Documentation

Project Structure

Description

With our project we have opted to use two primary apps, one for handling user functionality, and a second for handling any restaurant backend functionality. In the Accounts app our models essentially handle user accounts management and notifications that are sent to each user.
In the Restaurants app the models turn every aspect of the social network functionality, such as posts, following, liking, commenting into their own entities when needed and we also have a primary model for restaurants which references other models like Menu that represent the menu containing food item objects of a specific restaurant in Restify.
We use Generic APIViews to handle the creation, deletion, retrieval and updating of these objects on the backend. Token authentication is used for user accounts authentication for logging users in.

Apps:

- Restaurants
- Accounts

Models

Accounts App:
- UserProfile(AbstractUser)
    - **Fields**: phone_number, profile_picture
- Notifications
    - **Fields:** type, date, to_user, from_user, post, restaurant

Restaurants App:
- Post
    - **Fields**: title, owner, picture, topic, description, created
- Restaurant
    - **Fields**: owner, name, phone_number, address, postal_code, email, logo
- RestaurantImage
    - **Fields:** img, restaurant
- FoodItem
    - **Fields:** name, description, price, menu
- Menu
    - **Fields:** menu_name, restaurant
- Comment
    - **Fields:** title, text, author, rating, restaurant

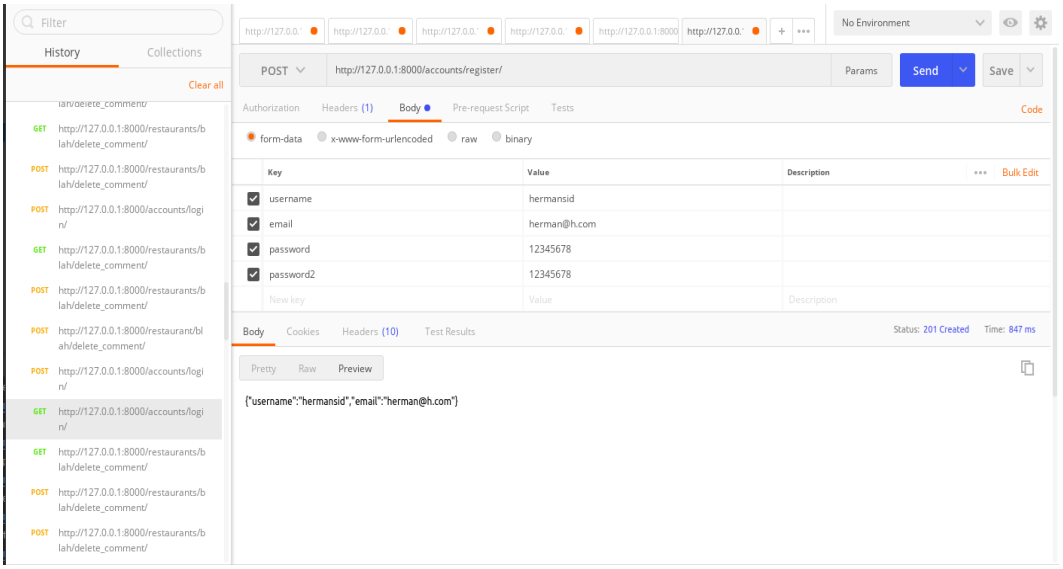
Accounts

Endpoints:

POST	/accounts/register/	Checks validity of sign up form data and creates new user object

Fields Required:
username
email
password
password2

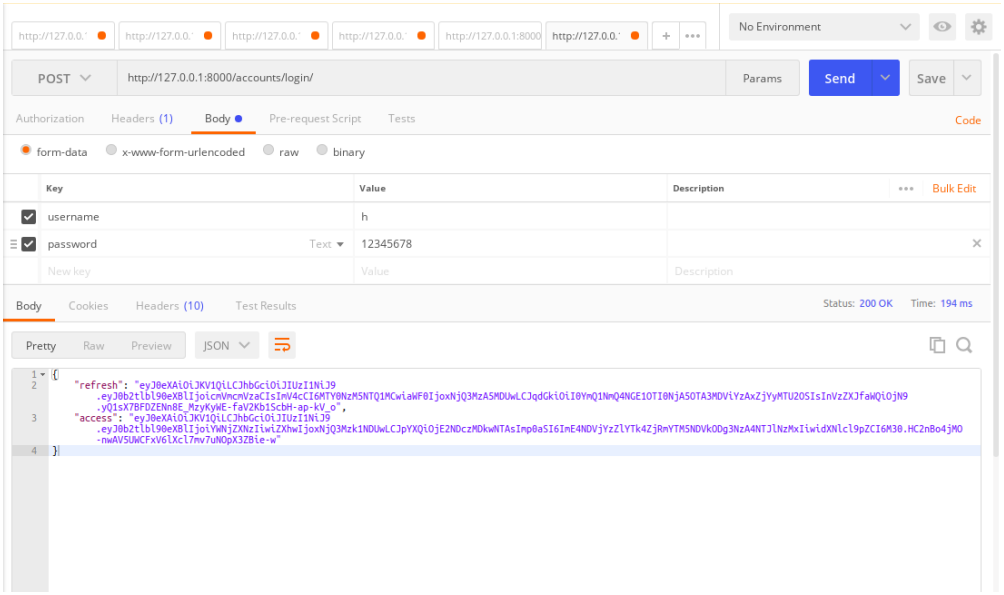
Example Request & Response



POST	/accounts/login/	Obtains access token based on login credentials entered for authentication

No fields required, once Access token is given, copy the access token string and add to Headers section as a new key ‘Authorization’, the value of this key will be ‘Bearer ’ followed by the access token string.

Example Request & Response

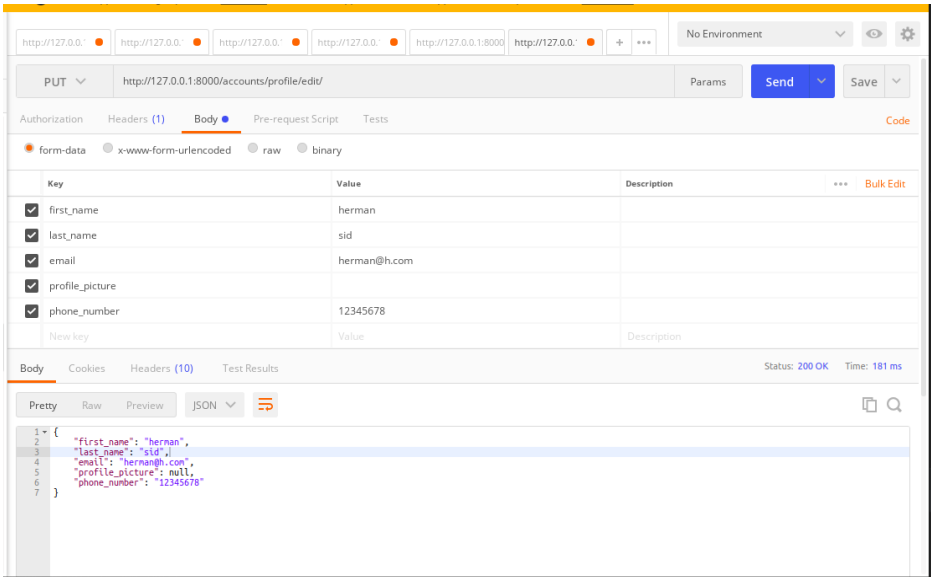


PUT	/accounts/profile/edit/	Makes changes to profile data of user after retrieving the user object associated with authenticated user

<u>Fields required:</u>
first_name
last_name
email
profile_picture
phone _number

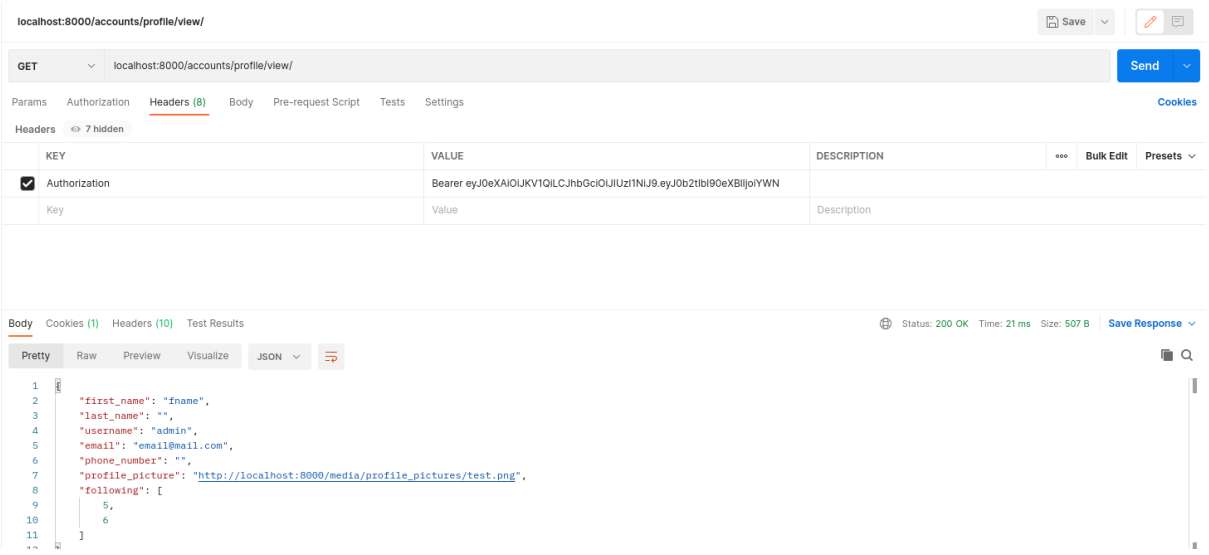
Of the above fields, it is only mandatory to fill email, rest are optional

Example Request & Response



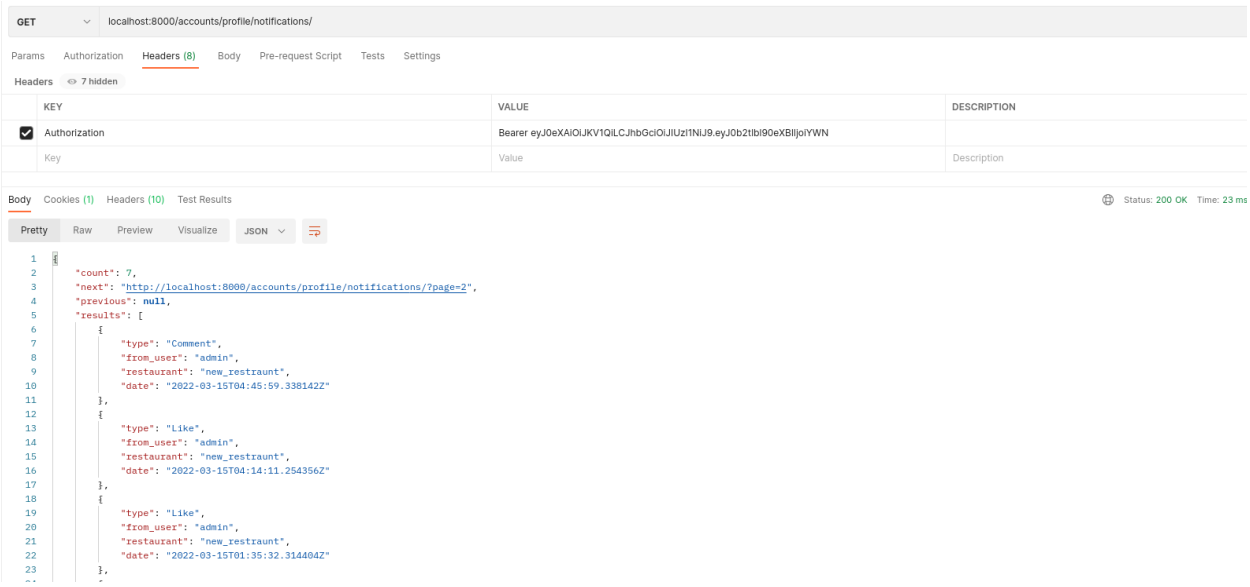
GET	/accounts/profile/view/	Returns the information about the currently logged in user: first name, last name, username, email, phone number, profile picture, and a list of restaurants the user is following.

Example Request & Response



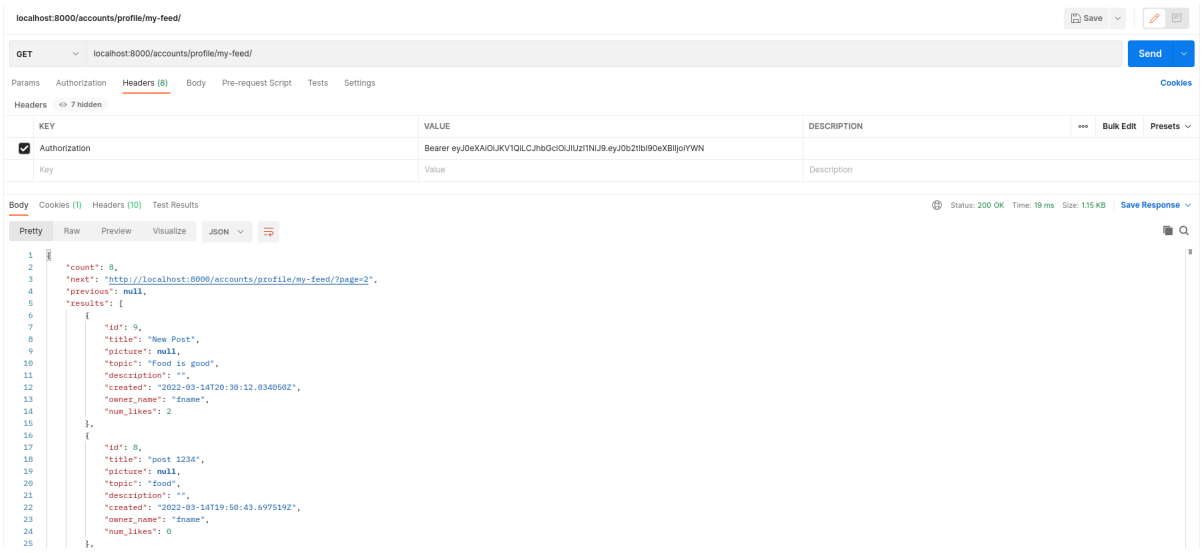
GET	/accounts/profile/notifications/	Returns a list of the currently logged in user’s notifications ordered with the newest notification first. A user gets a notification when another user likes, comments, or follows their restaurant or when a restaurant they follow makes a post.

Example Request & Response



GET	/accounts/profile/my-feed/	Returns a list of all the posts from restaurants that the user follows sorted by creation date with the newest first.

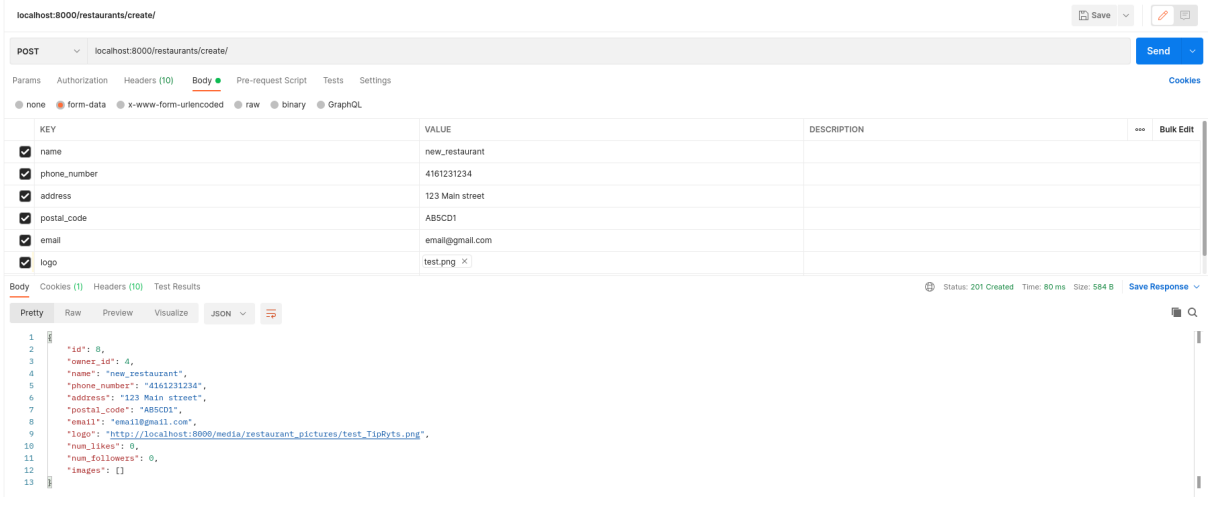
Example Request & Response



Restaurant

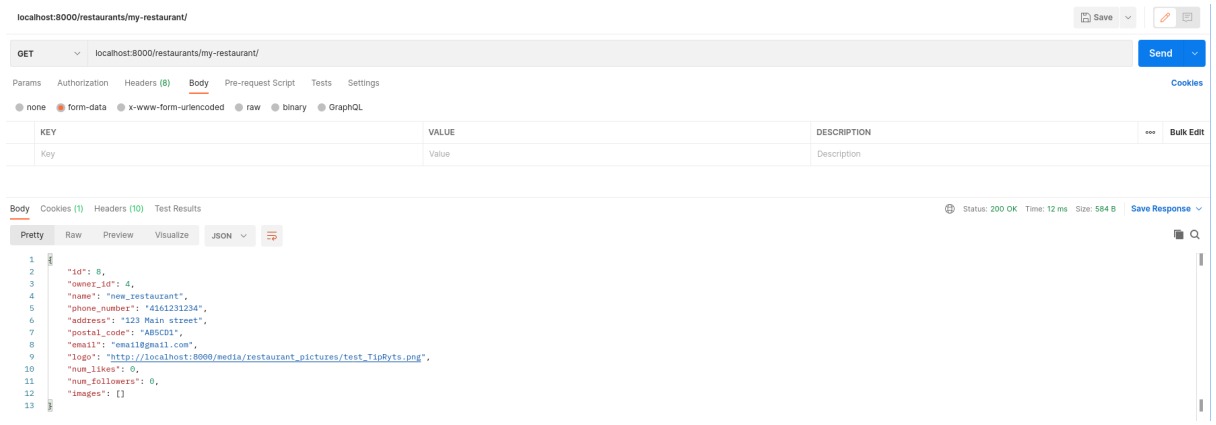
POST	/restaurants/create/	Creates a new restaurant with the owner set to the currently logged in user. If the user already has a restaurant, then it returns an error message. Fields: Name - **required** Phone Number - **required** Address - **required** Postal code - **required** Email - **required** Logo - **required**

Example Request & Response



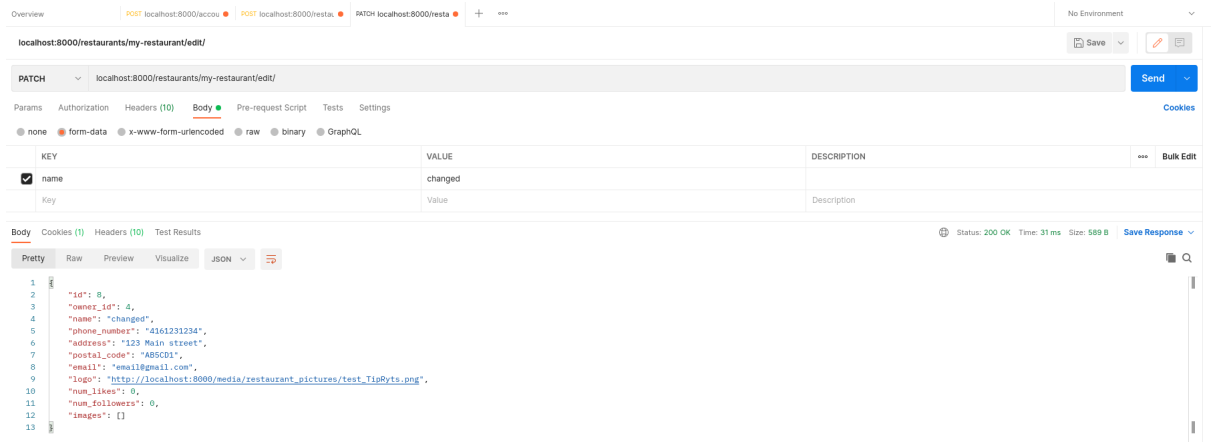
GET	/restaurants/my-restaurant/	Returns the information about your restaurant. Returns 404 if the user does not have a restaurant.

Example Request & Response



GET & PUT/PATCH	/restaurants/my-restaurant/edit/	The GET request will return the information about the logged in users restaurant in the same way as above. A PUT/PATCH request will update the restaurant's information.

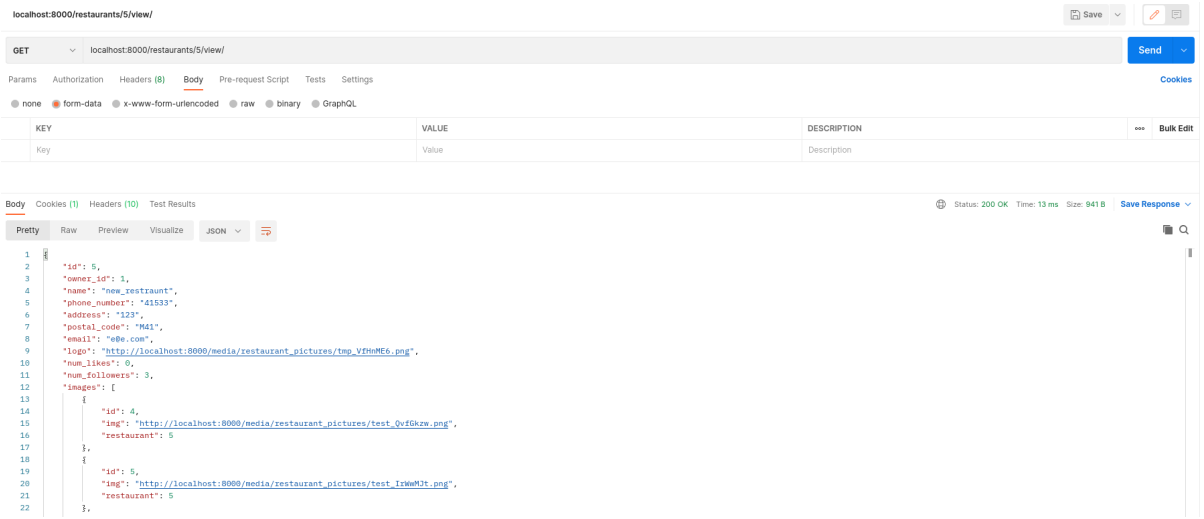
Example Request & Response



Here the name attribute was changed to “changed”. The other attributes can be updated similarly.

GET	/restaurants/<int:rid>/view/	This will return the information about an arbitrary restaurant where <int:rid> is the id of the restaurant we want to view.

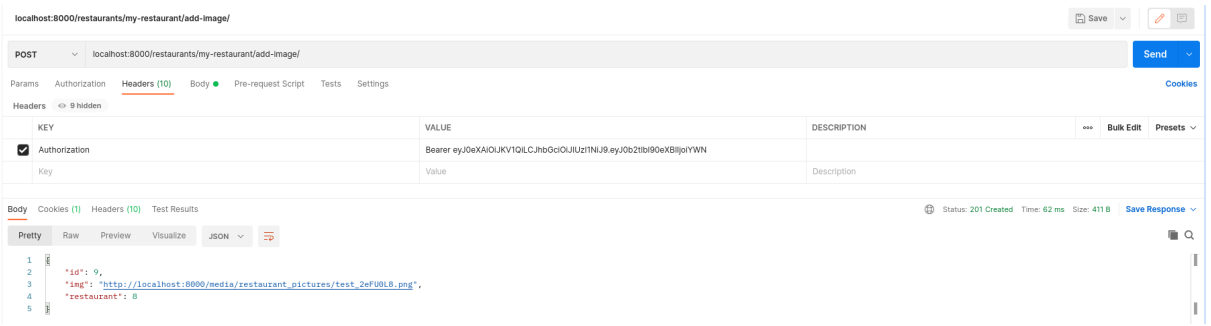
Example Request & Response



Here we are viewing the restaurant with id 5.

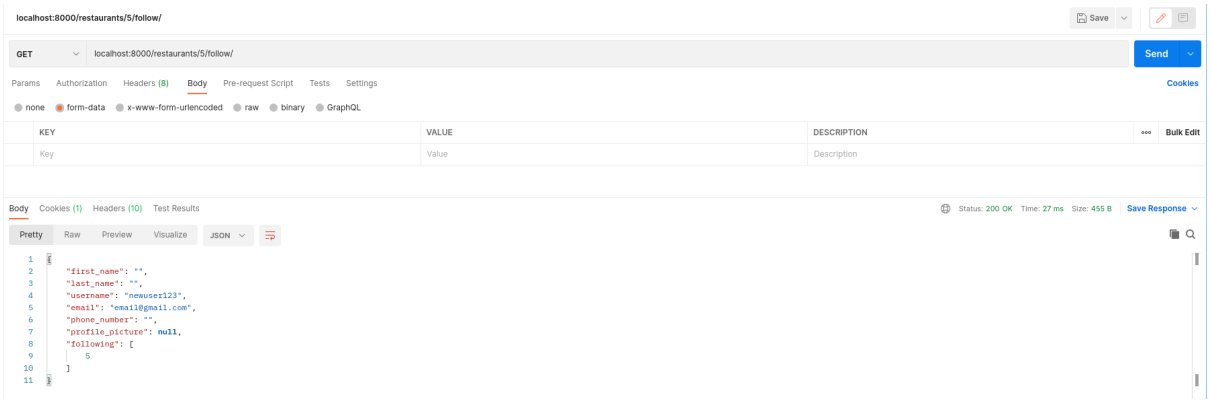
POST	/restaurants/my-restaurant/add-image/	This will add an image to the restaurant that is owned by the currently logged in user. If the user does not own a restaurant, it will return an error message. Fields Img - **required**

Example Request & Response



GET	/restaurants/<int:rid>/follow/	Follow the restaurant with id *rid*. Returns 404 if a restaurant with rid does not exist. Otherwise, it will add the restaurant to the list of the user's followed restaurants and return the current user object.

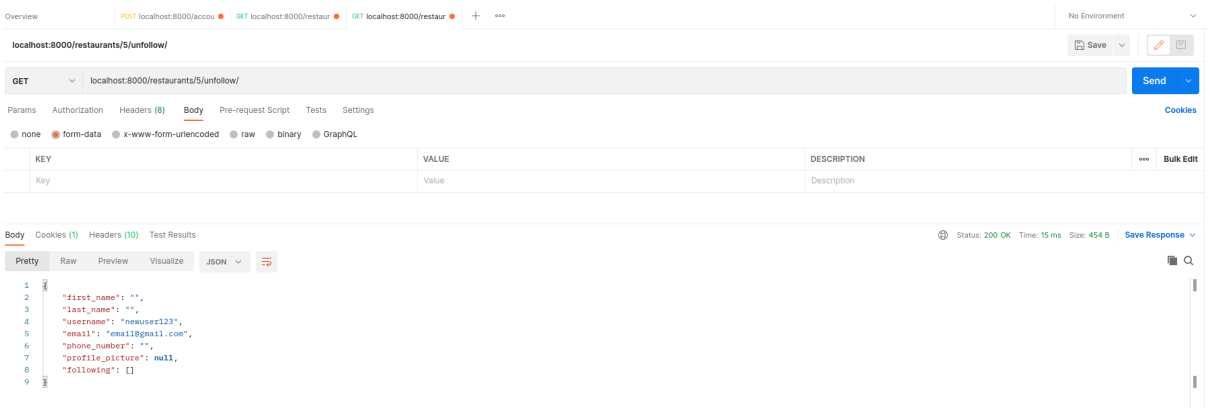
Example Request & Response



The user is now following the restaurant with id=5.

GET	/restaurants/<int:rid>/unfollow/	Unfollow the restaurant with id *rid.* If the user is not following the restaurant, it will do nothing. Otherwise the restaurant is removed from the users following list and the user object is returned.

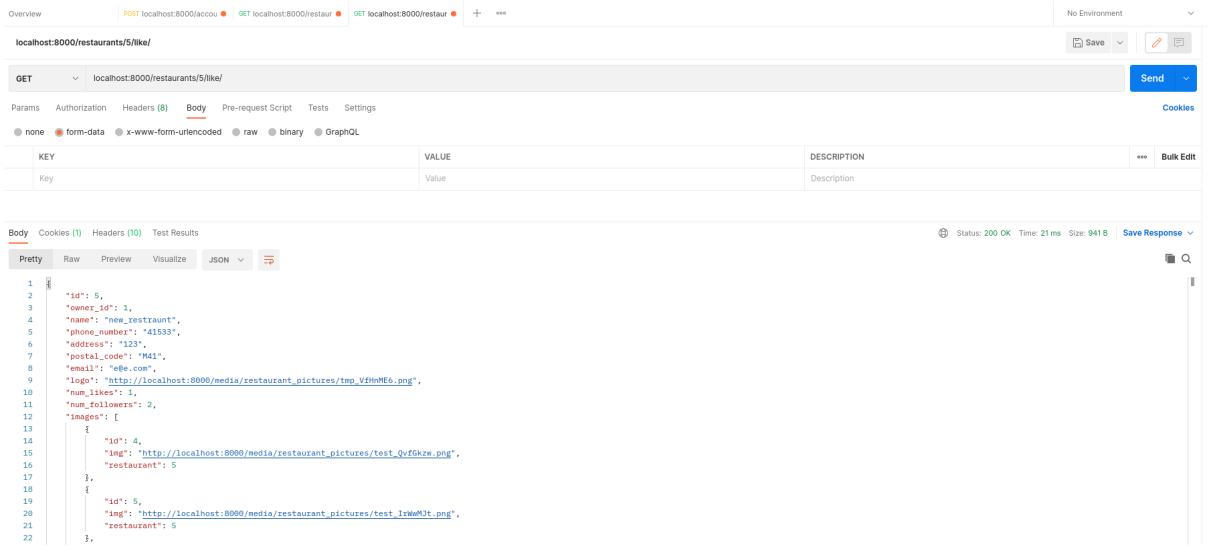
Example Request & Response



Unfollows the restaurant with id=5.

GET	/restaurants/<int:rid>/like/	Likes the restaurant with id *rid* for the currently logged in user. Returns the restaurant object with id=rid.

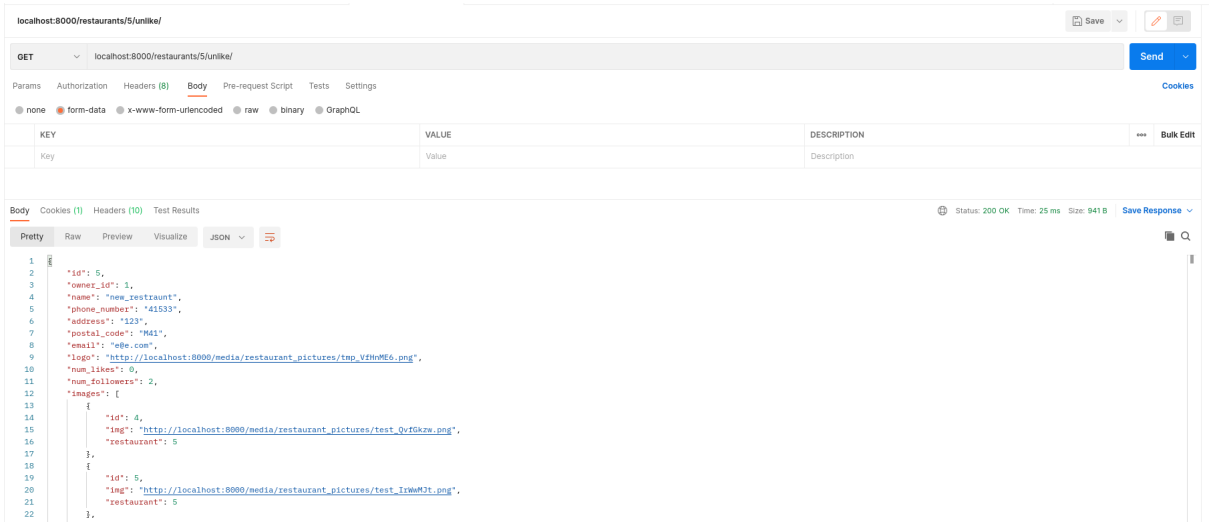
Example Request & Response



The logged in user likes the restaurant with id=5.

GET	/restaurants/<int:rid>/unlike/	Unlikes the restaurant with id *rid* for the logged in user. Returns the restaurant object with id=rid.

Example Request & Response



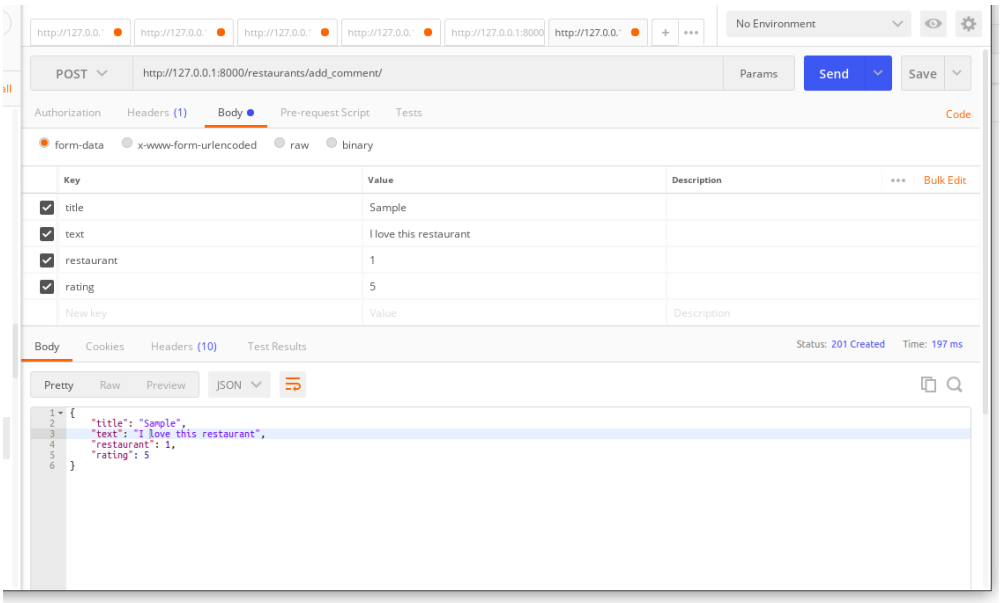
Unlikes the restaurant with id=5.

Comments

POST	/restaurants/add_comment/	Adds a comment to a restaurant by authenticated user

Fields required:
title: string
text: string
restaurant: Integer ID
rating: integer between 0 and 5

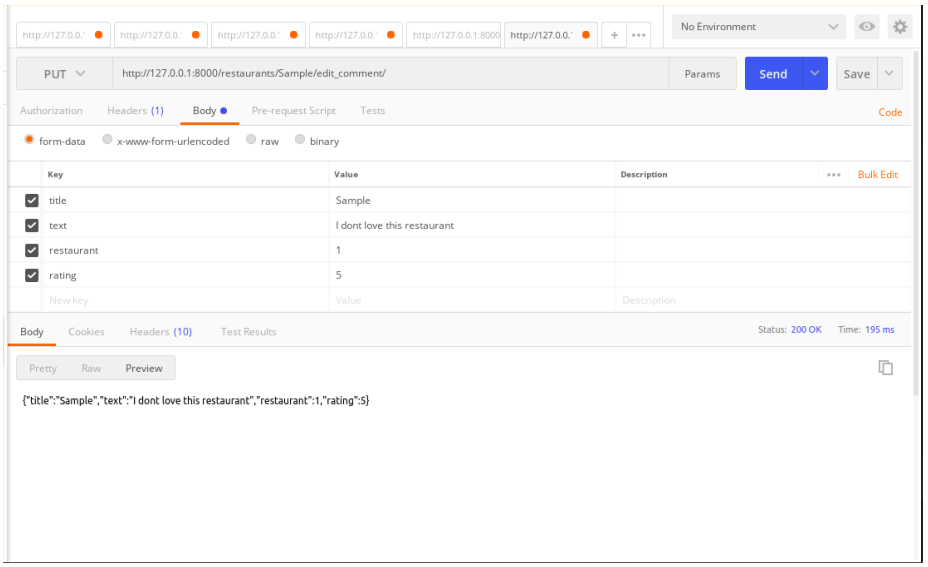
Example Request & Response



PUT	/restaurants/<str:title>/edit_comment	Retrieves comment from its title given in URL and makes changes to it if user is the author of comment

Fields required:
title: string
text: string
restaurant: Integer ID
rating: integer between 0 and 5

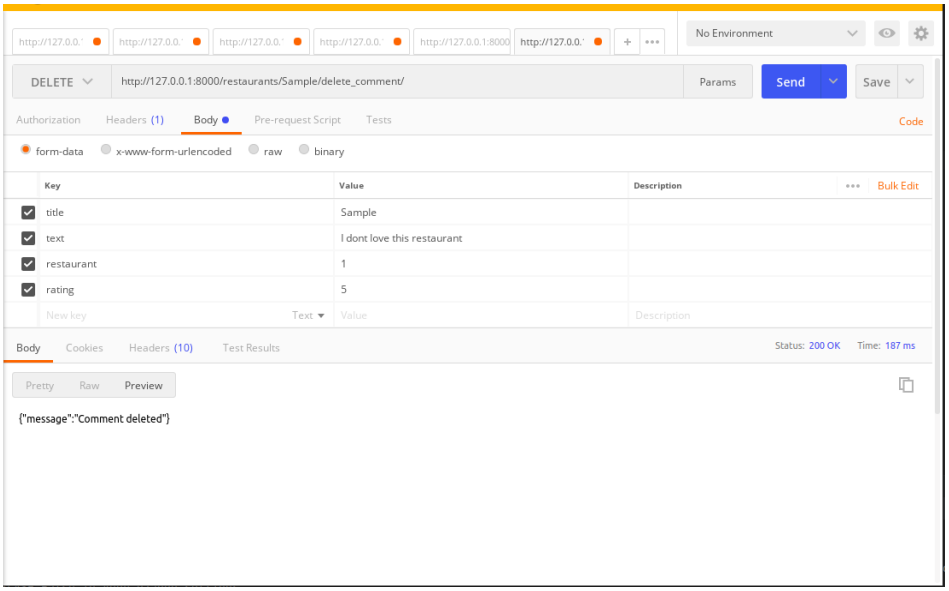
Example Request & Response



DELETE	/restaurants/<str:title>/delete_comment	Retrieves comment based on URL provided title and deletes it if user is the author of the comment

No fields required

Example Request & Response



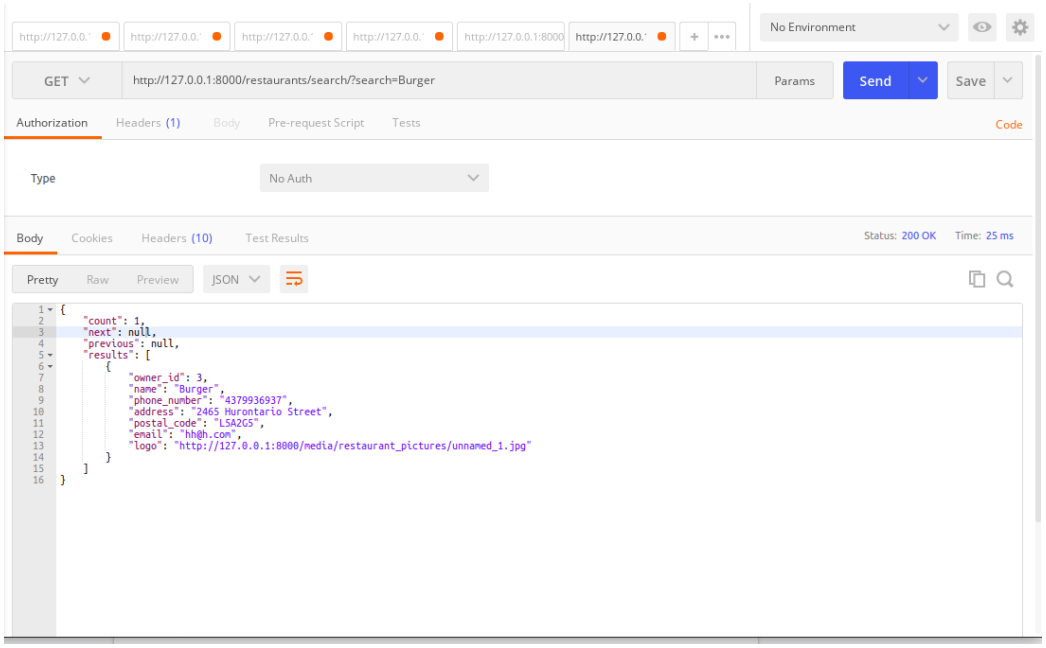
Restaurant Search

Endpoints:

GET	/restaurants/search/	Conducts a search for restaurant object(s) based on search input

For this no fields are required however to conduct the search, an argument ?search is required to be filled in the url, Thus if I wanted to search for keyword ‘Burger’ the url would be **/restaurants/search/?search=Burger**

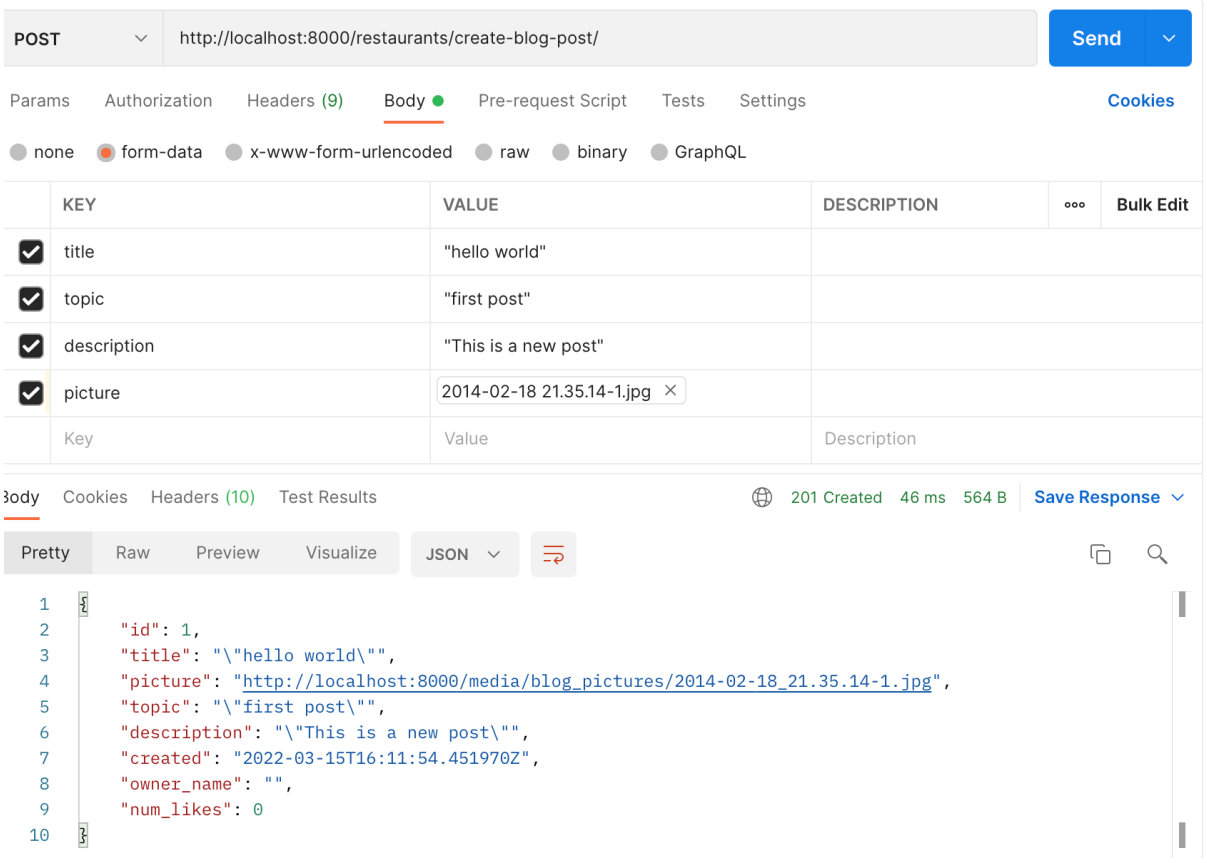
Example request and response



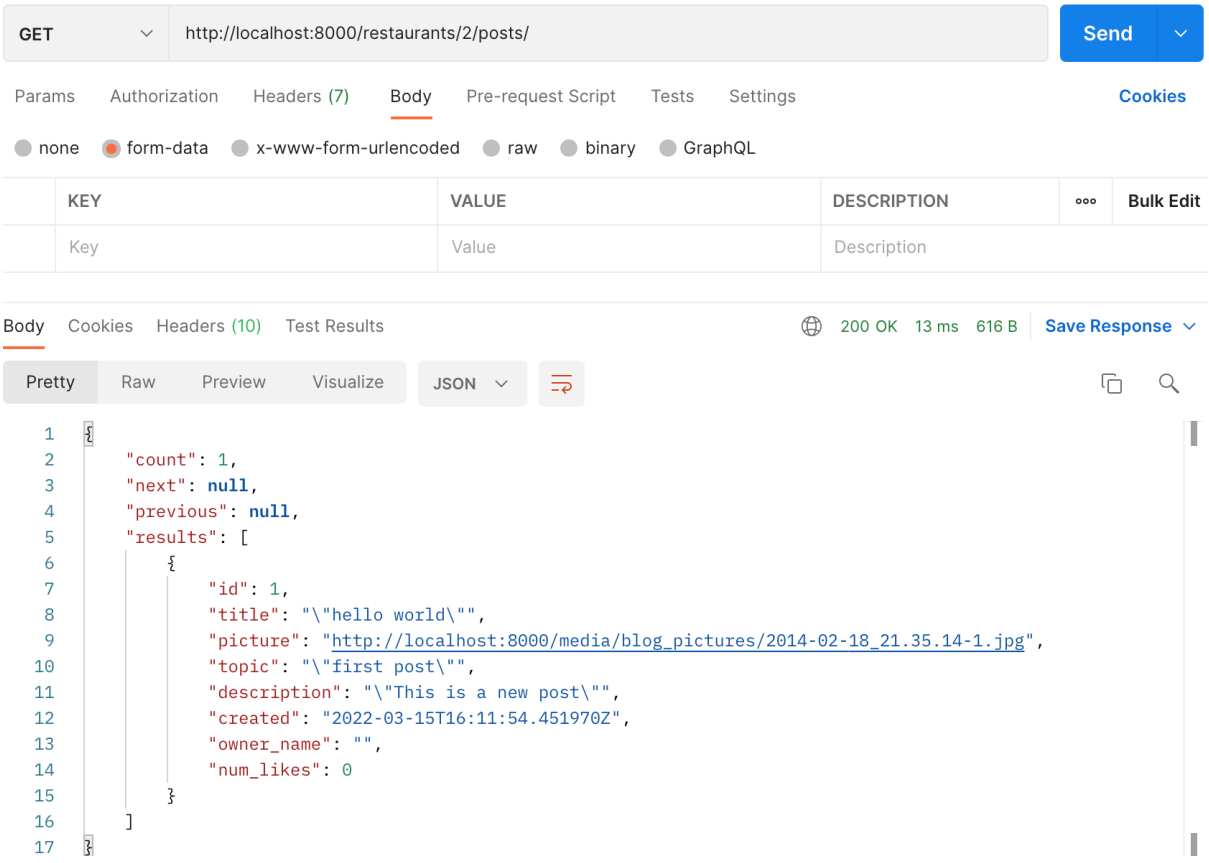
Blog Posts

Endpoints:

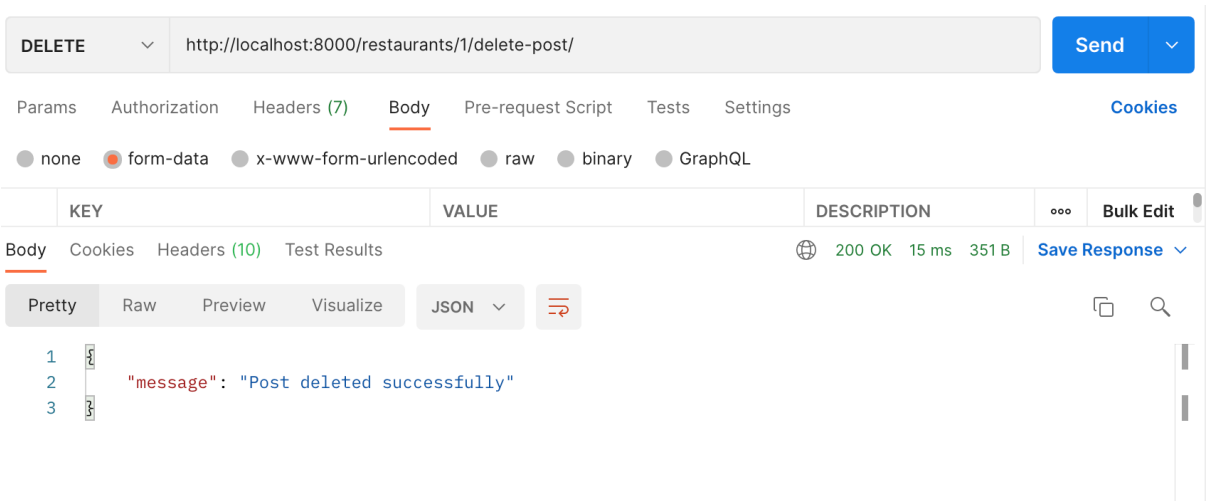
POST	/restaurants/create-blog-post/	Add a restaurant blog post. The author is the restaurant owner who is currently logged in. Request Body Parameters: *title* : string **(required)** *picture* : file *topic*: string **(required)** *description* : string



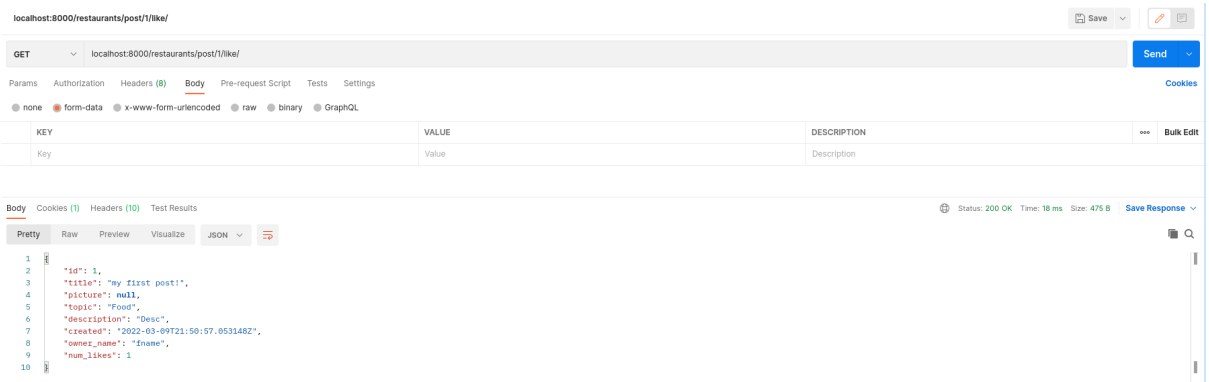
GET	/restaurants/<int:owner_id>/posts/	View blog posts created by a restaurant owner. *owner_id* is a restaurant owner's ID.



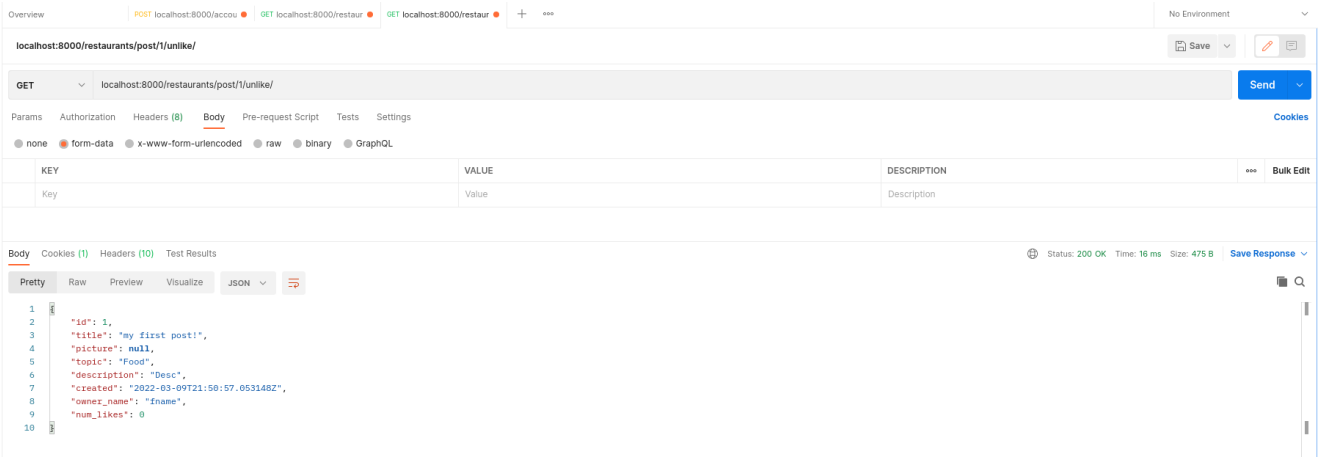
DELETE	/restaurants/<int:pk>/delete-post/	Remove a restaurant blog post. *Pk* is the ID for the post



GET	/restaurants/post/<int:pid>/like/	Likes the post with id *pid* if it exists and returns the serialized post object. Otherwise returns 404.



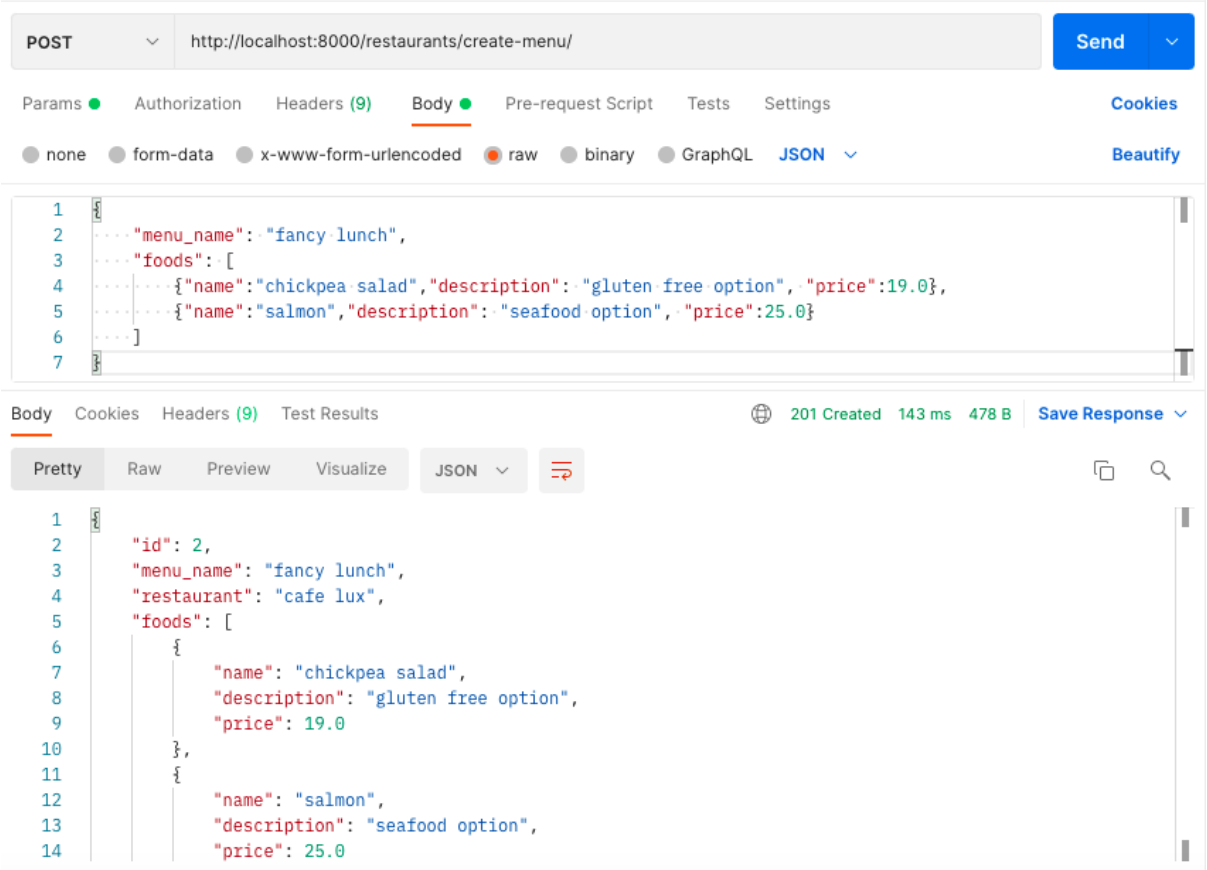
GET	/restaurants/post/<int:pid>/unlike/	Unlikes the post with id *pid* if it exists. Returns the serialized post object or 404 if it doesn't exist.



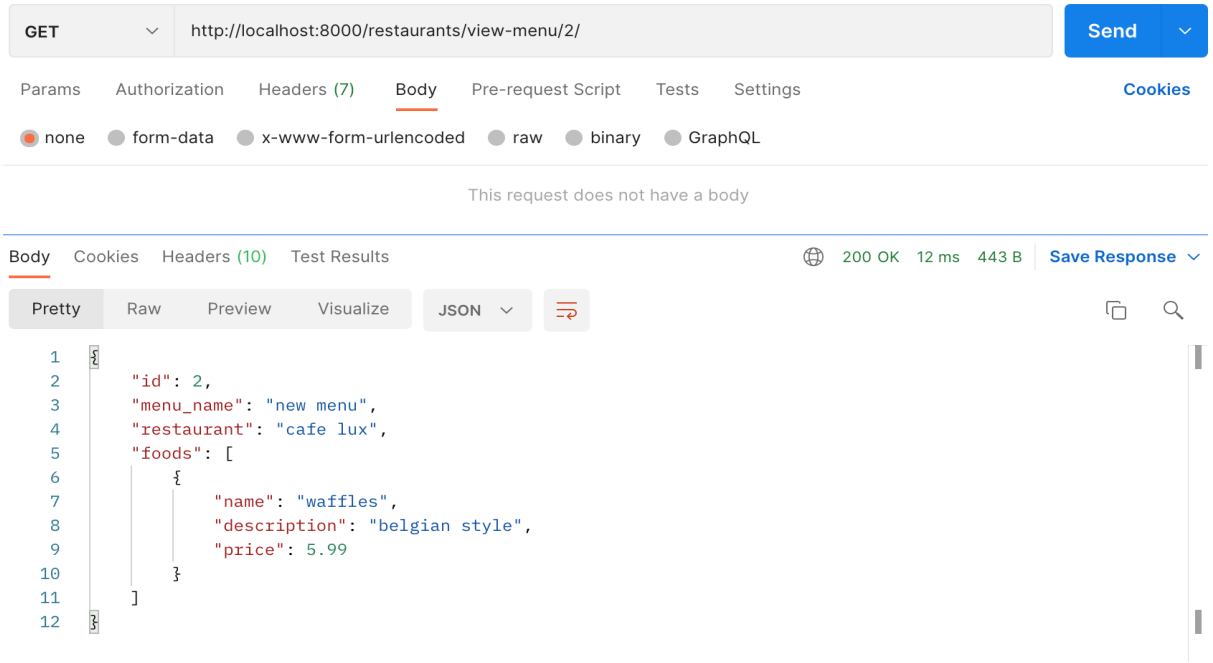
Menu

Endpoints

POST	/restaurants/create-menu/	Create a menu with food items. Request Body Parameters: - *menu_name*: string **required** - *foods*: a list of FoodItems **required**



GET	/restaurants/view-menu/<int:pk>	View menu. *PK* is the menu ID.



DELETE	/restaurants/delete-menu/<int:pk>	Delete a menu by its ID (PK)