BLOG API

A
Mini Project Report

Submitted in partial fulfilment of the
Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

By

P Lalith Srinivas, 1602-19-733-021

B Shree Vaishnavi Reddy, 1602-19-733-047

Department of Computer Science & Engineering

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Ibrahimbagh, Hyderabad-31

2021

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Computer Science & Engineering

DECLARATION BY THE CANDIDATE

I, **B Shree Vaishnavi Reddy,** bearing hall ticket number, **1602-19-733-047**, hereby declare that the project report entitled **"BLOG API"** Department of Computer Science & Engineering, VCE, Hyderabad, is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering**.

This is a record of bonafide work carried out by me and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

B Shree Vaishnavi Reddy,

1602-19-733-047.

Vasavi College of Engineering (Autonomous)

(Affiliated to Osmania University)

Hyderabad-500 031

Department of Computer Science & Engineering

BONAFIDE CERTIFICATE

This is to certify that the project entitled **"BLOG API"** being submitted by **B Shree Vaishnavi Reddy,** bearing the roll number 1602-19-733-047**,** in partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science & Engineering is a record of bonafide work carried out by him/her under my guidance.

Dr. T. Adilakshmi,

Professor & HOD,

Dept. of CSE,

ACKNOWLEDGEMENT

We take this opportunity with pride and enormous gratitude, to express the deeply embedded feeling and gratefulness to our respectable guide R. Sateesh Kumar, Department of Computer Science and Engineering, whose guidance was unforgettable and innovative ideas as well as his constructive suggestions has made the presentation of our thesis a grand success.

We are thankful to Dr. T. Adhilakshmi, Head of Department (CSE), Vasavi College of Engineering for their help during our course work.

Finally, we would like to express our heart full thanks to the management and support of our college, Vasavi College of Engineering for providing the necessary arrangements and support to complete our seminar work successively.

ABSTRACT

Blog API project is a completely back-end based project which provides APIs from the backend server which can consumed from the frontend server and customize the UI and UX of the blog platform.

Blogging is one of the most powerful tools to spread information and knowledge, but the user experience also creates a major impact on the way the content on the platform is consumed. But the UI preference options are not universal, so Blog API project provides the backend API service for the individual or team bloggers to customize their user interface based on the theme and culture of their group or individuals.

Blog API project is coded in Django using Django REST Framework (DRF) and HTML using the SQLite3 database engine. This project has the implementation of basic features that most blog platforms have, like adding comments to the posts, upvoting the posts and we have the feature of CRUD (Create, Retrieve, Update and Delete) on all the posts, comments and votes. Our Blog API project also has the Google Authentication API integrated for user login and sign-up into the blogging platform.

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Overview

Sharing Information is one of the powerful activities in our day to day lives which effect our thoughts and actions. After digitalization of almost everything, blogging has become one of the best tools available on the internet to share our thoughts, ideas and opinions and develop a network of good connections. Blogging tends to provide more exposure and helps in improving business by increasing reach. So, this Blog API project is an attempt to replicate the existing blog platforms with few added features enhancing the user connection and experience.

1.2 Motivation

The main motivation in making this project is when I tried to start blogging my travel experiences, most of the platforms had a fixed template and were not customizable according to theme of my content and style of blogs. I thought that, being CSE student, I can customize the UI experience I want to give the viewers so I thought of developing a backend API that can also help other students and groups to maintain their blogging sites with their own themes and UI, making it the main purpose and motivation for us doing this project as Mini Project in Python.

1.3 Problem Definition

The main problem we are trying to solve through this project is the unavailability of blog APIs to customize the themes and UI of the blogging sites based on the content being shared on the site and to the content creator's preference. There are very limited choices for customizable pages leaving content creators to restrict their creativity visually. So, we made an API service provider for the content creators.

1.4 Objectives

The main objective in doing this project is to get more exposure towards server-side framework which we chose to be Django and DjangoRestFramework. Other objectives include learning and getting more knowledge on designing the schemas and database relations in developing the project.

We also worked with social authentication providers like google and google developer console learning more deeply about the tokens and authorizations. Over all developing more insight and knowledge towards REST API building and working with web apps.

1.5 Scope

We can see a very good scope for this project which will develop and promote the habit of blogging, developing the blogging community. This project can also be refined of its features and functionalities, through which can find its implementation in real life in college or office workspace to share our experiences and learnings, encouraging peer learning and building a sense of community to discuss content and multiply knowledge. This can also find its application in libraries and reading clubs where readers can blog about the books, among travel lovers forming a travel community and sharing experiences through blogs like bikers' gang, geeks group and so on. Blogging is a very powerful tool to connect the people around the world, scope of this project is huge. By making few changes, this project can be used widely by many users.

2. OVERVIEW OF PROPOSED DESIGN

In Blog API project, what we have proposed was to prepare an API service in a way such that individuals or a group can use these APIs to build websites for their team or it can be used by individual bloggers deciding on the theme and style of their personal blogging site. Basic features such as the creating, deleting, retrieving, and updating the content can be done only by the admin or the original user of the post. We have also enabled features such as the commenting and upvoting on the posts which brings traffic to the site. Also, the feature of saving the posts without publishing them (as drafts) make it even better for the team to synchronize and make their upcoming posts ready.

The special feature we are planning to add to this blog platform is integrating it with the GOOGLE Authentication API which can make it very easy for the visiting users as well as the admins to access the site instead of filling forms at the time of signup.

We are also thinking of adding the search bar for filtering the posts on the dashboard according to few tag and key aspects of the articles being published on the platform.

3. SYSTEM DESIGN

A major part of system design in the project was to design the schemas for the database models which we can query during the data retrieval process.

There are 5 schemas designed for the models in this project.

1. Post

2. Users

3. Comments

4. Post votes

5. Category

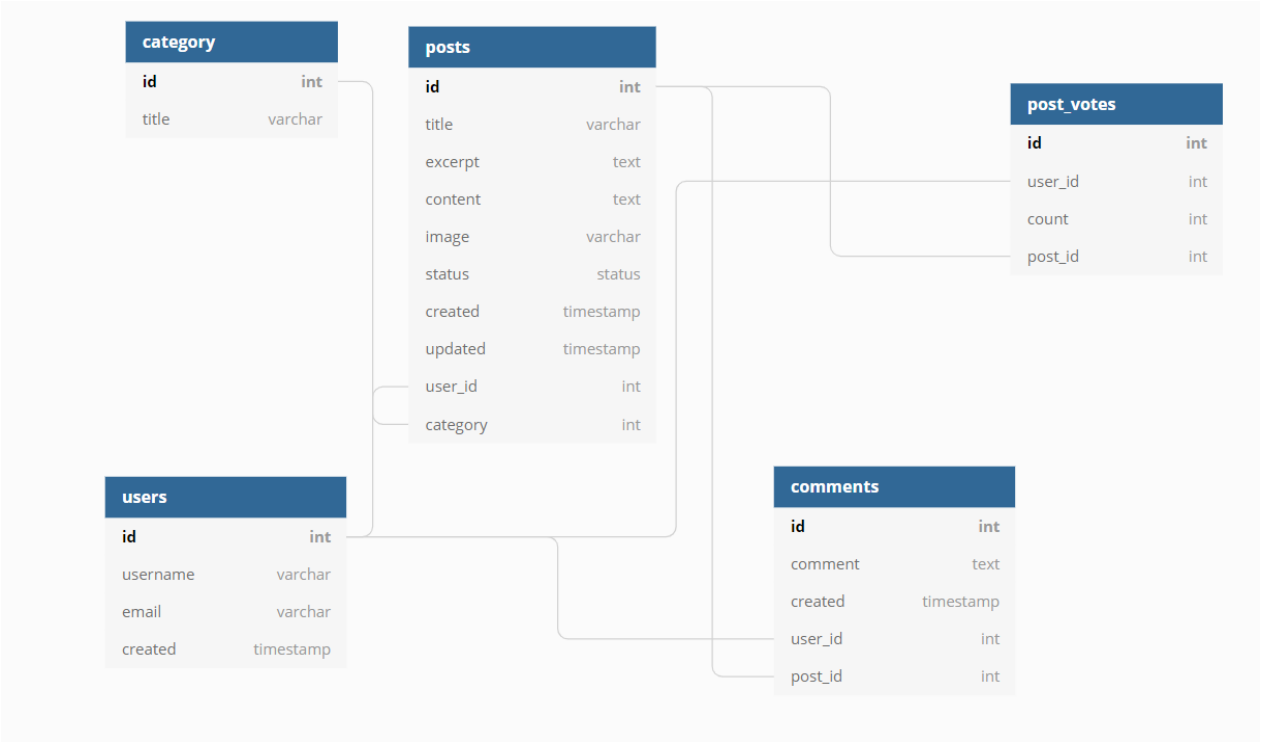

Fig 3.1

4. ANALYSIS AND UNDERSTANDING OF IMPLEMENTATION

This is the complete overview of the project directories with a tree view of the project folder.

Fig. 4.1

4.1 EXPLANATION

db.sqlite3

This is the database file where all the data that is generated will be stored. It is a local file treating PC as the host, where the server can be controlled from terminal or command prompt.

manage.py

This file is a command-line utility that lets the programmer to interact with the project.

Pipfile & Pipfile.lock

This file has the data about the version of python, and other packages that have been installed and used in the project along with the versions used in this specific project. Some of them include djangorestframework, django, django-allauth.

4.2 base (Project directory)

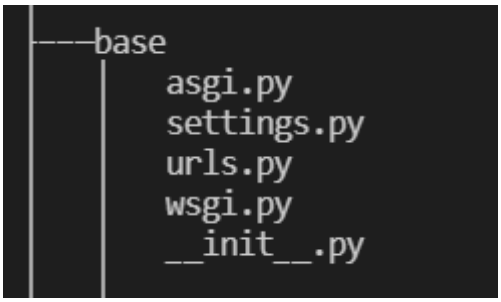

Fig.4.2

asgi.py , wsgi.py and __init__.py are the default files included in any django project representing the server gateway interfaces.

settings.py

This is the file which contains information such as the installed apps, middleware , template directories, database connection, social account providers.

urls.py

urls.py is the file that shows all the routes that can be accessed in the project from the base directories. These urls patterns also contain the routes to connect to the apps that will be included in the project and each app will again have its own urls specific to the routes avalabile in that app.

```
urlpatterns = [
    path('', TemplateView.as_view(template_name='blog/index.html')),
    path('admin/', admin.site.urls),
    path('api/', include('blog.urls'), name='blog'),
    path('api-auth/', include('rest_framework.urls', namespace='rest_framework')),
    path('accounts/', include('allauth.urls')),

]
```

Fig. 4.3

The paths need to be added at the end of the url of the site. Considering the example of localhost http://127.0.0.1.8000/admin takes us to the admin login and admin access where the users who is a superadmin or has the staff access to the project can access this route and login to the admin administration to make changes in the data.

http://127.0.0.1.8000/api is the connecting route to the app which is called “blog” to include it’s base route in main project urls.

http://127.0.0.1.8000/api-auth and http://127.0.0.1.8000/accounts are related to the login and authentication of the user via google social account provider.

4.3 blog (App)

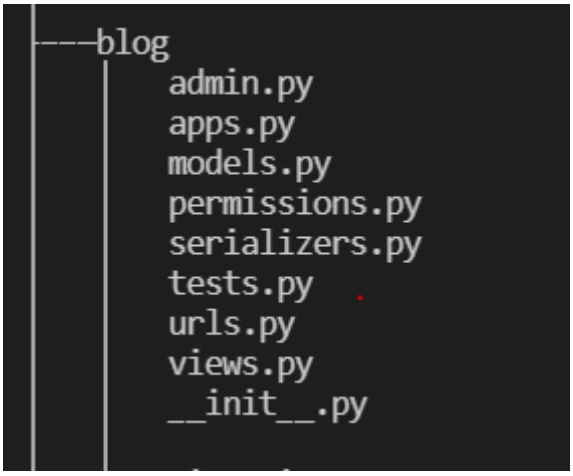

Fig. 4.4

"blog" is the web app that is created to add to the project "base" and add the functionalities and features and run it as a project with the web app integrated in it.

admin.py

This is a command line utility for administrative tasks which is automatically created by Django on each Django project. In this file we add the database models which we have used in the project to add them to the admin access site making the data and attributes manipulation easy for the superadmin and the staff access users.

apps.py

apps.py is also a default file which is created by Django. It is the configuaration where the database models are connected to the app. "models.py" is the file where the database models are located.

models.py

models.py is the file where all the database mdoels used in the app are located. These models are based on the schema that was designed for the entities and the attributes respectively.

Apart from the default user model in django we have four database models designed for the app.

Category :- It is used to store the data of the category of the post.

Post :- It has attributes related to the details of the posts that are to be given from the API.

Post models has two Foreign Key connections with Category model and User model.

Category model and Post are linked by “Many to One” relation and User and Post are linked by “Many to One” relation.

Vote :- Has two attributes of user and post linked through Foreign Key. Both the User and the Post models are linked through “Many to Many” realtion.

Comment :- Apart from the basic attributes that are required for post this model and “Many to Many’ realtion with the Post and the User models

Example of the posts model code used in our “blog” app:

```
class Post(models.Model):

    class PostObjects(models.Manager):
        def get_queryset(self):
            return super().get_queryset() .filter(status='published')

    options = (
        ('draft', 'Draft'),
        ('published', 'Published'),
    )

    category = models.ForeignKey(
        Category, on_delete=models.PROTECT, default=1)
    title = models.CharField(max_length=255)
    excerpt = models.TextField(null=True)
    content = models.TextField(null=False)
    image_url = models.URLField()
    created = models.DateTimeField(auto_now=True)
    updated = models.DateTimeField(auto_now_add=True)
    status = models.CharField(
        max_length=10, choices=options, default='published')
    published = models.DateTimeField(default=timezone.now)
    author = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name='blog_posts')
    objects = models.Manager()
    postobjects = PostObjects()

    class Meta:

        ordering = ('-published',)

    def __str__(self):
        return self.title
```

Fig. 4.5

permissions.py

This file is completely dedicated to the custom permission that need to be used in the app to restrict the API signal access based on the user permissions.

```
class PostUserWritePermission(BasePermission):
    message = 'Editing posts is restricted to the
    author only.'

    def has_object_permission(self, request, view,
    obj):

        if request.method in SAFE_METHODS:
            return True

        return obj.author == request.user
```

Fig. 4.6

The permission class “PostUserWrittenPermission” is a custom permission written on the “BasePermission”. The overview of this permission is to restrict the editing and updating access of the posts published by the author only. So that the complete access to the posts lies with the author himself and the superadmin. If the specific user is not the author that account will have only the “GET” request enabled which is enabling only the view option of the post

“SAFE_METHODS” here are the request methods which restrict the POST, PUT and DELETE access to the non-author user of the post.

serializers.py

The main work of the serlializers.py is to convert the following HTML accepted data which is non-object format to the JSON format in the structure of objects which can be used to send data as APIs at the time of request methods accordingly.

So, we pass the data from the models to the serializers which we want to send during API call.

```
class PostSerializer(serializers.ModelSerializer):
    author = serializers.ReadOnlyField
    (source='author.username')
    votes = serializers.SerializerMethodField
    ('get_votes')

    class Meta:
        model = Post
        fields = ('id', 'title', 'author', 'excerpt',
                  'content', 'status', 'image_url',
                  'votes',)
```

Fig. 4.7

This is one of the serializers written in the app. “PostSerializer” , “VoteSerializer”, “CommentSerializer”

The fields attribute in the “Meta” contains the tuple of attributes of the database model which need to be serialized.

tests.py

This is the file which contains the tests for the models, permissions , serializers and the views written in the app to ensure every functionality is going right and helps us at the time of debugging to understand which part of the project has bugs to be fixed by running the tests.

urls.py

Earlier we have come across an urls.py file which is the url pattern defining page of the project. There we used the http://127.0.0.1.8000/api to connect with the app "blog". So that integration of the url pattern takes us to this urls.py file where the url routes defined for this specific app are defined.

```
urlpatterns = [
    path('posts/', PostList.as_view(), name='post-list'),
    path('posts/<int:pk>', PostDetail.as_view(), name='post-detail'),
    path('posts/<int:pk>/vote', VoteCreate.as_view(), name='votes'),
    path('posts/<int:pk>/comments', CommentCreate.as_view(), name='comments-list'),
    path('posts/<int:pk>/comments/<int:com>',
         CommentDetail.as_view(), name='comments-detail'),
]
```

Fig. 4.8

The routes defined here are to be added to the urls patterns from the project urls.py.

Here "posts/" meant http://127.0.0.1.8000/api/posts this the url which takes us to the home route where all the posts by the user will be visible. The data that will be avaiable in the API retrived will be the same data that was used from the models and serialized in the serializers.py

Explanation of Routes:

http://127.0.0.1.8000/api/posts

The above url will get to the home route showing all the posts and it's data in the JSON format.

http://127.0.0.1.8000/api/posts/<int:pk>

The above url is to view on specific posts "<int:pk>" is refering to the post of id which is equal to the post id that was assigned in the database tabels. Here "pk" means primary key which is used to search in the database tables. So the above route is like a filter which picks one specific posts of all the published posts and the same route for author will have the access to edit and update, but for a non-author has only read-only access.

http://127.0.0.1.8000/api/posts/<int;pk>/vote

This route directs us to the page where we can upvote a one particular post based on the primary key "pk" used in the route. One specific user can only add one vote for a particular post, maintaining the correct count of the upvotes for that post. The user also has the option of deleting their upvote in the same url route.

http://127.0.0.1.8000/api/posts/<int:pk>/comment

This url takes us to the page where the user can comment on a particular post whose id is taken from the url i.e the primary key "pk" . Each user can add any number of comments to the post and the specific author of the comment only has the access to edit, update or delete the comment.

http://127.0.0.1.8000/api/posts/<int:pk>/comments/<int:pk>

This is the url that takes us to page where the specific author can do the edit, update or delete the content in the comments. It uses the new parameter "com" as the primary key to seach for the specific comment on the specific post with the id of primry key "pk".

views.py

This is the main files where all the models, serializers, permissions and url patterns connect to create the complete information that is to be display on the page when accessed from url route from the database based on serialized data and permissions. The name itself is self-explanatory that is, it is used to create the views of the APIs. In this, we use the generics from DjangoRestFramework to create the API views which makes it easy to use the default django views or even customizable by overriding the default functions to make it more convenient for the project.

```
class PostList(generics.ListCreateAPIView):
    permission_classes = [permissions.IsAuthenticated]
    queryset = Post.postobjects.all()
    serializer_class = PostSerializer

    def perform_create(self, serializer):
        serializer.save(author=self.request.user)


class PostDetail(generics.RetrieveUpdateDestroyAPIView):
    permission_classes = [PostUserWritePermission]
    queryset = Post.objects.all()
    serializer_class = PostSerializer
```

Fig. 4.9

```
class CommentCreate(generics.ListCreateAPIView):
    serializer_class = CommentSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        post = Post.objects.get(pk=self.kwargs['pk'])
        return Comment.objects.filter(post=post)

    def perform_create(self, serializer):
        serializer.save(author=self.request.user,
                        post=Post.objects.get(pk=self.kwargs['pk']))
```

Fig. 4.10

4.4 templates (directory)

Fig. 4.11

This is the directory that contains the frontend pages for the basic, home and redirect routes. We have implemented the basic index page which redirects user to google login and registration.

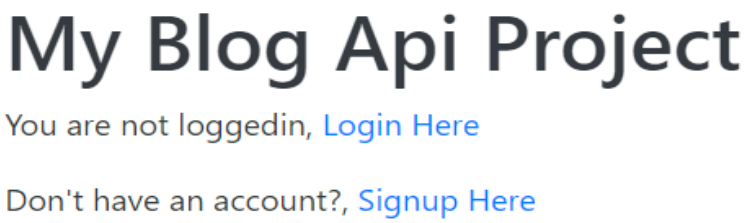

Fig. 4.12

Clicking the “Login Here” or “Signup Here” in the above screen will redirect to the default html provided by Google API where we can login or signup through our Google accounts. After logging in, it will take us to the home page of the Blog API.

5. RESULTS

As mentioned in the Overview of the project (module 2), we were able to implement most of the features and functionalities which were proposed at the starting of the project. The CRUD (create, retrieve, update, and delete) features along the respective permissions that are to be taken care of. The only one feature we were not able to add was the search filter which will enable the users to use that filter to get the targeted posts based on the user who posted it, based on the category that the post was made in, or based on the posts that the current user has upvoted to and so on.

We could not implement it as we were using the google OAuth as the social provider for authentication into our project which made the User model schema to be the default one by the Django so adding filters will not be possible at this level project. Also, the data is maintained in access tokens format with the google authentication which will be stored in hash on which filters cannot be used.

Here are the output results of the web APIs as per routes and urls that were in the url patterns in the project as well as the blog app.

Landing on the localhost

http://127.0.0.1.8000

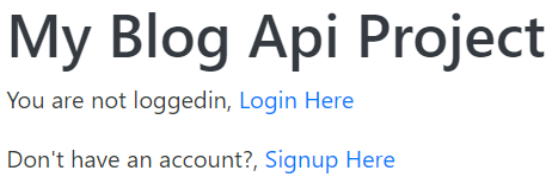

Fig. 5.1

http://127.0.0.1.8000/accounts/login

This is the default template given when Google authentication API is used.

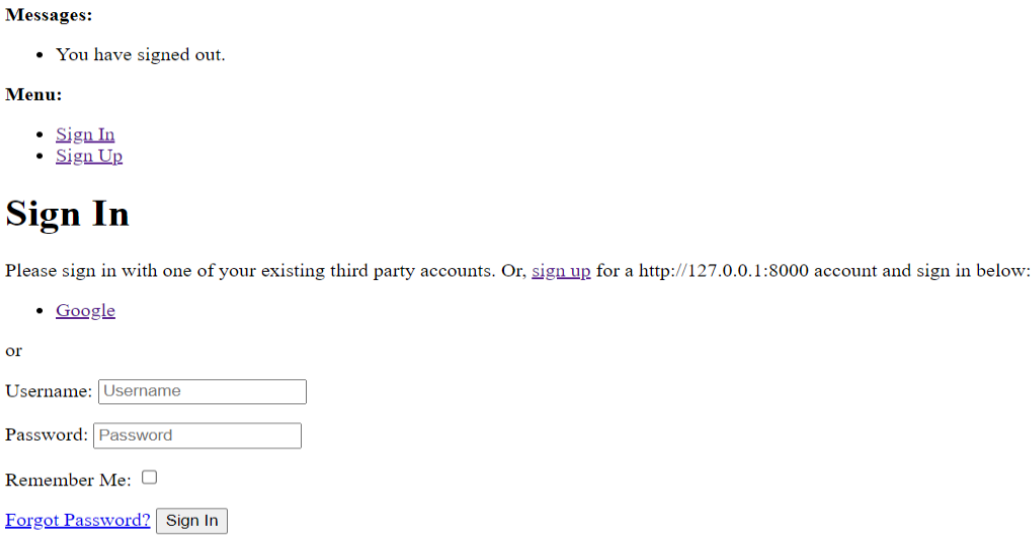

Fig. 5.2

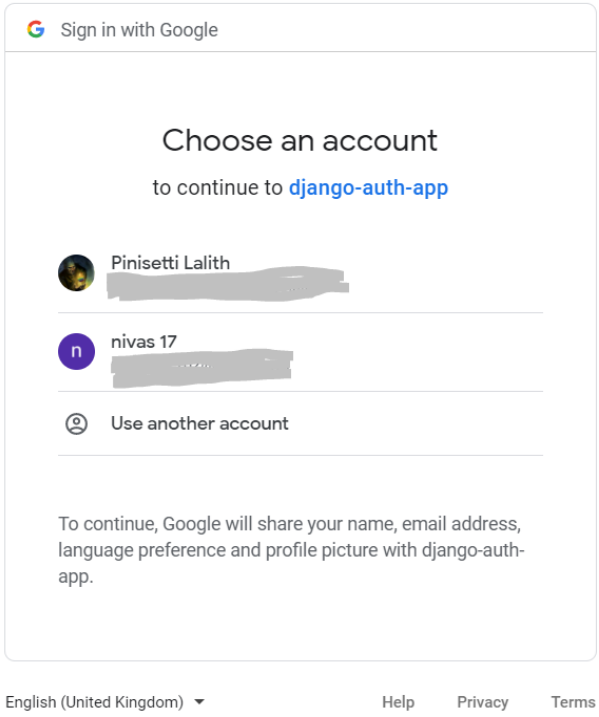

Fig. 5.3

Succesfully logging in or signing up with Google takes us to the same page, telling us that we have logged into the BlogAPI project.

Fig. 5.4

Checking out the routes individually and the output.

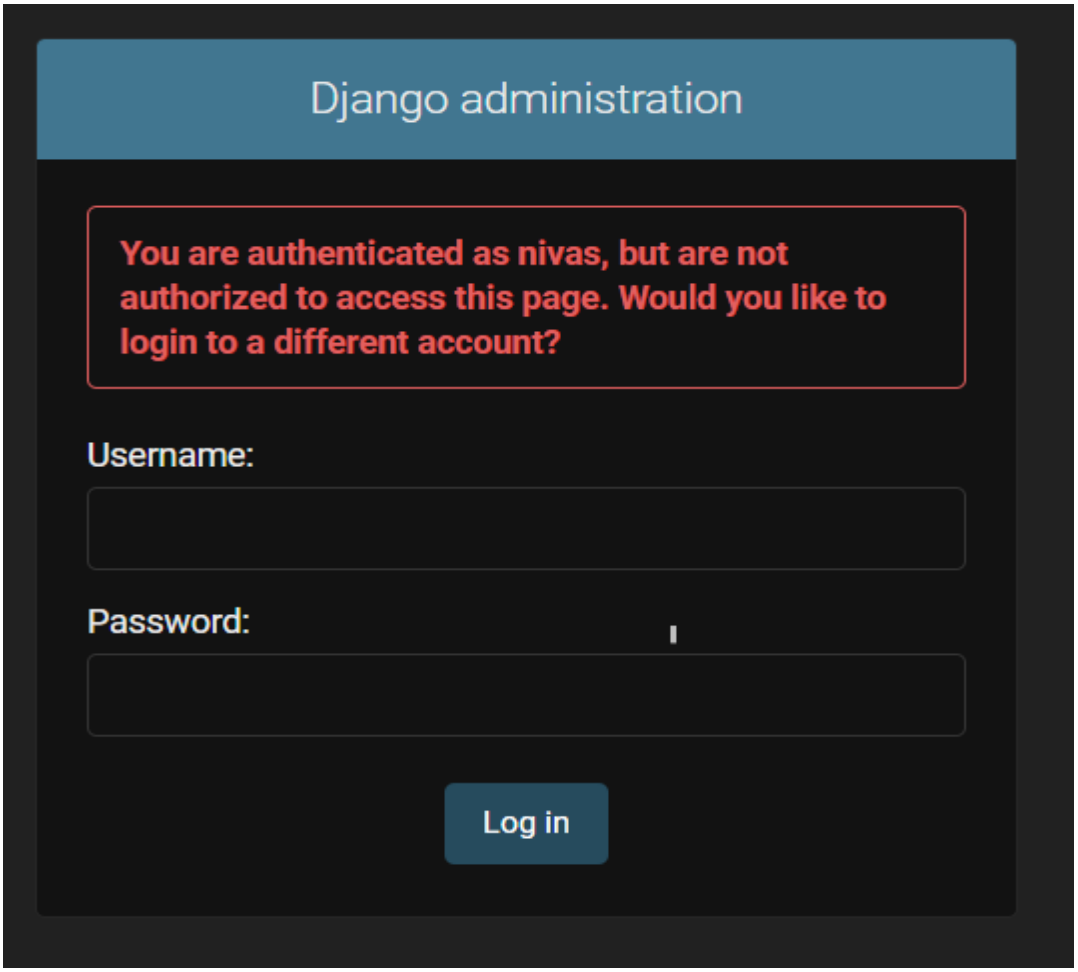

Fig. 5.5

This is the page that is displayed when a non staff member tries to access the administration.

Once an admin user logs in, the data models are visible for manipulation.

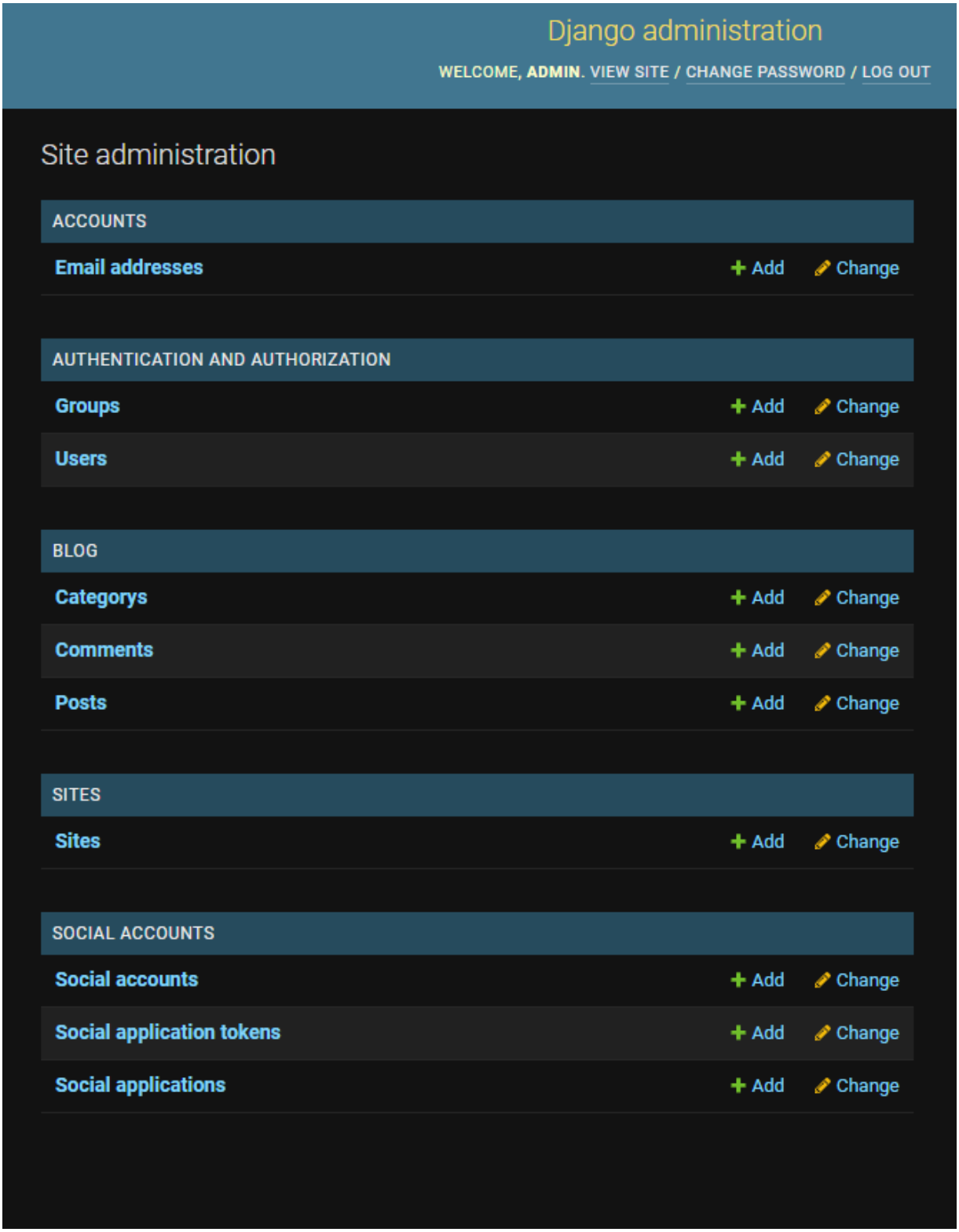
Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Site administration
ACCOUNTS
Email addresses + Add Change
AUTHENTICATION AND AUTHORIZATION
Groups + Add Change
Users + Add Change
BLOG
Categorys + Add Change
Comments + Add Change
Posts + Add Change
SITES
Sites + Add Change
SOCIAL ACCOUNTS
Social accounts + Add Change
Social application tokens + Add Change
Social applications + Add Change

Fig. 5.6

http://127.0.0.1.8000/api/posts

This url shows all the posts that are visible which are published by the author.

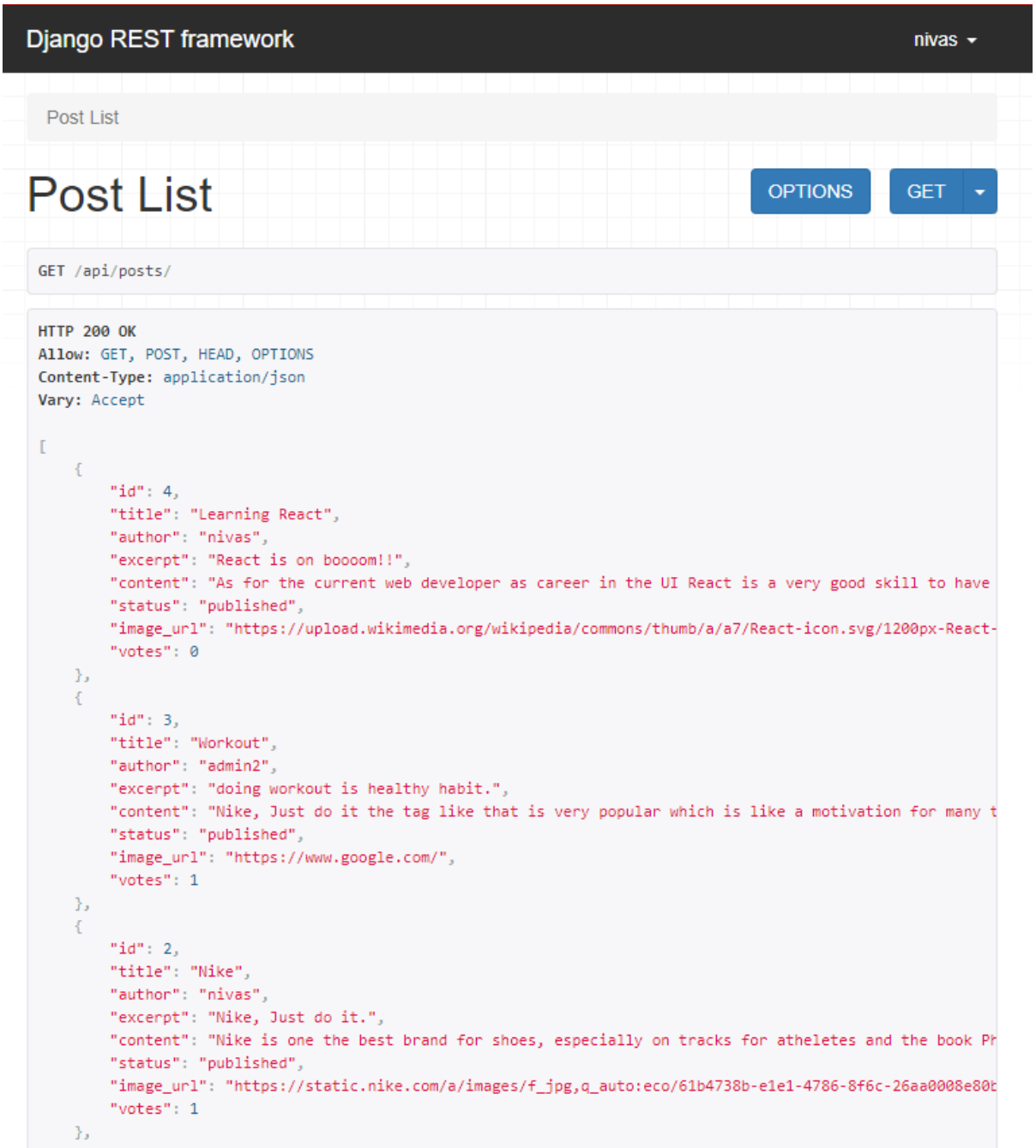

Fig. 5.7

Fig. 5.8

On the same page, every user will have this form which they can fill to post content on the site.

http://127.0.0.1.8000/api/posts/2

Going to this route will make it possible to update and delete the post if the current user is same as the author of the post, otherwise the user will only have the view access of the post.

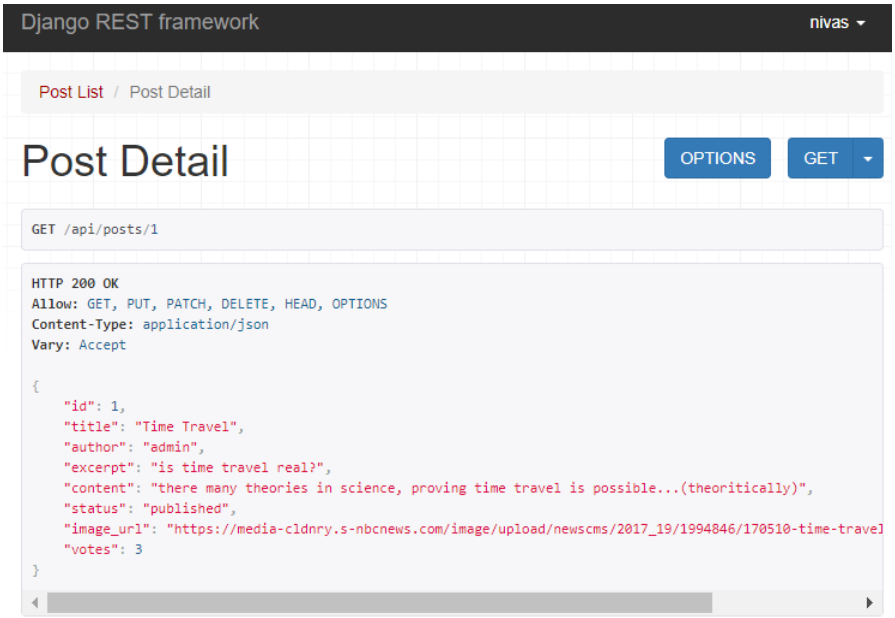

Fig. 5.9

This account of the current user is seen on the top right of the page which shows information about the current active user on the site.

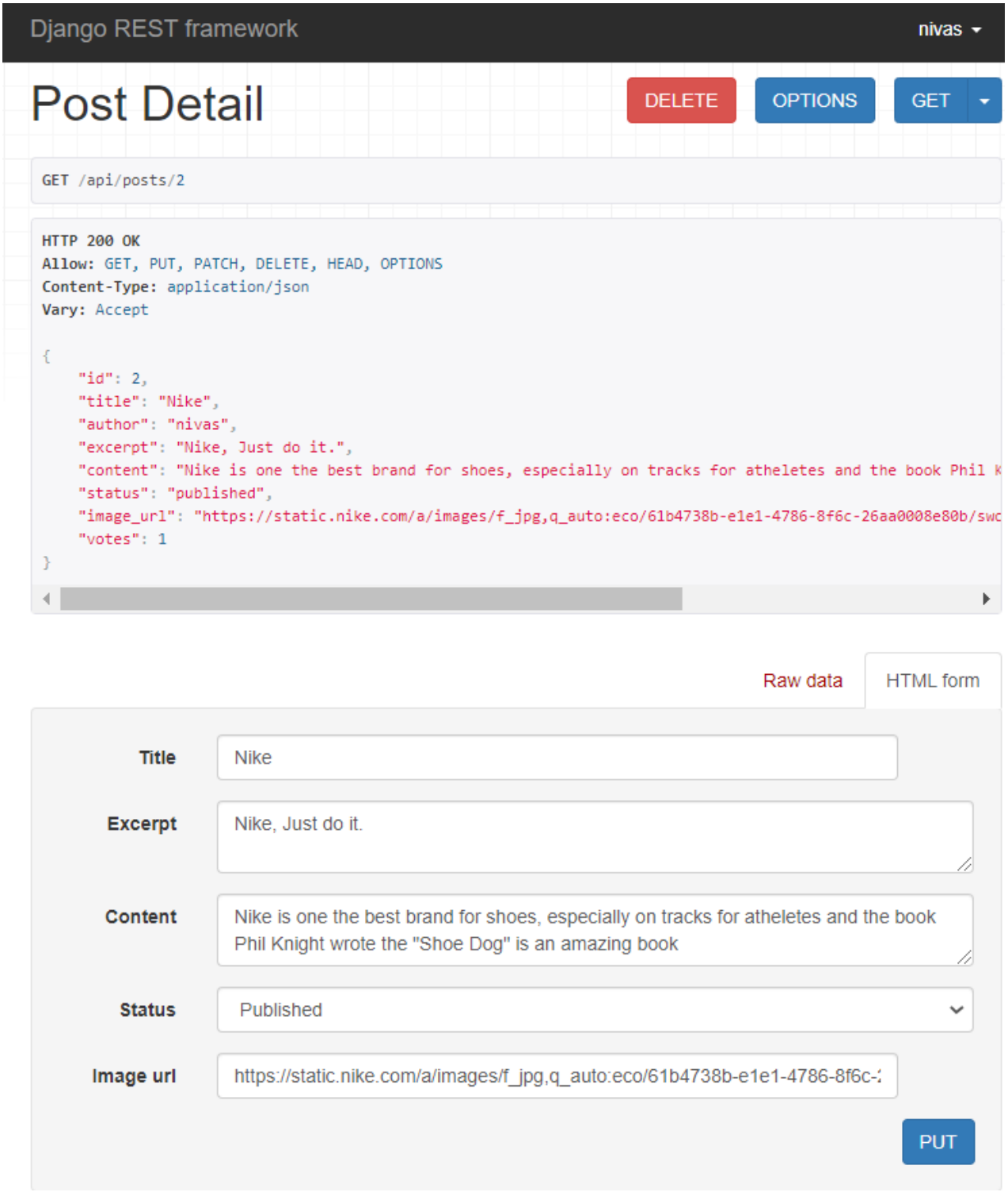
Django REST framework
nivas
Post Detail
DELETE
OPTIONS
GET
GET /api/posts/2
HTTP 200 OK
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
{
"id": 2,
"title": "Nike",
"author": "nivas",
"excerpt": "Nike, Just do it.",
"content": "Nike is one the best brand for shoes, especially on tracks for atheletes and the book Phil K
"status": "published",
"image_url": "https://static.nike.com/a/images/f_jpg,q_auto:eco/61b4738b-e1e1-4786-8f6c-26aa0008e80b/swc
"votes": 1
}
Raw data
HTML form
Title
Nike
Excerpt
Nike, Just do it.
Content
Nike is one the best brand for shoes, especially on tracks for atheletes and the book Phil Knight wrote the "Shoe Dog" is an amazing book
Status
Published
Image url
https://static.nike.com/a/images/f_jpg,q_auto:eco/61b4738b-e1e1-4786-8f6c-
PUT

Fig.5.10

http://127.0.0.1.8000/api/posts/3/comments

This route takes to the page which shows the data related to comments of the particular post.

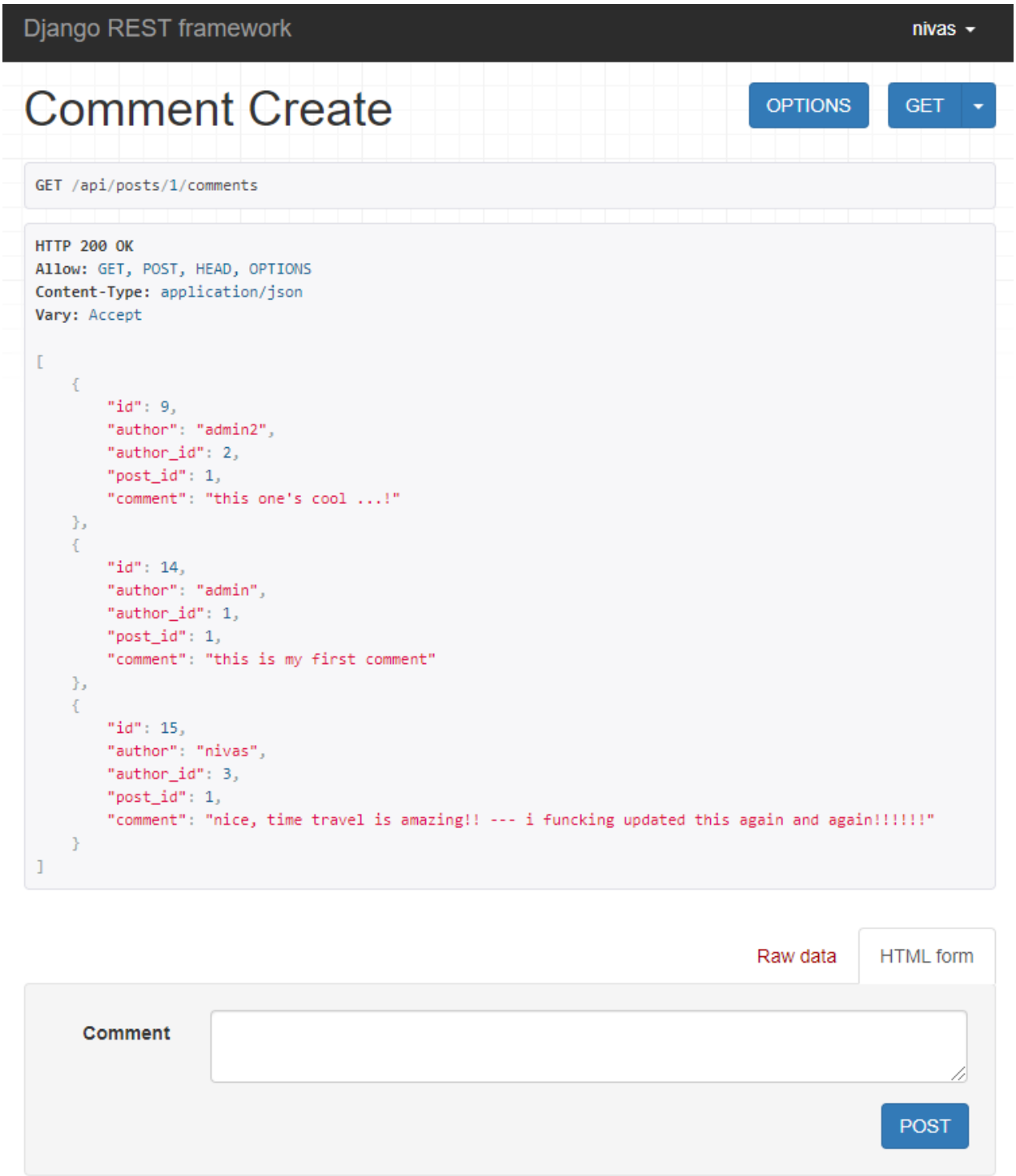

Fig 5.11

http://127.0.0.1.8000/api/posts/1/comments/14

This comment is written by admin user and not the current user i.e not “nivas” so this user will not have access to edit the respective comment.

Fig 5.12

http://127.0.0.1.8000/api/posts/1/comments/15

This comment is written by the same user as the current user so the user will have access to edit the comment written on this post of comment id 14.

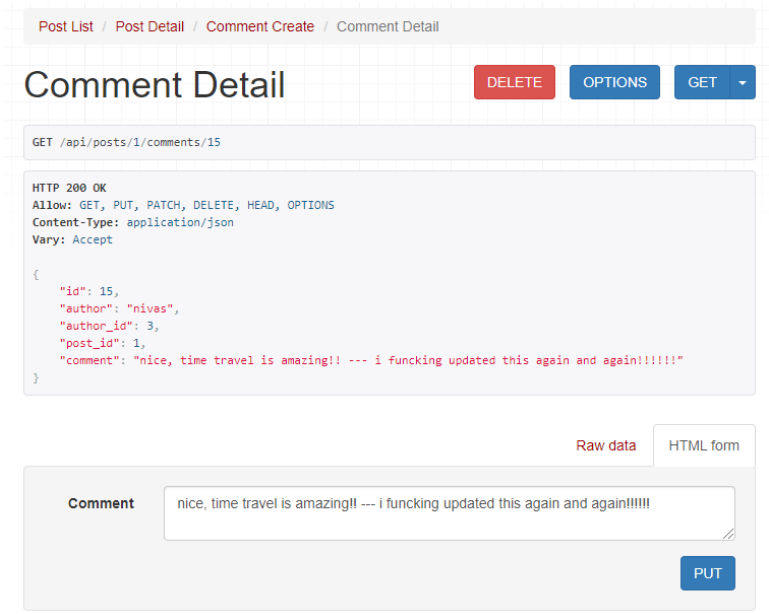

Fig 5.13

http://127.0.0.1.8000/api/posts/1/vote

This route is useful for upvoting the posts which will be visible in the posts detail route (the count of number of votes on that post goes up). But one user can vote only once for a given post. In case he tries to vote again, the below message is displayed.

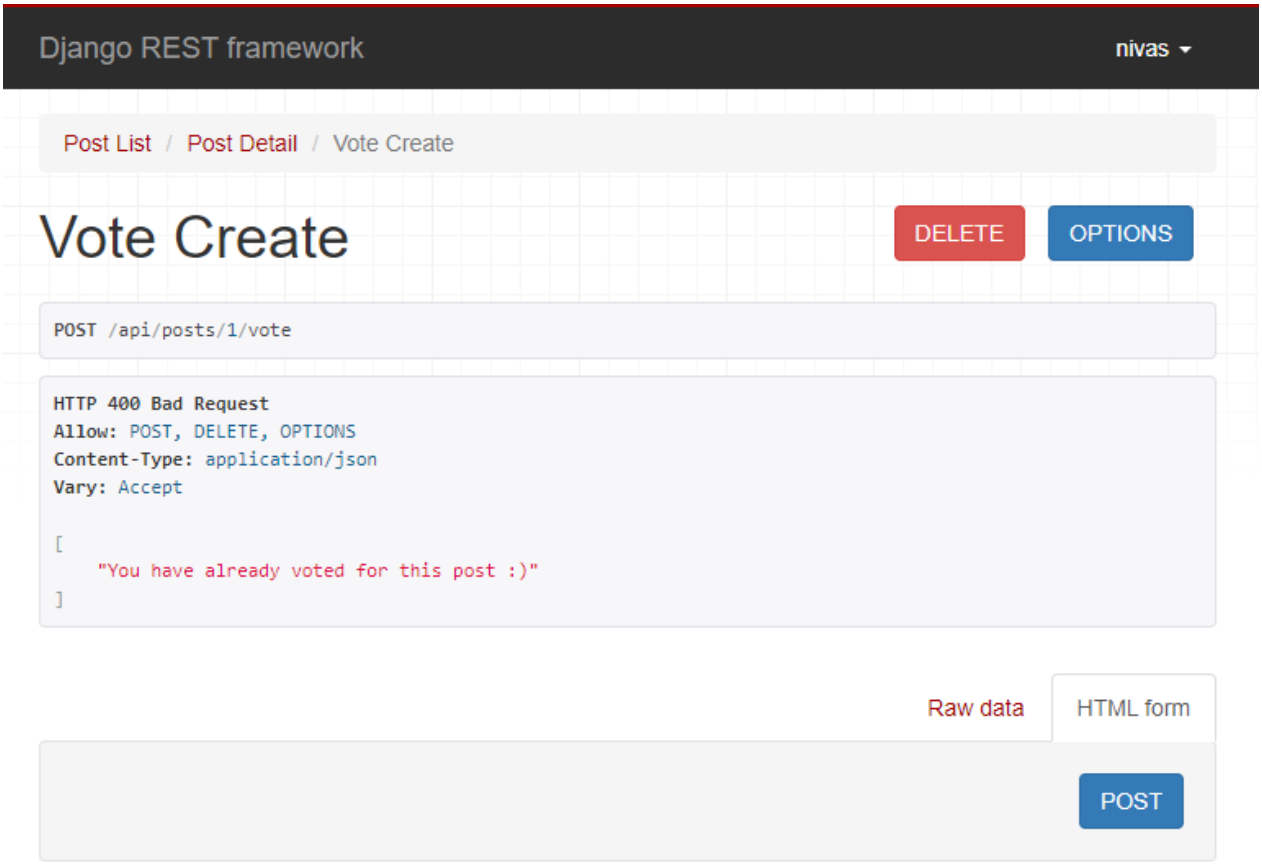

Fig. 5.14

http://120.0.0.1.8000/accounts/logout

This is the route used to logout of the account of the current user creating the below view which is provided by Google Authentication API.

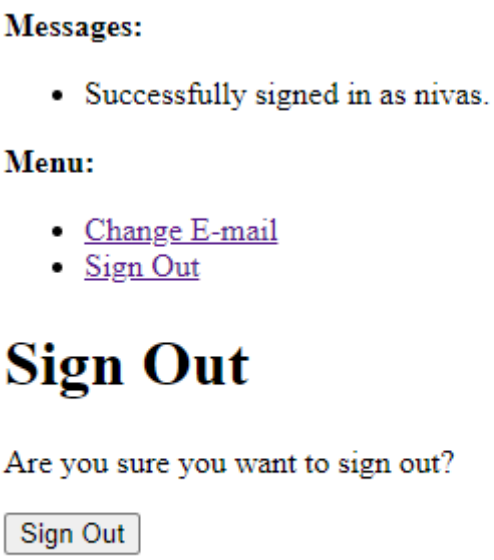

Fig. 5.15

Signing out of the site will redirect to the home page i.e

http://127.0.0.1.8000

and thus completing the execution flow of the project.

6. CONCLUSION & FUTURE WORK

This is a complete Blog API which was made according to the proposed model of the project.

The main features which were implemented as mentioned to the Blog API are the posting of blogs, commenting on the posts, and also voting on the posts with all the CRUD operations according to the required permissions were implemented.

Future Work:

The next main step of implementation will be working on taking the project to the next stage that is to implement the searching and the filtering options based on the category of post, based on the user who posted, based on the upvotes by the user on the posts and more. Furthermore, next step would be to implement the custom user model and custom authentication for the blog site based on the authorization and authentication that admin of the blog site wants.

Taking project to the next level, we will be implementing the UI and UX work with React.js on the front end, thus completely making it an end-to-end full stack project.

Working on such project, which are more inclined towards the backend, server and databases improved our knowledge on databases and models. The overall learning process and the challenges we faced in implementing the Blog API according to the project requirements made it very useful and we learnt a lot.

The resources which we followed in the process of making this project were the official documentation and Django reference books which have been mentioned in the references. These resources were used as per the requirement of the project to fulfil the functionalities and implement the project.

7. REFERENCES:

Fig. 6.1

https://www.django-rest-framework.org/

Fig. 6.2

https://www.djangoproject.com/

Fig. 6.3

https://djangoforbeginners.com/

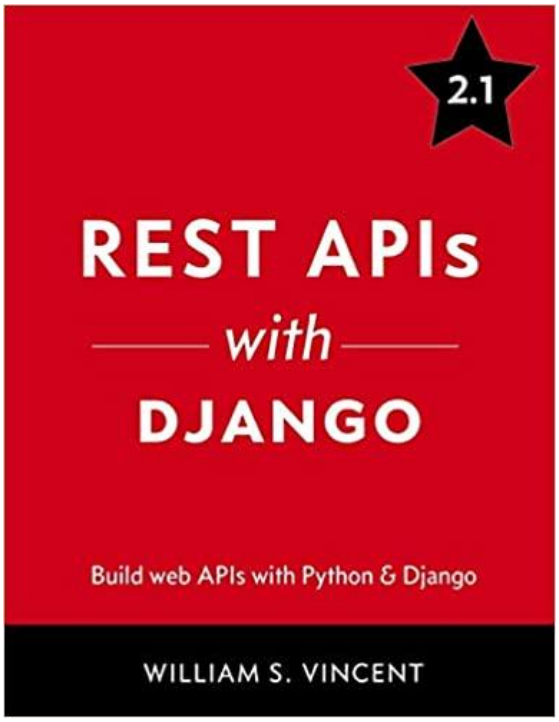

Fig 6.4

https://wsvincent.com/django-rest-framework-tutorial/