culturemesh-api Documentation

Release beta

Stanford Code the Change

Jul 27, 2019

Table of Contents

The mission of CultureMesh is to connect diasporas around the world by providing a platform where users can find and connect with people from their place of origin or that speak their language.

The CultureMesh API is the single interface to CultureMesh's database and services. You can connect to the API to retrieve networks, posts, and more.

Note: The CultureMesh API is meant to be accessed by CultureMesh-owned services like the Android app and the lite webapp. It is NOT meant to be a publicly-facing developer API.

The CultureMesh API started as a Stanford Code the Change project.

CHAPTER 1

Getting Started

1.1 Overview

The CultureMesh API is a Python Flask application. It has no front-end and all of its blueprints are API endpoints that return or modify data from a MySQL database.

The API endpoints and data exchange formats are all implemented according to our Swagger API spec. We use this file as a structured specification format – we do not use Swagger's code generation features. You can visit https://editor.swagger.io/ and copy the Swagger spec file into the editor to see a nice UI for the API methods and objects.

Note: We currently have no way of running the API locally. We developed using a "rehearsal" database and URL endpoint.

Generating the required infrastructure for running and testing the API locally without relying on a live (even if rehearsal) site is a great idea for future work.

1.2 Running Locally

1. Clone the repository

   ```
   $ git clone https://github.com/Code-the-Change/culturemesh-api.git
   ```

2. Install python from https://python.org or via your favorite package manager
3. Install `virtualenv`

   ```
   $ pip3 install virtualenv
   ```

4. If you get a note from `pip` about `virtualenv` not being in your `PATH`, you need to perform this step. `PATH` is a variable accessible from any bash terminal you run, and it tells bash where to look for the commands you enter. It is a list of directories separated by `:`. You can see yours by running `echo $PATH`. To run

`virtualenv` commands, you need to add python's packages to your `PATH` by editing or creating the file `~/.bash_profile` on MacOS. To that file add the following lines:

```
PATH="<Path from pip message>:$PATH"
export PATH
```

5. Then you can install dependencies into a virtual environment

```
$ cd culturemesh-api
$ virtualenv .env
$ source .env/bin/activate
$ pip install -r requirements.txt
```

6. Create a local testing database. If you have access to the rehearsal database on the server, download it through the web interface instead of creating a new one.

7. Install `mysql` and start the local server. For example, you can use homebrew like so:

```
$ brew install mysql
```

8. The server by default has no root password, does not start automatically, and is only accessible over localhost. Start it, login, and create a new database.

```
$ mysql.server start
$ mysql -uroot
```

This will get you into a SQL command prompt, where you should run the following to create the database:

```
mysql> CREATE DATABASE <database_name>
```

Note: It would be great to simulate a mysql user with limited privileges, but unfortunately this is more difficult to get working locally. See the following links for information on how to do so.

https://www.digitalocean.com/community/tutorials/how-to-create-a-new-user-and-grant-permissions-in-mysql

https://stackoverflow.com/questions/22267114/python-mysqldb-error-1045-access-denied-for-user

https://stackoverflow.com/questions/6885164/pymysql-cant-connect-to-mysql-on-localhost

https://stackoverflow.com/questions/6562691/python-3-2-script-to-connect-to-local-mysql-database#6562701

Now you can load the downloaded file into the database.

```
mysql> USE <database_name>;
mysql> SOURCE <path_to_database_file>;
mysql> SHOW TABLES;
```

After the `SHOW TABLES;` command, you should see a bunch of tables listed.

9. Follow the instructions in *Secrets* to create the credentials file. Fill in the database information for the root user as follows (if you left the root user without a password):

```

```

sql = { 'MYSQL_DATABASE_USER': 'root', 'MYSQL_DATABASE_PASSWORD': None, 'MYSQL_DATABASE_DB': '<database_name>', 'MYSQL_DATABASE_HOST': 'localhost'

}

10. Now, you can run the API locally by executing `python run.py`. Then, send API requests (e.g. via Postman) to http://127.0.0.1:5000 as displayed in the output from running the python file.

1.3 Contributing

Note: Before contributing or writing code, be sure to scan the codebase first. There are certain recurring paradigms that you should follow. For example, we include utilities used by more than one blueprint as well as SQL statement helpers in *apiutils.py*, and we keep blueprint-specific utilities in *util.py* files in their corresponding directories.

Note: See section on Deployment for more information on how the API is is run.

All changes you make to the directory should go into a separate branch which you push and submit a pull request for:

1. Install dependencies

```
$ cd culturemesh-api
$ virtualenv .env
$ pip install -r requirements.txt
```

2. Create a new branch

```
$ git checkout -b my-new-branch
```

3. Make some awesome commits

4. Push the branch:

```
$ git push -u origin my-new-branch
```

5. Make sure there are no merge conflicts with master

6. Submit a pull request.

Warning: When opening the Pull Request choose the `alanefl` base fork, not `ericshong`'s

7. Select your reviewers

8. Wait until at least one other person submits a positive review (or make the requested changes). Once a positive review is submitted, you can merge the branch yourself from the GitHub website if your reviewer has not already done so.

9. Update your local master branch and delete the old one

```
$ git checkout master && git pull
$ git branch -d my-new-branch
```

1.3.1 Secrets

To run the API you need to write an `api/credentials.py` file that contains the following contents:

```
sql = {
    'MYSQL_DATABASE_USER': '<user>',
    'MYSQL_DATABASE_PASSWORD': '<password>',
    'MYSQL_DATABASE_DB': '<db>',
    'MYSQL_DATABASE_HOST': '<db host>'
}

api = {
    'key': '<api key>'
}

host_path = {
    'image_uploads': '<path to image uploads location>'
}

secret_key = "<secret key for auth>"
```

Contact Ken if you are interested in contributing.

CHAPTER 2

Deploying the API

We have been hosting the CultureMesh API on Bluehost on the same server as the main CultureMesh website. Bluehost uses Apache, so we use `flup` and Fast CGI in combination with a custom `.htaccess` file to serve the Flask endpoints.

We specify in `api.fcgi` that a Python 3 binary from a dedicated virtual environment is to be used to run the Flask application.

Here are the docs on Flup and Fast CGI.

Currently, there is no more action for deployment than installing the API source on the Bluehost server, creating the `credentials.py` file, and letting Apache, `flup`, and the Python 3 binary run and serve the API Flask application.

CHAPTER 3

Future Work

3.1 Testing

There is little to no formal testing on the API. Unit tests and integrations tests (and a Travis build that runs them) are high priority for future work.

3.2 Network Discovery

The API should provide endpoints for recommending Networks to a user.

3.3 Security

We use token-based authentication for some of the sensitive endpoints. This is still not good practice – something like Auth0 should be the end goal before the API is used to operate on sensitive data.

CHAPTER 4

Developer Notes

- We are currently using Python 3.6.1 to run the Flask app.
- This project along with the CultureMeshFFB, CultureMesh Android, and the original CultureMesh for web are all on GitHub.

CHAPTER 5

Contact

ken [at] culturemesh [dot] com