Lec-7 MVC Architecture & Rest Principles

Agenda

- MVC architecture
- Rest Principles
- Movies model
- Router and mounting

How to organize files

File Structure

- `api.js`: This is the main entry point of your application. It creates the Express app instance, sets up middleware, and routes.
- `routes`: This directory will contain separate files for different routes.
- `controllers`: This directory will contain functions that handle the logic for each route.
- `models`: This directory will contain your database models.
- `middleware`: This directory will contain custom middleware functions

Creating Routes

- Create a new file within the `routes` directory, e.g., `authRouter.js`.
- Define your routes using Express's router object:

```
const express = require('express');
const authRouter = express.Router();

// Define routes
authRouter
    .post("/login", loginHandler)
    .post("/signup", signupHandler)
    .get("/logout", logoutHandler)
    .get("/profile", protectRouteMiddleware, profilehandler);
```

Creating Controllers

- Create a new file within the `controllers` directory, e.g., `authController.js`.
- Define the functions that will handle the logic for each route:

```
const UserModel = require("../model/userModel");
const util = require("util");
const jwt = require("jsonwebtoken");

const promisify = util.promisify;
const promisdiedJWTsign = promisify(jwt.sign);
const promisdiedJWTverify = promisify(jwt.verify);

async function signupHandler(req, res) {

    // 3. create the user
    try {
        const userObject = req.body;
        // 1. user -> data get , check email , password
        if (!userObject.email || !userObject.password) {
            return res.status(400).json({
                "message": "required data missing",
                status: "failure"
            })
        }
        // 2. email se check -> if exist -> already loggedIn
        const user = await UserModel.findOne({ email: userObject.email });
        if (user) {
            return res.status(400).json({
                "message": "user is already logged in",
                status: "success"
            })
        }
        const newUser = await UserModel.create(userObject);
        // send a response
        res.status(201).json({
            "message": "user signup successfully",
            user: newUser,
            status: "success"
        })
    } catch (err) {
        console.log("err", err);
        res.status(500).json({
            message: err.message,
            status: "failure"
```

```
        })
    }
}

async function loginHandler(req, res) {
        //...
}

async function protectRouteMiddleware(req, res, next) {
        //...
}

async function isAdminMiddleWare(req, res, next) {
        //...
}

async function profilehandler(req, res,) {
        //...
}

async function logoutHandler(req, res) {
        //...
}
```

- Always ensure that the required dependencies are enabled or not.

Exporting

- Export the controller function form `authController.js`

```
// Fuctions ...

module.exports = {
    signupHandler, loginHandler, protectRouteMiddleware, isAdminMiddleWare,
profilehandler, logoutHandler
}
```

- When exporting multiple functions in Node.js and Express.js, the most common and recommended approach is to encapsulate them within an object and export them.
- Export the routs from `authRouter.js`

```
// routes

module.exports= authRouter;
```

Importing

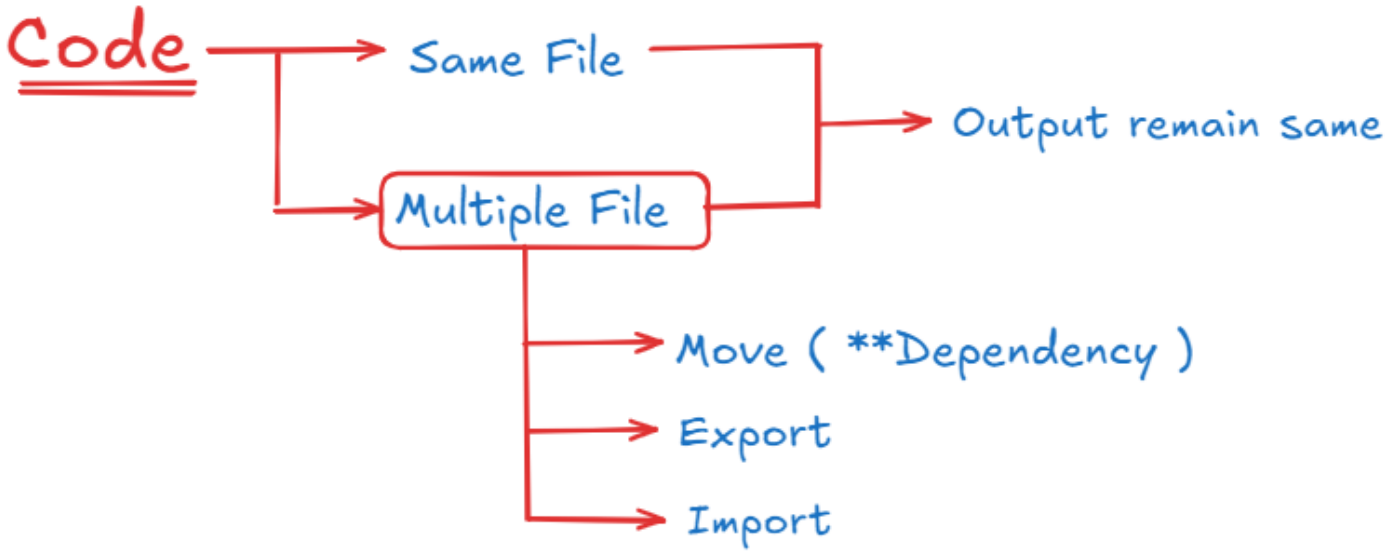
- In `api.js`, import the routes :

```
const authRouter = require("./router/authRouter.js");
app.use(authRouter, "/api/auth");
```

- In the `authRouter.js` route file, import the corresponding controller functions:

```
const express = require('express');
const authRouter = express.Router();
const { loginHandler, signupHandler, logoutHandler, protectRouteMiddleware,
profilehandler } = require("../controller/authController");

// ... routes
```



Rest Principles

REST (Representational State Transfer) is an architectural style for designing distributed systems, particularly web services. It provides a set of guiding principles for creating scalable, reliable, and maintainable APIs.

Key Principles of RESTful APIs

Client-Server Architecture

- Separation of concerns between the client and server.
- Clients handle user interface and presentation, while servers handle data storage and processing.

Resource-Based Routing

- Data is exposed as resources `(e.g., /users, /content)`.
- Each resource is uniquely identified by a route.

Note: Resources on which routes should be decided for ex in an Ecommerce Website we can have

- Users
- Products
- Reviews
- Booking
- Returns

Actions are done by HTTP method

- `GET`: Retrieve a resource.
- `POST`: Create a new resource
- `PATCH`: Update an existing resource.
- `DELETE`: Delete a resource.

Example of routes instead of `app.get("/getUser")` it should be `app.get("/user")`.
Instead of `app.get("/reviewforIphone14")` it should be `app.get("/review/iphone14")`.

Statelessness

- Each request is treated as a self-contained unit, containing all the necessary information.
- No session state is maintained on the server between requests.

- Improves scalability and reliability.
  Instead of `app.get("/nextPage")` it should be `app.get("/5")`.

MVC Architecture

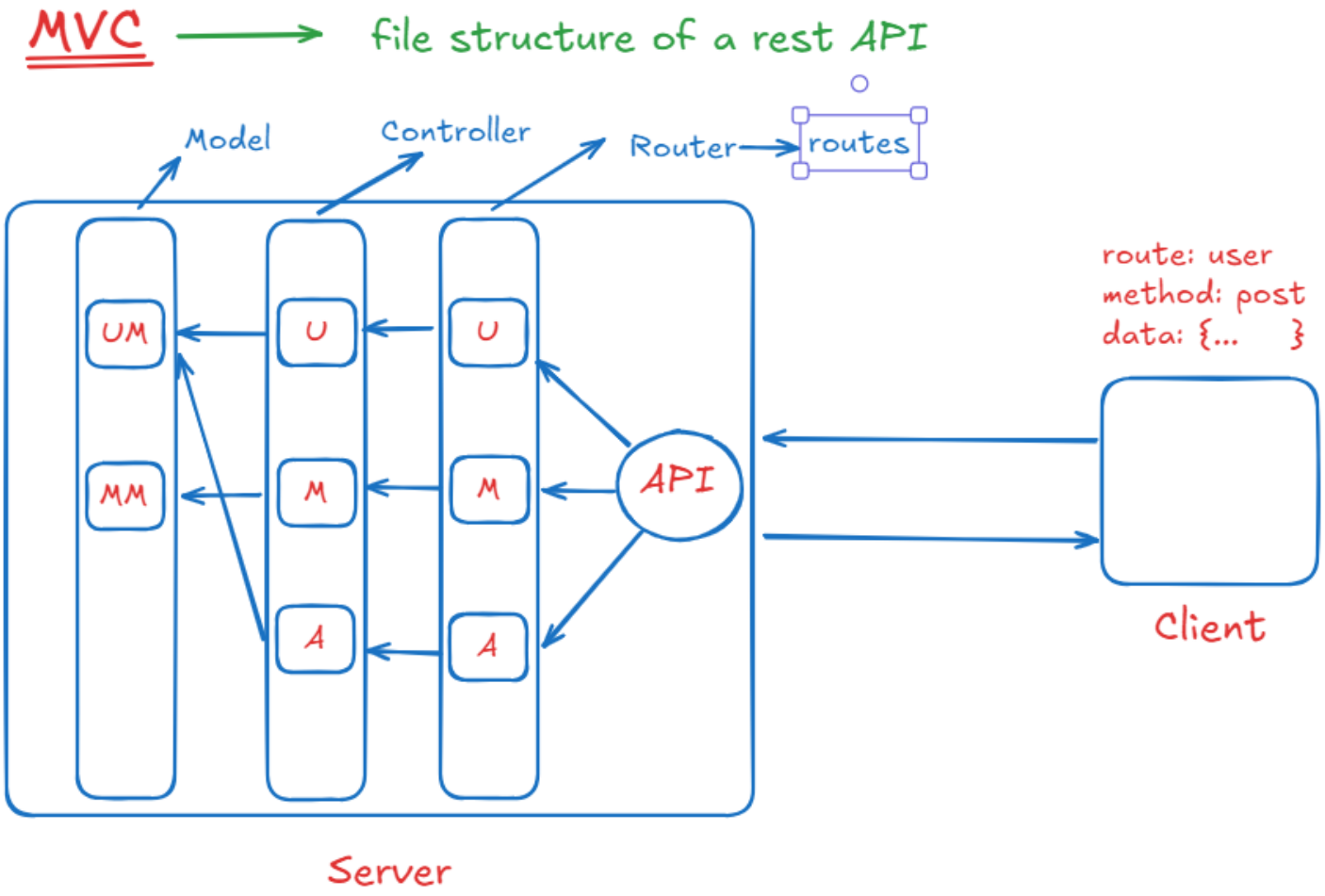
Model-View-Controller (MVC) is a design pattern that separates an application into three main components:

- **Model:** Represents the data and business logic of the application. It handles data access, validation, and updates.
- **View:** Represents the user interface and presentation of the data. It displays the data to the user and allows for interaction.
- **Controller:** Acts as the intermediary between the model and view. It handles user input, updates the model, and determines which view to display.

Benefits of MVC Architecture

- **Separation of Concerns:** Clear separation of responsibilities between the Model, View, and Controller components, leading to better code organization, maintainability, and reusability.
- **Scalability:** Easier to add new features or modify existing ones without affecting the entire application.
- **Testability:** Independent testing of each component is possible, improving code quality and reliability.
- **Maintainability:** Changes to the Model, View, or Controller can be made independently, reducing the risk of unintended side effects.
- **Reusability:** Components can be reused in different contexts, saving development time and effort.

Let's understand with diagram



Time stamp

- Intro (start - 0:19:33)
- File Organize (0:19:33 - 0:33:00)
- Movies (0:33:00 - 0:59:00)
- Rest Principles (0:59:00 - 1:14:00)
- MVC architecture (1:14:00 - end)