📘 MVC Architecture

🔰 What is MVC Architecture?

MVC stands for:

- **M** – Model
- **V** – View
- **C** – Controller

It is a **design pattern** used to **separate application logic into three interconnected parts**. This helps in organizing code better and making it easier to manage, scale, and debug.

❗ MVC is not a programming language or framework. It's just a **pattern** (a structured way to organize code).

📂 Overview of MVC Components

Let’s understand each part of MVC in the context of a **MERN stack (MongoDB, Express.js, React.js, Node.js)** project:

---

1. 🧠 Model (M)

- **What it does:**
  Handles **data logic**. Communicates with the database.
- **In MERN (Backend):**
  - You define **Mongoose models** for MongoDB collections.
  - Example: A `User` model with fields like name, email, password.
- **Example Code:**

```
js

// models/User.js
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: String,
```

```
  email: String,
  password: String,
});

const User = mongoose.model('User', userSchema);
module.exports = User;
```

---

2. 🛠️ Controller (C)

- **What it does:**
  Handles the **business logic** and **requests**.
  - It processes the request from client
  - Talks to the model (for DB read/write)
  - Sends a response back to the client
- **In MERN:**
  These are JavaScript functions (usually in a separate folder called `controllers/`) that handle routes' logic.
- **Example Code:**

```
js

// controllers/userController.js
const User = require('../models/User');

const getAllUsers = async (req, res) => {
  try {
    const users = await User.find();
    res.status(200).json(users);
  } catch (err) {
    res.status(500).json({ message: err.message
});
  }
};

module.exports = { getAllUsers };
```

---

3. 💻 View (V)

- **What it does:**
  Presents the data to the user – the **UI** part.
- **In MERN:**

- **React.js** is used for the frontend – this is your View.
- It consumes data from API and shows it to the user.

- **Example Code (React):**

```
jsx

// components/UserList.js
import React, { useEffect, useState } from
'react';
import axios from 'axios';

const UserList = () => {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    axios.get('/api/users').then(res => {
      setUsers(res.data);
    });
  }, []);

  return (
    <div>
      <h1>User List</h1>
      <ul>
        {users.map(user => (
          <li key={user._id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
};

export default UserList;
```

🔁 How These Three Work Together (MVC Flow)

1. **User** interacts with the **View (React UI)** – clicks a button or submits a form.
2. That triggers an **HTTP request** to the backend **Controller (Express route)**.
3. The **Controller** calls the appropriate **Model (Mongoose)** to access or update the database.
4. The **Model** interacts with **MongoDB** and returns data.

5. The **Controller** sends back a **response (JSON)**.
6. The **View** updates the UI with the data.

💡 Why Do We Use MVC Architecture?

✅ Benefits of Using MVC in MERN Projects

Benefit	Explanation
Separation of concerns	Each part has a separate responsibility: Data, UI, Logic
Cleaner Codebase	Easier to read, manage, and debug
Reusability	Models and Controllers can be reused across different views
Scalability	Easier to add new features without breaking others
Team Collaboration	Frontend and Backend teams can work independently
Testing	Easier to write tests for each component separately

📂 Folder Structure of MVC in a MERN Project (Backend Only)

```
bash

backend/
│
├── models/              # MongoDB Models
│   └── user.js
│
├── controllers/         # Business logic
│   └── userController.js
│
├── routes/              # Express Routes
│   └── userRoutes.js
│
├── server.js            # Entry point
├── config/              # DB config
```

```
|    └── db.js
```

🌐 Example Route Usage

```
js

// routes/userRoutes.js
const express = require('express');
const router = express.Router();
const { getAllUsers } =
require('../controllers/userController');

router.get('/users', getAllUsers);

module.exports = router;
js

// server.js
const express = require('express');
const connectDB = require('./config/db');
const userRoutes = require('./routes/userRoutes');

const app = express();
connectDB();

app.use(express.json());
app.use('/api', userRoutes);

app.listen(5000, () => {
  console.log('Server running on port 5000');
});
```
