A practical exercise to get hands on express js and mongodb

1 Start mongodb server

- Go to your terminal
- Type mongod (this will start a local mongodb server under the address mongodb://127.0.0.1:27017 or mongodb://localhost:27017

2 Create express project

- Go to your preferred location on your pc
- Create a folder name it movie-app
- Inside this folder create 2 sub folders server and client
- Inside server folder open the terminal and type npm init -y (this will start a blank javascript project
- Install the basic dependencies needed to run a basic express project by running the following command :
    - npm install express body-parser dotenv
    - npm install -D nodemon
- In the root folder of your server project create .env file inside of it put PORT=4000
- In the root folder of your server project, create . gitignore file and ignore inside of node_modules folder and .env file
- In the root folder of your server project create a sub folder called app within app create a file named server.js
- inside the server.js file add the basic configuration to start an express project:
- const express = require(“express”)
- const bodyParser = require(“body-parser”)
- require(“dotenv”).config()
- const app = express()
- const PORT = process.env.PORT
- app.use(bodyParser.json())

- app.use(bodyParser.urlEncoded({extended:true})
-app.get(“/”,(req,res)=>res.send(“Hello world!”))
-app.listen(PORT,()=>console.log(`App is running on port ${PORT}`)

- Configure the necessary scripts in package.json file to start your application: in the scripts sections add:

“start”:”node app/server.js”,
“dev”:”nodemon app/server.js”

- To start your application in the root folder of your project open the terminal and type npm run dev this will start a local server on port 4000
- To access your server in the browser type http://localhost:4000

3 Database connection

- Open the terminal in the root folder of your server project and type npm install mongoose
- In the app folder create a sub folder name it config within it create a file called db.js
- Inside db.js file put the database connection configuration:
    - const mongoose = require("mongoose")
    - const {MONGO_DEV_URL} = process.env
    - exports.connect = ()=>{
    - mongoose.connect(MONGO_DEV_URL,{
    - useNewUrlParser:true,
    - useUnifiedTopology:true
    - }).then((x)=>{
    - console.log(`Connected to Mongo Database name: ${x.connections[0].name}`)
    - }).catch(err=>{
    - console.log(`Error connecting to mongo: ${err.reason}`)
    - })
    - }
    - Inside .env filed create a variable named MONGO_DEV_URL:

      MONGO_DEV_URL=mongodb://127.0.0.1:27017/movieDb

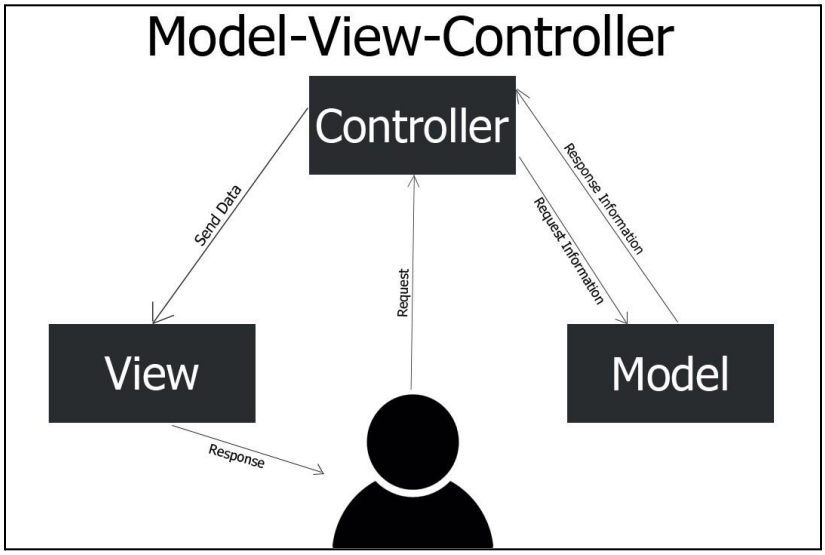
- Go to your server.js file and add the database connection function by adding this before body parser configuration

  require(“./config/db”).connect()

  This line will connect you to the database every time you start your express server.

3 Creating project folders

In our project we will follow mvc architecture



MVC stands for Model-View-Controller, which is a way to organize code in computer programs.

Here's a simple way to think about it:

Imagine you're building a toy car out of blocks. The blocks are like the code in a program.
The "model" is like the blueprint for the car. It defines what the car looks like and how it works. In a program, the model is the code that stores and manages the data or information used in the program. For example, in a game, the model might store information about the player's score or inventory.

The "view" is like the way the car looks when it's finished. In a program, the view is the code that creates what the user sees on the screen, like the buttons or text fields.

The "controller" is like the person who builds the car. They follow the blueprint (the model) to create the car, and they put the finishing touches on it to make it look just right (the view). In a program, the controller is the code that controls how the user interacts with the program. For example, when you click a button, the controller responds by running a piece of code that updates the model and changes what you see on the screen.

So, in short, MVC is a way of organizing code in a program that separates the data or information (model), how the user sees it (view), and how the user interacts with it (controller).

In a MERN application, the model layer is implemented using MongoDB, the view layer is implemented using React, and the controller layer is implemented using Express.js and Node.js. This separation of concerns allows developers to organize their code in a way that makes it easier to maintain, test, and scale their application.

To create our project folders :

- In the app folders create 3 sub folders (controllers , routes, models)

4 Create our first Rest api

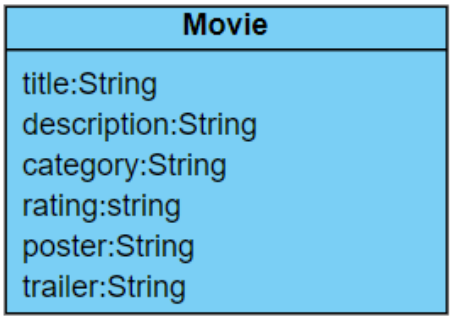
4.1 Creating the starter files that we will need

- In the models folder create a file named movie.js here we will implement our model layer
- In the controllers folder create a file named MovieController here we will implement our controller layer
- In the routes folder create a file named index.js this will be our global router and create also a file named movie.js this will be our movie router

4.2 Implement the movie model

In the models folder inside the movie.js file create the movie model using the mongoose library

- First create the movie schema following this class schema



- All the fields are required
- After creating the schema , create the movie model

4.3 Implement the movie controller

In the controllers folder inside the MovieController file do the following:

- Create and export 5 async functions

getAllMovies : a function that let us retrieve the movies list from the database

getMovieById: a function that let us get the details of one movie

addNewMovie: a function that let us insert a new movie to the database

updateMovie: a function that let us update the information of a movie

deleteMovie: a function that let us delete one movie from the database

For the functions implementation we will get back to it later

4.3 Implement the movie router

In the routes folder inside movie.js file create a router that has five routes:

- A route to get all the movies
- A route to get a movie by id
- A route to add new movie
- A route to update a movie by id

- A route to delete a movie by id

Example: how you can create this router:

```
const express = require(“express”)
const MovieController = require(“../controllers/MovieController”)
const router = express.Router()
module.exports = ()=>{
  router.get(“/”,MovieController.getAllMovies)
  return router
}
```

In this example we have linked the path of the route with getAllMovies function that we created earlier inside our controller , so from now on whenever this request get launched the getAllMovies Function that is responsible for getting the movies list from the database will be executed

Complete the rest of the routes

After completing the routes go inside routes folder inside the index.js file and import the movie router that you have implemented and create create a global router in this file this global router is responsible for creating a middleware with pathname /movies this middleware will use the movie router function

4.3 Implement the movie controller methods

To make your rest api functional you have to implement the methods that we created inside the MovieController file

4.3 Testing our rest api

After building our rest api now it's time to test:

Run your server by taping npm run dev in the root folder

That go to postman or insomnia and start testing the different routes that we have created