Overview and API Documentation of CMS Application

MUNAWAR VP

Python Djnago Full-Stack Developer

Submitting To:

UpForce Tech

Ahmedabad, Gujarat

The CMS Application is developed as per the instruction with the technologies used Python, Django and Django REST Framework.

The complete Source code is added in my github and the link is provided below.

Link : <https://github.com/munawarvp/cms-application.git>

Note:

Create a virtual environment and install requirements.txt for running the project.

APIs and Description

All the APIs are Tested and the Postman’s collection link is provided below.

Link:

<https://www.postman.com/munawarvp/workspace/public-workspace/collection/26537958-2ec693ef-fac5-43a1-8f72-07dbb219a095?action=share&creator=26537958>

1. **create-user/**

For a creating new user with all user credential such as:

First\_name

Last\_name

Username

Email

Phone\_number

Password

Customized Django’s Built-in User model for storing user information.

1. **login-user/**

For authentication user so that, user authorization can be implemented for restricting in many private APIs

Token generation after authentication and token will be sent as response.

1. **list-users/**
2. **user/:user\_id**

For getting details of a particular User. used\_id will be passed in the route.

1. **update-user/:user\_id**

For updating a particular user, user\_id should be passed in the route.

1. **delete-user/:user\_id**

For deleting a particular user, user\_id should be passed in the route.

1. **create-blog/**

For creating a blog based on a particular user. User will be a Foreign key in the blog model.

user

Title

Description

Content

Private (Bool)

created\_date

1. **list-blogs/**

While listing the blogs an additional field is passed that is the total likes got for that particular blog.

1. **blog/:blog\_id**

For getting a particular blog based on the id passed. If the blog is private only the creator of the blog can see, Each request token will be passed in request headers and decoding it, cross check it with the creator of the blog.

Based on that the response is sent back

1. **update-blog/:blog\_id**
2. **delete-blog/:blog\_id**

Update/Delete of a blog is restricted to the Creator of the blog same as the token based cross checking.

1. **like-blog/**

Based on the input of blog\_id and user\_id a new object is created in the Like table. Which makes a relationship between like and user to particular blog.

We can take the count of the like object for a blog to get the number of likes it has.

User in the like table is One-To-One relationship, So that one user cannot have multiple likes for a blog.

1. **blog-likes/:blog\_id**

For getting a single blog details along with the number of likes it has with one query. From the Like model we can access the blog models data since it is related to the blog model.

1. **dislike/:blog\_id**

For removing the like of a blog, the user\_id will be passed in the headers and particular like will be deleted from the Like model.