RESTIFY

Phase 2 Documentation

Setup:

`cd into P2/ as cloned from MarkUs`
`source setup.sh`
`./run.sh`

Project Structure:

File Structure and Description of Views:

/P2
/restify **(Project)**
- settings.py
- …

/accounts
- /migrations
- /views
  - add_restaurant.py: For adding a restaurant to an account
  - edit_profile.py: For editing an account's information
  - feed.py: For viewing the feed of an account
  - get_profile.py: For viewing an account's information
  - like_follow.py: For liking/following restaurants/blogposts using this account
  - register.py: For creating a new account
  - view_notification.py: For viewing a notification sent to this account
- serializers.py
- models.py: Contains the models for Accounts (see Models section)
- …

/restaurants
- /migrations
- /views
  - add_comment.py: To add a comment to a restaurant
  - blogpost_likes.py: To get the number of likes a blogpost has
  - blogposts.py: To get the blogposts posted by a restaurant
  - comment.py: To retrieve the comments on a restaurant's page
  - edit_menu.py: To edit a restaurant's menu
  - edit_restaurant.py: To edit a restaurant's information

followers.py: To retrieve a list of followers of a restaurant
gallery.py: To retrieve all photos of a restaurant
get_contact_info.py: To retrieve contact info of a restaurant
menu.py: To retrieve the menu of a restaurant
my_restaurant.py: Retrieve the information of a restaurant
search.py: To search for restaurants on Restify
serializers.py
models.py: Contains models for Restaurants (see Models section)
…
startup.sh: Script to setup venv and make migrations
run.sh: Script to run the server
docs.pdf: Documentation of our project
…

Apps/Organization of Code:

Since Restify is a social media platform for restaurants, we have two main sets of objects that are present on the website: accounts (users of the website) and restaurants (the entities present on the social media part of the website).

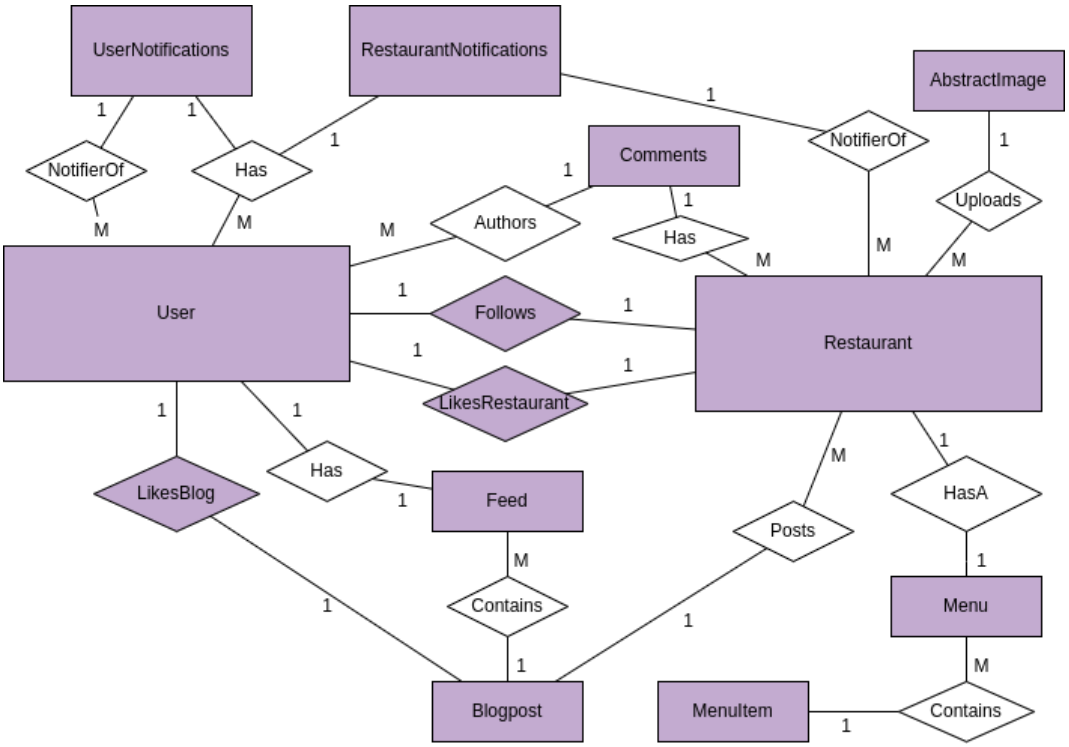
Because of this, we have two apps in our project: **Restaurants** and **Accounts**. This also allows us to logically separate the URL namespaces of the project. Anything prefixed with /accounts/ is likely an endpoint that acts on a user's account, without the need for restaurant data. Likewise, anything prefixed with /restaurants/ is likely an endpoint that acts on a restaurant's data, regardless of the account's data.

Restaurants handles any operations, models, and views pertaining to a restaurant's data. For example, viewing blog posts, writing a blog post, posting a comment, adding a menu item, viewing a menu, or viewing a restaurant's general information.
Accounts handles anything related to a user account's data. This includes the notifications a user receives, creating a restaurant for a user, registration and log in, and more.

Endpoints will be listed and described in this documentation.

Models:

Below is a diagram of the relationships between the models. The attributes of each model are omitted for simplicity. Any models depicted in the diagram that were explicitly created in Django are filled in purple, anything else is added for clarity on what the relationships are.



Here are the attributes in detail, foreign keys are in **bold**:

- For the Accounts App:
  - AbstractUser (Extends Django default User model)
    - Additional attributes aside from the default model are:
    - owned_restaurant: The **Restaurant** owned by this user, NULL otherwise
    - is_owner: whether or not this account owns a restaurant
    - avatar: the avatar image of this user
    - phone_num: user's phone number
  - Feed
    - user: the **AbtractUser** that has this post in their feed
    - post: the specified **Blogpost**
  - LikesRestaurant
    - user: the **AbstractUser** on Restify who liked this restaurant

        - restaurant: the **Restaurant** that was liked by <user>
    - LikesBlog
        - user: the **AbstractUser** on Restify who liked this blogpost
        - blogpost: the **Blogpost** that was liked by <user>
    - Follows
        - user: the **AbstractUser** who is following this restaurant
        - restaurant: the **Restaurant** that was liked by <user>

Below we have two classes of Notifications.

UserNotifications are sent to regular users and are triggered by Restaurants. They are created when Restaurants update their menu, or upload a blogpost.

RestaurantNotifications are sent to restaurant owners are are triggered when a user likes a blogpost/restaurant, follows a restaurant, or posts a comment on a restaurant's page. We split Notifications into these classes because the type of the notifier attribute could be from either a user or from a restaurant.

- UserNotifications:
    - user: the **AbstractUser** who will be shown this notification
    - description: what the notification is for, for example " uploaded a new blogpost" or " updated their menu"
    - link: a link to the source of the notification, optional. For example, a notification for an update to a menu should link to the restaurant's menu
    - notifier: the **Restaurant** who is sending out this notification
    - datetime: the date and time this notification was created
- RestaurantNotifications:
    - user: the **AbstractUser** who will be shown this notification
    - description: what the notification is for, for example " started following you" or " liked your blogpost"
    - link: a link to the source of the notification, optional
    - notifier: the **AbstractUser** who is sending out this notification
    - datetime: the date and time this notification was created

- For the Restaurants App:
    - Restaurant:
        - name: name of the restaurant
        - description: description of the restaurant, optional

    - owner: the **AbstractUser** that created this restaurant
    - address: address of the restaurant
    - postal_code: postal code of the restaurant
    - logo: the restaurant's logo
    - phone_num: phone number of the restaurant
    - followers: number of followers on Restify, default 0
    - likes: number of likes on Restify, default 0, not to be confused with the number of likes a restaurant's blogpost may have.
- Blogposts
    - title: title of the blogpost
    - body: the blogpost text body
    - image: one image to be added to the post, optional
    - author: name of person who wrote the post
    - restaurant: **Restaurant** who this blogpost belongs to
    - date: the date and time this post was created
    - likes: number of likes on Restify, default 0
- Comments
    - author: the **AbstractUser** who wrote the comment
    - restaurant: the **Restaurant** that was commented on
    - text: the comment text
- Menu
    - owner: the **Restaurant** that owns this menu
- MenuItem
    - name: the name of the menu item
    - menu: the **Menu** this item belongs to
    - description: description of the menu item
    - price: price of the menu item
    - category: the category this menu item belongs to, such as "appetizer", "dessert", "drink", these categories are chosen by the restaurant owner
- AbstractImage
    - image: the image itself
    - uploader: the **Restaurant** that is uploading this image
    - description: a meta description of the image

Endpoints:

Endpoints for the Accounts App:

- **Endpoint:** `/accounts/register/`
  - Methods: POST
  - View Class: `CreateAPIView`
  - Authenticated: `NO`
  - Fields: `username, password1, password2, first_name, last_name email,avatar, phone_num`
  - Description: Creates a new user with fields and returns JSON response of the new user
  - Validation errors:
    - The two password fields didn't match
    - This password is too short. It must contain at least 8 characters
    - Enter a valid email address
  - Example response:

```
{"username": “Stinkyboy”, “first_name”: “Stinkiest”, “last_name”: “Boy”, “email”: “stinkyboy@gmail.com”, “avatar”: "http://127.0.0.1:8000/buddy.jpeg", “phone_num”: “1234567890”}
```

- **Endpoint:** `/accounts/login/`
  - Methods: POST
  - Authenticated: `NO`
  - Fields: `username, password`
  - Description: Returns authentication token if username and password match credentials of an existing user
  - Validation errors:
    - `No active account found with the given credentials`
  - Example response:

```
{"refresh":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTY0NzQ2MDY5OCwiaWF0IjoxNjQ3Mzc0Mjk4LCJqdGkiOiIzN2U5NDE4NWNkMzg0MDJkOWQ2Njk5NTBjYTBkYjc3ZSIsInVzZXJfaWQiOjJ9.7-Jv90bhtJ37XxcPAZraTvic2rMIZn32CEgylJ9UnQI",
    "access":
"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjQ3NDYwNjk4LCJpYXQiOjE2NDczNzQyOTgsImp0aSI6Ijk2
```

NmFhMjc5NWExNjQyYTQ4NTE4ZTgyMjM0NzYyOGM1IiwidXNlcl9pZCI6Mn0.Pu8
6oZEe64JYS8DlCzjFLUsDeuZxNzUqwpK9LMHRShY"}

- **Endpoint:** `/accounts/profile/`
  - Methods: `GET`
  - View class: `RetrieveAPIView`
  - Authenticated: YES (return 401 UNAUTHORIZED otherwise)
  - Fields: first_name, last_name, email, avatar, owned_restaurant, phone_num
  - Description: Return a JSON string with the above fields, disclosing data on the logged-in user
  - Example response:
    {"first_name":"Johnny", "last_name":"Stewart",
    "email":"Johnnyboy@gmail.com",
    "avatar":null,"owned_restaurant":null,"phone_num":null}

- **Endpoint:** `/accounts/profile/edit/`
  - Methods: `GET, PUT, PATCH`
  - Authenticated: YES
  - View class: `RetrieveAPIView, UpdateAPIView`
  - Fields/payload: `first_name, last_name, email, avatar, owned_restaurant, phone_num, password1, password2`
  - Description: Endpoint to allow a user to edit their credentials.
  - Validation errors:
    - `The two password fields didn't match`
    - `This password is too short. It must contain at least 8 characters`
    - `Enter a valid email address`
  - Example response:
    {"first_name":"Stinkiesthaha","last_name":"Boy","email":"stinky
    boy@gmail.com","avatar":"http://127.0.0.1:8000/buddy.jpeg","own
    ed_restaurant":null,"phone_num":"1234567890"}

- **Endpoint:** `/accounts/notifications/<id>/view/`
  - Methods: `GET`
  - Fields: `url`

    - Authenticated: YES
    - Description: Endpoint to retrieve the URL link of a specified notification with id `<id>` in a JSON. If the logged-in user is not the receiver of this notification, 401 is returned. If the notification with id `<id>` does not exist, 404 is returned.
    - Additional notes: On the viewing of a notification, the notification is deleted from the user's list of notifications. We do this so the user's notifications tab only contains notifications they have not clicked on/viewed. If a notification does not have a URL associated with it, the value shown is null.
    - Example response:
    - `{"error":"Notification with this ID does not exist."}`
    - `{"url": "http://localhost:8000/restaurants/45"}`
    - `{"url": null}`
    - `{"error":"You are not authorized to view this notification."}`

- **Endpoint**: `/accounts/add-restaurant/`
    - Methods: POST
    - Fields/payload: `name, description, address, postal_code, phone_num, logo`
    - Authenticated: YES
    - Description: Endpoint to create a new restaurant, with the owner set to the currently logged-in user. If the user already owns a restaurant where they are the owner, we return 400 BAD REQUEST. On success, a JSON with the posted data is returned. The `description` field is NOT required.
    - Example response:
    - `{"name":"McDonalds","description":"yes, this is my restaurant!","address":"1234 Mickey St.","postal_code":"123456","logo":"http://localhost:8000/mcd.png","phone_num":"4162908173"}`
    - `{"error": "You may only create up to one restaurant."}`

- **Endpoint:** `/accounts/like/restaurant/<restaurant_id>/`
    - Methods: `POST, DELETE`
    - Authenticated: YES (return 401 UNAUTHORIZED otherwise)
    - Validation errors:
        - `Restaurant with id=<restaurant_id> does not exist`
        - `Already liked restaurant`
        - `Restaurant is not liked`

    - Description: Endpoint to like/unlike a restaurant with `id = <restaurant_id>`. On success for POST, the logged in user likes the restaurant and a LikesRestaurant object is created. The like count for the restaurant is increased by 1 and a `RestaurantNotification` object is created. On success for `DELETE`, the logged in user unlikes the restaurant and the `LikesRestaurant` object is deleted and restaurant likes count is decremented.
    - Example response:
    - `{"user": 2, "restaurant" : 2}`
    - `{"detail": "Already liked restaurant"}`
    - `{"detail": "Restaurant with id=4 does not exist"}`

- **Endpoint:** `/accounts/follow/restaurant/<restaurant_id>/`
    - Methods: `POST, DELETE`
    - Authenticated: YES (return 401 UNAUTHORIZED otherwise)
    - Fields/payload: restaurant, user (restaurant is given through url and user is given from request.user)
    - Validation errors:
        - `Restaurant with id=<restaurant_id> does not exist`
        - `Already following restaurant`
    - Description: Endpoint to follow/unfollow a restaurant with id = <restaurant_id>. On success for POST, the logged in user follows the restaurant and a FollowsRestaurant object is created. The follow count for the restaurant is increased by 1 and a RestaurantNotification object is created. On success for DELETE, the logged in user unfollows the restaurant and the FollowsRestaurant object is deleted and restaurant follower count is decremented.
    - Example response:
    - `{"user": 2, "restaurant" : 2}`
    - `{"detail": "Already following restaurant"}`
    - `{"detail": "Restaurant with id=4 does not exist"}`

- **Endpoint:** `/accounts/like/blogpost/<blogpost_id>/`
    - Methods: `POST, DELETE`
    - Authenticated: YES
    - Description: Endpoint to like/unlike a restaurant with `id = <restaurant_id>`. On success for POST, the logged in user

likes the restaurant and a LikesRestaurant object is created. The like count for the restaurant is increased by 1 and a `RestaurantNotification` object is created. On success for `DELETE`, the logged in user unlikes the restaurant and the `LikesBlog` object is deleted and restaurant likes count is decremented.

- Example response:
- `{"user": 2, "restaurant" : 2}`
- `{"detail": "Already liked blogpost"}`
- `{"detail": "Blogpost with id=4 does not exist"}`

- **Endpoint:** `/accounts/feed/`
  - Methods: `GET`
  - Authenticated: `Yes`
  - View class: `ListAPIView`
  - Description: Returns a JSON response of all blogposts of the user's feed.
  - Example response:

```
{"count":2,"next":null,"previous":null,"results":[
      {"id":1,"title":"Epic bean
      sauce","image":null,"body":"bleh bleh
      bleh","author":"Sally","date":"2022-03-09T15:53:31.539576
      Z","likes":0,"restaurant":1},
      {"id":2,"title":"Another epic bean
      sauce","image":null,"body":"blah blah
      blah","author":"Sam","date":"2022-03-09T15:55:01.920747Z"
      ,"likes":0,"restaurant":1}
]}
```

Endpoints for the Restaurants App:

- **Endpoint**: `/restaurants/search/<query>/`
  - Methods: `GET`
  - Authenticated: No
  - View class: `ListAPIView`
  - Fields/payload: `id, name, followers, address, postal_code, logo`
  - Description: Returns a JSON of the above attributes for any Restaurant objects that contain the keywords written in `<query>`,

either in their name, menu items, or their address. These JSON results are sorted by *number of followers* as popularity.
    - Additional notes: Strings in the query can be separated by a space, and punctuation should be used. For example, “Sally’s saloon” will not return if you type “sallys”, but it will return if you simply type “sally”.
    - Example response (in this example, we searched “beer” and both Sally’s Saloon and Joe’s Bar sell beer):

```
{"count":2,"next":null,"previous":null,"results":
[{"id":1,"name":"Sally's Saloon","address":"736 Sally
St.","followers":8,"postal_code":"123456","logo":"http://localh
ost:8000/sallylogo.jpg"},
{"id":2,"name":"Joe's Bar","address":"1234 Brock
St.","followers":3,"postal_code":"197373","logo":"http://localh
ost:8000/joesbar.jpg"}]}
```


- **Endpoint**: `/restaurants/<id>/menu/`
    - Methods: `GET`
    - View class: `ListAPIView`
    - Fields: `id, name, description, price, category, menu`
    - Authenticated: No
    - Description: Returns a JSON response of all menu items belonging to a specific restaurant, as referenced by its `<id>`. If there is no restaurant with that id, HTTP 404 is returned. Sorted by category.
    - Example response:

```
{"count":1,"next":null,"previous":null,"results":[{"id":1,"name
":"Ice cream","description":"Our famous ice
cream!","price":10,"category":"Dessert","menu":1}]
```


- **Endpoint**: `/restaurants/<id>/blogposts/`
    - Methods: `GET`
    - View class: `ListAPIView`
    - Fields: `title, author, restaurant, body, image, date, likes`
    - Authenticated: No
    - Description: Returns a JSON response of all the blog posts belonging to a specific restaurant, as referenced by its `<id>`. If

there is no restaurant with that id, HTTP 404 is returned. Results are sorted by most recent upload date.
    - Example response:

```
{"count":2,"next":null,"previous":null,"results":[
      {"id":1,"title":"Epic bean
      sauce","image":null,"body":"bleh bleh
      bleh","author":"Sally","date":"2022-03-09T15:53:31.539576
      Z","likes":0,"restaurant":1},
      {"id":2,"title":"Another epic bean
      sauce","image":null,"body":"blah blah
      blah","author":"Sam","date":"2022-03-09T15:55:01.920747Z"
      ,"likes":0,"restaurant":1}
]}
```


- **Endpoint**: `/restaurants/<id>/followers/`
    - Methods: GET
    - Fields: `username`
    - Authenticated: Yes
    - Description: Returns a JSON response of all users that are following the restaurant with id `<id>`. Only the usernames of the users are provided.
    - Example response:

```
{"count":3,"next":null,"previous":null,"results":[{"username":"
testuser"},{"username":"Sally"},{"username":"Joe-Bob"}]}
```


- **Endpoint**: `/restaurants/<id>/likes/`
    - Methods: GET
    - Fields: `username`
    - Authenticated: Yes
    - Description: Returns a JSON response of all users that have liked the page of the restaurant with id `<id>`. Only the usernames of the users are provided. Not to be confused with the users who like specific blog posts.
    - Example response:

```
{"count":3,"next":null,"previous":null,"results":[{"username":"
admin"},{"username":"testuser"},{"username":"Joe-Bob"}]}
```


- **Endpoint**: `/restaurants/blogposts/<blogpost-id>/likes/`
    - Methods: GET
    - Fields: `count`

    - Authenticated: No
    - Description: Returns a JSON response of the number of likes a blogpost has received as `count`.
    - Example response:
      `{"count":1}`

- **Endpoint**: `/restaurants/<id>/edit/`
    - Methods: `PATCH`
    - Fields/payload: `name, description, address, postal_code, phone_num, logo`
    - Authenticated: YES
    - Description: Endpoint to edit the values of the restaurant with id <id>. The `description` field is NOT required. If the user requesting to edit this restaurant is not the owner, we return 401 UNAUTHORIZED, and if the restaurant with id <id> does not exist, we return 404 NOT FOUND. On success, nothing is returned.
    - Example response:
    - `{"error":"You are not the owner of this restaurant"}`

- **Endpoint**: `/restaurants/<id>/comments/`
    - Methods: `GET`
    - Authenticated: No
    - Description: Endpoint to view all the comments made by users on a restaurant's page.
    - Example response:
    - `{"author": "user", "restaurant": "restaurant name", "text": "something random"}`

- **Endpoint**: `/restaurants/<id>/gallery/`
    - Methods: `GET`
    - Fields/payload: `restaurant`
    - Authenticated: No
    - Description: Endpoint to show all the images that belong to a restaurant.
    - Example response:
    - `{"image:""http://localhost:8000/image.jpeg, "description": "image description"}`

- **Endpoint**: `/restaurants/<id>/contact/`
    - Methods: `GET`
    - Authenticated: YES
    - Description: Endpoint to show the contact information of a restaurant, so all the information you would see on the restaurant “About” page, which is the address, postal code, and phone number of the restaurant.
    - Example response:
    - `{"address": "123 John Street","postal_code": "123123","phone_num": "416111111"}`

- **Endpoint**: `/restaurants/<id>/`
    - Methods: `GET`
    - Fields/payload: `name`
    - Authenticated: YES
    - Description: Endpoint to show all the information about the restaurant, so everything that you see on the restaurant home page..
    - Example response:
    - `{"name": "restaurant name","description": "description","address": "address","postal_code": "123123","logo": "http://localhost:8000/mylogo.jpeg","phone_num": "416222333"}`
    - `{"error":"You are not the owner of this restaurant"}`

- **Endpoint**: `/restaurants/<id>/add-comment/`
    - Methods: `POST`
    - Fields/payload: `text`
    - Authenticated: YES
    - Description: Endpoint for a user to add a comment on a restaurant’s page.
    - Example response:
    - `{“text”: “random comment”}`

- **Endpoint**: `/restaurants/<id>/add-image/`
    - Methods: `POST`
    - Fields/payload: `image, restaurant, description`

    - Authenticated: YES
    - Description: Endpoint for a restaurant owner to add an image for their restaurant.
    - Example response:
    - `{"description": “description","restaurant": "restaurant", "image": "http://localhost:8000/newimage.jpeg"}`
    - `{"error":"You are not the owner of this restaurant"}`

- **Endpoint**: `/restaurants/add-menu/`
    - Methods: POST
    - Fields/payload: `owner`
    - Description: Endpoint to add a menu to a restaurant.
    - Example response:
    - `{“owner”: “name of restaurant”}`
    - `{"error":"You are not the owner of this restaurant"}`