ONEFIN

Backend Assignment Fresher

Introduction

The assignment is to judge your understanding of various concepts which are required to develop a working Django project. To submit the assignment push your code on Github and share the link of the repository with us. You will have three days time since you receive the assignment to finish the assignment.

We will evaluate the assignment by asking these questions:

- Is your code actually doing what is asked in the assignment?
- Is your code clean and modular? Are you cleanly factoring your views to keep most of the code outside of views?
- Is your code following PEP8 code formatting guidelines?
- Have you followed Django development best practices?
- Have you added tests for your code?
- Is your requirements file updated with all requirements?

We need to develop a web application which allows people to create collections of movies they like. There are two parts to this:

(1) Integration with an API which serves a list of movies and their genres, details are specified here.

(2) Use the API to list movies for the users of your web application[1]. They should be able to see the list of movies and add any movie which they like into their collections. Each user can create multiple collections and multiple movies into the collections. Develop APIs for this application per specification below. No frontend is required to be developed.

[1] Note that only an API serving JSON responses per the specification is to be created and no HTML viewer of the API responses is needed.

Integrate with movie listing API

Integrate a third party API which serves a list of movies. You have to use basic auth for authentication, id and secret for which are given below.

Notes
Note that the password has no space in it, if you copy-paste the password, you may end up putting a space in it.

```
Username:
iNd3jDMYRKsN1pjQPMRz2nrq7N99q4Tsp9EY9cM0

Password[2]:
Ne5DoTQt7p8qrgkPdtenTK8zd6MorcCR5vXZIJNfJwvfafZfcOs4reyasVYddTyXCz9h
cL5FGGIVxw3q02ibnBLhblivqQTp4BIC93LZHj4OppuHQUzwugcYu7TIC5H1

GET https://demo.credy.in/api/v1/maya/movies/

The response is a paginated list of movies, returning 10 movies at a
time:

{
    “count”: <total number of movies>,
    “next”: <link for next page, if present>,
    “previous”: <link for previous page>,
    “data”: [
        {
            “title”: <title of the movie>,
            “description”: <a description of the movie>,
            “genres”: <a comma separated list of genres, if
present>,
            “uuid”: <a unique uuid for the movie>
        },
        ...
    ]
}
```

[2] Note that there is no space in the password. Many times while copy pasting the password, there is an additional space or newline which gets copied, which causes authentication to fail.

Implement APIs for your web application

In your web application, you should allow users to register using a username and password, which should return a JWT token which should be used for authentication. All requests except the registration one should be authenticated. Post-registration, the user should be able to create collections and add movies to their collections, view, modify and delete them, essentially, you need to create CRUD APIs for collections. For all APIs and responses related to collections, you can create models, where all data related to those APIs are stored.

```
POST http://localhost:8000/register/

Request Payload:

{
    “username”: <desired username>,
    “password”: <desired password>
}

Response:

{
    “access_token”: <Access Token>
}

GET http://localhost:8000/movies/

This should return a paginated list of movies which are available. Note that this API should be actually calling the third party API. This data should not be obtained from your models/database. Sample response below:

{
    “count”: <total number of movies>,
    “next”: <link for next page, if present>,
    “previous”: <link for previous page>,
    “data”: [
        {
            “title”: <title of the movie>,
            “description”: <a description of the movie>,
```

```
            “genres”: <a comma separated list of genres, if
present>,
            “uuid”: <a unique uuid for the movie>
        },
        ...
    ]
}
```

GET http://localhost:8000/collection/

This should return my collection of movies and my top 3 favourite genres based on the movies across all my collections. Note that the response of this API need not include the actual movies inside the collections, there is a separate API for that purpose.

```
{
    “is_success”: True,
    “data”: {
        “collections”: [
            {
                “title”: “<Title of my collection>”,
                “uuid”: “<uuid of the collection name>”
                “description”: “My description of the collection.”
            },
            ...
        ],
        “favourite_genres”: “<My top 3 favorite genres based on the
movies I have added in my collections>.”
    }
}
```

POST http://localhost:8000/collection/

This API creates a collection. The data for the movies should be saved in the database in this API.

Request payload:

```
{
    “title”: “<Title of the collection>”,
    “description”: “<Description of the collection>”,
```

```
    “movies”: [
        {
            “title”: <title of the movie>,
            “description”: <description of the movie>,
            “genres”: <generes>,
            “uuid”: <uuid>
        }, ...
    ]
}
```

Response payload:

```
{
    “collection_uuid”: <uuid of the collection item>
}
```

PUT http://localhost:8000/collection/<collection_uuid>/

This should update the movie list in the collection.

Request:

```
{
    “title”: <Optional updated title>,
    “description”: <Optional updated description>,
    “movies”: <Optional movie list to be updated>,
}
```

GET http://localhost:8000/collection/<collection_uuid>/

Response:

```
{
    “title”: <Title of the collection>,
    “description”: <Description of the collection>,
    “movies”: <Details of movies in my collection>
}
```

DELETE http://localhost:8000/collection/<collection_uuid>/

This should delete the collection.