MERN Stack MVC Pattern

What is MVC?

MVC (Model-View-Controller) is a software design pattern used to separate the logic of an application into three interconnected components:

- **Model**: Handles data and business logic (e.g., MongoDB + Mongoose schema).
- **View**: The user interface (e.g., React.js).
- **Controller**: Handles user input and communicates between Model and View (e.g., Express route controllers).

Folder Structure (Example)

- project-root/
    - backend/
        - controllers/
            - userController.js
        - models/
            - User.js
        - routes/
            - userRoutes.js
        - config/
        - server.js
    - frontend/
        - public/
        - src/
            - components/
            - pages/
            - App.jsx
            - index.jsx

Flow of Data in MVC (MERN)

1. **Client (React)** sends a request to the backend.
2. **Express Router** directs request to a **Controller**.
3. **Controller** interacts with the **Model** (MongoDB).
4. **Model** sends back data → Controller → Response to Client (View updates).

Backend Code Example

Model (models/User.js)

```
import mongoose from 'mongoose';
const userSchema = new mongoose.Schema({
  name: String,
  email: String,
  password: String,
}, { timestamps: true });
export default mongoose.model('User', userSchema);
```

Controller (controllers/userController.js)

```
import User from '../models/User.js';
export const getUsers = async (req, res) => {
  try {
    const users = await User.find();
    res.json(users);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

Routes (routes/userRoutes.js)

```
import express from 'express';
import { getUsers } from '../controllers/userController.js';

const router = express.Router();
router.get('/', getUsers);

export default router;
```

Server (server.js)

```
import express from 'express';
import mongoose from 'mongoose';
import userRoutes from './routes/userRoutes.js';

const app = express();
app.use(express.json());

mongoose.connect('mongodb://localhost:27017/mvcapp')
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));
app.use('/api/users', userRoutes);
app.listen(5000, () => console.log('Server running on port 5000'));
```

Frontend (React)

Fetch Data in Component

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
function UserList() {
  const [users, setUsers] = useState([]);
  useEffect(() => {
    axios.get('/api/users')
      .then(res => setUsers(res.data))
      .catch(err => console.error(err));
  }, []);
  return (
    <div>
      <h1>User List:</h1>
      {users.map(user => (
        <p key={user._id}>{user.name}</p>
      ))}
    </div>
  );
}

export default UserList;
```

Router (routes/userRoutes.js)
Routers handle API paths and connect them to controller logic.

```
const express = require('express');
const router = express.Router();
const { getUsers, createUser } = require('../controllers/userController');

router.get('/', getUsers);        // GET /api/users
router.post('/', createUser);     // POST /api/users

module.exports = router;
```

Controller (controllers/userController.js)
Controllers contain logic.

```
const User = require('../models/userModel');

// Get all users
const getUsers = async (req, res) => {
  const users = await User.find();
  res.json(users);
};

// Create new user
const createUser = async (req, res) => {
  const { name, email } = req.body;
  const user = await User.create({ name, email });
  res.status(201).json(user);
};

module.exports = { getUsers, createUser };
```

Model (models/userModel.js)
Defines schema and model using Mongoose.

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
}, { timestamps: true });

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

Main App File (app.js)

```
const express = require('express');
const mongoose = require('mongoose');
const userRoutes = require('./routes/userRoutes');
require('dotenv').config();

const app = express();
app.use(express.json());

mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log("MongoDB Connected"))
  .catch(err => console.log(err));

app.use('/api/users', userRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Practice Set

Task 1: Create Product API (Same MVC Structure)

- Routes: /api/products
- Model: productModel.js with name, price, category
- Controller: getProducts, createProduct

Task 2: Create Auth System

- User login with JWT token
- Controller: registerUser, loginUser
- Add middleware: authMiddleware.js for protecting routes

Project Structure

```
project/
|
 ├── controllers/
|          └── authController.js
|
 ├── middleware/
|          └── authMiddleware.js
|
 ├── models/
|          └── userModel.js
|
 ├── routes/
|          └── authRoutes.js
|
 ├── config/
|          └── db.js
|
 ├── app.js
└── .env
```

userModel.js

```
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
}, { timestamps: true });

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

authController.js

```
const User = require('../models/userModel');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');

// Generate Token
const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: '7d' });
};
// Register
const registerUser = async (req, res) => {
  const { name, email, password } = req.body;
  const userExist = await User.findOne({ email });
  if (userExist) return res.status(400).json({ message: "User already exists" });

  const hashedPassword = await bcrypt.hash(password, 10);
  const user = await User.create({ name, email, password: hashedPassword });

  res.status(201).json({msg:”user created!”  });
};

// Login
const loginUser = async (req, res) => {
  const { email, password } = req.body;
  const user = await User.findOne({ email });

  if (!user) return res.status(400).json({ message: "User not found" });

  const isMatch = await bcrypt.compare(password, user.password);
  if (!isMatch) return res.status(401).json({ message: "Invalid credentials" });

  res.json({
    _id: user._id,
    token: generateToken(user._id),
  });
};

module.exports = { registerUser, loginUser };
```

authMiddleware.js

```
const jwt = require('jsonwebtoken');
const User = require('../models/userModel');

const protect = async (req, res, next) => {
```

```
  let token;

  if (req.headers.authorization?.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);
      req.user = await User.findById(decoded.id).select('-password');
      next();
    } catch (error) {
      res.status(401).json({ message: 'Not authorized, token failed' });
    }
  }

  if (!token) {
    res.status(401).json({ message: 'Not authorized, no token' });
  }
};

module.exports = { protect };
```

authRoutes.js

```
const express = require('express');
const router = express.Router();
const { registerUser, loginUser } = require('../controllers/authController');

router.post('/register', registerUser);
router.post('/login', loginUser);

module.exports = router;
```

app.js

```
const express = require('express');
const mongoose = require('mongoose');
const dotenv = require('dotenv');
const authRoutes = require('./routes/authRoutes');

dotenv.config();
const app = express();
app.use(express.json());

mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log('MongoDB connected'))
  .catch(err => console.log(err));

app.use('/api/auth', authRoutes);

const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

.env

```
MONGO_URI=your_mongodb_connection
JWT_SECRET=your_secret_key
```

Usage of Protected Route Example

Add in your routes/userRoutes.js

```
const express = require('express');
const router = express.Router();
const { protect } = require('../middleware/authMiddleware');

router.get('/profile', protect, (req, res) => {
  res.json({ user: req.user });
});

module.exports = router;
```

Testing the Flow

Register a user:

POST /api/auth/register

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "123456"
}
```

Login user:

```
POST /api/auth/login
{
  "email": "john@example.com",
  "password": "123456"
}
```

Order Model (Mongoose Schema)

```
const mongoose = require('mongoose');

const orderSchema = new mongoose.Schema({
  user: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User', // refers to the User model
    required: true
  },
  products: [
    {
      product: {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'Product', // refers to the Product model
        required: true
      },
      quantity: {
        type: Number,
        required: true,
        default: 1
      }
    }
  ],
  shippingAddress: {
    address: String,
    city: String,
    state: String,
    postalCode: String,
    country: String
  },
  paymentMethod: {
    type: String,
    required: true
  },
  paymentStatus: {
    type: String,
    enum: ['Pending', 'Paid', 'Failed'],
    default: 'Pending'
  },
  orderStatus: {
    type: String,
    enum: ['Processing', 'Shipped', 'Delivered', 'Cancelled'],
    default: 'Processing'
  },
```

```
  totalAmount: {
    type: Number,
    required: true
  },
  orderedAt: {
    type: Date,
    default: Date.now
  }
});

module.exports = mongoose.model('Order', orderSchema);
```

Example API Usage (Postman)
POST /api/orders

```
{
  "user": "64c9e38f8e364d001f0dd9f5",
  "products": [
    { "product": "64ca0ed5583243001fa274ae", "quantity": 2 },
    { "product": "64ca0f15583243001fa274b1", "quantity": 1 }
  ],
  "shippingAddress": {
    "address": "123 Street",
    "city": "Delhi",
    "state": "Delhi",
    "postalCode": "110001",
    "country": "India"
  },
  "paymentMethod": "Credit Card",
  "totalAmount": 500
}
```