

Adding HTML to your Express server

Welcome to this lab activity

In this lab activity, you will explore how to add HTML pages to your Express web application. Imagine that you would like to replace the 'Hello World!' message on your previous lab activity with a long text or HTML. Including HTML syntax in your web server is not good practice as it will lead to messy code and poor maintainability. Let's see how you can separate HTML code and use Express to serve and display your pages.

The EJS templating language

Later in this lab activity, you will install and use the EJS templating language to build HTML pages. 'EJS' means Embedded JavaScript Templating and it is a language that lets you generate HTML markup with plain JavaScript. With EJS, you can generate HTML pages and make them dynamic. For this activity, you will explore how to use EJS to render static pages.

Task 1: Create a new Express project

Create a new project in Visual Studio Code. You will follow the steps from the last lab to create a Node.js/NPM/Express project:

1. Create a new folder called `06_html_express` on your computer.
2. Open the folder in Visual Studio Code.
3. Open a Terminal pane in Visual Studio Code.
4. Run the following command in the Terminal pane:

```
npm init
```

Just hit enter to accept the default answers.

5. Run the following command in the Terminal pane:

```
npm install express
```


This command will install the **Express** framework within your working directory so that you can use it.

Task 2: Organise the route handlers

In the previous lab, you added some route handlers to `index.js`:

```
JS index.js X
JS index.js > ...
1  const express = require("express");
2
3  const app = express();
4  const port = 8081;
5
6  app.get("/", (req, res) => res.send("Hello World!"));
7
8  app.get("/about", (req, res) => res.send("<h1>This is the about page</h1>"));
9
10 app.listen(port,
11   () => console.log(`Node server is running on port ${port}...`));
12
```

As your application grows, you will have more route handlers. It makes sense to organise them so that you can split them out into separate files. The usual way to do this in an Express app is to move the route handler code into one or more code files in a `routes` subfolder and then to **require** the code files in `index.js`. Let's do that now. You will create the same application that you made in the last lab, but with the route handlers in a separate file in the `routes` subfolder.

In Visual Studio Code, you can create a subdirectory in your project by clicking the New Folder  icon, which is next to the new file icon in the explorer pane.

6. Click the New Folder icon and create a new folder called `routes`.
7. In the `routes` folder, create a new file called `main.js`.
8. Add the following code to `main.js`:

```
// Create a new router
const express = require("express");
const router = express.Router();

// Handle the main routes

router.get("/", (req, res) => res.send("Hello World!"));

router.get("/about", (req, res) => res.send("<h1>This is the about
page</h1>"));

// Export the router object so index.js can access it
module.exports = router;
```

Note that the route handlers for `/` and `/about` are now located in `main.js` rather than in `index.js`.

9. In the main folder, create a new file called `index.js`.

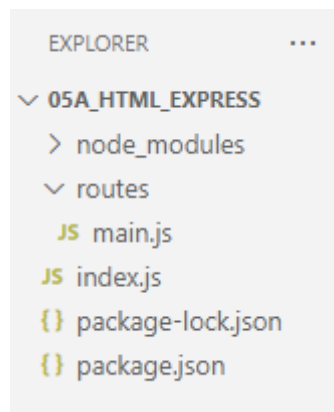
10. Add the following code to `index.js`:

```
// Set up express
const express = require ("express");
const app = express();
const port = 8081;

// Load the route handlers
const mainRoutes = require("../routes/main");
app.use('/', mainRoutes);

// Start listening for HTTP requests
app.listen(port,
  () => console.log(`Node server is running on port ${port}...`));
```

At this stage, your Visual Studio Code project should look like this:



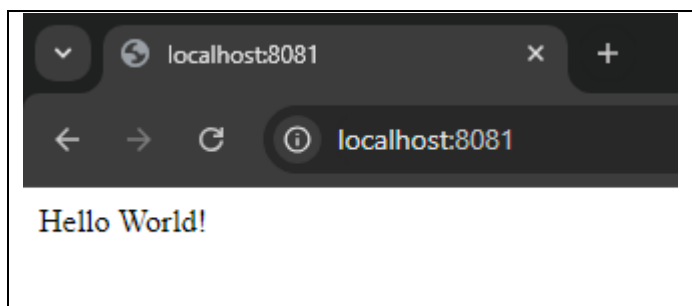
11. Run the `index.js` file with the following Terminal command:

```
node index.js
```

12. Browse to the two routes:

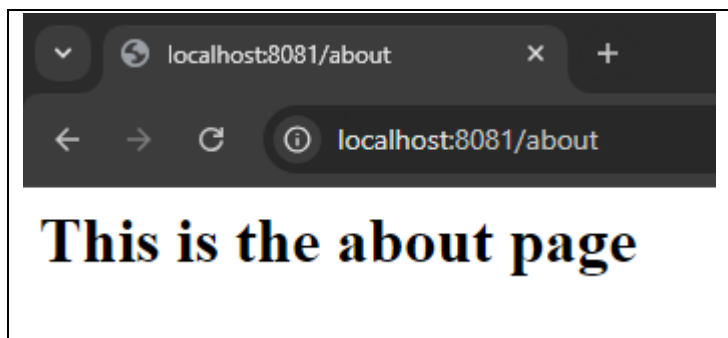
```
http://localhost:8081/
```

This should show the following:



```
http://localhost:8081/about
```

This should show the following:



This is the same app that we had in the last lab, but we have moved the route handlers into a separate code file for better organisation. Seasoned coders will recognise this as an example of **refactoring**!

Task 3: Install EJS

In the current code, the HTML is embedded in the JavaScript code's `res.send()` call. As the HTML you want to render gets more complex, this situation will soon become unmanageable. Let's do some more refactoring, moving the HTML code out into separate HTML files and making the HTML a bit more interesting. For this task, you will use the EJS templating engine. The true power of EJS will become apparent in the next lab, but for now use it to render static HTML pages.

First, let's install the EJS module.

13. Run the following command in the Terminal pane:

```
npm install ejs
```

Once you run the above Terminal commands, your `package.json` file should contain both the **Express** and **EJS** modules:

```
{ } package.json ×
{ } package.json > ...
1  {
2    "name": "05_html_express",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "",
10   "license": "ISC",
11   "dependencies": {
12     "ejs": "^3.1.10",
13     "express": "^4.19.2"
14   }
15 }
16 |
```

Task 4: Tell Express that we are using EJS

Express can utilise different rendering/templating engines for rendering HTML pages. You need to tell Express that you want to use EJS.

- 14.** Add the following lines to `index.js` before the loading of the route handlers:

```
// Set the rendering engine for Express to EJS
app.set("view engine", "ejs");
```

Task 5: Add the HTML pages

Let's create a folder to hold all the HTML pages. You will call this subfolder `views`.

- 15.** In Visual Studio Code, click the New Folder icon and create a new folder called `views`.

Now add an HTML file for the main page.

- 16.** Add a file called `index.ejs` to the `views` folder.

- 17.** Add the following content to `index.ejs`:

```
<!doctype html>
<html>
  <head>
    <title>This is the title of the webpage!</title>
  </head>
```

```
<body>
  <h1> This is home page </h1>
  <p>Homepage This is an example paragraph. Anything in the
<strong>body</strong> tag will appear on the page, just like this
<strong>p</strong> tag and its contents.</p>
</body>
</html>
```

As you can see, this is just HTML.

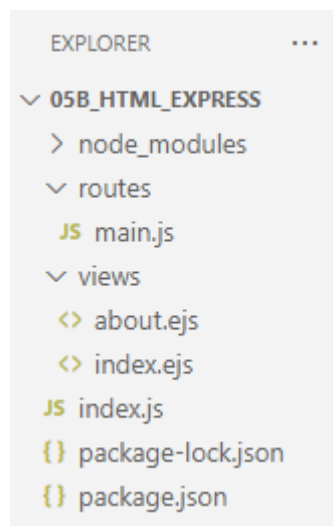
Now add an HTML for the about page.

18. Add a file called `about.ejs` to the views folder.

19. Add the following content to `about.ejs`:

```
<!doctype html>
<html>
  <head>
    <title>This is the about page title!</title>
  </head>
  <body>
    <h1> This is the about page </h1>
  </body>
</html>
```

At this point, your project structure should look like this:



Task 7: Update the route handler to render the HTML pages

Now you have some nice HTML pages in the `.ejs` files, but you need the route handler code to tell Express to render the files using the EJS template engine.

20. Modify main.js, replacing the res.send() calls with calls to res.render():

```
// Create a new router
const express = require("express");
const router = express.Router();

// Handle the main routes

router.get("/", (req, res) => {
  res.render("index.ejs")
});

router.get("/about", (req, res) => {
  res.render("about.ejs")
});

// Export the router object so index.js can access it
module.exports = router;
```

Task 6: Run the code

21. Run the index.js file with the following Terminal command:

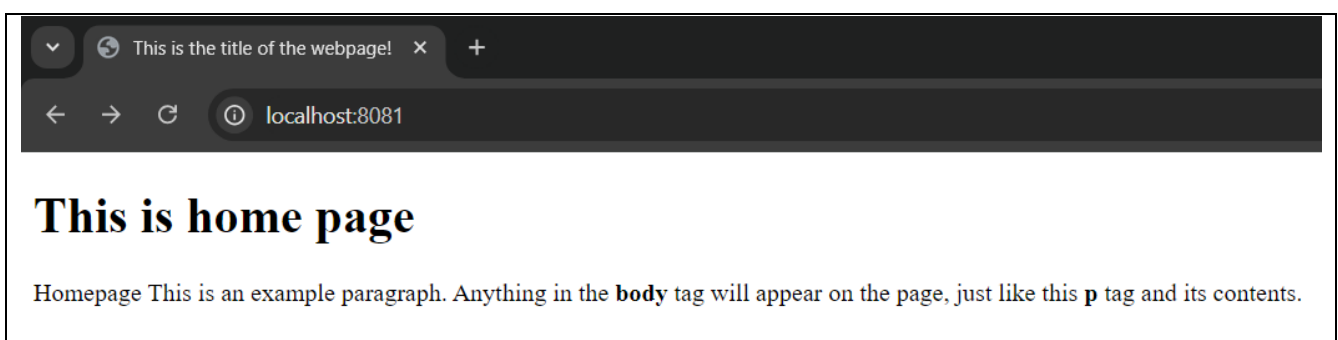
```
node index.js
```

The above command will start a web server running on port 8081.

22. Browse to the two routes:

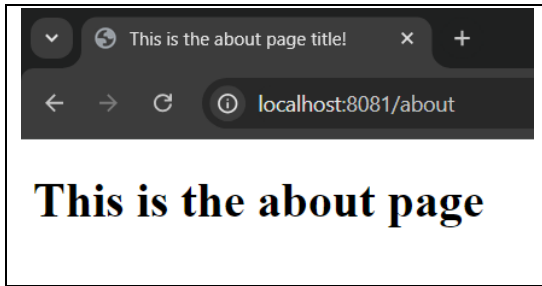
```
http://localhost:8081/
```

This should show the following:



```
http://localhost:8081/about
```

This should show the following:



Task 7: Explore further

When tackling these lab activities, it's always good to stretch yourself by doing some research and attempting some changes on your own.

Can you add another route for a contact page, rendering an HTML file? Think about where you need to place the HTML file and how you need to modify your middleware code to render it.

End of lab

Congratulations on completing this lab. There was quite a lot for you to do! You've refactored your code to make it more manageable as it grows and have set up the foundations for the next labs.

In the next lab activity, you will explore how to make your HTML pages dynamic.