Systers Portal Documentation

Release 1.0

Rose Robinson, Ana Balica, Chitra Khatwani

Nov 16, 2018

Contents

Systers Portal is a platform for Systers communities to share information within and with other communities. The following documentation is intended for Systers Portal developers and administrators.

CHAPTER 1

Administration of Systers Portal

1.1 Configuration of social accounts

Systers Portal offers the possibility of user authentication using local and social accounts. There are 4 social apps available:

- facebook
- github
- google
- twitter

Each social app requires prior configuration. It is necessary to add a *SocialApp* record per provider via the Django admin containing app credentials. The process of configuration for each provider is described in detail below.

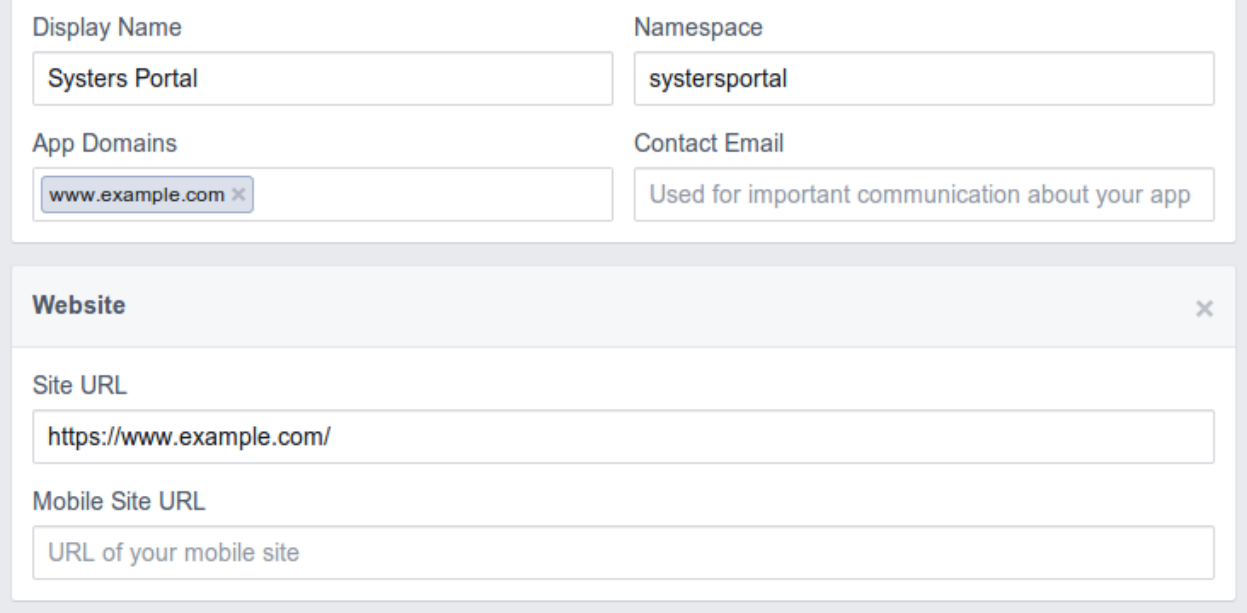
Warning: The providers' interface for generating API keys and client IDs might have changed slightly since this tutorial was written, but the general process should be the same.

Note: Throughout this tutorial https://www.example.com domain is used for demonstration purposes. It should be replaced with a real domain name, e.g. https://systers.org or http://localhost:8000 if developing on a local machine.

1.1.1 Facebook

The following section describes the configuration of social login using Facebook OAuth2. First signup as Facebook developer. From the top toolbar go to *Apps > Create a New App*. Fill the form, click *Create App*. Choose *Display Name*, unique *Namespace* and category.

On the left sidebar of the app click on *Settings*. On *Settings* - *Basic* page it is necessary to add a platform to the app. Select platform - *Website*. Set *Site URL* to be https://www.example.com. Same domain name add to *Add Domains*. Save changes.



If you are testing Facebook OAuth locally, then you can add http://localhost to *App Domains* and http://localhost:8000 to *Site URL*. It won't work, if you use loopback IP address 127.0.0.1 instead of localhost.

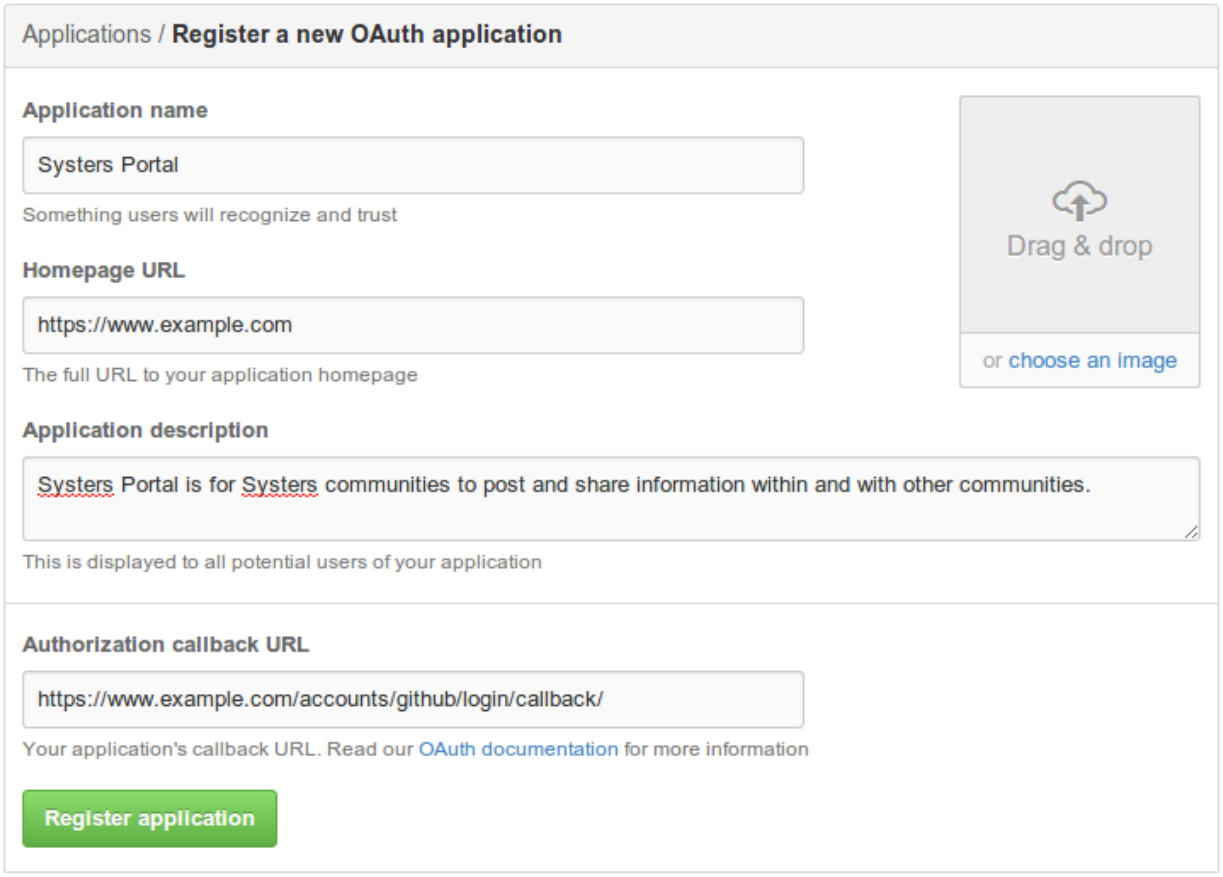
On the *Dashboard* page you can find the App ID and App Secret. Go to Systers Portal and create a new social app at https://www.example.com/admin/socialaccount/socialapp/add/. Select *Facebook* provider, choose a suggestive name (e.g. *Facebook OAuth2*), copy the App ID and the App Secret to the *SocialApp* record. Choose the available sites for which to enable the social auth with Facebook and hit Save.

To test if Facebook social auth using OAuth2 is configured properly, go to https://www.example.com/accounts/login/ and try to login using Facebook. You will be asked to authorize the application and will be redirected to a page to enter your username and email that you want to use for login.

1.1.2 Github

Register a new oAuth application at https://github.com/settings/applications/new. Fill the form. Enter the *Homepage URL* - https://www.example.com. The *Authorization callback URL* should be of the form:

```
https://www.example.com/accounts/github/login/callback/
```



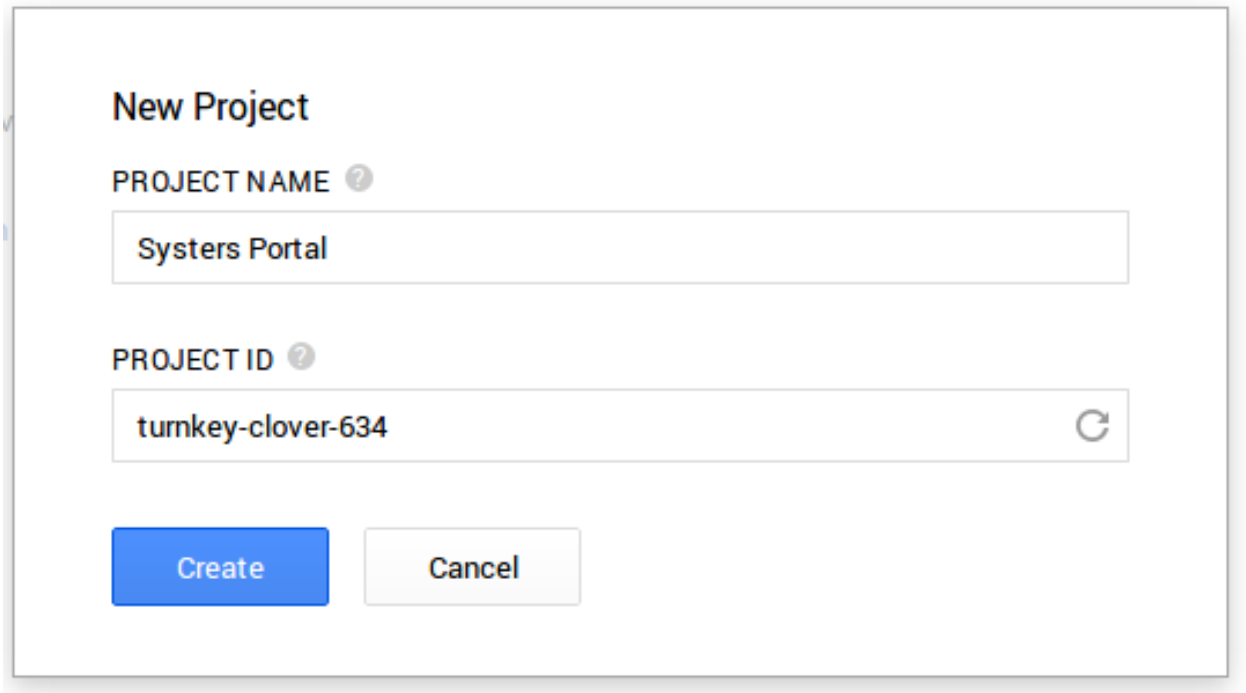
Immediately after you will be redirected to the Application dashboard to access the newly generated Client ID and secret. Create a new social app for Systers Portal at https://www.example.com/admin/socialaccount/socialapp/add/. Select *Github* provider, choose a suggestive name (e.g. *Github OAuth*), copy the Client ID and the secret to the *SocialApp* record. Choose the available sites for which to enable the social auth with Github and hit Save.

To test if Github social auth is configured properly, go to https://www.example.com/accounts/login/ and try to login using Github. You will be asked to authorize the application and will be redirected to a page to enter your username and email that you want to use for login.

1.1.3 Google

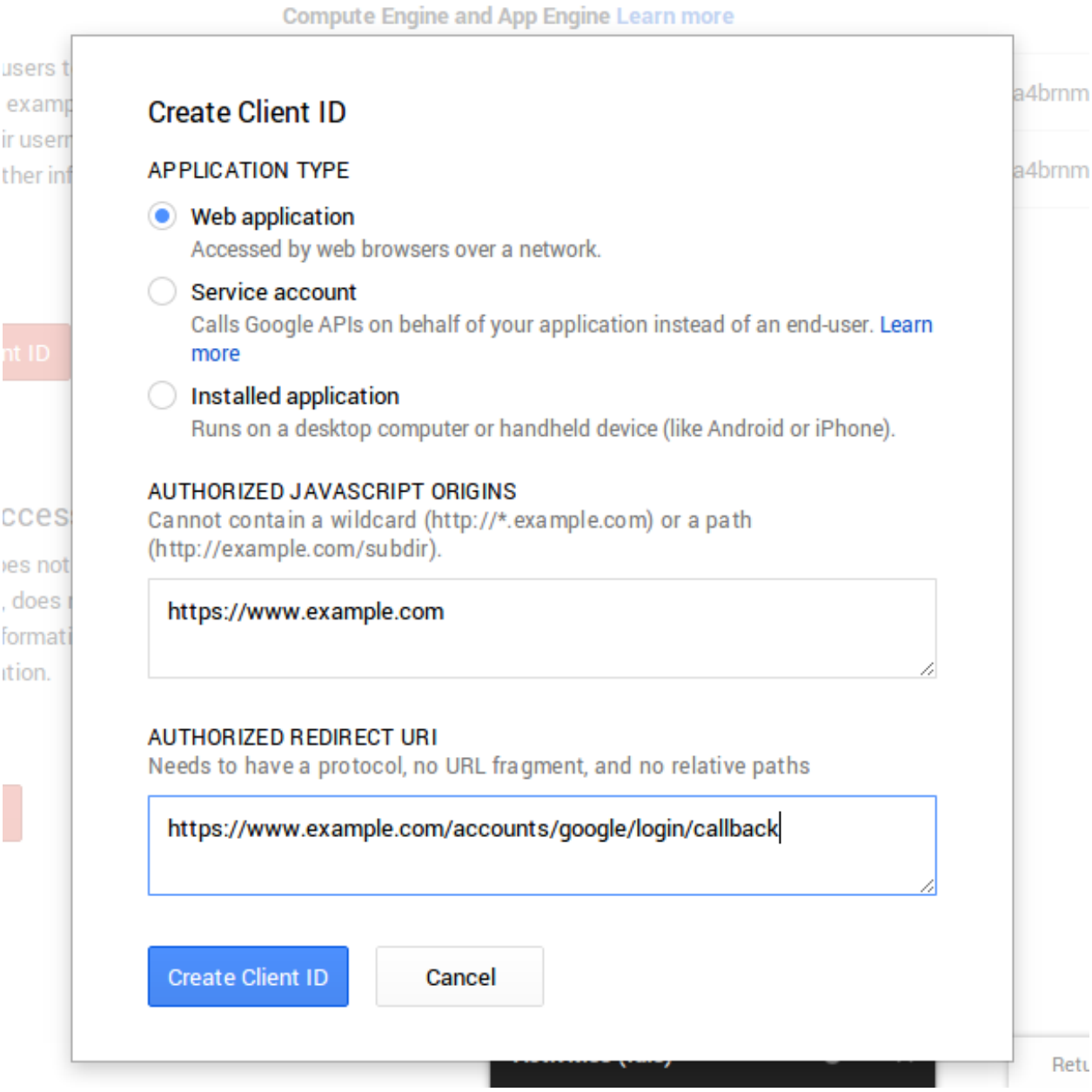
The Google provider is OAuth2 based. More on using OAuth2 to access Google APIs: https://developers.google.com/accounts/docs/OAuth2

Go to Google Developers Console and create a new project. A window will pop up for you to fill in project information. Enter a suggestive name for the new project (e.g. *Systers Portal*), let the system generate a unique project ID and click *Create*.

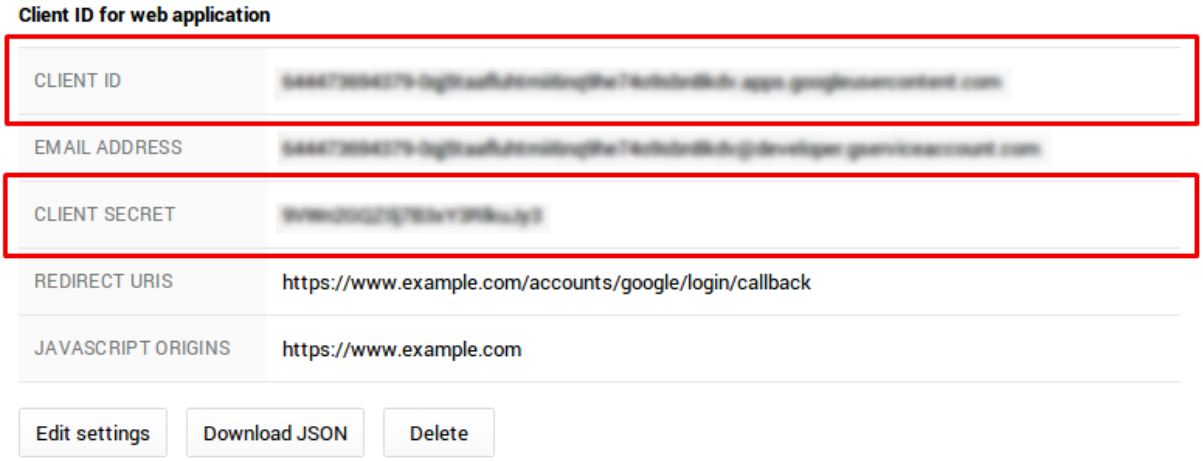


Right after the project will be created, you will be redirected to Project Dashboard. Click on *Enable an API* button. On the left sidebar in the section *APIs & AUTH* access *Credentials* page. A popup will appear to enter details to create a client ID. Choose *Web application*. For *Authorized Javascript Origins* enter the website domain name. In *Authorized Redirect URI*, type the development callback URL, which must be of the form:

```
https://www.example.com/accounts/google/login/callback
```



The system will generate a client ID and a secret for the following web application.



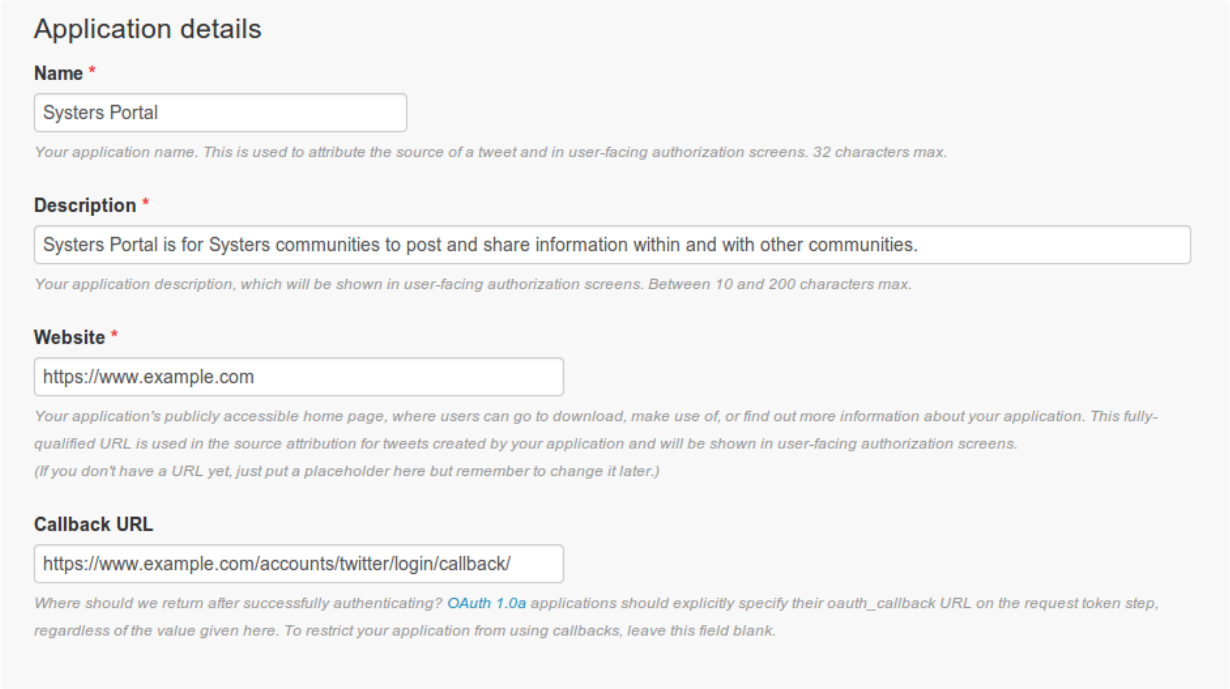
Create a new social app for Systers Portal at https://www.example.com/admin/socialaccount/socialapp/add/. Select *Google* provider, choose a suggestive name (e.g. *Google OAuth2*), copy the Client ID and the Client Secret from Google Developer Console to the *SocialApp* record. Choose the available sites for which to enable the social auth with Google and hit Save.

To test if Google social auth is configured properly, go to https://www.example.com/accounts/login/ and try to login using Google.

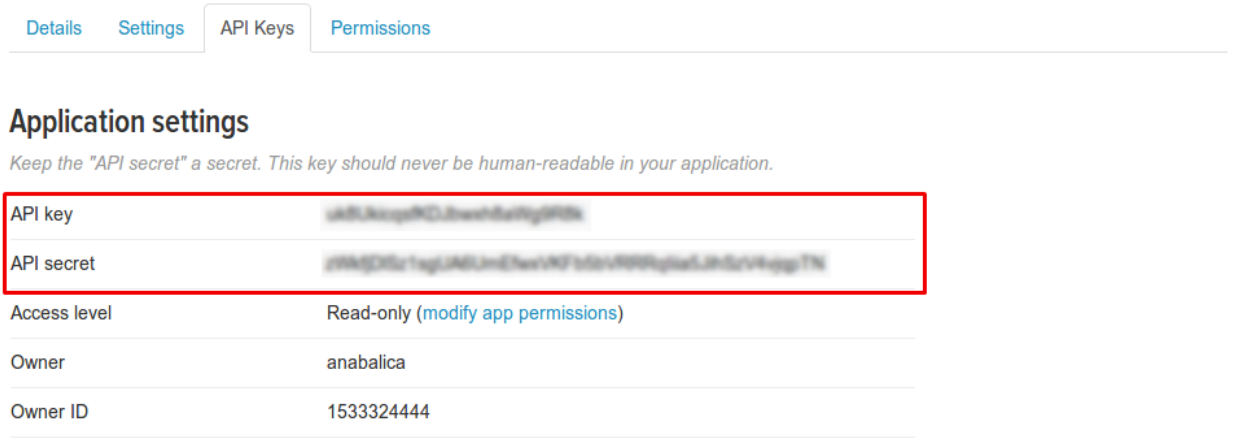
1.1.4 Twitter

Go to Twitter Apps page and create a new app. Fill the form. Enter the *Website* - https://www.example.com. The *Callback URL* should be of the form:

```
https://www.example.com/accounts/twitter/login/callback/
```



On the Application Management page of Systers Portal, access the *API keys* tab. It contains the API key and API secret necessary to enable authentication with twitter.



Create a new social app for Systers Portal at https://www.example.com/admin/socialaccount/socialapp/add/. Select *Twitter* provider, choose a suggestive name (e.g. *Twitter OAuth*), copy the API key and the API secret to the *SocialApp* record. Choose the available sites for which to enable the social auth with Github and hit Save.

To test if Twitter social auth is configured properly, go to https://www.example.com/accounts/login/ and try to login using Twitter. You will be asked to authorize the application and will be redirected to a page to enter your username and email that you want to use for login.

CHAPTER 2

Development of Systers Portal

2.1 Prerequisites

Before diving into development, please take a minute to check if you have all the necessary tools and skills to get started. You should be familiar with the following:

- WindowsUnix environment (GNU/Linux, OS X, etc)
- Git version control system: committing, pushing, pulling, branching
- Python programming language
- Django framework
- HTML, CSS, JavaScript for front-end coding
- Unittesting
- PEP8

If you don't know yet any of these, please take some time to read about, understand and practise the tools.

2.1.1 Unix System

Unix systems are so far the most developer-friendly environments. If you don't have a Unix system installed on your machine, try out Ubuntu. It is a free operating system with constant updates, friendly X Window System and comes with Python pre-installed. From the developer's perspective you should be comfortable using a terminal. Theoretically is it possible to stick with Windows OS, though none or very little assistance will be provided for OS specific issues.

2.1.2 Windows System

If you are not familiar with any unix system and want to get started on windows that's fine.

2.1.3 Git

If you have never heard of git, go ahead and read the first 3 chapters of Pro Git book. On the same page you can find cheatsheets, video lessons and a reference manual.

2.1.4 Python

Please be sure you speak fluent Python, as it is the main language Systers Portal is written in. Essentially you should know how to invoke Python interpreter, manipulate numbers, strings, lists, tuples, dictionaries, sets, use control flow tools (if, for, break, continue, pass), I/O operations, errors and exceptions, classes, inheritance. It is a plus if you know about decorators, regular expressions, generators, iterators.

2.1.5 Django

Django is one of the most popular Python web framework. Django official website contains pretty detailed documentation. At first try out the tutorial and build a small app. After you feel confident about Django, scroll through Systers Portal codebase to check your understanding. If some parts seem complicated, go back to the documentation to focus on a specific topic or layer.

2.1.6 HTML, CSS, JavaScript

The good news is that the front-end of Systers Portal is kept as simple as possible. You should be comfortable with writing HTML, CSS and maybe a bit of JavaScript.

2.1.7 Unittesting

"Untested code is broken code", that's why we try to write unittests for every new functionality. Testing helps us validate the functionality we already have and check whether the new code is implemented correctly. For that we use the Django unittest module. There are plenty of good resource on unittesting, but you can start with Django testing.

2.1.8 PEP8

PEP8 is the style guide for Python code. We are following the guide throughout the whole codebase and check everything against a PEP8 linter. So please take some time to read through this document - https://www.python.org/dev/peps/pep-0008.

2.2 Contributing

If you want to start contributing to Systers Portal or you are a regular contributor, this is the place for you. It covers such topics as setting up the project, working and updating pull requests, Continuous Integration with Travis.

Note: The `$` symbol denotes the shell prompt, don't type it.

2.2.1 Setup the project

Before doing any actual work, it is necessary to prepare your local machine for developement and deploy Systers Portal locally.

1. Install `git` on your local computer. If on Ubuntu, run:

```
$ sudo apt-get install git-core
```

2. Set up your name and email address in `git` configuration:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

3. Go to http://github.com and create a GitHub account, if you don't have one yet. Use the previously introduced email to create the GitHub account. That way GitHub will be able to associate your `git` commits with your GitHub profile.

4. Fork the Systers Portal repo. In the upper left corner there is **Fork** button. A "copy" of the project will appear in the list of your repositories.

5. Generate SSH keys and add the public key to your GitHub account. GitHub provides a very good article on how to do that.

6. Go to the page where the forked Systers Portal is. Copy the *SSH clone URL* from the right sidebar. If your username is *john-doe*, then the url will be `git@github.com:john-doe/portal.git`. Now clone the forked project to your system files:

```
$ git clone git@github.com:john-doe/portal.git
```

7. Install the project locally for development using this guide from project README.

8. Running tests and flake8 successfully with no errors or failures ensures that you have installed everything correctly.

2.2.2 Make a Contribution

Say you have decided to fix a bug or implement a new feature. This guide will show you step by step how to make a contribution to Systers Portal.

1. Systers Portal stable branch is `master`. Periodically we make a release when the codebase is stable and can be used in production. `Develop` branch is the branch that has all the latest changes and should be used as a base branch for development. And so enter the `portal` directory and checkout to `develop` branch.

```
$ cd portal
$ git checkout develop
```

2. Choose a task to work on. It can be a beginners task or a sophisticated feature you want to implement for Portal. For beginners we have tasks with easy TODO tag on GitHub issues.

3. Create a new feature branch from `develop` branch. The feature branch should have a short and relevant name.

```
$ git checkout -b <feature-branch-name>
```

4. Work on the task. Write tests if necessary. When you consider the task done, you should check the following:

   (a) PEP8 style - `flake8 systers_portal`. If `flake8` gives you warnings, please correct them. There should be nothing on output.

(b) Tests - `python systers_portal/manage.py test --settings=systers_portal.settings.testing`. This command will run all the tests. If a test will fail or give errors, please check the traceback and correct the issues. Rerun the tests.

(c) Check test coverage:

```
$ coverage run systers_portal/manage.py test --settings=systers_portal.
↪settings.testing
$ coverage report
```

If the total coverage percentage is lower than the number that appears on the coverage badge on GitHub, then write tests to keep it on the same level or improve the test coverage.

(d) If you have made any changes to the HTML or have manipulates the DOM using JavaScript, please check the validity of the file. Open the page in the browser, copy the page source code and paste it here. If there are errors, please correct them and revalidate.

5. If all the checks have passed, you can commit your changes. Please be careful to not commit any user-specific files or changes that are out of scope. You can make more commits, if you consider it is necessary.

```
$ git add <file1> <file2>
$ git commit -m "relevant commit message"
```

6. Make sure you have the latest `develop` branch before pushing the feature branch. For that checkout to `develop` and pull the latest changes. If `develop` branch was updated, you should switch back to your feature branch and rebase your work against `develop`. Solve conflicts, if there are any.

```
$ git checkout develop
$ git pull origin develop
$ git checkout <feature-branch-name>
$ git rebase develop
```

7. Push the feature branch:

```
$ git push origin <feature-branch-name>
```

2.2.3 Make a Pull Request

We use peer code review to accept or reject the changes made by contributors. It helps to prevent many mistakes and guarantee project quality. For that we use GitHub pull requests.

1. Go to GitHub and make a pull request. Choose a relevant title, add description if necessary and point to the task you have solved. The source branch of the pull request should be your feature branch and the target branch should be `develop` branch. Review your pull request and make sure everything is alright.
2. Someone from the team will review your code, provide feedback and if everything is ok, will merge your changes. If asked to make any changes, update the pull request by one of the two strategies presented below.
3. Don't forget to sync your fork in case the upstream repo was updated.

2.2.4 Update a Pull Request

Quite often the reviewer will leave comments and ask you to make some changes to the initial code. There are 2 strategies how to update your pull request.

Update same pull request

1. Checkout on the feature branch - the source branch of the pull request.
2. Work on enhancements and suggestions.
3. Make a commit with `amend` option. It will update your last commit and will change the SHA-1 of that commit.

```
$ git add <file1> <file2>
$ git commit –amend
```

4. Make a force push to the feature branch. This will update the pull request automatically. But will not notify the reviewer about it, so consider leaving a comment about it in the pull request. The benefit is that the reviewer can see a diff between the previous submission and the new one.

Create a new pull request

1. Checkout on `develop` branch and create a new feature branch with an incremented version value at the end of the feature branch name.

```
$ git checkout develop
$ git checkout –b <feature-branch-name>2
```

2. Apply the same changes you have made on the first version of the feature branch, additionally applying enhancements and suggestions left by the reviewer.
3. Make a commit:

```
$ git add <file1> <file2>
$ git commit –m "relevant commit message"
```

4. Push the feature branch to GitHub and create a pull request.
5. Close the previous pull request manually.

2.2.5 Continuous Integration with Travis

Travis CI Systers Portal - https://travis-ci.org/systers/portal

Coveralls Systers Portal - https://coveralls.io/r/systers/portal

For continuous integration we use Travis CI service. Every time we make a push to Systers Portal repo, Travis builds our project and runs the tests. It also notifies us about any errors or failures, that way preventing us from breaking the project.

Along with Travis CI, we use code test coverage metric using coveralls service. Please note that high coverage is not a guarantee for good tests.

2.3 Architecture

This page will give you a generic overview on Systers Portal project architecture. As any other Django project Portal organizes its functionality in several apps:

- **blog** - handles showing, adding, editing and deleting news and resources.
- **common** - generic functionality that can't be part of any other app. For example, landing, about, contact pages, generic models, helpers, mixins that are used in several apps.
- **community** - community and subcommunities functionality, like adding new communities, views and editing community profiles, showing, adding, editing and deleting community pages, managing permissions regarding each community.

- **membership** - handles showing, creating, approving and rejecting join requests to a community, removing and inviting users to become members of a community.
- **users** - showing and editing user personal profile.
- **meetup** - handles meetup locations and meetup functionality.

The templates are placed inside `systers_portal/templates` folder organized in a folder structure similar to the apps tree. Respectively the templates location matches the views location.

Each app along with models, views, urls and other app related code (admin, forms, mixins, utils) contains a folder with migrations and tests. The migrations are generated by Django and shouldn't be edited manually. The tests folder mimics the app modules covering each module with a separate test file. For example, tests for `app_name/models.py` are contained in the `app_name/tests/test_models.py`.

2.4 Communities

2.4.1 Creating a community

In order to create a new community, it is enough to fill out necessary fields in a Community form.

When a new community is created, the following actions are triggered:

1. 4 new Groups are created using the name of the community
2. each new group is being assigned a set of usual Django permissions and row level permissions for the new Community object
3. the admin of the Community is being added to the Community admin group
4. the admin of the Community is being added to Community members

Suppose we create a community named *Systers* with an admin called *Foo*. This is what is going to happen:

1. **4 new Group are created with the following names:**
   - *Systers: Community Admin*
   - *Systers: User and Content Manager*
   - *Systers: Content Manager*
   - *Systers: Content Contributor*

   The naming of groups is important and helps identify a community with its auth Groups.
2. Groups are being assigned specific permissions. All permissions are listed in this file.
3. "Foo" user is added to the *Systers: Community Admin* group.
4. "Foo" user is added to *Systers* members.

2.4.2 Editing a community

Community profile can be edited by changing any of the Community fields. On community update, the actions are trigged only if name or admin have changed.

If community name changed then:

1. community groups will be renamed according to new community name

If community admin changed then:

1. old community admin is removed from Community admin group
2. new community admin is added to Community admin group
3. new community admin is added to Community members

Suppose we rename the community from *Systers* to *Systers++*. Hence all the community groups will be renamed:

- from *Systers: Community Admin* to *Systers++: Community Admin*
- from *Systers: User and Content Manager* to *Systers++: User and Content Manager*
- from *Systers: Content Manager* to *Systers++: Content Manager*
- from *Systers: Content Contributor* to *Systers++: Content Contributor*

Suppose we change community admin from *Foo* user to *Bar* user. This is what is going to happen:

1. user *Foo* is removed from *Systers++: Community Admin* group
2. user *Bar* is added to *Systers++: Community Admin* group
3. user *Bar* is added to *Systers++* community members

2.4.3 Deleting a community

When a community is deleted, all the associated Groups are also deleted.

Suppose we delete the community named *Systers*. In this case the following groups will be deleted:

- *Systers: Community Admin*
- *Systers: User and Content Manager*
- *Systers: Content Manager*
- *Systers: Content Contributor*

2.5 Groups and Permissions

As it was previously discussed, each community is associated with 4 auth groups. Each group implies a specific set of permissions:

- **Community: Content Contributor** – a user from this group can:
  - add/change any tags and resource types
  - add/change Community news and resources
- **Community: Content Manager** – a user from this group can do everything a user from **Community: Content Contributor** can do, plus:
  - delete any tags or resource types
  - delete Community news or resources
  - add/change/delete a Community page
  - approve/delete comments to Community posts (news, resources)
- **Community: User and Content Manager** – a user from this group can do everything a user from **Community: Content Manager** can do, plus:
  - add/change/delete members of the Community

  - approve Community join requests

- **Community: Community Admin** – a user from this group can do everything a user from **Community: User and Content Manager** can do, plus:
  - edit Community profile

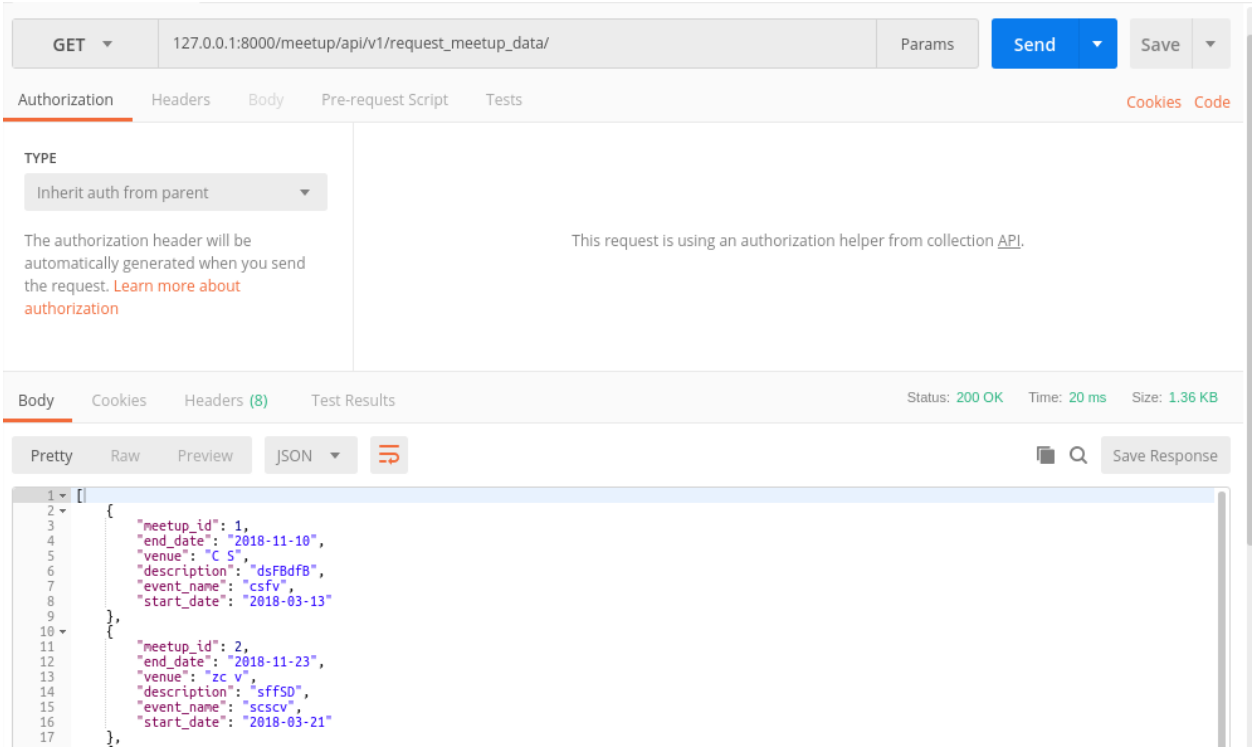
2.6 Api to communicate with VMS

The purpose of building this API was to provide information about the meetups in Portal to VMS so that their volunteers could participate in them. Presently, there are three fields of meetups which are sent to VMS via the API:

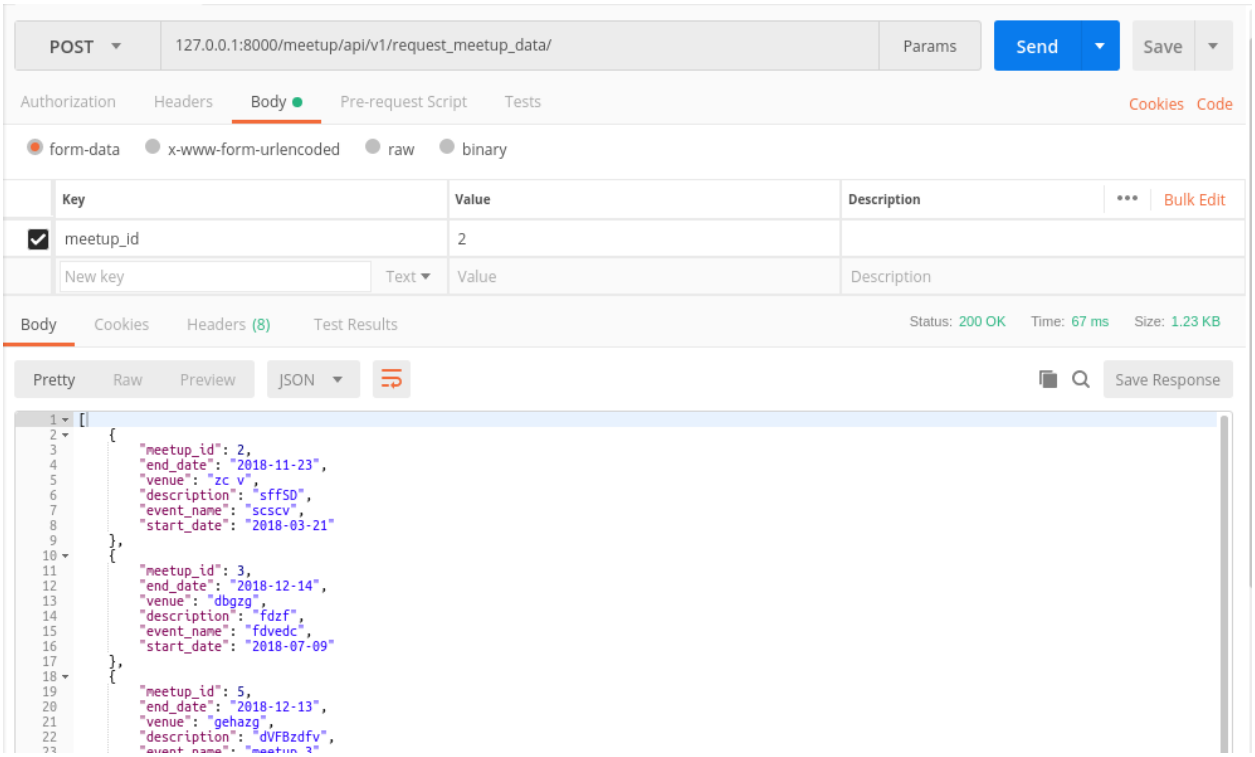
1. Event Name - Title of the event going to be held.
2. Start Date - The date from which the event would start.
3. End Date - The date on which the event would end.
4. Description - The description of the event.
5. Meetup Id - The unique Id of the meetup.
6. Venue - The location of the event.

Users can send in a `GET` or a `POST` request to https://localhost/meetup/api/v1/request_meetup_data/ to access the meetup data.

In case of a `GET` request, a list containing the details of all meetups in the ascending order of their dates will be returned :



In case of a `POST` request,a list containing the details of all meetups occurring after the ID value sent in the request object will be returned :



2.6.1 Details of the API

1. Use Case : Send event details for volunteers to contribute accordingly.
2. API Method : GET/POST
3. URL Parameters : `https://localhost/meetup/api/v1/request_meetup_data/`
4. Request Body:

```
{
  "Meetup-Id" : "id after which all meetups are required"
}
```

5. Response Body - Success:

```
{
  ""
  "Event Name/Title" : " --------"
  "Start Date" : "----------"
  "End Date" : "----------"
  "Description" : "----------"
  "Meetup Id" : "----------"
  "Venue" : "----------"
}
```

6. Response Body - Error:

```
{
  "message": "Please send a proper request"
}
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search