Mozillians Documentation

Release 0.1

Matthew Riley MacPherson, Giorgos Logiotatidis

Sep 21, 2018

Contents

Mozillians.org is the community phonebook for Mozilla. The instructions here are written for developers. If you're a user, visit the site for more information.

This application is built on top of Playdoh.

Contents:

CHAPTER 1

Installation

1.1 Docker Installation

Mozillians development environment can be installed using **docker**. This way we run Mozillians and all it's dependencies as docker containers. Here you can find more info about what docker is.

1.1.1 Dependencies

1. You need to install docker in your system. The installation guide covers many operating systems but for now we only support Linux and Mac OS X. *Version required*: 1.3.1 or newer.
2. We are using an orchestration tool for docker called docker-compose that helps us automate the procedure of initiating our docker containers required for development. Installation instructions can be found in Compose's documentation. *Version required*: 1.0.1 or newer.

Running Docker on Mac

Here are some notes regarding running Docker on Mac.

- Docker cannot run natively on Mac because it is based on a Linux kernel specific feature called LXC.
- When running docker in Mac via **boot2docker** you are running a lightweight Linux VM in Virtualbox that hosts the docker daemon and the LXC containers.
- We are running docker client in our host system that connects to the docker daemon inside boot2docker VM.
- We are using docker's *volume sharing* feature in order to share the source code with the Mozillians container. This is not directly supported in Mac. As a workaround boot2docker implements this feature by sharing the folder with Virtualbox first.
- The extra layer that we are adding using Virtualbox might cause some performance issues. This is a trade-off for having an easily reproducible stack without installing everything manually.

More information regarding boot2docker can be found in the documentation.

Here are some extra steps in order to run Mozillians on Mac:

1. Make sure *boot2docker* is initialized:

```
$ boot2docker init
```

2. Make sure *boot2docker* VM is up and running:

```
$ boot2docker up
```

3. Export *DOCKER_HOST* variables using the following command:

```
$ $(boot2docker shellinit)
```

Note: You need to make sure to run `$(boot2docker shellinit)` in each new shell you are using, or export it globally in order not to repeat this step every time you are working on mozillians.

1.1.2 Building mozillians

1. Fork the main Mozillians repository.
2. Clone your fork to your local machine:

```
$ git clone git@github.com:YOUR_USERNAME/mozillians.git mozillians
(lots of output - be patient...)
$ cd mozillians
```

3. Configure your local Mozillians installation:

```
$ cp mozillians/env-dist mozillians/.env
```

4. Start `MySQL` and `ElasticSearch` containers:

```
$ docker-compose up -d db es
```

5. Update the product details:

```
$ docker-compose run web python manage.py update_product_details -f
```

6. Import cities_light details:

```
$ docker-compose run web python manage.py cities_light
```

7. Create the database tables and run the migrations:

```
$ docker-compose run web python manage.py migrate --noinput
```

8. Load the timezone tables to MySQL:

```
$ docker-compose run db /bin/bash
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo/ | mysql -uroot -proot -h db_1␣
↪mysql
```

9. Create a superuser:

```
$ docker-compose run web python manage.py createsuperuser
```

10. Give permissions to the user:

```
$ docker-compose run web ./scripts/su.sh
```

11. Run mozillians:

```
docker-compose up
```

    (a) Load http://127.0.0.1:8000 or (for Mac users only) `<IP>:8000` where `<IP>` is the one returned by `boot2docker ip` command.

    (b) Sign in with persona to create your profile.

    (c) Stop the server with `Ctrl^C`.

    Note: In case this command doesn't work, you can run `./scripts/su.sh` inside the container. In order to get shell access please run `docker-compose run web /bin/bash`. To login to mozillians.org for dev purposes please visit http://127.0.0.1:8000/admin and login with the credentials issued in the previous step. The signin button doesn't work locally.

1.1.3 Running mozillians

1. Run Mozillians:

```
$ docker-compose up
(lots of output - be patient...)
```

2. Develop!

1.2 VirtualEnv Installation

Note: Installing Mozillians might be daunting. Ask for help using IRC in #commtools on irc.mozilla.org. Ping *giorgos*, *nemo-yiannis* or *tasos*, they will be happy to help.

1.2.1 Dependencies

Prerequisites: You'll need python2.7, python2.7-dev, virtualenv, pip, a C compiler (for building some of the Python packages, like the DB interface), mysqlclient and mysql-dev (or the equivalent on your system), a MySQL server, gettext, git, and lessc. Also, since we use elasticsearch, you will need a JAVA runtime environment.

There are almost certainly other requirements that we're so used to having installed we've forgotten we have them, so don't be shy about asking on IRC for help if you run into unexpected errors.

You will want a *nix box, ideally the latest versions of Debian or Ubuntu since that's what most of the core developers are using and it's most likely to work.

If you're on Ubuntu or Debian, you might start with:

```
$ sudo apt-get install build-essential git-core \
python2.7 python2.7-dev python-virtualenv python-pip \
gettext libjpeg-turbo8-dev \
mysql-client mysql-server libmysqlclient-dev default-jre \
libxslt2.1 libxslt1-dev libjpeg-dev zlib1g-dev libpng12-dev
```

Then install node and lessc (you only need node for `lessc`).

`nodejs` is not packaged for every distribution so we will not get into details as that would require different instructions for every distribution. You might want to take a look at nodejs github wiki. Just bare in mind that `lessc` must be installed after `nodejs`, since you have to use `npm`, the package manager of `nodejs`.

Note: Make sure your node version `node -v` is greater than v0.6.12 or there will be issues installing less.

When you want to start contributing...

1. Fork the main Mozillians repository (https://github.com/mozilla/mozillians) on GitHub.
2. Clone your fork to your local machine:

```
$ git clone git@github.com:YOUR_USERNAME/mozillians.git mozillians
(lots of output - be patient...)
$ cd mozillians
```

3. Create your python virtual environment:

```
$ virtualenv venv
```

4. Activate your python virtual environment:

```
$ source venv/bin/activate
(venv) $
```

Note: When you activate your python virtual environment, 'venv' (virtual environment's root directory name) will be prepended to your PS1.

5. Install development requirements:

```
(venv)$ python ./scripts/pipstrap.py
(venv)$ pip install --require-hashes --no-deps -r requirements/dev.txt
(lots more output - be patient again...)
(venv) $
```

Note: Since you are using a virtual environment, all the python packages you will install while the environment is active will be available only within this environment. Your system's python libraries will remain intact.

Note: Mac OS X users may see a problem when pip installs PIL. To correct that, install freetype, then do:

```
sudo ln -s /opt/local/include/freetype2 /opt/local/include/freetype
```

Once complete, re-run the pip install step to finish the installation.

6. Configure your local mozillians installation:

```
(venv)$ cp mozillians/env-dist mozillians/.env
```

   The provided configuration uses a MySQL database named *mozillians* and accesses it locally using the user *mozillians*.

7. Download ElasticSearch:

```
(venv)$ wget https://download.elasticsearch.org/elasticsearch/elasticsearch/
↪elasticsearch-2.4.5.tar.gz
(venv)$ tar zxf elasticsearch-2.4.5.tar.gz
```

   and run:

```
(venv)$ ./elasticsearch-2.4.5/bin/elasticsearch -d
```

   This will run the elasticsearch instance in the background.

1.2.2 MySQL setup

Setting up a MySQL user and database for development:

1. Install the MySQL server. Many Linux distributions provide an installable package. If your OS does not, you can find downloadable install packages on the MySQL site.
2. Start the mysql client program as the mysql root user:

```
$ mysql -u root -p
Enter password: ........
mysql>
```

3. Create a `mozillians` user:

```
mysql> create user 'mozillians'@'localhost';
```

4. Create a `mozillians` database:

```
mysql> create database mozillians character set utf8;
```

5. Give the mozillians user access to the mozillians database:

```
mysql> GRANT ALL PRIVILEGES ON mozillians.* TO "mozillians"@"localhost";
mysql> EXIT
Bye
$
```

6. Install timezone info tables in mysql:

```
(venv)$ mysql_tzinfo_to_sql /usr/share/zoneinfo/ | mysql -uroot -p mysql
```

1.2.3 Running Mozillians

1. Update product details:

```
(venv)$ ./manage.py update_product_details -f
```

2. Apply migrations:

```
(venv)$ ./manage.py migrate
```

3. Create user:

   (a) Run server:

   ```
   ./manage.py runserver 127.0.0.1:8000
   ```

   (b) Load http://127.0.0.1:8000 and sign in with Persona, then create your profile.

   (c) Stop the server with `Ctrl^C`.

   (d) Vouch your account and convert it to superuser:

   ```
   ./scripts/su.sh
   ```

4. Develop!

   Now you can start *contributing to Mozillians*.

5. When you're done:

   When you are done with your coding session, do not forget to kill the *elasticsearch* process and deactivate your virtual python environment by running:

```
(venv)$ deactivate
$
```

6. Next time:

   Next time, before starting you will need to activate your environment by typing:

```
$ . $VIRTUAL_ENV/bin/activate
```

   and start *elasticsearch* server again:

```
(venv)$ ./elasticsearch-2.4.5/bin/elasticsearch -d
```

Have fun!

CHAPTER 2

Testing

2.1 Testing Mozillians Code

- To run mozillians.org tests:

```
$ ./manage.py test
```

- If you need a fresh test database:

```
$ FORCE_DB=1 ./manage.py test
```

- To run all tests in a class:

```
$ ./manage.py test mozillians.users.tests.test_models:UserProfileTests
```

- If you want to run a single test:

```
$ ./manage.py test mozillians.users.tests.test_models:UserProfileTests.test_get_
↪attribute_with_public_level
```

  to run only *test_get_attribute_with_public_level* test from the *UserProfileTests* class in the *mozillians/users/tests/test_models.py* test file.

2.2 Test Coverage

You can combine *nose* testing with the *coverage* module to get the code coverage of the tests. To get a coverage report for the 'users' package run:

```
$ coverage run --omit='*migrations*' manage.py test --noinput
$ coverage xml --omit='*migrations*' $(find mozillians -name '*.py')
```

Then visit *htmlcov/index.html* to get the coverage results.

2.3 Test Cases for NDA renewal feature

Verify that the inviter, who is also a group curator is able to renew a user membership 2 weeks before it expires

Preconditions:

1. Create a closed group with terms in mozillians.org
2. Set the membership to expire after several days (>14)
3. Login as group curator and invite a mozillian to the closed group
4. Login as the mozillian who was invited by curator and accept the invitation to the group and the terms
   - The mozillian becomes a group member

Steps:

1. Verify that the group member will receive an email notification 2 weeks before their membership expires
2. Verify that the inviter will receive an email notification 2 weeks before user's membership expires
3. Login to mozillians.org as the inviter (and also group curator)
4. Navigate to group's page
5. Select "Renewals" option in filter dropdown, then click Filter button
6. Click "Confirm Request" for user's membership renewal request
7. Select "Members" option in filter dropdown, then click Filter button
8. Verify that the user added in precondition is displayed in Group members section

Expected: The user continues to be a member of the group, as the membership was renewed by inviter

Verify that a group curator is able to renew a user membership 2 weeks before it expires, if the inviter is no longer a curator

Preconditions:

1. Create a reviewed group with terms
2. Set the membership to expire after several days (>14)
3. Login as group curator and invite a mozillian to the closed group
4. Login as the mozillian who was invited by curator and accept the invitation to the group and the terms
   - The mozillian becomes a group member
5. Add a new curator to the group and remove the initial curator (who is also the inviter) from the group curators list

Steps:

1. Verify that the group member will receive an email notification 2 weeks before their membership expires
2. Verify that the group curator will receive an email notification 2 weeks before user's membership expires
3. Verify that the inviter will not receive an email notification 2 weeks before user's membership expires, as he/she is not a group curator anymore
4. Login to mozillians.org as the group curator

5. Navigate to group's page
6. Select "Renewals" option in filter dropdown, then click Filter button
7. Click "Confirm Request" for user's membership renewal request
8. Select "Members" option in filter dropdown, then click Filter button
9. Verify that the user added in precondition is displayed in Group members section

Expected: The user continues to be a member of the group, as the membership was renewed by curator

Verify the status of a user whose membership to an open group without terms reached expiration date

Preconditions:

1. Create an open group
2. Set the membership to expire after several days (>14)
3. Have a user who joined the group

Steps:

1. Login to mozillians.org as the curator of the open group, when the membership reached the expiration date
2. Navigate to group page
3. Select "Members" option in filter dropdown, then click Filter button
4. Verify that the user added in precondition is no longer displayed in Group members section (it was removed)

Expected: The user is not a group member. The user will receive an email "Removed from Mozillians group "X""

Verify the status of a user whose membership to a reviewed group without terms reached expiration date

Preconditions:

1. Create a reviewed group with no terms
2. Set the membership to expire after several days (>14)
3. Have a user added to the group

Steps:

1. Verify that the user will receive an email notification when the membership reached the expiration date ("Status changed for Mozillians group "X"")
2. Login to mozillians.org as the curator of the reviewed group, when the membership reached the expiration date
3. Navigate to group page
4. Select "Members" option in filter dropdown, then click Filter button
5. Verify that the user added in precondition is no longer displayed in Group members section
6. Select "Pending Members" option in filter dropdown, then click Filter button
7. Verify that the user added in precondition is displayed in Group members section
8. Click "Confirm Request" for that user
9. Select "Members" option in filter dropdown, then click Filter button
10. Verify that the user added in precondition is displayed in Group members section

Expected: After step 8: The user should receive an email saying "Accepted to Mozillians group "X"" After step 10: The user is a member of the group again

Verify the status of a user whose membership to a reviewed group with terms reached expiration date

Preconditions:

1. Create a reviewed group with terms
2. Set the membership to expire after several days (>14)
3. Have a user added to the group

Steps:

1. Verify that the user will receive an email notification when the membership reached the expiration date ("Status changed for Mozillians group "X"")
2. Login to mozillians.org as the curator of the reviewed group, when the membership reached the expiration date
3. Navigate to group page
4. Select "Members" option in filter dropdown, then click Filter button
5. Verify that the user added in precondition is no longer displayed in Group members section
6. Select "Pending Members" option in filter dropdown, then click Filter button
7. Verify that the user added in precondition is displayed in Group members section
8. Click "Confirm Request" for that user
9. Select "Pending Terms" option in filter dropdown, then click Filter button
10. Verify that the user added in precondition is displayed in Group members section

Expected: After step 8: The user should receive an email saying "Accepted to Mozillians group "X"" After step 10: If the user accepts the terms, he/she will be member of the group again

Verify the status of a user whose membership to a closed group without terms reached expiration date

Preconditions:

1. Create a closed group with no terms.
2. Set the membership to expire after several days (>14).
3. Have a user added to the group.

Steps:

1. Verify that the user will receive an email notification when the membership reached the expiration date ("Status changed for Mozillians group "X"")
2. Login to mozillians.org as the curator of the reviewed group, when the membership reached the expiration date
3. Navigate to group page
4. Select "Members" option in filter dropdown, then click Filter button
5. Verify that the user added in precondition is no longer displayed in Group members section
6. Select "Pending Members" option in filter dropdown, then click Filter button
7. Verify that the user added in precondition is displayed in Group members section
8. Click "Confirm Request" for that user

9. Select "Members" option in filter dropdown, then click Filter button
10. Verify that the user added in precondition is displayed in Group members section

Expected: After step 8: The user should receive an email saying "Accepted to Mozillians group "X"" After step 10: The user is a member of the group again

Verify the status of a user whose membership to a closed group with terms reached expiration date

Preconditions:

1. Create a closed group with terms
2. Set the membership to expire after several days (>14)
3. Have a user added to the group

Steps:

1. Verify that the user will receive an email notification when the membership reached the expiration date ("Status changed for Mozillians group "X"")
2. Login to mozillians.org as the curator of the closed group, when the membership reached the expiration date
3. Navigate to group page
4. Select "Members" option in filter dropdown, then click Filter button
5. Verify that the user added in precondition is no longer displayed in Group members section
6. Select "Pending Members" option in filter dropdown, then click Filter button
7. Verify that the user added in precondition is displayed in Group members section
8. Click "Confirm Request" for that user
9. Select "Pending Terms" option in filter dropdown, then click Filter button
10. Verify that the user added in precondition is displayed in Group members section

Expected: After step 8: The user should receive an email saying "Accepted to Mozillians group "X"" After step 10: If the user accepts the terms, he/she will be member of the group again

Verify the status of a user whose request to join a group (with membership set to expire) was never accepted by the curator

Precondition:

1. Create a reviewed group with no terms
2. Set the membership to expire in a few days
3. Have a user who submitted a request to join the group

Steps:

1. Login to mozillians.org as the curator of the group, when the membership reached the expiration date
2. Navigate to group page
3. Select "All" option in filter dropdown, then click Filter button
4. Verify that the user added in precondition is not displayed in Group members section (the user was removed)

Expected: The user is not a member of the group The user will receive an email "Removed from Mozillians group "X""

CHAPTER 3

Invitation System

Mozillians has an inivitation system that let's vouched users invite others to join Mozillians. These users who join are automatically vouched.

3.1 Inviting en-masse

Let's say you have a large list of contributors to invite to your phonebook, well we thought of that.

You can format a file (`myfriends.txt`) with one email address per line:

```
bob@thebobcats.com
juno@reactor.org
diane@hunters.org
```

And feed it on the admin node like so:

```
./manage.py cron invite myfriends.txt
```

And voila! Invitations will be mailed to your friends.

This creates one `Invite` and sets the `receiver` to `ZUUL`. This also sends an invitation email to each recipient.

CHAPTER 4

How to Contribute

Thank you for your interest in contributing to Mozillians! There are always bugs to file; bugs to fix in code; improvements to be made to the documentation; and more.

The below instructions are for software developers who want to work on Mozillians code.

4.1 Git workflow

When you want to start contributing, you should *follow the installation instructions*, then...

1. (Optional) Set your cloned fork to track upstream changes (changes to the main repository), then fetch and merge changes from the upstream branch:

```
$ git remote add --track master upstream git://github.com/mozilla/mozillians
$ git fetch upstream
$ git merge upstream/master
```

2. Set up a branch for a particular set of changes and switch to it:

```
$ git branch my_branch
$ git checkout my_branch
```

3. Commit changes to the code!
4. Code!
5. Lint the code:

```
$ flake8 mozillians
```

   and fix any errors.

6. Run the tests:

```
$ ./manage.py test
```

and make sure that all tests pass.

Learn more about *testing*.

7. Commit changes to the code!
8. When you're done, figure out how many commits you've made:

```
$ git log
```

9. Squash all those commits into a single commit that has a good git commit message. (Example assumes you made 4 commits):

```
$ git rebase -i HEAD~4
```

10. Use the interactive editor that pops up to pick/squash your commits:

```
pick 01d1239 [fix bug 893291] Make it go to 11
squash 32as32p added the library and made some minor changes
squash 30ame3z build the template
squash 91pcla8 ugh fix a semicolon bug in that last commit
```

11. Push your changes to your fork:

```
$ git push origin my_branch
```

12. Issue a pull request on GitHub
13. Wait to hear from one of the core developers

If you're asked to change your commit message, you can amend the message and force commit:

```
$ git commit --amend
$ git push -f origin my_branch
```

If you need more Git expertise, a good resource is the Git book.

4.2 Templates

Mozillians.org uses Jinja templates, which are similar to Django templates but have some differences.

Some helpers are available in all Jinja templates in Mozillians.org.

4.2.1 display_context

Return a marked-up chunk of content containing the items in the template context, if `settings.DEBUG` is True. Otherwise returns an empty string.

By default, callables are omitted. Pass include_callables=True to include them.

The format of the result is:

```
<dl class="jinja-context">
  <dt>key</dt><dd>value</dd>
  <dt>key</dt><dd>value</dd>
  ...
</dl>
```

`repr` is applied to the values to format them.

Example usage:

```
{{ display_context() }}

{{ display_context(include_callables=True) }}
```

4.2.2 get_context

Provide access to the Jinja `Context` object in case you want to do more complicated things with it. Typically, `display_context()` is easier to use.

If `settings.DEBUG` is not True, returns an empty dictionary.

Example usage:

```
{% set context=get_context() %}
{% for k, v in context|dictsort %}
    {% if not is_callable(v) %}
        {{ k }}: {{ v }}<br/>
    {% endif %}
{% endfor %}
```

4.2.3 is_callable

Return True if thing is callable.

See get_context() for example usage.

4.3 Server architecture

Stage

- *URL:* https://web-mozillians-staging.production.paas.mozilla.community/
- *Deploy:* Manual (Chief)

Production

- *URL:* http://www.mozillians.org/
- *Deploy:* Manual (Chief)

You can check the currently deployed git commit by checking https://www.mozillians.org/media/revision.txt.

4.4 Pushing to production

In 2013 Mozillians code is released on Thursdays, after QA and developers agree that code is ready to push to production. The list of code scheduled for any particular release is here: https://wiki.mozilla.org/Mozillians#Releases

4.5 What to work on

Mozillians development follows a schedule and a roadmap managed by the Mozillians product and development team. Bugs that the team has committed to work on are generally given a *target milestone* and are *assigned* to a developer. Other bugs are fair game; but they're not all aligned with the product's current evolution. So if you are not familiar with the project and its roadmap, you may want to find one of the core team in IRC and ask before working on a particular bug.

- All outstanding bugs
- Good first bugs
- Submit a bug

CHAPTER 5

Mozillians API

The Mozillians.org API is a REST API that provides detailed information about users and groups in Mozillians.org. This document explains how to use the Mozillians.org API to enhance your application.

> **Note:** All endpoints of the API require authentication. Users can restrict visibility of their personal data by editing their profiles.

5.1 Using API Data

The Mozillians.org API exposes personal data about people who have created profiles on Mozillians.org and who have chosen to expose that data to Mozilla's community or corporation sites. Applications that consume Mozillians.org API data must protect all data retrieved from the Mozillians.org API as specified by Mozilla's websites privacy policy. Furthermore, they must follow these guidelines:

1. **Do not store copies of Mozillians.org data.** If your application requires data served by the Mozillians.org API, it should request that data from the API, and should not make durable/permanent local copies of data retrieved from the API. Any exception to this rule requires a data safety review.
2. **Do not expose Mozillians.org data to an audience it was not intended for.** Mozillians.org data is visible, by default, to vouched members of Mozillians.org. Your application must not expose it to a wider audience unless specifically allowed by per-field privacy level or following a data safety review.
3. **Respect per-field privacy levels.** Certain fields retrieved from the Mozillians.org API may be subject to user-configured privacy levels. These privacy levels may be less restrictive than the default ("public") or more restrictive ("privileged"). *In future releases of the API*, a particular field's privacy level may accompany the field in the API response. Your application must respect and enforce any privacy level present in an API response.

If you believe an application is misusing Mozillians.org API data, please file a bug.

5.1.1 API v2

Getting an API Key

With API v2 we created a dedicated API management page for all API needs. You can review, create and delete your API v2 keys by accessing this page.

- Visit your edit profile page.
- Navigate to `Services > Developers > Manage API keys` or click here.

By default, vouched users are able to automatically get an API key with a `PUBLIC` access level. In order to get a key with elevated permissions you'll have to file a bug.

Privacy

API v2 is designed with users privacy in mind first. Each API key has a privacy level assigned to it. In order for the API consumer to access a `user/group/skill` detail, the key's privacy level should be greater or equal to the fields privacy value. Else the field value in the response is empty.

Authentication

API consumers should either provide the api key as a get parameter `api-key` or as an HTTP header `X-API-KEY`.

API Methods

Users

The `users` method of the *Mozillians API* returns user profile information.

Endpoint

```
https://mozillians.org/api/v2/users/
```

Parameters

`api-key` *Required* **string** - The application's API key

`is_vouched` *Optional* **string (True/False)** - Return only vouched/unvouched users

`username` *Optional* **string** - Return user with matching username

`full_name` *Optional* **string** - Return user with matching full name

`ircname` *Optional* **string** - Return user with matching ircname

`email` *Optional* **string** - Return user with matching primary/alternate email

`country` *Optional* **string** - Return users with matching country

`region` *Optional* **string** - Return users with matching region

`city` *Optional* **string** - Return users with matching city

`page` *Optional* **integer** - Return results contained in specific page

language *Optional* **string** - Return users speaking language matching language code

group *Optional* **string** - Return users who are members of given group name

skill *Optional* **string** - Return users with skill matching skill name

Return Codes

Code	Description
200:	OK Success!
403:	Wrong api-key or api-key not activated OR application not authorized

Examples

Filter user by email address:

Request:

```
/api/v2/users/?api-key=12345&email=test@example.com
```

Response:

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "username": "test@example.com",
            "is_vouched": true,
            "_url": "https://mozillians.org/api/v2/users/1111/"
        }
    ]
}
```

Get details for a user:

Request:

```
/api/v2/users/1111?api-key=12345
```

Response:

```
{
    "username": "test@example.com",
    "full_name": {
        "value": "Test Example",
        "privacy": "Public"
    },
    "email": {
        "value": "test@example.com",
        "privacy": "Mozillians"
    },
    "alternate_emails": [
        {
```

(continues on next page)

(continued from previous page)


```
            "email": "test2@example.com",
            "privacy": "Mozillians"
        }
    ],
    "bio": {
        "html": "<p>Bio test</p>",
        "value": "Bio test",
        "privacy": "Public"
    },
    "photo": {
        "privacy": "Public",
        "150x150": "https://mozillians.org/media/uploads/sorl-cache/00/f7/
↪00f760770a0bed60d936ee377788888.jpg",
        "500x500": "https://mozillians.org/media/uploads/sorl-cache/00/f7/
↪00f760770a0bed60d936ee377788888.jpg",
        "value": "https://mozillians.org/media/uploads/sorl-cache/00/f7/
↪00f760770a0bed60d936ee377788888.jpg",
        "300x300": "https://mozillians.org/media/uploads/sorl-cache/00/f7/
↪00f760770a0bed60d936ee377788888.jpg"
    },
    "ircname": {
        "value": "testexample",
        "privacy": "Public"
    },
    "date_mozillian": {
        "value": "2012-11-01",
        "privacy": "Public"
    },
    "timezone": {
        "utc_offset": 180,
        "value": "Europe/Athens",
        "privacy": "Public"
    },
    "title": {
        "value": "",
        "privacy": "Public"
    },
    "story_link": {
        "value": "",
        "privacy": "Public"
    },
    "languages": {
        "value": [
            {
                "code": "el",
                "english": "Greek",
                "native": "Ελληνικ"
            },
            {
                "code": "en",
                "english": "English",
                "native": "English"
            }
        ],
        "privacy": "Public"
    },
    "external_accounts": [],
```


(continues on next page)

(continued from previous page)


```
    "websites": [],
    "tshirt": {
        "privacy": "Privileged",
        "value": 9,
        "english": "Straight-cut Large"
    },
    "is_public": true,
    "is_vouched": true,
    "_url": "/api/v2/users/1111/",
    "url": "https://mozillians.org/en-US/u/testexample/",
    "city": {
        "value": "Buenos Aires",
        "privacy": "Public"
    },
    "region": {
        "value": "Ciudad de Buenos Aires",
        "privacy": "Public"
    },
    "country": {
        "code": "ar",
        "value": "Argentina",
        "privacy": "Public"
    }
}
```


Filter API responses:

By *country*:

```
/api/v2/users/?api-key=12345&country=Greece
```

By *ircname*:

```
/api/v2/users/?api-key=12345&ircname=mr_amazing
```

Groups

The `groups` method of the *Mozillians API* returns information about groups.

Endpoint

```
https://mozillians.org/api/v2/groups/
```

Parameters

`api-key` *Required* **string** - The application's API key

`name` *Optional* **string** - Return results matching the given name

Available filters **icontains** - Return results containing the given name

Example: /api/v2/groups?api-key=12345name__icontains=foo

`curator` *Optional* **integer** - Return results matching given mozillians id

functional_area *Optional* **True/False** - Return results containing only groups that are functional areas

members_can_leave *Optional* **True/False** - Return results containing groups with `members_can_leave` policy

accepting_new_members *Optional* **True/False** - Return results containing only groups with `accepting_new_members` policy

page *Optional* **integer** - Return results contained in specific page

Return Codes

Code	Description
200:	OK Success!
403:	Wrong api-key or api-key not activated OR application not authorized

Examples

Get groups:

Request:

```
/api/v2/groups?api-key=12345
```

Response:

```
{
      "count": 1628,
      "next": "https://mozillians.org/api/v2/groups/?page=2",
      "previous": null,
      "results": [
          {
              "id": 262,
              "url": "https://mozillians.org/en-US/group/airmozilla/",
              "name": "air mozilla",
              "member_count": 17,
              "_url": "https://mozillians.org/api/v2/groups/262/"
          },
          {
              "id": 12520,
              "url": "https://mozillians.org/en-US/group/air-mozilla-
↪contributors/",
              "name": "air mozilla contributors",
              "member_count": 11,
              "_url": "https://mozillians.org/api/v2/groups/12520/"
          },
          {
              "id": 11427,
              "url": "https://mozillians.org/en-US/group/alumni/",
              "name": "alumni",
              "member_count": 34,
              "_url": "https://mozillians.org/api/v2/groups/11427/"
          },
          {
```

(continues on next page)

(continued from previous page)

```
            "id": 12400,
            "url": "https://mozillians.org/en-US/group/amara/",
            "name": "amara",
            "member_count": 1,
            "_url": "https://mozillians.org/api/v2/groups/12400/"
        }
    ]
}
```

Get details for group having id 509:

```
/api/v2/groups/509/?api-key=12345
```

Skills

The `skills` method of the *Mozillians API* returns information about skills.

Endpoint

```
https://mozillians.org/api/v2/skills/
```

Parameters

`api-key` *Required* **string** - The application's API key

`page` *Optional* **integer** - Return results contained in specific page

`name` *Optional* **string** - Return results matching the given name

Available filters **icontains** - Return results containing the given name

Example: /api/v2/skills?api-key=12345name__icontains=foo

Return Codes

Code	Description
200:	OK Success!
403:	Wrong api-key or api-key not activated OR application not authorized

Examples

Get skills:

Request:

```
/api/v2/skills?api-key=12345
```

Response:

```
{
    "count": 7011,
    "next": "https://mozillians.org/api/v2/skills/?page=2",
    "previous": null,
    "results": [
        {
            "id": 6124,
            "url": "https://mozillians.org/en-US/skill/nodejs-3/",
            "name": ".nodejs",
            "member_count": 10,
            "_url": "https://mozillians.org/api/v2/skills/6124/"
        },
        {
            "id": 6162,
            "url": "https://mozillians.org/en-US/skill/php-3/",
            "name": ".php",
            "member_count": 91,
            "_url": "https://mozillians.org/api/v2/skills/6162/"
        },
        {
            "id": 5295,
            "url": "https://mozillians.org/en-US/skill/project-management-
↪marketing-fundamentals-logistic/",
            "name": ".project management .marketing fundamentals .logis",
            "member_count": 28,
            "_url": "https://mozillians.org/api/v2/skills/5295/"
        },
        {
            "id": 5415,
            "url": "https://mozillians.org/en-US/skill/0654598641/",
            "name": "0654598641",
            "member_count": 1,
            "_url": "https://mozillians.org/api/v2/skills/5415/"
        }
    ]
}
```


Get details for skill having id 509:

```
/api/v2/skills/509/?api-key=12345
```

CHAPTER 6

MySQL DB Anonymization

Mozillians uses the production database for testing on stage and dev. We provide a script to anonymize a database to remove some personal information for stage, and all personal information for dev.

1. Using the script:

```
$ cd scripts/mysql-anonymize
$ python anonymize.py anonymize_dev.yml > anon.sql
$ mysql < anon.sql
```

Note: Make sure your database is named `mozillians`. If it isn't, you can change the name in the .yml file you are using, it's clearly noted on the second line of each of the .yml configuration files.

CHAPTER 7

Internationalization

How to help with internationalization of Mozillians.org.

7.1 Installation

The message files used for translation are not in the same repository as the code, so if you are going to work on internationalization of Mozillians, you'll need to do a little more installation work.

1. Install *git* client
2. Clone the messages files repository under *locale* like this:

```
git clone https://github.com/mozilla-l10n/mozillians-l10n.git locale
```

Note: The directory in the git repository is named `locales` but it has to be checked out to a local directory named `locale`.

7.2 Working on internationalization

Having checked out the message files, you should be able to use the following instructions for Mozillians.org internationalization.

- For strings in python code we are using django's l10n functionality.
- For strings in jinja2 templates we are using Puente.

7.3 Managing Strings

Note: This section is for mozillians.org core developers. Other Mozillians *do not* have to do any of the following to contribute to mozillians.org.

7.3.1 Update Pontoon

When we commit new strings or we alter already existing strings in our codebase we need to perform a string merge to update Pontoon. Pontoon is the tool localizers use to translate mozillians.org strings.

Steps to follow to perform a string merge:

1. Update your local git repository:

```
cd locale
git checkout master
git pull origin master
```

2. String extract and merge:

```
./manage.py extract
./manage.py merge
```

3. Check the diff to make sure things look normal:

```
cd locale
git status
git diff
```

Note: Make sure things look normal. Changes in libraries (e.g. tower) can break things, like remove half of the strings.

4. Lint translations. See *Linting translations*.
5. Commit to git repository:

```
git commit -a -m "Mozillians string merge"
```

6. Push changes to *master* branch:

```
git push origin master
```

7. Optionally update production. See *Updating Production Translations*.

7.3.2 Linting translations

Sometimes translations have coding errors. Fortunately there is tool called dennis which will find all the errors.

1. Make sure you have dennis:

```
pip install dennis
```

2. Run dennis linter:

```
dennis-cmd lint locale
```

3. If dennis returns no errors or warnings your job is done. Otherwise continue reading.
4. Visit each file that dennis reports and locate the problematic translation:
   (a) Sometimes translations with variables are missing special characters. This can be easily fixed and you can do it. Here's an example:

   Here is the original, English string:

   ```
   msgid "Sorry, we cannot find any mozillians with skill %(name)s"
   ```

   and a incomplete Spanish translation:

   ```
   msgstr "Discúlpanos, pero no encontramos ningún mozillero en %(name)"
   ```

   The Spanish translation is missing a final *s* right after *%(name)*. The missing character is part of the variable definition and without it the template engine cannot parse the template.

   We fix the incomplete translation by adding the missing character.

   (b) If the translation needs attention from the translator we add *fuzzy* flag to the translation. This way we don't delete the broken translation but we instruct the template engine not to use it.

   For example for this translation:

   ```
   #: mozillians/templates/groups/skill.html:31
   msgid "Sorry, we cannot find any mozillians with skill %(name)s"
   msgstr "Something is wrong here"
   ```

   we add a line like this:

   ```
   #: mozillians/templates/groups/skill.html:31
   #, fuzzy
   msgid "Sorry, we cannot find any mozillians with skill %(name)s"
   msgstr "Something is wrong here"
   ```

7.3.3 Updating Production Translations

Production server https://mozillians.org checks out translations from the *production* branch instead of *master*.

1. Make sure that the translations in *master* have no errors. See *Linting translations*

> **Warning:** Translations with errors can bring (pages of the) website down. The template engine will fail to parse the strings and a 500 error will be returned to users. It is really important that translations copied to production are correct.

2. Checkout production branch if you don't have it already:

```
cd locale
git fetch origin
git checkout production
```

3. Merge current *master* into *production*:

```
git merge master
```

4. Verify that everything looks good:

```
git status
git diff
```

5. Commit merge to production branch:

```
git commit -a -m "Update mozillians production strings."
```

6. Push new strings to production branch:

```
git push origin production
```

7. Production will get the new translations on next push.

CHAPTER 8

Basket integration

In order to facilitate mass emailing and subscribing to Mozilla newsletters, Mozillians.org is integrated with Basket.

8.1 How does it work?

8.1.1 HTTP API calls

Basket exposes an HTTP API that allows consumers to interact with Mozilla's newsletters. Specifically we are using basket-client a Python implementation that makes it easier to integrate with django apps.

The Basket endpoints that Mozillians.org is using are the following:

- Lookup user
  - Retrieve user information based on their email
- Subscribe user
  - Subscribe user to the defined newsletters
- Unsubscribe user
  - Unsubscribe user from the defined newsletters

8.1.2 Mozillians.org newsletters

In Mozillians.org we maintain 2 newsletters

- `mozilla-phone` for all our vouched users
- `mozillians-nda` for all the members of the NDA group

8.1.3 Implementation architecture

In order to avoid blocking the user HTTP request/response cycle we are heavily using celery to make all the Basket API interactions asynchronous. That means that all Basket API calls are being done in the background and not necessarily in the exact time that an action triggered the API call.

Note: All Basket API related code is behind waffle switches. That means that in order to enable Basket integration, `BASKET_SWITCH_ENABLED` should be enabled. Same way you can disable all Basket API calls by disabling this switch.

Our celery tasks are implemented as chains of subtasks. This way we can easily re-use generic chunks of code and abort the chain of tasks in case something goes wrong. Here are our task definitions.

- `subscribe_user_to_basket(instance_id, newsletters=[])`
  - Lookup user in Basket
  - Based on the lookup results subscribe user to newsletters defined
- `unsubscribe_from_basket_task(email, newsletters=[])`
  - Lookup user in Basket
  - Based on the lookup results unsubscribe user from newsletters defined
- `update_email_in_basket(old_email, new_email)`
  - Lookup user's old email in Basket
  - Based on the lookup results unsubscribe old email from all the mozillians.org newsletters that user is subscribed
  - Subscribe new email to the newsletters defined above

8.1.4 Newsletter policies

When a Mozillian:

- becomes a member of the NDA group, we trigger a subscription to `mozillians-nda`.
- leaves the NDA group, we trigger an unsubscription from `mozillians-nda`.
- become vouched, we trigger a subscription to `mozilla-phone`.
- changes their primary email, we trigger an email change in basket.

8.1.5 Administrative actions

In order to allow Mozillians.org admins manage basket subscriptions we expose the following tasks as admin actions:

- `subscribe_user_to_basket`
- `unsubscribe_from_basket_task`

Note: There is no logic implemented behind these admin actions. That means that admins are explicitly allowed to subscribe/unsubscribe mozillians even when policies are not met.

8.1.6 Deployment details

All three Mozillians.org environments (dev/stage/prod) are Basket enabled. For development purposes, mozillians-dev and mozillians-stage are pointing to a sandboxed basket instance (basket-dev).

CHAPTER 9

Indices and tables

- genindex
- modindex
- search