Documentation

Project Structure and App List:

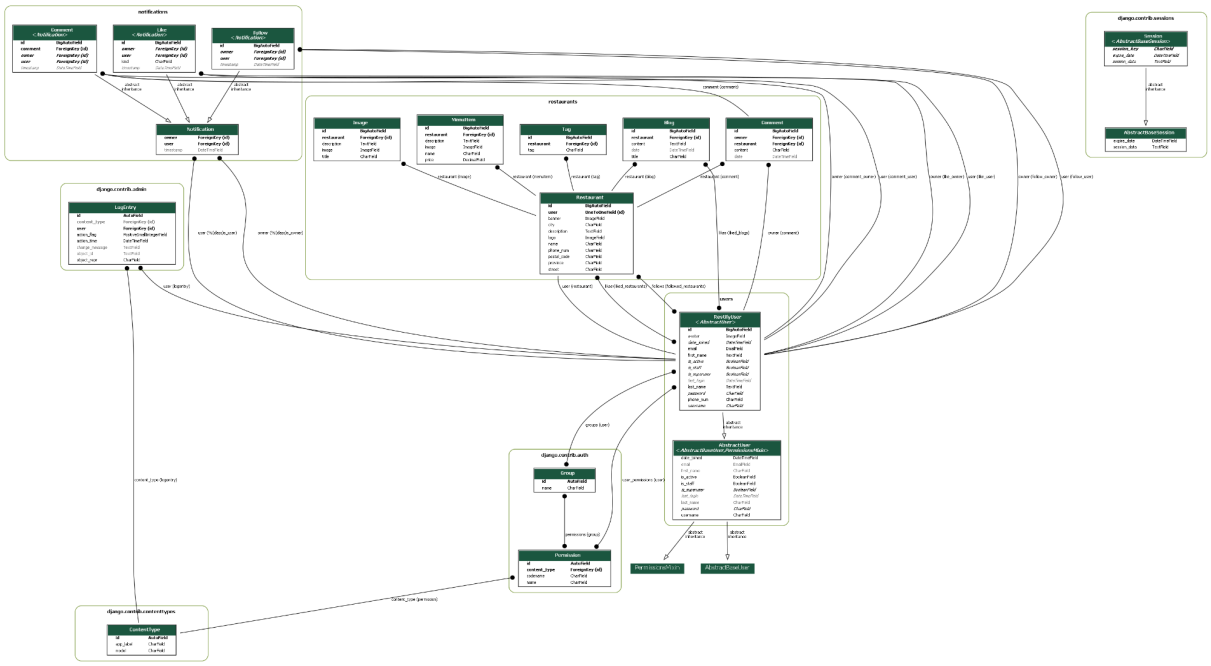
Our project is structured into 3 apps, Users, Restaurants, and Notifications. Each app has its own url path prefix, tests, views, models, and serializers (for rest_framework). For project mana

Notifications has 4 models each representing a different kind of notification, here we're using inheritance from the Notification class to improve code reuse, however the 3 concrete notification types are distinct and need to each have their own table. This app has only one endpoint, /notifications/, which returns a paginated list of notifications for a user. Unfortunately we weren't able to make use of rest_framework conveniences here because of the complexity involving using multiple relations. On the positive side, the function is very flexible and quite easy to follow because little framework 'magic' is involved.

Our Restaurants app is where the bulk of our functionality lives and contains many models and endpoints. For most of these endpoints we heavily leverage rest_framework to do most of the work for us, all we need to do is define our models, serializers, and the framework takes care of the rest.

The Users app is relatively small, but also somewhat unique. The app contains our customer user model that we hook into the Django framework and use throughout our program, this custom model allows us to have avatars, phone numbers, make a few fields required like names, and make usernames and emails the same field. This app contains the endpoints for signing up, logging in, and also getting a users feed (list of blog posts by restaurants the user follows).

Model Design:



Endpoints:

Accounts

Requirements:

- As a user, I can sign up, log in, log out, and edit my profile. Profile information should include first and last name, email, avatar, and phone number.

Endpoint: `/users/signup/`

`Methods:` `POST`

`View class:` `SignUpView`

`Authenticated:` `No`

`Payload:`

```
{
```

```
    “first_name”: string,
    “last_name”: string,
    “email”: string,
    “avatar”: image,
    “phone_num”: string,
    “password1”: string,
    “password2”: string
}
```

Description: Allows an user to sign up for an account.

Notes:

- All fields are required or 400 is returned.
- When the email is not a valid email or has been taken by another account, 400 is returned.
- When the avatar is not an image, 400 is returned.
- When phone_num is not in the format of ###-###-####, 400 is returned.
- When password1 and password2 don’t match, are less than 8 characters, is a commonly used password, or consists only of numbers, 400 is returned.

Endpoint: /users/signin/

Methods: POST

View class: TokenObtainPairView

Authenticated: No

Payload:

```
{
    “email”: string,
    “password”: string
```

```
}
```
Return:
```
{
     “access”: string,
     “refresh”: string
}
```
Description: Allows an user to sign in to their account.

Notes: All fields are required or 400 is returned. If the credentials are incorrect, 401 is returned.

Endpoint: /users/

Methods: PATCH, GET

View class: GetUpdateUserView

Authenticated: Yes (return 401 otherwise)

Payload:

```
{
     “first_name”: string,
     “last_name”: string,
     “avatar”: image,
     “phone_num”: string,
     “password1”: string,
     “password2”: string,
     “password”: string # their current password
}
```
Return:
```
{
     “first_name”: string
     “last_name”: string
     “avatar”: string (link)
     “phone_num”: string
     “email”: string
```

}

Description: Allows an user to edit their profile.

Notes:

- The user's original password is always required, 400 otherwise.
- The password1 and password2 fields must be supplied together (or not at all) and meet the same requirements as /users/signup/, 400 otherwise.
- Per PATCH, you can supply any subset of the rest of the fields as long, given they are not empty and meet the corresponding requirements in /users/signup.

Restaurants

Requirements:

- As a user, I can create at most one restaurant, of which I will become the **owner**. A restaurant is generally created by specifying its name, address, logo, address, postal code, and phone number.
- As an owner, I want to add/update the menu of my restaurant. A menu is effectively a list of food items, their descriptions, and prices.
- As an owner, I can edit the general information of my restaurant and add/remove pictures to it. As an owner, I can add/remove blog posts to my restaurant.As a user, I can sign up, log in, log out, and edit my profile. Profile information should include first and last name, email, avatar, and phone number.

Endpoint: /restaurants/

Methods: POST, PATCH, DELETE

View class: CreateUpdateDeleteRestaurantView

Authenticated: Yes (return 401 otherwise)

Payload:

```
POST /restaurants
{
     “name”: string,
     “street”: string,
     “city”: string,
     “province”: string
     “postal_code”: string
     “logo”: file,
     “phone_num”: string
     “banner”: file,
     “description”: string
}
```

Description: Allows an logged in user to create, update, or delete their restaurant.

Notes:

-All fields required and of valid format or 400 is returned.

-Creating more than one restaurant returns 403.

-Deleting a non-existent restaurant returns 404.

Endpoint: /restaurants/:id/

Methods: GET

View class: GetRestaurantView

Authenticated: No

Payload:

```
{
     “id”: int,
```

```
    “name”: string,
    “street”: string,
    “city”: string,
    “province”: string,
    “postal_code”: string,
    “logo”: string
    “phone_num”: string,
    “banner”: string
    “description”: string
    “follows”: int
    “likes”: int
}
```

Description: Allows anyone to see restaurant information provided a restaurant id.

Notes:

-Returns 404 if the restaurant with the given restaurant id doesn’t exist.

Endpoint: /restaurants/menu/

Methods: POST

View class: CreateMenuItemView

Authenticated: Yes (return 401 otherwise)

Payload:

```
{
    “name”: string,
    “description”: string,
    “price”: float,
    “image”: file
}
```

Description: Allows the restaurant to create a new menu item.

Notes:

-All fields required and of valid format or 400 is returned.

Endpoint: /restaurants/menu/:id

Methods: PATCH, DELETE

View class: UpdateDeleteMenuItemView

Authenticated: Yes (return 401 otherwise)

Payload:

```
{
     “name”: string,
     “description”: string,
     “price”: float,
     “image”: file
}
```

Description: Allows the restaurant to edit or delete their own menu items.

Notes:

-Returns 404 if the menu with the given menu id doesn’t exist.

-Deleting a non-existent menu item returns 404.

-If the user who is not the owner of the restaurant tries to update a menu item, return 403.

Endpoint: /restaurants/:id/menu?cursor=_

Methods: GET

View class: GetMenuListView

Authenticated: No

Payload:

```
[
     {
          “name”: string,
          “description”: string,
          “price”: float,
          “image”: file
     },
     ...
]
```

Description: Retrieves the entire menu (all menu items) from the restaurant. Returns a list of up to 10 menu items at a time.

Notes:

-Uses cursor pagination.

-Returns 404 if the restaurant with the given restaurant id doesn’t exist.

Endpoint: /restaurants/images/

Methods: POST

View class: CreateImageView

Authenticated: Yes (return 401 otherwise)

Payload:

```
{
     “image”: file,
     “title”: string,
     “description”: string
}
```

Description: Allows the restaurant to add a new image.

Notes:

-All fields required and of valid format or 400 is returned.

Endpoint: /restaurants/images/:id

Methods: PATCH, DELETE

View class: UpdateDeleteMenuItemView

Authenticated: Yes (return 401 otherwise)

Payload:

```
{
     “image”: string,
     “title”: string,
     “description”: string
}
```

Description: Allows the restaurant to edit or delete their own images.

Notes:

-Returns 404 if the image with the given menu id doesn’t exist.
-Deleting a non-existent image returns 404.

-If the user who is not the owner of the restaurant tries to update an image, return 403.

Endpoint: /restaurants/:id/images?cursor=_

Methods: GET

View class: GetImageListView

Authenticated: No

Payload:

```
[
     {
          “image”: string,
          “title”: string,
          “description”: string
     },
     ...
]
```

Description: Retrieves all images from the restaurant. Returns a list of up to 10 images at a time.

Notes:

-Uses cursor pagination.

-Returns 404 if the restaurant with the given restaurant id doesn’t exist.

Endpoint: /restaurants/:id/blog

Methods: POST

View class: CreateBlogPostView

Authenticated: Yes (return 401 otherwise)

Payload:

```
{
     “title”: string,
     “content”: string
}
```

Description: Allows the restaurant to add a new blog post.

Notes:

-All fields required and of valid format or 400 is returned.

Endpoint: /restaurants/blog/:id

Methods: DELETE

View class: DeleteBlogPostView

Authenticated: Yes (return 401 otherwise)

Description: Allows the restaurant to delete their own blog posts.

Notes:

-Returns 404 if the blog with the given blog id doesn’t exist.

-Deleting a non-existent blog returns 404.

-If the user who is not the owner of the restaurant tries to delete the blog, return 403.

Endpoint: /restaurants/:id/blog?cursor=_

Methods: GET

View class: GetBlogPostListView

Authenticated: No

Payload:

```
[
     {
          “title”: string,
          “content”: string
```

```
        “likes”: int,
        “date”: datetime
    },
    ...
]
```

Description: Retrieves all blog posts from the restaurant. Returns a list of up to 10 blog posts at a time.

Notes:

-Uses cursor pagination.

-Returns 404 if the restaurant with the given restaurant id doesn’t exist.

Requirements:

- As a user, I want to search through restaurants by their name, foods, or address. Most popular restaurants should come up first in the results.
- As a user, I can select a restaurant and move to its page, where I can see its general information, images, number of followers, likes, menu, blog posts, and comments.

Endpoint: `/restaurants?types={food|name|address}&search=””&cursor=_`

Methods: GET

View class: GetBlogPostListView

Authenticated: No

Payload:

```
[
    {
        “id”: int,
```

```
        "name": string,
        "street": string,
        "city": string,
        ...
    },
    ...
]
```

Description: Retrieves restaurant results.

Notes:

- Uses cursor pagination.
- Query params are mutually-exclusive
- If type is not one of food, name, or address, 400 is returned.
- Search and cursor params are required.

Requirements:

- As a user, I can like/unlike a restaurant or one or more of its blog posts.
- As a user, I can follow/unfollow a restaurant. The restaurant page should indicate whether I have already followed or liked that restaurant.
- As a user, I want to have a **feed** that lists the blog posts of all restaurants that I follow, showing the newest posts on the top. This feed is not bound to a specific restaurant. It is a separate page from the blog post section of a restaurant.
- As a user, I can post a comment on a restaurant. The comment is about the entire restaurant, not a specific blog post.

Social network

```
Endpoint: /restaurants/:id/like/
Methods: GET, POST
Authenticated: Yes
Notes:
  - GET fetches whether you currently like the restaurant with
```

id.
- POST toggles whether you like the restaurant with id.
- 404 is returned when an invalid id is provided.
- 400 is returned if one attempts to like their own restaurant.

Endpoint: /restaurants/:id/follow/

Methods: GET, POST

Authenticated: Yes

Notes:

- GET fetches whether you currently follow the restaurant with id.
- POST toggles whether you follow the restaurant with id.
- 404 is returned when an invalid id is provided.
- 400 is returned if one attempts to follow their own restaurant.

Endpoint: /restaurants/blog/:id/like/

Methods: GET, POST

Authenticated: Yes

Notes:

- GET fetches whether you currently like the blog with id.
- POST toggles whether you like the blog with id.
- 404 is returned when an invalid id is provided.
- 400 is returned if one attempts to like their own blog.

Endpoint: /users/feed?cursor=_

Methods: GET

View class: FeedView

Authenticated: Yes

Payload:

```
[
     {
          “title”: string,
          “restaurant”:{
```

```
            “id”: int,
            “name”: string
        },
        “content”: string
        “likes”: int,
        “date”: datetime
    },
    ...
]
```

Description: Retrieves all the feed of a user. Returns a list of up to 10 elements at a time.

Notes:

-Uses cursor pagination.

Endpoint: /restaurants/:id/comments and /restaurants/:id/comments?cursor=_

Methods: POST, GET

View class: GetCreateCommentsView

Authenticated: Yes (return 401 otherwise)

Payload:

```
{
    “content”: string
}
```

Return:

```
[
    {
        “user”: {
            “id”: int,
            “name”: string,
```

```
            “avatar”: string
        },
        “content”: string,
        “likes”: int,
        “date”: datetime
    },
    ...
]
```

Description: Allows the user to add a comment to add a restaurant given the restaurant id.

Notes:

-All fields required and of valid format or 400 is returned.

-Uses cursor pagination.

-Returns 404 if the comment with the given comment id doesn’t exist.

Notifications

Requirements:

- As an owner, I want to see notifications. I want to be notified when someone follows, likes, and posts a comment on my restaurant. Likes could be either to the whole restaurant or a specific blog post.
- As a user, I want to see notifications. I want to be notified when a restaurant I follow makes a new blog post or updates its menu.

Notes:

Paginated using “cursor” and “page_size” query params. These parameters are not required, cursor defaults to 0 and page_size defaults to 10.

Description: Returns a list of notifications for a user

Endpoint: `/notifications`

Methods: `GET`

View method: `notifications`

Authenticated: `Yes (return 401 otherwise)`

Payload:

```
[
     {
          “type”: “comment”,
          “comment”: {
               “id”: int,
               “content”: string,
               “date”: datetime
          },
          “user”: {
               “id”: int,
               “first_name”: string,
               “last_name”: string
          }
          “timestamp”: datetime
     },
     {
          “type”: “like”
          “kind”: Post | Restaurant,
          “user”: {
               “id”: int,
               “first_name”: string,
               “last_name”: string
          }
          “timestamp”: datetime
     },
     {
```

```
          “type”: “follow”,
          “user”: {
               “id”: int,
               “first_name”: string,
               “last_name”: string
          }
          “timestamp”: datetime
     },
     ...
]
```

Description: Retrieves all notifications for an user/owner.

Notes:

-Uses cursor pagination.

-Returns 404 if the restaurant with the given restaurant id doesn’t exist.