django-facebook-graph Documentation

Release 0.1

FEINHEIT GmbH

November 03, 2014

Contents

Version: 0.1 structured

Contents:

CHAPTER 1

Installation

1.1 Add 'facebook' to your INSTALLED_APPS

The app has a lot of models. To only create the tables you need you have to add the models that you want separately. Here is an example:

```
INSTALLED_APPS = (
    ...
    'facebook',
    'facebook.modules.profile.page',
    'facebook.modules.profile.user',
    'facebook.modules.profile.event',
    'facebook.modules.profile.application',
    'facebook.modules.connections.post',
    ...
)
```

1.2 Add the middlewares

The `SignedRequestMiddleware` is the main middleware that stores the signed request in a special session object and allows your app to access it. Most of the framework expects this middleware to be installed to function correctly.

Because Facebook calls your page initially with POST, you need to pay attention to put the `SignedRequestMiddleware` before the `CsrfViewMiddleware` but after the `SessionMiddleware`.

The `FakeSessionCookieMiddleware` reads the session key from request.GET. This is necessary for Safari 5.0.1 (OSX Leopard) since that browser does not support sessions in iFrames. It comes with a template tag, too.

The `AppRequestMiddleware` adds some tools to help dealing with app requests:

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'facebook.middleware.FakeSessionCookieMiddleware', # for Safari 5.0.1
    'facebook.middleware.SignedRequestMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    <other middlewares>,
    'facebook.middleware.AppRequestMiddleware', # optional.
)
```

1.3 Add the URLs

The basic URLconf entry adds the channel URL, the deauthorize view and some debug tools:

```
url(r'^facebook/', include('facebook.urls')),
```

The registration backend URL activates login functionality through Facebook Connect:

```
url(r'^accounts/', include('facebook.backends.registration.urls')),
```

1.4 The App Settings Dict

This dict stores all the details that facebook provides. You should have an entry for every app in your project. It is recommended to use different app (and therefore a different version of this dict) for local development:

```
FACEBOOK_APPS = {
    'name' : {
            'ID': '?????????',
            'SECRET': '?????????',
            'CANVAS-PAGE': 'https://apps.facebook.com/yourapp',
            'CANVAS-URL': '',
            'SECURE-CANVAS-URL': '',
            'REDIRECT-URL': 'mydomain.com/facebook/redirect/?next=%2F%2Fwww.facebook.com%2Fpages%2F'
            'DOMAIN' : 'localhost.local:8000',
            'NAMESPACE': 'mynamespace',
    }
}
```

1.5 The Facebook Javascript SDK

For any client side Facebook integration you need the Javascript SDK.

Add the fb namespace to the `<html>` tag:

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:fb="https://www.facebook.com/2008/fbml">
```

Add this to the header section of your base template:

```
{% load fb_tags %}
<script type="text/javascript">
    FACEBOOK_APP_ID = '{% fb_app_id %}';
    FACEBOOK_REDIRECT_URL = document.location.protocol + '//' + '{% fb_redirect_url %}';
    FACEBOOK_CHANNEL_URL = '{% url channel %}';
    FACEBOOK_APP_NAMESPACE = '{% fb_app_namespace %}'; // needed for og actions.
</script>
<script type="text/javascript" src="{{ STATIC_URL }}facebook/fb_utils.js"></script>
```

Or this if you use the FeinCMS page extension to discern between different Facebook applications in one installation:

```
{% load fb_tags %}
<script type="text/javascript">
    FACEBOOK_APP_ID = '{% fb_app_id feincms_page.facebook_application %}';
    FACEBOOK_REDIRECT_URL = document.location.protocol + '//' + '{% fb_redirect_url feincms_page.face
    FACEBOOK_CHANNEL_URL = '{% url channel %}';
```

```
</script>
<script type="text/javascript" src="{{ STATIC_URL }}facebook/fb_utils.js"></script>
```

Add this to the bottom of your base template in the scripts section:

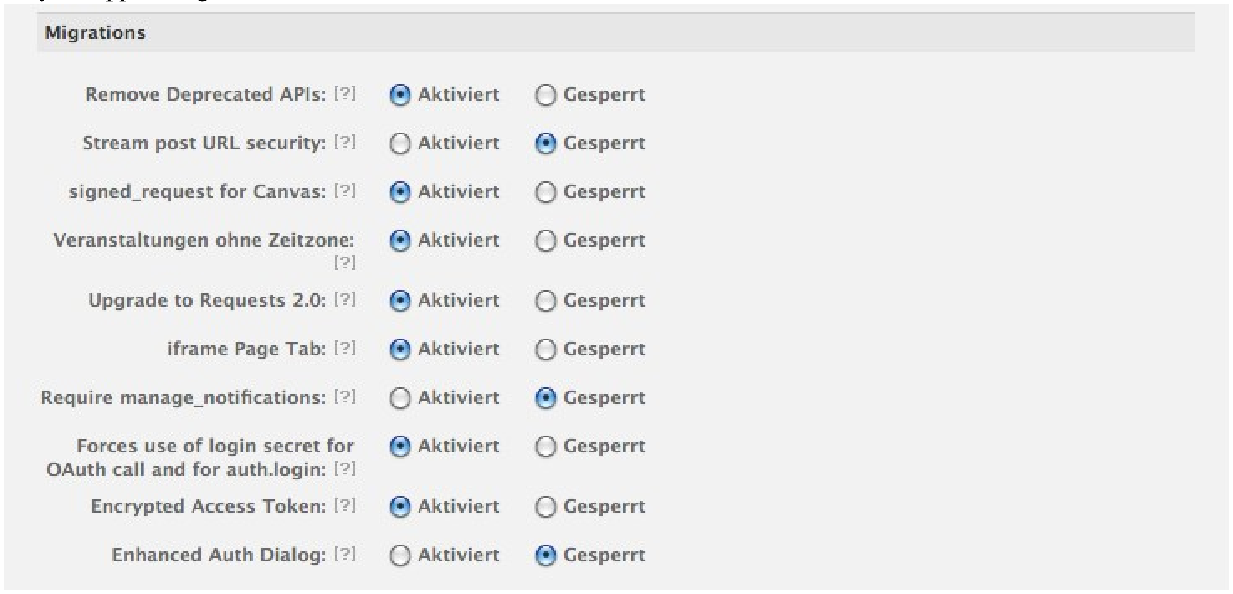
```
<div id="fb-root"></div>
<script type="text/javascript">
(function(d){
     var js, id = 'facebook-jssdk', ref = d.getElementsByTagName('script')[0];
     if (d.getElementById(id)) {return;}
     js = d.createElement('script'); js.id = id; js.async = true;
     js.src = "//connect.facebook.net/en_US/all.js";
     ref.parentNode.insertBefore(js, ref);
   }(document));
</script>
```

The Facebook script is loaded asynchronously. Therefore you have to use the FQ, a simple script queue for inline javascript code that needs the Facebook object. The FQ is run when the SDK has been loaded and the user login status determined. Adding code which is run as soon as the Facebook API is ready is simple:

```
FQ.add(function() {
    // your code here
});
```

1.6 Create a Facebook App

Create a new Facebook app on https://developers.facebook.com/apps. You need to have a verified Facebook account. If you don't, Facebook will ask you to verify your account. `django-facebook-graph` uses OAuth 2.0. Activate it in your app settings:



Migrations

Remove Deprecated APIs: [?]	[✓ Aktiviert] Gesperrt
Stream post URL security: [?]	Aktiviert [✓ Gesperrt]
signed_request for Canvas: [?]	[✓ Aktiviert] Gesperrt
Veranstaltungen ohne Zeitzone: [?]	[✓ Aktiviert] Gesperrt
Upgrade to Requests 2.0: [?]	[✓ Aktiviert] Gesperrt
iframe Page Tab: [?]	[✓ Aktiviert] Gesperrt
Require manage_notifications: [?]	Aktiviert [✓ Gesperrt]
Forces use of login secret for OAuth call and for auth.login: [?]	[✓ Aktiviert] Gesperrt
Encrypted Access Token: [?]	[✓ Aktiviert] Gesperrt
Enhanced Auth Dialog: [?]	Aktiviert [✓ Gesperrt]

1.7 Local Facebook development

If you want to develop locally, follow these steps:

- Create a separate app and set http://localhost.local:8000/ as site URL.
- Map localhost.local to 127.0.0.1 in your `/etc/hosts` file (`/private/etc/hosts` on OS X)

Now you can open your app on Facebook and it will load the data from your runserver. On Firefox you can even chose 'Open Frame in new tab' for quicker page reloads, once you've opened the page in Facebook and the cookie is set.

For Facebook connect, make sure you use the URL localhost.local:8000 and not localhost:8000. This will not work. Facebook enforces the Site URL.

1.8 Facebook Connect support for your website

The Facebook Connect support consists of two parts: A backend for django-registration which creates users and an authentication backend which is responsible for the actual login on a Django website.

1.8.1 Setting the authentication backend

We want to handle logins with the default backend first and fall back to the Facebook authentication backend if the default backend couldn't handle the login request:

```
AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend',
    'facebook.backends.authentication.AuthenticationBackend',
)
```

Currently `django-facebook-graph` only supports Facebook Connect with the Login Button. The Registration Widget is not supported.

CHAPTER 2

Getting started with django-facebook-graph

You need to create a Facebook Application on Facebook Developers for nearly every functionality of `django-facebook-graph`.

> https://developers.facebook.com/apps

For detailed installation instructions check out the *Installation* section.

2.1 Facebook Connect support for your website

Currently the framework supports user login with the facebook login button. It's fairly plug and play. Make sure you have added the authentication backend and login URL as described *Installation* instructions.

2.1.1 Adding the Facebook login to your website

Make sure your `<html>` tag contains the necessary XML namespace information and these FMBL tag to the place where you want the login button to appear:

```
<fb:login-button scope="email" onlogin="window.location.href='{% url auth_login %}?next=/'"></fb:log
```

Checkout the facebook documentation on the login button: http://developers.facebook.com/docs/reference/plugins/login/

2.2 Using the Graph API

You can generate a Graph instance with the following command:

```
from facebook.utils import get_graph
graph = get_graph(request)
```

To make a Graph request to Facebook, simply use `graph.request()`. I.e. to get a certain message object:

```
fb_message = graph.request('%s' % post_id)
```

You can also create facebook user objects like so:

```
from facebook.models import User
user = User(id=graph.me['id'])
user.get_from_facebook(graph=graph, save=True)
```

`django-facebook-graph` stores as much data as possible in the session to minimize requests to Facebook. You can access the session class directly to get informations about the current user do this:

```
from facebook.utils import get_session
fb = get_session()
signed_request = fb.signed_request
```

2.2.1 About the Access Token

Facebook distinguishes between the app access token and the user access token. A user access token is needed for requests that need a user's permission. It's generally more powerful than an app access token. You should generally get the user access token when you pass the request argument to the `get_graph` function.

However, some operations require the app access token, like deleting app requests or saving user score. You can implicitly get the app access token by just calling `graph = get_graph()` without providing the request object, or explicitly by calling `get_static_graph()`.

2.3 Sending posts onto a Facebook wall

2.3.1 Ensure your app has sufficient permissions

This snippet can be used to ask for the `publish_stream` extended permission:

```
function get_publish_perms(callback_fn) {
    FB.login(function(response) {
        if (response.session) {
            if (response.perms) {
                // fb.perms.push(response.perms);
                if (response.perms.indexOf('publish_stream') != -1) {
                    callback_fn();
                }
            } else {
                alert('No permission.');
            }
        } else {
            // user is not logged in
        }
    }, {perms:'publish_stream'});
}
```

To determine whether a permission is already provided you culd use the following snippet:

```
FB.getLoginStatus(function(response) {
    fb.loginStatus = response;
    fb.perms = $.parseJSON(response.perms).extended;
}, true);
```

The second parameter, `true` causes a reload of the login status. This adds the permissions to the response too, which is very helpful for us.

The permissions are also loaded into the fb object on page load. So you could also just try:

```
if (fb.perms.indexOf('publish_stream') != -1) {
    post_to_wall();
} else {
    FB.login(function(response){
            },
            {perms: 'publish_stream' }
    );
}
```

The logical consequence if the if-statement fails would be to make a call to `FB.login()` to show a login window. The problem here is that most browsers block the popup if it doesn't follow an immediate user action. It is therefore recommended to attach the above function to a click event on a button.

2.3.2 Actually create a Facebook wall post

Now that everything else is taken care of actually creating the wall post is easy:

```
from facebook.utils import get_graph
def my_view(request, ...):
    graph = get_graph(request)
    graph.put_wall_post('Hello World!', {
        'name': 'Link name',
        'link': 'http://www.example.com/at/this/location/',
        })
```

It might still be a good idea to enclose the `put_wall_post` call in `try..except` clause.

Keep in mind that if too many users remove a wallpost that had been created through the Graph API your app will get classified as spam.

CHAPTER 3

Django-facebook-graph reference

3.1 Deauthorization callback

There is a default url that can be called for the deauthorization callback:

```
http://<canvas url>/facebook/deauthorize/<app name>/
```

The app name parameter is optional but needed if you have multiple apps to decrypt the signed request. The default action is to delete the user model and all related entries.

3.2 Testing the deauthorization callback

If you are logged in to django you can test the deauthorization callback by calling this url:

```
http://localhost.local:8000/facebook/deauthorize/<app name>/?userid=<user_id>
```

You will be shown a page like the one in the django admin that shows you which entries would be deleted on a deauthorization callback.

CHAPTER 4

Use Cases for django-facebook-graph

4.1 Facebook Connect with Django

You can automatically login a user that has already been authenticated. Add this to your `login.html` template:

```
{% block extra-js %}
{% if not request.user.is_authenticated %}
<script type="text/javascript">
FQ.add(function(){
    if(fb.status == 'connected') {
        window.location = '{% url fb_connect %}?next={% url myview %}';
    }
});
</script>
{% endif %}
{% endblock %}
```

For views that require a logged in user you could either check the status on the server with `graph.me`, or better, on the client side with the following piece of code:

```
{% if request.user.is_authenticated %}
FQ.add(function(){
    if (fb.status == 'not_authorized') {
        window.location = '{% url fb_logout %}';
    }
});
{% endif %}
```

The advantage of checking the status in the browser is that the response time is usually shorter.

4.2 App Tabs and Facebook login

It is not easy to do a deeplink into an app tab. Facebook doesn't really support it. The only workaround is using the *Redirect2AppDataMiddleware* and calling facebook/redirect with the url urlencoded as app_data parameter.

That's how it looks:

```
'REDIRECT-URL': 'http://apps.facebook.com/<MY_APP_NAMESPACE>/facebook/redirect/?next=http%3A%2F%2Fww
```

Make sure you have the *canvas url* parameters in the developer app set to the root or wherever facebook should fetch the redirect from.

4.3 Login while keeping the App requests

On some browsers you have to make sure that all protocols match. I.e. if your iframe is loaded via https you cannot redirect to a http url. The problem here is that if Facebook itself is on a http url the redirect will be to the http url as well.

```
<fb:login-button scope="email" onlogin="login_redirect();"></fb:login-button>

function login_redirect(){
    window.location = ('https:' == location.protocol ? 'https://' : 'http://')+'yourdomain.com{% url
}
```

4.4 Fetch a user’s newsfeed

This one is tricky due to Facebook’s new privacy policy. You could might want to use:

```
graph.request('me/feed')
```

Unfortunately this returns only your own posts as well as friend’s posts that have been marked as public. Posts from friends that have been marked as ‘Friends only’ won’t show up. But you can use:

```
graph.request('me/home')
```

This returns the last 25 posts from the user’s wall. Unfortunately this does not really work with test users.

4.5 Deeplinks into Facebook Tabs

Getting a static URL that is shareable on Facebook of a page within a Tab is tricky. The only way to do deeplinking is to add the path as app_data parameter to the URL and then parse it. Unfortunately a URL like this will not be accessible by the Facebook linter since it cuts off all GET parameters of facebook URLs.

A workaround for this is using the canvas URL and top-redirecting into the tab using the path as app_data parameter.

That’s where the redirect_to_page decorator comes in. Decorate your index view and every view that might get called directly and it checks if the page is correctly embedded within a Facebook page. If it’s not it will redirect the user into the tab.

The decorator needs a new value in the app_dict. A list of allowed Facebook Page IDs for the tab: PAGES=[<page_id>,...]. The decorator needs to be called with the app name as parameter:

```
from facebook.decorators import redirect_to_page

@redirect_to_page('myapp')
def index(request):
    more code here.
```

CHAPTER 5

Page Login

To allow an app to post to a Facebook Page, a page administrator needs to grant the app the manage_pages permission. If you have multiple apps, you have to define the following in settings.py:

```
DEFAULT_POST_APP = 'myapp'
```

You need to have a Facebook login button on the Page admin template and make sure you are logged in. Then select the pages you want to have the app access to and choose 'Get an access token for the selected page(s)' from the admin actions. If everything worked out, you should have a checkmark on the right. From now on your app can do the same things you can do.

Keep in mind that new access token expire after 60 days.

CHAPTER 6

Using django-facebook-graph with FeinCMS

The facebook page extension allows you to have several Facebook apps assigned to different FeinCMS pages. This way you can have multiple tabs on your Facebook page and manage them in a single admin.

6.1 Facebook Application Extension

In your models.py add:

```
from facebook.feincms.extensions import facebook_application

Page.register_extension(facebook_application)
```

Make sure you have installed the fb.Page module In the FeinCMS admin add your Facebook pages. The FeinCMS pages have two additional fields for the facebook app and page. If the page is set, the content can only be displayed within that page. The app is used to decrypt the signed request.

6.2 Content Type Extension

This is a monkey-patch for FeinCMS contents. It adds two fields: render_like and app_installed. It allows to control if a content type is displayed whether a user has liked a page or not or has installed the app.

Because it's a monkey-patch the order of import is important (models.py):

```
from feincms.content.richtext.models import RichTextContent
from facebook.feincms.extensions import content_type_extension

content_type_extension(RichTextContent)
```

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*