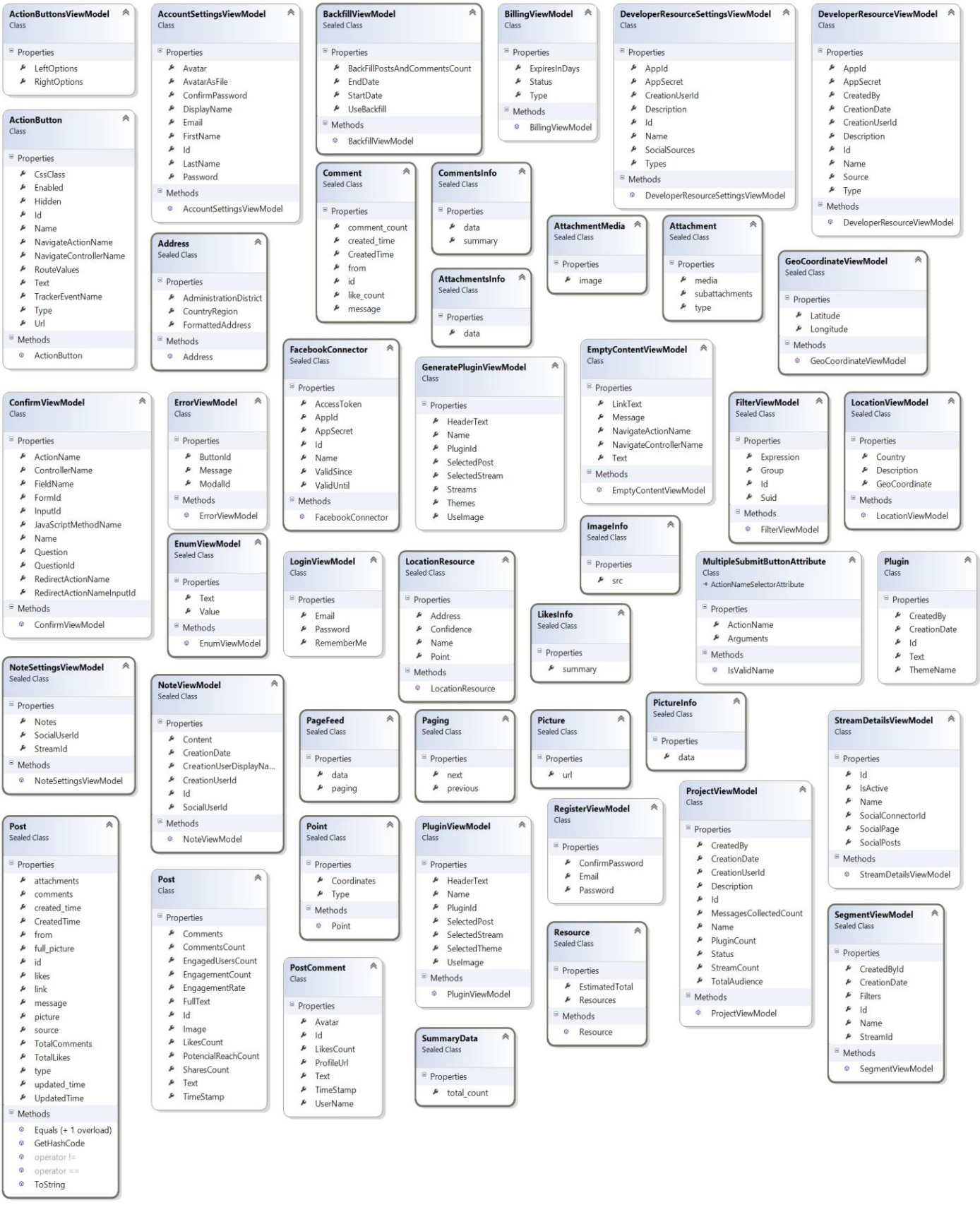
Services Management

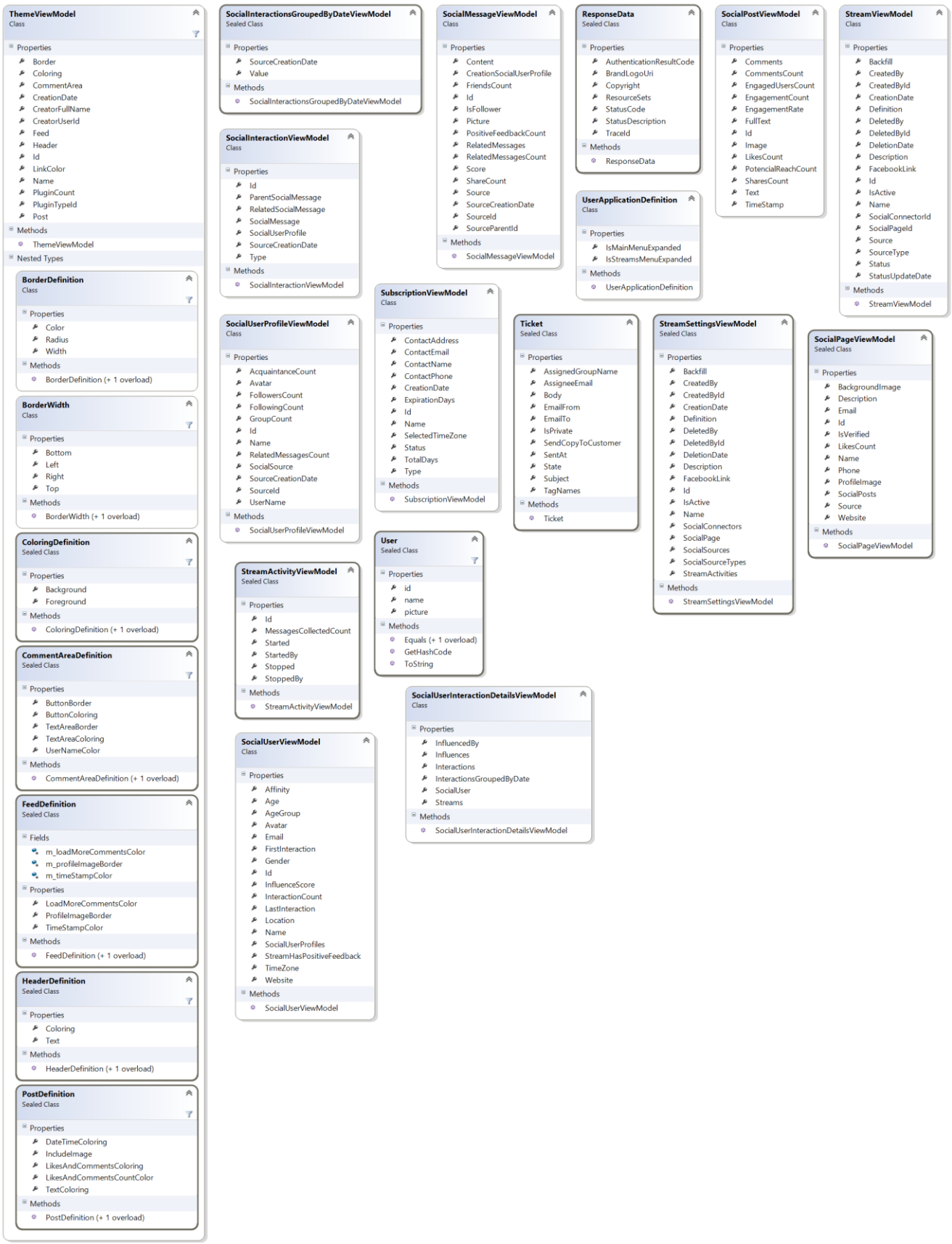


Enums



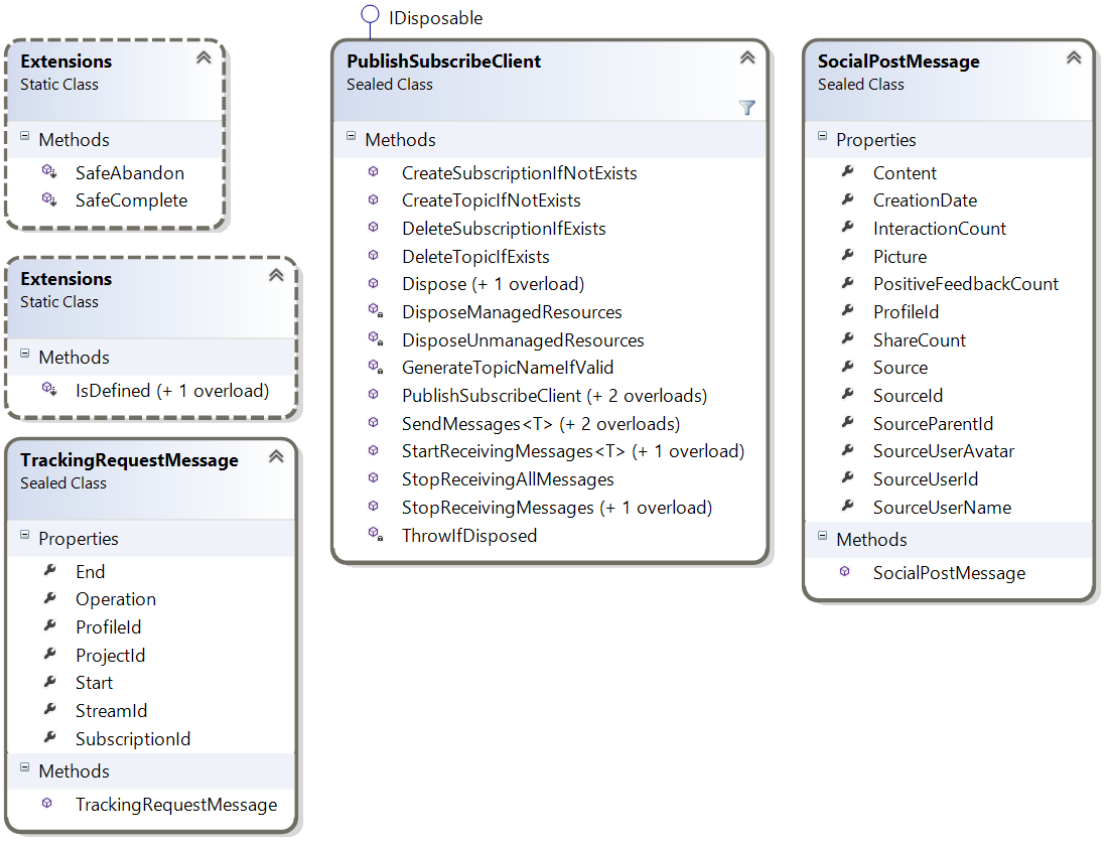
Data


ActionButtonsViewModel
Class
Properties
LeftOptions
RightOptions
ActionButton
Class
Properties
CssClass
Enabled
Hidden
Id
Name
NavigateActionName
NavigateControllerName
RouteValues
Text
TrackerEventName
Type
Url
Methods
ActionButton
AccountSettingsViewModel
Class
Properties
Avatar
AvatarAsFile
ConfirmPassword
DisplayName
Email
FirstName
Id
LastName
Password
Methods
AccountSettingsViewModel
Address
Sealed Class
Properties
AdministrationDistrict
CountryRegion
FormattedAddress
Methods
Address
BackfillViewModel
Sealed Class
Properties
BackFillPostsAndCommentsCount
EndDate
StartDate
UseBackfill
Methods
BackfillViewModel
Comment
Sealed Class
Properties
comment_count
created_time
CreatedTime
from
id
like_count
message
CommentsInfo
Sealed Class
Properties
data
summary
AttachmentsInfo
Sealed Class
Properties
data
BillingViewModel
Class
Properties
ExpiresInDays
Status
Type
Methods
BillingViewModel
AttachmentMedia
Sealed Class
Properties
image
Attachment
Sealed Class
Properties
media
subattachments
type
DeveloperResourceSettingsViewModel
Class
Properties
AppId
AppSecret
CreationUserId
Description
Id
Name
SocialSources
Types
Methods
DeveloperResourceSettingsViewModel
DeveloperResourceViewModel
Class
Properties
AppId
AppSecret
CreatedBy
CreationDate
CreationUserId
Description
Id
Name
Source
Type
Methods
DeveloperResourceViewModel
GeoCoordinateViewModel
Sealed Class
Properties
Latitude
Longitude
Methods
GeoCoordinateViewModel
FacebookConnector
Sealed Class
Properties
AccessToken
AppId
AppSecret
Id
Name
ValidSince
ValidUntil
Methods
FacebookConnector
GeneratePluginViewModel
Class
Properties
HeaderText
Name
PluginId
SelectedPost
SelectedStream
Streams
Themes
UseImage
EmptyContentViewModel
Class
Properties
LinkText
Message
NavigateActionName
NavigateControllerName
Text
Methods
EmptyContentViewModel
ConfirmViewModel
Class
Properties
ActionName
ControllerName
FieldName
FormId
InputId
JavaScriptMethodName
Name
Question
QuestionId
RedirectActionName
RedirectActionNameInputId
Methods
ConfirmViewModel
ErrorViewModel
Class
Properties
ButtonId
Message
ModalId
Methods
ErrorViewModel
EnumViewModel
Sealed Class
Properties
Text
Value
Methods
EnumViewModel
LoginViewModel
Class
Properties
Email
Password
RememberMe
LocationResource
Sealed Class
Properties
Address
Confidence
Name
Point
Methods
LocationResource
ImageInfo
Sealed Class
Properties
src
LikesInfo
Sealed Class
Properties
summary
FilterViewModel
Sealed Class
Properties
Expression
Group
Id
Suid
Methods
FilterViewModel
LocationViewModel
Sealed Class
Properties
Country
Description
GeoCoordinate
Methods
LocationViewModel
MultipleSubmitButtonAttribute
Class
ActionNameSelectorAttribute
Properties
ActionName
Arguments
Methods
IsValidName
Plugin
Class
Properties
CreatedBy
CreationDate
Id
Text
ThemeName
NoteSettingsViewModel
Sealed Class
Properties
Notes
SocialUserId
StreamId
Methods
NoteSettingsViewModel
NoteViewModel
Sealed Class
Properties
Content
CreationDate
CreationUserDisplayNa...
CreationUserId
Id
SocialUserId
Methods
NoteViewModel
PageFeed
Sealed Class
Properties
data
paging
Paging
Sealed Class
Properties
next
previous
Picture
Sealed Class
Properties
url
PictureInfo
Sealed Class
Properties
data
StreamDetailsViewModel
Class
Properties
Id
IsActive
Name
SocialConnectorId
SocialPage
SocialPosts
Methods
StreamDetailsViewModel
Post
Sealed Class
Properties
attachments
comments
created_time
CreatedTime
from
full_picture
id
likes
link
message
picture
source
TotalComments
TotalLikes
type
updated_time
UpdatedTime
Methods
Equals (+ 1 overload)
GetHashCode
operator !=
operator ==
ToString
Post
Class
Properties
Comments
CommentsCount
EngagedUsersCount
EngagementCount
EngagementRate
FullText
Id
Image
LikesCount
PotencialReachCount
SharesCount
Text
TimeStamp
Point
Sealed Class
Properties
Coordinates
Type
Methods
Point
PluginViewModel
Class
Properties
HeaderText
Name
PluginId
SelectedPost
SelectedStream
SelectedTheme
UseImage
Methods
PluginViewModel
RegisterViewModel
Class
Properties
ConfirmPassword
Email
Password
ProjectViewModel
Class
Properties
CreatedBy
CreationDate
CreationUserId
Description
Id
MessagesCollectedCount
Name
PluginCount
Status
StreamCount
TotalAudience
Methods
ProjectViewModel
Resource
Sealed Class
Properties
EstimatedTotal
Resources
Methods
Resource
SegmentViewModel
Sealed Class
Properties
CreatedById
CreationDate
Filters
Id
Name
StreamId
Methods
SegmentViewModel
PostComment
Class
Properties
Avatar
Id
LikesCount
ProfileUrl
Text
TimeStamp
UserName
SummaryData
Sealed Class
Properties
total_count

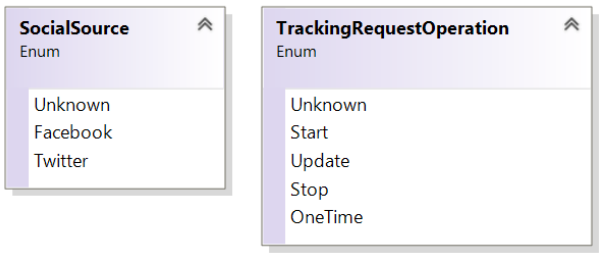

ThemeViewModel
Class
Properties
Border
Coloring
CommentArea
CreationDate
CreatorFullName
CreatorUserId
Feed
Header
Id
LinkColor
Name
PluginCount
PluginTypeId
Post
Methods
ThemeViewModel
Nested Types
BorderDefinition
Class
Properties
Color
Radius
Width
Methods
BorderDefinition (+ 1 overload)
BorderWidth
Class
Properties
Bottom
Left
Right
Top
Methods
BorderWidth (+ 1 overload)
ColoringDefinition
Sealed Class
Properties
Background
Foreground
Methods
ColoringDefinition (+ 1 overload)
CommentAreaDefinition
Sealed Class
Properties
ButtonBorder
ButtonColoring
TextAreaBorder
TextAreaColoring
UserNameColor
Methods
CommentAreaDefinition (+ 1 overload)
FeedDefinition
Sealed Class
Fields
m_loadMoreCommentsColor
m_profileImageBorder
m_timeStampColor
Properties
LoadMoreCommentsColor
ProfileImageBorder
TimeStampColor
Methods
FeedDefinition (+ 1 overload)
HeaderDefinition
Sealed Class
Properties
Coloring
Text
Methods
HeaderDefinition (+ 1 overload)
PostDefinition
Sealed Class
Properties
DateTimeColoring
IncludeImage
LikesAndCommentsColoring
LikesAndCommentsCountColor
TextColoring
Methods
PostDefinition (+ 1 overload)
SocialInteractionsGroupedByDateViewModel
Sealed Class
Properties
SourceCreationDate
Value
Methods
SocialInteractionsGroupedByDateViewModel
SocialInteractionViewModel
Class
Properties
Id
ParentSocialMessage
RelatedSocialMessage
SocialMessage
SocialUserProfile
SourceCreationDate
Type
Methods
SocialInteractionViewModel
SocialUserProfileViewModel
Class
Properties
AcquaintanceCount
Avatar
FollowersCount
FollowingCount
GroupCount
Id
Name
RelatedMessagesCount
SocialSource
SourceCreationDate
SourceId
UserName
Methods
SocialUserProfileViewModel
StreamActivityViewModel
Sealed Class
Properties
Id
MessagesCollectedCount
Started
StartedBy
Stopped
StoppedBy
Methods
StreamActivityViewModel
SocialUserViewModel
Class
Properties
Affinity
Age
AgeGroup
Avatar
Email
FirstInteraction
Gender
Id
InfluenceScore
InteractionCount
LastInteraction
Location
Name
SocialUserProfiles
StreamHasPositiveFeedback
TimeZone
Website
Methods
SocialUserViewModel
SocialMessageViewModel
Class
Properties
Content
CreationSocialUserProfile
FriendsCount
Id
IsFollower
Picture
PositiveFeedbackCount
RelatedMessages
RelatedMessagesCount
Score
ShareCount
Source
SourceCreationDate
SourceId
SourceParentId
Methods
SocialMessageViewModel
SubscriptionViewModel
Class
Properties
ContactAddress
ContactEmail
ContactName
ContactPhone
CreationDate
ExpirationDays
Id
Name
SelectedTimeZone
Status
TotalDays
Type
Methods
SubscriptionViewModel
User
Sealed Class
Properties
id
name
picture
Methods
Equals (+ 1 overload)
GetHashCode
ToString
SocialUserInteractionDetailsViewModel
Class
Properties
InfluencedBy
Influences
Interactions
InteractionsGroupedByDate
SocialUser
Streams
Methods
SocialUserInteractionDetailsViewModel
ResponseData
Sealed Class
Properties
AuthenticationResultCode
BrandLogoUri
Copyright
ResourceSets
StatusCode
StatusDescription
TraceId
Methods
ResponseData
UserApplicationDefinition
Class
Properties
IsMainMenuExpanded
IsStreamsMenuExpanded
Methods
UserApplicationDefinition
Ticket
Sealed Class
Properties
AssignedGroupName
AssigneeEmail
Body
EmailFrom
EmailTo
IsPrivate
SendCopyToCustomer
SentAt
State
Subject
TagNames
Methods
Ticket
SocialPostViewModel
Class
Properties
Comments
CommentsCount
EngagedUsersCount
EngagementCount
EngagementRate
FullText
Id
Image
LikesCount
PotencialReachCount
SharesCount
Text
TimeStamp
StreamSettingsViewModel
Sealed Class
Properties
Backfill
CreatedBy
CreatedById
CreationDate
Definition
DeletedBy
DeletedById
DeletionDate
Description
FacebookLink
Id
IsActive
Name
SocialConnectors
SocialPage
SocialSources
SocialSourceTypes
StreamActivities
Methods
StreamSettingsViewModel
StreamViewModel
Class
Properties
Backfill
CreatedBy
CreatedById
CreationDate
Definition
DeletedBy
DeletedById
DeletionDate
Description
FacebookLink
Id
IsActive
Name
SocialConnectorId
SocialPageId
Source
SourceType
Status
StatusUpdateDate
Methods
StreamViewModel
SocialPageViewModel
Sealed Class
Properties
BackgroundImage
Description
Email
Id
IsVerified
LikesCount
Name
Phone
ProfileImage
SocialPosts
Source
Website
Methods
SocialPageViewModel

Anexo 6

Mensagens (Messaging) - Diagrama de Classes

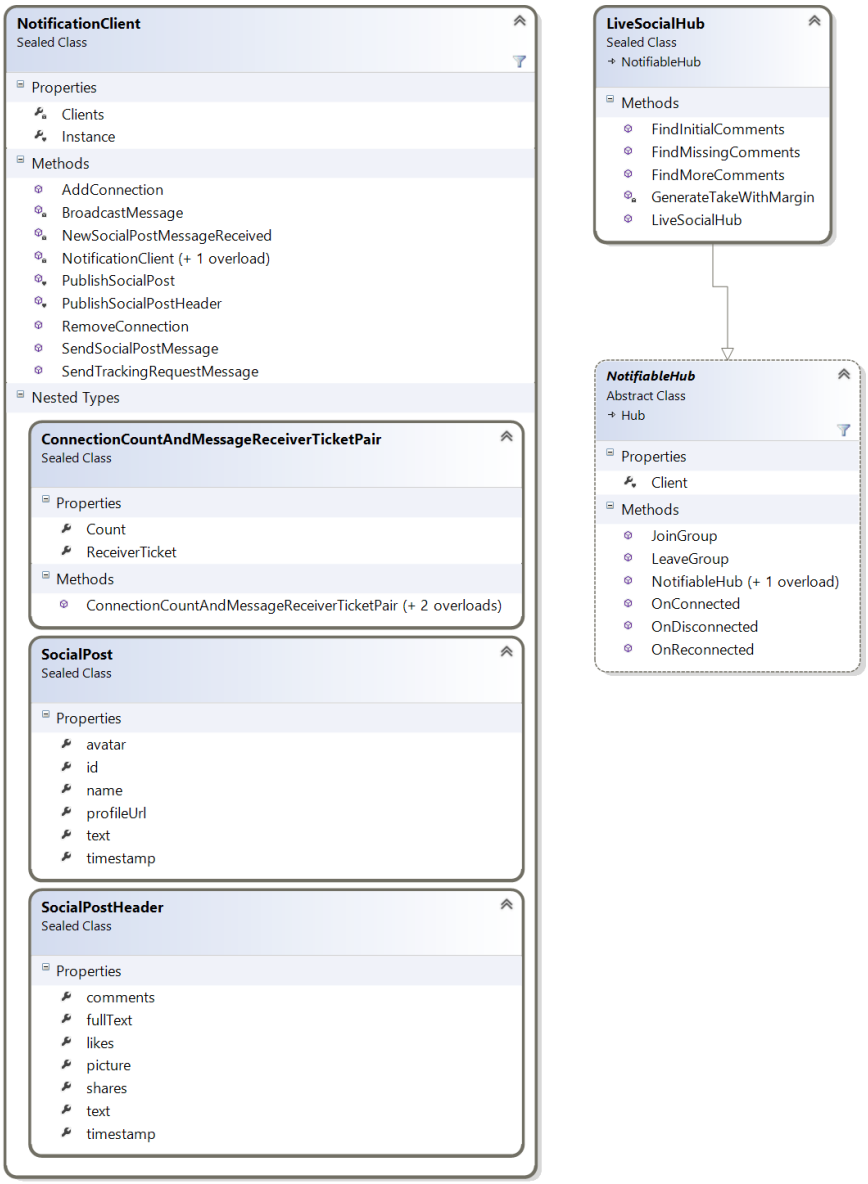

IDisposable
Extensions
Static Class
Methods
SafeAbandon
SafeComplete
Extensions
Static Class
Methods
IsDefined (+ 1 overload)
TrackingRequestMessage
Sealed Class
Properties
End
Operation
ProfileId
ProjectId
Start
StreamId
SubscriptionId
Methods
TrackingRequestMessage
PublishSubscribeClient
Sealed Class
Methods
CreateSubscriptionIfNotExists
CreateTopicIfNotExists
DeleteSubscriptionIfExists
DeleteTopicIfExists
Dispose (+ 1 overload)
DisposeManagedResources
DisposeUnmanagedResources
GenerateTopicNameIfValid
PublishSubscribeClient (+ 2 overloads)
SendMessages<T> (+ 2 overloads)
StartReceivingMessages<T> (+ 1 overload)
StopReceivingAllMessages
StopReceivingMessages (+ 1 overload)
ThrowIfDisposed
SocialPostMessage
Sealed Class
Properties
Content
CreationDate
InteractionCount
Picture
PositiveFeedbackCount
ProfileId
ShareCount
Source
SourceId
SourceParentId
SourceUserAvatar
SourceUserId
SourceUserName
Methods
SocialPostMessage

Enums


SocialSource
Enum
Unknown
Facebook
Twitter
TrackingRequestOperation
Enum
Unknown
Start
Update
Stop
OneTime

Anexo 7

Notificações (Notifications) – Diagrama de Classes


NotificationClient
Sealed Class
Properties
Clients
Instance
Methods
AddConnection
BroadcastMessage
NewSocialPostMessageReceived
NotificationClient (+ 1 overload)
PublishSocialPost
PublishSocialPostHeader
RemoveConnection
SendSocialPostMessage
SendTrackingRequestMessage
Nested Types
ConnectionCountAndMessageReceiverTicketPair
Sealed Class
Properties
Count
ReceiverTicket
Methods
ConnectionCountAndMessageReceiverTicketPair (+ 2 overloads)
SocialPost
Sealed Class
Properties
avatar
id
name
profileUrl
text
timestamp
SocialPostHeader
Sealed Class
Properties
comments
fullText
likes
picture
shares
text
timestamp
LiveSocialHub
Sealed Class
NotifiableHub
Methods
FindInitialComments
FindMissingComments
FindMoreComments
GenerateTakeWithMargin
LiveSocialHub
NotifiableHub
Abstract Class
Hub
Properties
Client
Methods
JoinGroup
LeaveGroup
NotifiableHub (+ 1 overload)
OnConnected
OnDisconnected
OnReconnected