

ITE5315 Project

Team: -

Samip Vichare | N01545366

Hardi Shastri | N01521762

Project duration: Week11~13

Project submission:

- Phase 0: Nov 29 during the class, project planning, basic setup
- Phase 1:, Dec 5 submission + Dec 6 class discussion
- Phase 2: Dec 12 (Complete/Submit project + video)
- Phase 3: Dec 13 Project presentation/evaluation

Project Team: Team of two

Assessment Weight: 20% of your final course Grade

Assessment includes:

- Submitting project week by week (deliverables in each week is posted in BB) (8%)
- Final project submission includes all project file + project document + Video recording (10%)
- Project peer evaluation + presentation in Week 13 (2%)

Submission Deadline: End of week13

Objective: To explore more on developing db-driven Node/Express app (REST API)

Description: Using the concepts that we learned in the course and, we are going to develop a secure db-driven Node/Express app.

Project Specification:

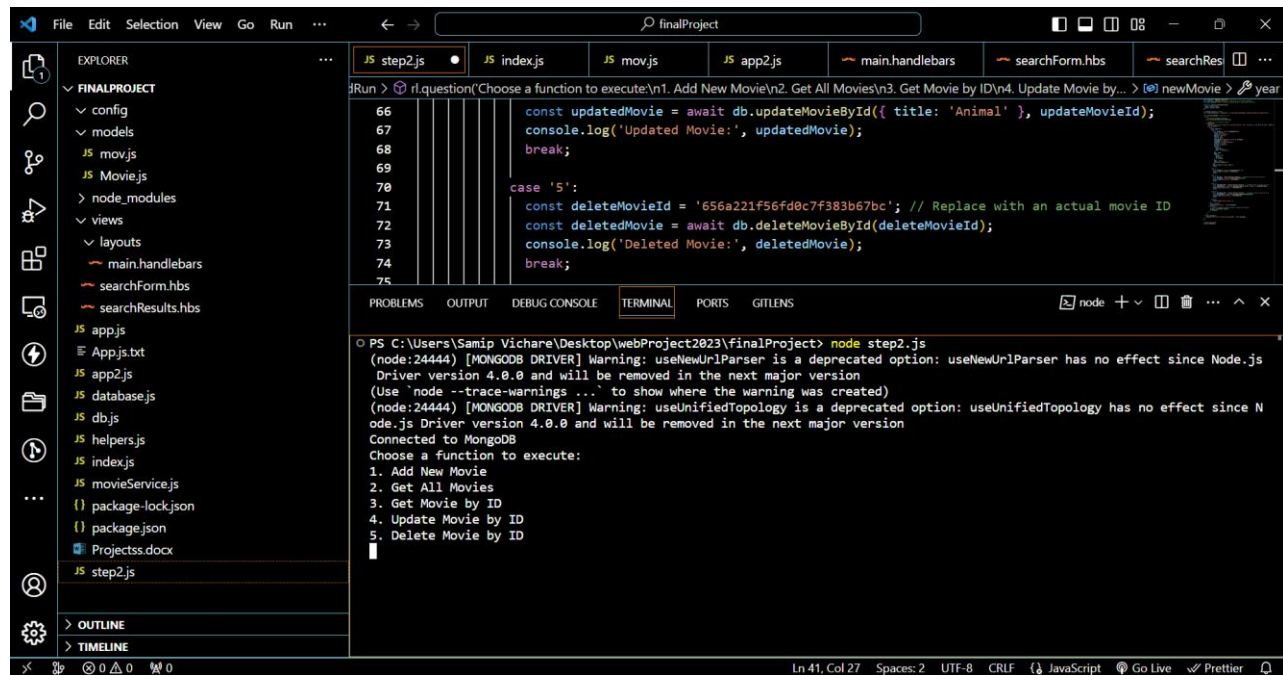
Step 1: Loading the "Sample Movie Data" in MongoDB Atlas

The screenshot displays the MongoDB Compass web interface. The top navigation bar shows the connection name 'admin.2e0txqd....' and the 'Documents' tab for the 'sample_mflix.movies' collection. The left sidebar lists the database structure, with 'sample_mflix' expanded to show collections like 'comments', 'embedded_movies', 'movies', 'sessions', 'theaters', and 'users'. The main panel shows the 'sample_mflix.movies' collection with 21.3k documents and 2 indexes. The 'Documents' tab is active, displaying a list of documents. The first document is expanded, showing its JSON structure:

```
{
  "_id": ObjectId('573a1390f29313caabed42e8'),
  "plot": "A group of bandits stage a brazen train hold-up, only to find a determ...",
  "genres": Array (2),
  "runtime": 11,
  "cast": Array (4),
  "poster": "https://m.media-amazon.com/images/M/MV5BMTU3NjESNzYTYyNS00NDVmLWIwYj_-",
  "title": "The Great Train Robbery",
  "fullplot": "Among the earliest existing films in American cinema - notable as the _",
  "languages": Array (1),
  "released": 1903-12-01T00:00:00.000+00:00,
  "directors": Array (1),
  "rated": "TV-G",
  "awards": Object,
  "lastupdated": "2015-08-13 00:27:59.177000000",
  "year": 1903
}
```

Step 2: Building a Web API

- You need to install express, cors, mongoose + set up Git repository
- Add a module to interact with Movie MongoDB (Similar to assignment 3)
 - "Initializing" the Module before the server starts
- To ensure that we can indeed connect to the MongoDB Atlas cluster with our new connection string, we must invoke the `db.initialize("connection string...")` method and only start the server once it has succeeded, otherwise we should show the error message in the console



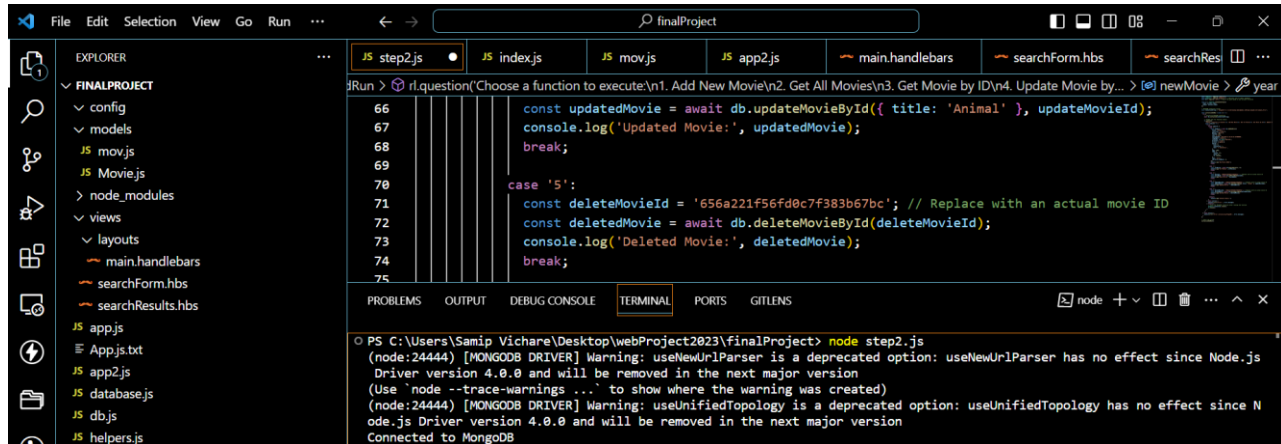
The screenshot shows the Visual Studio Code editor interface. The Explorer panel on the left displays the project structure for 'finalProject', including files like `config`, `models`, `mov.js`, `Movie.js`, `node_modules`, `views`, `layouts`, `main.handlebars`, `searchForm.hbs`, `searchResults.hbs`, `app.js`, `App.js.txt`, `app2.js`, `database.js`, `db.js`, `helpers.js`, `index.js`, `movieService.js`, `package-lock.json`, `package.json`, `Projectss.docx`, and `step2.js`. The main editor area shows the content of `step2.js`, which includes code for updating and deleting movies using MongoDB. The terminal panel at the bottom shows the output of running `node step2.js`, displaying MongoDB driver warnings and a prompt to choose a function to execute.

```
66 const updatedMovie = await db.updateMovieById({ title: 'Animal' }, updateMovieId);
67 console.log('Updated Movie:', updatedMovie);
68 break;
69
70 case '5':
71 const deleteMovieId = '656a221f56fd0c7f383b67bc'; // Replace with an actual movie ID
72 const deletedMovie = await db.deleteMovieById(deleteMovieId);
73 console.log('Deleted Movie:', deletedMovie);
74 break;
75
```

```
PS C:\Users\Samip Vichare\Desktop\webProject2023\finalProject> node step2.js
(node:24444) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js
Driver version 4.0.0 and will be removed in the next major version
(Use 'node --trace-warnings ...' to show where the warning was created)
(node:24444) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since N
ode.js Driver version 4.0.0 and will be removed in the next major version
Connected to MongoDB
Choose a function to execute:
1. Add New Movie
2. Get All Movies
3. Get Movie by ID
4. Update Movie by ID
5. Delete Movie by ID

```

- This module will provide the 6 (promise-based) functions required by our Web API for this particular dataset
- `db.initialize("Your MongoDB Connection String Goes Here")`: Establish a connection with the MongoDB server and initialize the "Movie" model with the "Movie" collection (used above)



```

66 const updatedMovie = await db.updateMovieById({ title: 'Animal' }, updateMovieId);
67 console.log('Updated Movie:', updatedMovie);
68 break;
69
70 case '5':
71   const deleteMovieId = '656a221f56fd0c7f383b67bc'; // Replace with an actual movie ID
72   const deletedMovie = await db.deleteMovieById(deleteMovieId);
73   console.log('Deleted Movie:', deletedMovie);
74   break;
75

```

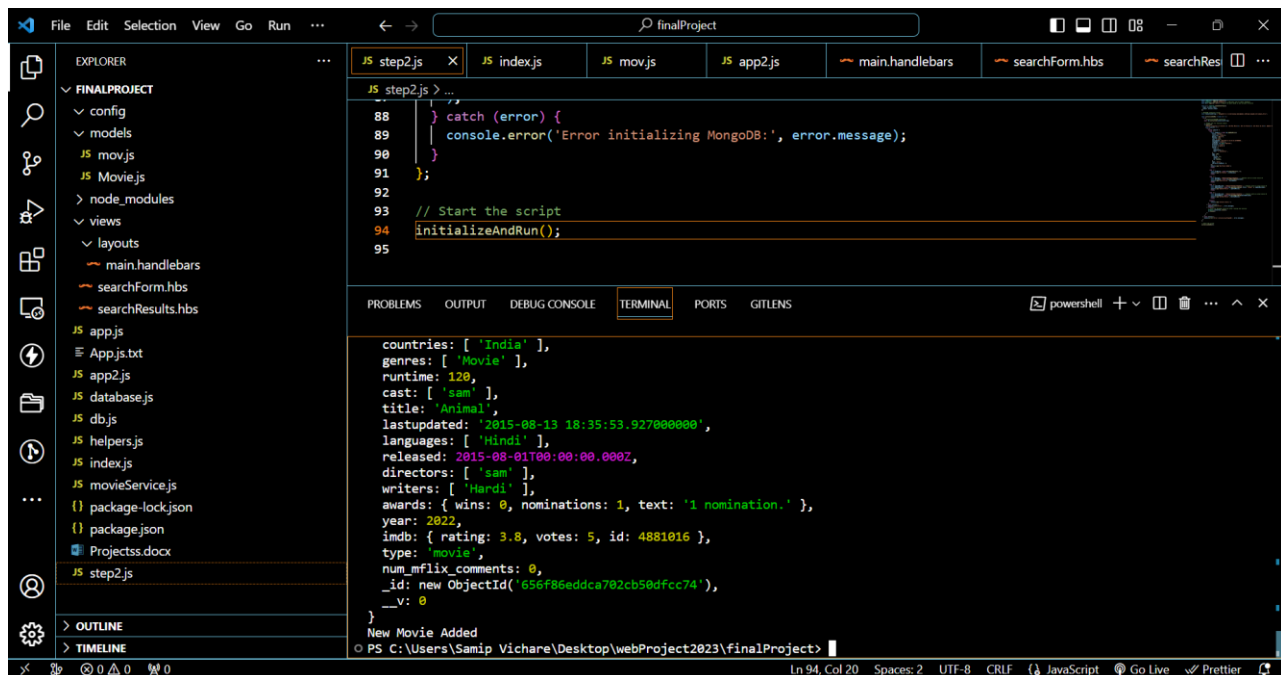
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```

PS C:\Users\Samip Vichare\Desktop\webProject2023\finalProject> node step2.js
(node:24444) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js
Driver version 4.0.0 and will be removed in the next major version
(Use 'node --trace-warnings ...' to show where the warning was created)
(node:24444) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since N
ode.js Driver version 4.0.0 and will be removed in the next major version
Connected to MongoDB

```

- `db.addNewMovie(data)`: Create a new Movie in the collection using the object passed in the "data" parameter



```

88   } catch (error) {
89     console.error('Error initializing MongoDB:', error.message);
90   }
91 }
92
93 // Start the script
94 initializeAndRun();
95

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

```

countries: [ 'India' ],
genres: [ 'Movie' ],
runtime: 120,
cast: [ 'sam' ],
title: 'Animal',
lastupdated: '2015-08-13 18:35:53.927000000',
languages: [ 'Hindi' ],
released: 2015-08-01T00:00:00.000Z,
directors: [ 'sam' ],
writers: [ 'Hardi' ],
awards: { wins: 0, nominations: 1, text: '1 nomination.' },
year: 2022,
imdb: { rating: 3.8, votes: 5, id: 4881016 },
type: 'movie',
num_mflix_comments: 0,
_id: new ObjectId('656f86eddc702cb50dfcc74'),
_v: 0
}
New Movie Added
PS C:\Users\Samip Vichare\Desktop\webProject2023\finalProject>

```

- `db.getAllMovies(page, perPage, title)`: Return an array of all Movies for a specific page (sorted by `Movie_id`), given the number of items per page. For example, if page is 2 and `perPage` is 5, then this function would return a sorted list of Movies (by `Movie_id`), containing items 6 – 10. This will help us to deal with the large amount of data in this dataset and make paging easier to implement in the UI later. Additionally, there is an optional parameter "title" that can be used to filter results by a specific "title" value

The screenshot shows the VS Code editor with the `index.js` file open. The function `getAllMovies` is implemented, which uses `mongoose.connection.db.collection('movies')` to query the database. The terminal output shows the function being called with `2` for page and `5` for perPage, resulting in an array of 5 movie objects.

```

75 // Implement the getAllMovies function
76 function getAllMovies(page, perPage, title) {
77   // Default values
78   const defaultPerPage = 10;
79   const defaultPage = 1;
80   // Validate inputs
81   if (!page || !perPage || !title) {
82     console.error('Invalid choice. Please provide valid inputs for page, perPage, and title.');
83     return;
84   }
85   // Close the MongoDB connection after running the function
86   mongoose.connection.close();
87   r1.close();
88 }
89
90 // Call the function
91 getAllMovies(2, 5, 'A group of handits stage a brazen train hold-up, only to find a determined posse hot on their heels.');
```

```

2. Get All Movies
3. Get Movie by ID
4. Update Movie by ID
5. Delete Movie by ID
2
All Movies: [
  {
    awards: { wins: 1, nominations: 0, text: '1 win.' },
    imdb: { rating: 7.4, votes: 9847, id: 439 },
    writers: [],
    _id: new ObjectId('573a1390f29313caabed42e8'),
    plot: 'A group of handits stage a brazen train hold-up, only to find a determined posse hot on their heels.',
    genres: [ 'Short', 'Western' ],
    runtime: 11,
    cast: [
      'A.C. Abadie',
      'Gilbert M. 'Broncho Billy' Anderson',
      'George Barnes',
      'Justus D. Barnes'
    ]
  }
]
```

- `db.getMovieById(Id)`: Return a single Movie object whose "`_id`" value matches the "Id" parameter

The screenshot shows the VS Code editor with the `index.js` file open. The function `getMovieById` is implemented, which uses `mongoose.connection.db.collection('movies')` to query the database. The terminal output shows the function being called with `4881016` for the ID, resulting in a single movie object.

```

50 // Implement the getMovieById function
51 function getMovieById(Id) {
52   // Default values
53   const defaultId = '4881016';
54   // Validate inputs
55   if (!Id) {
56     console.error('Invalid choice. Please provide a valid ID for the movie.');
57     return;
58   }
59   // Close the MongoDB connection after running the function
60   mongoose.connection.close();
61   r1.close();
62 }
63
64 // Call the function
65 getMovieById(4881016);
```

```

4. Update Movie by ID
5. Delete Movie by ID
3
Movie by ID: {
  awards: { wins: 0, nominations: 1, text: '1 nomination.' },
  imdb: { rating: 3.8, votes: 5, id: 4881016 },
  _id: new ObjectId('636f86eddc702cb50dfcc74'),
  countries: [ 'India' ],
  genres: [ 'Movie' ],
  runtime: 120,
  cast: [ 'sam' ],
  title: 'Animal',
  lastupdated: '2015-08-13 18:35:53.927000000',
  languages: [ 'Hindi' ],
  released: '2015-08-01T00:00:00.000Z',
  directors: [ 'sam' ],
  writers: [ 'Hardi' ],
  year: 2022,
  type: 'movie',
  num_mflix_comments: 0,
  _v: 0
}
```

- `updateMovieById(data,id)`: Overwrite an existing Movie whose `"_id"` value matches the `"id"` parameter, using the object passed in the `"data"` parameter.

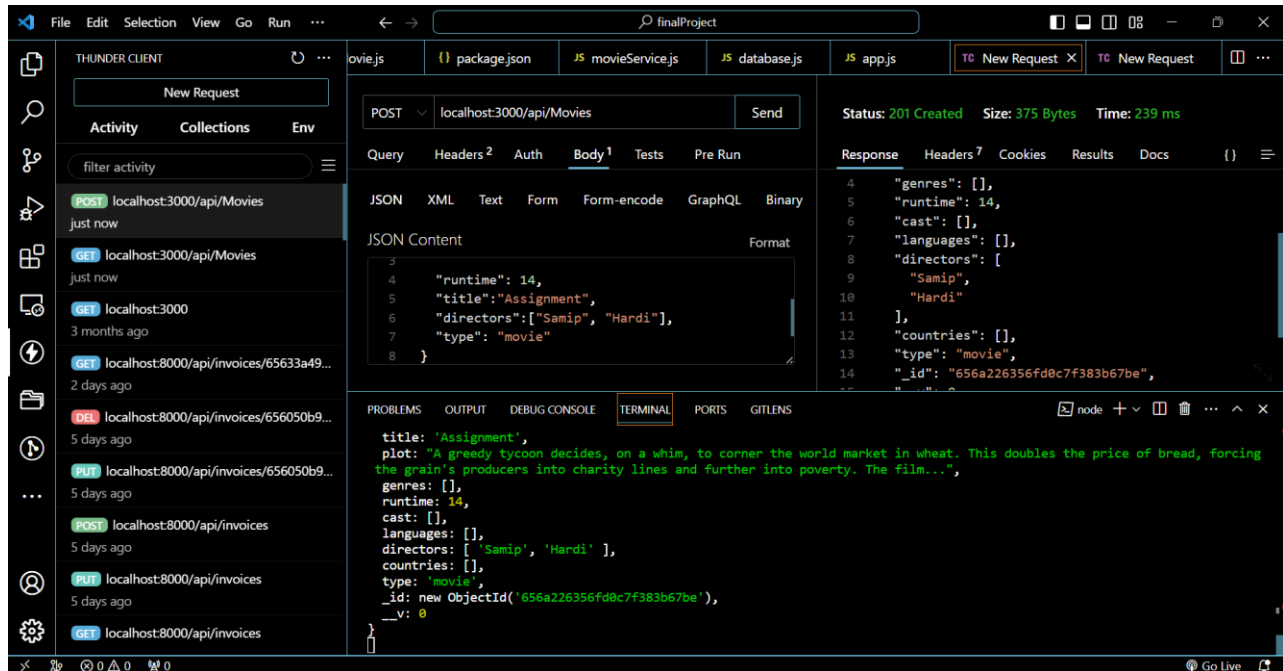
- deleteMovieById(id): Delete an existing Movie whose "_id" value matches the "id" parameter

The image shows a Visual Studio Code editor window with the following components:

- File Explorer (Left Sidebar):** Displays the project structure under 'FINALPROJECT'. Files include 'config', 'models', 'mov.js', 'Movie.js', 'node_modules', 'views', 'layouts', 'main.handlebars', 'searchForm.hbs', 'searchResults.hbs', 'app.js', 'App.js.txt', 'app2.js', 'database.js', 'db.js', 'helpers.js', 'index.js', 'movieService.js', 'package-lock.json', 'package.json', 'Projectss.docx', and 'step2.js' (selected).
- Code Editor (Main Area):** Shows the content of 'step2.js'. The code defines a function to add a new movie to a database. It includes a comment about the function's purpose and a console log for the new movie.
- Terminal (Bottom):** Displays the output of the code execution, showing the details of a deleted movie, including awards, IMDb rating, ID, countries, genres, runtime, cast, title, last updated time, languages, release date, directors, writers, and year.
- Status Bar (Bottom):** Shows the current file path: 'PS C:\Users\Samip Vichare\Desktop\webProject2023\finalProject>'.

Add the routes : The next piece that needs to be completed before we have a functioning Web API is to actually define the routes (listed Below). Note: Do not forget to return an error message if there was a problem and make use of the status codes 201, 204 and 500 where applicable.

- POST /api/Movies
 - This route uses the body of the request to add a new "Movie" document to the collection and return the created object / fail message to the client.



```

{
  "plot": "A greedy tycoon decides, on a whim, to corner the world market in wheat. This
doubles the price of bread, forcing the grain's producers into charity lines and further into
poverty. The film...",

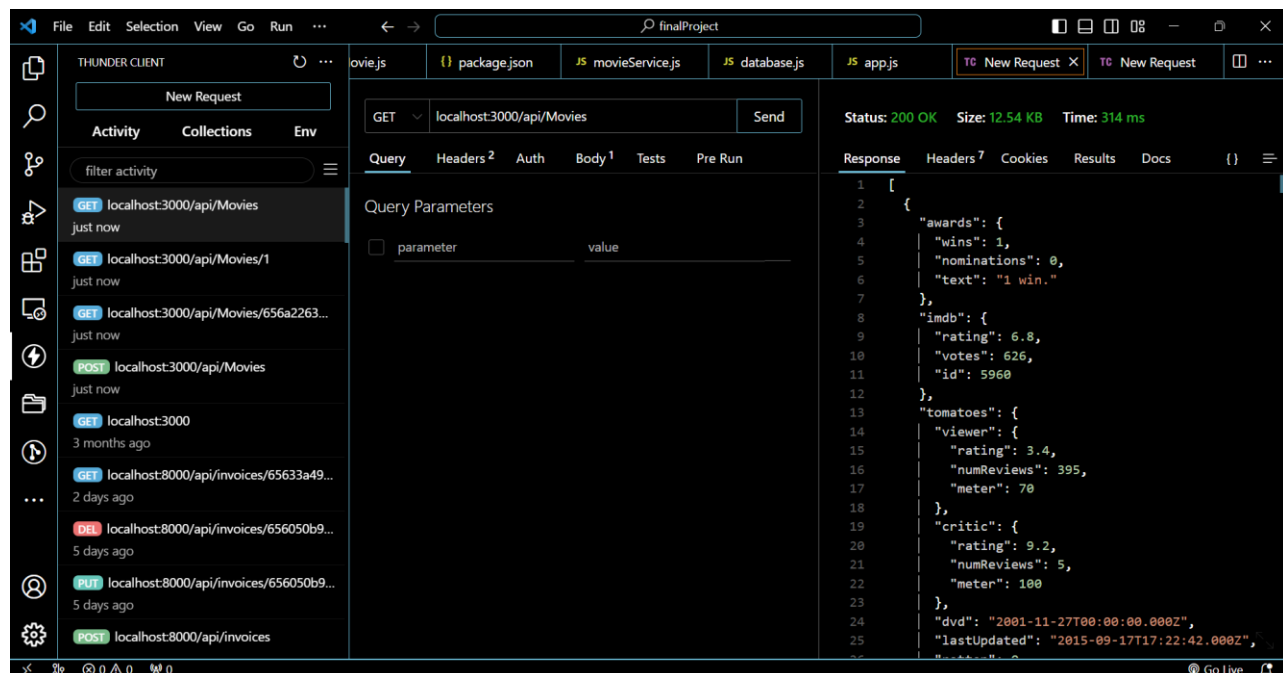
```

```

  "runtime": 14,
  "title": "Assignment",
  "directors": ["Samip", "Hardi"],
  "type": "movie"
}

```

- GET /api/Movies
 - This route must accept the numeric query parameters "page" and "perPage" as well as the (optional) string parameter "title", ie: /api/movies?page=1&perPage=5&title=The Avengers. It will use these values to return all "Movie" objects for a specific "page" to the client as well as optionally filtering by "title", if provided (in this case, it will show both "The Avengers" films).



- EXTRA CHALLENGE (bonus): add query param validation to your route in order to make sure that the params you expect are present, and of the type you expect. You can do this using packages like <https://www.npmjs.com/package/celebrate> or <https://express-validator.github.io/docs/check-api.html> . If the params are incorrect, your route should return a 400 response (client error) vs. 500 (server error).

- GET /api/Movies
 - This route must accept a route parameter that represents the `_id` of the desired movie object, ie: `/api/movies/573a1391f29313caabcd956e`. It will use this parameter to return a specific "Movie" object to the client.

The screenshot displays the Thunder Client interface with a REST client request and its response. The request is a GET call to `localhost:3000/api/Movies/656a226356fd0c7f383b67`. The response is a JSON object representing a movie.

Request Details:

- Method: GET
- URL: `localhost:3000/api/Movies/656a226356fd0c7f383b67`
- Status: 200 OK
- Size: 375 Bytes
- Time: 307 ms

Response Details:

```
1 {
2   "_id": "656a226356fd0c7f383b67be",
3   "title": "Assignment",
4   "plot": "A greedy tycoon decides, on a whim, to
5     corner the world market in wheat. This
6     doubles the price of bread, forcing the
7     grain's producers into charity lines and
8     further into poverty. The film...",
9   "genres": [],
10  "runtime": 14,
11  "cast": [],
12  "languages": [],
13  "directors": [
14    "Samip",
15    "Hardi"
16  ],
17  "countries": [],
18}
```

Terminal Output:

```
Server is running on port 3000
PS C:\Users\Samip Vichare\Desktop\WebProject2023\finalProject> node app.js
Connected to MongoDB Atlas
Server is running on port 3000
```

- PUT /api/Movies
 - This route must accept a route parameter that represents the `_id` of the desired movie object, ie: `/api/movies/573a1391f29313caabcd956e`. It will use this parameter to return a specific "Movie" object to the client.

The screenshot shows the Thunder Client interface. On the left, a list of requests is visible. The main panel shows a PUT request to `localhost:3000/api/Movies/656a226356fd0c7f383b67be` with a status of 200 OK. The response body is a JSON object:

```
{
  "_id": "656a226356fd0c7f383b67be",
  "title": "Assignment3",
  "plot": "A greedy tycoon",
  "genres": [],
  "runtime": 15,
  "cast": [],
  "languages": [],
  "directors": [
    "Samip"
  ],
  "countries": [],
  "type": "movies",
  "_v": 0
}
```

The terminal at the bottom shows the command `node app.js` and the output indicating the server is running on port 3000 and the data was updated successfully.

- DELETE /api/Movies
 - This route must accept a route parameter that represents the `_id` of the desired movie object, ie: `/api/movies/573a1391f29313caabcd956e` as well as read the contents of the request body. It will use these values to update a specific "Movie" document in the collection and return a success / fail message to the client.

The screenshot shows the Thunder Client interface. On the left, a list of requests is visible. The main panel shows a DELETE request to `localhost:3000/api/Movies/656a226356fd0c7f383b67be` with a status of 200 OK. The response body is a JSON object:

```
{
  "message": "Movie deleted successfully"
}
```

The terminal at the bottom shows the command `node app.js` and the output indicating the server is running on port 3000 and the data was updated successfully.

Step 3: Add UI/Form

You want to demonstrate your skill in working with Template Engines and Form, but you don't want to apply this for the entire application.

Try to make a new route which works similar to `"/api/Movies?page=1&perPage=5&title= The Avengers"` and take the 'page', 'perPage' and 'title' through FORM and display the output using Template Engine!

Use your creativity to design the layout and apply proper css style/format.

[Home](#) [Register](#) [Login](#) [Logout](#)

Search Movies

Page:

Per Page:

Title:

[Home](#) [Register](#) [Login](#) [Logout](#)

Search Results

The Avengers

Plot: Two British agents (John Steed and Emma Peel) team up to stop Sir August De Wynter from destroying the world with a weather changing machine.

Genres: Action, Adventure, Sci-Fi

Runtime: 89 minutes

Directors: Jeremiah S. Chechik

IMDb Rating: 3.7 (34077 votes)

Cast: Ralph Fiennes, Uma Thurman, Sean Connery, Patrick Macnee

Released:

Languages: English

Rated: PG-13

Awards: 1 win & 11 nominations.

Year: 1998

Country: USA



The Avengers

Plot: Earth's mightiest heroes must come together and learn to fight as a team if they are to stop the mischievous Loki and his alien army from enslaving humanity.

Genres: Action, Adventure, Sci-Fi

Runtime: 143 minutes

Directors: Joss Whedon

Step 4 : Add security feature to the app:

- A) Use Environment Variable for your Connection String:** Your solution currently has your database connection string (with username and password!) hard coded into your source code. This is a potential security risk. If this code is shared between users on a team, or if it is pushed to a public repo on GitHub, your password is now public too.
- B) The app gives access to Movie data. How do you limit the user-access, so that only authorized users can open the app or access to specific route?**
 - a. You are asked to use the security features like Password Encryption, JWT, Session, Cookie that we have learned in the course in order to allow only authorized user to use some of the special routes! You may also think about adding "API-key" to the route (like "openweathermap") for the authorized user
 - b. The goal is encrypt sensitive data in DB and use JWT(full mark) to secure routes for authorized users (users can be designed locally in app or from a DB collection). However, you can plan other simpler implementations and still get full mark depends to your strategy.

Register

Name:

Email:

Password:

[Home](#) [Register](#) [Login](#) [Logout](#)

Login

Email:

Password:

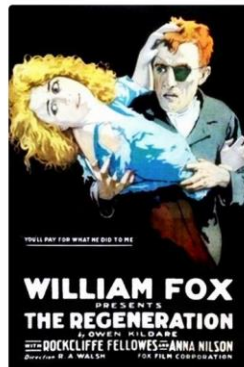
Step 5 : Add new functionality/feature to the app

You can use your creativity to add a new functionality to the app. This can be adding a new UI/route/DB operation, or using a new npm package that enhances your design.

Printing top 10 movies on the dashboard based on imdb rating.

Welcome To Final Project

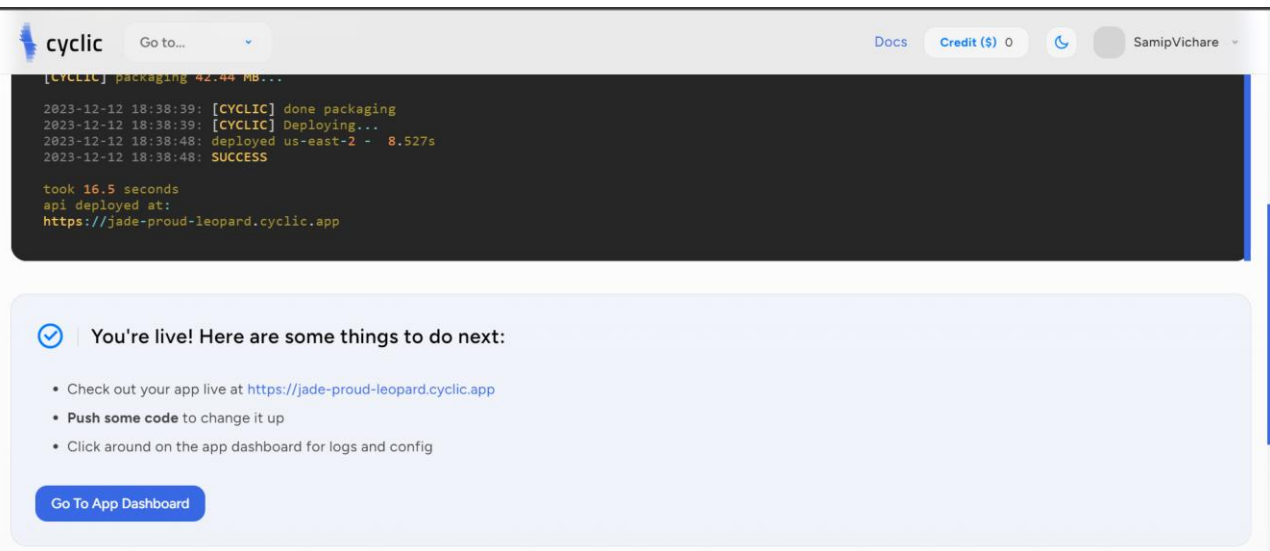
Top 10 Movies on IMDB



Regeneration

Step 6: Pushing to Cyclic

Once you are satisfied with your application, deploy it to Cyclic



<https://jade-proud-leopard.cyclic.app/>

Bonus :

- **(additional 5%):** Add the GraphQL to the app in order to extract and manipulate Movie data. This can be applied into the data extraction or even data manipulation routes.
- **(additional 2%):** Design a JS front-end (jQuery , Native JS, React, ...) to utilize the API that you designed and use your creativity to add some bootstrap flavor to that 😊

Project expectation:

- Following best practices in handling async tasks
- Following best practices in error handling
- Practice good way of structuring NodeJs app and separate functionalities/layers
- Practice good way of commenting codes
- I am open to utilizing new techniques/packages as far as you are able to understand and explain it. However, I would prefer to see you use the practices that we did in other activities during this course.

Project Submission:

- Add the following declaration at the top of .js files

```

/*****
***
* ITE5315 – Project
* I declare that this assignment is my own work in accordance with Humber Academic Policy.
* No part of this assignment has been copied manually or electronically from any other source
* (including web sites) or distributed to other students.

```

*

* Group member Name:_____Student IDs:_____Date: _____

***/

- Compress (.zip) the files in your Visual Studio working directory (this is the folder that you opened in Visual Studio to create your client side code).
- Complete & Submit the project document (given template).
- Record a detailed walk-through/demonstration video presentation of the project

Important Note:

- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.
- **LATE SUBMISSIONS for assignments.** There is a deduction of 1% per hour for Late submissions, and after two days it will grade of zero (0).
- Assignments should be submitted along with a video-recording which contains a detailed walkthrough of solution. Without recording, the assignment can get the maximum of 1/3 of the total.