Deployment Instructions

The following set of instructions will walk through the entire deployment process from start to finish, including the installation of any prerequisite software and deployment tools.

Prerequisites

This application is built for and intended to be run on a Linux System.

Before proceeding further, it is important that the following tool(s) are installed and available for use on the target deployment host:

Install Docker/NGINX

Docker

- Docker
- Docker Compose

The above links lead to the official documentation for installing Docker and Docker Compose. A simple test to verify if the above tools are installed, is to open a terminal and run the following commands:

```
$ docker -v
Docker version 20.10.17, build 100c701

$ docker compose version
Docker Compose version v2.10.2
```

If running the above commands does not result in output similar to above (the versions may be newer), then the necessary tools may not have been installed correctly.

NGINX

- NGINX

The above link leads to the installation guide for NGINX. A simple test to verify if NGINX was installed correctly, is to open a terminal and run the following commands:

```
$ nginx -v
nginx version: nginx/1.17.0
```

If running the above command does not result in output similar to above (the version may be newer), then NGINX may not have been installed correctly.

Register a Google Cloud OAuth 2.0 Application

Follow the 'Prerequisites' section of the following guide, ensuring that the URLs that you choose here match that of your desired deployment URL, and that you save the secret key, and client ID in a secure location for later use.

Register a Github Oath 2.0 Application

Follow this guide, again ensuring that the URLs chosen here match that of your desired deployment URL, and that you save the secret key, and client ID in a secure location for later use.

Configure Stripe Application and Product

- Create a Stripe account
- Create a monthly subscription 'product', making sure to save the corresponsing 'Price ID'.

Clone the Repository

Clone out the application repository on the deployment host in a location which you have permission.

Deployment Configuration

With the prerequisites out of the way, we can begin to configure the application for deployment.

Since this is a full-stack web application, there are four notable components of the application which have their own configuration values.

- NodeJS (Backend)
- ReactJS (Frontend)
- PostgreSQL (Database)
- NGINX (Web hosting)

Luckily, other than NGINX, this application is entirely configurable via a set of predefined environment variables, and therefore the configuration for these components is consolidated.

Additionally, there are a few required Docker configuration options that will be explained at the end.

The following section will go over each component's configuration separately, although you may notice that the configuration files are shared between components. This is no issue, as each of the below sections will only cover the set of environment variables required by that specific component.

Backend Configuration

All of the configuration variables associated with the backend should be configured under the cloned repository's `deployment/.env` file. Some of the configuration values below will have default values.

```
# Full URL (with port) that the application's backend will be
# accessible at. If deploying for local access,
# this value can left as the default. Otherwise, it should be
# set to the full URL where you expect your backend to be
# accessible externally.
BACKEND_URL=http://localhost:3000
```

```
# Full URL (with port) that the application's frontend will be
# accessible at from the backend. If deploying for local access,
# this value can left as the default. Otherwise, it should be
# set to the full URL where you expect your backend to be
# accessible externally.
FRONTEND_URL=http://localhost:8080

# Admin interface configuration
# This application comes with a built in 'admin' interface for the
# application. This interface will be available externally at
# ${BACKEND_URL}/api/admin, and therefore
# requires a secure set of credentials for access. Ensure that these
# credentials are not commited to the repository or shared
# with others.
ADMIN_EMAIL=
ADMIN_PASSWORD=

### Authentication Variables ###
## Google
# Variables associated with Google OAuth 2.0.
# These values can be found on your google API dashboard
# if you have registered the application correctly.
# Again, ensure that the CLINET_ID and CLIENT_SECRET values are not # shared with
anyone outside of this file.
GOOGLE_CLIENT_SECRET=
GOOGLE_CLIENT_ID=
GOOGLE_CALLBACK_URL=

## Github
# Variables associated with Github OAuth 2.0.
# These values can be found under your Github Developer Settings
# if you have registered the application correctly.
# Again, ensure that the CLINET_ID and CLIENT_SECRET values are not # shared with
anyone outside of this file.
GITHUB_CLIENT_SECRET=
GITHUB_CLIENT_ID=
GITHUB_CALLBACK_URL=

## Session Variables
# This variable is required to compute the hash for all
# browser sessions created by the application.
# Ensure that this value is sufficiently long and random, and is not
# shared with anyone.
SESSION_SECRET=

### Stripe Variables ###
# Stripe Secret key
# This the is the secret key associated with your Stripe account
# and application. It can be found on your Stripe dashboard
# and should not be shared with anyone.
STRIPE_SECRET_KEY=

# This the is the publishable key associated with your Stripe
# account and application. It can be found on your Stripe dashboard
```

```
# and will be used on the front-end.
STRIPE_PUBLISHABLE_KEY=

# The Price ID of the subscription we created in the prerequisites
# section.
STRIPE_SUBSCRIPTION_ID=
```

Frontend Configuration

All of the configuration variables associated with the frontend should be configured under the cloned repository's `frontend/.env` file. This information **will** be commited to the repository, and viewable by users. Therefore, make sure none of these configuration values contain sensitive information.

```
# The Frontend equivalent of the backend's BACKEND_URL and
# FRONTEND_URL variables. These values should be consistent
# with the ones configured in the other file.
REACT_APP_BACKEND_URL=http://localhost:3000
REACT_APP_FRONTEND_URL=http://localhost:8080
```

Databse Configuration

The following configuration values are related to the deployment of, and connection to the PostgreSQL database that is required by the application. All configuration for this component should be configured under the cloned repository's `deployment/.env` file.

```
# Name of the postgres database to create/connect to
POSTGRES_DB=postgres

# The user/pass create and log into postgres as for use
# inside the application
POSTGRES_USER=postgres
POSTGRES_PASSWORD=password

# Local volume to mount postgres data into on the deployment host.
# Will not be automatically created!
# Optional. Uncomment if desired
# PG_VOLUME_DIR=./pgdata
```

NGINX Configuration

This configuration is not necessary if you are only deploying the app locally. If you only want the app to be accessible on the deployment host, you can skip this configuration step.

Create the following NGINX 'server block' as a new file under `/etc/nginx/sites-available`, replacing any templated variables with the desired values (<>)

```
server {
    listen [::]:80;
    listen 80;
    server_name <desired-server-name>
    location / {
        include proxy_params;
        proxy_set_header Host $host
        proxy_set_header X-Real-IP $remote_addr
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_pass http://localhost:8080
    }
    location /api {
        include proxy_params;
        proxy_set_header Host $host
        proxy_set_header X-Real-IP $remote_addr
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_pass http://localhost:3000
    }
    location /admin/ {
        include proxy_params;
        proxy_set_header Host $host
        proxy_set_header X-Real-IP $remote_addr
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_pass http://localhost:3000/admin
    }
}
```

Once this is configured, you can validate and restart NGINX via the following commands to allow your changes to take effect:

```
sudo nginx -t
sudo systemctl restart nginx
```

If either of these commands fail, be sure to check your NGINX configuration for errors.

Additional Docker Configuration

At this point, all of the application specific configuration should be completed. However, before we can deploy the application, there are a couple more variables required. These must also exist under the cloned repository's `deployment/.env` file.

```
# Ports you want the application components to be exposed
# on locally. The values for this aren't too important,
# however if you change them, you must modify any reference
# to the default port values in the above configuration to
```

```
# match.
FRONTEND_EXPOSE_PORT=8080
BACKEND_EXPOSE_PORT=3000
POSTGRES_PORT=5432

# Docker image 'tags' of the existing frontend/backend
# container images that you want to deploy.
# Must already be created via the 'build-images.sh' script.
# For the sake of this guide, you can leave them as 'main',
# as we will be creating the images with that tag.
FRONTEND_IMAGE_TAG=main
BACKEND_IMAGE_TAG=main
```

Building the Application Images

Once the application and your deployment host has been configured to your liking, we can bow build the application container images in preparation for deployment via Docker Compose.

The cloned repository already comes pre-equipped with a script to carry out this action under `deployment/build-images.sh`. This bash script supports a single environment variable called `TAG`, which must be set to the value configured above for the `FRONTEND_IMAGE_TAG` and `BACKEND_IMAGE_TAG` application environment variables configured in the previous section.

An example run of this command is as follows:

```
$ cd deployment
$ TAG=main ./build-images.sh
# Script output
...
```

This command will take some time, as it has to compile all of the application code, so be patient. You can verify that the above script succeeded, by running the following command:

```
$ docker images
REPOSITORY  TAG    IMAGE_ID  CREATED  SIZE
frontend    main   ...       ...      ...
backend     main   ...       ...      ...
```

Deploying the application

Once all of the above configuration steps have been performed, and you have validated that that values you entered are correct, we can proceed with deploying the application. On the deployment host, change into the repository's `deployment` directory, and run the following command to start the application.

```
docker compose -f deployment-compose.yaml up -d
```

Give it a few moments to spin up all of the application components, and afterwards, your application should be available at the URL(s) that you configured above! You can view each running container via the following command:

```
docker ps -a
```

To troubleshoot an individual component and view its logs, you can run the following command, where `<component>` refers to the name of the running image obtained via the above command:

```
docker logs <component>
```

Finally, in order to 'tear-down' or 'undo' a deployment of the application, you can run the following command from the same `deployment` directory:

```
docker compose -f deployment-compose.yaml down
```