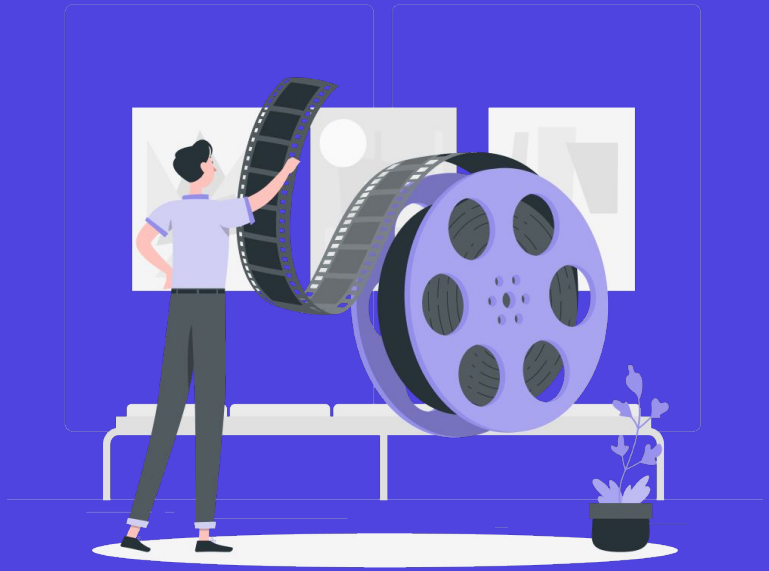
Building the Movie Resource API



Relevel
by Unacademy

Agenda

- What is a movie booking app?
- App overview
- Requirement gathering and actor profiles
- Use cases
- Setting up project structure and database
- Setting up data models for the movie item
- API for CRUD operation on movie resource
- Ability to create, read, update and delete movies.

Movie Booking Application

We are going to create a movie booking application similar to the BookMyShow application. In this application, users can book movies directly from the application and payment can also be done online. This helps in pre-booking of the movie. Users can book movies at any instance of time. Below are the main features of the movie booking application -

- Users can book movies anytime anywhere online.
- Users can view the movies and their show timings
- Cinema owners can inform the users about the new release and discounts and can get feedback from them as well
- Payment can easily be done online without any issue
- Users can pre-plan their schedules to enjoy the movie.

Feature - 1: Authentication and Authorization

- ROOT admin will be registered directly in the DB (i.e. no API endpoint).
- Other System Admin will be registered in the app with the approval of the root admin.
- Clients can register to the system but will require approval from the System Administrator. Clients are the owners of theatre.
- Customers can directly register to the system.
- Login API will return system admin, clients and customers an access token that will be required to access the secure API endpoints (highlighted below)

Feature - 2: Show Movie Halls and Movies

- Setting up the model for the Theatre resource
- Ability to create a new Theater
- Ability to update an existing theater
- Ability to get all the theaters
- Ability to filters theaters based on the pin/city
- Ability to delete an existing theater

Feature - 3: Create/Update/Delete Movie Halls and Movies.

- Add the movies inside a theatre
- Remove the movie inside the theatre
- Get the list of theatres in which a movie is running.
- Search if a movie is running in any theatres
- Search if a movie is running in any specific theatres

Feature - 4: Booking/Updating/Canceling Tickets

- Set up data model for booking and transaction
- Authenticated APIs for allowing authenticated customers to perform booking
- Ability to cancel the booking
- Ability to complete payment within a given time frame

Feature of CRM Application to be developed in this session

- API for getting the list of all movies
- API for getting the movie based on movie ID
- API for getting the movie based on the movie name
- API for deleting the movie
- API for updating the movie

Actor Profiles

- System Admin - Administrator of the entire system
  - CRUD operation on all resources.
  - CRUD operation on Clients.
- Clients - Clients are owners of a movie hall. A client can be the owner of multiple halls.
  - CRUD operation on the theatre owned by them.
- Registered User - Customers visiting the website and have details registered into the system.
  - Browse Movies and Theatres.
  - Book/Update/Cancel tickets.
- Unregistered User - Customers visiting the website and don't have details registered in the system.
  - Browse Movies and Theatres.

Setting up project structure and database

1) Nodejs -

We are going to use the Node.js environment for our application.

JavaScript used to only work inside the browsers. For running JS code, browsers use the JavaScript engine.

NodeJS is a JavaScript runtime environment that allows us to run JS code outside of a web browser. To achieve this, NodeJS uses Chrome's V8 Engine which is open source(free to use) and very performant. A Node.js app runs in a single process, without creating a new thread for every request. NodeJS has a unique advantage because billions of frontend developers that write JavaScript for the browser are now able to write the server-side code in addition to the client-side code without the need to learn a completely different language.

Setting up project structure and database

2) Express Framework -

Express is a Node.js web application framework that provides multiple features to develop web and mobile applications.

Features -

Middleware between request

Routing Concept

Dynamic Operation

Asynchronous Operations

Setting up project structure and database

3) MongoDB -

We will use MongoDB as our database. MongoDB is a document-oriented database that is non-structured.

We can have a dynamic schema that can be scaled easily. It stores data in the form of documents that are in JSON format that are scalable and easier to maintain without keeping some specific constant structure.

4) Mongoose -

We will use mongoose as our object modeling tool. It is used to do object mapping between Nodejs and MongoDB. We can use it as an interface to the database which can be used to create, update, delete and query the records.

Setting up project structure and database

Perform the below steps to setup the code

1. Github repo link - https://github.com/Vishwa07dev/mba_backend/tree/session1
2. Which branch to use - session1
3. How to start the applications -

```
cd mba_backend
npm install
npm run devStart
```

List of the REST APIs involved in this session

- Create a new movie - POST /mba/api/v1/movies/
- Get all the movies - GET /mba/api/v1/movies/
- Update the movie based on movie id - PUT /mba/api/v1/movies/6245ef3fbddfa2ae0d2bba50
- Get the movie based on movie id - GET /mba/api/v1/movies/6245ef3fbddfa2ae0d2bba50
- Get the movie based on movie name - GET /mba/api/v1/movies?name=Jalsa
- Delete the movie - DELETE /mba/api/v1/movies/6245ef3fbddfa2ae0d2bba50

Project Structure

Dependencies to be Installed

- Npm package - After cloning the GitHub repo, use npm install command to install all the dependencies in the code
- MongoDB - MongoDB should be installed in the system to run the application. https://www.mongodb.com/try/download/community - Link to download mongoDB for reference

Folder Structure

- Models - We will write our data models in this folder
- Middlewares - We will write our validation logic here
- Controllers - We will write actual logic in this
- Routes - In this, we will write all the routes of our APIs
- Configs - In this, we will mention all the configuration-related things like the JWT key and so on.
- Utils - In this, we will write our constants to be used in the code

Module Structure

Session 1 Features

- Setting up project structure and database
- Setting up data models for movie item
- API for CRUD operation on movie resource-
- Ability to create, read, update and delete movies.

Session 2 Features

- Setting up the model for the Theatre resource
- Ability to create a new Theater
- Ability to update an existing theater
- Ability to get all the theaters
- Ability to filters theaters based on the pin/city
- Ability to delete an existing theater

Module Structure

Session 3 Features

- Add the movies inside a theatre
- Remove the movie inside the theatre
- Get the list of theatres in which a movie is running.
- Search if a movie is running in any theatres
- Search if a movie is running in any specific theatres

Session 4 Features

- Set up data model for user
- User registration
- Implementation and validation of JWT token
- Login API
- Update password
- Registration of system admin and client

Module Structure

Session 5 Features

- Custom middleware to validate the request body of user registration
- Custom middleware to validate the request body for movie resource
- Custom middleware to validate the request body for theater resource

Session 6 Features

Add authentication in theater APIs

Add authentication and authorization on admin APIs-

1. Admins can Create/Update/Delete any movies.
2. Admins can Create/Update/Delete any movies in any theater
3. Admins can Create/Update/Delete any theaters
4. Admin can update the details of any type of user.

Only authenticated users should be allowed to use any other API

Module Structure

Session 7 Features

- Set up data model for booking and transaction
- Authenticated APIs for allowing authenticated customers to perform booking
- Ability to cancel the booking
- Ability to complete payment within a given time frame

Data Models

- Movie Item
- Name
- Description
- Release Date
- Release Status
- Director
- Language
- Poster URL
- Casts
- Trailer URL

API - Create a New Movie

In this API, we are creating a movie. Before creating the movie, we need to validate the request first. Let’s have a look at all the steps needed one by one -

API - Create a New Movie

Validate the request

[Code Link] - https://github.com/Vishwa07dev/mba_backend/blob/session1/middlewares/verifyMovieReqBody.js

Validate the movie name

Here, we will check if the movie name is provided in the request or not. If not, then we will send an error saying that “Movie name is not provided!”

```
if (!req.body.name) {
        return res.status(400).send({
            message: "Failed! Movie name is not provided !"
        });  }
```

API - Create a New Movie

Validate the movie status

Here, we will check if the movie status is provided in the request or not. If not, then we will send an error saying that “Movie release status is not provided!”

Also, we will check if the status value is correct or not. It should be one from the below values -

- UNRELEASED
- RELEASED
- BLOCKED

API - Create a New Movie

```
 if (!req.body.releaseStatus) {
        return res.status(400).send({
            message: "Failed! Movie release status is not provided !"
        });
    }
const releaseStatus = req.body.releaseStatus;
    const releaseStatusTypes = [constants.releaseStatus.unrealeased,
constants.releaseStatus.released, constants.releaseStatus.blocked];
    if (!releaseStatusTypes.includes(releaseStatus)) {
        return res.status(400).send({
            message: "Movie release status provided is invalid. Possible
values UNRELEASED | RELEASED | BLOCKED "
        });
```

API - Create a New Movie

Validate the movie release date

Here, we will check if the movie release date is provided in the request or not. If not, then we will send an error saying that “Movie release date is not provided!”

```
if (!req.body.releaseDate) {
        return res.status(400).send({
            message: "Failed! Movie release date is not provided !"
        });

    }
```

API - Create a New Movie

Validate the movie director

Here, we will check if the movie director is provided in the request or not. If not, then we will send an error saying that “Movie director is not provided!”

```
if (!req.body.director) {
        return res.status(400).send({
            message: "Failed! Movie director is not provided !"
        });

    }
```

Create the movie

[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session1/controllers/movie.controller.js

Here, we will use the Movie schema of our database and create a movie by giving all the details as an object to create the function of the Movie.
We will give below details to create a movie -

- Movie Name
- Description
- Casts
- Director
- Trailer URL
- Poster URL
- Language
- Release Date
- Release Status

Create the movie

```
exports.createMovie = async (req, res) => {
    const movieObject = {
        name: req.body.name,
        description: req.body.description,
        casts: req.body.casts,
        director: req.body.director,
        trailerUrl: req.body.trailerUrl,
        posterUrl: req.body.posterUrl,
        language: req.body.language,
        releaseDate: req.body.releaseDate,
        releaseSatus: req.body.releaseSatus    }
    const movie = await Movie.create(movieObject);
    res.status(201).send(movie);}
```

Create the movie

Request

MovieBookingApplication / session1 / localhost:8080/mba/api/v1/movies/ Save

POST localhost:8080/mba/api/v1/movies/

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

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {
2      "name": "Namastey London",
3      "description": "Indian Romance Movie",
4      "casts": [
5          "Akshay Kumar",
6          "Katrina Kaif"
7      ],
8      "trailerUrl": "http://namasteylondon/trailers/1",
9      "posterUrl": "http://namasteylondon/posters/1",
10     "language": "Hindi",
11     "releaseDate": "23-03-2007",
12     "director": "Vipul Amrutlal Shah",
13     "releaseStatus": "RELEASED"
14 }
```

Create the movie

Response

Body Cookies Headers (7) Test Results Status: 201 Created Time: 42 ms Size: 625 B

Pretty Raw Preview Visualize JSON


```
4     "casts": [
5         "Akshay Kumar",
6         "Katrina Kaif"
7     ],
8     "trailerUrl": "http://namasteylondon/trailers/1",
9     "posterUrl": "http://namasteylondon/posters/1",
10    "language": "Hindi",
11    "releaseDate": "23-03-2007",
12    "director": "Vipul Amrutlal Shah",
13    "releaseStatus": "RELEASED",
14    "_id": "62487f8dc068756bc6777eb1",
15    "updatedAt": "2022-04-02T16:53:33.981Z",
16    "__v": 0
```

Get the movies

[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session1/controllers/movie.controller.js

Here, we will use the find function of the Movie schema present in our database.
We also have used a query object here for the movie name. If the movie name will be provided then it will give a movie based on the movie name itself.
If we are not giving a name in the API as a query parameter, it will give a list of all the movies

Get the movies

```
exports.getAllMovies = async (req, res) => {

    const queryObj = {};

    if (req.query.name != undefined) {
        queryObj.name = req.query.name;
    }
    const movies = await Movie.find(queryObj);
    res.status(200).send(movies);

}
```

Get the movies

Request – Get All the movies

GET localhost:8080/mba/api/v1/movies

Params Authorization Headers (6) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Get the movies

Response

Body Cookies Headers (7) Test Results Status: 200 OK Time: 13 ms Size: 1011 B

Pretty Raw Preview Visualize JSON


```
2   {
3       "_id": "62486afcc068756bc6777e9d",
4       "name": "Sharmaji Namkeen",
5       "description": "Comedy Masala Movie : Updated",
6       "casts": [
7           "Rishi Kapoor",
8           "Juhi Chawla"
9       ],
10      "trailerUrl": "http://sharmajinamkeen/trailers/1",
11      "posterUrl": "http://sharmajisamkeen/posters/1",
12      "language": "Hindi",
13      "releaseDate": "31-03-2022",
14      "director": "Hitesh Bhatia",
15      "releaseStatus": "RELEASED",
16      "updatedAt": "2022-04-02T15:25:48.629Z",
17      "__v": 0
18  },
19  {
20      "_id": "62487f8dc068756bc6777eb1",
```

Get the movies

Request – Get the movie based on the movie name

MovieBookingApplication / session1 / localhost:8080/mba/api/v1/movies/

Save

GET localhost:8080/mba/api/v1/movies?name=Sharmaji Namkeen

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

	KEY	VALUE	DESCRIPTION
☑	name	Sharmaji Namkeen	
	Key	Value	Description

Get the movies

Response

Body Cookies Headers (7) Test Results Status: 200 OK Time: 12 ms Size: 625 B

Pretty Raw Preview Visualize JSON


```
{
    "_id": "62486afcc068756bc6777e9d",
    "name": "Sharmaji Namkeen",
    "description": "Comedy Masala Movie : Updated",
    "casts": [
        "Rishi Kapoor",
        "Juhi Chawla"
    ],
    "trailerUrl": "http://sharmajinamkeen/trailers/1",
    "posterUrl": "http://sharmajisamkeen/posters/1",
    "language": "Hindi",
    "releaseDate": "31-03-2022",
    "director": "Hitesh Bhatia",
    "releaseStatus": "RELEASED",
```

Get the movie based on the movie id

In this API, we will get a movie based on the movie id

[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session1/controllers/movie.controller.js

Here, we will use the findOne function of the Movie schema present in our database.
We will pass movie id as a parameter to the findOne function of the Movie schema. This will give us the movie having movie id equal to the passed one

Get the movie based on the movie id

```
exports.getMovie = async (req, res) => {
    const movie = await Movie.findOne({
        _id: req.params.id
    });
    res.status(200).send(movie);

}
```

Get the movie based on the movie id

Request

GET localhost:8080/mba/api/v1/movies/62486afcc068756bc6777e9d

Params Authorization Headers (6) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Get the movie based on the movie id

Response

Body Cookies Headers (7) Test Results Status: 200 OK Time: 12 ms Size: 625 B

Pretty Raw Preview Visualize JSON


```
{
    "_id": "62486afcc068756bc6777e9d",
    "name": "Sharmaji Namkeen",
    "description": "Comedy Masala Movie : Updated",
    "casts": [
        "Rishi Kapoor",
        "Juhi Chawla"
    ],
    "trailerUrl": "http://sharmajinamkeen/trailers/1",
    "posterUrl": "http://sharmajisamkeen/posters/1",
    "language": "Hindi",
    "releaseDate": "31-03-2022",
    "director": "Hitesh Bhatia",
    "releaseStatus": "RELEASED",
```

Update the movie based on the movie id

In this API, we will update a movie based on the movie id
Before updating, we will first validate the request same as done while creating the movie.
[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session1/controllers/movie.controller.js

Here, first, we will fetch the movie based on the movie id given in the request. We will use the findOne function of the Movie schema to fetch the movie.

- If the movie does not exist, we will send an error message saying that bMovie being updated doesn't exist"
- If the movie exists, we will proceed with updating the movie
- We will fetch all the details of the movie from the request one by one.
- Once done, we will use the save function of the Movie schema to save the updated movie in the database.

Update the movie based on the movie id

```
exports.updateMovie = async (req, res) => {
    const savedMovie = await Movie.findOne({ _id: req.params.id });
    if (!savedMovie) {
        res.status(400).send({
            message: "Movie being updated doesn't exist"        });     }
    savedMovie.name = req.body.name != undefined ? req.body.name :
savedMovie.name,
        savedMovie.description = req.body.description != undefined ?
req.body.description : savedMovie.description,
        savedMovie.casts = req.body.casts != undefined ? req.body.casts :
savedMovie.casts,
        savedMovie.director = req.body.director != undefined ?
req.body.director : savedMovie.director,
```

Update the movie based on the movie id

```
        savedMovie.trailerUrl = req.body.trailerUrl != undefined ?
req.body.trailerUrl : savedMovie.trailerUrl,
        savedMovie.posterUrl = req.body.posterUrl != undefined ?
req.body.posterUrl : savedMovie.posterUrl,
        savedMovie.language = req.body.language != undefined ?
req.body.language : savedMovie.language,
        savedMovie.releaseDate = req.body.releaseDate != undefined ?
req.body.releaseDate : savedMovie.releaseDate,
        savedMovie.releaseSatus = req.body.releaseSatus != undefined ?
req.body.releaseSatus : savedMovie.releaseSatus
    var updatedMovie = await savedMovie.save();
    res.status(200).send(updatedMovie); }
```

Update the movie based on the movie id

Request

PUT localhost:8080/mba/api/v1/movies/62486afcc068756bc6777e9d

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

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1  {
2      "name": "Sharmaji Namkeen",
3      "description": "Comedy Masala Movie : Updated",
4      "casts": [
5          "Rishi Kapoor",
6          "Juhi Chawla"
7      ],
8      "trailerUrl": "http://sharmajinamkeen/trailers/1",
9      "posterUrl": "http://sharmajisamkeen/posters/1",
10     "language": "Hindi",
11     "releaseDate": "31-03-2022",
12     "director": "Hitesh Bhatia",
13     "releaseStatus": "UNRELEASED"
14 }
```

Update the movie based on the movie id

Response

Body Cookies Headers (7) Test Results Status: 200 OK Time: 30 ms Size: 625 B

Pretty Raw Preview Visualize JSON


```
2    "_id": "62486afcc068756bc6777e9d",
3    "name": "Sharmaji Namkeen",
4    "description": "Comedy Masala Movie : Updated",
5    "casts": [
6        "Rishi Kapoor",
7        "Juhi Chawla"
8    ],
9    "trailerUrl": "http://sharmajinamkeen/trailers/1",
10   "posterUrl": "http://sharmajisamkeen/posters/1",
11   "language": "Hindi",
12   "releaseDate": "31-03-2022",
13   "director": "Hitesh Bhatia",
14   "releaseStatus": "RELEASED",
```

Delete the movie based on the movie id

In this API, we will delete a movie based on the movie id

[Code Link] -
https://github.com/Vishwa07dev/mba_backend/blob/session1/controllers/movie.controller.js

Here, we will use the deleteOne function of the Movie schema present in our database.
We will pass the movie id as a parameter to the deleteOne function of the Movie schema. This will delete the movie having a movie id equal to the passed one

Delete the movie based on the movie id

```
exports.deleteMovie = async (req, res) => {

    await Movie.deleteOne({
        _id: req.params.id
    });
    res.status(200).send({
  message : "Successfully delete movie with id [ " b req.params.id b " ]"
    });
};
```

Delete the movie based on the movie id

Request

DELETE localhost:8080/mba/api/v1/movies/62486afcc068756bc6777e9d

Params Authorization Headers (6) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

Delete the movie based on the movie id

Response

Body Cookies Headers (7) Test Results Status: 200 OK Time: 9 ms Size: 311 B

Pretty Raw Preview Visualize JSON

```
1 {
2     "message": "Successfully delete movie with id [ 6245f0babddfa2ae0d2bba5d ]"
3 }
```

Practice Code

- Write an API that can delete a movie based on the movie name as a query parameter. If the movie doesn't exist then it will throw an error message saying that "Movie doesn't exist with the given name". If a movie exists, then delete the movie using the movie id
- Similar to a Movie, write an API to create a Theatre having Name, City, Description, and PIN Code attributes

MCQs

1. **Which function of MongoDB Schema is used to delete the document?**
   A. deleteOne()
   B. delete()
   C. Both A and B
   D. None

MCQs

1. **Which function of MongoDB Schema is used to delete the document?**
   A. deleteOne()
   B. delete()
   C. Both A and B
   D. None

Answer: A

MCQs

2. **Who are clients in the movie booking application?**
   A. Owner of Cinema Hall
   B. Customers
   C. Both A and B
   D. None

MCQs

2. **Who are clients in the movie booking application?**
   A. Owner of Cinema Hall
   B. Customers
   C. Both A and B
   D. None

Answer: A

MCQs

3. **What are the examples of Movie Booking Applications in the industry?**
   A. BookMyShow
   B. InoxMovies
   C. PVRCinemas
   D. All of the above

MCQs

3. **What are the examples of Movie Booking Applications in the industry?**
   A. BookMyShow
   B. InoxMovies
   C. PVRCinemas
   D. All of the above

Answer: D

MCQs

4. **Which statement is correct about Mongoose?**
   A. Java Library to connect with mongoDB
   B. Python Library to connect with mongoDB
   C. PHP library to connect with mongoDB
   D. Modelling application data in node.js

MCQs

4. **Which statement is correct about Mongoose?**
   A. Java Library to connect with mongoDB
   B. Python Library to connect with mongoDB
   C. PHP library to connect with mongoDB
   D. Modelling application data in node.js

Answer: D

MCQs

5. **Assume Theatre to be a resource. We want to update the Theatre attributes, which of the following would be the correct REST endpoint for doing the same?**
   A. PUT /app/api/v1/theatres/update
   B. PUT /app/api/v1/theatre/update/{id}
   C. PUT /app/api/v1/theatre/{id}
   D. PUT /app/api/v1/theatres/{id}

MCQs

5. **Assume Theatre to be a resource. We want to update the Theatre attributes, which of the following would be the correct REST endpoint for doing the same?**
   A. PUT /app/api/v1/theatres/update
   B. PUT /app/api/v1/theatre/update/{id}
   C. PUT /app/api/v1/theatre/{id}
   D. PUT /app/api/v1/theatres/{id}

Answer: C

Thank You!