# Zds member Documentation

***Release 1.0***

**firm1**

October 26, 2015

Contents

Zds Member it's a fork of zds-site member app to make a reusable application.

Github

CHAPTER 1

# Summary

## 1.1 Installation

Install the development version:

```
pip install zds-member
```

Add `member` to your `INSTALLED_APPS` setting:

```
INSTALLED_APPS = (
    # ...
    "member",
    # ...
)
```

See the list of settings to modify the default behavior of zds-member and make adjustments for your website.

Add `member.urls` to your URLs definition:

```
urlpatterns = patterns("",
    ...
    url(r"^members/", include("member.urls")),
    ...
)
```

Once everything is in place make sure you run `syncdb` (Django 1.4 and 1.6) or `migrate` (Django 1.7) to modify the database with the `member` app models.

## 1.2 Configuration

Configuration and list of available settings for zds-member

### 1.2.1 Specifying files

```
ZDS_MEMBER = {
        'bot_account': "admin",
        'anonymous_account': "anonymous",
        'external_account': "external",
        'bot_group': 'bot',
        'members_per_page': 100,
```

```
    }

APP_SITE = {
        'name': u"ZesteDeSavoir",
        'litteral_name': u"Zeste de Savoir",
        'email_noreply': "noreply@example.com",
}

ZDS_MEMBER_SETTINGS = {
        'paginator': {
            'folding_limit': 4
        }
    }
```

### 1.2.2 Url

in `url.py`

```
(r'^members/', include('member.urls'))
```

### 1.2.3 Templates

A complete set of working templates is provided with the application. You may use it as it is with a CSS design of yours, re-use it or extend some parts of it.

Relations between templates:

```
base.html
|_ member
|   |_ base.html
|   |_ index.html
|   |_ login.html
|   |_ profile.html
|   |_ register
|   |   |_ base.html
|   |   |_ index.html
|   |   |_ send_validation_email.html
|   |   |_ send_validation_email_success.html
|   |   |_ success.html
|   |   |_ token_already_used.html
|   |   |_ token_failed.html
|   |   |_ token_success.html
|   |_ settings
|   |   |_ account.html
|   |   |_ base.html
|   |   |_ memberip.html
|   |   |_ profile.html
|   |   |_ promote.html
|   |   |_  unregister.html
|   |   |_  user.html
|   |_ new_password
|   |_ forgot_password
|_ misc
|   |_ badge.part.html
|   |_ member_item.part.html
```

## 1.3 Usage

Describes how to use zds-member when it is installed and configured.

### 1.3.1 Custom forms

You can replace the default forms in views.

Examples:

## 1.4 Features

### 1.4.1 Sign up

The registration of a member is in two phases:

- The member creates their account and provides a username, a password and valid email address.
- A confirmation email is sent with a token that will activate the account.

**Warning:**
- Commas are not allowed in the nickname, which also can not begin or end with spaces.
- The password must be at least 6 characters.

### 1.4.2 Unsuscribe

### 1.4.3 Promote member

In order to manage the members directly from the site (ie without having to go through the Django admin interface), a promote interface was developed. This interface allows:

1. Add / Remove a member / group (s)
2. Add / Delete superuser status to a member
3. (De) activate an account

First point allows to pass a member in new group. If other groups are emerging (validator) then it will be possible here also to change it. The second point can provide access to the member at the django interface and this promotion interface. Finally, the last point simply concerns the account activation (normally made by the Member at registration).

It is managed by the PromoteMemberForm form available in the `zds/member/forms.py`. It's then visible through the template `member/settings/promote.html` that may be accessed as root user by the profile of any member.

### 1.4.4 Add karma

### 1.4.5 Reset Password

When a member forgets their password, you can reset it. The old password is deleted and the user can choose a new one. For this, he went on the password reset page (‘‘members/reinitialisation /) from the login page.

On this page the user has to enter his username or email address. For this, click on the link to the form. When the user clicks the submit button, a token is randomly generated and is stored in a database.

A message is sent to the user's email address. This email contains a reset link. This link contains a parameter, the reset token and directs the user to address `members/new_password/`.

This page allows you to change the user's password. The user completes the form and clicks the submit button. If the password and the confirmation field and the corresponding password is business rules, the password is changed. The system displays a message confirming the password change.

**Warning:**
- The password must be at least 6 characters.
- The link is valid for one hour. If the user does not click on the link in the allotted time, an error message is displayed.
- The password reset token is valid only once. If the user tries to change their password with the same token, a 404 page is displayed to the user.

## 1.5 Django Back-end

**Documented files:**



### 1.5.1 Models

**class** `member.models.`**`Ban`**(**args*, ***kwargs*)
This model stores all sanctions (not only bans). It stores sanctioned user, the moderator, the type of sanctions, the reason and the date. Note this stores also un-sanctions.

**class** `member.models.`**`KarmaNote`**(**args*, ***kwargs*)
A karma note is a tool for staff to store data about a member. Data are:

•A note (negative values are bad)

•A comment about the member

•A date

This helps the staff to react and stores history of stupidities of a member.

**class** `member.models.`**`Profile`**(**args*, ***kwargs*)
A user profile. Complementary data of standard Django *auth.user*.

**`can_read_now`**()
Check if you can read a web site content as user. If you can't read, you can't login on website. This happens when you have been banned (temporarily or definitively)

**Returns** `False` if you are banned, `True` else.

**Return type** bool

**can_write_now**()
Check if you can write something on a web site as user. This happens when you have been reading only (temporarily or definitively)

**Returns** `False` if you are read only, `True` else.

**Return type** bool

**get_absolute_url**()
Absolute URL to the profile page.

**get_avatar_url**()
Get the avatar URL for this profile. If the user has defined a custom URL, use it. If not, use Gravatar.

**Returns** The avatar URL for this profile

**Return type** str

**get_city**()
Uses geo-localization to get physical localization of a profile through its last IP address. This works relatively good with IPv4 addresses (~city level), but is very imprecise with IPv6 or exotic internet providers.

**Returns** The city and the country name of this profile.

**Return type** str

**is_private**()
Check if the user belong to the bot's group or not

**Returns** `True` if user belong to the bot's group, `False` else.

**Return type** bool

**class** member.models.**TokenForgotPassword**(**args*, ***kwargs*)
When a user forgot its password, the website sends it an email with a token (embedded in a URL). If the user has the correct token, it can choose a new password on the dedicated page. This model stores the tokens for the users that have forgot their passwords, with an expiration date.

**get_absolute_url**()

**Returns** The absolute URL of the "New password" page, including the correct token.

**class** member.models.**TokenRegister**(**args*, ***kwargs*)
On registration, a token is send by mail to the user. It must use this token (by clicking on a link) to activate its account (and prove the email address is correct) and connect itself. This model stores the registration token for each user, with an expiration date.

**get_absolute_url**()

**Returns** the absolute URL of the account validation page, including the token.

member.models.**auto_delete_token_on_unregistering**(*sender*, *instance*, ***kwargs*)
This signal receiver deletes forgotten password tokens and registering tokens for the un-registering user;

member.models.**logout_user**(*username*)
Logout the member.

**Parameters** **username** – the name of the user to logout.

### 1.5.2 Views

**class** member.views.**MemberDetail**(***kwargs*)
Displays details about a profile.

**model**
alias of User

**class** member.views.**MemberList**(***kwargs*)
Displays the list of registered users.

**class** member.views.**RegisterView**(***kwargs*)
Create a profile.

**form_class**
alias of RegisterForm

**class** member.views.**SendValidationEmailView**(***kwargs*)
Send a validation email on demand.

**class** member.views.**UpdateMember**(***kwargs*)
Updates a profile.

**form_class**
alias of ProfileForm

**class** member.views.**UpdatePasswordMember**(***kwargs*)
User's settings about his password.

**class** member.views.**UpdateUsernameEmailMember**(***kwargs*)
User's settings about his username and email.

**form_class**
alias of ChangeUserForm

member.views.**active_account**(*request*)
Active token for a user.

member.views.**forgot_password**(*request*)
If the user forgot his password, he can have a new one.

member.views.**generate_token_account**(*request*)
Generate token for account.

member.views.**get_client_ip**(*request*)
Retrieve the real IP address of the client.

member.views.**login_view**(*request*)
Log in user.

member.views.**logout_view**(*request*, **args*, ***kwargs*)
Log out user.

member.views.**member_from_ip**(*request*, **args*, ***kwargs*)
Get list of user connected from a particular ip

member.views.**modify_karma**(*request*, **args*, ***kwargs*)
Add a Karma note to the user profile

member.views.**new_password**(*request*)
Create a new password for a user.

member.views.**settings_promote**(*request*, **args*, ***kwargs*)
Manage the admin right of user. Only super user can access

member.views.**unregister**(*request*, **args*, ***kwargs*)
allow members to unregister

member.views.**warning_unregister**(*request*, **args*, ***kwargs*)
Displays a warning page showing what will happen when user unregisters.

### 1.5.3 Forms

**class** member.forms.**ChangeUserForm**(**args*, ***kwargs*)
Update username and email

**class** member.forms.**LoginForm**(*next=None*, **args*, ***kwargs*)
The login form, including the "remember me" checkbox.

**class** member.forms.**MiniProfileForm**(**args*, ***kwargs*)
Updates some profile data: biography, website, avatar URL, signature.

**class** member.forms.**NewPasswordForm**(*identifier*, **args*, ***kwargs*)
Defines a new password (when the current one has been forgotten)

**class** member.forms.**ProfileForm**(**args*, ***kwargs*)
Updates main profile rules:

- Display email address to everybody
- Display signatures
- Display menus on hover
- Receive an email when receiving a personal message

**class** member.forms.**PromoteMemberForm**(**args*, ***kwargs*)
Promotes a user to an arbitrary group

**class** member.forms.**RegisterForm**(**args*, ***kwargs*)
Form to register a new member.

**clean**()
Cleans the input data and performs following checks:

- Both passwords are the same
- Username doesn't exist in database
- Username is not empty
- Username doesn't contain any comma (this will break the personal message system)
- Username doesn't begin or ends with spaces
- Password is different of username
- Email address is unique through all users
- Email provider is not a forbidden one

Forbidden email providers are stored in *forbidden_email_providers.txt* on project root.

**Returns** Cleaned data, and the error messages if they exist.

## 1.6 REST API

### 1.6.1 Member's Infos

#### List

**GET /api/**
List of website's members

**Query Parameters**

- **page_size** – number of users. default is 10

**Status Codes**

- 200 OK – no error

#### Details

**GET /api/** (**int:** *user_id*) **/**
Gets a user given by its identifier.

**Example request**:

```
GET /api/800/ HTTP/1.1
Host: example.com
Accept: application/json, text/javascript
```

**Example response**:

```
HTTP/1.1 200 OK
Vary: Accept
Content-Type: text/javascript
{
   "pk": 800,
   "username": "firm1",
   "is_active": true,
   "date_joined": "2014-07-28T02:57:31",
   "site": "http://zestedesavoir.com",
   "avatar_url": "http://static.wamiz.fr/images/animaux/rongeurs/large/souris.jpg",
   "biography": "I'm beautiful",
   "sign": "cool",
   "show_email": false,
   "show_sign": true,
   "hover_or_click": true,
   "email_for_answer": false,
   "last_visit": "2015-10-20T03:24:06"
}
```

**Parameters**

- **user_id** (*int*) – user's unique id

**Status Codes**

- 200 OK – no error
- 404 Not Found – there's no user with this id

**GET /api/mon-profil/**
Gets informations about identified member

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – user are not authenticated

**Request Headers**

- Authorization – OAuth2 token to authenticate

**PUT /api/** (**int:** *user_id*) **/**
Updates a user given by its identifier.

**Parameters**

- **user_id** (*int*) – user's unique id

**JSON Parameters**

- **pk** (*int*) – user's unique id

**Status Codes**

- 200 OK – no error
- 404 Not Found – there's no user with this id

**Request Headers**

- Authorization – OAuth2 token to authenticate

### 1.6.2 Sanctions

#### Read Only

**POST /api/** (**int:** *user_id*) **/lecture-seule/**
Applies a read only sanction at a user given.

**Parameters**

- **user_id** (*int*) – user's unique id

**JSON Parameters**

- **pk** (*int*) – user id to read only
- **ls-jrs** (*string*) – Number of days for the sanction.
- **ls-text** (*string*) – Description of the sanction.

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – Not authenticated
- 403 Forbidden – Insufficient rights to call this procedure. Must to be a staff user.
- 401 Unauthorized – Not found

**Request Headers**

- Authorization – OAuth2 token to authenticate

**DELETE /api/** (**int:** *user_id*) **/lecture-seule**
Removes a read only sanction at a user given.

**Parameters**

- **user_id** (*int*) – user's unique id

**JSON Parameters**

- **pk** (*int*) – id of read only user

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – Not authenticated
- 403 Forbidden – Insufficient rights to call this procedure. Must to be a staff user.
- 401 Unauthorized – Not found

**Request Headers**

- Authorization – OAuth2 token to authenticate

### Ban

**POST /api/** (**int:** *user_id*) **/ban/**
Applies a ban sanction at a user given.

**Parameters**

- **user_id** (*int*) – user's unique id

**JSON Parameters**

- **pk** (*int*) – user id to ban
- **ban-jrs** (*string*) – Number of days for the sanction.
- **ban-text** (*string*) – Description of the sanction.

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – Not authenticated
- 403 Forbidden – Insufficient rights to call this procedure. Must to be a staff user.
- 401 Unauthorized – Not found

**Request Headers**

- Authorization – OAuth2 token to authenticate

**DELETE /api/** (**int:** *user_id*) **/ban/**
Removes a ban sanction at a user given.

**Parameters**

- **user_id** (*int*) – user's unique id

**JSON Parameters**

- **pk** (*int*) – id of banned user

**Status Codes**

- 200 OK – no error
- 401 Unauthorized – Not authenticated
- 403 Forbidden – Insufficient rights to call this procedure. Must to be a staff user.
- 401 Unauthorized – Not found

**Request Headers**

- Authorization – OAuth2 token to authenticate

## /api

GET /api/, 10
GET /api/(int:user_id)/, 10
GET /api/mon-profil/, 10
POST /api/(int:user_id)/ban/, 12
POST /api/(int:user_id)/lecture-seule/, 11
PUT /api/(int:user_id)/, 11
DELETE /api/(int:user_id)/ban/, 12
DELETE /api/(int:user_id)/lecture-seule, 11

## m

## W