IXN Website, Events, SSIGs Documentation

Leo McArdle, Cameron Mcloughlin, Minjae Kang

Apr 28, 2018

Contents:

CHAPTER 1

Development Setup

The recommended development environment is Docker, but a Vagrant environment is also provided.

1.1 Cloning the repository

Both environments require you to:

1. Clone the repository: `git clone https://github.com/UCLComputerScience/103P_2018_team51.git`
2. Enter the cloned repository: `cd 103P_2018_team51`
3. Copy the *env-dist* file to *.env*: `cp env-dist .env`

1.2 Docker

Ensure you've installed Docker Compose by following the instructions here: https://docs.docker.com/compose/install/

Then, build the container with: `docker-compose build`.

Next, you'll need to run the database migrations, to set up the database. To do this:

1. Open a shell within the docker container: `docker-compose run web sh`
2. Within the container, run: `python3 manage.py migrate`

Now, close the container shell with `exit` (or open a new terminal window) and run the container with: `docker-compose up`.

You should now be able to access the site at: http://localhost:8000

1.2.1 Stopping the container

To stop the running container, go to the terminal it's running in and hit **Ctrl + C**. Docker will then gracefully bring the container down.

If the container is hanging, hit **Ctrl + C** again to kill it.

1.2.2 Updating the container

When pulling in new changes to the `requirements.txt` file from git, the container will need to be rebuilt.

To rebuild and bring the container up in one step run: `docker-compose up --build`.

1.2.3 Tips, tricks and oddities

Running Docker as root

If running `docker-compose` as root, as is recommended, then all files and directories created in the source directory within the container will be owned by root, and git will be unable to properly version control them. Run `sudo chown -R $USER:$USER .` outside the container to update their ownership to your user.

1.3 Vagrant

Download Vagrant and install it from here: https://www.vagrantup.com/downloads.html

You'll also need to install VirtualBox from here: https://www.virtualbox.org/wiki/Downloads

Bring the machine up with: `vagrant up`.

After the machine has finished booting, open a shell within it with: `vagrant ssh`.

Next, you'll want to navigate to the directory holding the project's source code with: `cd /vagrant`.

Here, run the database migrations to set up the database with `python3 manage.py migrate`, and bring the server up with `gulp`.

You should now be able to access the site at: http://localhost:8000

1.3.1 Stopping the machine

To stop the running machine:

1. Kill the server with: **Ctrl + C**
2. Exit from the machine's shell with: `exit`
3. Stop the machine with: `vagrant halt`

1.3.2 Updating the machine

When pulling in changes to the `requirements.txt` file from git, those new requirements will need to be installed in the machine.

Do this by running `pip3 install -r requirements.txt` from the `/vagrant` path within the machine.

1.3.3 Tips, tricks and oddities

Using Windows as a host

It seems that on Windows, the `up.sh` provisioning script isn't run on every `vagrant up`. This will usually be apparent if running `gulp` produces an error about it not being installed. This can be resolved by running `./up.sh` from the `/vagrant` path within the machine, every time after you bring it up.

It also seems that the correct database configuration options aren't set. Resolve this by updating your `.env` file to include:

```
DATABASE_HOST=localhost
DATABASE_USER=vagrant
DATABASE_PASSWORD=vagrant
```

1.4 Admin Interface

Create a superuser to access the admin interface at http://localhost:8000/admin.

Do this by running `python3 manage.py createsuperuser` within your development environment.

UPI stands for Unique Person Identifier, a unique id given to every member of UCL, set this to your own UPI (found on your UCL ID card) if you want to be able to log into the admin interface through UCL API OAuth.

1.5 Authentication

The project makes use of UCL API OAuth for authentication.

1.5.1 Reverse Proxy

UCL API prohibits setting localhost as as callback URL, so you'll need to set up a reverse proxy to access your local development server through a remote url.

One solution is localtunnel, which can be used by following the instructions here: https://localtunnel.github.io/www/.

1.5.2 OAuth Credentials

Generate your credentials by following the instructions here: *UCL API OAuth Credentials*.

Then update your `.env` file to include the *Client ID* and *Client Secret* from the UCL API dashboard, for example:

```
UCLAPI_CLIENT_ID=0123456789.0123456789
UCLAPI_CLIENT_SECRET=0123456789abcdef
```

Test you've setup your credentials correctly by attempting to log in by visiting `/auth`.

1.6 Google Maps API Key

Generate your credentials by following the instructions here: *Google Maps API Key*.

Then update your `.env` file to include the API key:

```
GOOGLE_MAPS_KEY=0123456789abcdef
```

CHAPTER 2

Development Methodology

2.1 Git Workflow

2.1.1 Fetching the latest changes

The latest changes can be fetched with: `git fetch origin`.

Using `git fetch` is preferred over `git pull` as it means there's no risk of modifying the branch you're currently on - it gives you full control over what you want to do with the changes you've fetched.

2.1.2 Creating a feature branch

Create a new branch to work on your feature with the latest commits from the master branch with: `git branch -b feature-name origin/master`

2.1.3 Committing the feature

Once you've finished working on your feature, you'll want to commit it to your branch. Do this by:

1. Checking the current status of your files with: `git status`
2. Staging all the changes to files you want to commit with: `git add file_one file_two`
3. Committing the staged changes with a descriptive commit message: `git commit -m "commit message"`

If you've been working on a feature linked to an issue, which is usually the case, you'll want to end the commit message with the issue number. For example, if I'd been working on issue 7, my commit command might look something like: `git commit -m "commit message #7"`.

2.1.4 Updating the branch with the latest changes

While you've been working on your feature, there may have been updates to the master branch which you'll want to include in your feature branch. Do this by:

1. Fetching the latest changes with: `git fetch origin`
2. Rebasing your branch with the changes with: `git rebase origin/master`

At this point the rebase may succeed, or it may fail. If it fails run `git status` and follow the instructions to resolve the merge conflicts.

2.1.5 Pushing the branch

After committing your feature and rebasing with master, push the branch to the origin remote with: `git push origin feature-name`.

2.1.6 Creating a pull request

Now that your feature branch is on the origin remote, go to https://github.com/UCLComputerScience/103P_2018_team51/compare/maste
name and click **Create pull request** to create a pull request.

Your commit will be reviewed, and if approved, rebased into the master branch.

If changes were requested, you can make them in your local branch, commit and push to the remote branch as before.

CHAPTER 3

API Setup

3.1 UCL API OAuth Credentials

Create a new app at: https://uclapi.com/dashboard/.

Then fill in the OAuth Callback URL to be the remote url of your server, followed by `/auth/callback`. If using a localtunnel development server, this will be something like:

```
https://abcdefghij.localtunnel.me/auth/callback
```

Test you've setup your credentials correctly by attempting to log in by visiting `/auth`.

3.2 Google Maps API Key

A Google Maps API key is necessary for displaying the maps on event pages.

To get your key:

1. Visit https://console.developers.google.com/cloud-resource-manager and sign in
2. Click "Create a Project"
3. Give your project a name, perhaps "SSIG site dev"
4. Click "Create"
5. Click on your newly created project
6. Visit https://console.developers.google.com/apis/api/maps-backend.googleapis.com/overview
7. Click "Enable", wait for the API to be enabled for your project
8. Visit https://console.developers.google.com/apis/credentials/wizard?api=maps-backend.googleapis.com
9. Click "What credentials do I need?"
10. Your API key will be displayed

11. Click “Restrict key” to restrict with what sites and APIs the key can be used (recommended in production)

CHAPTER 4

Azure Deployment Manual

4.1 PostgreSQL server

1. Create a new PostgreSQL server: https://portal.azure.com/#create/Microsoft.PostgreSQLServer
2. Open the configuration page for the PostgreSQL server you've created
3. Navigate to "Connection Security" under "Settings" in the sidebar
4. Toggle "Allow access to Azure services" to "On"
5. Click "Add My IP"
6. Click "Save"

4.2 Create Web App

1. Create a new Web App in the same resource group as your PostgreSQL server: https://portal.azure.com/#create/Microsoft.WebSite
2. Open the configuration page for the Web App you've created
3. Navigate to "Extensions" under "Development Tools in" the sidebar
4. Click "Add", and install "Python 3.6.4 x86"

4.3 Configure Web App

Now we need to provide the Web App with some settings to run:

1. Navigate to "Application settings" under "Settings" on the sidebar.
2. Scroll down to "Application settings" and click "Add new setting" for each of the following:

Name	Value
DATABASE_NAME	postgresql
DATABASE_USER	*enter the postgresql username you previously created*
DATABASE_PASSWORD	*enter the postgresql password you previously created*
DATABASE_HOST	*enter the domain of the postgresql server you previously created*
SECRET_KEY	*randomly generate a string and enter it here*
ALLOWED_HOSTS	*enter the domain of the web app you've created*
UCLAPI_CLIENT_ID	*create UCL API OAuth Credentials and enter the client id here*
UCLAPI_CLIENT_SECRET	*create UCL API OAuth Credentials and enter the client secret here*
GOOGLE_MAPS_KEY	*create a Google Maps API Key and enter it here*

Finally click "Save".

4.4 Create Deployment User

Follow the documentation here: https://docs.microsoft.com/en-gb/azure/app-service/app-service-deployment-credentials

4.5 Deploy Web App

1. Clone the repository: `git clone https://github.com/UCLComputerScience/103P_2018_team51.git`
2. Enter the cloned repository: `cd 103P_2018_team51`
3. Navigate to your Web App's overview page
4. Create a new remote from the "Git clone url" on the Web App Overview page: `git remote add azure <paste "Git clone url" here>`
5. Deploy the app to azure with: `git push azure master`
6. Enter the previously created deployment credentials when prompted

4.6 Create superuser

Run the `python3 manage.py createsuperuser` command locally by setting the database settings on the command line:

```
DATABASE_NAME=postgresql DATABASE_USER=<enter value> DATABASE_PASSWORD=<enter value>␣
↪DATABASE_HOST=<enter value> python3 manage.py createsuperuser
```

UPI stands for Unique Person Identifier, a unique id given to every member of UCL, set this to your own UPI (found on your UCL ID card) if you want to be able to log into the admin interface through UCL API OAuth.

You can now log into the admin interface at: https://your-domain/admin/login