

# Backend Assignment

Introduction 1

Integrate with movie listing API 2

Implement APIs for your web application 3

Implement a request counter middleware 6

## Introduction

The assignment is to judge your understanding of various concepts which are required to be applied in a Django project to make it resilient and working under concurrent requests. 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?
- Does your counter implementation work in parallel execution of code?
- Is your code following PEP8 code formatting guidelines?
- Have you followed Django development best practices?
- Have you added tests for your code? You can use Factory boy for generating fixtures. Are meaningful assertions present in your tests?
- 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 three 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. 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.
3. We also want to have a monitoring system of counting the number of requests done to your project. Write a custom middleware for counting all requests coming to your server too, and an API to monitor it too. Details [here](#).

## Integrate with movie listing API

Integrate a third party API which serves a list of movies. Note that this API is flaky, and often fails and/or time outs please built-in retry mechanisms in your project. You have to use basic auth for authentication, id and secret for which are given below.

### Notes

- Do not hardcode the client and client secret in your code, you can get it from os environment variables.
- 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:

Ne5DoTQt7p8qrgkPdtenTK8zd6MorcCR5vXZIjNfJwvfafZfc0s4reyasVYddTyXCz9h  
cL5FGGIVxw3q02ibnBLhblivqQTp4BIC93LZHj40ppuHQUzwugcYu7TIC5H1

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>
    },
    ...
  ]
}
```

## 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": <genres>,
      "uuid": <uuid>
    }, ...
  ]
}
```

Response payload:

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

PUT [http://localhost:8000/collection/<collection\\_uuid>](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.

## Implement a scalable request counter middleware

Below API should return the number of requests which have been served by the server till now. Note that this should also work in a concurrent environment and should be scalable. There should be another API to reset the counter.

GET http://localhost:8000/request-count/

Response:

```
{  
    "requests": <number of requests served by this server till now>.  
}
```

POST http://localhost:8000/request-count/reset/

Response:

```
{  
    "message": "request count reset successfully"  
}
```