

# Atelier de développement d'application Facebook

Importation de données d'un groupe  
Facebook vers une BD externe

*Ce document explique en détail comment développer une vraie application Facebook permettant d'importer les données d'un groupe Facebook vers une base de données externe*

Bilali Soidik

Master Qualité du logiciel

## بسم الله الرحمن الرحيم

# Table des matières

|   |    |
|---|----|
| Table des matières.....   | 1  |
| Introduction .....  | 3  |
| Chapitre 1 : Création et Initialisation de l'application .....                          | 4  |
| I. Inscription en tant que développeur(obligatoire).....                                | 4  |
| II. Création d'application.....   | 5  |
| III. Paramètres de l'application.....   | 6  |
| A. Paramètres essentiels.....   | 6  |
| B. Mode d'intégration de l'application à Facebook.....                                  | 7  |
| C. Intégration avec le mode Website With Facebook Login.....                            | 9  |
| 1.L'URL du site.....  | 9  |
| D. Création d'URL pour site local avec hôte virtuel sous apache .....                   | 9  |
| 1.Configuration de Windows et Linux pour créer un nom de domaine.....                   | 9  |
| 2.Création d'hôte virtuel sous apache.....  | 10 |
| 3.Intégration de l'URL.....   | 11 |
| Chapitre 2. Développement de l'application.....   | 12 |
| I. Architecture Technique .....   | 12 |
| A. Plateformes et outils utilisés.....  | 12 |
| B. Mode choisit pour le développement d'application Facebook.....                       | 13 |
| II. Cas classique avec le php-sdk.....  | 13 |
| III. Utilisation de FOSFacebookBundle .....   | 14 |
| A. Téléchargement et Installation avec composer .....                                   | 14 |
| B. Mise en place et intégration du SDK JavaScript .....                                 | 16 |
| C. Ajout du bouton de connexion aux Templates.....                                      | 17 |
| D. Déclenchement de la connexion à l'application Symfony après connexion à Facebook.... | 18 |
| IV. Manipulation de l'API Facebook.....   | 19 |
| A. Avec php-sdk.....  | 19 |
| B. Avec le SDK de JavaScript.....   | 20 |
| C. Les permissions de l'application .....   | 21 |
| D. Access Token ( le jeton d'accès ) .....  | 22 |
| 1.Expiration de l'acces token et obtention d'access token à longue vie. ....            | 22 |
| E. Les requêtes sur les bases de données de Facebook .....                              | 23 |

|   |    |
|---|----|
| 2.Graph Api .....   | 24 |
| 1.FQL Query .....   | 28 |
| V. Architecture Fonctionnelle.....  | 30 |
| A. Approche architecture récupération de données depuis Facebook.....                     | 30 |
| B. Model conceptuel de l'application.....   | 31 |
| 1.Structure et Contenu du bundle crée.....  | 31 |
| 2.Les Entités composants les données d'un Groupe (Modèle Orienté Objet) .....             | 35 |
| 3.Schéma de la base de données .....  | 36 |
| 4.Les triggers pour la mise à jour automatique de certains données.....                   | 36 |
| VI. Importation des données d'un Groupe Facebook.....                                     | 38 |
| A. Importation des données d'indentification d'un groupe .....                            | 39 |
| B. Importation des membres du groupe .....  | 40 |
| 1.Principe.....   | 40 |
| C. Importation des postes du groupes .....  | 43 |
| 1.Limites de Graph Api.....   | 43 |
| 2.Limites de FQL .....  | 44 |
| 3.Importation de tous les postes.....   | 44 |
| 4.Importation selon la date de mise à jour, date considéré par défaut sur graph api ..... | 50 |
| 5.Importation selon la date de publication.....   | 51 |
| D. Importation des jaimés d'un postes.....  | 53 |
| E. Importation des commentaires d'un poste.....   | 54 |
| VII. Astuces et Techniques de mise à jour des données d'un groupe .....                   | 55 |
| A. Mise à jour des membres du groupe par de nouveaux inscrits.....                        | 55 |
| 1.Principe.....   | 55 |
| B. Mise à jour des postes du groupe.....  | 56 |
| 1.Principe.....   | 56 |
| C. Mise à jour des commentaires d'un postes existant. ....                                | 56 |
| 1.Principe.....   | 56 |
| D. Mise à jour des jaimés.....  | 57 |
| 1.Principe.....   | 57 |
| VIII. Déploiement de l'application.....   | 58 |
| Conclusion et perspective .....   | 60 |

# Introduction

---

De nos jours, le développement d'applications sur des plateformes web reste un des choses d'actualité et qui demandent un effort pour y arriver vu que les documentations pour leurs développements sont bien là mais pas trop simplistes à trouver la bonne sauce à nos besoins, vu qu'ils traitent beaucoup de données.

Ainsi, à travers ce document nous somme allé droit au but, en expliquant en long et en large comment créer une application qui permet de récupérer les données d'un groupe Facebook vers une base de données externe à Facebook.

Cet atelier traite le travail cité ci-dessus, en commençant depuis la création de l'application auprès de Facebook en initialisant les paramètres, et l'intégration de l'application à développer avec Facebook.

Par suit nous avons traité la partie du développement de l'application en utilisant Symfony 2 et les deux sdk de php et JavaScript. Nous avons expliqué comment développer l'application avec les sdk directement, mais on a utilisé un Bundle Spécifique nommé FOSFacebookBundle, pour nous faciliter la tâche, vous n'êtes pas obligé d'utiliser les bundle pour travailler.

Nous avons traité en détails chaque requêtes de récupération de données, et des illustrations de code, et des schémas.

L'application développé possède une documentation du code simple à naviguer pour comprendre ce qu'on a fait.

Vous pouvez retrouver le code source et la documentation sur un dépôt sur le site github.com nommé FbAppGroup <https://github.com/bilaalsoidik/FbAppGroup>

# Chapitre 1 : Création et Initialisation de l'application

## I. Inscription en tant que développeur(obligatoire)

Pour développer une application Facebook, il faut d'abord se rendre sur le lien <https://developers.facebook.com/> et avant tout s'enregistrer en tant que développeur Facebook comme l'illustre l'image de la **Figure 1**

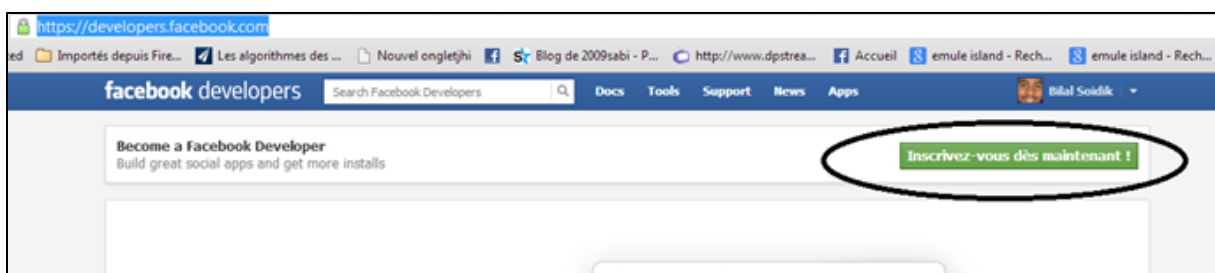


Figure 1 Accueil de la section développeur

Après avoir cliqué sur le bouton d'inscription en tant que développeur, il y'a quatre étapes à réaliser, comme l'illustre l'image de la Figure 2

Un ensemble de phase vous sont demandé pour finaliser votre enregistrement. Si vous êtes bien inscrit vous recevrez un mail de confirmation et vous recevrez les publication sur le blog développeur de Facebook.

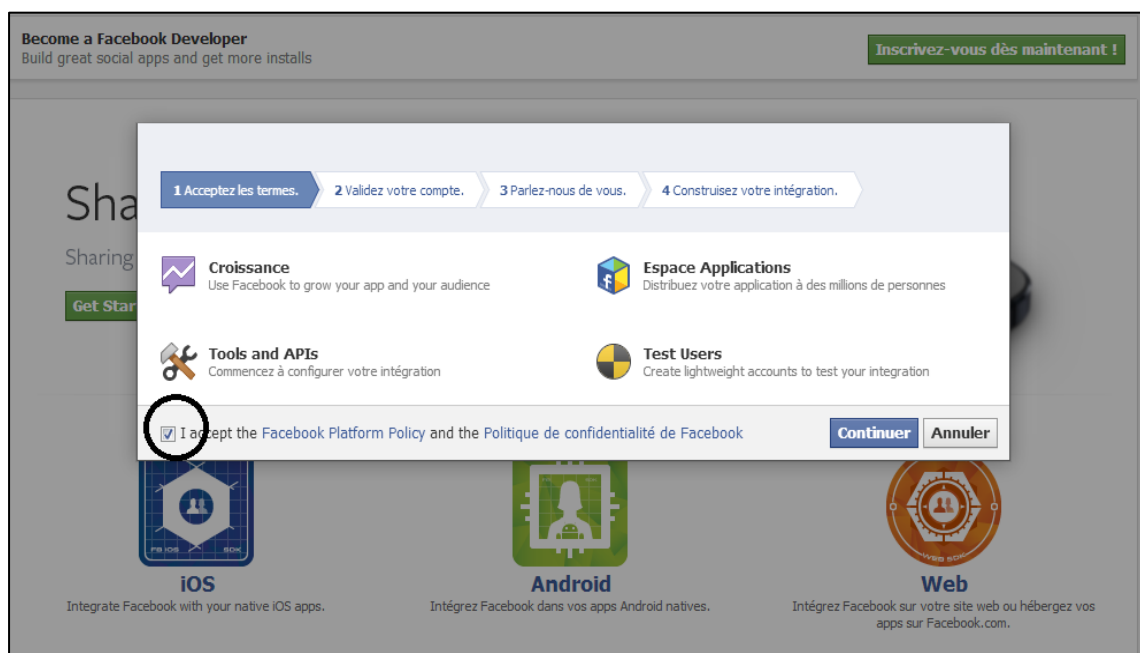


Figure 2 Inscription en tant que développeur

Un bouton de création d'application apparaît . Alors que si vous n'étiez pas inscrit le bouton n'allait pas apparaître.

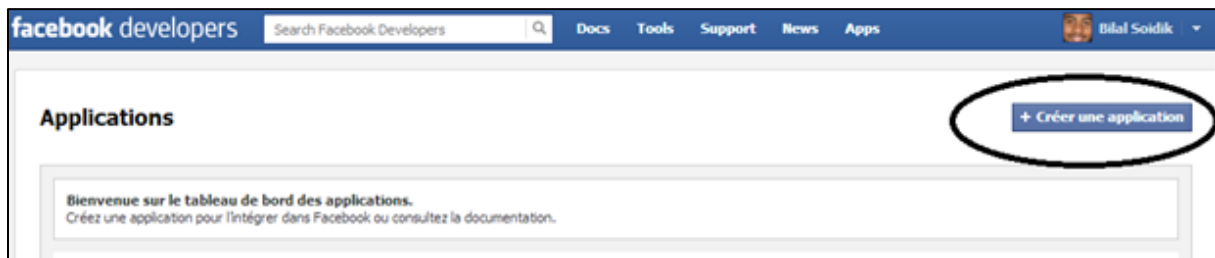


Figure 3 Création d'application



Il est important de noter qu'il y avait un bug dernièrement le lien. <https://developers.facebook.com/> vous redirigeait vers [www.facebook.com](http://www.facebook.com) je ne sais pas si c'était pour les inscrits en tant que développeur ou pas !

A noter que si vous êtes déjà inscrit en tant que développeur alors la rubrique développeur va s'ajouter à votre menu Facebook voir Figure 4. Vous aurez la liste des applications et la rubrique de création d'une nouvelle application.

## II. Création d'application

Ainsi, vous pouvez commencer la création de vos applications en vous rendant sur le même lien <https://developers.facebook.com/> . Ou en le créant sur le menu Facebook. Ou en vous rendant directement sous la rubrique apps <https://developers.facebook.com/apps>.

En cliquant sur créer une application que ça soit sur le menu, on vous demande de fournir des données d'identification de l'application voir Figure 5 dont le nom est le seul paramètre obligatoire ce dernier ne doit pas contenir des mots relatifs au

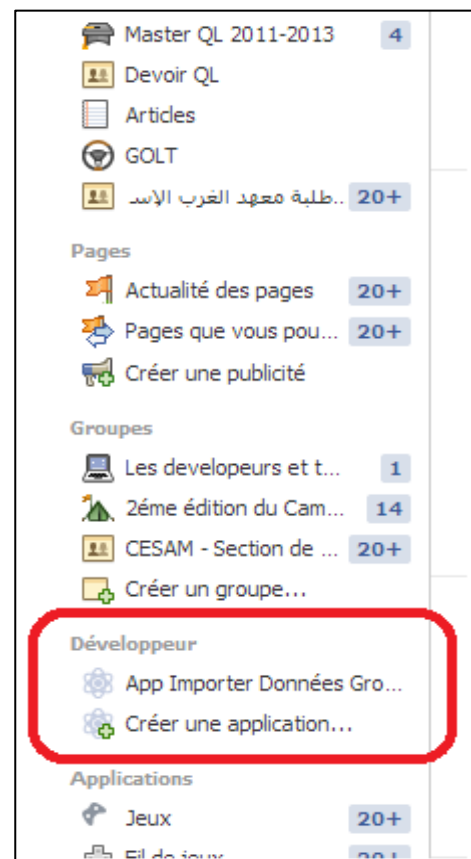


Figure 4 Rubrique développeur

terme « Facebook » ou connus comme désignant le réseau Facebook, même si le mot est concaténé avec un autre. Ainsi, les mots comme « Facebookapp », « FB\_monapp », « Face\_monapp » seraient rejetés . Ce nom aussi, ne doit pas être pris par une autre application. Sinon, le système affichera « echec ».

Notre application porte le nom de **App Importer Données Groupe** pour designer qu'il s'agit d'une application qui récupère les données d'une groupe Facebook. Les espaces et les accents sont acceptés car ce n'est pas le nom qu'on utilise pour s'identifier mais c'est l'id de l'application et la clé secrète.

Figure 5 Création d'application, information d'indentification

### III. Paramètres de l'application

#### A. Paramètres essentiels

Après avoir fourni les données (dans notre cas, nous avons juste fourni le nom et un nom de domaine, je vais détailler comment utiliser les nom d'hôtes associé aux adresses IP à la place des nom de domaines afin de travailler en local sans avoir à héberger l'application ailleurs), vous passez à l'étape suivante. Sur la page de la Figure 6, on nous affiche l'identifiant de notre application et la clé secrète, qui sont obligatoires à fournir le moment où notre application veut s'initialiser et s'identifier auprès du serveur Facebook et après l'indentification l'accès aux données se fait avec une clé, un jeton d'accès qui s'appelle `access_token` en anglais et terme utilisé dans l'API. Nous allons détailler le concept de l'`access_token` par la suite.

The screenshot shows the Facebook Developers interface for configuring an application. The left sidebar lists various settings categories. The main content area is titled 'Applications > App Importer Données Groupe > Essentiel'. It displays the app's ID and secret, a warning that the app is in Sandbox Mode, and fields for 'Display Name', 'Namespace', 'Contact Email', and 'App Domains'. There are also checkboxes for 'Website with Facebook Login', 'App on Facebook', and 'Mobile Web'.

Figure 6 Paramètres de l'application

Après cela vous devrez fournir quelques informations générales sur votre application, notamment : le nom du domaine de votre site web qui va devoir héberger l'application, l'adresse électronique, .... Nous nous allons travailler en local.

Par défaut, l'application est en mode « bac à sable » : ce signifie qu'elle est visible uniquement aux administrateurs, développeurs et testeurs. Dans le cas de sa désactivation, l'application devient accessible à tout le monde, à condition que l'application soit soumise et validé par Facebook, pour cela il faudra soumettre la page de détail de l'application, en tout cas cela entre dans la phase de déploiement de l'application, à voir en fin de ce rapport.

## B. Mode d'intégration de l'application à Facebook

En bas de page, nous devons choisir de 1 à 6 modes d'intégration de notre application à Facebook comme illustre la Figure 7, on peut choisir plusieurs à la fois. Alors comme vous voyez il y a en premier lieux Website With Facebook Login qui demande l'url de la page où le bouton de login à Facebook se trouve. Ce type d'intégration permet à l'application d'interagir avec les serveurs Facebook sans que l'utilisateur soit obligé d'être sur le site Facebook.com, il faut que l'utilisateur se logue et donne les permissions à l'application. On va revenir sur les permissions.



L'application peut faire des requêtes recevoir les données de même qu'une application intégré sur le site Facebook.

Namespace: [?]

Adresse électronique de contact: [?] bilalsoidik@outlook.com

App Domains: [?] fbgrpdonnees.com x

Mode bac à sable: [?] ☒ Activé ☐ Désactivé

Select how your app integrates with Facebook

☒ Website with Facebook Login x  
URL du site: [?] http://www.fbgrpdonnees.com/app\_dev.php

☒ App on Facebook Use my app inside Facebook.com.

☒ Mobile Web Bookmark my web app on Facebook mobile.

☒ Application iOS native Publish from my iOS app to Facebook.

☒ Application Android native Publish from my Android app to Facebook.

☒ Onglet Page Build a custom tab for Facebook Pages.

Enregistrer les modifications

Figure 7. Différents modes d'intégration de l'application à Facebook

Le 2<sup>e</sup> mode c'est le mode App on Facebook, c'est une application qui s'intègre dans un canevas sur le site Facebook.com, ça ne sera pas un vrai canevas Html5 mais c'est un iframe qui montre une toute indépendance. Mais ce n'est pas possible d'utiliser directement ce mode surtout pour les tests et le développement car Facebook vous donnera accès au canevas une fois l'application publié. Donc le mieux c'est de développer dans un site à part et puis une fois l'application soumise et validé vous aurez accès au canevas sur Facebook, en tout cas il n'y a pas de différence avec les autres modes en matière de données et permissions. Il y a plusieurs modes, notamment pour les applications mobiles, et ainsi de suite.

Notre application nous avons utilisés le mode de **site web avec connexion à Facebook**.

## C. Intégration avec le mode Website With Facebook Login

### 1. L'URL du site

A souligner que cet url est obligatoire, et sachez bien que même si vous avez la clé secrète et l'id de l'application et que vous voulez faire une requête a à Facebook pour identifier l'application alors si Facebook détecte une page de votre site autre que celle provenant de cet URL et que vous voulez vous loguer depuis cette page alors vous aurez une erreur et vous ne pourrez jamais vous identifier.

**L'URL doit impérativement correspondre à votre site où il y a le bouton de connexion ou le Toile URL, ou du domaine qui est un sous-domaine des domaines fournis.**

Sinon vous aurez le message d'erreur suivant :

Given URL is not allowed by the Application configuration.: Un ou plusieurs des URL donnée n'est pas autorisé par les paramètres de l'application. Il doit correspondre au Site Web ou Toile URL ou du domaine doit être un sous-domaine d'un des domaines de l'application.

## D. Création d'URL pour site local avec hôte virtuel sous apache

Dans notre exemple nous avons d'abord créer un projet Symfony 2 sous le nom **FbAppGroup** et l'avons inclut sur le répertoire des projets php dans le serveur web. Et avons choisit [fbgrpdonnees.com](http://fbgrpdonnees.com) comme nom de domaine, ça dépendra de vous si vous utilisez Wamp ou EasyPhp. Pour pouvoir créer un URL comme dans notre exemple, il faut créer un hôte virtuel qui point sur le répertoire `_CHEMIN_ABSOLUE_/FbAppGroup/web/` de notre projet Symfony 2 et lui donner un nom de serveur. En suite faire pointer l'adresse IP de notre machine vers ce nom de domaine dans le fichier hosts. Pour plus de détails en cas de besoins voici les étapes :

### 1. Configuration de Windows et Linux pour créer un nom de domaine

Sous Windows, on veut qu'il se comporte en DNS, alors il faut chercher le fichier hosts qui se trouve sous dossier etc sous le chemin suivant `C:\WINDOWS\system32\drivers\etc\hosts` .

En parcourant vous devez trouver quelque chose comme ça :

```
127.0.0.1      localhost
::1           localhost
```

Ajouter les lignes suivantes selon votre choix de nom de domaine :

```
127.0.0.1      fbgrpdonnees.com
127.0.0.1      www.fbgrpdonnees.com
```

Enregistrer et Fermer

Remarque :

Si vous ne pouvez pas éditer le fichier, vérifiez que celui-ci n'est pas en lecture seule.

Sous Vista, vous risquez de rencontrer encore plus de problèmes avec l'UAC. Faites une copie du fichier sur votre bureau, mettez les droits d'écriture, faites la modification, remettez le fichier en lecture seule, et écrasez le fichier original.

Sous Linux il faut taper dans un terminal la commande suivante pour éditer le fichiers :

```
sudo gedit /etc/hosts
```

Et il faut saisir les données comme dans le cas de windows.

## 2. Création d'hôte virtuel sous apache

- Sous EasyPhp qu'on a utilisé, lorsque celui-ci est lancé, cliquez sur l'icône dans la barre des tâches, cliquez sur "Apache" et vous devriez voir dans les entrées du menu déroulant un fichier le sous menu configuration. Ouvrez ce fichier et nous allons faire les modifications nécessaires.

Ou bien si EasyPhp est installé sous la racine du disque C il faut ouvrir le fichier httpd.conf se trouvant sous :

```
C:\EasyPHP-DevServer-X.X.XX\binaries\apache\conf
```

Il faut dé-commenter la ligne se trouvant juste après

```
# Virtual hosts
```

```
Include conf/extra/httpd-vhosts.conf
```

Après ouvrir le fichier `httpd-vhosts.conf` se trouvant sous

```
C:\EasyPHP-DevServer-X.X.XX\binaries\apache\conf\extra\
```

Ajouter les lignes suivantes :

```
<VirtualHost *:80>
    DocumentRoot "C:/EasyPHP-DevServer-X.X.X/data/localweb"
    ServerName localhost
</VirtualHost>
<VirtualHost *:80>
    DocumentRoot "C:/EasyPHP-DevServer-
X.X.XX/data/localweb/FbAppGroup/web"
    ServerName fbgrpdonnees.com
    ServerAlias www.fbgrpdonnees.com
</VirtualHost>
```

Enregistrer et Redémarrez le serveur.

Pour Linux chercher le fichier `httpd.conf` et ajouter les ligne tel qu'on a ajouté pour Wamp

A présent si vous taper [www.fbgrpdonnees.com/app\\_dev.php](http://www.fbgrpdonnees.com/app_dev.php) vous allez vous retrouver l'index de votre projet Symfony 2.

### 3. Intégration de l'URL

Et comme nous allons intégrer le bouton de connexion sur la route racine / qui pointe sur `app_dev.php/` alors il faut indiquer le lien [http://www.fbgrpdonnees.com/app\\_dev.php](http://www.fbgrpdonnees.com/app_dev.php) sur le champ URL du site sur les paramètres de l'application Facebook comme sur la Figure 7.

# Chapitre 2. Développement de l'application

## I. Architecture Technique

### A. Plateformes et outils utilisés

- OS : Windows XP, mais parfois l'application est consulté depuis une machine du réseau local sous ubuntu 13.04. Alors l'url fut configuré pour écouter en local, mais là pour celui qui veut faire ça, il doit ajouter l'IP de la machine serveur et distant dans la liste des hôtes autorisés par Symfony dans le fichier `app_dev.php`
- Plateforme EasyPHP 13.1VC9 avec php 5.4 et Apache 2.2 [www.easyphp.org](http://www.easyphp.org)
- Symfony 2.3.2, utilisant twig comme moteur de template et des fichiers.yml pour la configuration.
- Facebook PHP-SDK, mais on a utilisé FOSFacebookBundle qui inclut le SDK toute fois on a montré comment utiliser le SDK sans ce bundle.
- Facebook JavaScript SDK, dont l'utilisation est facilitée avec FOSFacebookBundle
- JQuery 1.9.1 ;
- JQuery UI 1.10.3 version customisée ;
- Le plugin jquery.cookie.js pour faciliter l'utilisation des cookies avec JQuery à télécharger ici <https://github.com/carhartl/jquery-cookie> et l'insérer dans toute page où on veut utiliser les cookies.
- Serialiser 2.4dev de Symfony (optionnel) : pour faciliter l'encodage des objets au format JSON pour envoyer au client, il suffit d'ajouter `"symfony/serializer": "2.4.*@dev"` au require dans le fichier `composer.json` et faire un update.
- ApiGen pour la génération de la documentation professionnelle du code source.

## B. Mode choisit pour le développement d'application Facebook

Dans ce projet, on a créé notre propre bundle pour gérer l'application Facebook. Dans notre cas il était possible d'utiliser classiquement le php-sdk de Facebook directement sans bundle spécifique, mais le projet ne serait pas bien structuré et simple à utiliser contrairement à la manière dont on va le voir avec le bundle FOSFacebookBundle, surtout sur la gestion de session et la sécurité.

## II. Cas classique avec le php-sdk

Dans le cas classique directe sans un bundle spécifique et sans trop de configuration il faut télécharger le sdk depuis le lien <https://github.com/facebook/php-sdk/> et le déziper dans le dossier monProjet/vendor/ et puis dans le dossier app à l'intérieur du fichier autoload.php il faut ajouter ceci à la fin :

```
require_once __DIR__ .  
'/ ../vendor/facebook/src/facebook.php' ;
```

Et puis à l'intérieur d'un contrôleur il faut commencer à coder en appelant un objet Facebook comme ceci :

```
$facebook = new \Facebook(array(  
    'appId' => 486996558052473,  
    'secret' => 573060401ac4899c156f4aacbc683d74,  
));  
  
//Recupération de l'id de l'utilisateur courant  
//ça retourne 0 s'il n'est pas connecté  
$user = $facebook->getUser();
```

### IMPORTANT !

Pour la suite sur comment utiliser l'api Facebook via l'objet \$facebook pour faire les requête sur le serveur Facebook ça sera la même chose qu'avec le bundle, seulement on aura l'objet \$facebook facilement on aura pas à fournir chaque fois les données de paramètre et la session se gère automatiquement. Alors si vous ne voulez

pas utiliser le bundle, vous allez voir la partie après installation de FOSFacebookBundle.

### III. Utilisation de FOSFacebookBundle

FOSFacebookBundle est un Bundle développé par la communauté Friends Of Symfony 2 qui ont développé pas mal de bundle pour faciliter l'utilisation de Symfony.

Il est important de souligner que le FOSFacebookBundle intègre le PHP-SDK officiel de Facebook et le SDK de JavaScript pour structurer l'application à la fois côté client et serveur.

#### A. Téléchargement et Installation avec composer

Nous allons utiliser de façon basique le bundle, nous n'allons pas touché les aspects avancés, qui ne font pas l'objet du travail. A présent les étapes d'installation :

1. Ajoutons la ligne suivante dans le fichier `composer.json` dans la liste des requies du projet :

```
{
  "require": {
    "friendsofsymfony/facebook-bundle": "1.2.*"
  }
}
```

2. Exécutons la commande qui va suivre pour télécharger et installer le bundle depuis un dépôt sur [gintub.com](https://github.com). **Attention !** Il faut s'assurer que composer est bien installé et intégré dans votre path ou bien `composer.phar` existe sur le même répertoire avec le fichier `composer.json` ou bien fournir un chemin absolu vers le fichiers `composer.phar`. Etant donné que le bundle à des dépendances alors selon les version ça peut créer des problèmes d'installation alors au lieu de faire un update spécifique pour le bundle j'ai fait un update global des vendors et tout s'est bien déroulé lancer la commande suivante :

```
$ php composer.phar update
```

Ou bien si vous avez inclut composer dans vos variables paramètres, après avoir poointé sur la racine du projet alors il faut juste faire :

```
$ composer update
```



La documentation indique de lancer la commande qui suit, mais avec les dépendance incompatibles ça peut échouer, donc vaut mieux tout mettre à jour, c'est pour ça que la commande précédant est meilleur que la suivante.

```
$ php composer.phar update
friendsofsymfony/facebook-bundle
```

Quand le téléchargement et l'installation finissent vous aurez deux répertoires dans vos vendor le répertoire `facebook/` contenant le php-sdk de Facebook et le répertoire `friendsofsymfony/` contenant le FOSFacebookBundle.

3. Ajouter le bundle à l'applicationKernel sous le fichier `app/AppKernel.php`

```
public function registerBundles()
{
    return array(
        // ...
        new FOS\FacebookBundle\FOSFacebookBundle(),
        // ...
    );
}
```

4. Ajouter les lignes suivantes qui ne sont autre que des routes dans le fichier `app/config/routing.yml` ou le fichier routing de votre bundle et les faire pointer sur les action du contrôleur actuel, disons le contrôleur d'accueil par exemple , l'un va pointer sur l'action juste après que le login à Facebook à réussi et l'autre pour la déconnexion à Facebook. Alors c'est à vous de définir les actions et le contrôleur. Soit

```
_security_check:
    pattern:  /login_check
    default : .....
```



```
_security_logout:
  pattern:    /logout
  default :    .....
```

5. Configurer le service Facebook dans le fichier de configuration `app/config/config.yml`

```
fos_facebook:
  alias      : facebook
  app_id     : 486996558052473
  secret    : 573060401ac4899c156f4aacbc683d74
  cookie    : true
  permissions:[email,user_groups,friends_groups,read_stream]
```

**L'app\_id** doit correspondre l'id de l'application fourni par Facebook de même que **la clé secrète**.

Ce qui est déjà installé nous permet de travailler et atteindre nos objectifs, mais si vos configurations son en xml ou que vous voulez mettre en place la sécurité dans votre application alors referez-vous à la documentation officielle du bundle dur github.com :

<https://github.com/FriendsOfSymfony/FOSFacebookBundle/blob/1.2.1/Resources/doc/1-basic-usage.md>

## B. Mise en place et intégration du SDK JavaScript

Le SDK JavaScript travail avec le Bundle côté serveur pour permettre les interactions côté client et initialisation notamment la connexion, l'écoute de statut de la session et la mise en place des cookies que le SDK côté serveur lit directement sans l'intervention du développeur.

Un Template d'aide est inclut pour charger le SDK de JavaScript et initialiser les paramètres des conteneurs de services.

Alors dans un de vos Templates, le Template initiale index par exemple ou le layout, il faut ajouter les directives suivants en fonction du type de votre Template twig ou php.

```
<!--A l'intérieur d'un template twig -->
```

```
{{facebook_initialize({'xfbml': true, 'fbAsyncInit': 'onFbInit()'
; ' }) }}
```

Ou bien :

```
<?php // A l'intérieur d'un template php?>
<?php echo $view['facebook']->initialize(array('xfbml' =>
true, 'status' => true , 'fbAsyncInit' => 'onFbInit();')) ?>
```

Personnellement j'ai utilisé un Template twig et j'ai intégré cela sur mon `layout.html.twig` qui se charge avec tous les Templates.

Notez que `fbAsyncInit` est un paramètre pour vous aider à exécuter du code JavaScript après la fonction initialisation de façon asynchrone, juste après l'appel de `FB.init({...});`.

`onFbInit()` est une fonction JavaScript à définir dans nos script pour nous permettre d'écouter de façon asynchrone l'initialisation de l'objet `FB` qui est l'objet principal de **sdk JavaScript** de Facebook qui permet d'interagir avec le serveur Facebook. Alors quand l'objet `FB` s'initialise on met le code à exécuter tout juste après dans `onFbInit()`. Il est le moment de l'appel de `facebook_initialize`. Il change de statut en fonction de l'état de la session, avant login, après login, logout, ou rafraichissement.... On va découvrir ses méthodes au cours du document.

Si vous allez ajouter la balisage XFBML à votre site, vous pouvez également déclarer l'espace, peut-être dans la balise HTML d'ouverture:

```
<html xmlns:fb="http://www.facebook.com/2008/fbml">
```

## C. Ajout du bouton de connexion aux Templates

Ajouter ceci à un des Templates en fonction du type de Templates :

```
{# A l'intérieur d'un template twig #}
{{ facebook_login_button({'autologoutlink': true,
'label': 'Se connecter', width:130, 'size': 'xlarge'}) }}
```

Ou bien

```
<?php // A l'intérieur d'un template en php ?>
```

```
<?php echo $view['facebook']->loginButton(
array('autologoutlink'=> true) ) ?>
```

On fixe autologoutlink à true, le bouton de logout va s'afficher si la session est en cours est en cours et l'application s'est déjà autorisé auprès de Facebook.

On peut personnaliser le bouton de connexion en fixant les données des paramètres suivant :

- Label: Le texte qui apparaît dans le bouton.
- showFaces: Indique si vous souhaitez afficher les photos de profils sous le bouton Connexion une fois l'utilisateur connecté.
- Largeur: La largeur du plugin en pixels. Largeur par défaut: 200px.
- maxRows: Le nombre maximal de lignes de photos de profil à afficher. Valeur par défaut: 1.
- scope: Une liste séparée par des virgules de droits étendus.
- registrationUrl: Page d'inscription url. Si l'utilisateur ne s'est pas inscrit pour votre site, ils seront redirigés vers l'URL que vous spécifiez dans le paramètre registrationUrl.
- size: Différents boutons de taille: petit, xlarge moyen, grand, (par défaut: moyen).
- OnLogin: Définir une URL pour être redirigé après la connexion réussie

## D. Déclenchement de la connexion à l'application Symfony après connexion à Facebook.

Il est important de noter dans cet approche nous pouvons nous connecter et nous connecter à Facebook et se déconnecter mais la connexion à notre application Symfony doit être déclencher après la connexion(déconnexion) à Facebook.

Pour faire cela il va falloir s'enregistrer à l'évènement `auth.statusChange` de FB et après rediriger l'utilisateur au bon url, `_security_check` après statut de login et `_security_logout` après statut de logout comme on les a défini. Alors ajoutez les scripts JavaScript suivants à l'initialisation de du SDK JavaScript que le listener d'évènement `onFBinit()` soit bien déclenché.

```
<script>
function goLogIn(){
```

```

        window.location.href = "{{ path('_security_check')
    }}";
    }
    function onFbInit() {
        if (typeof(FB) != 'undefined' && FB != null ) {
            FB.Event.subscribe('auth.statusChange', function (
            response ) {
                if (response.session || response.authResponse) {
                    setTimeout(goLogIn, 500);
                }
                else {
                    window.location.href = "{{ path('_security_logout') }}";
                }
            }
        }
    }
</script>

```

Nous attendons 500ms avant de rediriger l'utilisateur, ceci afin de laisser le navigateur s'occuper de la cookie Facebook. Vous pouvez éviter cette étape, mais vous pourriez obtenir ce message d'erreur:

"The Facebook user could not be retrieved from the session."

## IV. Manipulation de l'API Facebook

### A. Avec php-sdk

On avait déjà montré comment récupérer l'api Facebook de façon classique en utilisant directement le sdk de php. Alors avec FOSFacebookBundle pour récupérer l'api à l'intérieur d'un contrôleur quelconque de notre projet c'est très simple pas besoin d'insérer des paramètres tout se fait automatiquement, il suffit d'écrire :

```
$facebook = $this->get('fos_facebook.api');
```

Et c'est partie vous pouvez faire ce que vous voulez 😊 !!!!!!!!!!!

Exemple :

```
$userid=$facebook->getUser()
```

Si vous êtes connecté vous aurez l'id de l'utilisateur courant sinon ça retourne 0. Et avec ça vous pouvez tester sur l'état de la session.

Exemple de requête :

```
$user=$facebook->api('/me') ;  
$id=$usr['id'] ;  
$nom=$usr['name']
```

`'/me'` est une requête sur Graph API, ça retourne les données de l'utilisateur en cours, on va détailler les requêtes graphapi et FQL.

Avec FOSFacebookBundle, `$facebook` est une instance de la classe `FacebookSessionPersistence` qui hérite de la classe abstraite `BaseFacebook` de php-sdk comme avec la classe `Facebook` du php-sdk qui hérite lui aussi de `BaseFacebook`

`FacebookSessionPersistence` et `Facebook` n'ont pas de différence sur les méthodes ils en ont tous les deux les 22 méthodes de la classe `BaseFacebook` seulement `FacebookSessionPersistence` le gère très bien en faisant persister la session et dès que vous rappelez quelque part avec :

`$this->get('fos_facebook.api')` dans vos contrôleurs, alors chaque action qui l'appelle c'est comme si récupère un objet session et fait un démarrage de la session pour la page courante. Mais pour ajouter des attributs à la session il faut l'utiliser la méthode classique de Symfony 2 en appelant `$this->get('session')`.

## B. Avec le SDK de JavaScript

Côté on peut manipuler l'API Facebook en utilisant l'objet FB du SDK JavaScript, et avec les requêtes on nous retourne des objets formatés JSON déjà parsés, mais la réponse on la traite sur la fonction callback une fois la réponse obtenue

Exemple de requête :

```
FB.api('/me', function(reponse){  
    if (!reponse || reponse.error) {
```

```

    console.log("Message d'erreur "+reponse.error.message)}
else{
    alert(" votre nom  :" + reponse.name+
        " votre email :"+ reponse.email) ;


    }
}) ;

```

## C. Les permissions de l'application

Avant de passer aux requêtes, on explique les permissions qu'on a définies sur les paramètres du service Facebook. Les permissions sont l'ensemble de droits de lecture ou d'écriture dont votre application demande à l'utilisateur pour pouvoir interagir avec ses données. L'autorisation se fait au premier moment de l'utilisation de l'application. Par défaut l'application demandera à l'utilisateur la permission d'accès au profil **public** même si vous ne l'avez pas précisé.

Pour la liste exhaustive de toutes les permissions rendez-vous sur le lien <https://developers.facebook.com/tools/> et sous le lien [Graph API Explorer](#) c'est l'explorateur du Graph API dont on va expliquer son fonctionnement.

En cliquant sur  vous allez pouvoir voir la liste des permissions possibles pour les applications, il y a les permissions sur les données de l'utilisateur et les données des amis et les permissions étendues comme illustre la Figure 8.

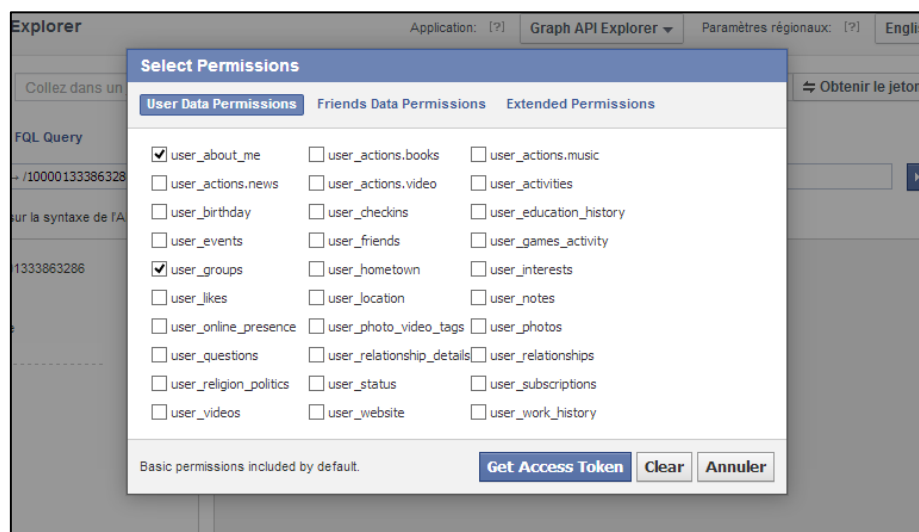


Figure 8 Sélection de permission et obtention d'access\_token

Notre application utilise les permission suivantes :

```
[email, user_groups, friends_groups, read_stream]
```

C'est à dire on demande les permission de :

- lire l'email de l'utilisateur en cours
- avoir accès aux groupes de l'utilisateur
- lire les groupes des amis de l'utilisateur
- lire la file d'actualité de l'utilisateur y compris ceux publié sur ses groupes

`read_stream` est un des extended permission

## D. Access Token ( le jeton d'accès )

C'est une clé que Facebook génère en fonction des permissions que votre application a reçu, une fois l'utilisateur est connecté et autorisé. Ceci conformément au protocole standard sécuritaire OAuth pour l'interaction des applications avec d'autres plateformes de développement sur internet comme les GoogleApps, et FacebookApps et autres.

L'`access_token` est stocké dans les cookies et le sdk côté serveur le lit automatiquement s'il veut faire des requêtes sur Facebook.

### 1. Expiration de l'access token et obtention d'access token à longue vie.

Un `access_token` expire dans 2heures si l'utilisateur n'a pas demandé un autre avant ce délais, à chaque demande de nouveau l'ancien expire. En cas d'expiration vous n'allez pas pouvoir exécuter des requêtes depuis votre serveur sur Facebook et vous aurez une `FacebookApiException` donc il vous faut un `access_token` active.

Côté client en JavaScript FB lui-même il peut se rafraîchir pour obtenir un nouveau `access_token`. Pour obtenir un jeton d'accès à longue durée il faut exécuter la méthode `$facebook->setExtendedAccessToken()` sur le serveur pour obtenir un `access_token` à durée de 60 jours. Il faudra remplacer l'ancien stocké sur les cookies par le nouveau, pour que si vous exécutez une requête côté client avec FB vous allez lui fournir l'`access_token` vous n'allez pas lui laisser se rafraîchir s'il trouve que ce qu'il a a expiré car dès qu'il demande un autre même ce de 60 jour va expirer. On a exécuter ceci dans notre travail.

```
$facebook->setExtendedAccessToken() ;  
$access_token=$facebook->getAccessToken();  
    $temp=60*24*3600;  
    setcookie('accessToken',$access_token,time()+$temp);
```

Et si on perd la session ou ce de 60 jours expire, pour une raison ou que le délais passe?

Là d'abord il faut s'assurer quand vous exécuter une requête sur Facebook d'avoir catcher l'exception `FacebookApiException` et donc en cas d'expiration de l'access\_token il faut rediriger l'utilisateur vers l'URL de connexion.

Exemple :

```
try {  
    //Possible que l'access_token à expiré  
    //Ou qu'il n y a pas de session  
    $result = $facebook->api($requête);  
  
    } catch (\FacebookApiException $e){  
        return $this->redirect($facebook->getLoginUrl());  
    }  
}
```

Il va faire une double redirection pour chaque cas suivant :

- Si l'utilisateur est connecté sur Facebook, l'application va se rafraîchir et récupérer un nouveau access\_token et retourner à la même page où il était.
- S'il n'y avait pas de session c'est simple il va afficher directement la page de connexion à Facebook. Et une fois connecté il va rediriger l'utilisateur vers la même page où l'exception est levé, là où il était.

## E. Les requêtes sur les bases de données de Facebook

Pour pouvoir interroger les bases de données de Facebook vous avez deux méthodes à des syntaxe différents. Mais d'après nos découvertes on ne peut se passer d'un deux 100% , si on cherche un maximum d'information, parfois l'un est mieux que l'autre. Ces deux méthode sont les requête de Graph Api qui est nouveau et les requête FQL(Facebook Query Language). On a utilisé les méthodes.



## 2. Graph Api

Le Graph Api L'API est le principal moyen d'obtenir des données à l'intérieur et en dehors du graphe social de Facebook. Il s'agit d'une API HTTP de bas niveau que vous pouvez utiliser pour interroger les données, créer de nouvelles histoires, télécharger des photos et une variété d'autres tâches que l'application peut avoir besoin de faire.

### a. Principe

Le principe est basé sur la théorie des graphes, chaque chose sur Facebook est considéré comme un objet, avec ces caractéristiques. Alors chaque objet est vu comme un nœud d'un graphe avec son id qui a des connections avec d'autres nœuds. Donc à partir d'un nœud on peut accéder aux nœuds connections qui sont aussi des nœuds avec leurs connections.

### b. Graphe Api Explorer

Pour faciliter à formuler et visualiser nos requêtes et la structure des réponses retournés avant de les utiliser dans nos codes, Facebook a mis en place l'outil explorateur de graph, qui peut d'exécuter les requêtes. Allons y au lien <https://developers.facebook.com/tools/explorer>

Il faut demander un jeton d'accès selon les données que vous voulez consulter En entrant il nous donne la première requête :

```
/100001951987264?fields=id,name
```

Chaque objet on peut on a accès avec son id ici c'est mon id et après le point d'interrogation, on spécifie les champs qu'on veut, si on écrit seulement :

```
/100001951987264
```

Ça affichera tous les données de mon profil que l'application a droit de voir, c'est comme si on fait

```
/me
```

Les requêtes sont très simple pour un nœud de `id='id'` à sélection pour spécifier les champs alors on fait `?fields=` et on classe les champs linéairement

séparés par des virgules. Mais si on veut sélectionner un sous champ contenu dans un objet si la requête contient un objet, alors on fait

```
/id?fields=champ1,champ2,champ3Objet.fields(souschamp1,souschamp2)
```

Si on veut sélectionner des champs spécifiques pour une connexion, qui est en normalement une liste d'objet alors on fait la même chose.

Exemple la liste des groupes en sélectionnant l'id, le nom et la privatisation (fermé, ouvert, ou secret) :

```
/me?fields=id,name,groups.fields(id,name,privacy)
```

Exemple sur la liste membres d'un groupe dont l'id est 'id\_groupe' on fait :

```
/id_groupe?fields=id,members.fields(id,name)
```

On peut aussi faire comme ça pour l'arborescente, que ça soit objet ou liste :

```
/id?fields=champ1,listObjets.fields(souschamp1,sousObjet.fields(ssch1)..)..
```

Pour les listes on peut appliquer les filtres comme on veut, l'ordre n'est pas important on peut faire, par exemple pour sélectionner seulement les 10 derniers membres inscrits d'un groupe on fait :

```
/id_groupe?fields=id,members.fields(id,name).limit(10)
```

Ou faire

```
/id_groupe?fields=id,members.limit(10).fields(id,name)
```



Facebook filtre les données dont vous avez le droit de voir alors s'il arrive qu'une donnée est masquée pour vous alors la limite peut vous retourner moins que la limite fixée. **C'est pour cela qu'il ne faut jamais utiliser la limit comme critère d'arrêt d'une boucle car vous risquez d'avoir des erreurs, il faut utiliser un foreach vous prenez seulement le contenu.**

Autre exemple si vous voulez sélectionner des membres à partir d'un utilisateur donné de id=123456 on fait :

```
/id_groupe?fields=id,members.__after_id(123456).limit(10).fields(id,name)
```

Ou à partir d'un offset, depuis par exemple la 60<sup>e</sup> ligne vers 10 membres après :

```
/id_groupe?fields=id,members.limit(10).offset(60).fields(id,name)
```

Il est notamment possible de faire requêtes plus simple de la manière suivante pour les connections

```
{object_id}/connection?fields=champ1,champ2
```

Par exemple pour les membres on peut faire

```
{id_groupe}/members?fields=id,name
```

**Pour plus de détails sur les requêtes** à utiliser pour importer les données d'un groupe il faut se référer à la section Importation des données d'un Groupe Facebook, en suivant le lien [VI](#).

Sur le Graph Api Explorer par défaut c'est lui-même qui est une application qui est sélectionnée mais vous pouvez choisir une de vos application c'est l'occasion de voir si les permissions choisis vous retournent réellement ce que vous voulez.

Pour formuler la requête graphiquement et choisir les champs il faut cliquer sur le bouton + comme sur la **Erreur ! Source du renvoi introuvable.** Il y a deux compartiment en haut les **fields** (champs) les champs de l'objet et après les champs les **connections** il faut juste défiler

Quand vous sélectionnez encore un objet parmi les connections entre eux aussi il y a le + pour choisir les sous champs, comme la sélection de la liste des groupe de l'utilisateur Figure 10.

Il y le champs **limit** pour limiter le nombre par requête car un grand nombre de données risque de prendre trop de temps avant de répondre.

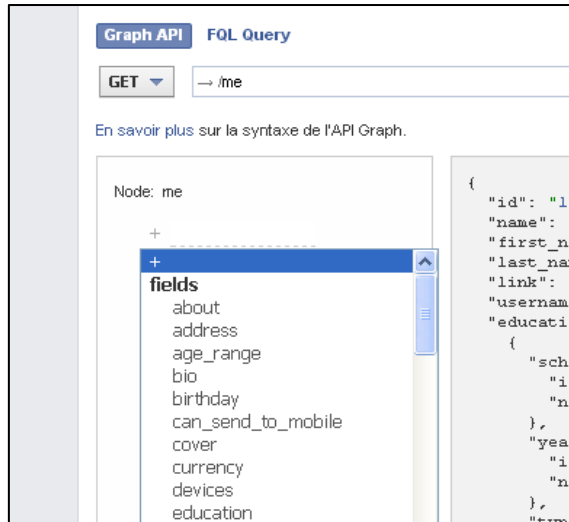


Figure 10 Choix d'un champs sur le graphe api

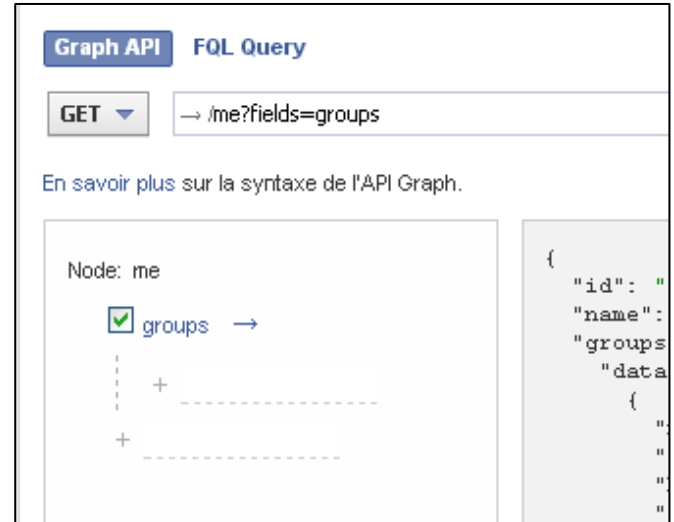


Figure 9 Liste de groupe de l'utilisateur

Chaque objet a son id qui est un lien vers lui et ses champs et ses connexion, par exemple si on clique sur l'id d'un groupe parmi les groupes de l'utilisateur on aura les champs possible pour groupe comme sur la Figure 11.

Pour un groupe nous avons les connexions les **docs** pour désigner les documents, **feed** pour désigner les postes publié, eux aussi ont des connexion vers les commentaires et les likes (j'aimes), **members** pour les membres du groupe, **picture** pour les photos.

On va vous montrer les requêtes qu'on a utilisé sur notre application.

**Pour exécuter une requête** de Graph Api il faut la passer tel quel en paramètre au fonction `api()` en ajoutant le slash(/) au début et récupérer les données par clé étant données qu'il sont retourné en tableau associatif.

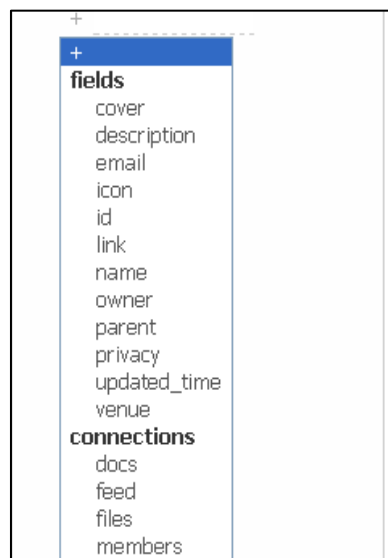


Figure 11 Champs et connexions d'un groupe

## 1. FQL Query

FQL ( Facebook Query Language) c'est le langage de requête sur la base de données de Facebook qui se structure comme SQL. Avec l'outil Graph API Explorer vous pouvez exécuter vos requêtes FQL avant de les utiliser dans votre application. Il suffit de basculer vers FQL Query. L'outil vous offre une autocomplétion.



Figure 12 Exemple de fql query



Il faut savoir que les requêtes **FQL** et de **Graph Api** n'ont pas forcément les même nom de table ou de champs. Par exemple : dans **Graph Api** les postes sont désigné par **feed** or que sur **FQL** elles sont désigné par **stream**(flux). Un autre exemple : sur **FQL** l'id utilisateur sur la table **user** c'est **uid** et l'id du groupe sur la table **group** c'est **gid**, pour les post **post\_id**, alors dans **Graph Api** ils sont tous nommés **id** dans leurs objets correspondants. Pour cela, il faut s'assurer du vrai nom sur la référence en ligne.

Pour plus d'information sur les tables il faut visiter la référence de **FQL** <https://developers.facebook.com/docs/reference/fql/>

Pour lancer une requête **FQL** depuis l'application ce n'est pas la même chose qu'avec le graph api. Pour simplifier il suffit d'exécuter ceci :

```
$result = $facebook->api(array(
```

```
'method' => 'fql.query',
'query' => 'SELECT name,email FROM user WHERE uid=me()'
));
```

## IMPORTANT

Contrairement à **Graph Api** quand vous faites une requête avec **FQL** et que vous voyez la structure de la réponse sur l'explorateur alors la réponse qui est au format JSON contient un tableau de données comme ça :



```
{ data[
  "champ1" : "donnée1"
  "champ2" :{ "sousch2" : "donnée2" }
  .... ] }
```

Alors pour récupérer les données il ne faut pas faire :

```
$result['data']['champ1'] //ERREUR
$result['data']['champ1']['sousch2'] //ERREUR
```

Il faut faire directement

```
$result['champ1'] //CORRECTE
$result['champ1']['sousch2'] //CORRECTE
```

Avec le SDK de JavaScript c'est simple vous n'êtes pas obligé d'utiliser le tableau de paramètre, il suffit d'utiliser le graph Api pour exécuter une requête FQL:

```
FB.api("/fql?q={SELECT name,email FROM user WHERE uid=me()}" ,
function(reponse) {
...
} ) ;
```

## V. Architecture Fonctionnelle

Alors comme vous le savez c'est une application Symfony 2 son architecture est connu, c'est le modèle MVC mais ici nous n'allons pas parlé de l'architecture de Symfony mais de la manière dont on a conçu l'application. Alors notre application nous permet :

- Se connecter à et se déconnecter de Facebook.
- Choisir et enregistrer un groupe dont on veut récupérer les données
- Récupérer les membres d'un groupe ;
- Récupérer les postes d'un groupe ;
- Récupérer a liste des commentaire pour chaque postes associés aux personnes qui ont commentés et la liste des jaimés pour chaque postes

### A. Approche architecture récupération de données depuis Facebook

Nous avons adopté deux architectures différentes d'importation de données.

#### a. Première approche architectural

La première qui est plus complexe c'est ce qu'on a fait pour importer un groupe.

Alors le principe c'est qu'on exécute une requête sur le serveur Facebook depuis notre client en utilisant l'objet FB du sdk JavaScript, on récupère une liste données, l'utilisateur choisit l'objet dont il veut et on récupère automatiquement un formulaire depuis notre serveur pour remplir les données. L'utilisateur lui-même clique sur le bouton enregistrer.

#### b. Deuxième approche architectural

Approche très simple principalement utilisé pour les grand nombre de données : on envoie des données clés de notre requête à un action spécifique sur notre serveur, lui il formule la(es) requête(s), il sollicite le serveur Facebook, si tout marche bien il récupère les données, et il les enregistre en base de données voir. On a tout traité en développant un unique Bundle.

## B.Model conceptuel de l'application

### 1. Structure et Contenu du bundle crée.

En bref nous avons un seul bundle FBGroupeBundle, contenant 2 contrôleurs structuré en esquisse comme illustre la **Erreur ! Source du renvoi introuvable..**

Figure 13 Structure du bundle

#### Les contrôleurs créés

Les actions de nos deux contrôleurs cité en haut sont ceux définis par la Figure 14.

Comme vous voyez pour sur la figure les méthodes qui sont **public** portent un petit icone vert, mais les méthodes privés non. Nous avons des variables privés pour le contrôleur qu'on pas généré sur le diagramme.

Pour plus d'information il faut consulter la documentation du code sous le dossier **Documentation du code**. Nous avons bien commenté notre code.



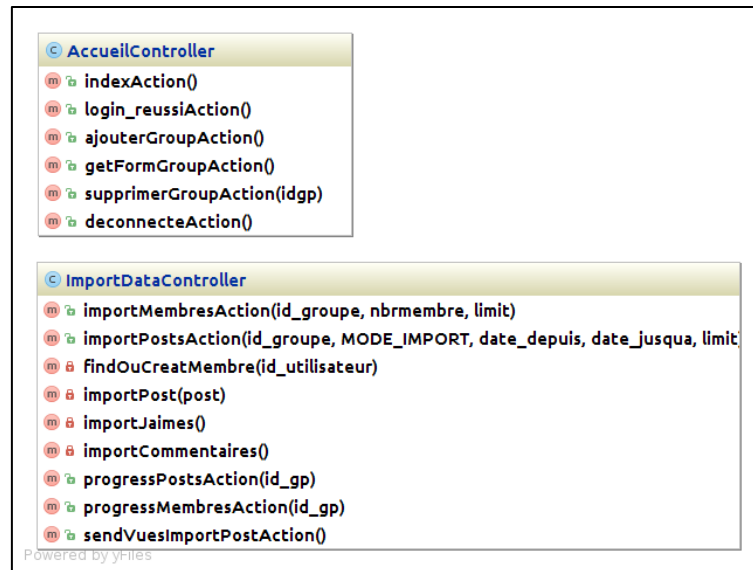


Figure 14 Diagramme de classe des contrôleurs

La documentation du code emporte le code aussi avec elle, bien structurée, vous pouvez consulter facilement, donc chaque méthode ou class du répertoire src du projet, il y a un lien vers la page correspondant sur la documentation. Tout ce qui est en bleu c'est que c'est un lien hypertexte vers son index ou son code source. Voyons pour le contrôleur ImportDataController la documentation va vous afficher ceci :

### Extrait de la documentation du code

## Class ImportDataController

Symfony/Bundle/FrameworkBundle/Controller/Controller

**FB/groupeBundle/Controller/ImportDataController**

Namespace: [FB/groupeBundle/Controller](#)

Located at [FB/groupeBundle/Controller/ImportDataController.php](#)

### Sommaire des méthodes

|   |   |
|---|---|
| <p><b>public</b></p> <p>Symfony/Component/HttpFoundation/Response</p> | <p><a href="#">importMembresAction</a>( integer <i>\$id_groupe</i>,integer <i>\$nbrmembre</i>, integer <i>\$limit</i> = 40 )</p> <p>L'action qui permet de récupérer les membres d'un groupe Facebook vers notre base de données, le nombre de membre le récupère un tableau des id et on prend la longueur du tableau, ceci côté client avec FB.</p> |
|---|---|

|   |  |
|---|--|
| public<br>Symfony/Component/HttpFoundation/Response | <code>importPostsAction( integer <b>\$id_groupe</b>,<br/>const <b>\$MODE_IMPORT</b>, string <b>\$date_depuis</b> = null,<br/>integer <b>\$date_jusqua</b> = null, integer <b>\$limit</b> = 25 )</code><br>L'action qui permet de récupérer les postes vers notre base données selon différents type de mode 'importation Le mode d'importation nous permettra de connaitre la requête à exécuter |
| private <b>FB/groupeBundle/Entity/Utilisateur</b>   | <code>findOuCreatMembre( bigint <b>\$id_utilisateur</b> )</code><br>Cette méthode cherche un utilisateur dans la base donnée avec son id, s'il ne le trouve pas il le crée en récupérant ses information de facebook   |
| private   | <code>importPost( array(stream) <b>\$postes</b> )</code><br>Cette méthode récupère un enregistrement dans un tableau représentant un postes provenant de Facebook et l'enregistre dans la base de données, elle accède au propriétés privés du contrôleur  |
| private   | <code>importJaiemes( )</code><br>A appeler après l'initialisation d'un postes. Cette méthode accède au propriétés privés du contrôleur   |
| private   | <code>importCommentaires( )</code><br>A appeler après l'initialisation d'un postes et importation des jaiemes. Cette méthode accède au propriétés privés du contrôleur   |
| public<br>Symfony/Component/HttpFoundation/Response | <code>progressPostsAction( long <b>\$id_gp</b> )</code><br>Cette méthode nous permet de suivre la progression de l'importation des postes A chaque ajout on persiste un objet de PersistProgressPst  |
| public<br>Symfony/Component/HttpFoundation/Response | <code>progressMembresAction( long <b>\$id_gp</b> )</code><br>Cette méthode nous permet de suivre la progression de l'importation des membres A chaque ajout on persiste un objet de PersistProgressionMb   |
| public<br>Symfony/Component/HttpFoundation/Response | <code>sendVuesImportPostAction( )</code><br>Cette méthode nous permet d'importer le formulaire d'importation de groupe   |

### Sommaire des constantes

|         |                                     |  |
|---------|-------------------------------------|--|
| integer | <b>IMPORT_TOUT=1</b>                | Une constante qui désigne l'importation de tous les postes   |
| integer | <b>IMPORT_DEPUIS=2</b>              | Une constante qui désigne l'importation depuis une date donnée de mise à jour en fonction du nombre limite de postes à importer  |
| integer | <b>IMPORT_JUSQUA=3</b>              | Une constante qui désigne l'importation jusqu'a une date donnée de mise à jour en fonction du nombre limite de postes à importer |
| integer | <b>IMPORT_DEP_JUSQ=4</b>            | Une constante qui désigne l'importation entre deux date de mise à jour   |
| integer | <b>IMPORT_DEP_JUSQ_DATE_CREAT=5</b> | Une constante qui désigne l'importation entre deux date  |

|         |                                 |  |
|---------|---------------------------------|--|
|         |                                 | de publication   |
| integer | <b>IMPORT_DEP_DATE_CREAT=6</b>  | Une constante qui désigne l'importation depuis une date donnée de publication en fonction du nombre limite de postes à importer  |
| integer | <b>IMPORT_JUSQ_DATE_CREAT=7</b> | Une constante qui désigne l'importation jusqu'à une date donnée de publication en fonction du nombre limite de postes à importer |

## Sommaire des propriétés

|  |                                |  |
|--|--------------------------------|--|
| <b>private</b><br>FOS/FacebookBundle/Facebook/FacebookSessionPersistance | <b>\$facebook</b>              | C'est l'instance qui permet d'interagir avec le serveur Facebook, on le déclare privé que si on l'initialise au moins une fois on va pouvoir l'utiliser sur les méthodes privé   |
| <b>private</b> AbstractManagerRegistry                                   | <b>\$manager</b>               | C'est l'objet entityManager mais on utilise le manager car l'entity manager est déprécié dans les récentes versions  |
| <b>private</b> Groupe  | <b>\$groupe</b>                | C'est l'objet groupe courant dont on fait l'importation  |
| <b>private</b> FB/groupeBundle/Entity/Utilisateur                        | <b>\$new_usr</b>               | C'est l'utilisateur nouveau qui change à chaque importation d'un membre qui a posé, aimé ou commenté chaque itération sur la boucle d'importation  |
| <b>private</b> FB/groupeBundle/Entity/Post                               | <b>\$new_post</b>              | C'est le poste courant, changé à chaque itération sur la boucle d'importation  |
| <b>private</b> integer   | <b>\$progression</b>           | Nombre désignant la progression en cours une fois l'importation est finie  |
| <b>private</b> integer   | <b>\$nPostImporté</b>          | Nombre de poste importé une fois l'importation est finie   |
| <b>private</b> integer   | <b>\$timestamp_depuis</b>      | La date depuis laquelle on va importer les postes il est exprimé en timestamp qui est le temps unix à compter depuis le 1er Janvier 1970 à minuit. L'importation se fait depuis cette date jusqu'à une date plus proche  |
| <b>private</b> integer   | <b>\$timestamp_jusqu'au</b>    | La date jusqu'à laquelle on va importer les posts il est exprimé en timestamp qui est le temps unix à compter depuis le 1er Janvier 1970 à minuit. L'importation se fait depuis la limite ou la date jusqu'à cette date. |
| <b>private</b><br>Symfony/Component/HttpFoundation/Response              | <b>\$progressMbPersistance</b> | L'objet qui permet le suivi en temps réel de la progression de l'importation des membres la session est verrouillée  |

|   |  |  |
|---|--|--|
|   |  | par le thread en cours   |
| <b>private</b><br>Symfony/Component/HttpFoundation/Response | <b>\$progressPstPersi</b><br><b>stance</b> | L'objet qui permet le suivi en temps réel de la progression de l'importation des membres la session est verrouillée par le thread en cours |

## 2. Les Entités composant les données d'un Groupe (Modèle Orienté Objet)

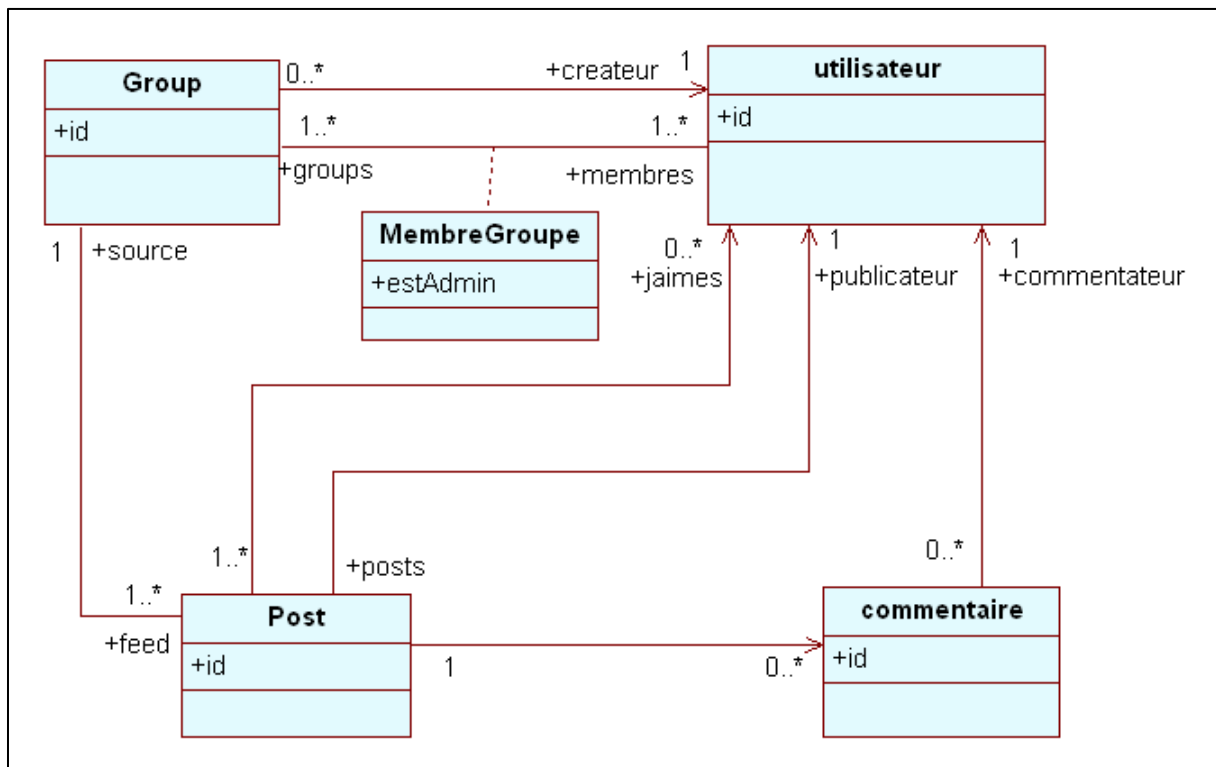


Figure 15 Diagramme de class

Nous n'avons pas tout détaillé sur le diagramme de classe car l'objectif c'est de comprendre le modèle objet de notre base de donnée, pour voir en détail les champs alors il faut se référer à la Figure 16.

Alors comme vous voyez un utilisateur forme 5 associations différentes, il est membre de groupe, publicateur de poste, créateur de groupe, commentateur de postes, et passionné par un poste.

### 3. Schéma de la base de données

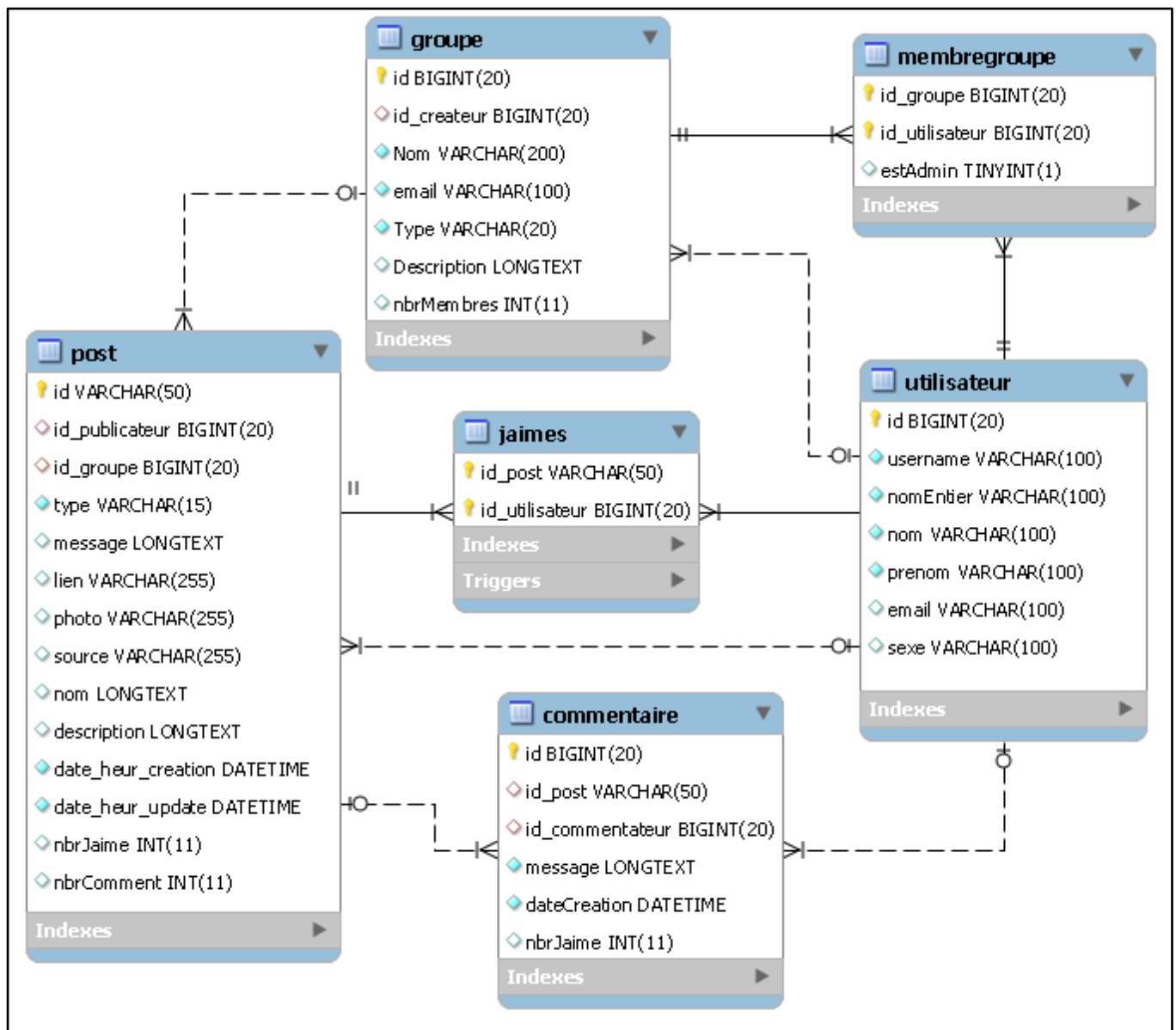


Figure 16 Schéma de la base de données

### 4. Les triggers pour la mise à jour automatique de certains données.

Il y a des données relatives qui peuvent changer pour une entité, comme le compteur du nombre de j'aime pour un post, qui dépendent des j'aime déjà insérés or les j'aime on les insère un à un alors il faut mettre à jour le compteur à chaque insertion, donc on a mis en place des triggers pour leurs mise à jour.

- a. La mise à jour automatique du compteur du nombre de de jaiimes dans une post une fois un enregistrement sur la table jaiime est effectué.

```
DELIMITER //
CREATE TRIGGER update_nbr_jaime AFTER INSERT ON jaiimes
FOR EACH ROW
BEGIN
SELECT nbrJaime INTO @nbrjaimes FROM Post WHERE Post.id =
NEW.id_post;
IF @nbrjaimes IS NULL
THEN
SET @nbrjaimes = 1 ;
ELSE
SET @nbrjaimes = @nbrjaimes +1 ;
END IF ;
UPDATE Post SET nbrJaime = @nbrjaimes WHERE id=NEW.id_post;
END//
DELIMITER ;
```

- b. La mise à jour automatique du compteur du nombre de commentaire dans une post une fois un enregistrement sur la table commentaire est effectué.

```
DELIMITER //
CREATE TRIGGER update_nbr_comment
AFTER INSERT ON commentaire
FOR EACH ROW
BEGIN
SELECT nbrComment INTO @nbrComent FROM Post WHERE
Post.id=NEW.id_post;
IF @nbrComent IS NULL
THEN
SET @nbrComent=0 ;
ELSE
SET @nbrComent=@nbrComent+1;
END IF ;
UPDATE Post SET nbrComment=@nbrComent WHERE id=NEW.id_post;
END //
DELIMITER ;
```

- c. La mise à jour automatique du compteur du nombre des membres d'un groupe une fois un membre est enregistré sur la table membre groupe.

```
DELIMITER //
CREATE TRIGGER update_nbr_membre
AFTER INSERT ON membre_groupe
FOR EACH ROW
BEGIN
SELECT nbrMembres INTO @nbrMembre FROM Groupe
WHERE Groupe.id=NEW.id_groupe;
IF @nbrMembre IS NULL
THEN
SET @nbrMembre=0 ;
ELSE
SET @nbrMembre=@nbrMembre+1;
END IF ;
UPDATE Groupe SET nbrMembres=@nbrMembre WHERE
id=NEW.id_groupe;
END //
DELIMITER ;
```

## VI. Importation des données d'un Groupe Facebook

Avant de commencer à importer les données, sachez bien que lors de l'importation des postes nous allons travailler avec des dates, et faire des conversion, assurez-vous dans vos paramètres que votre time zone est compatible avec le time zone que Facebook a enregistré pour votre compte, par exemple nous au Maroc c'est Africa/Casablanca.

De plus, les scripts que nous allons exécuter dépasseront les 30 secondes fixés par php, alors nous allons fixer ce paramètre à l'infini, pour que php laisse les scripts s'exécuter jusqu'à leur fin quel que soit leur durée.

Pour les deux il faut redéfinir la méthode `init()` de `AppKernel` dans le fichier `app/AppKernel.php` pour les configurer afin que ça marche sur toute l'application :

```
public function init(){
    ini_set('max_execution_time', 0); //durée infini des scripts
    date_default_timezone_set('Africa/Casablanca');
    parent::init();
}
```

## A. Importation des données d'indentification d'un groupe

Ceci on l'a fait avec la première approche architecturale de récupération de données, un peu complexe, qui est purement côté client comme suit:

- au moment du chargement de la page sachant qu'on a fait que `FB` existe dans la page en cours, on écoute si `FB` est bien initialisé, ce qui revient à redéfinir `onFbInit()` et on enregistre un événement d'écoute de changement de statut vers un statut de bien authentifié comme suit :

```
function onFbInit() {
    if (typeof(FB) !== 'undefined' && FB !== null )
    FB.Event.subscribe('auth.statusChange', function(response) {
        if (response.session || response.authResponse){
            {
                //ON TAPE NOS REQUÊTES PAR ICI
                //.....
            }
        }
    });
} //Fin onFbInit
```

- on charge la liste des groupes de l'utilisateur avec les données, mais la requête utilise l'`accessToken` stocké en cookie pas celle, déjà existant possible expiré, ou que `FB` peut se procurer s'il trouve que ce qui existait en cookies a expiré, car il peut invalider ce que . On stocke la liste, on a utilisé un autocompositeur JQuery UI pour faciliter l'utilisateur à choisir le groupe qu'il veut, voici la requête :



```

FB.api('/me?fields=groups.fields(name,id,icon,email,privacy,
description)&&access_token='+$.cookie('accessToken')
,function(reponse){
var tableaugroupes = reponse.groups.data ;

    // .....
}) ;

```



Il ne faut pas oublier que nous avons Récupérer notre access\_token et l'avons stocké sur les cookies, vous pouvez retourner au paragraphe sur Acces Token en tapant Ctrl et en cliquant sur [IV.D](#)

- L'utilisateur choisit le groupe dont il veut enregistrer sur l'autocompleteur, on pourrait les afficher sur une liste pour qu'il cherche,
- On récupère un formulaire et on remplit les champs par les données du groupe choisit, avec un script jQuery et
- L'utilisateur soumet le formulaire avec les données du groupe.

Pour plus de détaille referez-vous au code source du projet à la section des vues du bundle FbgroupeBundle </views/FbGroupeViews/InitGroup.html.twig> et consulter les fonction JavaScript : `onFbInit()` Ligne 6, `InitialiserRechGroupe()` Ligne 23 et `initAutoComlete()` Ligne 47.

## B. Importation des membres du groupe

Ça c'est purement côté serveur et utilisant que graph api mais on peut aussi utiliser FQL en interrogeant la table `group_member`, mais pour qu'on passe le nombre de membre sur la requête on a fait quelque requête côté client en JavaScript. L'action demande l'id du groupe, le nombre de membre, et la limit du bloc de requête.

### 1. Principe

- On pourrait jouer sur la pagination des requêtes fourni par Facebook, en jouant sur `__after_id` et `__before_id`, mais on a récupéré le nombre des membres du groupe côté client et le passé en paramètre à l'action pour critère

d'arrêt de la boucle et faciliter aussi l'interaction avec l'utilisateur, on peut aussi le faire côté serveur comme on verra avec les postes :

```
FB.api(id_groupe+'?fields=members.fields(id)&&access_token='+
$.cookie('accessToken'),function(reponse){
    //Instructions de teste
    nombreMembreGp=reponse.members.data.length;
//On construit la route avec les paramètres
});
```

- Une fois au serveur sur l'action

```
importMembresAction($id_groupe,$nbrmembre,$limit = 40 )
```

On récupère d'abord l'objet groupe dans notre base de données pour persister les membres avec :

```
$groupe = $em->find("FBgroupeBundle:Groupe", $id_groupe);
```

On récupère après le créateur du groupe depuis Facebook et on met à jour la table groupe par l'id du créateur du groupe. Pour se faire on cherche d'abord son id :

```
$createurGb = $facebook->api(
"/$id_groupe?fields=owner.fields(id)" );
$id_createur_gp = $createurGb["owner"]["id"];
```

- On récupère ses données pour l'ajouter à la table utilisateur, s'il n'existe pas

```
$usr = $facebook->api(
"/$id_createur_gp?fields=id,name,first_name,last_name,username
,gender,email" ) ;
$new_usr = (new Utilisateur($usr['id']))
    ->setNomEntier($usr['name'])
    ->setPrenom($usr['last_name'])
    //..... ;
$em->persist($new_usr) ;
//...
```

- On récupère les membres par bloc de requête, avec une limite pour éviter les requêtes longues et parfois sans réponse. On joue avec **limit** et **offset**, **limit** c'est

le nombre d'enregistrement qu'on veut par requête et **offset** c'est par où le curseur veut se positionner. Au début on initialise `$offset=0` après on boucle et

`$offset+=$limit ;` jusqu'à ce que `$offset>$nbremembre`

Après on boucle par un `foreach` sur chaque bloc de requête

```
$offset=0 ;
while ( $offset <= $nbrmembre ) {
$requête =
"/$id_groupe?fields=members.offset($offset).limit($limit).fields"."(id,name,first_name,last_name,username,gender,email,administrator)";

$Resultat = $facebook->api($requête);
foreach ($Resultat['members']['data'] as &$membre) {
//Bloc création utilisateur
$new_usr = new Utilisateur($membre['id']) ;
//.....
//Bloc ajout de l'utilisateur du table membre groupe
} //Fin de foreach
$offset+=$limit;
} //Fin while
```



Cette technique de **offset** et **limit** ne marche pas avec les postes, la filtre **offset** n'existe pas pour la connexion **feed**, c'est **limit** seulement qui existe, avec **feed** on utilise la pagination comme des curseurs utilisant **limit**, **until** et/ou **since** sur la datetime de mise à jour de la poste exprimé en unix time( tempstime)

- Une fois l'utilisateur persisté on l'ajoute au table membre groupe en précisant s'il est administrateur ou pas comme ça :

```
$membre_groupe = (new MembreGroupe($groupe, new_usr))
->setEstAdmin($membre['administrator']);
```

N'oubliez pas de tester en base de données si le membre existe ou bien cuatcher une exception `Doctrine\DBAL\DBALException` le moment du flush au cas où l'utilisateur existe ou le membre existe. Si l'exception est levé alors votre Manager sera forcément fermé alors il faut faire :

```
try {  
    //.....  
    $em->flush()  
} catch (Doctrine\DBAL\DBALException $e){  
    $this->container->get('doctrine')->resetManager();  
    $em = $this->getDoctrine()->getManager() ;  
    $groupe = $em->merge($groupe);  
}
```

## C. Importation des postes du groupes

Pour importer les postes alors là nous avons à interroger la connexion `feed` de pour un groupe dans graph api ou la table `stream` sur FQL. Il faut noter que nous pouvons pas récupérer tous les champs de feed et stream, on a sélectionné les plus importants selon les trois type de poste : **'status', 'photo', 'video', 'link'**

Nous avons traité 7 cas possibles d'importation de postes, soit en important tous les postes avec graph api, soit en important selon la date de mise à jour qui comporte 3 cas simples avec graph api, soit en important selon la date de publication du poste sur 3 cas aussi, en combinant entre graph api et FQL.

### 1. Limites de Graph Api

Graph Api malheureusement ne nous permet pas de faire des requêtes très personnalisés comme sélectionner selon la date de publication d'un post ou ordonner les requêtes selon un champs. Par défaut Graph Api , classe les postes d'un groupe selon leurs date de mise à jour `updated_date`, tel qu'on voit sur Facebook, alors un poste dernièrement commenté serait le premier à apparaitre sur la requête. Donc impossible de préciser une date exacte de publication et Récupérer le poste.

## 2. Limites de FQL

FQL quant à lui certes nous permet de faire des requêtes personnalisés mais la table `stream` correspondant à la table de tous les postes publiés sur Facebook, et trop complexe par ce qu'il est censé prendre toute sorte de publication sur Facebook, il est impossible de savoir si le poste récupéré dont la source est un groupe, c'est une vidéo ou pas, car le champ de type sera rempli du code correspondant à `groupPosted` au lieu de préciser que c'est une photo ou une vidéo, contrairement à graph api où le type est bien précisé si on interroge la connexion `feed`.

Il est impossible aussi de faire une requête sur un champ non indexé, on ne peut pas sélectionner les éléments dont la date de création `created_date` est égale à tel, mais il faut préciser un champ indexé et après ajouter un AND en donnant des filtres de `>=` ou `<=` mais jamais `=` sur un champ non indexé et OFFSET et LIMIT aussi marchent sur FQL, mais toujours à la fin de la requête.

## 3. Importation de tous les postes.

Facebook fixe une limite de 5000 qu'on ne peut pas dépasser sur n'importe quelle requête, seuls les 5000 derniers enregistrements qu'on peut sélectionner, à moins de les récupérer avant et de les stocker en base de données.

Mais il faut faire attention de ne pas oser demander les 5000 enregistrements à la fois sur les postes, la requête prendra trop de temps et voir sans réponse, alors il faut aller par bloc.

### a. Principe et application

Tout se traite côté serveur, en utilisant la 2<sup>e</sup> approche d'importation comme en **Erreur ! Source du renvoi introuvable.** et utilisant Graph Api seulement. La requête se traite avec l'action :

```
importPostsAction( integer $id_groupe, const $MODE_IMPORT,  
string $date_depuis = null, integer $date_jusqua = null,  
integer $limit = 25 )
```

On a besoin seulement des paramètres suivants :

```
integer $id_groupe, const $MODE_IMPORT , integer $limit
```

Les reste des paramètres sera ignoré si `$MODE_IMPORT==1`

On traité l'importation avec deux boucle, une boucle **while**, qui boucle par bloc pour optimiser les requêtes et recevoir les données a temps et une autre boucle **foreach** sur chaque bloc pour parcourir poste par poste pour enregistrer les données.

- On récupère comme d'habitude l'objet groupe par son id pour persister les nouveaux données avec
- Pour la boucle **while** on va utiliser la technique de pagination de Facebook, qui travaille comme des curseurs, à partir du bloc précédant on a accès au bloc suivant et vis-vers-ça. Il faut utiliser les filtres **until** ou **since**, chacun avec la **limit** qu'on fixe ou pas. **until** et **since** s'appliquent à un entier positif représentant le temps : qui est la conversion de la date-time de mise à jour du poste en tempstime c'est le unix-time, qui correspond aux nombre de secondes écoulés depuis le 1<sup>e</sup> janvier 1970 à minuit. Avec php on a utilisé la fonction de conversion c'est la fonction :

```
int strtotime(string date)
```

La date de mise à jour (on l'a déjà expliqué) nommé **updated\_time** c'est la date où le poste a été mise à jour, si le poste n'a jamais été commenté alors **updated\_time=created\_time**.

L'utilisation de since et until se fait comme suit:

- **since** ça veut dire depuis une date antérieur vers les données à date plus proche(supérieur) que la date fourni en paramètre du **since** alors si par exemple on veut sélectionner 10 postes commençant depuis 01/10/2013T12:00:00=1357819200 vers les postes à date plus grande que la date fourni, seulement 10 alors on fait ceci :

```
/id_groupe?fields=feed.since(1357819200).limit  
.(10).fields(id,type,message,...)
```

- **until** c'est le contraire il va sélection les postes en filtrant depuis la date fourni vers les 10 postes à date plus ancien que cette date commençant de cette date

```
/id_groupe?fields=feed.until(1357819200).limit
.(10).fields(id,type,message,...)
```

- On peut combiner **until** et **since** pour sélectionner les postes entre deux dates de mise à jours, il faut la limite pour qu'il ne vous donne pas une limite donnée alors qu'il y a encore de postes, mais **Attention !** il faut que le paramètre de until soit supérieur à celui de **since** , voici l'exemple de requête :

```
/id_groupe?fields=feed.until(1357819200).since(123001110
0).fields(id,type,message,...)
```

**La pagination** se fait comme suit : quand on fait une requête Facebook nous retourne la réponse mais à la fin de la réponse il nous donne un curseur pour pointer sur les prochaines ou précédents enregistrements tout en commençant du dernier pour les suivants ou le premier pour les précédents, ceci en introduisant automatiquement la limite fournie au premier requête, comme limit au curseur suivant ou précédant. Par exemple si on fait pour un groupe dont id=1389011011317375

```
/1389011011317375?fields=feed.until(1384338915).limit(10).fields(id,link,type,updated_time,created_time)
```

Il nous retourne ceci avec la pagination

```
{
  "id": "1389011011317375",
  "feed": {
    "data":
      [.....
      ],
    "paging":
      {
        "previous":
        "https://graph.facebook.com/1389011011317375/feed?limit=10&fields=id,link,created_time,type,updated_time&since=1384338915&__paging_token=1389011011317375_1431091430442666",
        "next":
```

```
"https://graph.facebook.com/1389011011317375/feed?limit=10&fields=id,link,created_time,type,updated_time&until=1384014327&__paging_token=1389011011317375_1430163903868752"

    }

}

}
```

Dans la requête on a 1389011011317375\_1431091430442666 est l'id du premier post sélectionné et 1389011011317375\_1430163903868752 est l'id du dernier sélectionné

Si la requête en cours est vide la pagination existe mais les deux id sont égaux et les paramètre de until et since sont égaux : **C'est la condition d'arrêt de notre boucle while.**

Comment récupérer ces paramètres avec php ? C'est simple on récupère à chaque fois le lien et après parce pour trouver until=datetime et since=datetime et on teste , on a créé un classe outils dans lequel on intégrer une fonction qui nous permet de récupérer un paramètre inclut dans un URL comme ça :

```
public static function paramDansURL($param,$url){
    $urlParcé=parse_url($url);//fonction php predenit
    $queryParts = explode('&', $urlParcé['query']);

    foreach ($queryParts as $v_param) {
        $item = explode('=', $v_param);
        if($param==$item[0]) return $item[1];
    }
}
```

Exemple :

```
$url='https://graph.facebook.com/1389011011317375/feed?limit=10&fields=id,link,created_time,type,updated_time&until=1384014327&__paging_token=1389011011317375_1430163903868752'

$date_until=paramDansURL('until',$url) ;
//$date_until="1384014327"
$date_until=intval($date_until) ;
```



```
//RESULTAT
//$date_until=1384014327
```

- Pour récupérer actuellement les postes alors on a initialiser nos deux entiers par la date d'aujourd'hui :

```
$this->tempstime_jusqua = strtotime(date("c"));
$this->tempstime_depuis = strtotime(date("c"))-30;
//On recule de 30 secondes juste pour créer la différence au
debut // mais par la suite ça sera des dates venant de
facebook
```

- On commence notre boucle on récupère les données et on chaque bloc après on met à jours les tempstime , mais il faut juste reculer de 1 car les requêtes font une inégalité strict sinon vous allez perdre des données :

```
while($this->tempstime_jusqua!=$this->tempstime_depuis){
    $this->tempstime_jusqua--; //On évite l'inégalité strict et
    ce //n'est pas un décompteur car tempstime_jusqua change

    $requete = "/$id_groupe?fields=feed.until($this-
    >tempstime_jusqua ).limit($limit).fields
    (id,type,message,link,full_picture,source,
    name,description,created_time,updated_time,from)";
    //On sollicite facebook
    $Resultat = $this->facebook->api($requete);
    //n'oubliez pas de catcher l'exception
    foreach($Resultat['feed']['data'] as $post){
        $this->importPost($post);
    } //END foreach

    $url_precedant=$Resultat['feed']['paging']['previous'];
    $url_suivant=$Resultat['feed']['paging']['next'];

    $this->tempstime_depuis = intval(Outils::paramDansURL('since',
    $url_precedant));
```

```
$this->tempstime_jusqua = intval(Outils::paramDansURL('until',
$url_suivant));

} //END While
```

La fonction `importPost()` nous permet de prendre un enregistrement d'un post Facebook et l'enregistrer en base de données, ensuite appeler la fonction `importJaimes()` sur le post en cours, flusher les données et appeler après `importCommentaires()`

Pour import post on a :

```
private function importPost($post){
    $this->new_post = new Post($post['id']);
    $this->new_post->setType($post['type'])
    ->setMessage(isset($post['message']) ? $post['message'] : "")
    ->setXXX
    //...
```

On attribue au poste le groupe courant, et on affecte le membre :

```
$this->new_post->setGroupe($this->groupe);
$id_publicateur = $post['from']['id'];
$this->findOuCreatMembre($id_publicateur);
$this->new_post->setPublicateur($this->new_usr);
```

La fonction `findOuCreatMembre($id_utilisateur)` est une fonction privé qui récupère l'id d'un utilisateur Facebook, il le cherche dans notre base de données, s'il ne le trouve pas il va le chercher sur Facebook, récupérer les données et l'enregistre comme membre du groupe courant et en son interne il affecte `$this->new_usr` au nouveau utilisateur enregistré.



Pour la récupération des jaimés et des commentaires nous allons en parler après avoir parlé des cas pour récupérer les postes

#### 4. Importation selon la date de mise à jour, date considéré par défaut sur graph api

Comme on a déjà dit on ne peut jamais sélectionner des postes d'une date exacte en donnant un seul paramètre date, mais on joue avec les filtre et des interval.

Alors le cas suivant est très simple il suffit de spécifier les paramètres de l'action, importPostAction et au lieu de deux boucle on a une seul boucle **foreach** donc selon le mode les paramètres doivent correspondre. On change juste la manière de formuler la requête. Il faut revoir la documentation pour comprendre le travail des constantes, le cas ici présent on traite les constantes suivantes : **IMPORT\_DEPUIS=2**, **IMPORT\_JUSQUA=3**, **IMPORT\_DEP\_JUSQ=4**, les deux premiers constantes demandent la limit et une seule date et le dernier les deux date : voici un extrait du code , on formule les requêtes :

```
switch ($MODE_IMPORT) {

case self::IMPORT_JUSQUA : $requete =
"/$id_groupe?fields=feed.until($this->tempstime_jusqua).limit(
$limit).fields(id,type,message,link,full_picture,source,
name,description,created_time,updated_time,from)";
                                break;

case self::IMPORT_DEPUIS : $requete =
"/$id_groupe?fields=feed.since($this->tempstime_depuis).limit(
$limit).fields(id,type,message,link,full_picture,source,
name,description,created_time,updated_time,from)" ;
                                break;

case self::IMPORT_DEP_JUSQ : $requete =
"/$id_groupe?fields=feed.since($this->tempstime_depuis).
until($this->tempstime_jusqua).limit(5000).fields(id,type,
message,link,full_picture,source,name,created_time,updated_time,from)";
                                break;
```



Une remarque sans la dernière requête de récupération entre deux dates, c'est une mauvaise pratique car il sollicite 5000 post à la fois, seulement j'étais pressé de faire, mais c'était pour lui forcer à me donner car Facebook peut choisir une limite comme il veut et laisser des postes qui sont dans l'intervalle des deux dates fournis.

Mais la meilleure pratique c'est de choisir une limite et boucler en utilisant le paging jusqu'à atteindre la date since si vous commencer par until, et chaque importation tester si la date du poste ne dépasse pas la date since qui est plus ancien.

## 5. Importation selon la date de publication

Pour ce cas, on précise encore qu'il n'est pas possible de donner une date x et récupérer les postes en demandant que date=x, date n'est pas un champs indexé sur Facebook, que ça soit Graph API ou FQL, alors on peut jouer avec les filtres utiliser les inégalités.

Le cas suivant on combine entre FQL et Graph Api. Mais comme j'ai dit FQL ne nous permet pas de remplir nos champs convenablement comme Graph Api les structure, tout reste que celui qu'on peut lui interroger sur les poste en fonction de created\_time comme 2<sup>e</sup> critère après le critère principale qui est le `source_id` qui est le dans notre cas l'id du groupe.

### a. Principe

Le principe consiste à formuler une requête qui va demander les ids des postes à récupérer sur FQL en ajoutant des critères sur la date de création, et après on boucle sur les ids on utilise Graph Api pour récupérer les données de chaque poste. Cette technique reste parfois inefficace car on peut perdre des données à cause d'un comportement miséreux de FQL que je vais expliquer.

Voici la manière de formulation des requêtes pour les 3 cas :

```
switch ($MODE_IMPORT) {  
  //Entre deux dates
```

```

case self::IMPORT_DEP_JUSQ_DATE_CREAT :
$requetFQL = "SELECT post_id FROM stream WHERE
source_id=$id_groupe AND created_time<=$this->tempstime_jusqua
AND created_time>=$this->tempstime_depuis LIMIT 5000";

        break;

//Depuis une date vers une limite
case self::IMPORT_DEP_DATE_CREAT :
$requetFQL = "SELECT post_id FROM stream WHERE
source_id=$id_groupe AND created_time>=$this->tempstime_depuis
LIMIT $limit";

        break;

//Jusqu'à une date depuis une limite
case self::IMPORT_JUSQ_DATE_CREAT :
$requetFQL = "SELECT post_id FROM stream WHERE
source_id=$id_groupe AND created_time<= $this-
>tempstime_jusqua LIMIT $limit";

        break;

    }

```

Au-dessus les cas qu'on récupère les requêtes alors le reste c'est de boucler sur les id un à un Après on fait :

```

$lesIdPosts = $this->facebook->api(array(
    'method' => 'fql.query',
    'query'   => $requetFQL
));

```

Et comme j'ai dit le mieux c'est de toujours aller par un LIMIT et un OFFSET on peut faire :

```

$requetFQL = "SELECT post_id FROM stream WHERE
source_id=$id_groupe AND created_time<=$this->tempstime_jusqua
AND created_time>=$this->tempstime_depuis OFFSET $offset LIMIT
$limit ";
//traitement
$offset+=$limit

```

### Comportement miséreux de FQL



A force de faire de requêtes et d'analyser les réponses, j'ai découvert un comportement anormal sur FQL, j'allais dire un bug mineur. Quand vous faites une requête sur stream en fixant sur les critères la date de création du poste, alors il est impossible de filtrer selon une date de création si la date de mise à jour n'est pas inclut dans l'intervalle de la filtre fournie, fermé ou ouvert.

Exemple : Si vous demander les postes dont `created_time < 1234560000` et que ça trouve qu'il y a un poste `p` dont `created_time = 1200000000` normalement il fait partie de la liste à sélectionner, alors ça trouve que la dernière mise à jour de `p` fait que `updated_time = 1234560001` alors `p` ne sera pas sélectionné. Car `1234560001 > 1234560000` qui est la date du critère, Facebook donne une très grande importance à l'`updated_time`.

Après avoir récupérer les ids nous allons récupérer simplement les données id par id de poste comme ceci :

```
foreach ($lesIdPosts as &$objet_post_id) {

    $id_post=$objet_post_id['post_id'];
    $requete =
    "/$id_post?fields=type,message,link,full_picture,source,
    name,created_time,updated_time,from";

    $post= $this->facebook->api($requete);

    $this->importPost($post);
} //fin foreach sur les id des postes
```

### D. Importation des jaimés d'un postes

L'importation se fait en appelant la méthode `importJaimes()` à l'itérieur de la méthode `importPost()`. Il se passe en important la liste des ids des personnes qui ont aimé le poste et insérer les ides dans la table **jaimés** qui forme une relation **ManyToMany** entre la table Post et la Table utilisateur. Pour compter le nombre

de j'aime il y a un trigger qui le fait directement. Avec gaph api afin de faciliter la tâche on interroge la connexion **likes** de **feed** comme suit :

```
$requete = "/" . $this->new_post->getId() . "?fields= likes.
limit( 5000).fields(id)";
$ResultJaimes = $this->facebook->api($requete);
```

Et puis on boucle comme suit :

```
foreach ($ResultJaimes['likes']['data'] as &$usr_passioné) {
//On cherche l'utilisateur
$this->findOuCreatMembre($usr_passioné['id']);
$this->new_post->addJaime($this->new_usr);
}
```

### Remarque

Vous ne pouvez jamais récupérer plus de 5000 j'aime mais vous pouvez récupérer moins.

## E. Importation des commentaires d'un poste

L'importation se fait appelant la méthode `importCommentaires()` à l'intérieur de la fonction `importPost()`. Il se réalise en récupérant les commentaires, le message et la date de commentaire et l'id de la personne qui a commenté et comme j'ai dit on ne peut pas récupérer plus de 5000 commentaires seuls les derniers commentaires peuvent être récupérés, voici la requête comme suit :

```
$requete = "/" . $this->new_post->
getId() . "?fields=comments.limit(
5000).fields(id,message,like_count,created_time,from)";
$ResultComments = $this->facebook->api($requete);
```

Après la récupération on fait les mêmes étapes de récupération de l'utilisateur :

```
$id_commentateur=$comment['from']['id'];
$this->findOuCreatMembre($id_commentateur);//$this->
new_usr=new Ut
$commentaire->setCommentateur($this->new_usr);
```

```
$commentaire->setPost($this->new_post);
```

## VII. Astuces et Techniques de mise à jour des données d'un groupe

Lors du développement de cette application nous n'avons pas développé les cas de mise à jour mais nous avons initié, faute de temps. Tout reste qu'au cas où l'utilisateur se met à répéter les même requêtes l'application est robuste, elle gère partout les exceptions, et il ne va pas se planter par ce qu'on a fait la même requête par erreur. Mais ce n'est pas la bonne manière de mettre à jour les données, alors nous allons expliquer les détail et quelque astuces des tables sur Facebook pour pouvoir mettre à jour nos données déjà existant en base de données.

### A. Mise à jour des membres du groupe par de nouveaux inscrits.

#### 1. Principe

Après avoir récupéré tous les membres c'est possible que de nouveaux membres soient inscrits au groupe, alors comment récupérer seulement les nouveaux alors qu'il n'y a pas de date d'inscription au table membres ?

Pour répondre à la question je tiens à noter que la date `update_time` existant dans la table membres de graph api n'a rien avoir avec la date d'inscription de l'utilisateur, c'est déjà testé, vous vous-même vous verrez que votre `updated_time` n'a rien n'avoir avec votre date d'inscription à un groupe, mais c'est la date de mise à jour du profile ou status de l'utilisateur sélectionné. Mais il n'existe pas de date d'inscriptions a moins les données que nous avons accès on ne sait pas si Facebook en son interne possède cela. Mais pas publiquement.

#### ➤ Astuce

Les membres du groupe même si on a pas la date d'inscription, ils sont ordonné en fonction de leurs dates d'inscriptions, alors pour cela nous avons créé une entité Historique qu'on stocke des données clés. Ainsi, quand on commence à importer les membres, le premier importé est classé en haut on stocke son id dans la table



Historique et après lorsque on veut faire une mise à jour on récupère son id et après on fait une requête en utilisant la filtre `__befor_id('notre_id')`. On récupère tous les membres nouvellement inscrit avant l'id en cours.

## B. Mise à jour des postes du groupe.

### 1. Principe

Dans ce cas de mise à jour des postes par de nouveaux postes publiés sur le groupe, après avoir récupérer certains, alors là il faut faire très attention, car on ne peut pas utiliser l'astuce précédemment mentionné pour les membres, car certes graph api classe les postes selon la date de mise à jour mais il ne faut pas oublier qu'un ancien poste possible qu'il vient d'être commenté donc il monte en haut. A moins d'utiliser le fait que si le poste existe déjà l'exception est bien géré pas de duplication, mais c'est l'occasion plutôt de mettre à jour ses commentaires ou j'aimes.

#### ➤ Astuce

Il faut sélectionner avec SQL dans notre base de données l'id et la date du poste dont la date de publication est la plus grande et puis utiliser les requêtes FQL en se basant de la date de publication, sélectionner ceux qui ont une date `created_time` plus grande que la date du post sélectionné. Il ne faut pas craindre le comportement mystérieux cité car ici `created_time` est toujours inférieur à `updated_time`.

## C. Mise à jour des commentaires d'un postes existant.

### 1. Principe

Pour mettre à jour les commentaires d'un post dont on a l'id c'est-à-dire existe déjà en base de données, alors il faut savoir que les commentaires n'ont pas de date de mise à jour, on pourrait utiliser l'astuce que graph api classe les données selon la date mais impossible de stocké quelque part l'id du dernier commentaire importé pour tous les postes.

Avant de lancer une mise à jour il faut d'abord compter les commentaires actuels du post en base de données et puis interroger la table `comment_info` avec FQL pour savoir s'il y a de nouveaux commentaires ou pas en comparant le contenu du champs `comment_count` avec le nombre de commentaires existant.

### ➤ Astuce

Pour faire la mise à jour, on va utiliser FQL pour cette mise à jour, mais il faut savoir que sur FQL la date du commentaire(**created\_time** sur Graph API) ne s'appelle pas ainsi sur FQL, il s'appelle juste **time()**. Pour la technique il faut sélectionner la plus grande date de création entre les commentaires du poste et puis demander et interroger la table comment et précisant le critère.

## D. Mise à jour des jaimés

### 1. Principe

On veut mettre à jour les jaimés d'un post existant sur notre groupe. Le cas ci présent est un peu délicat, un jaimé n'a ni date de jaimé ni date de mise à jour, certes ils sont classés par ordre d'apparition sur Facebook, mais pas de date dans les tables qu'on peut se baser.

Avant de lancer une mise à jour il faut d'abord compter les jaimés actuels du post en base de données, on peut le nommer `$nbr_jaimés_existant` et puis chercher le nombre total de jaimés sur Facebook en faisant requête Graph API suivante :

`{id_post}/likes?limit=0&summary=true` ça va nous retourner un objet comme suit :

```
{
  "data": [ ..... ],
  "summary":
  {
    "total_count":
    12 // exemple de nombre de jaimés total
  }
}
```

Ou bien en interrogeant la table `like_info` avec FQL pour savoir s'il y a de nouveaux jaimés ou pas en comparant le contenu du champ `like_count` avec `nbr_jaimés_existant`.

### Astuce

Un j'aime n'a pas de date, mais il faut faire attention de ne pas utiliser limit et offset banalement car les derniers j'aimes effectués sur le poste sont les premiers apparents sur la requête. L'astuce c'est de mettre `like_count` ou `total_count` dans une variable `$total_like_count` et puis faire :

```
$limit=$((total_like_count - $nbr_jaimes_existant))
```


Et c'est fini ☺ vous avez compris la suite !

Il suffit maintenant de faire graph api suivante:

```
/$id_post?fields=likes.limit($limit)
```

## VIII. Déploiement de l'application

Pour déployer l'application, il faut savoir que Facebook ne laisse aucune application être publiée sur le réseau que si elle est soumise pour vérification, l'application peut être développée, utilisée et testée par des personnes qu'on spécifie, mais pas tout le monde. Une page est disponible pour le remplir les détails de l'application et les soumettre. Pour cela il faut retourner sur Facebook pour voir les paramètres de l'application, il faut aller sur la rubrique Détails de l'application [https://developers.facebook.com/apps/{id\\_application}/appdetails](https://developers.facebook.com/apps/{id_application}/appdetails) ou bien aller dans <https://developers.facebook.com/apps> et puis sous

 **Modifier l'application**

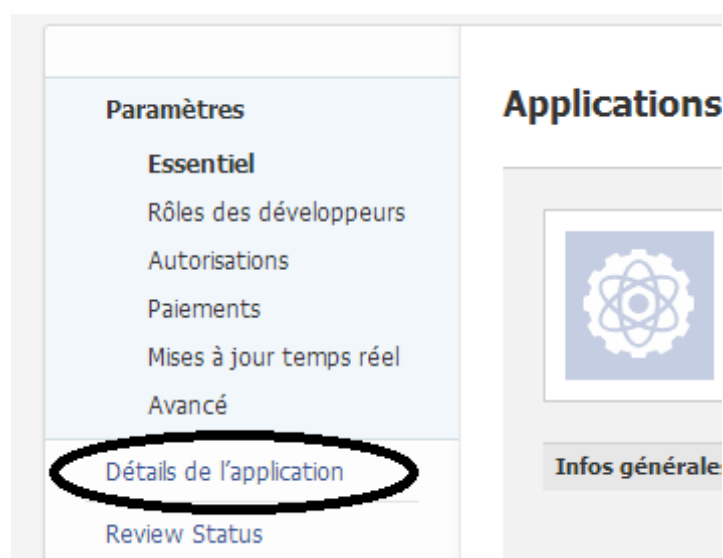


Figure 17 Détails de l'application

Et puis aller dans détails de l'application. Comme dans cette Figure 17. Il faut remplir la page et soumettre la page à Facebook, pour qu'ils valide l'application dans ce cas si vous hébergez l'application sur un serveur, vous devez pouvoir bien déployer l'application Symfony 2 il y a un bon tutoriel [ici](#) , après il faudra fournir les bon URL et noms de domaines

## Conclusion et perspective

---

En conclusion nous ne pouvons pas dire que nous avons traité tous les données d'un groupe, il y a notamment les documents et les fichiers, qui sont à récupérer aussi. Nous n'avons pas dit que nos méthodes, fonctions et astuces sont les meilleurs mais ça répond au besoin de l'utilisateur.

Pour le déploiement de l'application nous avons montré qu'il faut soumettre la page de détails avant de pouvoir changer le mode bac à sable, vers l'accès à tous les utilisateur sur Facebook.

Notre application développée vous pouvez la retrouver sur un dépôt sur le site github.com nommé FbAppGroup <https://github.com/bilaalsoidik/FbAppGroup>, il est open source, la licence qui vous permet de faire tout ce que vous voulez, comme la licence de Symfony 2.

Vous pouvez consulter la documentation du code fourni dans le dossier Documentation