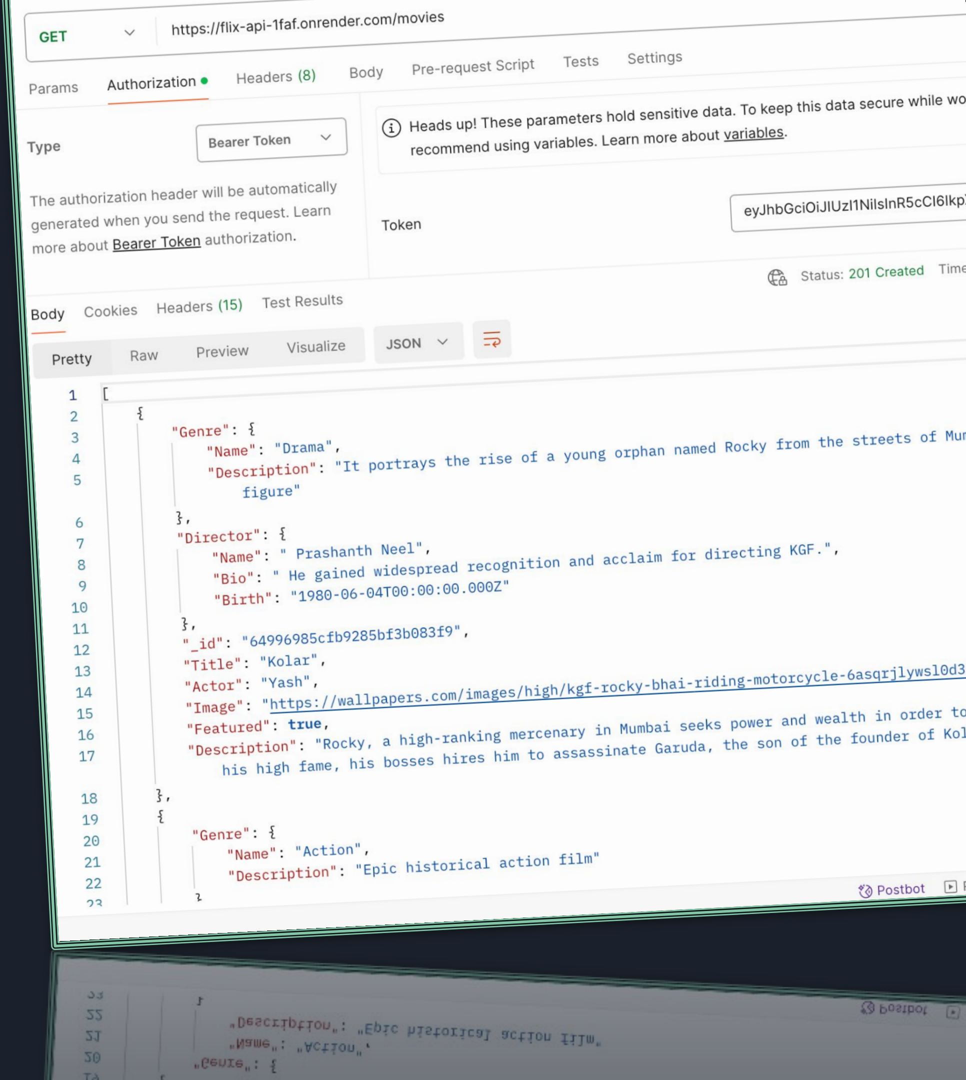
Case Study of FlixAPI

Sanju Shah - Full Stack Developer

Overview

FlixAPI is an RESTful API developed with the Node.js and express deployed in render that interacts with a non-relational database - MongoDB and allows users to access movie details and manage their accounts.

GitHub



GET https://flix-api-1faf.onrender.com/movies

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Heads up! These parameters hold sensitive data. To keep this data secure while wo recommend using variables. Learn more about variables.

Token: eyJhbGciOiJIUzI1NiIsInR5cCI6Ikp

Status: 201 Created Time

Body Cookies Headers (15) Test Results

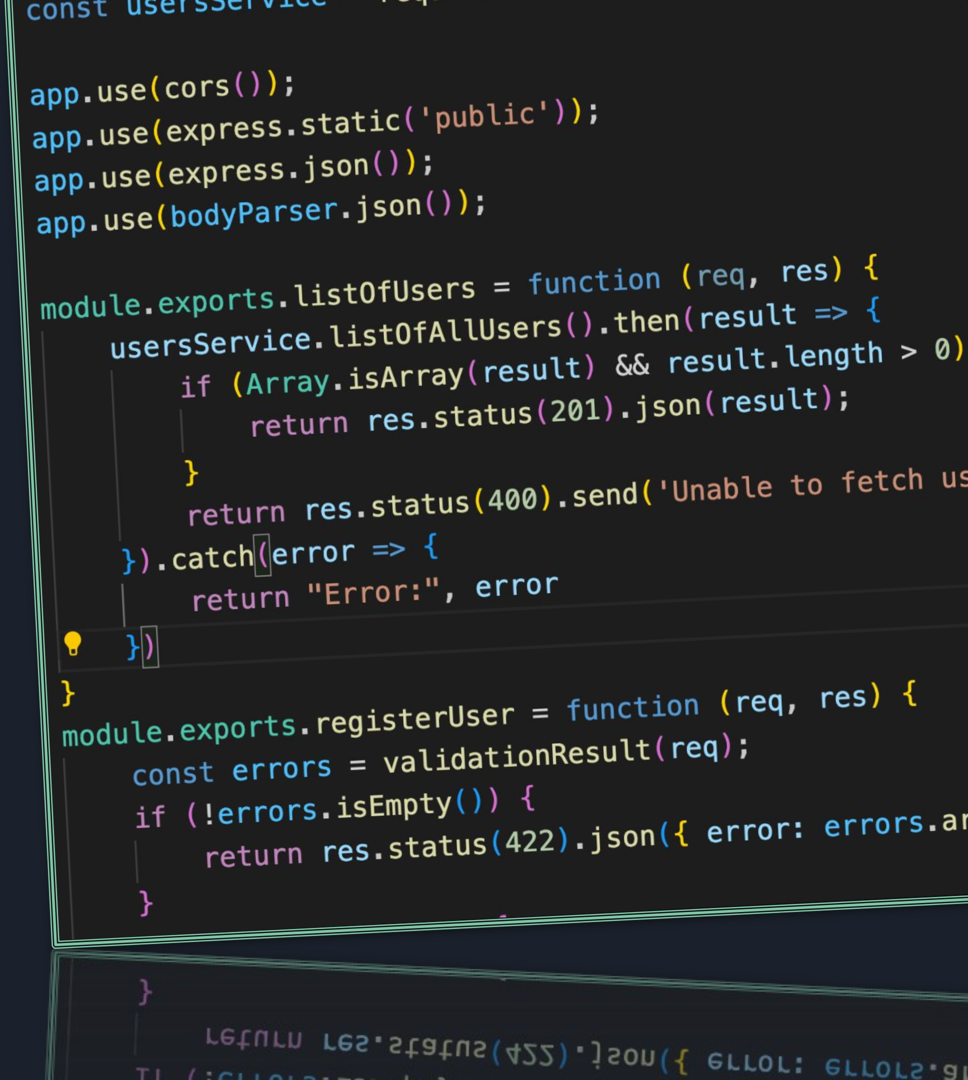
Pretty Raw Preview Visualize JSON

```
[
    {
        "Genre": {
            "Name": "Drama",
            "Description": "It portrays the rise of a young orphan named Rocky from the streets of Mun
                figure"
        },
        "Director": {
            "Name": " Prashanth Neel",
            "Bio": " He gained widespread recognition and acclaim for directing KGF.",
            "Birth": "1980-06-04T00:00:00.000Z"
        },
        "_id": "64996985cfb9285bf3b083f9",
        "Title": "Kolar",
        "Actor": "Yash",
        "Image": "https://wallpapers.com/images/high/kgf-rocky-bhai-riding-motorcycle-6asqrjlywsl0d3
        "Featured": true,
        "Description": "Rocky, a high-ranking mercenary in Mumbai seeks power and wealth in order to
            his high fame, his bosses hires him to assassinate Garuda, the son of the founder of Kol
    },
    {
        "Genre": {
            "Name": "Action",
            "Description": "Epic historical action film"
        }
```

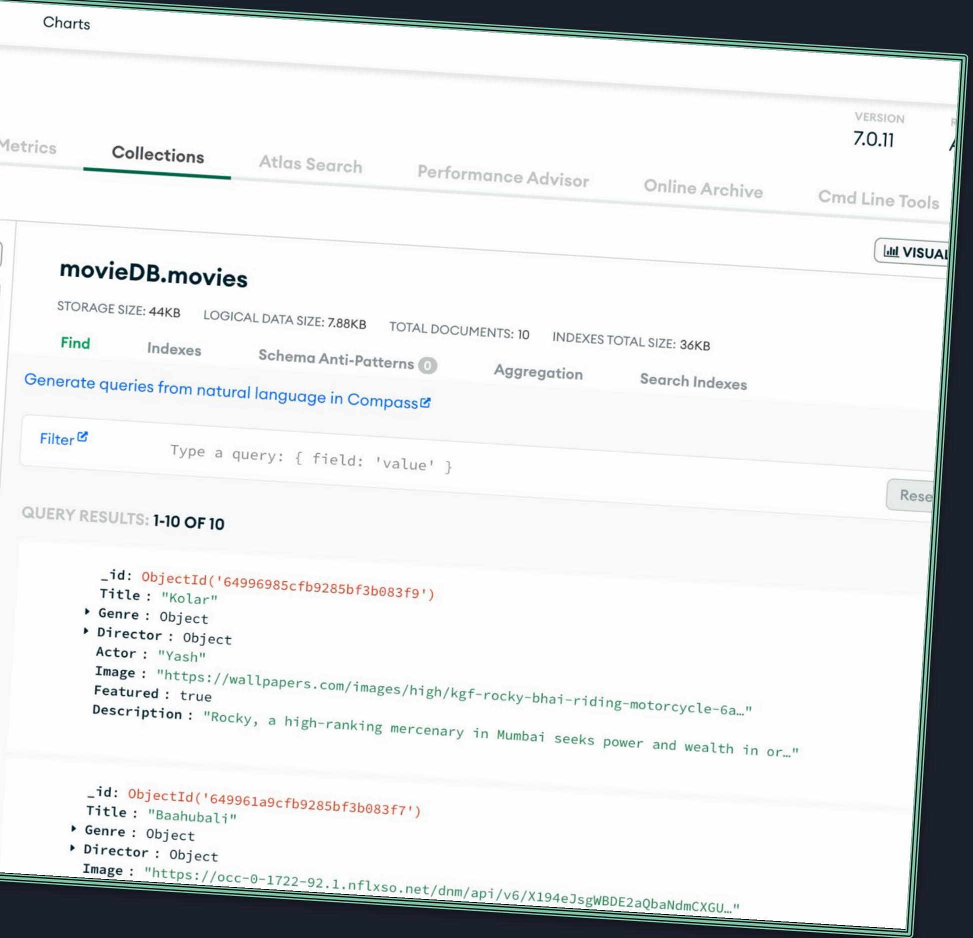
Postbot

Purpose

This project was created as part of Full-Stack Immersion course at CareerFoundry to enhance my skill in web development.



```
const usersService = req

app.use(cors());
app.use(express.static('public'));
app.use(express.json());
app.use(bodyParser.json());

module.exports.listOfUsers = function (req, res) {
    usersService.listOfAllUsers().then(result => {
        if (Array.isArray(result) && result.length > 0)
            return res.status(201).json(result);
        }
        return res.status(400).send('Unable to fetch us
    }).catch(error => {
        return "Error:", error
    })
}
module.exports.registerUser = function (req, res) {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(422).json({ error: errors.ar
    }
```


Charts

VERSION
7.0.11

Metrics | Collections | Atlas Search | Performance Advisor | Online Archive | Cmd Line Tools

VISUAL

movieDB.movies

STORAGE SIZE: 44KB LOGICAL DATA SIZE: 7.88KB TOTAL DOCUMENTS: 10 INDEXES TOTAL SIZE: 36KB

Find | Indexes | Schema Anti-Patterns 0 | Aggregation | Search Indexes

Generate queries from natural language in Compass

Filter Type a query: { field: 'value' }

Rese

QUERY RESULTS: 1-10 OF 10

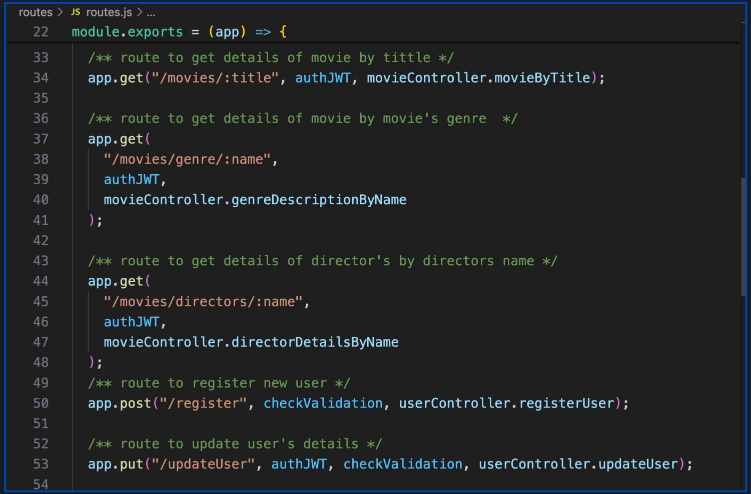
```
 _id: ObjectId('64996985cfb9285bf3b083f9')
 Title : "Kolar"
▸ Genre : Object
▸ Director : Object
 Actor : "Yash"
 Image : "https://wallpapers.com/images/high/kgf-rocky-bhai-riding-motorcycle-6a…"
 Featured : true
 Description : "Rocky, a high-ranking mercenary in Mumbai seeks power and wealth in or…"

 _id: ObjectId('649961a9cfb9285bf3b083f7')
 Title : "Baahubali"
▸ Genre : Object
▸ Director : Object
 Image : "https://occ-0-1722-92.1.nflxso.net/dnm/api/v6/X194eJsgWBDE2aQbaNdmCXGU…"
```

Objective

- The objective of this project was to build a server side application from scratch that uses node.js, express.
- Provide a Comprehensive Movie Database.
- Allow users to register, log in, update their profiles, and manage their favourite lists.

Technology Stack

- **Node.js** as runtime environment for executing JavaScript code on the server side.
- RESTful FlixAPI is created on **Express.js** framework.
- For deploying and hosting FlixAPI, **Render** cloud platform is used.
- **MongoDB** is used for storing movie and user data.
- For secure user authentication and authorization, JSON Web Tokens is implemented.


routes > JS routes.js > ...

```
module.exports = (app) => {

  /** route to get details of movie by tittle */
  app.get("/movies/:title", authJWT, movieController.movieByTitle);

  /** route to get details of movie by movie's genre  */
  app.get(
    "/movies/genre/:name",
    authJWT,
    movieController.genreDescriptionByName
  );

  /** route to get details of director's by directors name */
  app.get(
    "/movies/directors/:name",
    authJWT,
    movieController.directorDetailsByName
  );
  /** route to register new user */
  app.post("/register", checkValidation, userController.registerUser);

  /** route to update user's details */
  app.put("/updateUser", authJWT, checkValidation, userController.updateUser);

```

Features

- Display lists of movies and filter movie by genre, director.
- Display details of movies such as title, director, actors, plot summary, release year and genre.
- Register new user and enable existing users to log in with their credentials.
- Update user profiles such as email, password, and other profile details.
- Add/remove movie to the favorites list.
- Deregister users and delete their all associated data.

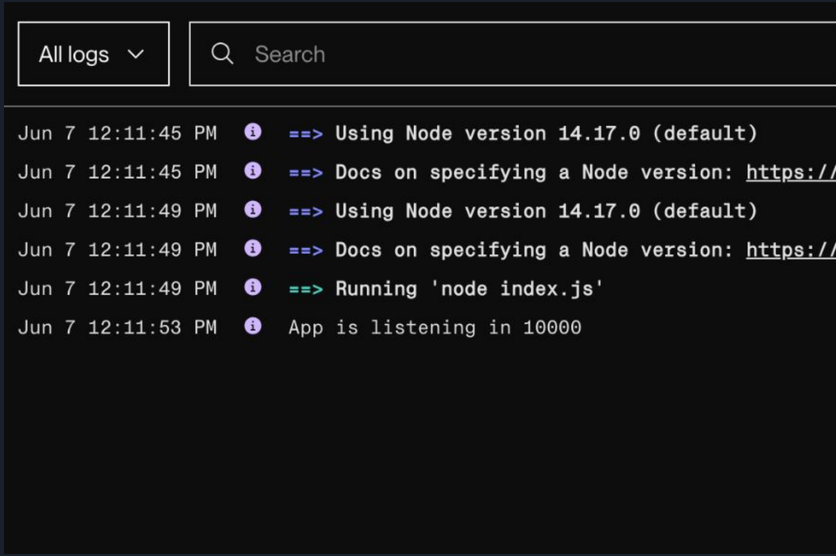
Development Process

- Identified features such as movie details retrieval and manage account.
- Outlined API endpoints, request methods with expected responses.
- Set up a Node.js environment with Express.
- Configured MongoDB to store movie and user data and Mongoose for schema modeling.Implemented JWT for secure user authentication.
- Configured environment variables for secure API key management and database connections.
- Deployed the application in render.
- Performed API testing with Postman to verify endpoint functionality.

Database & Deployment

- MongoDB for its NoSQL nature
  - Offering a cost-effective and flexible solution
  - Schema-less data model that supports rapid iteration and diverse data handling

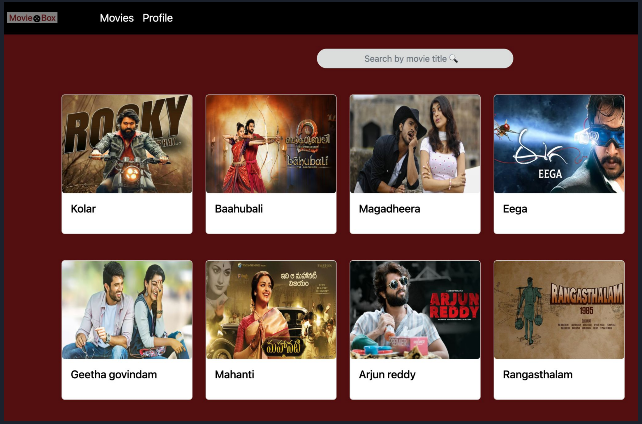
- Deployment on Render platform
  - Easiest and efficient platform for deployment
  - User-friendly interface and robust feature set

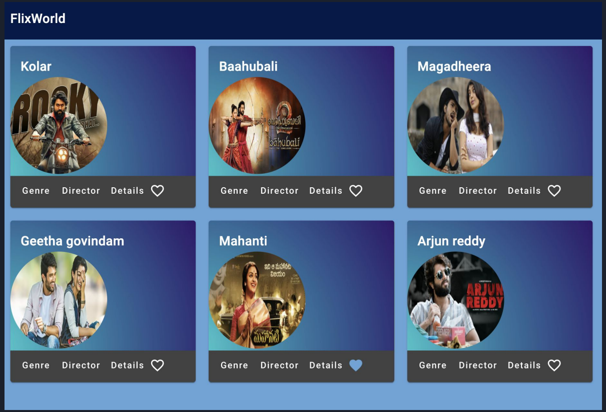


```
Jun 7 12:11:45 PM   ==> Using Node version 14.17.0 (default)
Jun 7 12:11:45 PM   ==> Docs on specifying a Node version: https://
Jun 7 12:11:49 PM   ==> Using Node version 14.17.0 (default)
Jun 7 12:11:49 PM   ==> Docs on specifying a Node version: https://
Jun 7 12:11:49 PM   ==> Running 'node index.js'
Jun 7 12:11:53 PM   App is listening in 10000
```

Currently Client Applications using FlixAPI

- MovieBox App : https://movie-box-glienicker.netlify.app/login
- FlixWorld App: https://sanjushahgupta.github.io/flix-client-angular/





Retrospective

Challenge & Solution

While developing this application, implementing secure user authentication and authorization proved to be quite challenging.

With the guidance of a mentor and extensive research, I successfully implemented secure authentication methods. This included adopting HTTPS for secure data transmission, using JWT (JSON Web Tokens) for stateless authentication, and encrypting sensitive data to protect user information.

Future Improvements

Looking ahead, I plan to enhance FlixAPI by adding more features like sorting movies based on release date and also notifying users when details of new movies are added.

Conclusion

The experience of developing FlixAPI was a journey of tackling challenges, learning new technologies, and improving problem-solving skills. The project has not only met its initial objectives but has also laid a strong foundation for future enhancements.

JUST
THANK YOU!