project structure: we have 3 applications, ‘account’, ‘restaurants’, and ‘social_network’, each has its own models, serializers, and views

list of apps for phase2:

- 'django.contrib.admin': the admin interface
- 'django.contrib.auth': the authentication system
- 'django.contrib.contenttypes': tracks all of the models installed in the project and provides a generic interface for models
- 'django.contrib.sessions': user sessions
- 'django.contrib.messages': displaying a message to the user after processing a user input
- 'django.contrib.staticfiles': collects static files from applications and stores it into a single location
- 'rest_framework': tool for building web APIs
- 'accounts': all user and admin related account activities for this project
- 'restaurants': all users restaurant
- 'social_network': all processes related to blogs comments and activities for likes and dislikes

design of models:

- accounts:
    - the model UserAccount extends an AbstractUser with an additional fields for a user avatar(ImageField) and phone_num(CharField)
- restaurants:
    - the model Restaurants has a field name(CharField), address(CharField), logo(ImageField), postal_code(CharField), phone_num(CharField), owner(UserAccount)-> the user that owns the restaurant.
    - the model RestaurantMenus has a field name(CharField), img(ImageField), price(FloatField), restaurant(ForeignKey to the model Restaurants)
- social_network:
    - the model RestaurantBlogs has a field date(DateTimeField, records the date and time of the blog post being created), title(CharField), content(CharField), likes_num(IntegerField), dislikes_num(IntegerField), poster(ForeignKey to the model UserAccount), restaurant(ForeignKey to the model Restaurants)
    - the model RestaurantBlogsImg has a field img(ImageField), blog(ForeignKey to the model RestaurantBlogs)
    - the model Comments has a field date(DateTimeField, records the date and time of the blog post being created), score(IntegerField),

content(CharField), likes_num(IntegerField), dislikes_num(IntegerField), poster(ForeignKey to the model UserAccount), restaurant(ForeignKey to the model Restaurants)

- the model Reply has a field date(DateTimeField, records the date and time of the blog post being created), poster(ForeignKey to the model UserAccount), comment(ForeignKey to the model Comments)
- the model Notification has a field from_user(ForeignKey to the model UserAccount), to_user(ForeignKey to the model UserAccount), time(DateTimeField, records the date and time of the notification being created), target_url(URLField), message(CharField)
- the model Following has a field follow_status(BooleanField), like_status(BooleanField), user(ForeignKey to the model UserAccount), restaurant(ForeignKey to the model Restaurants)
- the model Rated has a field blog(ForeignKey to RestaurantBlogs), users(ForeignKey to the model UserAccount)

Endpoints:

`localhost:8000/`

- `accounts/users/`
  method(s): **GET**
  description:
  admin can view all users account info

- `accounts/user/edit/`
  method(s): **GET**
  description:
  for the current logged in user,, they can edit and update their own account info

- `accounts/user/<int:id>/`
  method(s): **GET**
  description:

admin can view the user account <id>, if the user <id> doesn't exist, a 404 response is sent

- `accounts/user/signup/`
  method(s): **POST**
  payload:
  username = a unique string representation of a user's username
  first_name = a string representation of a user's first name
  last_name = a string representation of a user's last name
  email = a unique string representation of a user's email
  password1 = 8 character long password
  password2 = re enter password1
  phone_num = an int representation of a user's phone number
  description:
  the signup process of a new user, the entered username, email, and phone number must be unique, and password1 needs to match password2

- `accounts/user/login/`
  method(s): **GET**
  description:
  users can login with their username and password to retrieve an access token

- `restaurants/create_restaurants/`
  method(s): **POST**
  payload:
  name: string representing the name of the restaurant
  address: string representing the address of the restaurant
  logo: the picture of the restaurant
  postal_code: string representing the postal code of the restaurant
  phone_num: string representing the phone number of the restaurant
  owner: string representing the owner of the restaurant
  description:
  create a new restaurant which belongs to the login user if the user does not have one,, and each user can only have one restaurant, and the restaurant include those attributes:
  name, address, logo, postal_code, phone_num, owner
- `restaurants/delete_restaurants/`
  method(s): **DELETE**
  description:
  delete the restaurant if exist
- `restaurants/get_restaurants/<int:id>`
  method(s): **GET**
  payload:
  name: string representing the name of the restaurant
  address: string representing the address of the restaurant

logo: the picture of the restaurant
postal_code: string representing the postal code of the restaurant
phone_num: string representing the phone number of the restaurant
owner: string representing the owner of the restaurant

description:

list all the attributes that the restaurant have if it exist, and the id of the restaurant is in the url which is id, so that we can list the restaurant that we want with its restaurant id

- `restaurants/update_restaurants/`

method(s): `PUT`

payload:

name: string representing the name of the restaurant
address: string representing the address of the restaurant
logo: the picture of the restaurant
postal_code: string representing the postal code of the restaurant
phone_num: string representing the phone number of the restaurant

description:

update the attributes of the restaurant but not include the owner, because owner cannot be modify

- `restaurants/get_restaurant_menu/<int:restaurant_id>`

method(s): `Get`

description:

get all the menu of certain restaurants if exist

- `restaurants/create_restaurant_menu/`

method(s): `POST`

payload:

name: the name of the menu
img: the img url of the menu
price: the price of the menu
restaurant: id of the restaurant the blog is posted to

description:

create a menu of current user's restaurants if restaurant exists and not the same menu exists in the current restaurant

- `restaurants/update_restaurant_menu/`

method(s): `PUT`

payload:

name: the name of the menu
img: the img url of the menu
price: the price of the menu
new_name: the new name of the menu

description:

create a menu of current user's restaurants if restaurant exists and there has such menu exists in the current restaurant and the new_name will not conflict with the current name

- `restaurants/delete_restaurant_menu/`

  method(s): **DELETE**

  payload:

  name: the name of the menu

  img: the img url of the menu

  price: the price of the menu

  new_name: the new name of the menu

  description:

  delete a menu of current user's restaurants if restaurant exists and there has such menu exists in the current restaurant

- `social_network/create_blog/`

  method(s): **POST**

  payload:

  content: string representing the blog's main content

  title: string representing the blog's title

  restaurant: id of the restaurant the blog is posted to

  description:

  post a new blog to the restaurant specified, while notifying any following users in the process. If no authentication token is provided, a 401 response is sent. If the user is not the owner of the target restaurant, the request is responded with a 403. If the restaurant does not exist, a 404 response is sent.

- `social_network/get_blog/<int:id>`

  method(s): **GET**

  description:

  responses with a Blog specified by <id> in the URL dispatcher. If no blog exists with the given id, a 404 response is given.

- `social_network/delete_blog/`

  method(s): **DELETE**

  payload:

  id: id of the blog that needs to be deleted

  description:

  deletes the specified blog. If no authentication token is provided, a 401 response is sent. If the user is not the owner of the restaurant the blog is

posted to, the request is responded with a 403. If the blog does not exist, a 404 response is sent.

- `social_network/follow/`
  method(s): **`PUT`**, **`GET`**
  payload:
  restaurant: id of the restaurant the user wants to follow
  description:
  make the currently logged-in user follow the restaurant, notifying the restaurant owner in the process. If it's already followed, a 403 response is sent. If no authentication token is provided, a 401 response is sent. If the restaurant does not exist, a 404 response is sent. In all error cases, no change is made for the relationship between the user and the restaurant.
- `social_network/unfollow/`
  method(s): **`PUT`**, **`GET`**
  payload:
  restaurant: id of the restaurant the user wants to unfollow
  description:
  make the currently logged-in user unfollow the restaurant. If it's already not followed, a 403 response is sent. If no authentication token is provided, a 401 response is sent. If the restaurant does not exist, a 404 response is sent. In all error cases, no change is made for the relationship between the user and the restaurant.
- `social_network/like_restaurant/`
  method(s): **`PUT`**, **`GET`**
  payload:
  restaurant: id of the restaurant the user wants to like
  description:
  make the currently logged-in user like the restaurant, notifying the restaurant owner in the process. If the user already liked the restaurant, a 403 response is sent. If no authentication token is provided, a 401 response is sent. If the restaurant does not exist, a 404 response is sent. In all error cases, no change is made for the relationship between the user and the restaurant.
- `social_network/unlike_restaurant/`
  method(s): **`PUT`**, **`GET`**
  payload:
  restaurant: id of the restaurant the user wants to dislike
  description:
  make the currently logged-in user dislike the restaurant. If the user already disliked the restaurant a 403 response is sent. If no authentication token is provided, a 401 response is sent. If the restaurant does not exist, a 404 response is sent. In all error cases, no change is made for the relationship between the user and the restaurant.
- `social_network/like_blog/`
  method(s): **`PUT`**, **`GET`**

payload:
id: id of the blog the user wants to like
description:
make the currently logged-in user like the blog, notifying the restaurant owner in the process. If the user already liked/disliked the blog a 403 response is sent. If no authentication token is provided, a 401 response is sent. If the blog does not exist, a 404 response is sent. In all error cases, no change is made for the relationship between the user and the restaurant.

- `social_network/unlike_blog/`
  method(s): `PUT, GET`
  payload:
  id: id of the blog the user wants to dislike
  description:
  make the currently logged-in user dislike the blog. If the user already liked/disliked the blog a 403 response is sent. If no authentication token is provided, a 401 response is sent. If the blog does not exist, a 404 response is sent. In all error cases, no change is made for the relationship between the user and the restaurant.
- `social_network/getfeed/`
  method(s): `GET`
  requirements:
  a query string `?page=n` where n is an integer representing page number should be added to the end of the URL for pagination to work properly
  description:
  get a list of blog posts from followed restaurants, sorted in time descent order. If no authentication token is provided, a 401 response is sent.
- `social_network/create_comment/`
  method(s): `POST`
  payload:
  restaurant: id of the restaurant the user wants to post a comment on
  score: integer from 0 to 5 representing the rating given by the user
  content: a string representing the content of this comment
  description:
  creates a new comment under the specified restaurant, notifying the restaurant owner in the process. If no authentication token is provided, a 401 response is sent. If the restaurant does not exist, a 404 response is sent.