Design Project Group 4: Database Manager

Overview

This project serves as a database manager for the Data Science department. With the increase in students; it is no longer feasible to cut database credentials on paper. As such, this software will allow students to request databases with pre-loaded schemas from a certain course.

Requirements

Firstly, the software requires full administrative access to a PostgreSQL server. This server does not have to be hosted on the same machine. It is recommended that the PostgreSQL databases not be used for any other purpose than this software.

Install dependencies:

- python3-pip
- npm

Run dependencies:

- python3.6 or newer
- python3-venv
- python3-dev
- libpq-dev
- postgresql-client-common
- postgresql-client-[VERSION] (we use version 10)

Additionally, the following software is needed for the recommended production server setup:

- uWSGI (please install via pip3)
- a web server (e.g. nginx or apache2)
- an SMTP server (e.g. postfix)
- libapache2-mod-proxy-uwsgi (when using apache2)

Any pip and npm packages will be installed automatically during the installation steps

The Database Server

This service requires a PostgreSQL server to connect to. This server SHOULD not be used for any other purposes but this program, as that might cause unexpected behaviour. It requires a single superuser on this server. Importing the schema into this database will be done in a later step, when `install.sh` is ran

Settings

There are two kinds of settings: Public settings and private settings. The former can simply be accessed from `designproject/settings.py`, and are as follows. This file contains many settings that do not need to be edited, but some do:

- `DEBUG`: Whether the server is in debug mode. Set to `True` on development, and to `False` on production
- `EMAIL_SERVER` and `EMAIL_SENDER`: will be discussed under the "email" heading below.
- `DATABASE_SERVER`: the database server; this will be displayed to students as the server they should connect to
- `HOST_SERVER`: the domain that this service is hosted on. Used for links sent in emails
- `LOGFILE`: the file that the server logs to. The server must have write permissions on this file, or it will crash

Private settings are stored in the file `secret.py`.

There MUST exist a file `designproject/secret.py`. This file is not included in this repository for obvious reasons. The repository does, however, include secret.py.example, which can be used as a template.

The file MUST be a correct python file, and SHOULD contain nothing but the following variable definitions, all of which MUST be strings:

```
db_name = [DATABASE NAME]
db_user = [DATABASE USER]
db_password = [DATABASE PASSWORD]
db_host = [DATABASE SERVER]
db_port = [DATABASE PORT]

bitmask = [33 BYTES OF CRYPTORANDOM DATA, AS BASE64]
```

If the file does not exist, parse, or assign appropriate values to all of these variables, the program will crash.

To create the required 33 bits of data, you could use the command `head -c 33 /dev/urandom | base64`.

The database user MUST exist, and MUST have superuser privileges on the database server. The database server MUST run Postgres, and MUST be available from wherever this software is running. The database name SHOULD be something memorable and SHOULD NOT end on a number. Please ensure that there are no databases created outside of this program that use the database prefix

Installation and running for development (SKIP IF PRODUCTION)

To install the server for development, simply run init.sh. This will create a python virtual environment, and install the required python and node packages within it.

To run the server, simply run start.sh. This will compile the typescript, move static files to their correct location, and finally start a development server on port 8000. Note that this server can only be approached from the local host.

NB: In development mode, emails will be printed to the standard output instead of being sent as email

WARNING: do not use this way of setting up a server in production. It is not intended for high-demand environments, and using it in production may lead to terrible performance and security risks.

Installation in a Production environment.

Installing a django system in a production environment is somewhat less trivial than it may seem. Correct configuration requires the installation of several specific purpose components, each of which need to be connected to the others.

Because of previous experience, we have chosen to use Nginx as a webserver, and PostFix as an SMPT server. Alternatives could also be used with no impact on the functioning of this software, as long as they are configured in similar ways.

The first thing to do would be to install this project and its dependencies, and configure them for use in production. To do so, clone this repository into a location of your choice (e.g. /var/www/db/, run `init.sh`, which will set up a virtual environment, and `production.sh`, which will configure the front-end frameworks for production use.

For the basic configuration of uWSGI, Nginx, and this Django project, look no further than this excellent tutorial from the uWSGI website. Note that the uwsgi config file is already present in this project under the name of `db_uwsgi.ini`; this is usable for most systems, although you will probably want to edit the file paths.

In most environments, the default python version is lower than Python 3.7; in fact, it is often still Python 2. To ensure this software is ran using Python 3.7, a plugin for uWSGI must be installed. To install this plugin, approximately follow this tutorial. `db_uwsgi.ini` is already configured to use this plugin.

At this point, you should have set up the following.

- A basic Nginx configuration
- A uWSGI server in emperor mode
- A vassal to said server for this project.

This means that you should get a basic page from this software when requesting to `<server>:8000` over http.

You should now look at running this software as an actual website. For security reasons, the web server MUST be configured to use SSL, and only allow SSL connections to this software.

All static files are in `<project root>/static`. As such, the /media location from the tutorial can be removed.

Finally, email should be setup according to the instructions in the paragraph on it below

Installation under Apache

This tutorial will outline the steps required to install the software under Apache. It is intended for services that already have an Apache server running other websites, with a system administrator who is somewhat experienced with Apache httpd and whatever operating system they choose to use. Note that while in principle all operating systems are supported, in practice we have assumed the project will be ran in a UNIX-like environment. If you want to set up the system on a standalone server, we recommend the above tutorial for nginx.

Note that this tutorial will use the somewhat newer `mod_proxy_uwsgi`, instead of the older, and much uglier, `mod_wsgi`. This mod is NOT CONSIDERED STABLE UNLESS uWSGI is at least version 2.0.6, and Apache is at least version 2.4.9. If your system does not provide these versions, or you have an exceptionally high need for stability, we recommend the usage of the older `mod_wsgi` instead.

Throughout this tutorial, `[DOCUMENT ROOT]` will be used to indicate the absolute path to the directory in which you cloned this project. Standard practice is putting it somewhere in `/var/www/`

Please make sure all the files are owned by `www-data`, by running `sudo chown www-data:www-data [DOCUMENT ROOT]`

1. Install all the dependencies listed above. Take care to install uwsgi as a pip3 package, and not via the system dependencies (do not use the venv for this).
2. If your distribution does not come with Python3.6 or later, compile Python3.7 from source, then follow this tutorial to ensure uwsgi uses it. Also edit install.sh: the first line, `python3 -m venv venv` MUST use your new python3.7 installation.
3. Follow the steps in the paragraph on settings above. Take special care to create `secret.py` correctly.
4. Run install.sh. This will create a virtual environment, install the dependencies for the project in it, and also migrate the database.
5. Copy/rename uwsgi.ini.example to uwsgi.ini. Edit the values in uwsgi.ini (mostly filepaths) to correspond to your system. There are comments

explaining what each line is supposed to do, so it should be pretty self-explanatory. Note that most file paths are absolute. The socket file can be at any location of your choice; the "good practice" is to put it in /var/run, but as this requires permissions, the current default file puts it in /tmp instead. If you get errors related to socket.c, that probably means that you do not have file permissions on the socket file.

6. To test if you have correctly configured uwsgi, run `sudo -u www-data uwsgi --ini uwsgi.ini --http :8000`, which should now host the website on port 8000; you should see a login page once you go there. Note that static files, such as the CSS for the page, are not yet loaded in at this stage.
7. Now that we know uwsgi works, we can configure it as a service. Make the service `dab` by running (as root) `cp dab.service /etc/systemd/system ; systemctl start dab.service`.
8. If the previous step succeeded, we can start configuring apache. We require the modules `proxy` and `proxy_uwsgi` to be enabled and available to the rest of the configuration settings. The easiest way to do this in modern GNU/Linux package of Apache is to run `a2enmod proxy_uwsgi`. We assume you want to run the program under `/dab`. Unfortunately, you will also have to reserve `/static`.

To install, simply add the following to your Apache2 config file (either `apache.conf` or `sites-enabled/000-default.conf`, whichever works).

```
Alias /static/ "[DOCUMENT ROOT]/static"

<Directory "[DOCUMENT ROOT]/static">
    Require all granted
</Directory>
<Location /dab>
    ProxyPass unix:/data/dab/dab.sock|uwsgi://uwsgi-uds-design/
    ProxyPassReverse unix:/data/dab/dab.sock|uwsgi://uwsgi-uds-design/
</Location>
```

If you have changed the location of the socket in the previous steps, you must also change it here.

Please ensure that Apache2 has read access to the files in `[DOCUMENT ROOT]/static`. If this is not the case, CSS, Javascript, and media files can't be loaded.

Congratulations! If you run apache, you should now have the basic web page set up. You can test this by trying to access the page, at whatever port Apache runs on. There are now three things left to do:

This project MUST run over https. It is the responsibility of the system administrator to ensure there are proper SSL certificates present. After this has been arranged, `SSLRequireSSL` MUST be added to the Apache configuration of this site, probably in the `<Location /dab>` tag, to ensure that only SSL

connections will be accepted. Failure to comply with these instructions will cause passwords to be transmitted over unencrypted connections.

The second thing is email, described in the section below.

Finally, the project is rather useless without an administrator. To bootstrap one from a project with an empty database, we recommend the following procedure:

1. Register a student user via the regular register page
2. Run the file `promote.sh` in the document root, which will promote the user to admin given the email

The user should now be an administrator, who can create other administrators and teachers.

Email

This project sends email to confirm the users are somehow linked to the University of Twente. To ensure this email is sent properly, the following steps must be taken.

1. The setting for `EMAIL_SERVER` must be correct. If the server is on campus, you could simply point it to `smtp.utwente.nl`; however, we recommend hosting an SMTP server locally, in accordance with instructions below
2. The setting for `EMAIL_SENDER` must use a domain that is actually assigned to the hosting server
3. The SPF records for the domain must designate either the hosting server or `smtp.utwente.nl` as a permitted sender, preferably both. Failure to comply with this instruction will result in email appearing in the users spam folder.

Locally hosting an SMTP server

While it is possible to use `smtp.utwente.nl` directly, email is rather complicated, and the chances of the email correctly arriving in the users inbox increase when using a local SMTP server that relays to `smtp.utwente.nl` instead. This email server only needs to relay messages from the local host, coming in over a loopback connection. If you do not have one installed, we recommend installing Postfix using the package manager. On the UT campus, the only edit that needs to be made to Postfix is the following line in `/etc/postfix/main.cf`:

```
relayhost = smtp.utwente.nl
```

Once this is configured, the server should be able to create email.

Once finished, please edit some of the settings in designproject/settings.py as they will affect what the address of the database is that will be displayed to users