Deployment Guide

System Requirement

This deployment guide has been validated on a Duke VCM running Ubuntu 20.04.3 LTS.

Install development environment

Install *Node.js* via *nvm*

Install *nvm* and adds it to environment path

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.1/install.
sh | bash
export NVM_DIR="$([ -z "${XDG_CONFIG_HOME-}" ] && printf %s "${HOME}/.
nvm" || printf %s "${XDG_CONFIG_HOME}/nvm")"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

Install *Node 16*

```
nvm install 16
nvm use 16
```

Check current *Node* version with

```
node -v
```

Install *Nodejs* package manager *npm*

```
sudo apt update
sudo apt upgrade -y
sudo apt install npm -y
npm install npm@latest -g
```

Check current *npm* version with

```
npm -v
```

Install and configure *PostgreSQL* Database

```
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt
$(lsb_release -cs)-pgdg main" > /etc/apt/sources.list.d/pgdg.list'
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc |
sudo apt-key add -
sudo apt-get update
sudo apt-get -y install postgresql
```

PostgreSQL needs configuration. First, change password for user ‘postgres’

```
sudo passwd postgres
```

Now we can change user with the password we just set

```
su postgres
```

Access the database

```
psql
```

You can change the password used for user ‘postgres’ to access the database. It is different from the password we just set for Linux user 'postgres'

```
ALTER USER postgres PASSWORD '<new password>';
```

Create the database we need for this software

```
create database bus;
```

Install Queue Manager *RabbitMQ*

RabbitMQ is used as the queue management package for sending emails. Follow the instructions here to install it: `https://www.rabbitmq.com/install-debian.html#apt-quick-start-cloudsmith`

After installing, start the server by

```
sudo systemctl start rabbitmq-server
```

Install *Redis*

We use *bull* to schedule retrieving bus information from TransitTraq, a sketchy real-time bus location tracker. *Bull* depends on *Redis.* Follow the installation guide on *Redis'* website to install *Redis.* Currently, these commands are

```
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr
/share/keyrings/redis-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg]
https://packages.redis.io/deb $(lsb_release -cs) main" | sudo tee /etc
/apt/sources.list.d/redis.list
sudo apt-get update
sudo apt-get install redis
```

Configure deployment environment

Set `NODE_ENV` to `production`

```
export NODE_ENV=production
```

add this line to `~/.bashrc` to make sure `NODE_ENV` is set on logging in

Configure RabbitMQ to start on boot

```
sudo systemctl enable rabbitmq-server
```

Installing Process Manager pm2

```
npm install -g pm2
```

Configure pm2 to start on boot. This ensures that the website will still be online even the server restarts

```
pm2 startup systemd
```

And copy-paste the start-up script given

To make sure pm2 starts later than RabbitMQ, we need to edit `/etc/systemd/system/pm2-<username>.service`

Add a line `After=rabbitmq-server.service` under `[Unit]`

Allow Nodejs to bind ports under 1024 with setcap

```
sudo apt-get install libcap2-bin
sudo setcap cap_net_bind_service=+ep `which node`
```

Install SSL certificate

In short, go to certbot and follow the instructions given. To give a snapshot:

```
sudo apt -y install snapd
sudo snap install core; sudo snap refresh core
sudo snap install --classic certbot
sudo ln -s /snap/bin/certbot /usr/bin/certbot
sudo certbot certonly --standalone
```

Follow instructions when prompted.

To make sure our app has correct access to the certificate, we are going to create a safe user group `ssl-cert` and add root and current user to the safe group.

```
sudo addgroup ssl-cert
sudo adduser <name> ssl-cert
sudo adduser root ssl-cert
```

Transfer certificate folders to the safe user group

```
sudo chgrp -R ssl-cert /etc/letsencrypt/live
sudo chgrp -R ssl-cert /etc/letsencrypt/archive
```

And then configure folders' access permission

```
sudo chmod -R 750 /etc/letsencrypt/live
sudo chmod -R 750 /etc/letsencrypt/archive
```

Reboot (important!).

Configure the app

Add secrets

The app uses a couple secrets

1) React Google Map API secret

```
cd <path_to_PotatoServer>/view
touch .env
```

Add the secret to the file. It should has the format

```
REACT_APP_GOOGLE_MAPS_API=<the_actual_secret>
```

2) JSON Web Token keys

The app uses JWT to authenticate users. We need a pair of keys to make JWTs

```
cd <path_to_PotatoServer>/model
mkdir secrets
cd secrets
openssl genrsa -out jwt_private.key 512
openssl rsa -in jwt_private.key -pubout -out jwt_public.key
```

Configure back end environment

Make a file called `.env.production` in the back end

```
cd <path_to_PotatoServer>/model
touch .env.production
```

The file should contain port information, certificates location, Google Map API key and database connection parameters. Here’s an example

```
HTTP_PORT=80
HTTPS_PORT=443
CERTIFICATE_KEY_PATH=/etc/letsencrypt/live/potato.colab.duke.edu
/privkey.pem
CERTIFICATE_SERVER_PATH=/etc/letsencrypt/live/potato.colab.duke.edu
/cert.pem
CERTIFICATE_CHAIN_PATH=/etc/letsencrypt/live/potato.colab.duke.edu
/chain.pem
BASE_URL='potato.colab.duke.edu'

TYPEORM_CONNECTION = postgres
TYPEORM_HOST = localhost
TYPEORM_USERNAME = postgres
TYPEORM_PASSWORD = <database_password>
TYPEORM_DATABASE = bus
TYPEORM_PORT = 5432
TYPEORM_SYNCHRONIZE = true
TYPEORM_LOGGING = false
TYPEORM_ENTITIES = build/entity/**/*.js
```

Configure email provider

Add a file called `mailConfig.json` in the back end

```
cd <path_to_PotatoServer>/model
touch mailConfig.json
```

The file should configure RabbitMQ and contain email provider's information. Here's an example:

```
{
    "amqp": "amqp://localhost",
    "queue": "email-queue",
    "exchange-type": "fanout",
    "exchange-name": "email-exchange",

    "server": {
      "port": 587,
      "host": "smtp.gmail.com",
      "user": "potatowebservice@gmail.com",
      "password": <credential>
    }
  }
```

Build the app

Make sure the database is running

Check with

```
sudo service postgresql status
```

If not running, start with

```
sudo service postgresql start
```

Clone the repository

```
git clone -b main git@github.com:cadyzq/PotatoServer.git
```

Compile front end into static file

```
cd <path_to_PotatoServer>/view
npm install
npm run build
```

Build the back end

```
cd <path_to_PotatoServer>/model
npm install
npm run build
```

Start pm2

```
pm2 start <path_to_PotatoServer>/model/build/index.js
```

Yay! We are online!