Set up Roaster (development)

Please see Set up Roaster (production) below for a production setup.

Prerequisites

- Go ≥ v1.11
- NodeJS w/ npm
- Docker

Initial setup

We use PostgreSQL and Redis as our databases. The simplest way to set them up is using the official Docker images.

Redis

Run the Redis Docker image with (exposed at port: `6379`, non-persistent, w/o password):

```
docker run -it --rm -p 6379:6379 --name roaster-redis -d redis
```

PostgreSQL

Run the PostgreSQL Docker image with (exposed at port: `5432`), replace `<db password>` with any password you want:

```
docker run --name roaster-postgresql -e POSTGRES_PASSWORD=<db password> -p
5432:5432 -d postgres
```

The PostgreSQL database can be configured with pgAdmin4:

```
docker pull dpage/pgadmin4
docker run -p 80:80 \
    --name roaster-pgadmin4 \
    -e "PGADMIN_DEFAULT_EMAIL=user@domain.com" \
    -e "PGADMIN_DEFAULT_PASSWORD=SuperSecret" \
    -d dpage/pgadmin4 \
    --link roaster-pgadmin4:roaster-postgresql
```

Access the pgAdmin4 website at: http://localhost

Note: pgAdmin4 for Windows users

> If you are using Windows, the easiest thing to do is to install pgAdmin4 through the installer on their website. Start the application, choose new server, under the connection tab, type the IP that is shown when running the command `docker-machine ip` in your

Docker Shell. Use the port `5432`, name the server whatever you like, and the password is the one you set in the first step in this guide.

The PostgreSQL database should be setup with a new database with a empty schema called `roaster` and a user who only has privilege to that schema. The schema will be automatically generated by our forward engineering code in the Go server.

Roaster repository

Clone the Roaster repository:

```
git clone git@github.com:LuleaUniversityOfTechnology/2018-project-roaster.git
```

And enter the directory:

```
cd 2018-project-roaster
```

Backend (roasterd)

The backend is written in Go and therefore requires that Go is installed correctly on your machine. Also, a version ≥ v1.11 is required for Go modules which is used in this project.

Make sure you enable Go modules if you are using Go v1.11:

```
export GO111MODULE=on
```

Requirements (see Initial setup above)

- A PostgreSQL database must be set up with a scheme named `roaster` that the provided user has access to.
- A Redis server on localhost without any password listening on port `6379`.

Start web server (development mode)

To run the `roasterd` server in development mode, simply run:

```
go run github.com/LuleaUniversityOfTechnology/2018-project-roaster/cmd/roasterd \
    -dev-mode \
    -database-source="user=<db user> dbname=<db name> password=<db password>
sslmode=disable"
```

Note, if running on Windows, you might need additional flags. Such as the `host` flag for the PostgreSQL database, use the same IP as above for Windows (run `docker-machine ip` in your Docker Shell to show it). The Redis database might also need the `-redis-address=<ip from`

`docker-machine command>:6379` flag. If you encounter problems use the `-help` flag to see other options.

Frontend (roasterc)

The frontend is written in TypeScript, HTML and CSS. Everything is packaged and compiled using webpack.

First you have to be located in the `www/` folder:

```
cd www/
```

Then, install the required dependencies with:

```
npm install
```

Finally, start the frontend autobuild with:

```
npm start
```

Now, everytime you make any change to the frontend, everything will automatically recompile and can be accessed from: `http://localhost:5000` (hosted by the `roasterd` backend).

Setup Roaster (production)

Prerequisites

- Go ≥ v1.11
- NodeJS w/ npm
- Docker
- Host running Linux

Build production

The `tool/build-production.sh` script creates a `build/` folder with all the required binaries and data to run a production ready variant of Roaster.

Run the script with:

```
./tool/build-production.sh
```

When the script is done, enter the `build/` folder:

```
cd build
```

Configure and start backend (roasterd)

Requirements (see Initial setup above in developer guide)

- A PostgreSQL database must be set up with a scheme named `roaster` that the provided user has access to.
- A Redis server on localhost without any password listening on port `6379`.

Configure secret keys

The `roasterd` server requires three cryptographically secure random keys. They can be generated and set to their environment variables with:

```
export SESSION_BLOCK_KEY=$(LC_ALL=C tr -dc '[:alnum:]' < /dev/urandom | head -c32)
export SESSION_HASH_KEY=$(LC_ALL=C tr -dc '[:alnum:]' < /dev/urandom | head -c32)
export CSRF_KEY=$(LC_ALL=C tr -dc '[:alnum:]' < /dev/urandom | head -c32)
```

This will produce three unique keys with 32 bytes of random characters. Make sure that you do **NOT** use the same key for any of the environment variables. The keys are **REQUIRED** to be unique for the cryptography to work securely.

Start web server (production mode)

Run the `roasterd` server with:

```
./roasterd \
    -database-source="user=<db user> dbname=<db name> password=<db password>
sslmode=disable"
```

If you do not set the environment variables above you can expect a crash from the server telling you that you do not have enough amount of bytes for the keys.

A note about `sslmode`

The `sslmode` key can be set to one of:

- `require` - Use SSL/TLS w/o verification.
- `verify_ca` - Verify CA for SSL/TLS, but not the hostname.
- `verify_full` - Verify both CA and hostname for SSL/TLS.
- `disable` - No SSL/TLS.

Don't forget to use a reverse proxy in front of the web server that uses HTTPS/TLS.