C237: Software Application Development

LESSON 06A: DYNAMIC CONTENT WITH TEMPLATING ENGINES (EJS)

9.15AM – 11.30AM

Model View Controller (MVC)

SOFTWARE ARCHITECHTURAL PATTERN

What is Model-View-Controller (MVC)?

- A software architectural pattern used for designing and developing user interfaces, especially in web applications.
- The Model-View-Controller (MVC) framework is an architectural/design pattern that separates an application into three main logical components Model, View, and Controller.
- Each architectural component is built to handle specific development aspects of an application.
- While Node.js itself doesn't enforce any specific architecture, developers often implement MVC to organize their code for better maintainability and scalability.

What is Model-View-Controller (MVC)?

- Here's how MVC typically works in a Node.js context:

MODEL

- represents the **data layer** of the application. It handles interactions with the database or any other data storage mechanism.
- This can involve defining data schemas, performing **CRUD (Create, Read, Update, Delete)** operations, and enforcing business logic.
- Popular libraries such as **MySQL** are often used to facilitate these interactions.

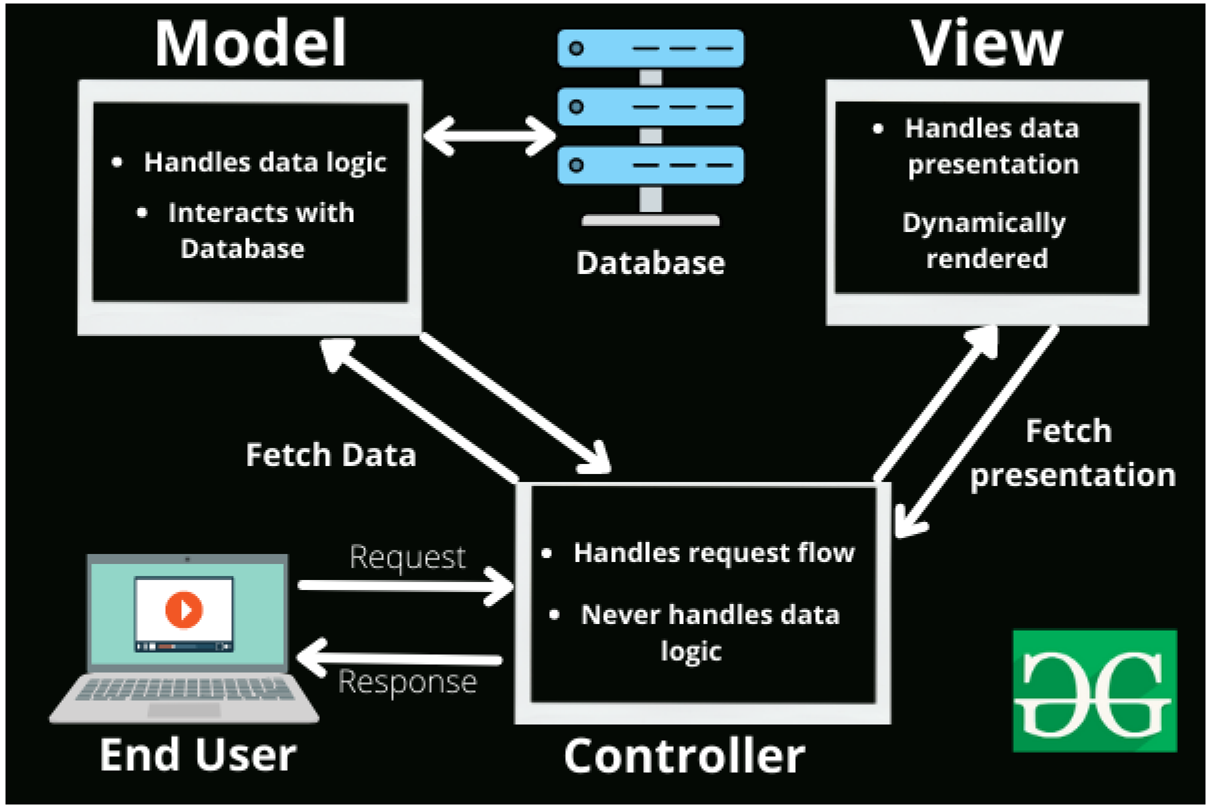
VIEW

- responsible for rendering the **user interface** and presenting data to the client.
- In traditional web applications, views are typically HTML templates that get populated with data from the model. Popular templating engines in the Node.js ecosystem include **EJS**, Pug, Handlebars, and Mustache.

CONTROLLER

- act as the **intermediary between the model and the view**. They handle incoming requests from clients, process data from the request (if necessary), interact with the model to retrieve or manipulate data, and finally determine which view to render with the processed data.
- Libraries like **Express.js** are commonly used to build controllers and manage routing in Node.js applications.

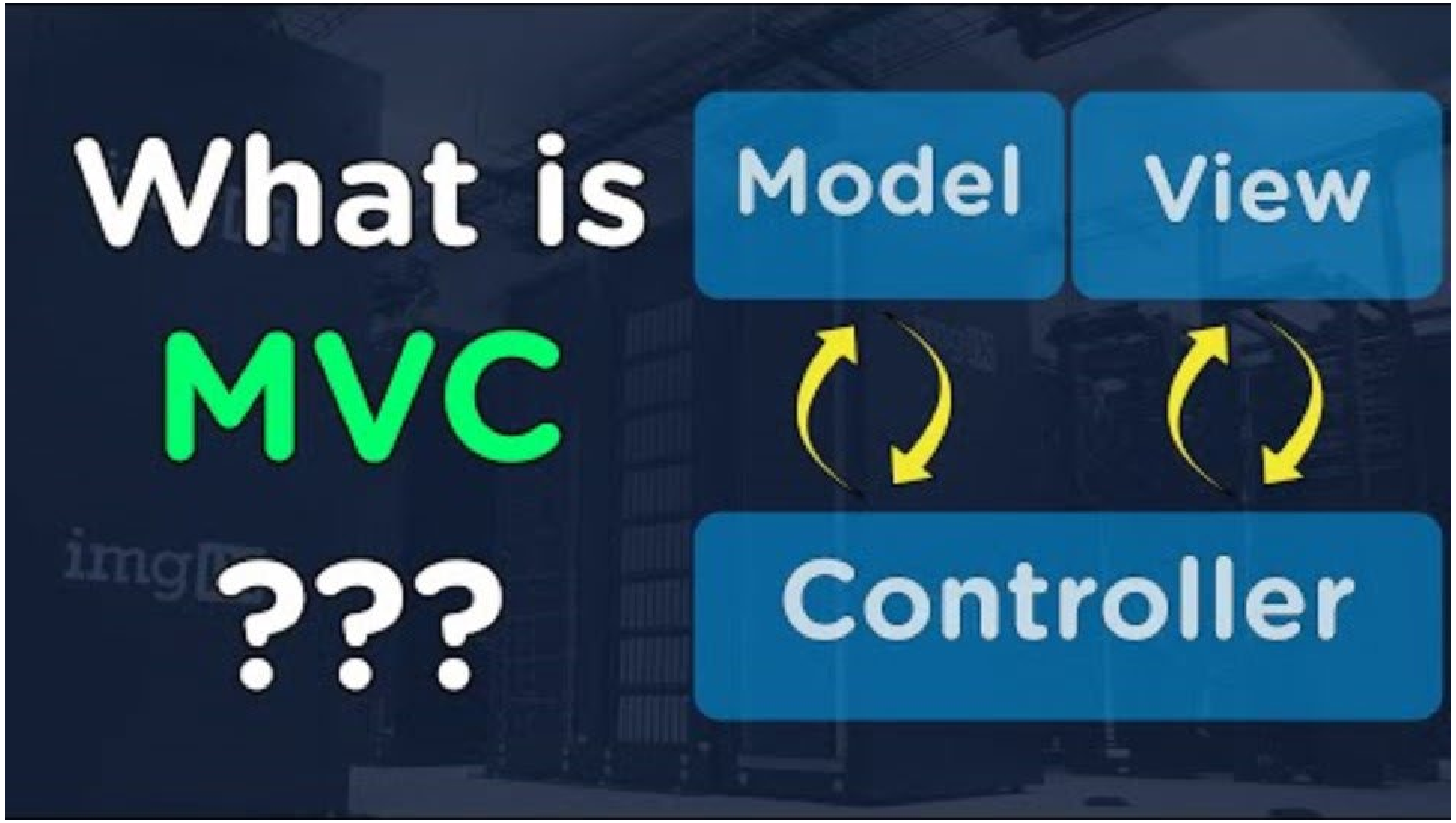
MVC Structure



MVC Architecture Design

https://www.geeksforgeeks.org/mvc-framework-introduction/

MVC Explained in 4 minutes



https://youtu.be/DUg2SWWK18I?si=o8ylGh-nTR5IIcVp&t=30

Template Engines

EMBEDDED JAVASCRIPT (EJS)

Template Engines

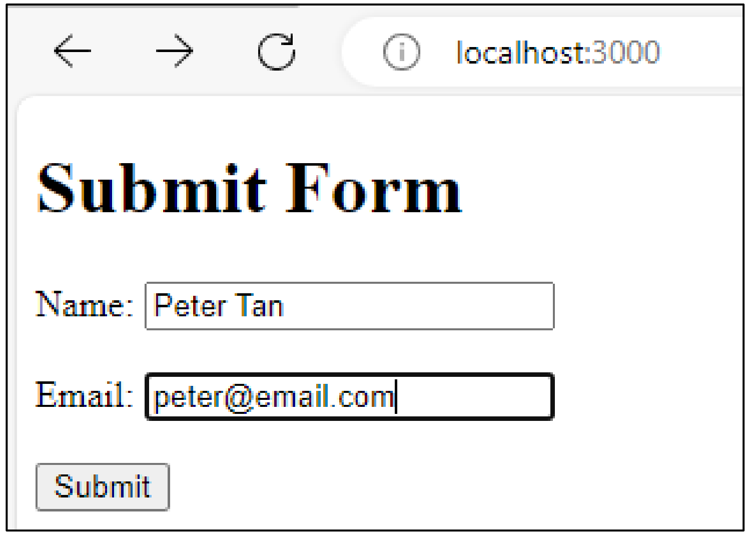
- Tools or libraries used in web development to generate HTML markup dynamically.
- Allow developers to create templates with placeholders for data and logic, which can then be filled in and processed to produce the final HTML output.
- Template engines offer several benefits, including code reusability, improved maintainability, and enhanced productivity.
- They simplify the process of generating dynamic content in web applications and help developers create more organized and readable code.
- There are many template engines used in web development:
  - EJS (Embedded Javascript)
  - Pug (formerly Jade)
  - Handlebars
  - Mustache

Embedded JavaScript (EJS)

- Widely used in web development for creating dynamic web pages, email templates, and other HTML-based content.
- EJS allows you to embed JavaScript code directly within HTML using **special tags**, making it easy to inject dynamic data into templates.
- The most common tags used are:
  - **<% %>:** These tags allow JavaScript code to run without outputting anything to the template. For example, you can use these tags to define variables or call functions.
  - **<%= %>:** These tags are used to output the result of a JavaScript expression to the template. For example, you can use these tags to insert data into the template.
  - **<%- %>:** These tags are used to output the result of a JavaScript expression to the template. However, unlike <%= and %> tags, they do not escape the output.
  - **<%# %>:** These tags add template comments.

Why EJS?

➢ Templating allows us to pass something from our server to be rendered in a HTML file.

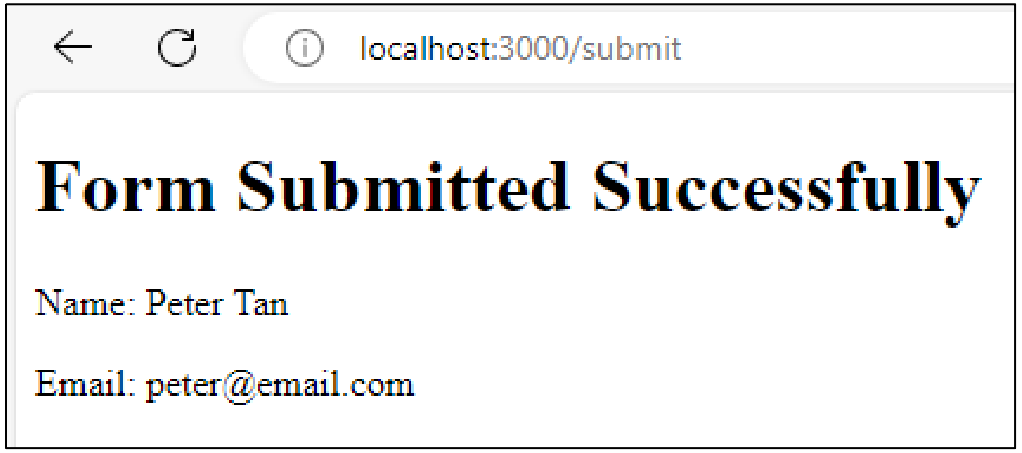


Submit Form

Name: Peter Tan

Email: peter@email.com

form.ejs



Form Submitted Successfully

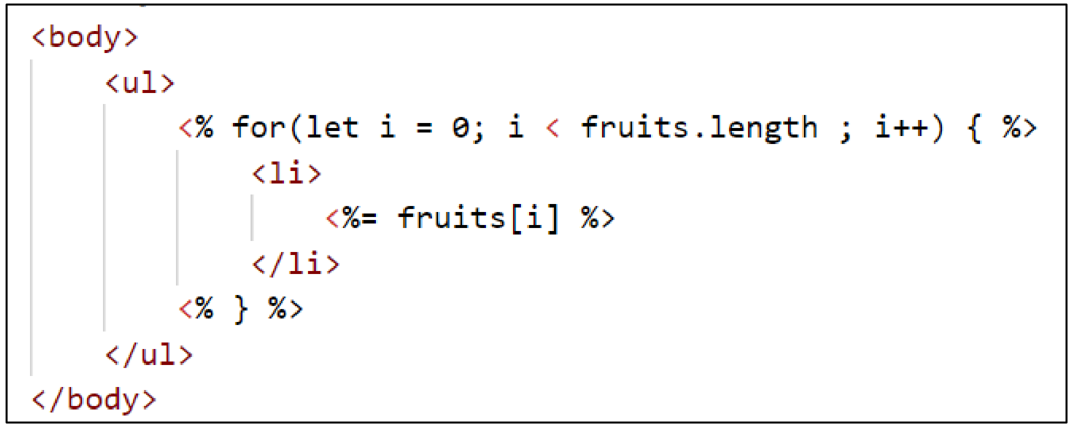
Name: Peter Tan

Email: peter@email.com

submitted.ejs

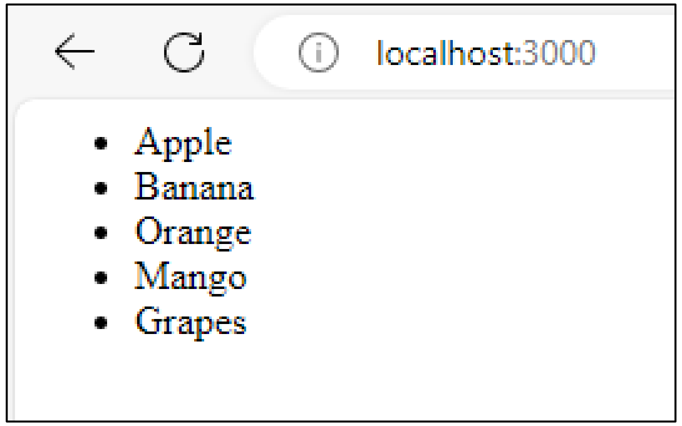
How does EJS work?

- It’s like having a little JavaScript module that can run JavaScript code inside a HTML file
- It ends with a .ejs file extension
- HTML but with bits of JavaScript enclosed inside a special syntax.
- Example, in this case, we are able to insert a for loop into our HTML file which loops through an array of fruits that’s been sent into this EJS file.



```
<body>
    <ul>
        <% for(let i = 0; i < fruits.length ; i++) { %>
            <li>
                <%= fruits[i] %>
            </li>
        <% } %>
    </ul>
</body>
```

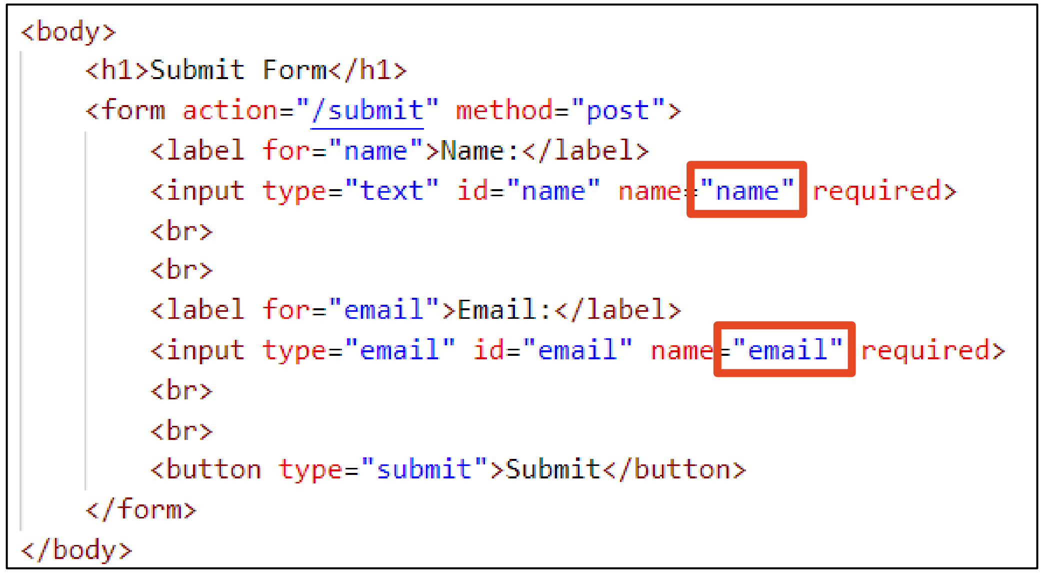
fruitsList.ejs



- Apple
- Banana
- Orange
- Mango
- Grapes

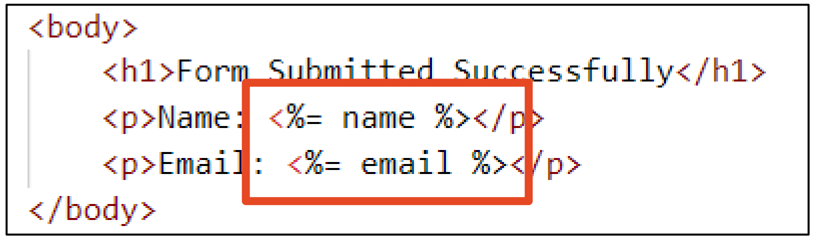
How does EJS work?

➢ Another example of how information entered in a form can be displayed after the form have been submitted.



```
<body>
    <h1>Submit Form</h1>
    <form action="/submit" method="post">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <br>
        <br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
        <br>
        <br>
        <button type="submit">Submit</button>
    </form>
</body>
```

form.ejs

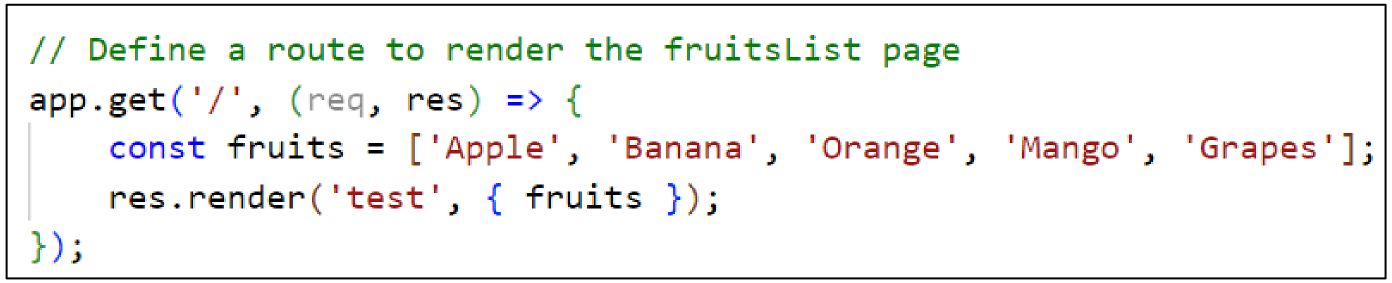


```
<body>
    <h1>Form Submitted Successfully</h1>
    <p>Name: <%= name %></p>
    <p>Email: <%= email %></p>
</body>
```

submitted.ejs

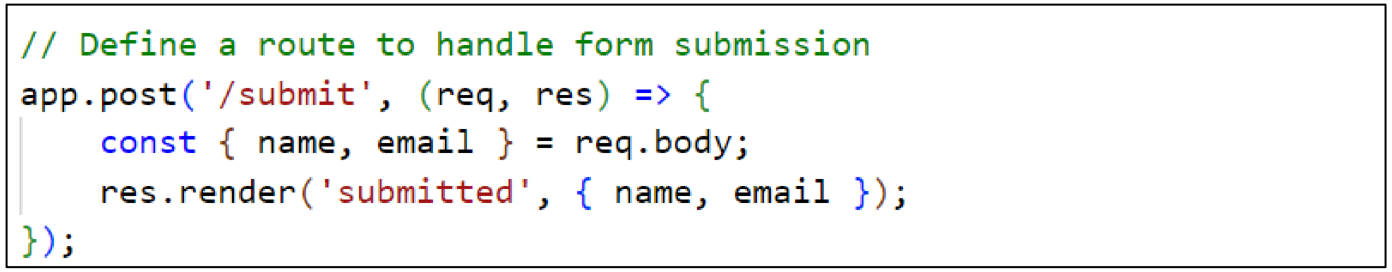
How is the data/information passed?

- Data/information can be passed using a method called **res.render()**.
- This will be done in the controller (app.js)
- Controller for fruitsList Page:



```
// Define a route to render the fruitsList page
app.get('/', (req, res) => {
    const fruits = ['Apple', 'Banana', 'Orange', 'Mango', 'Grapes'];
    res.render('test', { fruits });
});
```

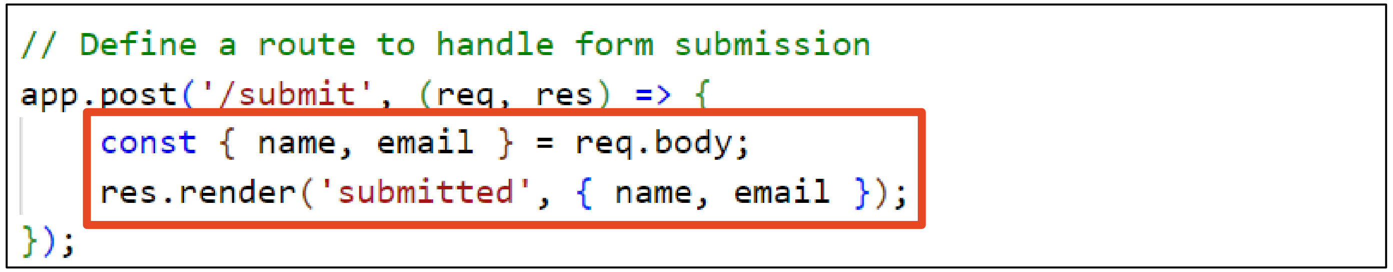
- Controller for form submission Page:



```
// Define a route to handle form submission
app.post('/submit', (req, res) => {
    const { name, email } = req.body;
    res.render('submitted', { name, email });
});
```

How is the data/information passed?

- Controller for form submission Page:



```
// Define a route to handle form submission
app.post('/submit', (req, res) => {
    const { name, email } = req.body;
    res.render('submitted', { name, email });
});
```

- In this case, we are rendering the submitted.ejs page
- An object with key of name and email (which came from the form input in the form page) will be passed over at the same time.

How is the data/information passed?

form opening tag

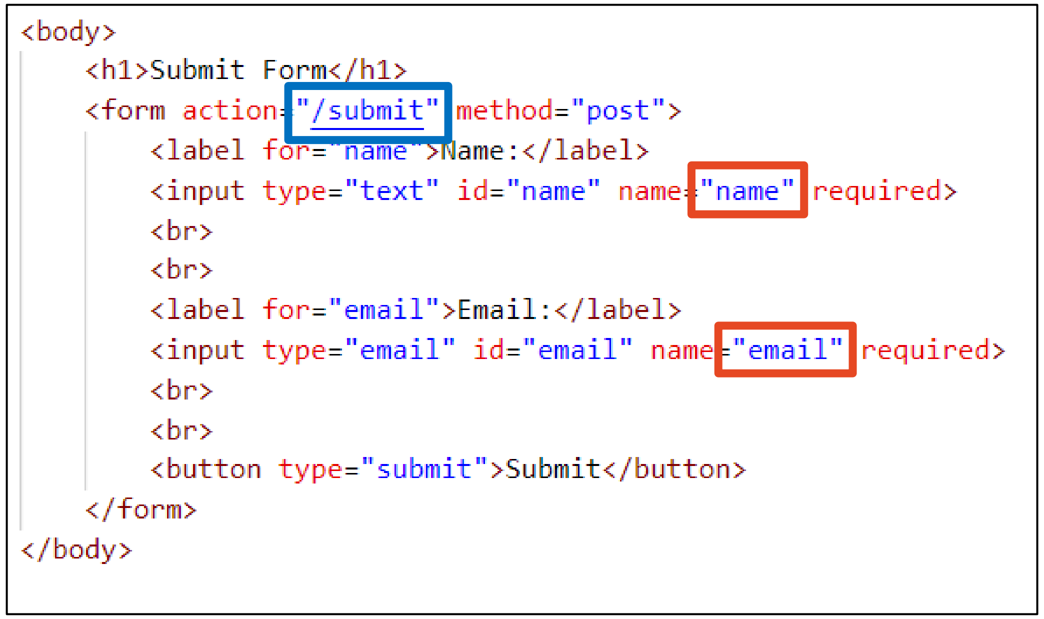
The action attribute determines the name of the page that will process the form data

The method attribute determines how the form data will be sent to the server. Here the method is POST.

```
<form action=“/endpoint" method="POST">

    // All the form contents go here

</form>
```

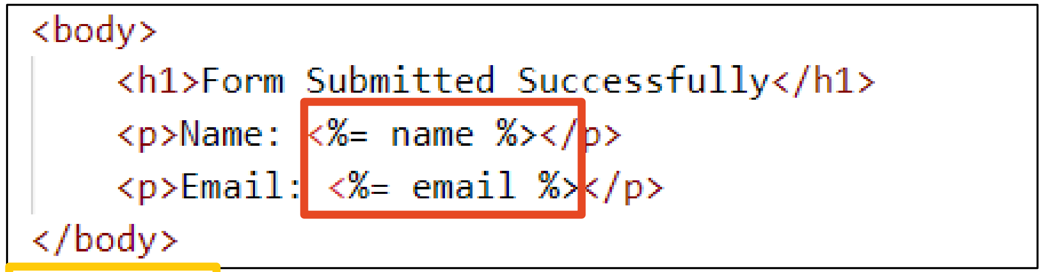
form closing tag – it ends the form

How is the data/information passed?



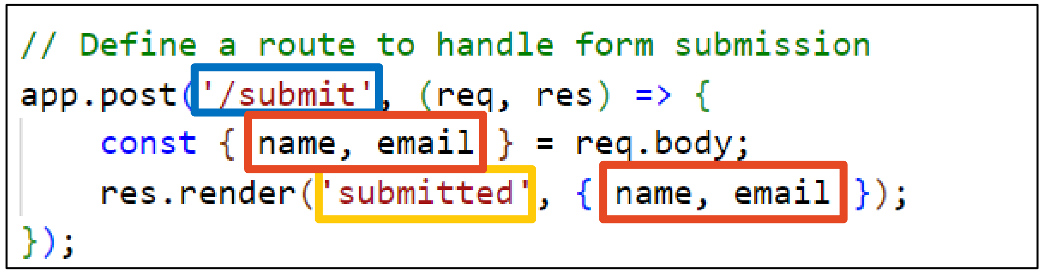
```
<body>
    <h1>Submit Form</h1>
    <form action="/submit" method="post">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <br>
        <br>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
        <br>
        <br>
        <button type="submit">Submit</button>
    </form>
</body>
```

form.ejs



```
<body>
    <h1>Form Submitted Successfully</h1>
    <p>Name: <%= name %></p>
    <p>Email: <%= email %></p>
</body>
```

submitted.ejs



```
// Define a route to handle form submission
app.post('/submit', (req, res) => {
    const { name, email } = req.body;
    res.render('submitted', { name, email });
});
```

app.js

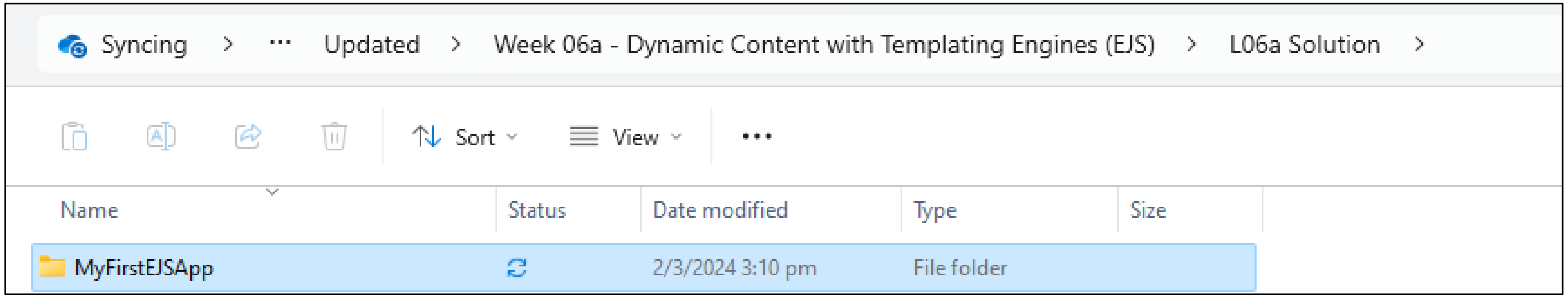
- Notice how the names (name and email) matches in the 3 files

EJS in Express Environment

SETTING UP EJS IN EXPRESS.JS

Setting up EJS in Express.js

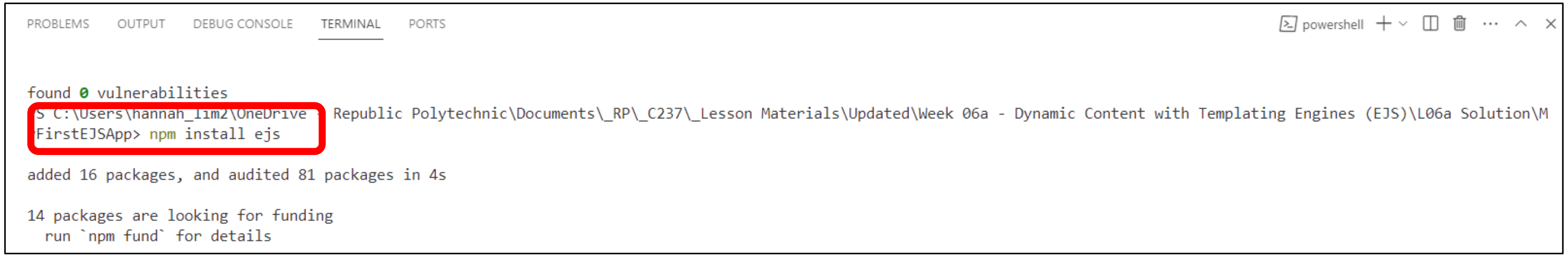
- **Open Visual Studio Code**
  - Launch VS Code on your laptop
- **Create a New Folder:**
  - Create a L06 folder in your C237 folder
  - Create a new folder (e.g. “MyFirstEJSApp”) in L06 Folder


Syncing > ⋯ Updated > Week 06a - Dynamic Content with Templating Engines (EJS) > L06a Solution >
Sort
View
⋯

Name	Status	Date modified	Type	Size
MyFirstEJSApp		2/3/2024 3:10 pm	File folder	

Setting up EJS in Express.js

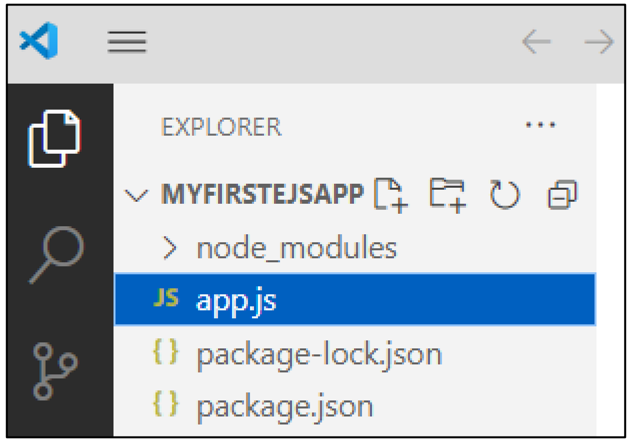
- **Open the folder which you just created in VS Code:**
  - Open VS Code and use the "File" menu to open your newly created folder.
  - Open a terminal within VS Code (Terminal > New Terminal).
  - Run the following command to create a package.json file for your project:
    - npm init –y
    - npm install express
    - **npm install ejs**



```
found 0 vulnerabilities
S C:\Users\hannah_lim2\OneDrive - Republic Polytechnic\Documents\_RP\_C237\_Lesson Materials\Updated\Week 06a - Dynamic Content with Templating Engines (EJS)\L06a Solution\M
FirstEJSApp> npm install ejs

added 16 packages, and audited 81 packages in 4s

14 packages are looking for funding
  run `npm fund` for details
```

Setting up EJS in Express.js

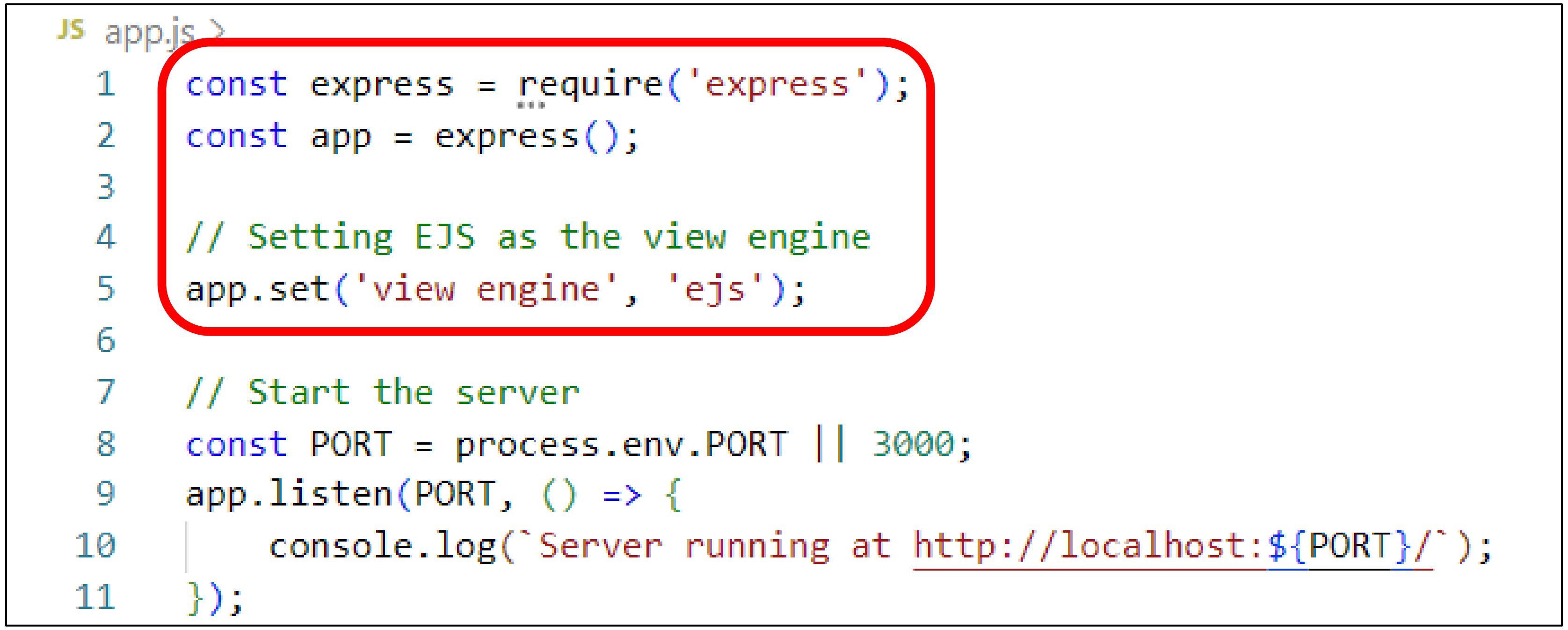
- Inside the ‘MyFirstEJSApp’ folder, create a new file with a ‘.js’ extension (e.g. app.js).



- Double click on the app.js file in VS Code to open the file.

Setting up EJS in Express.js

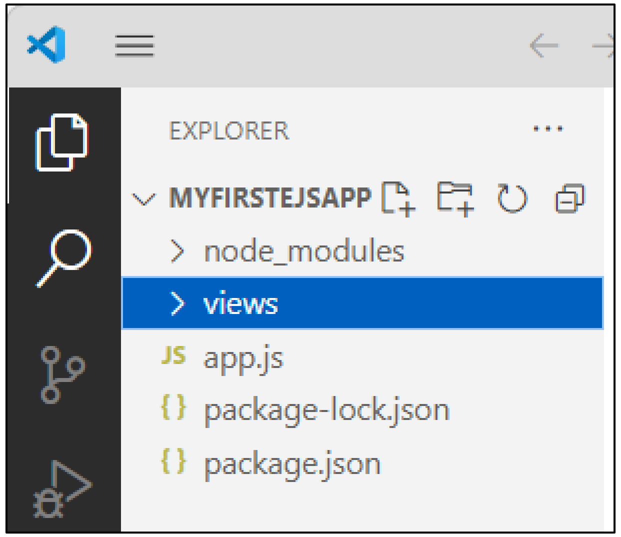
- To set up EJS in Express.js, you will require an **express** module to initialise an Express.js app.
- Then, set **ejs** as the view engine using the **app.set()** method:



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

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

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server running at http://localhost:${PORT}/`);
});
```

Setting up EJS in Express.js

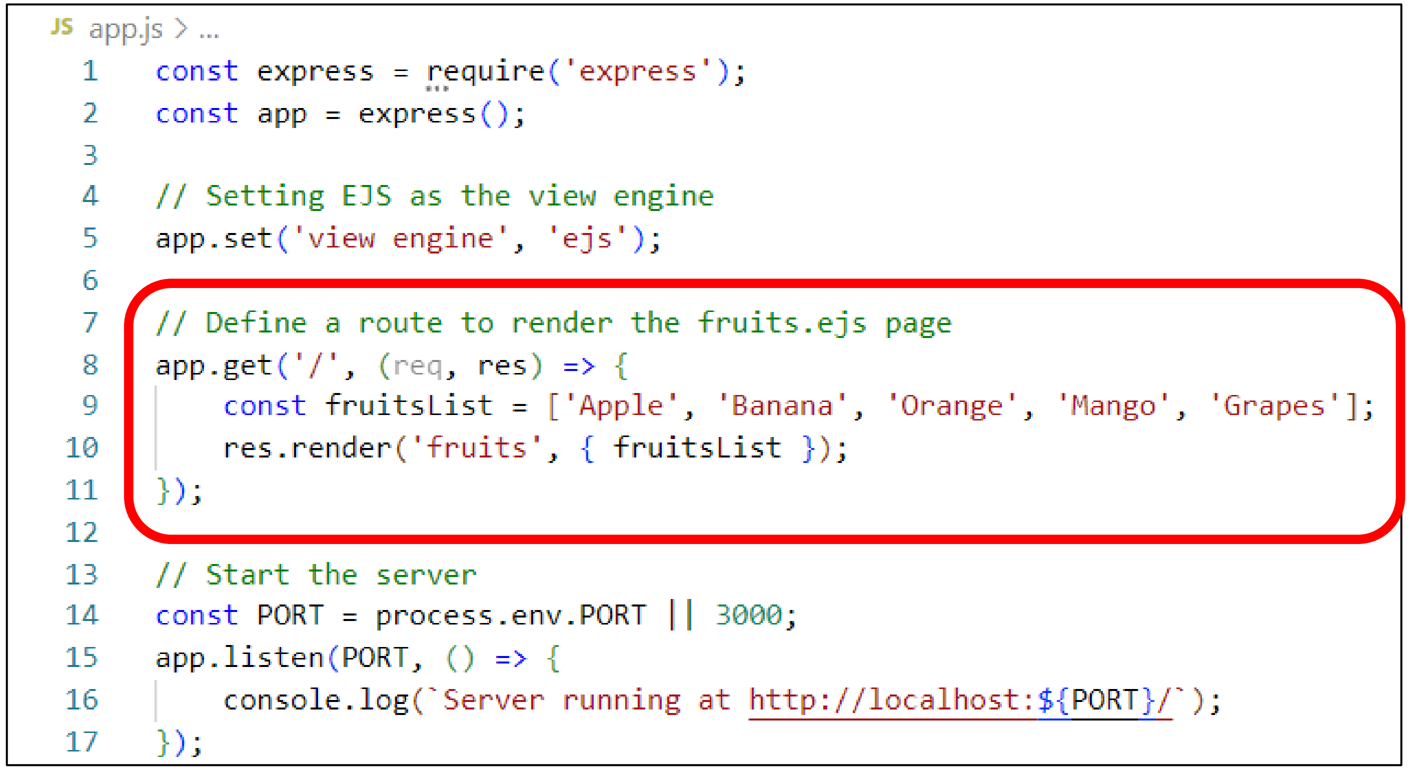
- After setting the **view engine** to **ejs**, you can now create a new folder in the root directory of your project called **views** and add your EJS templates there.



- It's important to remember that EJS always looks for templates in the **views directory**. As a result, it is critical that you always keep your templates in this directory so that the EJS engine can find and process them.

Setting up EJS in Express.js

- A fruits.ejs page in your views folder will be rendered using the following code in a route:



app.js > ...

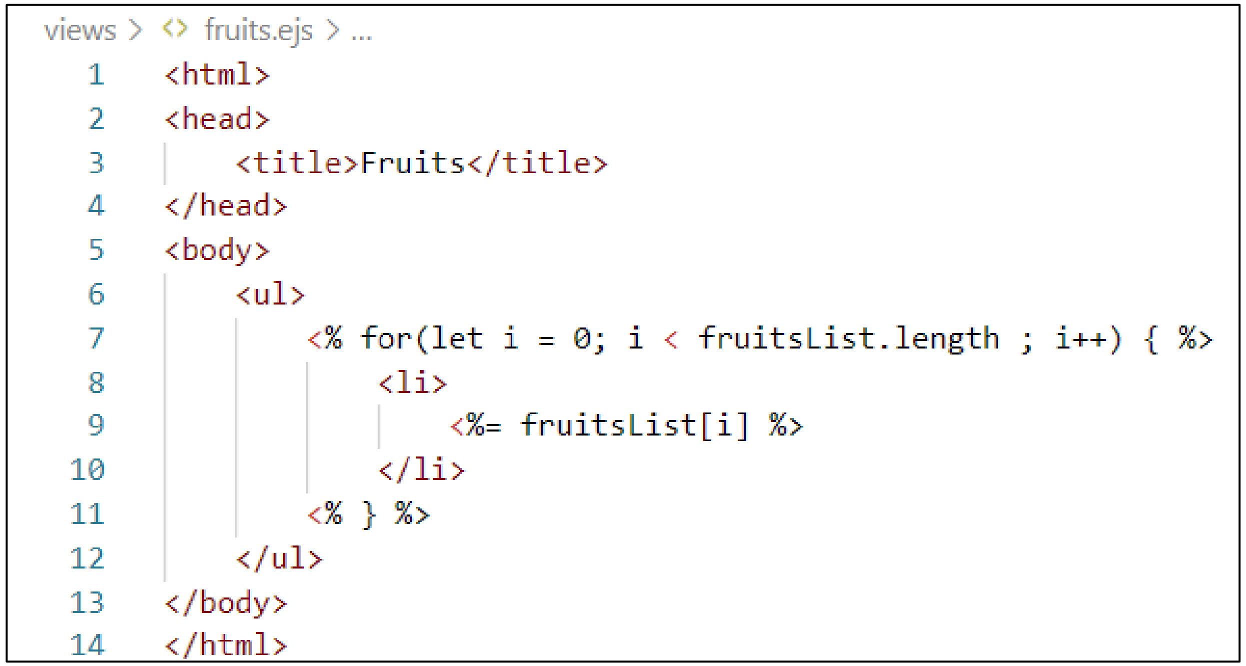
```
const express = require('express');
const app = express();

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

// Define a route to render the fruits.ejs page
app.get('/', (req, res) => {
    const fruitsList = ['Apple', 'Banana', 'Orange', 'Mango', 'Grapes'];
    res.render('fruits', { fruitsList });
});

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
    console.log(`Server running at http://localhost:${PORT}/`);
});
```

The **render** function, provided by Express, will look for a template with the same name as the first argument (‘**fruits**’ in this example) you pass to it in the **views folder** and renders it when the root route is accessed using the view engine you set up earlier.

- Add the highlighted code in app.js

Rendering an ejs file

- **Write your first ejs file:**
  - **In the views folder**, create a new file named **fruits.ejs**.

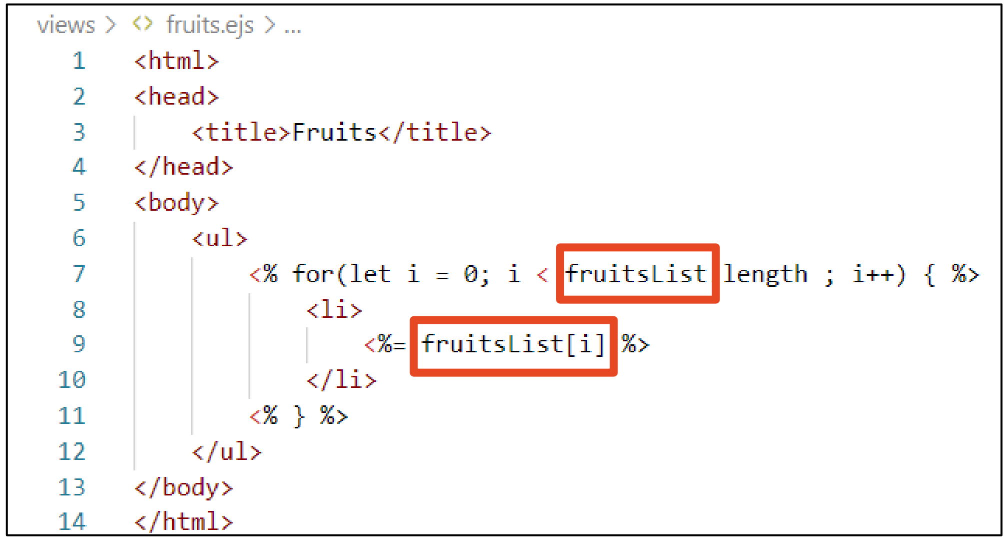


views > fruits.ejs > ...

```
<html>
<head>
    <title>Fruits</title>
</head>
<body>
    <ul>
        <% for(let i = 0; i < fruitsList.length ; i++) { %>
            <li>
                <%= fruitsList[i] %>
            </li>
        <% } %>
    </ul>
</body>
</html>
```

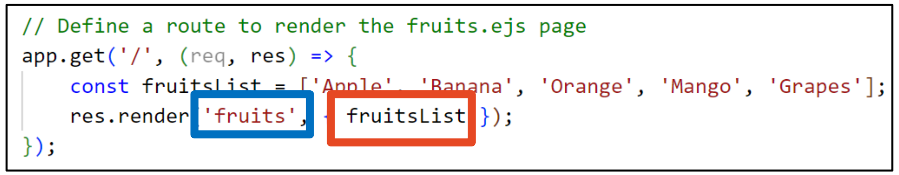
  - Save the file

How are the 2 files linked?



```
views > <> fruits.ejs > ...
<html>
<head>
    <title>Fruits</title>
</head>
<body>
    <ul>
        <% for(let i = 0; i < fruitsList.length ; i++) { %>
            <li>
                <%= fruitsList[i] %>
            </li>
        <% } %>
    </ul>
</body>
</html>
```

fruits.ejs



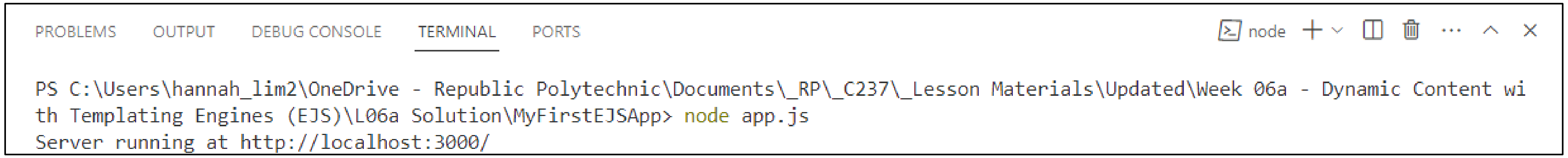
```
// Define a route to render the fruits.ejs page
app.get('/', (req, res) => {
    const fruitsList = ['Apple', 'Banana', 'Orange', 'Mango', 'Grapes'];
    res.render('fruits', { fruitsList });
});
```

app.js

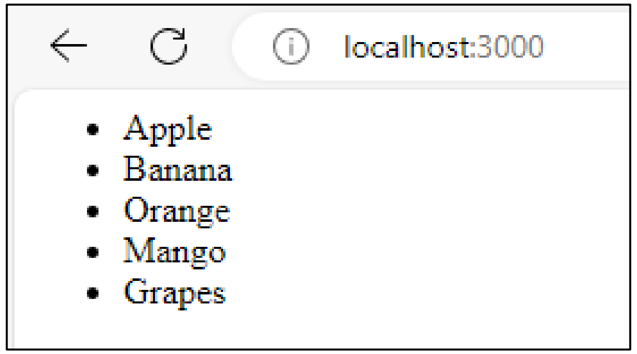
- Notice how the array name and file name matches in the 2 files
- Try changing the file name in res.render to fruits1.ejs. Run the file and see what happens?

Run your EJS in Express.js app

- Open the terminal in VS Code
- Run your app by typing the following command in your terminal: **node app.js**
- You should see the message "Server running at http://localhost:3000/"



```
PS C:\Users\hannah_lim2\OneDrive - Republic Polytechnic\Documents\_RP\_C237\_Lesson Materials\Updated\Week 06a - Dynamic Content wi
th Templating Engines (EJS)\L06a Solution\MyFirstEJSApp> node app.js
Server running at http://localhost:3000/
```

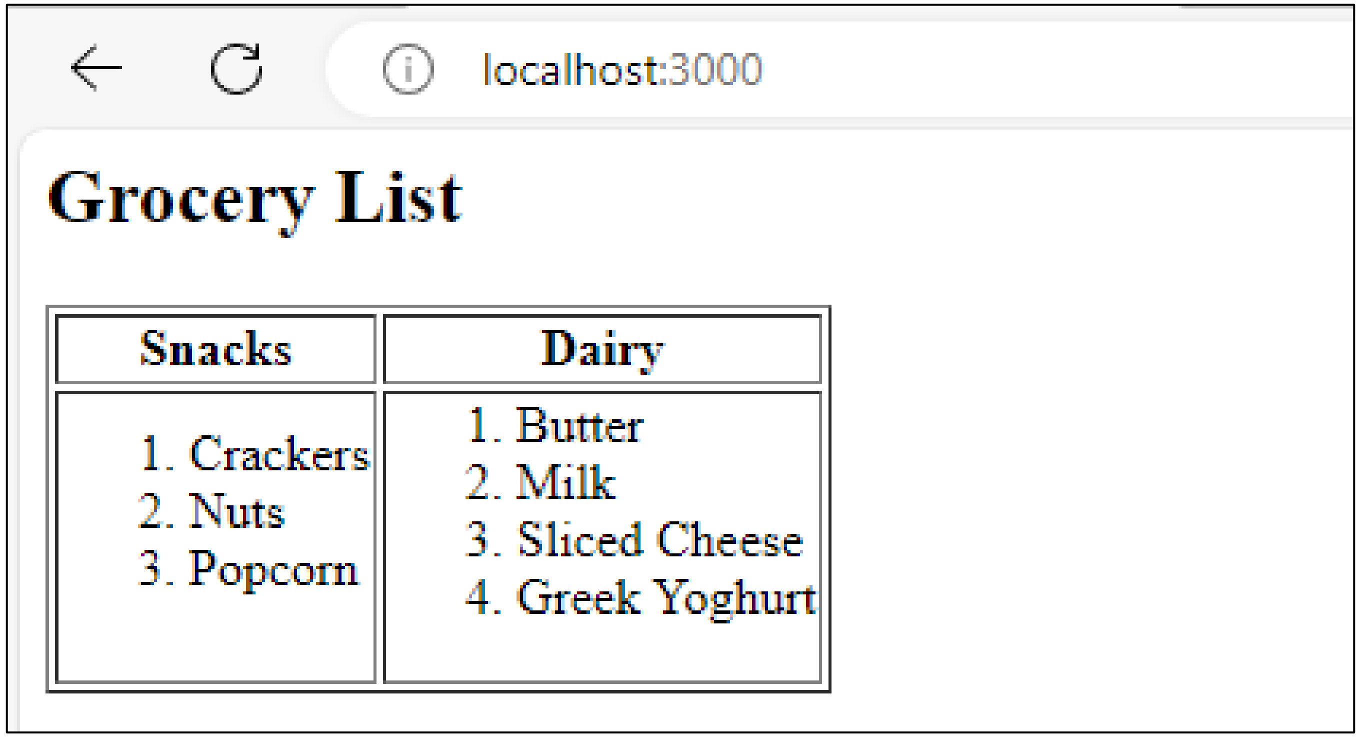


- Apple
- Banana
- Orange
- Mango
- Grapes

L06a Activity

- In your L06 folder, create a **GroceriesApp** Express App with an EJS template, that displays a **groceries list** page. The page should include **two lists**: **one for snacks and another for dairy products.** Write the code in app.js to pass these lists to the EJS template and render them on the page.
- Your task is to:
  - Set up a ExpressJS project.
  - Create two arrays: **snacksList and dairyList**, containing some sample items for each category.
  - **Configure an Express route to render an EJS template** that displays both lists on the grocery list page.
  - Pass the **snacksList** and **dairyList** arrays as variables to the EJS template.
  - Render the grocery list page with the two lists displayed in separate columns using a table.
- Refer to next slide for sample screenshot

L06a Activity – Sample Screenshot


localhost:3000

Grocery List

Snacks	Dairy
1. Crackers 2. Nuts 3. Popcorn	1. Butter 2. Milk 3. Sliced Cheese 4. Greek Yoghurt

What have you learnt?

- Model View Controller (MVC)
  - What is MVC
  - MVC Structure
- Template Engines – Embedded JavaScript (EJS)
  - Why EJS?
  - How does EJS work?
  - Setting up EJS in Express Environment