ClickToCall SkypeTest Documentation

Release 0.0.1

Andrea Mucci

August 04, 2015

Contents

CLick2Call is Python project test with tornado and PostgreSQL database Server.

This project was created for a demo and is not a complete click to call service.

The project support:

- Client App registration (oAuth 2.0)
- User-Agent login
- create a call, not SIP signaling or RTP flux, is only a REST call that open a call SESSION
- terminate a call
- limit ongoing and outgoing call for a user (if the user is registered with different apps, the limitation work as well)
- a very basic admin REST interface to check the user call status and the call informations.

CHAPTER 1

Requirements

- Python 2.7
- Tornado 2.3
- PostgreSQL

CHAPTER 2

Installation

Is possible to use this project in a virtualenv, install before a virtual env:

```
$ virtualenv venv --distribute
New python executable in venv/bin/python
Installing distribute..............done.
Installing pip..............done.
```

To activate the environment.

```
$ source venv/bin/activate
```

Now if you are in the root project folder you can use the requirements files to install all the the required packages

```
$ pip install -r requirements.txt
```

If all work well you have the project ready to use.

CHAPTER 3

Database Installation

The project have a sql folder with the database dump, this project work only with postgreSQL copy the dump file into postgrSQL To modify the satabase connections parameters you need to open and modify the `settings.py` file under the ``app folder‘:

```
DATABASE = {
    "name": "clickcall",
    "user": "postgres",
    "password": "mypassword",
    "host": "localhost",
    "port": "5432"
}
```

CHAPTER 4

Usage

If you have activate your virtualenv you can launch the project form the root folder.

```
$python manage.py
```

The default value lauch a web server to the port `8888` the Web Server accept internal connections ‘‘http://localhost:8888‘

CHAPTER 5

Contents

5.1 REST API

In this section will describe the REST API.

5.1.1 Authorization Process

This project use the oAuth 2.0 process (Draft 31). The Authentications Flow processes supported are two.

- Application Flow
- Client-Side Flow

Application Flow

This is used if the tester is with a web server application (php, java, python, ruby, C#, etc.). The sequence is described in this image.

The scenario is redirecting the User-Agent to the login page (popup etc.). The url to call from the Client Application is.

```
GET http://localhost:8888/oauth/auth
```

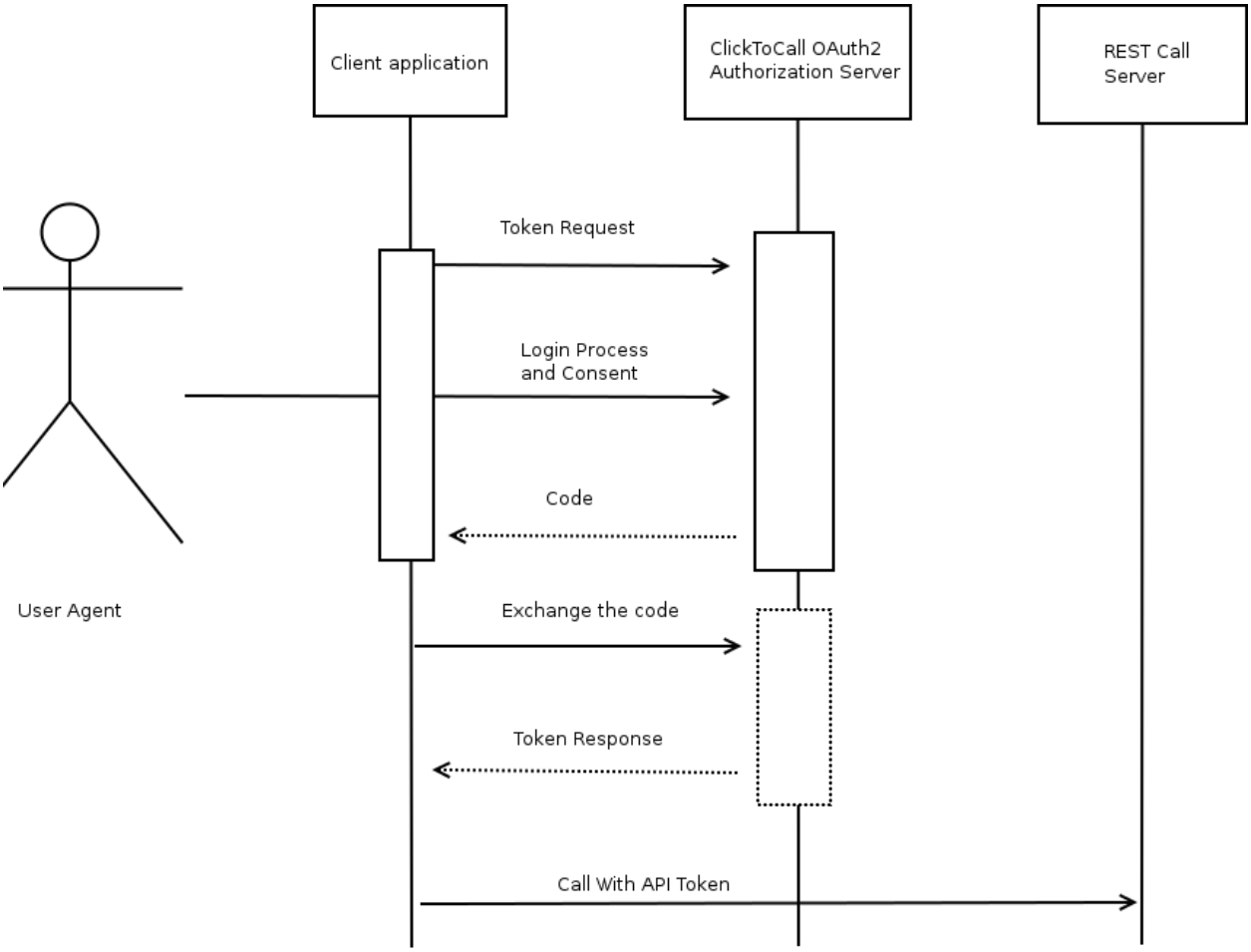
is **Mandatory** add the following parameter to this url

- *client_id*: * This is the client id added to the database in the client table *
- *response_type*: * For the Application Flow this value is `code` *
- *redirect_uri*: * This is the redirect uri added into the database in the client table for the client_id *
- *scope*: * The scope in this project is **not processed**, have only one scope and is call *

> **Warning:** This project have a test page for the redirect process, this page is
> http://localhost:8888/test
> this page return the code and the token after the grant process and the token creation!

So, in conclusion the final url for the client auth is.

```
GET http://localhost:8888/oauth/auth?client_id=<client_id>&response_type=code&redirect_uri=http://loc
```


Client application
ClickToCall OAuth2
Authorization Server
REST Call
Server
Token Request
Login Process
and Consent
Code
User Agent
Exchange the code
Token Response
Call With API Token

After calling this url, the system automatically redirect to the Login-Page, the redirection is a 302 Status Code The login Page URL where the User-Agent is redirect is:

```
GET http://localhost:8888/auth/login?client_id=<client_id>&response_type=code&redirect_uri=http://loc
```

The page show JSON formatted String with the parameters attached to the URL. If the User-Agent is already Logged, the System redirect with a 303 status code to the Grant page. If the user is not logged in, is necessary to send a post to the same url, with the `username` and `password` parameters.

```
POST http://localhost:8888/auth/login

body_parms:
client_id=<client_id>&response_type=code&redirect_uri=http://localhost:8888/test&scope=call&username=
```

If the login is processed correctly, the System redirect to the Grant Page. This is the process that authorize the Application-ID take some information of the User-Agent. In a classic project this page show a form with two button, *Accept Refuse*. In this project is returned a json string with the parameters passed to the URL from the login page. If the User-Agent never have Grnated the Application-ID must be send a POST to the same url:

```
POST http://localhost:8888/auth/grant

body_params:
client_id=<client_id>&response_type=code&redirect_uri=http://localhost:8888/test&scope=call&grant=tru
```

In this example the User-Agent has sent `grant=true` so he have accepted the Application Access. Now the system redirect to the `redirect_uri` passed as a parameter. *if you is used the test page ''http://localhost:8888/test'' you will take the code from the body that show a JSON String.* The redirect is:

```
GET http://localhost:8888/test?code=<code>
```

After this process will be can take the token with the code returned. To obtain the token you need to send a GET to this url and with thsi parameters.

```
GET http://localhost:8888/oauth/token?client_id=<client_id>&client_secret=<secret_id>&code=<code>&red
```

In this URl have one new parameter `client_secret`. This parameter is one column in the Client Table and is the *password* for the Application. After call this URL, the System will be redirect to the `redirect_uri` , like the code step, this page show in the body a JSON string with the token code, the redirection is with a 303 status code. The url called is:

```
GET http://localhost:8888/test#access_token=<token>&token_type=Bearer&expires_in=3600
```

Now with the token is possible to call the REST Server.

Client-Side Flow

This Flow is practically the same as the Application-Flow. This is used by Javascript Application (client-side). In the image have the flow view of the process.

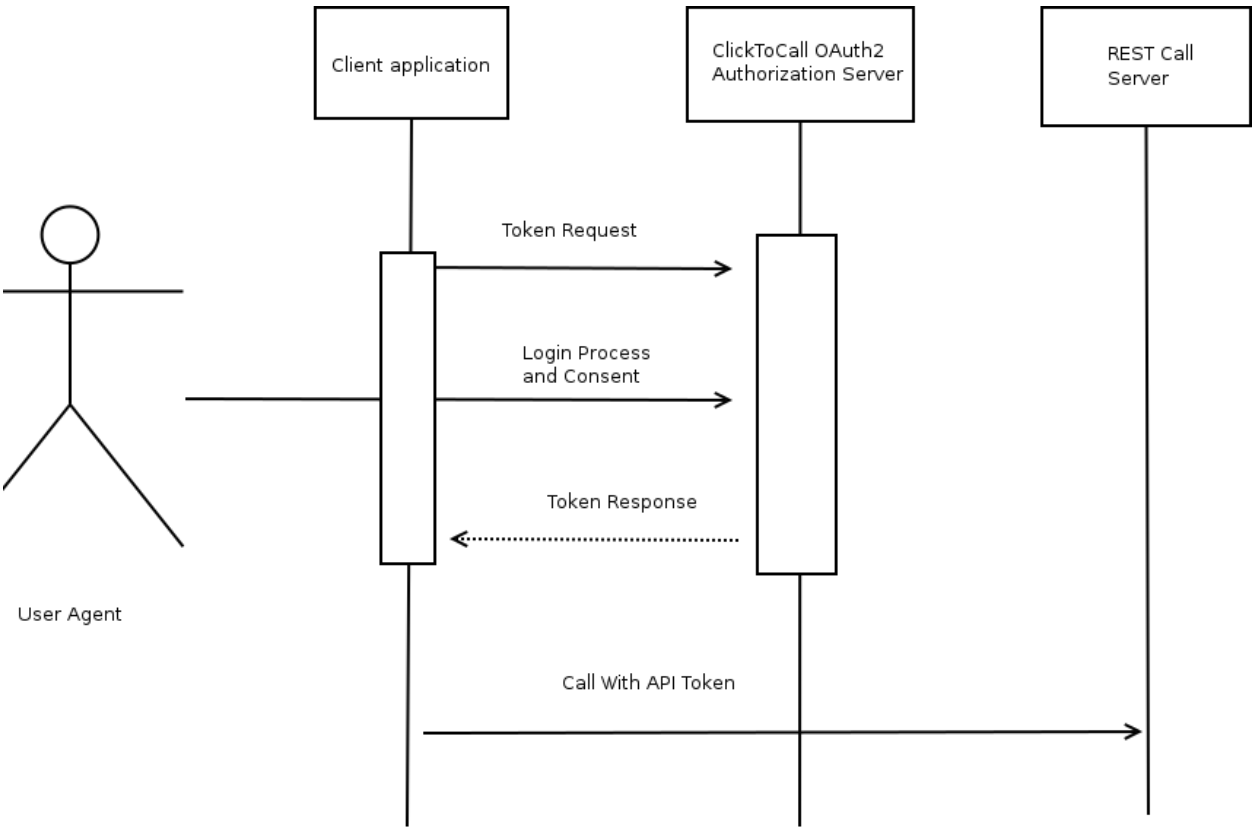
This process will not request the token from a code authentication. the url flow is:

```
GET http://localhost:8888/oauth/auth?client_id=<client_id>&response_type=token&redirect_uri=http://lo
```

login redirection:

```
GET http://localhost:8888/auth/login?client_id=<client_id>&response_type=token&redirect_uri=http://lo
```

Send the post to login the user:




```
POST http://localhost:8888/auth/login

body_parms:
client_id=<client_id>&response_type=token&redirect_uri=http://localhost:8888/test&scope=call&username
```

grant if is not granted, if the customer is granted will be automatically redirect to the redirect_uri:

```
POST http://localhost:8888/auth/grant

body_params:
client_id=<client_id>&response_type=token&redirect_uri=http://localhost:8888/test&scope=call&grant=t
```

after this step, the system redirect to the page and send the parameters as:

```
GET http://localhost:8888/test#access_token=<token>&token_type=Bearer&expires_in=3600
```

Now is possible to call the REST Sever

5.1.2 REST Call Interface

HowTo send the token for authentication

Is possible to send the token in two methods. as a URL parameter:

```
GET http://localhost:8888/rest/call?access_token=<access_token>
```

or in the header:

```
POST /rest/call HTTP/1.1
Host: localhost:8888
User-Agent: PersonalUser-Agent
Authentication: Bearer <your-token>
```

Is important to add the `Bearer` parameter to stay compliant with the oAuth2 protocol.

Make a call

Send a POST to this url:

```
POST http://localhost:8888/rest/call
```

The parameter to send in the post is `number=<number_to_call` If all work well you will obtain a status code 201, in the `Location` header you will obtain the GET URL with the token call to get the call information:

```
RESPONSE:
HTTP/1.1 201 OK
Server: xLightweb/2.6
Content-Length: 0
Location: http://localhost:8888/rest/call?token_call=<token_call>
Content-Type: application/x-www-form-urlencoded
```

Stop a Call

Send a PUT to this url:

```
PUT http://localhost:8888/rest/call
```

And in the PUT body this parameter:

```
token_call=<token_call>
```

This method return a status code 200

Get Call info

To get informations about a call you need to send a GET request:

```
GET http://localhost:8888/rest/call?token_call=<token_call>
```

Yu will obtain a JSON string or a xml string with the call information. if have some problems will return 409 error Conflict

Get All Ongoing Calls

To get all ongoing calls you need to send a GET to this url whitout any parameter:

```
GET http://localhost:8888/rest/call
```

The result is a JSON or a XML string with all the ongoing calls for the logged user.

5.2 Code API

5.2.1 Database Models

Base Class

Derived Classes

Generic Classes

5.2.2 Http Handler Classes

Base Classes

Authentication Derived Classes

REST Derived Classes

5.2.3 Exception Classes

Exceptions

Exception Module for Custome Errors

exception core.exceptions.**DBConnectionError**
Exception used to raise a Database Connection Error.

exception core.exceptions.**DBCursorError**
Exception used to raise a Cursor Connection Error.

exception core.exceptions.**DBQueryError**
Exception used to raise a Query Connection Error.

exception core.exceptions.**GenericException**

exception core.exceptions.**GrantNotAuthorized**
Exception used to raise a grant error.

exception core.exceptions.**ObjectDoesNotExist**
Exception used to except if the query return a value or not.

exception core.exceptions.**RestMaxCallError**
Exception used to raise the maximum ongoing call permitted.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

C

C

D

G

O

R