Adding HTML to your Express server

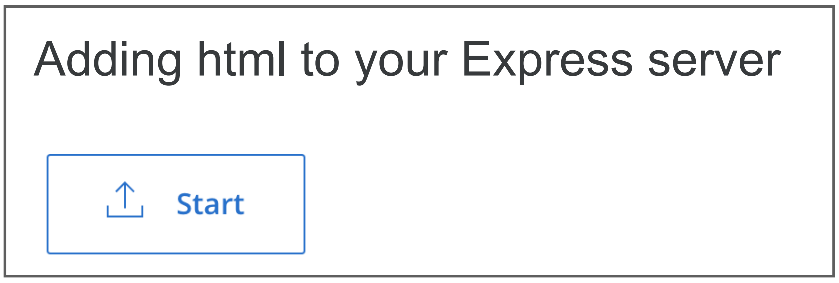
Welcome to this Lab activity

In this lab activity you will be exploring how to add html pages to your Express web application. Imagine you would like to replace the "Hello World!" message on your previous lab activity root route "/" with a long text or html. Including html syntax in your web server is not a 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!

Start the Lab environment application

It is simple to launch a lab exercise. You only need to click on the button "Start" below the activity title to enter a lab environment.

Let's explore this lab activity. Go ahead and click on the "Start" button!



The EJS templating language

Later in this lab activity you will be installing and using the EJS templating language to build html pages. EJS stands for Embedded Javascript Templating and it is a language that lets you generate html markup with plain javascript. With EJS you cannot only generate html pages but you can also make them dynamic. For the purpose of this activity you will explore how EJS is used to render static pages.

Task 1: Build a new Express web server with an html file for each route

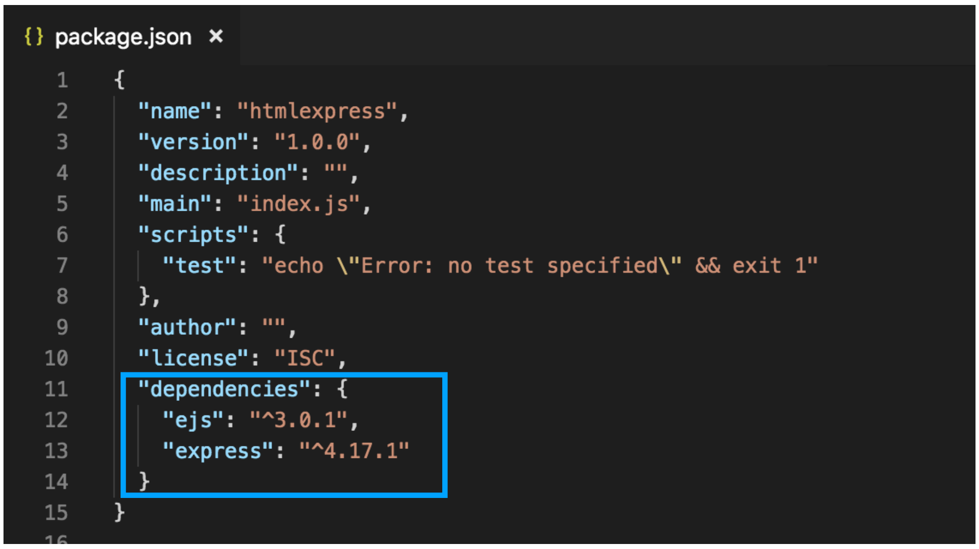
The folder structure has already been partially constructed for you and organised into different topics. For the purpose of this lab, you will be making changes inside the *topic3* folder structure. Let's get started!

Please do not delete or move any existing folders/files inside the lab environment.

Use the Visual Studio Code Terminal and run the following commands:

- **cd topic3/htmlExpress**: type this command and press *Enter*. This command will change your working directory to be the *htmlExpress* folder.
- **npm init**: type this command and press *Enter*. This command will set up a new or existing npm package. You can skip all the npm initialisation questions by pressing *Enter.* Once the npm package file has been created, you can view it inside the *htmlExpress* directory (*package.json*). The *package.json* file contains a set of information regarding your current node project.
- **npm install express --save**: type this command and press *Enter*. This command will install the *Express* framework within your working directory so that it will be available for you to use.
- **npm install ejs --save**: type this command and press *Enter*. This command will install the *EJS* framework within your working directory so that it will be available for you to use.

Once you run the above Terminal commands, your *package.json* file located inside the *topic3/ htmlExpress* folder should contain both the **Express** and **EJS** modules:



```
{
  "name": "htmlexpress",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "ejs": "^3.0.1",
    "express": "^4.17.1"
  }
}
```

The very next step is to add some code to the *index.js* file located inside the *topic3/htmlExpress* folder in order to create your web server.

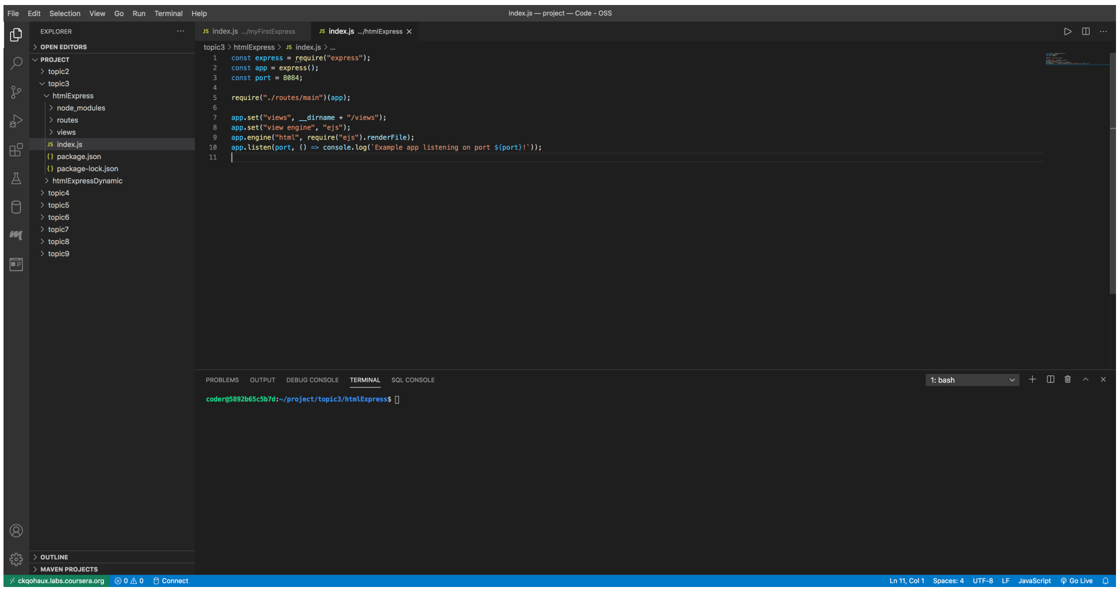
Add the following code to the *index.js* file and remember to save it:

```
const express = require ("express");
const app = express();
const port = 8084;

require("./routes/main")(app);

app.set("views",__dirname + "/views");
app.set("view engine", "ejs");
app.engine("html", require("ejs").renderFile);
app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

If you have correctly followed all the above steps, your environment should be similar to the one below:


File Edit Selection View Go Run Terminal Help
index.js — project — Code - OSS
EXPLORER
OPEN EDITORS
PROJECT
topic2
topic3
htmlExpress
node_modules
routes
views
index.js
package.json
package-lock.json
htmlExpressDynamic
topic4
topic5
topic6
topic7
topic8
topic9
OUTLINE
MAVEN PROJECTS
index.js .../myFirstExpress
index.js .../htmlExpress
topic3 > htmlExpress > index.js > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
1: bash
ckqohaux.labs.coursera.org
Connect
Ln 11, Col 1 Spaces: 4 UTF-8 LF JavaScript Go Live

```
const express = require("express");
const app = express();
const port = 8084;

require("./routes/main")(app);

app.set("views", __dirname + "/views");
app.set("view engine", "ejs");
app.engine("html", require("ejs").renderFile);
app.listen(port, () => console.log(`Example app listening on port ${port}!`));
```

```
coder@5892b65c5b7d:~/project/topic3/htmlExpress$
```

Questions

- What are the differences between this Express web server and the one you have created in your previous lab activity?
- You can include html code inside your server file as you did before, but what do you think happens when you include many html lines of code?
  Therefore, it is not advised to include html inside your server code. We create separate html files and we use the EJS templating engine to handle these html files.

It is better to organise your web application in separate parts instead of including all the code in one file, because your web application is going to grow in size and become bigger and bigger.

That is why four lines of code have been added to your web server code:

```
require("./routes/main")(app);
```

The above line requires the *main.js* file inside the *topic3/htmlExpress/routes* folder passing in the Express app as an argument. You will add all the routes to this file later.

```
app.set("views",__dirname + "/views");
```

The above line sets the path to the *topic3/htmlExpress/views* folder for the EJS engine. You will add all the html pages to this directory later.

```
app.set("view engine", "ejs");
```

The above line tells Express that you want to use EJS as the templating engine for this application.

```
app.engine("html", require(“ejs").renderFile);
```

The above line tells Express that you want to render html pages.

Now add the missing parts to both the *topic3/htmlExpress/routes and topic3/htmlExpress/views* folders to have your web application serve html pages.

Create two html files inside the *topic3/htmlExpress/views* folder called *index.html* and *search.html* and add the following html content:

Add the following to the *index.html* file:

```
<!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>
```

Add the following to the *search.html* file:

```
<!doctype html>
<html>
  <head>
    <title>This is the search page title!</title>
  </head>
  <body>
    <h1> This is the search page </h1>
    <p>This is an example of a paragraph inside the search page</p>
  </body>
</html>
```

At this point you have created all the html pages that you want to serve using the Express framework. There is only one missing part for your application to work correctly. You need to add the routes inside the *topic3/htmlExpress/routes/main.js* file.

Add the following code to the *main.js* file inside the *topic3/htmlExpress/routes/* folder:

```
module.exports = function(app) {

  app.get("/",function(req, res){
    res.render("index.html")
  });

  app.get("/search",function(req, res) {
    res.render("search.html");
  });

}
```

Now that you have all the routes set up and exported correctly, your application is complete and can run html pages.

Running the index.js file

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

- **node index.js**: type this command and press Enter. The node command, followed by the file name, tells Node.js to execute the content of the file.

The above command will start a web server running on port 8084.

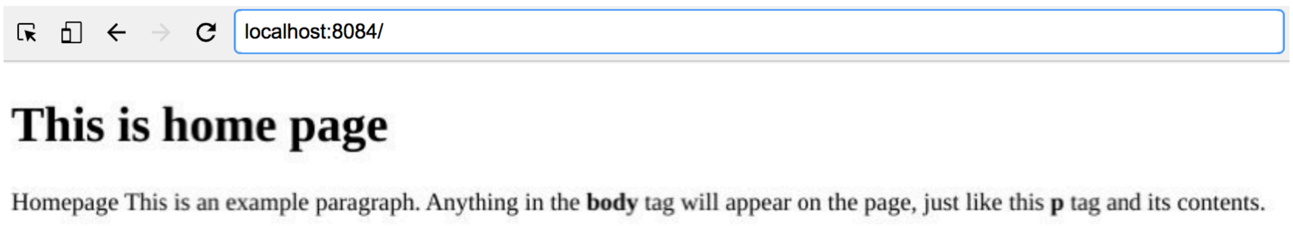
Task 2: Access your server via HTTP

Now that your code is running, serving your application on port 8084, you can access your web page from a browser!

Use the “Browser Preview” plugin to visualise your web application.

If you do not remember how to use the “Browser Preview” plugin, please refer to the “Creating my first Node.js web server” lab instructions.

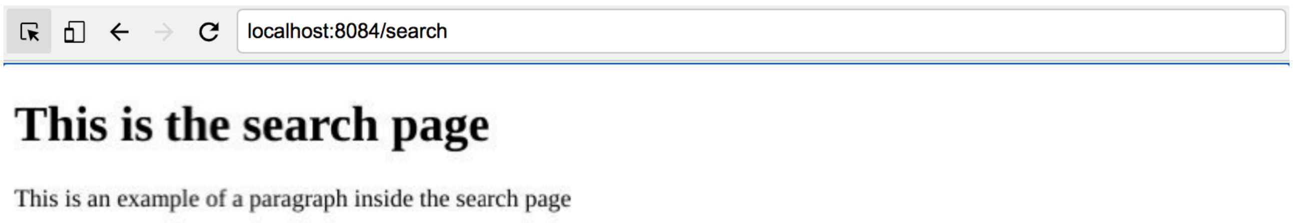
Type *localhost:8084* on the “Browser Preview” tab and press *Enter* on your keyboard to visualise your web application “root” page:


localhost:8084/

This is home page

Homepage This is an example paragraph. Anything in the **body** tag will appear on the page, just like this **p** tag and its contents.

Type *localhost:8084/search* on the “Browser Preview” tab and press *Enter* on your keyboard to visualise your web application “search” page:


localhost:8084/search

This is the search page

This is an example of a paragraph inside the search page

As you can see you app has now two routes; the root route “/” which renders the *index.html* page and the search route which renders the *search.html* page.

End of Section

Congratulations for completing this section. As long as you have saved your work, your files will remain when you close this lab activity so do not worry about losing your data. You have successfully created a web application that serves html files according to your routes. In the next lab activity you will explore how to make your html pages dynamic.