Day 11

- **MVC**
- **EJS**

Introduction to MVC

- MVC stands for **Model-View-Controller**, a design pattern used to develop scalable and maintainable software applications.
- It separates the application logic into three interconnected components:
  1. **Model**: Manages data and business logic.
  2. **View**: Handles the user interface and presentation logic.
  3. **Controller**: Acts as an intermediary between the Model and the View, handling user inputs and coordinating responses.

Components of MVC

1. Model

- **Responsibilities**:
  - Manages the application data and rules.
  - Interacts with the database or external APIs to fetch/store data.
  - Performs business logic operations.
- **Example**: In a student management app:

```
const mongoose = require('mongoose');
const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  major: String,
});
const Student = mongoose.model('Student', studentSchema);
module.exports = Student;
```

View

- **Responsibilities**:
  - Presents the data to the user in a specific format.
  - Does not contain business logic.
  - Can be implemented using HTML, EJS, or front-end frameworks.
- **Example**: In EJS:

```
<h1>Student List</h1>
<ul>
  <% students.forEach(student => { %>
    <li><%= student.name %> - <%= student.major %></li>
  <% }) %>
</ul>
```

Controller

- **Responsibilities**:
  - Receives user inputs (via HTTP requests).
  - Invokes the Model to process data.
  - Sends data to the View for rendering.

```
const Student = require('../models/student');

const getStudents = async (req, res) => {
  const students = await Student.find();
  res.render('students', { students });
};

module.exports = { getStudents };
```

Advantages of MVC

1. **Separation of Concerns**:
   - Each component handles a distinct responsibility.
   - Makes code cleaner and more modular.
2. **Reusability**:
   - Models and Views can be reused in other parts of the application.
3. **Scalability**:
   - Easier to extend the application with new features.
4. **Testability**:
   - Components can be tested independently.

How MVC Works

1. **User Input**: The user interacts with the View (UI).
2. **Controller**: The Controller processes the input, updates the Model, and determines which View to render.
3. **Model**: The Model retrieves or modifies data and returns the result to the Controller.
4. **View**: The View displays the data to the user.

File Organization in MVC

project/

├── app/

│ ├── models/

│ │ └── student.js

│ ├── views/

│ │ └── students.ejs

```
│   ├── controllers/
│   │   └── studentController.js
├── routes/
│   └── studentRoutes.js
├── config/
│   └── db.js
├── server.js
```

EJS (Embedded JavaScript)

What is EJS?

- **EJS (Embedded JavaScript)** is a templating engine for Node.js that allows embedding JavaScript directly into HTML to create dynamic web pages.
- It helps render HTML dynamically on the server-side by combining template files with data passed from the server.

Key Features of EJS

1. **Dynamic Content**:
   - Generates HTML dynamically by embedding JavaScript within the HTML.
2. **JavaScript Inside HTML**:
   - Allows using JavaScript logic (e.g., loops, conditionals) inside templates.
3. **Template Inheritance**:
   - Supports partials and layouts for reusing common components like headers and footers.
4. **Lightweight**:
   - Simple and fast to integrate with Node.js and Express.
5. **Compatibility**:
   - Works seamlessly with Express.js and supports static files.

Why Use EJS?

- To separate server-side logic from the user interface.
- To dynamically generate HTML with data from the server.
- To quickly prototype applications with server-side rendering

EJS Syntax

1. Output Data

- <%= data %>: Outputs escaped data to prevent Cross-Site Scripting (XSS).

- <%- data %>: Outputs raw, unescaped data (use cautiously).

```
<%= "Hello, World!" %> <!-- Output: Hello, World! -->
<%- "<h1>Welcome</h1>" %> <!-- Output: <h1>Welcome</h1> -->
```

Logic Tags

- <% %>: Used for control flow statements like loops and conditionals.
- <%# %>: Comments that are not rendered in the HTML.

```
<% if (user) { %>
  <p>Welcome, <%= user.name %>!</p>
 <% } else { %>
  <p>Please log in.</p>
 <% } %>
```

Include Partials

- Partials are reusable components (e.g., headers, footers).
- Syntax: <%- include('filename') %>

```
<%- include('header') %>
<h1>Homepage</h1>
<%- include('footer') %>
```

Looping

- Use <% %> for loops to dynamically generate repeated content.

```
<ul>
 <% users.forEach(user => { %>
  <li><%= user.name %> - <%= user.email %></li>
 <% }); %>
</ul>
```

Features of EJS

1. **Dynamic Content Rendering**
    - EJS can render dynamic data passed from the server:

```
<h1>Welcome, <%= user.name %>!</h1>
```

Partial Views

- Allows reuse of common components like headers, footers, etc.
- Example:
    - **header.ejs**:

```
<header>
```

```
  <h1>My Website</h1>
</header>
```

index.ejs

```
<%- include('header') %>
<p>Welcome to the homepage!</p>
```

Advantages of EJS

1. **Familiar Syntax**: Uses standard JavaScript for templating.
2. **Integration with Express**: Works seamlessly with Express.js.
3. **Reusable Components**: Easy to include partials and reuse templates.
4. **Supports Logic**: You can add JavaScript logic directly in templates.

Limitations of EJS

1. **Verbosity**:
   - Mixing HTML and JavaScript can make templates harder to read and maintain.
2. **Server-Side Only**:
   - EJS doesn't support client-side rendering.
3. **Lack of Advanced Features**:
   - No built-in support for two-way data binding or reactive updates.

When to Use EJS

- **Simple Applications**:
  - When you need server-side rendering for basic apps.
- **Prototyping**:
  - Quickly create dynamic prototypes with data from the server.
- **Static Content with Dynamic Data**:
  - Generate HTML pages with backend data.

Interview Questions and Answers:

What is MVC? Explain its components.

Answer: MVC stands for **Model-View-Controller**, a software design pattern that separates an application into three interconnected components:

1. **Model**:

    - Manages the data, business logic, and rules.
    - Handles database interactions.
2. **View**:
    - Responsible for the user interface and displaying data.
    - Interacts only with the controller to receive data.
3. **Controller**:
    - Acts as an intermediary between the Model and View.
    - Handles user inputs and updates the Model or View accordingly.

2. What are the benefits of using MVC?

Answer:

1. **Separation of Concerns**: Each component handles a specific aspect of the application.
2. **Scalability**: Easy to extend and maintain the application.
3. **Reusability**: Components like Views and Models can be reused across projects.
4. **Testability**: The separation makes unit testing more manageable.

3. How does the MVC flow work?

Answer:

1. **User Input**: The user interacts with the View (UI).
2. **Controller**: The Controller processes the input and communicates with the Model.
3. **Model**: The Model performs business logic, retrieves or updates data, and sends it back to the Controller.
4. **View**: The Controller updates the View with the processed data for the user.

4. How is MVC different from MVVM (Model-View-ViewModel)?

Answer:

- **MVC**:
    - The Controller acts as the intermediary between the View and Model.
    - Direct user interaction is handled by the Controller.
- **MVVM**:
    - Introduces a ViewModel that binds the View and Model.
    - The ViewModel contains UI logic and handles data binding automatically.

5. Can you explain the role of routing in MVC?

Answer: Routing maps a URL to a specific controller and action. It enables clean, user-friendly URLs and determines which controller method to invoke based on the request.

Example in **Express.js**:

```
app.get('/students', studentController.getAllStudents);
```

What are some real-life examples of MVC architecture?

Answer:

1. **E-commerce Websites**:
    - Model: Manages products, orders, and users.
    - View: Displays product listings, order forms.
    - Controller: Handles user actions like adding to cart or placing an order.
2. **Blog Platforms**:
    - Model: Stores blog posts, categories, and comments.
    - View: Renders blog content for readers.
    - Controller: Processes actions like posting a comment.

7. What is the difference between a thin controller and a fat controller?

Answer:

- **Thin Controller**:
    - Contains minimal logic and delegates tasks to Models and Services.
    - Preferred for maintainability and separation of concerns.
- **Fat Controller**:
    - Contains too much logic, making it harder to test and maintain.
    - Indicates poor architecture.

8. How do you handle errors in MVC?

Answer:

1. **Controller**:
    - Catch exceptions and send error responses.

```
app.get('/users', async (req, res) => {
   try {
    const users = await User.find();
    res.json(users);
   } catch (err) {
    res.status(500).json({ error: 'Server Error' });
   }
 });
```

Middleware:

- Use centralized error-handling middleware to catch errors globally.

View:

- Display user-friendly error messages.

How does data binding work in MVC?

Answer:

- **Two-Way Data Binding**: Changes in the Model automatically update the View, and changes in the View update the Model (common in frameworks like Angular).
- **One-Way Data Binding**: Data flows from the Model to the View (common in traditional MVC like Express.js).

10. How do you organize files in an MVC project?

Answer: Example Directory Structure:

```
project/
├── app/
│   ├── models/
│   │   └── user.js
│   ├── views/
│   │   └── users.ejs
│   ├── controllers/
│   │   └── userController.js
├── config/
│   └── db.js
├── routes/
│   └── userRoutes.js
├── server.js
```

What is EJS?

Answer: EJS (Embedded JavaScript) is a templating engine for Node.js that allows embedding JavaScript code into HTML files. It is used for rendering dynamic web pages on the server-side by combining data with HTML templates.

How do you install and configure EJS in a Node.js project?

Answer:

```
npm install ejs
```

Configuration in `server.js`.

```
const express = require('express');
const app = express();

// Set EJS as the view engine
app.set('view engine', 'ejs');

// Define the views directory
```

```
app.set('views', './views');
```

How do you pass data to an EJS template?

Answer: You can pass data to an EJS template using the `res.render` method:

```
app.get('/', (req, res) => {
    const data = { name: 'John Doe', age: 25 };
    res.render('index', { data });
  });
```

In the EJS file (`index.ejs`):

```
<h1>Hello, <%= data.name %>!</h1>
<p>You are <%= data.age %> years old.</p>
```

What are the key EJS tags, and how are they used?

Answer:

- `<%= %>`: Outputs escaped data (prevents HTML injection).
- `<%- %>`: Outputs raw HTML.
- `<% %>`: Executes JavaScript code (e.g., loops, conditionals).
- `<%# %>`: Comment (not rendered in the HTML).

```
<% if (data) { %>
    <h1>Hello, <%= data.name %>!</h1>
  <% } else { %>
    <p>No data available</p>
  <% } %>
```

What are partials in EJS, and how do you use them?

Answer: Partials are reusable components in EJS templates. They are included using the `<%- include() %>` syntax.

Example:

- **header.ejs**:

```
<header>
  <h1>My Website</h1>
</header>
```

index.ejs:

<%- include('header') %>

Welcome to the homepage!

How do you pass multiple variables to an EJS template?

Answer: You can pass multiple variables as an object in `res.render`:

```
app.get('/', (req, res) => {
    res.render('index', { name: 'John', age: 25 });
  });
```

In the EJS template:

```
<h1>Hello, <%= name %>!</h1>
<p>You are <%= age %> years old.</p>
```