My first Express web server

Welcome to this lab activity

In this lab activity, you will explore a very popular web application framework called **Express.** You will build the same application that you built before, but using the Express framework rather than raw Node.js.

What is Express?

Express is a very popular framework for Node.js designed for building web applications and APIs. Think of it as a module that you can require in your code and use it to create and maintain web servers. It is considered as the standard server framework for Node.js.

So far, you have been using the **http** module to create web servers and, although the code was not really complicated to write, **Express** offers a simpler code structure, which you can extend through additional modules. Let's get started!

Task 1: Create a new Node.js/NPM project

Create a new project in Visual Studio Code.

1. Create a new folder called `04_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.

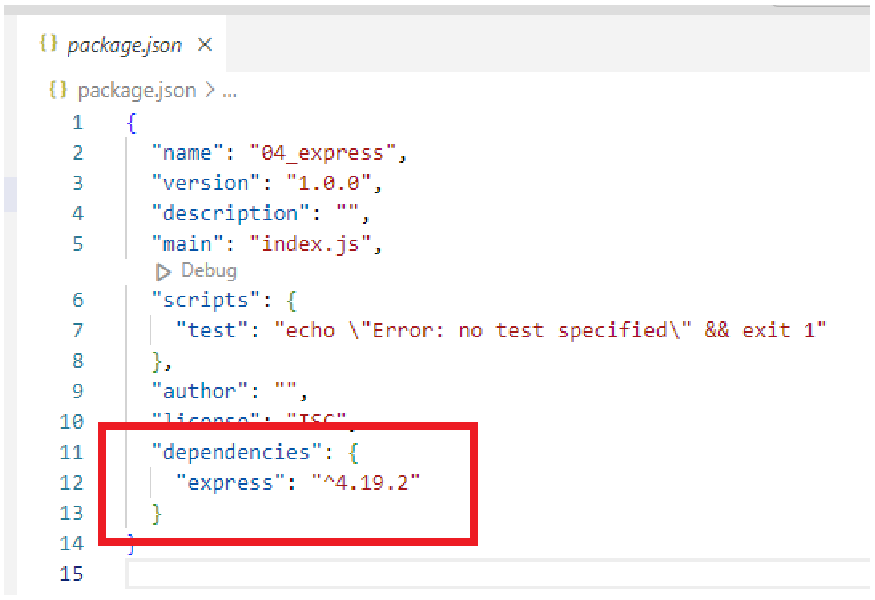
Task 2: Install the Express framework

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 it will be available for you to use.

When you run the **npm install** command, npm (the node package manager) will search for the package you are trying to install (in this case **Express**) and add it to your project folder. If you look at your explorer pane in Visual Studio Code, you can now see a completely new folder called `node_modules`. This folder stores all the packages that you have installed or will install in this project using npm. Furthermore, if you examine the `package.json` file, you will see that **Express** is now listed under the key "dependencies".



```
{
  "name": "04_express",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.19.2"
  }
}
```

Task 3: Create the code to serve a web page

You will now create the code to serve a web page using the Express framework.

6. Create a file called `index.js` in the project.
7. Add the following code to the `index.js` file and remember to save it:

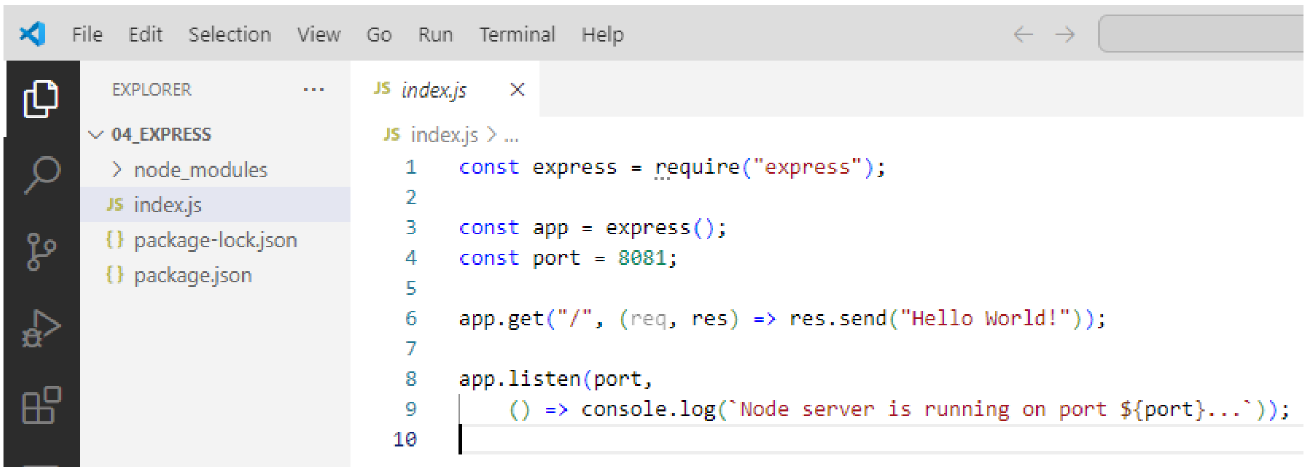
```
const express = require("express");

const app = express();
const port = 8081;

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

app.listen(port,
   () => console.log(`Node server is running on port ${port}...`));
```

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



```
const express = require("express");

const app = express();
const port = 8081;

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

app.listen(port,
    () => console.log(`Node server is running on port ${port}...`));
```

Task 3: Run the code

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

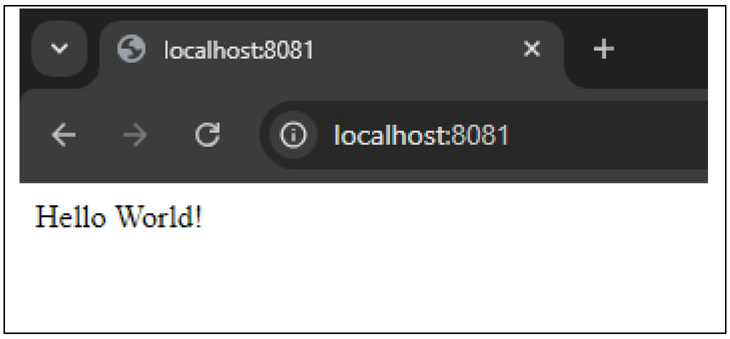
```
node index.js
```

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

Task 4: Access your server via HTTP

Now that your code is running and serving your application on port 8081, you can access your web page from a browser!

9. Start up a browser and navigate to http://localhost:8081. If you have correctly followed all the steps, your web browser should show the 'Hello World!' message.



Hello World!

Task 5: Understand the code

10. Compare the code in this lab, which uses the Express framework, with the code used in the previous labs, which used raw Node.js. They both effectively do the same thing. Here's a reminder of the raw Node.js code:

```
var http = require("http");
const port = 8081;

http.createServer(function(req, res) {
   res.writeHead(200, { "Content-Type": "text/plain" });
   res.end("Hello World!");
   }).listen(port, function() {
      console.log(`Node server is running on port ${port}...`);
});
```

Let's go through the Express code line-by-line.

The first line imports the Express module:

```
const express = require("express");
```

Next, create an app object, which is the object where you can access all of your Express methods.

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

Then define the port on which the application will listen for HTTP requests:

```
const port = 8081;
```

Now use the Express app object to create a 'get' route handler that handles the "/" route. You will look at this more in the next lab, but for now understand that this handles requests for the default web page at http://localhost:8081. The code handles any requests to this URL by returning the text 'Hello World!'.

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

Note the parameters to the function: `req` and `res`. `req` is the request object and holds information sent in the request from the client. res is the response object, used to build the response back to the client.

Finally, use the Express app object to start the web server listening for HTTP messages on the port:

```
app.listen(port,
   () => console.log(`Node server is running on port ${port}...`));
```

Note the function that runs when the server starts up successfully just prints out a message confirming that the app is running.

A note on JavaScript arrow functions

If you are fairly new to JavaScript, you may not be too familiar with the syntax for the functions used in the above code:

```
(req, res) => res.send("Hello World!")
```

These are called **arrow functions**. They provide a compacted version of the more usual JavaScript function syntax. The above function is equivalent to this:

```
function (req, res) {return res.send("Hello World!")}
```

Arrow functions are widely used, especially for these types of nameless (anonymous) functions used as parameters to other functions.

Task 6: 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.

The contents of the page 'Hello World!' are treated as HTML, which is the default content type that Express uses. Prove this by changing the 'Hello World!' string to use some HTML tags. For example:

```
"<h1>Hello World!</h1><p>This is my first Express program</p>"
```

End of section

Congratulations on completing this section.

Save your work – your files will remain when you close this lab activity so do not worry about losing your data.

Next, you will explore how to add more routes to your current Express web server and make your web application more interesting.