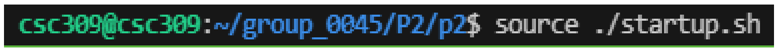
P2 Documentation

by: Vivekanand Gadhia and Piotr Szaran

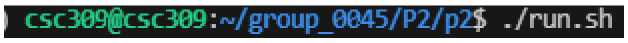
Project Structure:

- The project is structured into three applications: restaurants, accounts, and blogs. It comes with a startup script that should only be run inside of the main '/p2' folder where all the apps are found. The script is run in the terminal using 'source ./startup.sh' where the script will prepare a new virtual environment, install all the required packages (which are all located in a requirements.txt files) using pip, and rull all migrations.



```
csc309@csc309:~/group_0045/P2/p2$ source ./startup.sh
```

- Notice in the screenshot above that there is a 'P2' directory that houses our project directory 'p2'. When running the script, make sure to be in the little 'p2' directory where the scripts are located.
- Once running the startup script, one can start a server by using the other provided script run.sh. This script is run by typing './run.sh' in the terminal and should be run after running the startup script.



```
csc309@csc309:~/group_0045/P2/p2$ ./run.sh
```

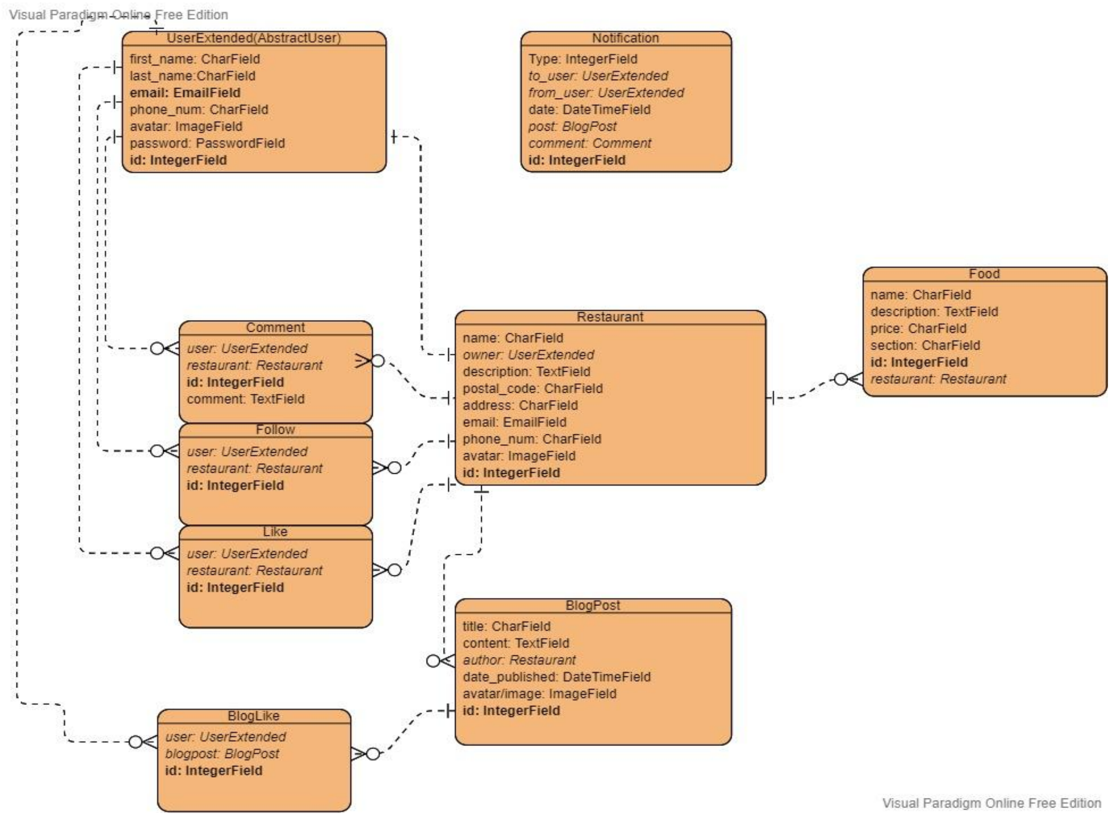
- Lastly, the database and virtual environment have been included in a .gitignore to avoid pushing and pulling any large and unnecessary files.

Apps:

- **restaurants:** The **restaurants** app contains the models, views and serializers for the Restaurant, Food, Like, Comment, and Follow models. Additionally, it has views for search and the view the restaurant data/menu for the user's owners restaurants and other restaurants by passing the primary key through the endpoint outlined in the below endpoints.
- **accounts:** The **accounts** app contains the user and notifications model and serializers as well as authentication and authorization views. This app is where registering, logging in and logging out also take place as well as the creation of a new restaurant and to get the notifications for the user.
- **blogs:** The **blogs** app contains all the models, views and serializers pertaining to blog posts. It also handles the likes for blog posts as well as a view to generate the user's feed based on followed restaurants.

<u>Model Design:</u>

The following ER Diagram was used in constructing and connecting the models together.



- **UserExtended** requires an email as its primary key as opposed to username which is the default for the **AbstractUser** model. This means the email acts as the username for distinction between entries as well as the auto-generated id values.
- **Restaurant** is the center of the diagram as most, if not all of the other models make some reference to it. In most cases, the user is often referenced through restaurant .owner and vice versa when using a filter query.
- **Food** is a model which contains any food item that any restaurant has on their menu. The menus for any given restaurant is achieved by filtering all entities in **Food** by getting the same restaurant.
- **Comment**, **Follow**, **Like**, and **BlogLike** use a similar structure to store specific relations between users, restaurants and blog posts. This was purposefully done in order to not conflict with each other by having them separated and to better separate the views in the apps as well.
- **Notification** is a model used to store and create notifications. For notification with a restaurant(i.e. new restaurant posts), the from_user will be the owner of the restaurants. The **Notification** model is not connected for simplicity sake as it will just be connected to the **UserExtended** model with references to the **Comment** and **BlogPost** models as well.

API Endpoints:

Accounts Endpoints:

Endpoint: /accounts/register/

Methods: POST

Fields/payload: username, email, password, password2, first_name, last_name, avatar, phone_number

Description: A user will be created where the unique field is a user's email. This email will act as a username for the account, accompanied by a password of the user's choice. Other field's a user can include are first_name, last_name, phone number, and an avatar.

Endpoint: /accounts/login/

Methods: POST

Fields/payload: username, password

Description: A user can enter their email and password combination to login into the website. Upon logging the user is authenticated if they have entered the right username and password. This returns the access and refresh tokens to access other pages. Refresh tokens last 60 minutes.

Endpoint: /accounts/login/ token_refresh

Methods: POST

Fields/payload: refresh_token

Description: Refreshes a token back to the 60 minute time limit.

Endpoint: /accounts/profile/view/

Methods: GET

Authentication: Bearer Token (From login)

Fields/payload: username, email, password, first_name, last_name, avatar, phone_number

Description: Returns a view of the profile which displays all of the fields that were filled out upon registering. Some fields may be empty since not every field is required.

Endpoint: /accounts/profile/edit/

Methods: GET, PUT

Authentication: Bearer Token (From login)

Fields/payload: email, password, first_name, last_name, avatar, phone_number

Description: Returns a view of the profile with the option of modifying the user's profile fields. If a field has been left blank from the start, the user now has a chance to fill it in.

Endpoint: /accounts/profile/notification/

Methods: GET

Authentication: Bearer Token (From login)

Fields/payload: type, to_user, from_user, date, post(optional), comment(optional)

Description: Returns a list of notification objects with the above fields. The to_user is the one receiving the notification from the from_user. In the case of a notification that a restaurant posted, the from_user will be the owner of the restaurant. The type field is an integer to represent the type of notification that occurred. 1 is a new like on either the restaurant or blog post, 2 is a new comment, 3 is a new post from a followed restaurant, 4 is a new follow, and 5 is a menu change or addition. The optional fields for post and comment are used to reference the specific post or comment associated with the notification.

Endpoint: /accounts/profile/addrestaurant/

Methods: POST

Authentication: Bearer Token (From login)

Fields/payload: name, description, postal_code, address, email, phone_num, avatar

Description: Creates a new restaurant for the logged in and authenticated user. The user will automatically become the owner of the new restaurant created. A user can only do this if they do not already have a restaurant.

Blogs Endpoints:

Endpoint: /blogs/feed/

Methods: GET

Authentication: Bearer Token (From login)

Fields/payload: title, content, author, date_published, avatar, id

Description: Returns a feed of all blog posts from followed restaurants of authenticated user. Uses ListAPIView to return a list of serialized blog posts with the above fields sent for each post.

Endpoint: /blogs/<int:pk>/

Methods: GET

Fields/payload: title, content, author, date_published, avatar, id

Description: Returns the above fields for a given blog post with pk from the endpoint.

Endpoint: /blogs/<int:pk>/like/

Methods: POST

Authentication: Bearer Token (From login)

Fields/payload: user, blogpost

Description: Creates an entity in the BlogLikes model with the user already set to the authenticated/logged in user and the post with id from the endpoint.

Restaurants Endpoints:

Endpoint: /restaurants/myRestaurant/blogs/

Methods: GET, POST

Authentication: Bearer Token (From login)

Fields/payload: title, content, author, date_published, avatar, id

Description: Returns all blog posts where the author is the user's restaurant. The GET method acts as the /blogs/feed/ endpoint, returning a list of BlogPosts where the author is the user's restaurant. The POST method creates a new BlogPost with the payload stated above. After sending the POST request, the GET request should update with the newly created BlogPost.

Endpoint: /restaurants/myrestaurant/about/

Methods: PUT, GET

Authentication: Bearer Token (From login)

Fields/payload: name, owner, description, postal_code, address, email, phone_num, avatar

Description: Returns the information of the user's restaurant. The GET method returns and serializes the fields of the user's restaurant. The PUT method allows the user to edit their restaurant's information.

Endpoint: /restaurants/myrestaurant/menu/

Methods: POST, GET

Authentication: Bearer Token (From login)

Fields/payload: name, price, section, description, restaurant

Description: The GET method displays a list of every menu item and the items given fields. The POST method lets you add new menu items to the restaurant's menu.

Endpoint: /restaurants/myrestaurant/comments/

Methods: GET

Authentication: Bearer Token (From login)

Fields/payload: user, restaurant, comment

Description: A list view of all the comments that users have left a specific restaurant on their restaurants page.

Endpoint: `/restaurants/myrestaurant/followers/`

Methods: `GET`

Authentication: Bearer Token (From login)

Fields/payload: `user, restaurant`

Description: A list of all the users that are following the logged in users restaurant.

Endpoint: `/restaurants/myrestaurant/likes/`

Methods: `GET`

Authentication: Bearer Token (From login)

Fields/payload: `user, restaurant`

Description: A list view of all the users that have liked the logged in users restaurant.

Endpoint: `/restaurants/<int:pk>/blogs/`

Methods: `GET`

Fields/payload: `author, title, content, avatar, date_published`

Description: A list view of another restaurant's blog posts sorted by newest to oldest.

Endpoint: `/restaurants/<int:pk>/about/`

Methods: `GET`

Fields/payload: `restaurant, owner, name, description, postal_code, address, email, phone_number, avatar`

Description: A view of `<int:pk>` restaurant's about page from the endpoint.

Endpoint: `/restaurants/<int:pk>/menu/`

Methods: `GET`

Fields/payload: `name, price, section, description, restaurant`

Description: A list view of `<int:pk>` restaurant's menu from the endpoint.

Endpoint: `/restaurants/<int:pk>/comments/`

Methods: `POST, GET`

Authentication: Bearer Token (From login)

Fields/payload: `user, restaurant, comment`

Description: The GET method returns a list view of restaurant `<int:pk>`’s comments given by the endpoint. The POST method writes a new comment to restaurant `<int:pk>`’s comment section.

Endpoint: `/restaurants/<int:pk>/follow/`

Methods: `GET`

Authentication: Bearer Token (From login)

Fields/payload: `user, restaurant`

Description: A GET request in which a specified user is now following restaurant `<int:pk>` given by the endpoint.

Endpoint: `/restaurants/<int:pk>/like/`

Methods: `GET`

Authentication: Bearer Token (From login)

Fields/payload: `user, restaurant`

Description: A GET request in which a specified user now likes restaurant `<int:pk>` given by the endpoint.

Endpoint: `/restaurant/search/`

Methods: `GET`

Fields/payload: `restaurant, address, food`

Description: A list view of every restaurant, address, or food item that matches the inputted field.
