Restify

Table of Contents

1. Setup
2. Project Structure
    - Apps
    - Models
3. APIs
    - accounts
    - restaurant
    - blog

Setup

Installation

Requirements: Python3.8

Run the following commands in the P2 directory to setup virtual environment, install all the required packages, make migrations and apply the migrations.

```
chmod +x startup.sh
source startup.sh
```

Run

Requirements: Complete the steps in Installation

Run the following commands in the P2 directory to start the server at port 8000

```
chmod +x run.sh
./run.sh
```

Project Structure

Apps

- accounts

  Stored under the "accounts" directory. Used for all user account related endpoints such as login, register, logout, view profile, edit profile and get notifications. All APIs for this app begin with "/accounts". The app also contains some helper functions in "utils.py". These helper functions are used to create notifications, and/or update the user's followed/liked list of restaurants/blogs. The file "serializer.py" contains all model serializers to help with the below views.
  - Views (All views are inside `accounts/views`)
    - LoginView: Inherits from ObtainAuthToken view so that in the response of the default post endpoint that comes with ObtainAuthToken, we can add a `is_owner` field which will help us later on in P3. The code is in `login.py`
    - RegisterView: Uses CreateApiView since when we register we create a new User object. The code is in `register.py`
    - ProfileView: Uses RetrieveUpdateAPIView since we want to get the user view their profile as well as allow them to update their profile. The code is in `profile.py`
    - LogoutView: Uses a generic APIView since we just need to delete the issued token to user. The code is in `logout.py`
    - NotificationView: Uses a ListAPIView since we want the user to see a paginated list of notifications they have. The code is in `notification.py`

- restaurant

  Stored under the restaurant directory. Used for all restaurant related endpoints such as create restaurant, update restaurant details, add menu items, edit menu items, add pictures, delete pictures, search for restaurants, view details of a restaurant, comment on a restaurant, follow a restaurant and like a restaurant. All APIs for this app begin with `/restaurant`. The file "serializer.py" contains all model serializers to help with the below views.
  - Views (All views are inside `restaurant/views`)
    - CreateRestaurant: Uses CreateApiView to create a new Restaurant object. The code is in `CreateRestaurant.py`
    - UpdateRestaurant: Uses UpdateAPIView to allow an owner to update the general details of a restaurant. The code is in `UpdateRestaurant.py`
    - AddPicture: Uses CreateApiView to create a new Picture object. The code is in `AddPicture.py`
    - DeletePicture: Uses DestroyAPIView since we want to delete a picture object that was added previously. The code is in `DeletePicture.py`
    - AddMenuItem: Uses CreateApiView to create a new Menu object for the given Restaurant object. The code is in `AddMenuItem.py`
    - UpdateMenu: Uses UpdateAPIView to allow an owner to update the general details of a menu item that restaurant has. The code is in `UpdateMenu.py`
    - SearchRestaurants: Uses a ListAPIView since we want the user to see a paginated list of restaurants they are searching for. The code is in `SearchRestaurants.py`

        - RestaurantDetail: Uses a RetrieveAPIView to get the restaurant object whose details the user wants to see. The code is in `RestaurantDetail.py`
        - LikeRestaurant: Uses a generic APIview as all we have to do is update the restaurants's like counter. The code is in `LikeRestaurant.py`
        - FollowRestaurant: Uses a generic APIview as all we have to do is update the restaurants's like counter. The code is in `FollowRestaurant.py`
        - CommentRestaurant: Uses ListCreateAPIView so that we get 2 endpoints: one for getting the list of comments for the given restaurant and one for posting comments on the given restaurant. The code is in `CommentRestaurant.py`

- blog

  Stored under the "blog" directory. Used for all blog related endpoints such as creating, viewing, updating, . All APIs for this app begin with "/blog". The file "serializer.py" contains all model serializers to help with the below views.
    - Views (All views and their code can be found inside the file `blog/views.py`)
        - blogView: Uses RetrieveAPIView since we are retrieving a blog object
        - blogCreateView: Uses CreateAPIView since we want to create a new blog object
        - blogDeleteView: Uses DestroyApiView since we are deleting a blog object
        - blogUpdateView: Uses UpdateAPIView since we want to update the blog object
        - blogListView: Uses a ListAPIView since we want the user to see a paginated list of blogs from restaurants that they follow
        - blogLikeView: Uses a generic APIview as all we have to do is update the blog's like counter

Models

User

User model is used to store all the user data who will be using Restify. Furthermore it is used to implement the auth system for the application. Defined in `accounts/models.py`

- Inherits username, password, first_name, last_name and email from AbstractUser
- phone_number: Phone number for the user
- avatar: An image for the user avatar/profile
- is_owner: Tells whether the user is a restauarant owner or not
- liked_restaurant: Many to many field to define a relation between the user and the restaurants this user has liked
- followed_restaurant: Many to many field to define a relation between the user and the restaurants this user has followed
- liked_blogs: Many to many field to define a relation between the user and the blogs this user has liked

Notification

Notification model is used to store notifications that users get. Each notification corresponds to an event that occurred and belongs to only one user. Defined in `accounts/models.py`

- timestamp: Date time field to record when this notication/event was created.
- message: Message for the notification

- user: The user for which this notification is for. Its foreign key field and maps to the User model

Restaurant

Restaurant model is used to store all the restaurants that are created on the Restify application. Each restaurant object has exactly one owner. Defined in `restaurant/models.py`

- owner: One to One field which refers to the User model in the accounts app. This refers to the owner of a restaurant object
- name: name of the restaurant object
- address: address of the restaurant object
- logo: logo of the restaurant object. This is an image field.
- postal_cdoe: postal code of the restaurant object
- phone_num: phone number of the restaurant object
- num_likes: number of likes this restaurant object has
- num_followers: number of followers this restaurant object has

Menu

Menu model is used to store the menu of each restaurant. Each menu object refers to one food item that is there on that restaurant's menu. Defined in `restaurant/models.py`

- restaurant: Foreign Key field which refers to the Restaurant model in the restaurant app. This refers to the restaurant object to whom this menu object belongs to.
- name: name of the food item
- description: description of the food item
- price: price of the food item

Comment

Comment model is used to store all the comments that are made about a particular restaurant. Defined in `restaurant/models.py`

- restaurant: Foreign Key field which refers to the Restaurant model in the restaurant app. This refers to the restaurant object about whom the comment was made
- username: username of the user who made the comment
- message: contents of the comment
- email: email of the user who made the comment

Picture

Picture model is used to store all the pictures that a restaurant object adds. Defined in `restaurant/models.py`

- restaurant: Foreign Key field which refers to the Restaurant model in the restaurant app. This refers to the restaurant object about who adds these pictures
- image: An ImageField to refer to the image stored in the picture object.

Blog

Blog model is used to store all the blogs that are created on the Restify application. Each blog object belongs to exactly one restaurant. Defined in `blog/models.py`

- created_timestamp: Date time field to record when this blog was created
- modified_timestamp: Date time field to record when this blog was last modified
- title: Title of the blog
- content: Content of the blog
- imgs: The blog's only image
- numLikes: The number of likes the blog has
- restaurant: The restaurant object that the blog belongs to

APIs

accounts

Endpoint: `/accounts/login/`

Method: `POST`

Description

This endpoint used to authenticate a user and obtain a token back. This token is to be passed in Authorization headers for endpoints requiring authentication.

Headers

N/A

Payload

`username`: The username of the user who you want to login as
`password`: The password of the corresponding username

Example

```
{
  "username": "owner",
  "password": "12345678"
}
```

Response

Success response fields:

`token`: The auth token used in Authorization headers
`is_owner`: Boolean specifying whether user you have logged in as has owner priveleges or not
`restaurant`: [OPTIONAL] If `is_owner` is `true` then the id of the owned restaurant is returned

Example

```
{
  "token": "f69555ec0dd10b13fd8b51aea6b057ee49cfa8d2",
  "is_owner": true,
  "restaurant": 1
}
```

Possible error responses example:

`400 Bad Request`

```
{ "username": ["This field is required."] }
```

```
{
  "non_field_errors": ["Unable to log in with provided credentials."]
}
```

Endpoint: `/accounts/logout/`

Method: `GET`

Description

This endpoint used to logout a user by deleting their issued token.

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response example:

```
{
  "message": "logged out"
}
```

Possible error responses example:

`Status: 401 Unauthorized`

```
{
  "detail": "Authentication credentials were not provided."
}
```

Endpoint: `/accounts/register/`

Method: `POST`

Description

This endpoint used to register a new User. The new user created is by default not an owner.

Headers

N/A

Payload

`username`: The username of the user
`password`: The password for the user
`first_name`: [OPTIONAL] First name of the user
`last_name`: [OPTIONAL] Last name of the user
`phone_number`: [OPTIONAL] Phone number of the user
`email`: [OPTIONAL] Email of the user

Example

```
{
  "username": "jack123",
  "password": "12345678",
  "first_name": "Jack",
  "last_name": "Ryan",
  "phone_number": "+16412902013",
  "email": "jack@jack.com"
}
```

Response

Success response fields:

`token`: The auth token used in Authorization headers
`is_owner`: `false`

Example

```
{
  "token": "f69555ec0dd10b13fd8b51aea6b057ee49cfa8d2",
  "is_owner": false
}
```

Possible error responses example:

`400 Bad Request`

```
{
  "username": ["A user with that username already exists."],
  "email": ["Enter a valid email address."],
  "phone_number": ["The phone number entered is not valid."]
}
```

Endpoint: `/accounts/profile/`

Method: `GET`

Description

This endpoint used to get all the user info of the currently logged in user.

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response fields:

`first_name`: First name of the user
`last_name`: Last name of the user
`phone_number`: Phone number of the user `email`: Email of the user
`avatar`: The url to the user's avatar image
`liked_restaurant`: List of restaurant ids liked by the user
`followed_restaurant`: List of restaurant ids followed by the user
`liked_blogs`: List of blog ids liked by the user
`is_owner`: Boolean whether this new user is an owner or not
`restaurant`: If `is_owner` is `true` then the id of the owned restaurant is returned otherwise `null`

Example

```
{
  "email": "jack@jack.com",
  "first_name": "Jack",
  "last_name": "Ryan",
  "phone_number": "+16412902013",
  "is_owner": true,
  "avatar": "http://localhost:8000/accounts/avatars/rainbow-
black_1_kV1SeaG.png",
  "liked_restaurant": [1],
  "followed_restaurant": [1],
  "liked_blogs": [2],
  "restaurant": 1
}
```

Possible error responses example:

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Authentication credentials were not provided."
}
```

Endpoint: `/accounts/profile/`

Method: `PUT`, `PATCH`

Description

This endpoint used to update the current user's info such as name, email, phone, avatar

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`first_name`: First name of the user
`last_name`: Last name of the user
`phone_number`: Phone number of the user `email`: Email of the user
`avatar`: Image for the user avatar. If this field is provided then the content type should be `form-data` for the body

Response

Success response fields:

`first_name`: First name of the user
`last_name`: Last name of the user
`phone_number`: Phone number of the user `email`: Email of the user
`avatar`: The url to the user's avatar image
`liked_restaurant`: List of restaurant ids liked by the user
`followed_restaurant`: List of restaurant ids followed by the user
`liked_blogs`: List of blog ids liked by the user
`is_owner`: Boolean whether this new user is an owner or not
`restaurant`: If `is_owner` is `true` then the id of the owned restaurant is returned otherwise `null`

Example

```
{
  "email": "jack@jack.com",
  "first_name": "Jack",
  "last_name": "Ryan",
  "phone_number": "+16412902013",
  "is_owner": true,
  "avatar": "http://localhost:8000/accounts/avatars/rainbow-
black_1_kV1SeaG.png",
  "liked_restaurant": [1],
  "followed_restaurant": [1],
  "liked_blogs": [2],
  "restaurant": 1
}
```

Possible error responses example:

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Authentication credentials were not provided."
}
```

`400 Bad Request`

```
{
  "email": ["Enter a valid email address."],
  "phone_number": ["The phone number entered is not valid."],
  "avatar": [
    "The submitted data was not a file. Check the encoding type on the
form."
  ]
}
```

Endpoint: `/accounts/notification/`

Method: `GET`

Description

This endpoint used to get all the notifications for the current user. This endpoint is paginated. It returns a max of 20 notifications at a time

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`page`: [OPTIONAL] URL query which gets the page number specified in the value. If the this query is not specified then by default the first page is returned

Example

`http://localhost:8000/accounts/notification/?page=2`

Response

Success response fields:

`count`: Total count of notifications `next`: URL to the next page if it exists i.e. if there is more data otherwise `null` `previous`: URL to the previous page. If it is the first page then `null` `results`: An array of notifications. Each notification has a `timestamp` key and a `message` key. `timestamp` key holds the timestamp of when the activity described in the `messsage` key took place. `message` describes the activity which caused to produce the notification

Example

```
{
  "count": 68,
  "next": "http://localhost:8000/accounts/notification/?page=3",
  "previous": "http://localhost:8000/accounts/notification/",
  "results": [
    {
      "message": "john liked your restaurant!",
      "timestamp": "2022-03-11T04:40:03.082085Z"
    },
    {
      "message": "sam liked your restaurant!",
      "timestamp": "2022-03-11T04:34:36.880009Z"
    },
    {
      "message": "jack liked your restaurant!",
      "timestamp": "2022-03-11T04:21:24.311398Z"
    },
    .
    .
    .
  ]
}
```

Possible error responses example:

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Authentication credentials were not provided."
}
```

restaurant

Endpoint: `/restaurant/create/`

Method: `POST`

Description

This endpoint allows a user to create a restaurant, and thus become an owner for that restaurant

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`name`: The name of the restaurant you want to create
`address`: The address of the restaurant you want to create
`logo`: [OPTIONAL] A file upload for the logo of the restaurant you want to create
`postal_code`: The postal code of the restaurant you want to create
`phone_num`: The phone number of the restaurant you want to create

Example

```
{
  "name": "dominos",
  "address": "3353 Mississauga road",
  "logo": "http://localhost:8000/Screen_Shot_2022-03-
08_at_12.46.34_PM.png",
  "postal_code": "L5L6A2",
  "phone_num": "+14379726888"
}
```

Response

Success response fields:

`id`: The unique identifier of the restaurant
`name`: The name of the newly created restaurant
`address`: The address of the newly created restaurant
`logo`: The logo of the newly created restaurant
`postal_code`: The postal code of the newly created restaurant
`phone_num`: The phone number of the newly created restaurant

Example

```
{
  "id": 1,
  "name": "dominos",
  "address": "3353 Mississauga road",
  "logo": "http://localhost:8000/Screen_Shot_2022-03-
08_at_12.46.34_PM.png",
  "postal_code": "L5L6A2",
  "phone_num": "+14379726888"
}
```


Possible error responses example:

`400 Bad Request`

If all the fields required to create a restaurant are not provided


```
{
  "detail": "HTTPStatus.BAD_REQUEST"
}
```


`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used


```
{
  "detail": "Invalid token."
}
```


`403 Forbidden`

When a user tries to create more than one restaurant


```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

Endpoint: `/restaurant/update/`

Method: `PUT/PATCH`

Description

This endpoint allows an owner to update the basic details of a restaurant
such as name, address, logo, postal code, phone number

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

In case of a PUT request all the below fields must be present,
and in case of a PATCH request atleast one of the fields must be present

`name`: [OPTIONAL] The updated name for the restaurant
`address`: [OPTIONAL] The updated address for the restaurant
`logo`: [OPTIONAL] The updated logo for the restaurant
`postal_code`: [OPTIONAL] The updated postal code for the restaurant
`phone_num`:[OPTIONAL] The updated phone number for the restaurant

Example

```
{
  "name": "dominos restaurant"
}
```

Response

Success response fields:

`id`: The unique identifier of the restaurant
`name`: The name of the given restaurant after the update
`address`: The address of the given restaurant after the update
`logo`: The logo of the given restaurant after the update
`postal_code`: The postal code of the given restaurant after the update
`phone_num`: The phone number of the given restaurant after the update

Example

```
{
  "id": 1,
  "name": "dominos restaurant",
  "address": "3353 Mississauga road",
  "logo": "http://localhost:8000/Screen_Shot_2022-03-
08_at_12.46.34_PM.png",
  "postal_code": "L5L6A2",
```

```
  "phone_num": "+14379726888"
}
```

Possible error responses example:

`400 Bad Request`

If no fields are provided for a PUT request

```
{
  "name": ["This field is required."],
  "address": ["This field is required."],
  "postal_code": ["This field is required."]
}
```

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

`403 Forbidden`

When a normal user who is not an owner of any restaurant
tries to update a restaurant's details

```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

Endpoint: `/restaurant/picture/add/`

Method: `POST`

Description

This endpoint allows an owner to add pictures to their restaurant

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`image`: The image file you want to add to the restaurant

Example

```
{
  "image": "http://localhost:8000/Screen_Shot_2022-03-
08_at_12.46.34_PM.png"
}
```

Response

Success response fields:

`id`: The unique identifier of the restaurant
`image`: The image that you just added to your restaurant

Example

```
{
  "id": 1,
  "image": "http://localhost:8000/Screen_Shot_2022-03-
08_at_12.46.34_PM.png"
}
```

Possible error responses example:

`400 Bad Request`
If no image file is provided

```
{
  "detail": "HTTPStatus.BAD_REQUEST"
}
```

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
```

```
}
```

`403 Forbidden`
When a normal user who is not an owner of any restaurant
tries to add pictures to a restaurant

```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

Endpoint: `/restaurant/picture/<int:picture_id>/delete/`

URL parameters

`<int:picture_id>` refers to the id of the picture
you want to delete. You can get the id of the picture as a response when you add that particular picture to your restaurant.

Method: `DELETE`

Description

This endpoint allows an owner to delete pictures they previously
added for their restaurant

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

N/A
since the required image was deleted so when it
tries to return that image you get a `204 No Content`
thus showing that the image was successfuly deleted

Possible error responses example:

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

`403 Forbidden`

When a normal user who is not an owner of any restaurant
tries to delete a restaurant's pictures

```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

`404 Not Found`

When you try to delete a picture which doesn't exist for that restaurant

```
{
  "detail": "HTTPStatus.NOT_FOUND"
}
```

Endpoint: `/restaurant/menu/add/`

Method: `POST`

Description

This endpoint allows an owner to add items to their restaurant's menu

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`name`: The name of the item to be added to the restaurant's menu
`description`: The description of the item to be added to the restaurant's menu
`price`: The price of the item to be added to the restaurant's menu

Example

```
{
  "name": "pizza",
  "description": "food",
  "price": "10"
}
```

Response

Success response fields:

`id`: The unique identifier of the restaurant
`name`: The name of the item added to the restaurant's menu
`description`: The description of the item added to the restaurant's menu
`price`: The price of the item added to the restaurant's menu

Example

```
{
  "id": 1,
  "name": "pizza",
  "description": "food",
  "price": "10"
}
```

Possible error responses example:

`400 Bad Request`
If no fields are provided to add the menu item

```
{
    "name": [
        "This field is required."
    ],
    "description": [
        "This field is required."
    ],
    "price": [
        "This field is required."
    ]
}
```

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

`403 Forbidden`

When a normal user who is not an owner of any restaurant
tries to add menu items to a restaurant

```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

Endpoint: `/restaurant/menu/<int:menu_id>/update/`

URL parameters

`<int:menu_id>` refers to the id of the menu item
you want to update. You can get the id of a menu item as a response when you add that menu item to the restaurant

Method: `PUT/PATCH`

Description

This endpoint allows an owner to update the details of the items in their
restaurant's menu such as the menu items' name, description, price

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

In case of a PUT request all the below fields must be present,
and in case of a PATCH request atleast one of the fields must be present

`name`: [OPTIONAL] The name of the menu item you want to update
`description`: [OPTIONAL] The description of the menu item you want to update
`price`: [OPTIONAL] The price of the menu item you want to update

Example

```
{
  "name": "pasta"
}
```

Response

Success response fields:

`id`: The unique identifier of the restaurant
`name`: The name of the menu item after the update
`description`: The description of the menu item after the update
`price`: The price of the menu item after the update

Example

```
{
  "id": 1,
  "name": "pasta",
  "description": "food",
  "price": "10"
}
```

Possible error responses example:

`400 Bad Request`
If no fields are provided for a PUT request

```
{
  "name": ["This field is required."],
  "description": ["This field is required."]
}
```

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
```

```
}
```

`403 Forbidden`

When a normal user who is not an owner of any restaurant
tries to update a menu item

```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

`404 Not Found`

When you try to update a menu item which doesn't exist

```
{
  "detail": "HTTPStatus.NOT_FOUND"
}
```

Endpoint: `/restaurant/search/`

Method: `GET`

Description

This endpoint allows a user to search for restaurants based on
the restaurant name, address, or the name of the food items it sells

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Query Parameters

`search`: [OPTIONAL] Get all restaurants that have either name, address or menu item that begins with value in the search query

Example

```
{
  "address": "mississauga"
}
```

Response

Success response fields:

`count`: The number of restaurants returned as a result of search
`next`: A link to the next page of results
`previous`: A link to the previous page of results
`results`: A list of restaurants which are returned as a result of search, each restaurant's details include their id, name, address, logo, postal code, and phone number.
The restaurants in this list are arranged in the order of their popularity so the restaurant with the most number of likes is on the top.

Example


```
{
  "count": 3,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 1,
      "name": "Starbucks",
      "address": "mississauga",
      "logo": "http://localhost:8000/restaurant/restaurant-
logo/Screen_Shot_2022-03-08_at_12.46.34_PM_3AcGgMc.png",
      "postal_code": "123456",
      "phone_num": "+14379726884"
    },
    {
      "id": 2,
      "name": "dominos",
      "address": "mississauga",
      "logo": "http://localhost:8000/restaurant/restaurant-
logo/Screen_Shot_2022-03-08_at_12.46.34_PM.png",
      "postal_code": "123456",
      "phone_num": "+14379726884"
    },
    {
      "id": 3,
      "name": "pizza hut",
      "address": "mississauga",
      "logo": "http://localhost:8000/restaurant/restaurant-
logo/Screen_Shot_2022-03-09_at_11.38.26_AM.png",
```

```
        "postal_code": "123456",
        "phone_num": "+14379726884"
      }
    ]
}
```

Possible error responses example:

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

Endpoint: `/restaurant/search/<int:restaurant_id>/`

URL parameters

`<int:restaurant_id>` refers to the id of the restaurant
you want to see the details of. You get this id as a response
when a restaurant is created, or if you try to search for this restaurant.
This id is also displayed as a response when an owner logs in.

Method: `GET`

Description

This endpoint allows a user to view the details of a restaurant,
which includes its name, address logo, postal code, phone number,
number of likes, number of followers, menu items, pictures, blogs, and comments

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response fields:

`id`: The unique identifier of the restaurant
`name`: The name of the restaurant
`address`: The address of the restaurant
`logo`: An image file for the restaurant's logo
`postal_code`: The postal code of the restaurant
`phone_num`: The phone number of the restaurant
`num_likes`: The number of likes for this restaurant
`num_followers`: The number of followers for this restaurant
`menu`: A list of menu items for this restaurant. Each menu item's details includes its name, description, and price
`pictures`: A list of pictures that this restaurant has
`blog`: A list of blogs that this restaurant has posted. Each blog's details include its timestamp when it was created, the tiemstamp when it was modified, title, content, images, and the number of likes
`comments`: A list of comments that this restaurant has

Example

```
{
  "id": 1,
  "name": "dominos",
  "address": "mississauga",
  "logo": "http://localhost:8000/restaurant/restaurant-
logo/Screen_Shot_2022-03-08_at_12.46.34_PM.png",
  "postal_code": "123456",
  "phone_num": "+14379726884",
  "num_likes": 2,
  "num_followers": 0,
  "pictures": [
    {
      "image": "http://localhost:8000/restaurant/restaurant-
pictures/Screen_Shot_2022-01-21_at_10.46.29_PM.png"
    }
  ],
}
```

Possible error responses example:

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
```

```
}
```

`404 Not Found`
When you try to get the details of a restaurant which doesn't exist

```
{
  "detail": "HTTPStatus.NOT_FOUND"
}
```

Endpoint: `/restaurant/<int:restaurant_id>/menu/list/`

URL parameters

`<int:restaurant_id>` refers to the id of the restaurant
you want to see the menu items of. You get this id as a response
when a restaurant is created, or if you try to search for this restaurant.
This id is also displayed as a response when an owner logs in.

Method: `GET`

Description

This endpoint allows a user to see the list of menu items
for a given restaurant

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response fields:

`count`: The number of menu items the given restaurant has
`next`: A link to the next page of results
`previous`: A link to the previous page of results

`results`: A list of menu items the given restaurant has. For each menu item the details displayed include name, description and price.

Example

```
{
  "count": 1,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 1,
      "name": "pizza",
      "description": "food",
      "price": 1
    }
  ]
}
```

Possible error responses example:

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

`404 Not Found`
When you try to get the menu items of a restaurant which doesn't exist

```
{
  "detail": "HTTPStatus.NOT_FOUND"
}
```

Endpoint: `/restaurant/<int:restaurant_id>/comment/`

URL parameters

`<int:restaurant_id>` refers to the id of the restaurant
you want to see the comments of. You get this id as a response
when a restaurant is created, or if you try to search for this restaurant.
This id is also displayed as a response when an owner logs in.

Method: `GET`

Description

This endpoint allows a user to see the list of comments
for a given restaurant

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response fields:

`count`: The number of menu items the given restaurant has
`next`: A link to the next page of results
`previous`: A link to the previous page of results
`results`: A list of commments the given restaurant has. For each comment
we can see its contents.

Example

```
{
  "count": 1,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 11,
      "message": "hi",
      "username": "john",
      "email": "sam@sam.com",
      "timestamp": "2022-03-15T23:46:25.154628Z"
    }
  ]
}
```

Possible error responses example:

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

`404 Not Found`

When you try to get the comments of a restaurant which doesn't exist

```
{
  "detail": "Not found."
}
```

Endpoint: `/restaurant/<restaurant_id>/like/`

Method: `PUT`

Description

This endpoint used to like a restaurant object

URL Parameter

`restaurant_id`: Unique ID of the restaurant to be liked

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`like`: 1 to like to blog, 0 to unlike the blog

Example

```
{
  "like": 1
```

```
}
```

Response

Returns 202 if the user's like/unlike is successful. Returns 200 if user tries to like when they have already liked or unlikes when they haven't liked

Success response fields:

`num_likes`: The number of likes of the restaurant that was liked

Example

```
{
  "numLikes": 1
}
```

Possible error responses example:

`404 Not Found`

```
{
  "detail": "Not found."
}
```

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```

`400 Bad Request`

```
{
  "like": ["This field is required."]
```

```
}
```

Endpoint: `/restaurant/<restaurant_id>/follow/`

Method: `PUT`

Description

This endpoint used to follow a restaurant object

URL Parameter

`restaurant_id`: Unique ID of the restaurant to be followed

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`like`: 1 to follow to blog, 0 to unfollow the blog

Example

```
{
  "follow": 1
}
```

Response

Returns 202 if the user's follow/unfollow is successful. Returns 200 if user follows when they are already following or unfollows when they aren't following

Success response fields:

`num_followers`: The number of followers of the restaurant that was followed

Example

```

```

```
{
  "num_followers": 1
}
```

Possible error responses example:

404 Not Found

```
{
  "detail": "Not found."
}
```

401 Unauthorized

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```

---

400 Bad Request

```
{
    "follow": [
        "Should be either 1 or 0"
    ]
}
```

Endpoint: /restaurant/<restaurant_id>/comment/

Method: POST

Description

This endpoint used to comment on a restaurant object

URL Parameter

`restaurant_id`: Unique ID of the restaurant to be commented on

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`message`: the comment's message

Example

```
{
    "message": "asdf",
    "username": "1",
    "email": "",
    "timestamp": "2022-03-16T02:55:01.422953Z"
}
```

Response

Success response fields:

`message`: The message of the comment created `username`: The username of the user who posted the comment `email`: The email of the user who posted the comment `timestamp`: The timestamp of when this comment was posted

Example

```
{
  "message": "hi",
  "username": "john",
  "email": "sam@sam.com",
  "timestamp": "2022-03-16T00:44:56.277955Z"
}
```

Possible error responses example:

`404 Not Found` When you try to comment on a blog that does not exist

```
{
  "detail": "Not found."
}
```

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Authentication credentials were not provided."
}
```

`400 Bad Request`

```
{
  "message": ["This field is required."]
}
```

Endpoint: `/restaurant/<restaurant_id>/blogs/`

Method: `GET`

Description

This endpoint used to get all blogs posted by the restaurant

URL Parameter

`restaurant_id`: Unique ID of the restaurant that the blogs will be from

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response fields:

`count`: Number of blogs posted by restaurant the user is following `next`: URL to the next page if it exists i.e. if there is more data otherwise `null`
`previous`: URL to the previous page. If it is the first page then `null`
`results`: An array of blogs. Each blog has a `id` key, `created_timestamp` key, a `modified_timestamp` key, a `title` key, a `content` key, a `imgs` key, and a `numlikes` key. These keys refer to the fields of the same name in the blogs model.

Example


```
{
  "count": 2,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 1,
      "created_timestamp": "2022-03-15T02:05:14.394406Z",
      "modified_timestamp": "2022-03-15T02:05:14.394470Z",
      "title": "Title 2",
      "content": "Ipsum Lorem 1",
      "imgs": null,
      "numLikes": 0
    },
    {
      "id": 2,
      "created_timestamp": "2022-03-15T02:11:20.498380Z",
      "modified_timestamp": "2022-03-15T04:02:39.051571Z",
      "title": "Title 2",
      "content": "Ipsum Lorem 2",
      "imgs": "http://127.0.0.1:8000/blog/blog-logo/1234_r3D3r4p.PNG",
      "numLikes": 1
    }
  ]
}
```


Possible error responses example:

`404 Not Found` When you try to get the blogs of a restaurant which doesn't exist


```
{
  "detail": "Not found."
}
```

`401 Unauthorized`

If a non-logged in user tries to access the endpoint

or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```

`400 Bad Request`

```
{
  "message": ["This field is required."]
}
```

---

blog

Endpoint: `/blog/create/`

Method: `POST`

Description

This endpoint used to create a blog object as a restaurant owner

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`title`: The title of the blog

`content`: The content of the blog

`imgs`[OPTIONAL]: Image file that will be the blog's image

Example

```
{
  "title": "owner",
  "content": "12345678",
  "imgs": "http://localhost:8000/accounts/avatars/rainbow-
black_1_kV1SeaG.png"
}
```

Response

Success response fields:

`id`: the unique identification number of the blog
`created_timestamp`: The time of when the blog was created
`modified_timestamp`: The time of when the blog was last modified (should be the same as `created_timestamp`)
`title`: The title of the blog created
`content`: The content of the blog created
`imgs`: The image of the blog created
`numLikes`: The number of likes the created blog has (always zero)

Example

```
{
  "id": 1,
  "created_timestamp": "2022-03-15T02:11:20.498380Z",
  "modified_timestamp": "2022-03-15T02:11:20.498434Z",
  "title": "Insert Blog Name Here",
  "content": "Ipsum Lorem",
  "imgs": "http://localhost:8000/accounts/avatars/rainbow-
black_1_kV1SeaG.png",
  "numLikes": 0
}
```

Possible error responses example:

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Authentication credentials were not provided."
}
```

`400 Bad Request` If the field(s) are missing

```
{
  "detail": ["title is missing", "content is missing"]
}
```

Endpoint: `/blog/<blog_id>/`

Method: `GET`

Description

This endpoint used to get a blog object

URL Parameter

`blog_id`: Unique ID of the blog to be viewed

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response fields:

`id`: the unique identification number of the blog
`created_timestamp`: The time of when the blog was created
`modified_timestamp`: The time of when the blog was last modified
`title`: The title of the blog
`content`: The content of the blog
`imgs`: The image of the blog
`numLikes`: The number of likes the blog has

Example

```
{
  "id": 1,
  "created_timestamp": "2022-03-15T02:11:20.498380Z",
  "modified_timestamp": "2022-03-15T02:11:20.498434Z",
  "title": "Insert Blog Name Here",
  "content": "Ipsum Lorem",
  "imgs": "http://localhost:8000/accounts/avatars/rainbow-
black_1_kV1SeaG.png",
  "numLikes": 0
}
```


Possible error responses example:

`404 Not Found` If the blog with specified id does not exist


```
{
  "detail": "Not found."
}
```


`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used


```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```

Endpoint: `/blog/feed/`

Method: `GET`

Description

This endpoint used to get the recommended feed, a list of blogs from restaurants the user follows, for the user

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Success response fields:

`count`: Number of blogs posted by restaurant the user is following `next`: URL to the next page if it exists i.e. if there is more data otherwise `null`
`previous`: URL to the previous page. If it is the first page then `null`
`results`: An array of blogs from restaurants that the user follows sorted by newest first. Each blog has a `id` key, `created_timestamp` key, a `modified_timestamp` key, a `title` key, a `content` key, a `imgs` key, and a `numlikes` key and a `id` key. These keys refer to the fields of the same name in the blogs model.

Example


```
{
  "count": 2,
  "next": null,
  "previous": null,
  "results": [
    {
      "id": 1,
      "created_timestamp": "2022-03-15T02:05:14.394406Z",
      "modified_timestamp": "2022-03-15T02:05:14.394470Z",
      "title": "Title 2",
      "content": "Ipsum Lorem 1",
      "imgs": null,
      "numLikes": 0
    },
    {
      "id": 2,
      "created_timestamp": "2022-03-15T02:11:20.498380Z",
      "modified_timestamp": "2022-03-15T04:02:39.051571Z",
      "title": "Title 2",
      "content": "Ipsum Lorem 2",
      "imgs": "http://127.0.0.1:8000/blog/blog-logo/1234_r3D3r4p.PNG",
      "numLikes": 1
    }
  ]
}
```


Possible error responses example:

`401 Unauthorized`

If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```

Endpoint: `/blog/<blog_id>/update/`

Method: `PUT`

Description

This endpoint used to update a blog object

URL Parameter

`blog_id`: Unique ID of the blog to be updated

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`title`: The new title of the blog
`content`: The new content of the blog
`imgs`: [OPTIONAL] The new image of the blog

Example

```
{
  "title": "Insert New Blog Name Here",
  "content": "Ipsum Lorem",
```

```
  "imgs": "http://localhost:8000/accounts/avatars/rainbow-
black_1_kV1SeaG.png"
}
```

Response

Success response fields:

`id`: the unique identification number of the blog
`created_timestamp`: The time of when the blog was updated
`modified_timestamp`: The time of when the blog was updated
`title`: The title of the updated blog
`content`: The content of the updated blog
`imgs`: The image of the updated blog
`numLikes`: The number of likes the updated blog has

Example

```
{
  "id": 1,
  "created_timestamp": "2022-03-15T02:11:20.498380Z",
  "modified_timestamp": "2022-03-15T02:11:20.498434Z",
  "title": "Insert Blog Name Here",
  "content": "Ipsum Lorem",
  "imgs": "http://localhost:8000/accounts/avatars/rainbow-
black_1_kV1SeaG.png",
  "numLikes": 0
}
```

Possible error responses example:

`404 Not Found` If the blog with specified id does not exist

```
{
  "detail": "Not found."
}
```

`403 Forbidden` If the user tries to update a blog which they do not own

```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

`401 Unauthorized`
If a non-logged in user tries to access the endpoint

or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```

`400 Bad Request` If the content field is missing

```
{
  "content": ["This field is required."]
}
```

Endpoint: `/blog/<blog_id>/delete/`

Method: `DELETE`

Description

This endpoint used to delete a blog object

URL Parameter

`blog_id`: Unique ID of the blog to be deleted

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

N/A

Response

Returns status `204` on Success

Success response fields:

N/A

Possible error responses example:

`404 Not Found` If the blog with specified id does not exist

```
{
  "detail": "Not found."
}
```

`401 Unauthorized`
If a non-logged in user tries to access the endpoint
or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```

`403 Forbidden` If the user tries to delete a blog which they do not own

```
{
  "detail": "HTTPStatus.FORBIDDEN"
}
```

Endpoint: `/blog/<blog_id>/like/`

Method: `PUT`

Description

This endpoint used for a user to like a blog object

URL Parameter

`blog_id`: Unique ID of the blog to be liked

Headers

`Authorization: Token {TOKEN_OBTAINED_FROM_LOGIN}`

Example

```
Authorization: Token 55d8de2d907664eb4dc4c7932a8c0f6ae41d3f50
```

Payload

`like`: 1 to like to blog, 0 to unlike the blog

Example

```
{
  "like": 1
}
```

Response

Returns 202 if the user's like/unlike is successful. Returns 200 if user tries to like when they have already liked or unlikes when they haven't liked

Success response fields:

`numLikes`: The number of likes the liked/unliked blog has

Example

```
{
  "numLikes": 0
}`404 Not Found`
```

Possible error responses example:

`404 Not Found` If the blog with specified id does not exist

```
{
  "detail": "Not found."
}
```

`401 Unauthorized`
If a non-logged in user tries to access the endpoint

or if an invalid token was used

```
{
  "detail": "Invalid token."
}
```

```
{
  "detail": "Authentication credentials were not provided."
}
```
