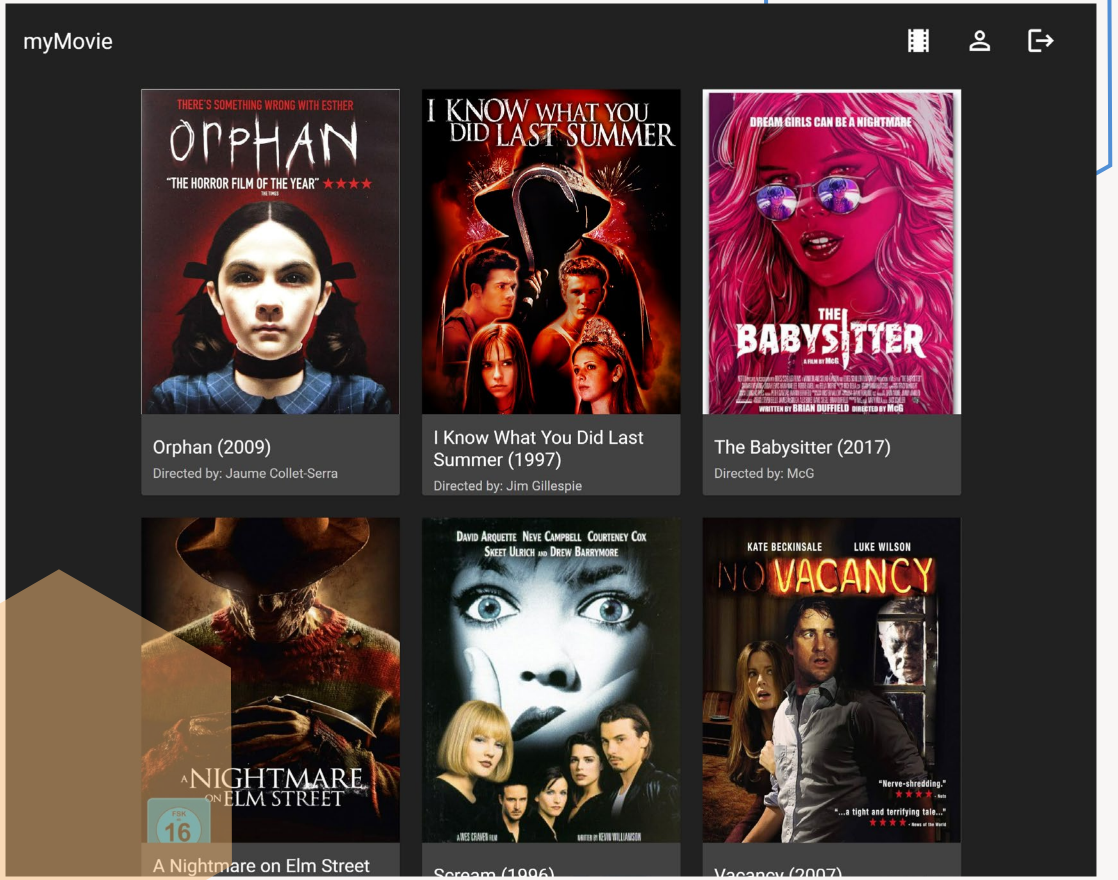
Case Study – Movie API

Isabel Matula



myMovie

Orphan (2009)
Directed by: Jaume Collet-Serra

I Know What You Did Last Summer (1997)
Directed by: Jim Gillespie

The Babysitter (2017)
Directed by: McG

A Nightmare on Elm Street

Overview

Objective

The aim of the project was to develop an API which provides users with access to information about different movies, directors, and genres. Users can sign up and create a list of their favorite movies.

Tools

This project utilized MongoDB, Express, and Node.js as well as Postman for testing the endpoints of the API. For authentication purposes, I've used JWT (JSON Web Token).

Duration

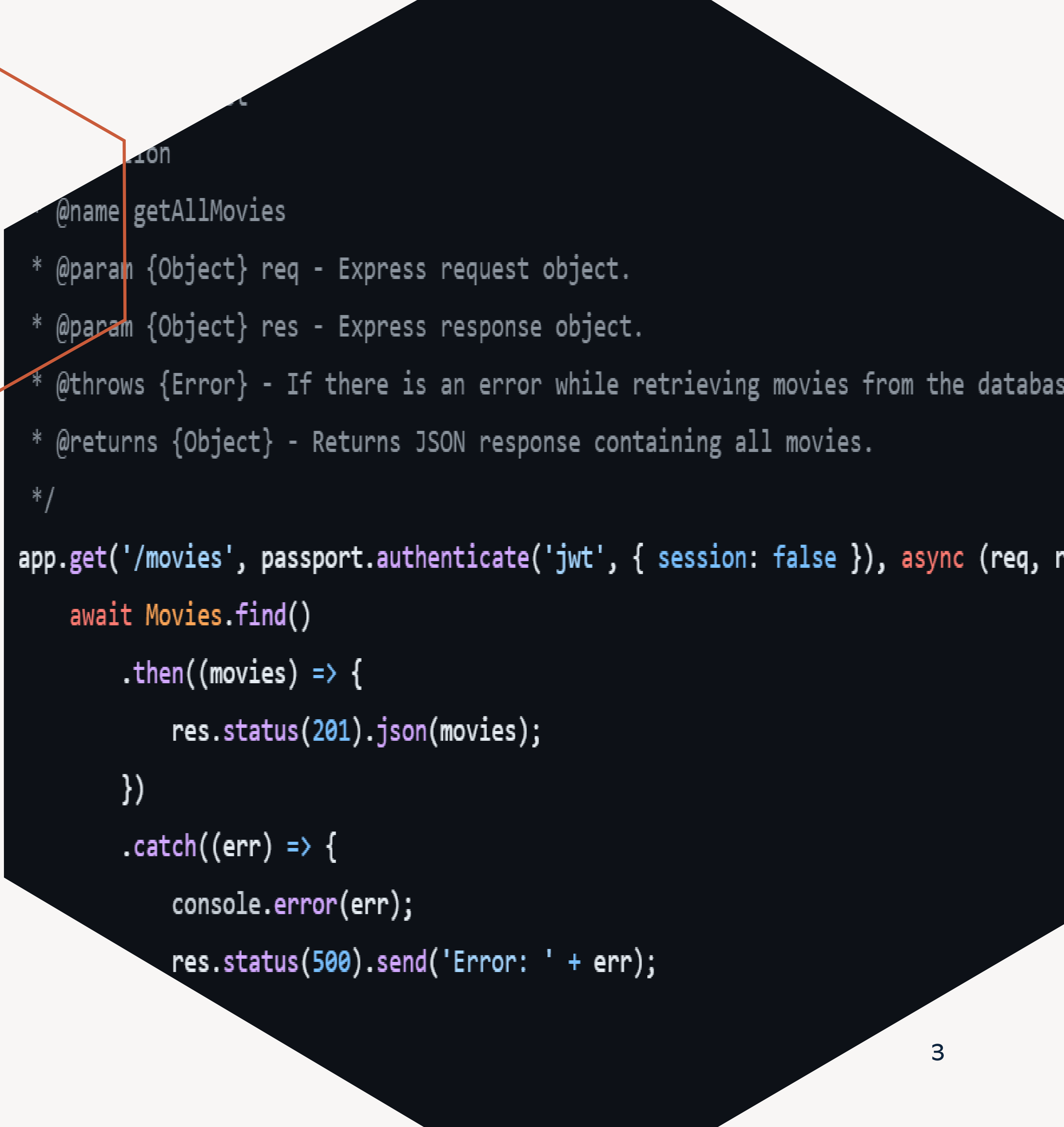
The project was completed within a timeframe of two weeks.

Purpose and Context

I created a REST API for an application called “myMovie" that interacted with a database storing movie data. Afterwards, I built the client-side component using React. Resulting in a full-stack web application demonstrating the use of the MERN stack, including APIs, web servers, databases, authentication, and more.

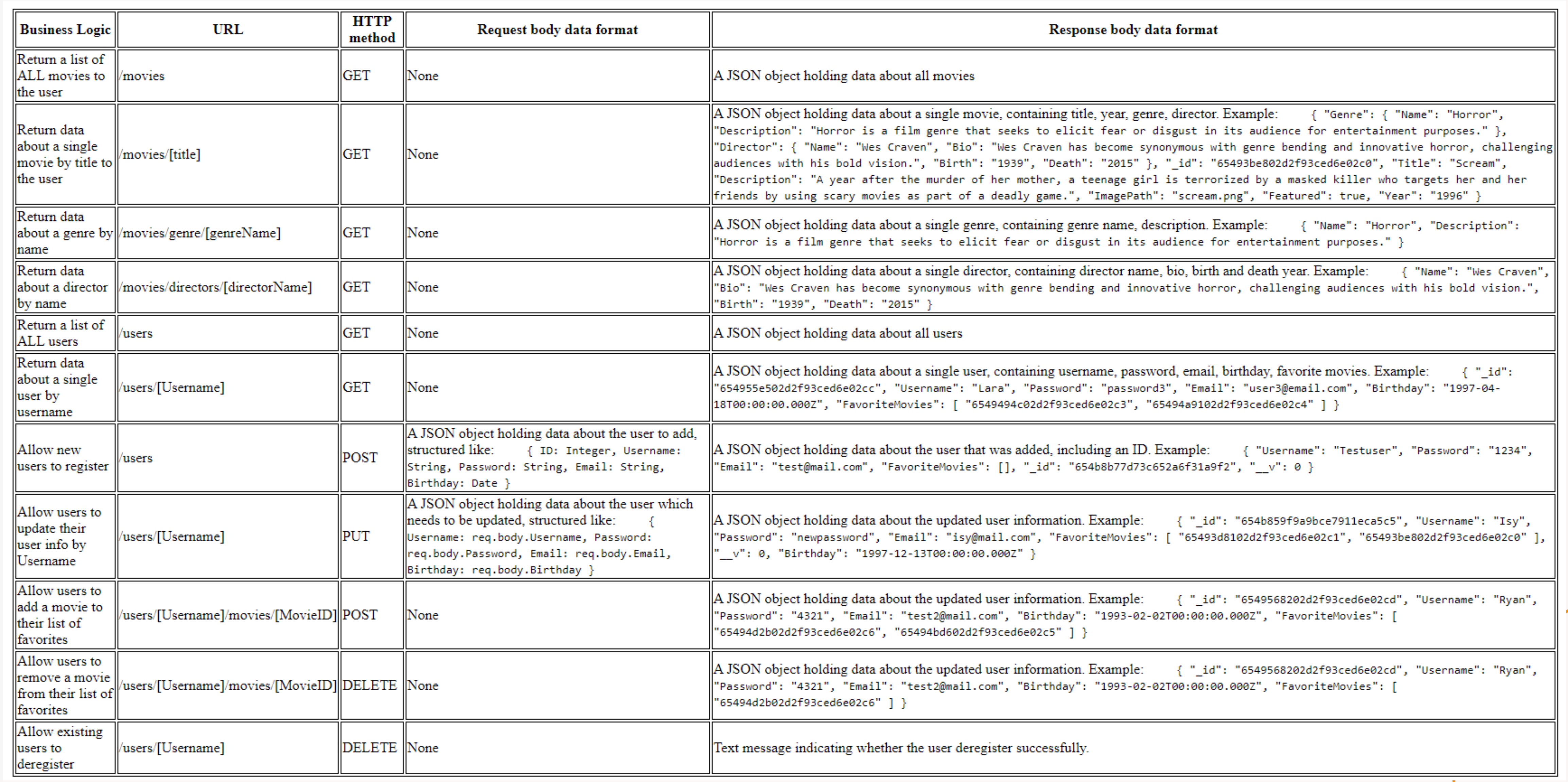
REST API

A REST API is a web service built on RESTful architecture. It uses HTTP methods like GET, POST, PUT, and DELETE for CRUD operations, and URLs to represent resources. For example, HTTP endpoints were defined to access collections like 'movies' and 'users' for tasks such as retrieving user info or updating movie details. These endpoints were tested using Postman.



```
           ion
 * @name getAllMovies
 * @param {Object} req - Express request object.
 * @param {Object} res - Express response object.
 * @throws {Error} - If there is an error while retrieving movies from the databas
 * @returns {Object} - Returns JSON response containing all movies.
 */
app.get('/movies', passport.authenticate('jwt', { session: false }), async (req, r
    await Movies.find()
        .then((movies) => {
            res.status(201).json(movies);
        })
        .catch((err) => {
            console.error(err);
            res.status(500).send('Error: ' + err);
```

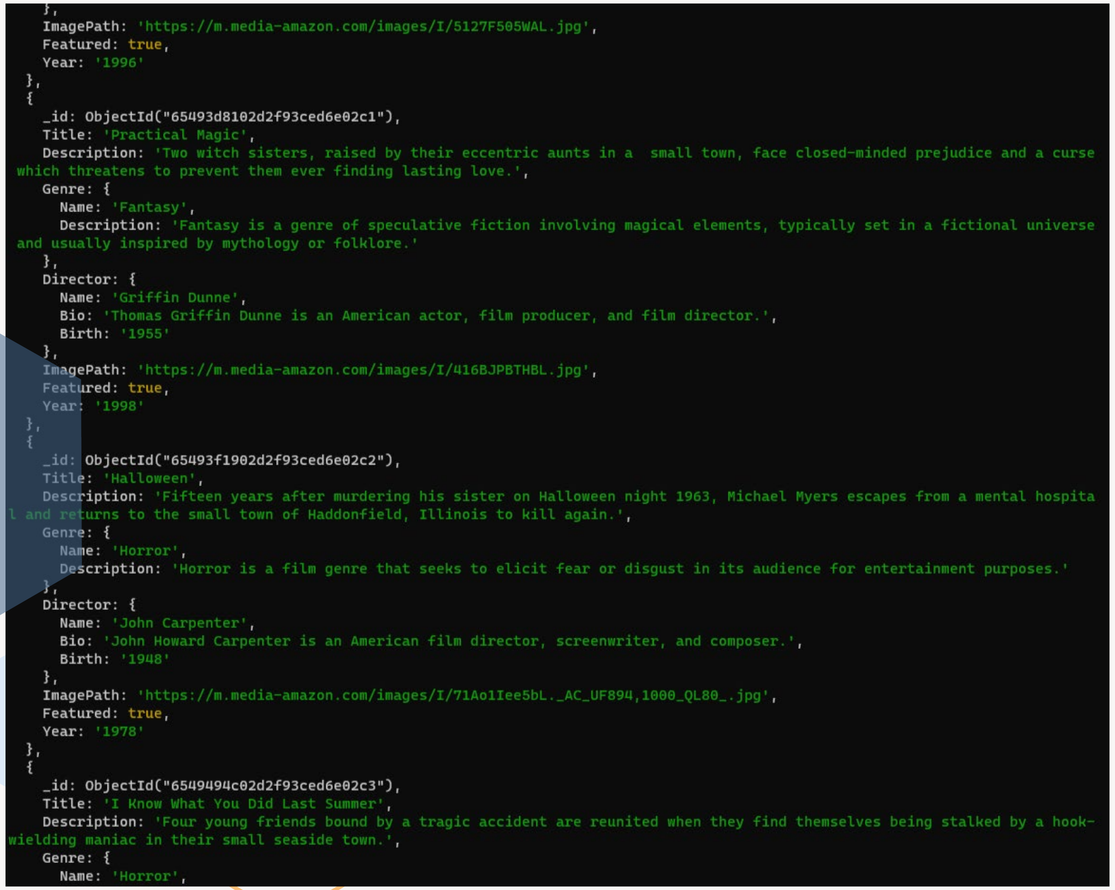
Endpoints



Business Logic	URL	HTTP method	Request body data format	Response body data format
Return a list of ALL movies to the user	/movies	GET	None	A JSON object holding data about all movies
Return data about a single movie by title to the user	/movies/[title]	GET	None	A JSON object holding data about a single movie, containing title, year, genre, director. Example: { "Genre": { "Name": "Horror", "Description": "Horror is a film genre that seeks to elicit fear or disgust in its audience for entertainment purposes." }, "Director": { "Name": "Wes Craven", "Bio": "Wes Craven has become synonymous with genre bending and innovative horror, challenging audiences with his bold vision.", "Birth": "1939", "Death": "2015" }, "_id": "65493be802d2f93ced6e02c0", "Title": "Scream", "Description": "A year after the murder of her mother, a teenage girl is terrorized by a masked killer who targets her and her friends by using scary movies as part of a deadly game.", "ImagePath": "scream.png", "Featured": true, "Year": "1996" }
Return data about a genre by name	/movies/genre/[genreName]	GET	None	A JSON object holding data about a single genre, containing genre name, description. Example: { "Name": "Horror", "Description": "Horror is a film genre that seeks to elicit fear or disgust in its audience for entertainment purposes." }
Return data about a director by name	/movies/directors/[directorName]	GET	None	A JSON object holding data about a single director, containing director name, bio, birth and death year. Example: { "Name": "Wes Craven", "Bio": "Wes Craven has become synonymous with genre bending and innovative horror, challenging audiences with his bold vision.", "Birth": "1939", "Death": "2015" }
Return a list of ALL users	/users	GET	None	A JSON object holding data about all users
Return data about a single user by username	/users/[Username]	GET	None	A JSON object holding data about a single user, containing username, password, email, birthday, favorite movies. Example: { "_id": "654955e502d2f93ced6e02cc", "Username": "Lara", "Password": "password3", "Email": "user3@email.com", "Birthday": "1997-04-18T00:00:00.000Z", "FavoriteMovies": ["6549494c02d2f93ced6e02c3", "65494a9102d2f93ced6e02c4"] }
Allow new users to register	/users	POST	A JSON object holding data about the user to add, structured like: { ID: Integer, Username: String, Password: String, Email: String, Birthday: Date }	A JSON object holding data about the user that was added, including an ID. Example: { "Username": "Testuser", "Password": "1234", "Email": "test@mail.com", "FavoriteMovies": [], "_id": "654b8b77d73c652a6f31a9f2", "__v": 0 }
Allow users to update their user info by Username	/users/[Username]	PUT	A JSON object holding data about the user which needs to be updated, structured like: { Username: req.body.Username, Password: req.body.Password, Email: req.body.Email, Birthday: req.body.Birthday }	A JSON object holding data about the updated user information. Example: { "_id": "654b859f9a9bce7911eca5c5", "Username": "Isy", "Password": "newpassword", "Email": "isy@mail.com", "FavoriteMovies": ["65493d8102d2f93ced6e02c1", "65493be802d2f93ced6e02c0"], "__v": 0, "Birthday": "1997-12-13T00:00:00.000Z" }
Allow users to add a movie to their list of favorites	/users/[Username]/movies/[MovieID]	POST	None	A JSON object holding data about the updated user information. Example: { "_id": "6549568202d2f93ced6e02cd", "Username": "Ryan", "Password": "4321", "Email": "test2@mail.com", "Birthday": "1993-02-02T00:00:00.000Z", "FavoriteMovies": ["65494d2b02d2f93ced6e02c6", "65494bd602d2f93ced6e02c5"] }
Allow users to remove a movie from their list of favorites	/users/[Username]/movies/[MovieID]	DELETE	None	A JSON object holding data about the updated user information. Example: { "_id": "6549568202d2f93ced6e02cd", "Username": "Ryan", "Password": "4321", "Email": "test2@mail.com", "Birthday": "1993-02-02T00:00:00.000Z", "FavoriteMovies": ["65494d2b02d2f93ced6e02c6"] }
Allow existing users to deregister	/users/[Username]	DELETE	None	Text message indicating whether the user deregister successfully.

Create Non-Relational Database

MongoDB organizes data into collections of documents. Each collection represents a type of data entity, and each document represents an instance of that entity. MongoDB's dynamic schema allows for flexible data storage, where each document can have different attributes. This means you can easily add or remove data without changing the schema. For my movie app, I created two collections: "users," where each document represented a user (with fields like username and birthday), and "movies," where each document represented a movie (with fields like title and genre).



```
    },
    ImagePath: 'https://m.media-amazon.com/images/I/5127F505WAL.jpg',
    Featured: true,
    Year: '1996'
  },
  {
    _id: ObjectId("65493d8102d2f93ced6e02c1"),
    Title: 'Practical Magic',
    Description: 'Two witch sisters, raised by their eccentric aunts in a  small town, face closed-minded prejudice and a curse
 which threatens to prevent them ever finding lasting love.',
    Genre: {
      Name: 'Fantasy',
      Description: 'Fantasy is a genre of speculative fiction involving magical elements, typically set in a fictional universe
 and usually inspired by mythology or folklore.'
    },
    Director: {
      Name: 'Griffin Dunne',
      Bio: 'Thomas Griffin Dunne is an American actor, film producer, and film director.',
      Birth: '1955'
    },
    ImagePath: 'https://m.media-amazon.com/images/I/416BJPBTHBL.jpg',
    Featured: true,
    Year: '1998'
  },
  {
    _id: ObjectId("65493f1902d2f93ced6e02c2"),
    Title: 'Halloween',
    Description: 'Fifteen years after murdering his sister on Halloween night 1963, Michael Myers escapes from a mental hospita
l and returns to the small town of Haddonfield, Illinois to kill again.',
    Genre: {
      Name: 'Horror',
      Description: 'Horror is a film genre that seeks to elicit fear or disgust in its audience for entertainment purposes.'
    },
    Director: {
      Name: 'John Carpenter',
      Bio: 'John Howard Carpenter is an American film director, screenwriter, and composer.',
      Birth: '1948'
    },
    ImagePath: 'https://m.media-amazon.com/images/I/71Ao1Iee5bL._AC_UF894,1000_QL80_.jpg',
    Featured: true,
    Year: '1978'
  },
  {
    _id: ObjectId("6549494c02d2f93ced6e02c3"),
    Title: 'I Know What You Did Last Summer',
    Description: 'Four young friends bound by a tragic accident are reunited when they find themselves being stalked by a hook-
wielding maniac in their small seaside town.',
    Genre: {
      Name: 'Horror',
```

Authentication & Authorization

Authentication

- Initially, user authentication is handled through basic HTTP authentication. When users log in, they provide a username and password, which will be sent in the header of the HTTP request.
- After successful authentication, the application generates a JWT for the user. Subsequent requests will then be authenticated and authorized using JWT-based authentication.

Authorization

- For authorization the application uses JWT authentication to ensure that only authenticated users with a valid token can access the API endpoints.

Data Security

- Implemented password hashing to ensure that user passwords remain unreadable, also by the database manager.

Conclusion

The initial goal of building an API using MongoDB, Express, and Node.js was successfully achieved. The final product allows users to access information about various movies, directors, and genres, update their profiles, and create a list of favorite movies. The most challenging part of the project was implementing authentication and authorization, as well as ensuring data security by hashing passwords. This process consumed a significant amount of time. However, creating the database with Mongoose was relatively straightforward and enjoyable.

Thorough testing was crucial to confirm that all endpoints functioned correctly. Moving forward, I would dedicate even more time to improving authentication and data security, recognizing their crucial importance. Overall, this project provided valuable insights and reinforced the significance of robust security measures in application development.