Project Structure:

In our project, we decided to divide our models and endpoints into two applications, users and restaurants. The Users app focuses on user-related models, such as CustomUser (which describes a customer or owner), and notifications that are invoked between users. Users' app also contains both register views for customer and owner (and their restaurant), as well as feed view, notification view, profile view, and index view, all of which interact with the user. Restaurant application focuses on models and views related to the actual restaurants themselves. Therefore, it contains models that pertain to it, such as Restaurant, Menu item, blog post, blog post likes, comments, and followers, all of which are contained in a restaurant's ecosystem. The views that are contained within the restaurant app include restaurant view (for info), menu view and menu item view for a restaurant, comment/comments view for all or individual comments of a restaurant, as well as follower and blogpost/blogpost likes view for a restaurant. Authorization checks are also made when users of a given object, such as a menu item or blog post, are changed or updated.

Apps: 2 apps

Users and Restaurants

Models: 8 models

Notification: Hritik
ID: Primary Key
Recipient ID: Foreign key to a user receiving the notification
Sender ID: Foreign key to a user invoking the notification
Content: The content of the notification
Read: Boolean that is true if this notification has been read
Date: The DateTime that the notification occurred

Restaurant: Hritik
ID: Primary Key
Owner ID: Foreign key to a user
Name: Name of restaurant
Bio: The description of the restaurant
Address: The address
FollowersCount: The number of followers for this restaurant
Phone: Phone number
LogoURL: URL for Logo of the restaurant
CoverURL: URL for background cover image for restaurant posts

Followers: Hritik
ID: Primary Key
User ID: FK to User

Restaurant ID: FK to restaurant

MenuItem: **Wolfgang**
ID: Primary Key
Restaurant ID: FK To Restaurant that owns this item
Name: Name of the item
Content: Description of the item
Picture URL: Optional field for picture URL
Price: Price of item

BlogPost: Wolfgang
ID: Primary Key
Restaurant ID: FK To Restaurant
Title: Title of the post
LikeCount: The number of likes for this post
Content: Description of the post
DateTime: The date and time the post was created

BlogPostLikes: Wolfgang
ID: Primary key
BlogPostID: FK to BlogPost
UserID: FK to user

Comment: Ramel
ID: Primary Key
User ID: FK to user
RestaurantID: FK to restaurant
Content: The content of the comment
Date: The DateTime that the notification occurred

CustomUser (extends AbstractUser) : Ramel
ID: Primary Key
Email: email of the user
Password: password of the user
Phone_number: phone number of the user
First_name: first name of the user
Last_name: last name of the user
Profile Picture: URL to user profile picture
Cover Picture: URL to cover picture for profile

Django Endpoints: 15 endpoints

APPLICATION: USER

/register/customer/: **Wolfgang**
POST: Registers customer users if the given information is valid and displays their info after registering as a JSON object, else returns an error for missing information

- Payloads:
    - Charfield/text: username, password1, password2, first_name, last_name, email, pfp_url, cover_url
    - Integerfield/integer: phone_number

/register/owner/: **Wolfgang**
POST: Registers owner user and their restaurant if the information is valid and displays restaurant and owner info after registering as a JSON object, else returns an error for missing information

- Payloads:
    - Charfield/text: username, password1, password2, first_name, last_name, email, pfp_url, cover_url, name, bio, address, logo, cover
    - Integerfield/integer: phone_number, phone

/login: Ramel
POST: Posts email and password, returns JSON object with refresh and access tokens on success, displays errors on fields otherwise

- Payloads:
    - Charfield/text: username, password

/feed/: Ramel
GET: Retrieves list of restaurants currently logged in user is subscribed to

/index: Hritik
GET: If supplied query string for search, uses that to filter restaurants based on a string. Otherwise, returns a list of all restaurants

- Payloads:
    - Q (to query or not), type (name, item, address)

/profile: Hritik
GET: Returns currently logged in user's profile information
PUT/PATCH: Edits currently logged-in user's profile information.

- Payloads:
    - Charfield/text: username, password, first_name, last_name, email, pfp_url, cover_url
    - Integerfield/integer: phone_number

/notifications: Hritik
GET: fetch current logged in user's notifications and set them to read

APPLICATION: RESTAURANT

restaurant/<restaurant>/info/: Wolfgang
Description: Basic restaurant information, such as name, address, cover picture, phone number
GET: Retrieves the current restaurant information
PUT/PATCH: Send payload that will update the fields if the user is the owner of the restaurant

- Payloads:
    - Charfield/text:name, bio, address, logo, cover
    - Integerfield/integer: phone

restaurant/<restaurant>/comments: Ramel
GET: retrieve all comments
POST: Creates a new comment from the logged-in user with a text payload that holds the comment. Create a new notification for the owner user of the restaurant.

- Payloads:
    - Charfield/text: content (comment contents)

restaurant/comments/<id>: Ramel
GET: Retrieve comment using its id

restaurant/<restaurant>/followers: Ramel
GET: Retrieves all followers
POST: add the logged-in user to the followers model with reference to <restaurant>. Create new notification for the owner of restaurant. Nothing happens if user is already following restaurant.

restaurant/<restaurant>/menu: Wolfgang
GET: Retrieves all menu items that the specified restaurant has created
POST: add a new menu item model instance, and validate that the user is the owner of <restuarant>

- Payloads:
    - Charfield/text: name, content, picture
    - Integerfield/integer: price

restaurant/menu/<pk>: Wolfgang
GET: Retrieve menu item using its id
PUT/PATCH: send payload that will update/change the fields of a given menu item if a user is the owner of the given restaurant who owns the menu
DELETE: deletes the specified menu item with <pk> if a user is the owner of the restaurant that owns the item.

- Payloads:
    - Charfield/text: name, content, picture

- Integerfield/integer: price

restaurant/<restaurant>/blogposts: Hritik
GET: Retrieves all blog posts for the restaurant
POST: add a new blog post model instance, and validate that the user is the owner of <restuarant>. Create a new notification for all users that follow this restaurant.

Payload:
Charfield: Title, Content

restaurant/blogposts/<pk>/: Hritik
GET: Retrieves the blogpost with the id <pk>
DELETE: Deletes the blogpost with id <pk> if the currently logged in user is the owner of the restaurant the blogpost belongs to.

restaurant/blogposts/<blogpost>/like: Ramel
GET: Retrieves all users that liked the post
POST: the currently logged-in user creates an entry in the Likes model and notifies the restaurant owner that the post has been liked