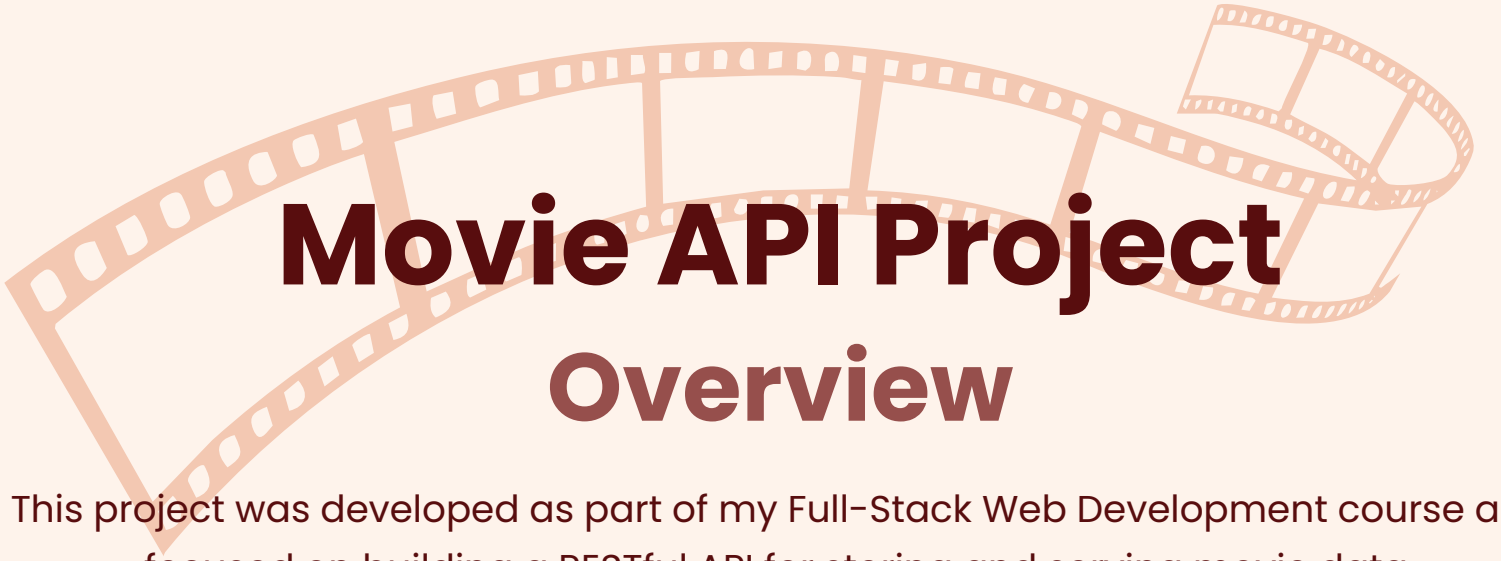
Movie API Project

A RESTful API
delivering dynamic movie data



Movie API Project

Overview

This project was developed as part of my Full-Stack Web Development course and focused on building a RESTful API for storing and serving movie data.

The API allowed client applications to retrieve detailed movie information, including titles, descriptions, genres, directors, and poster images. While the functionality appeared straightforward, the process required learning to work in a non-visual environment and maintaining meticulous consistency across the data layer.

Tech Stack: Node.js, Express.js, MongoDB, Mongoose, Postman, JavaScript

Data Model & Database

To start, I designed the MongoDB schema to capture the full range of movie details:

Title & Description

Core identifying information

Director

A nested object including biography data

Genre

A nested object with name and description fields

Image Path

A link to the movie poster

{"Title": "Creating the Data"}

```
{
  "_id": "0bjectId("6626c73603169fdf54117b7b"),
  "Iitle": "Inception",
  "Description": A thief who steals corporate secrets
  through the use of dream-sharing technology is given
  the inverse task of planting an idea into the mind
  of a C.E.O., but his tragic past may doom the project
  and his team to disaster.",
  "Genre": "Adventure",
  "Description": "Adventure fiction is a type of
  fiction that usually presents danger, or gives
  the reader a sense of excitement."
  "Director": "Christopher Nolan",
  "Bio": "Best known for his cerebral, often nonlinear,
  storytelling, acclaimed Academy Award winner writer/
  director/producer Sir Christopher Nolan CBE was born
  in London, England",
  "ImagePath": "https://m.media-amazon.com/images/M/MV5
  BMjAxMzY7NjcxNFSBMl5BanBnXkftZTcwNTI50TM0Mw
  @._V1_.jpg"
}
```

Then, I created the MongoDB collection manually through the terminal. This approach taught me how each field needed to align precisely with the schema - down to the casing of each property name.

Challenge:
Using the terminal as the primary interface was a big adjustment. As a visual learner, I found it difficult to track which entries were complete or correct. Even a small inconsistency, like typing "director" instead of "Director", could cause the entire query to fail or return incomplete data.

Solution:
I created a reference document listing the exact field names and their casing. After each insert, I immediately tested retrieval to catch errors early.

{"Title": "Creating the Data"}

```
{
  "_id": "0bjectId("6626c73603169fdf54117b7b"),
  "Iitle": "Inception",
  "Description": A thief who steals corporate secrets
  through the use of dream-sharing technology is given
  the inverse task of planting an idea into the mind
  of a C.E.O., but his tragic past may doom the project
  and his team to disaster.",
  "Genre": "Adventure",
  "Description": "Adventure fiction is a type of
  fiction that usually presents danger, or gives
  the reader a sense of excitement."
  "Director": "Christopher Nolan",
  "Bio": "Best known for his cerebral, often nonlinear,
  storytelling, acclaimed Academy Award winner writer/
  director/producer Sir Christopher Nolan CBE was born
  in London, England",
  "ImagePath": "https://m.media-amazon.com/images/M/MV5
  BMjAxMzY7NjcxNFSBMl5BanBnXkftZTcwNTI50TM0Mw
  @._V1_.jpg"
}
```

Then, I created the MongoDB collection manually through the terminal. This approach taught me how each field needed to align precisely with the schema - down to the casing of each property name.

Challenge:
Using the terminal as the primary interface was a big adjustment. As a visual learner, I found it difficult to track which entries were complete or correct. Even a small inconsistency, like typing "director" instead of "Director", could cause the entire query to fail or return incomplete data.

Solution:
I created a reference document listing the exact field names and their casing. After each insert, I immediately tested retrieval to catch errors early.

Building REST Endpoints

I used Express.js to create a series of REST endpoints that exposed the movie data to clients.

Endpoints included:

GET /movies	**GET /movies/:Title**	**GET /genres/:Name**
GET /directors/:Name	**GET /users**	**GET /users/:Username**
POST /users	**PUT /users/:Username**	**DELETE users/:ID**

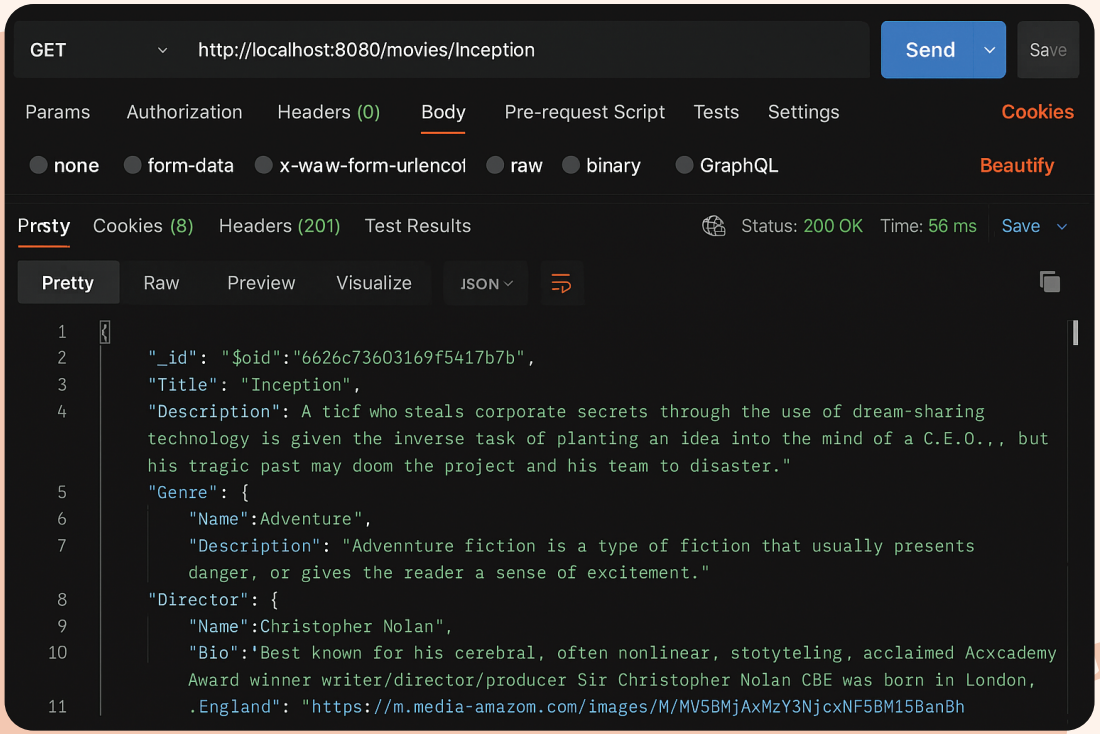
These endpoints relied on Mongoose queries to retrieve the correct documents from MongoDB.

Challenge:
Mapping URL parameters to database queries took trial and error. Early on, I sometimes used the wrong property in my filter (for example, querying by _id when the route expected a title).

Solution:
I created a consistent pattern for naming routes and tested each endpoint in Postman with various cases and inputs.

Testing & Iteration

With the API routes in place, I moved into a testing phase using Postman.



Key test cases included:

- Retrieving all movies to confirm that array responses were formatted properly.
- Testing single-record lookups by title, genre, and director.
- Verifying that nested fields (Genre and Director) were returned as complete objects.
- Every test surfaced opportunities to improve consistency and error handling.

Challenges & Roadblocks

Throughout the project, I encountered several obstacles:

Terminal Learning Curve

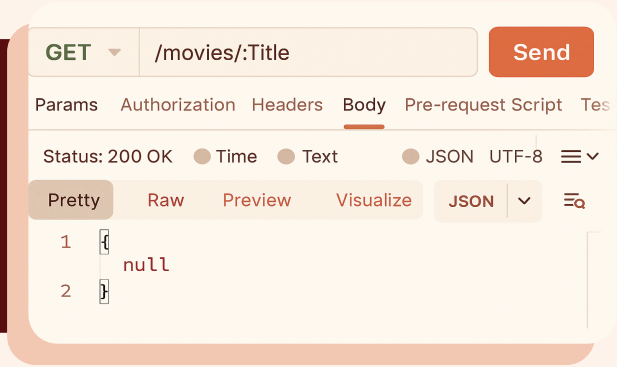
Coming from a visual background, adjusting to a text-only environment was difficult. I often had to repeat commands and cross-reference data to stay organized.

Data Consistency Errors

Typos and inconsistent field names were a recurring problem. This slowed development and required frequent manual corrections in MongoDB.

Query Testing Returns Null

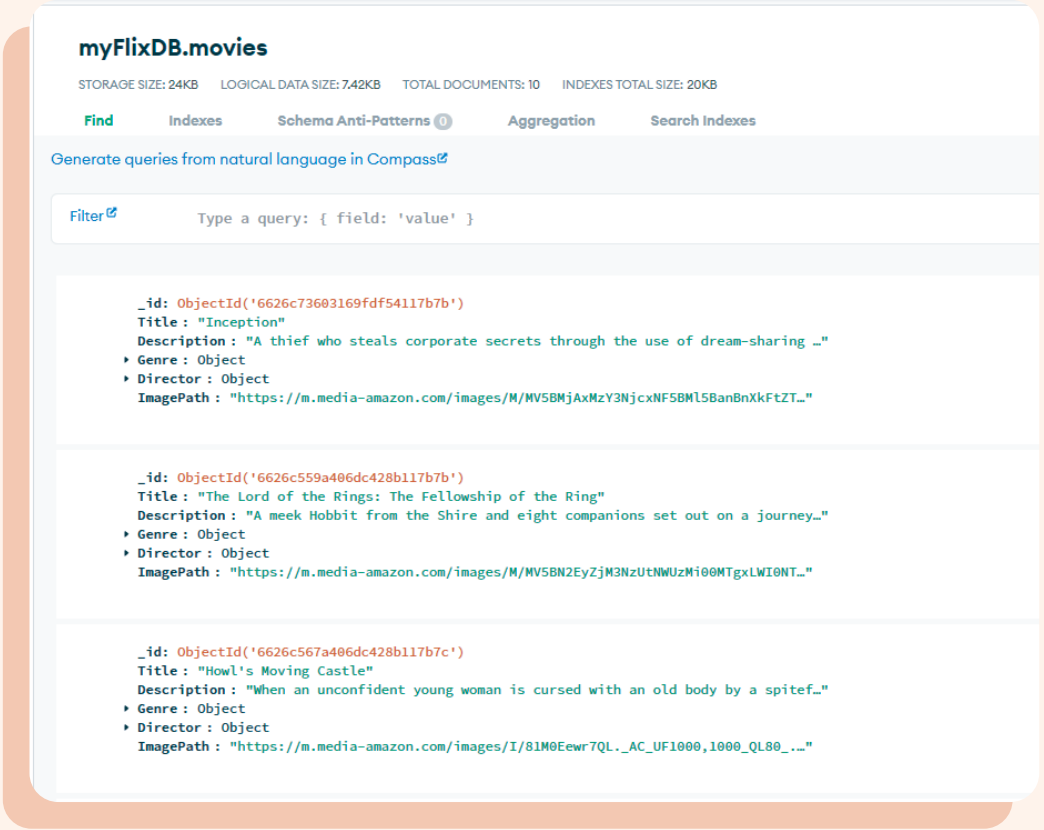
When a query returned null or incomplete data while testing on Postman, it wasn't always clear whether the issue was in the route handler, the query itself, or the underlying database entries.



Overall Solution

I learned to slow down and double-check each step before moving on. Maintaining a clear reference sheet and testing frequently helped catch issues early.

Results & Reflection



The API ultimately served as the foundation for a client-side movie app that consumed its endpoints to display dynamic content. By the end of the project, I had created and tested 10 movie entries with detailed metadata, and 3 users, with the capability to edit the users.

Beyond the technical skills, this experience taught me the value of precision, patience, and iterative testing. Even a simple API requires careful planning and attention to detail to work reliably.

Final Thoughts

What appeared to be a straightforward project was actually a critical learning experience in data modeling, API design, and testing discipline. Completing this API was a milestone in my development journey, and it laid the groundwork for more complex full-stack applications.