Lab 7: Node and Express

Before attempting this lab, you will need to have completed **Lab 6: JavaScript Async and Fetch**.

Learning objectives

By completing this lab, you should be able to:
- Write simple backend Node application
- Install and use nodemon package
- Run Node application
- Create routes for handling HTTP requests using Express API
- Serve static files such as HTML pages on Node server
- Write client-side JS code to handle form data

Background

Node (or more formally Node.js) (MDN) is an open-source, cross-platform runtime environment that allows developers to create all kinds of server-side tools and applications using JavaScript.

The runtime is intended for use outside of a browser context (i.e. running directly on a computer or server OS). As such, the environment omits browser-specific JavaScript APIs and adds support for more traditional OS APIs including HTTP (MDN) and file system libraries.

The advantages of Node include:
- it was designed to optimize throughput and scalability in web applications and is a good solution for many common web-development problems (e.g. real-time web applications).
- the code is written in "plain old JavaScript", which means that less time is spent dealing with "context shift" between languages when you're writing both client-side and server-side code.
- the node package manager (NPM) (source) provides access to hundreds of thousands of reusable packages. It also has best-in-class dependency resolution and can also be used to automate most of the build toolchain.
- Node.js is portable. It is available on Microsoft Windows, macOS, Linux, Solaris, FreeBSD, OpenBSD, WebOS, and NonStop OS. Furthermore, it is well-supported by many web hosting providers, that often provide specific infrastructure and documentation for hosting Node sites.
- it has a very active third-party ecosystem and developer community, with lots of people who are willing to help.

Routing is an important concept in Web Development. It is a mechanism by which requests (as specified by a URL (MDN) and HTTP method (MDN)) are routed to the server code that handles them. In this context if our server is based on Node, then Node will handle different HTTP (MDN) requests accordingly and appropriately. To help us on this task, we will use Express.

Express (MDN) is the most popular Node web framework, and is the underlying library for a number of other popular Node web frameworks. It provides mechanisms to:

- Write handlers for requests with different HTTP verbs at different URL paths (routes).
- Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
- Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- Add additional request processing "middleware" at any point within the request handling pipeline.

While Express itself is fairly minimalist, developers have created compatible middleware packages to address almost any web development problem. There are libraries to work with cookies, sessions, user logins, URL parameters, POST data, security headers, and many more.

For this module, we will be using Node and Express for our backend processing. In this lab, we will create a simple server-side web application using Express for NodeJS.

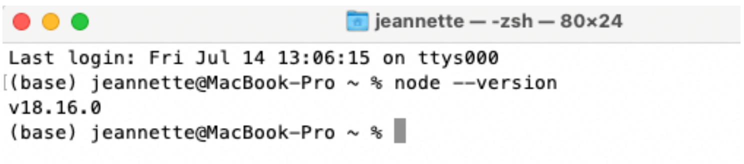
Exercises

1. Create a new folder in your student U: drive called `node-express`. Copy all the files in `js-async-fetch` folder to this new folder.

2. **If you are using a lab machine for this lab, then you can skip these tasks and jump to (4).** Install Node and Express on your own machine
   i. Node:
      a) For windows machine follow the instructions created by Microsoft https://docs.microsoft.com/en-us/windows/dev-environment/javascript/nodejs-on-windows or follow instructions on MDN
      b) For Mac machines - follow Node web page and download the latest package: https://nodejs.org/en/download/ or install using package manager : https://nodejs.org/en/download/package-manager/#macos or follow instructions on MDN
   ii. Express:
      a) For windows machines, follow the instructions on this link: https://expressjs.com/en/starter/installing.html

b) For Mac machines, follow the instructions on this link: https://www.npmjs.com/package/express

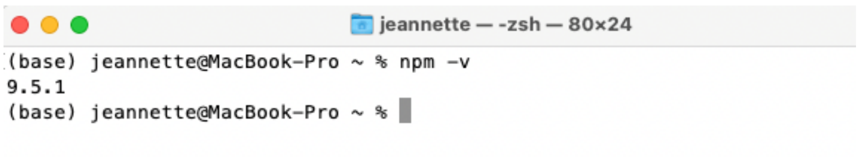
For information, I use Mac and my Node version is shown in Figure 1.



```
jeannette — -zsh — 80×24
Last login: Fri Jul 14 13:06:15 on ttys000
[(base) jeannette@MacBook-Pro ~ % node --version
v18.16.0
(base) jeannette@MacBook-Pro ~ %
```

Figure 1. The Node version on Jeannette's machine

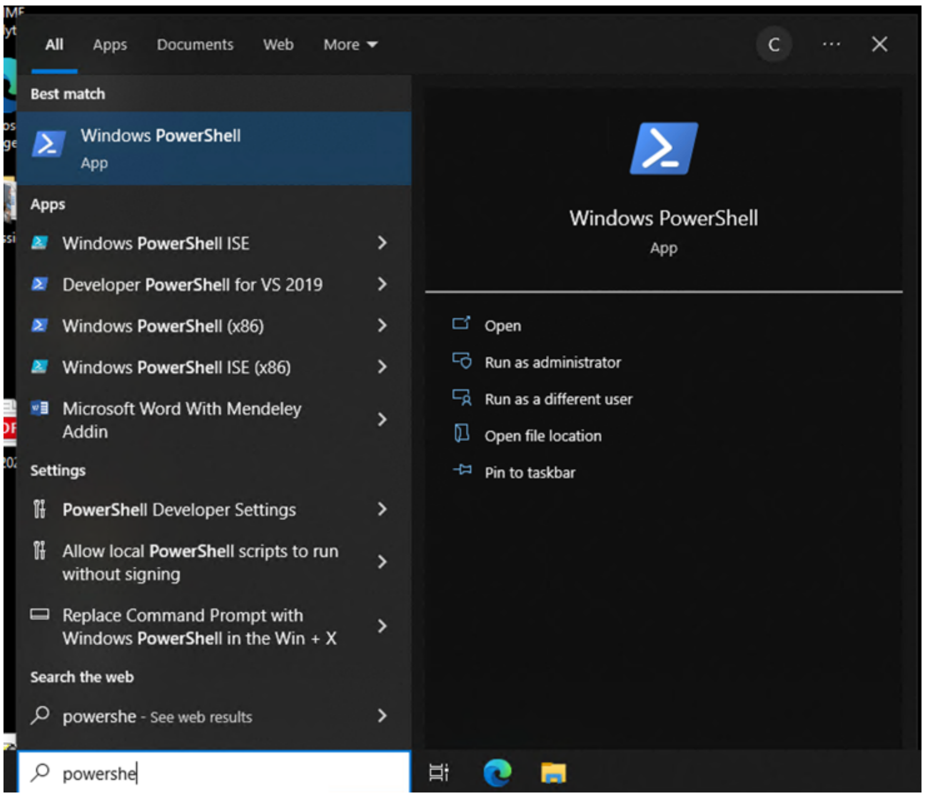
3. Check if NPM (Node package manager) (source) is installed on the machine. We will use `npm` to create our server. Figure 2 below shows the version on my local machine.



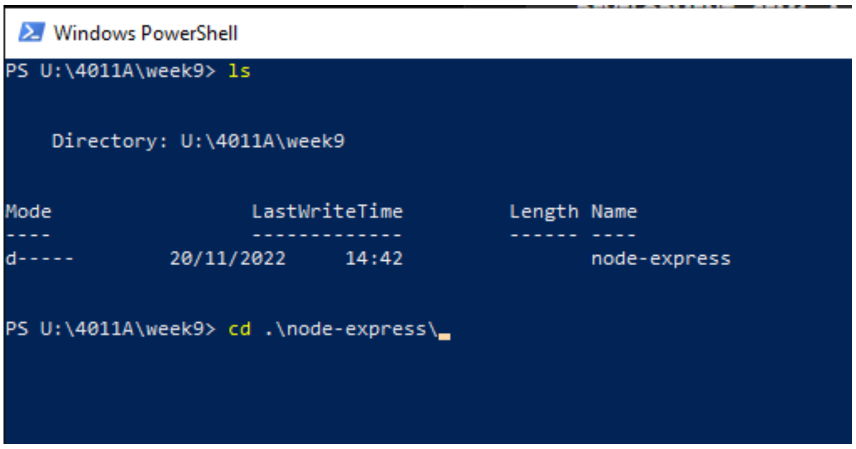
```
jeannette — -zsh — 80×24
(base) jeannette@MacBook-Pro ~ % npm -v
9.5.1
(base) jeannette@MacBook-Pro ~ %
```

Figure 2. Check the version of the library package

4. On a lab machine, bring up a Windows PowerShell.


All
Apps
Documents
Web
More
Best match
Windows PowerShell
App
Apps
Windows PowerShell ISE
Developer PowerShell for VS 2019
Windows PowerShell (x86)
Windows PowerShell ISE (x86)
Microsoft Word With Mendeley Addin
Settings
PowerShell Developer Settings
Allow local PowerShell scripts to run without signing
Replace Command Prompt with Windows PowerShell in the Win + X
Search the web
powershe - See web results
Windows PowerShell
App
Open
Run as administrator
Run as a different user
Open file location
Pin to taskbar
powershe

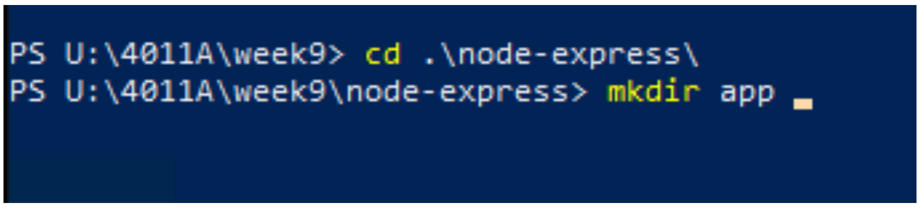
5. On Windows PowerShell, navigate to the project root folder in your student U: drive. Figure 3 below shows my project root folder is “node-express”. Your project root folder should be `node-express`, the one you created earlier on your U: drive.



```
Windows PowerShell
PS U:\4011A\week9> ls


    Directory: U:\4011A\week9


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        20/11/2022     14:42                node-express


PS U:\4011A\week9> cd .\node-express\
```

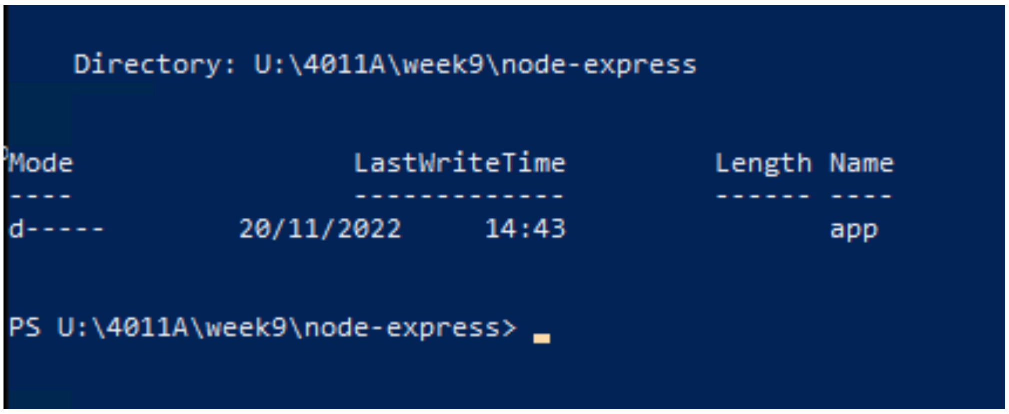
Figure 3. Navigate to project root directory on a Terminal window

6. We will create a new folder in the project root directory. On PowerShell - type `mkdir app` (Figure 4) and hit enter, the output is shown in Figure 4a



```
PS U:\4011A\week9> cd .\node-express\
PS U:\4011A\week9\node-express> mkdir app
```

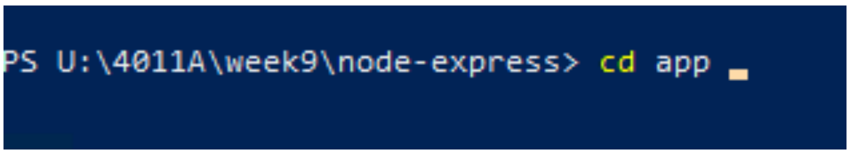
Figure 4. command on a Terminal window



```
    Directory: U:\4011A\week9\node-express


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        20/11/2022     14:43                app


PS U:\4011A\week9\node-express>
```

Figure 4a. The output showing a new directory called ‘app’.

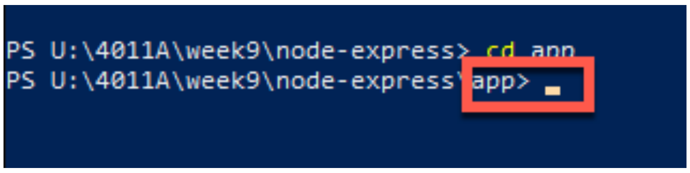
7. Now go to this app directory by typing `cd app` (Figure 5) and hit enter



```
PS U:\4011A\week9\node-express> cd app
```

Figure 5. command on a Terminal window

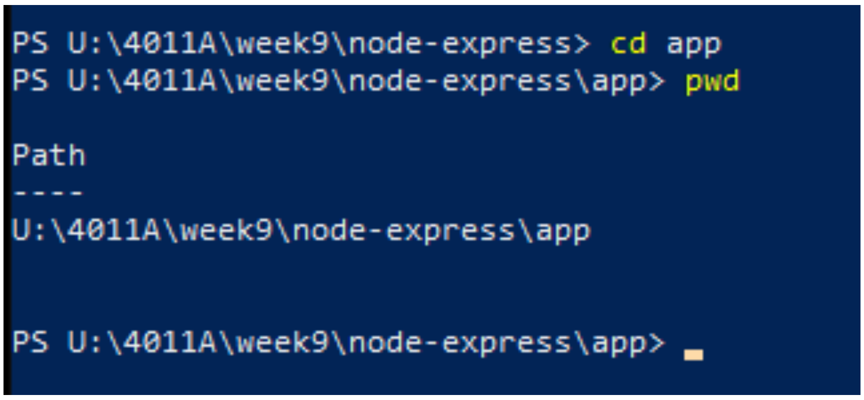
8. Double check that we are in the `app` folder which is inside the project root directory (Figure 6)



```
PS U:\4011A\week9\node-express> cd app
PS U:\4011A\week9\node-express\app> _
```

Figure 6. command on a Terminal window

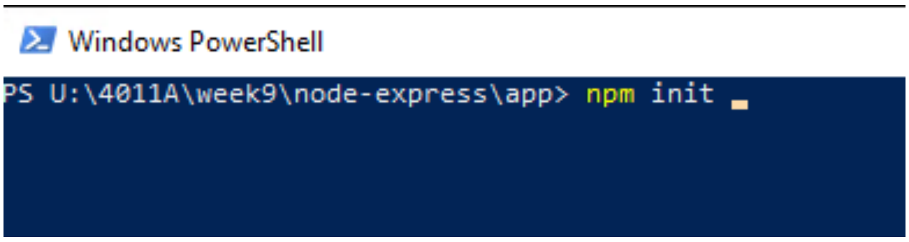
Type command `pwd`, and you will see the path and location of your current directory (Figure 6a)



```
PS U:\4011A\week9\node-express> cd app
PS U:\4011A\week9\node-express\app> pwd

Path
----
U:\4011A\week9\node-express\app


PS U:\4011A\week9\node-express\app> _
```

Figure 6a. The output of command pwd

9. Now type `npm init` (Figure 7) and hit enter.

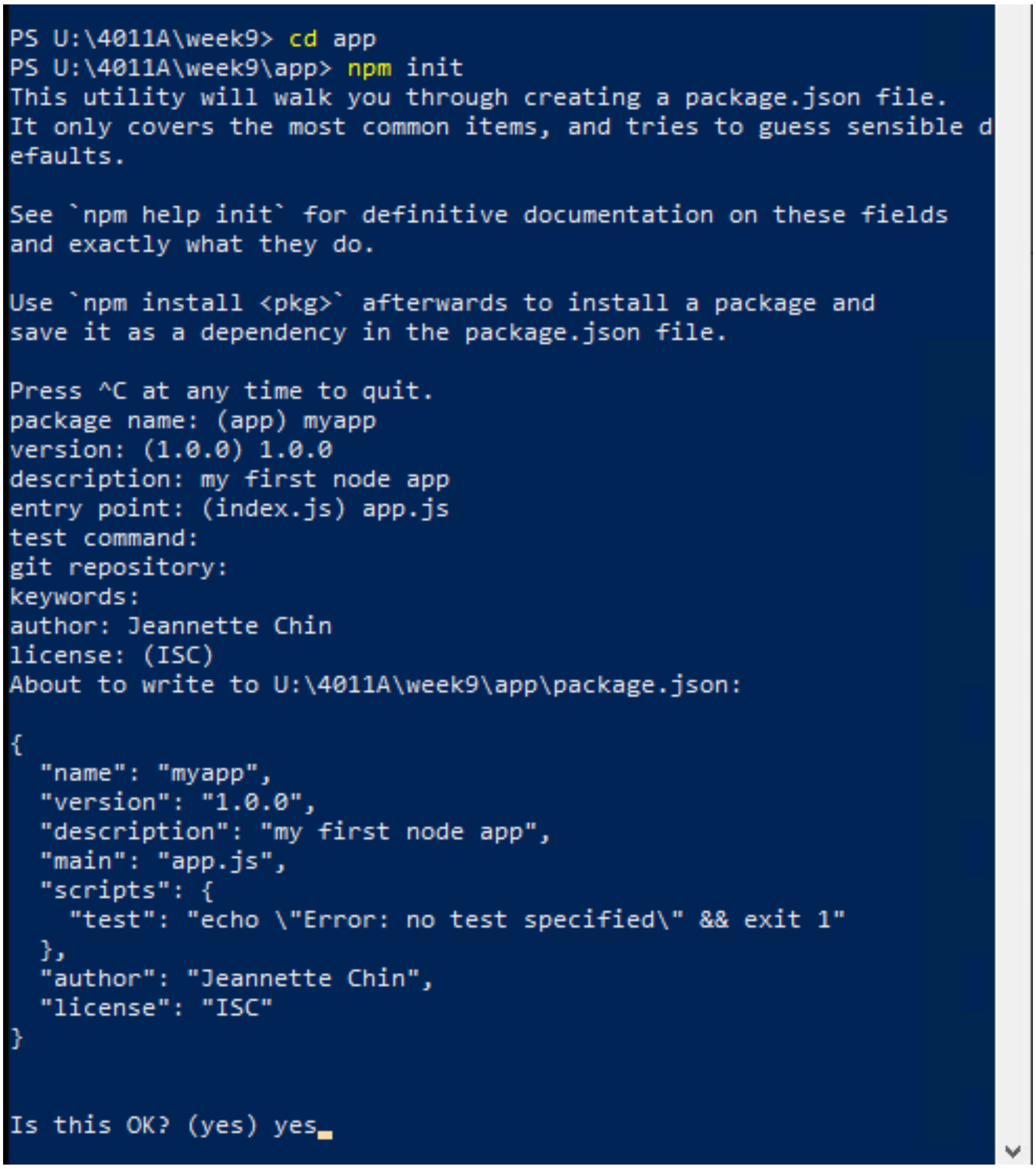


```
PS U:\4011A\week9\node-express\app> npm init _
```

Figure 7. command on a Terminal window

You will prompt to answer some questions, type in the answers, or just hit enter to skip. Figure 8 below shows my answers to some of these questions. Suggest you use my answers to these questions so that you can follow the instructions later, specifically on

- Package name: myapp
- Version: 1.0.0
- Entry point: app.js



```
PS U:\4011A\week9> cd app
PS U:\4011A\week9\app> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible d
efaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (app) myapp
version: (1.0.0) 1.0.0
description: my first node app
entry point: (index.js) app.js
test command:
git repository:
keywords:
author: Jeannette Chin
license: (ISC)
About to write to U:\4011A\week9\app\package.json:

{
  "name": "myapp",
  "version": "1.0.0",
  "description": "my first node app",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jeannette Chin",
  "license": "ISC"
}


Is this OK? (yes) yes
```

Figure 8. Answers to the utility questions

10. Now on the file window, go to the project root folder. You will see the file structure as shown in Figure 9. The `npm` utility program has created a new file called `package.json` for us in the `app` folder.

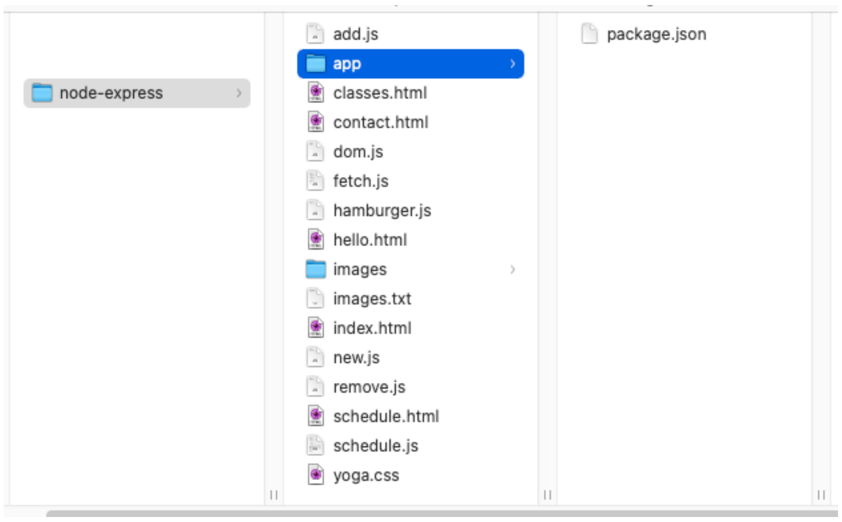
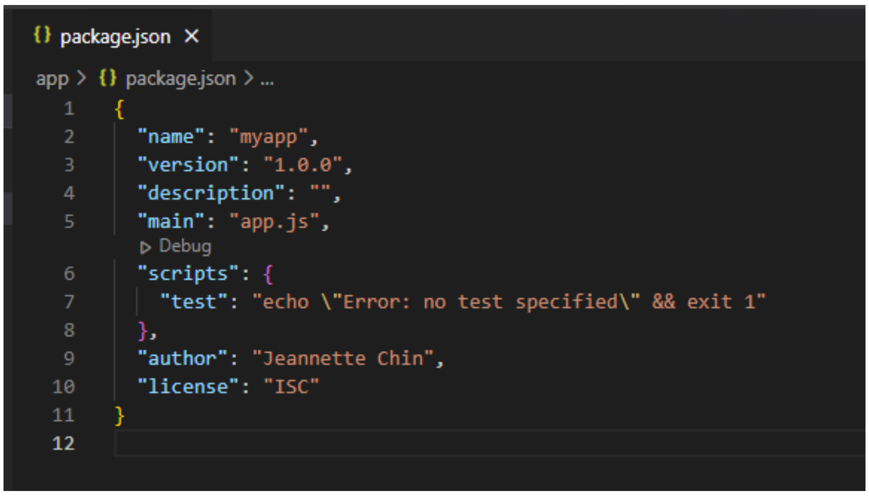


Figure 9. The file structure of Lab 7

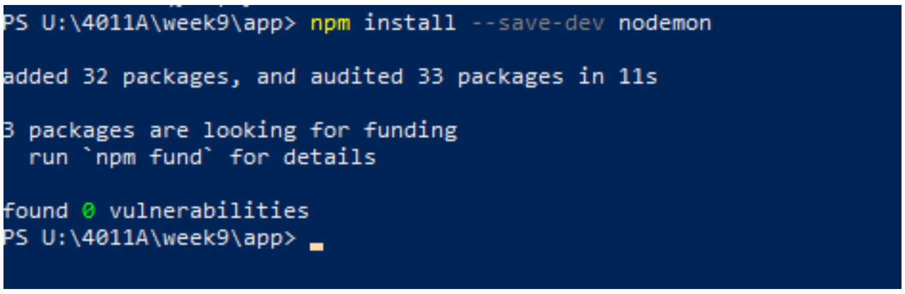
11. Open this file `package.json` on your Visual Studio Code editor. My output is shown in Figure 10 below for you to compare.



```
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Jeannette Chin",
  "license": "ISC"
}
```

Figure 10. Jeannette's `package.json` file.

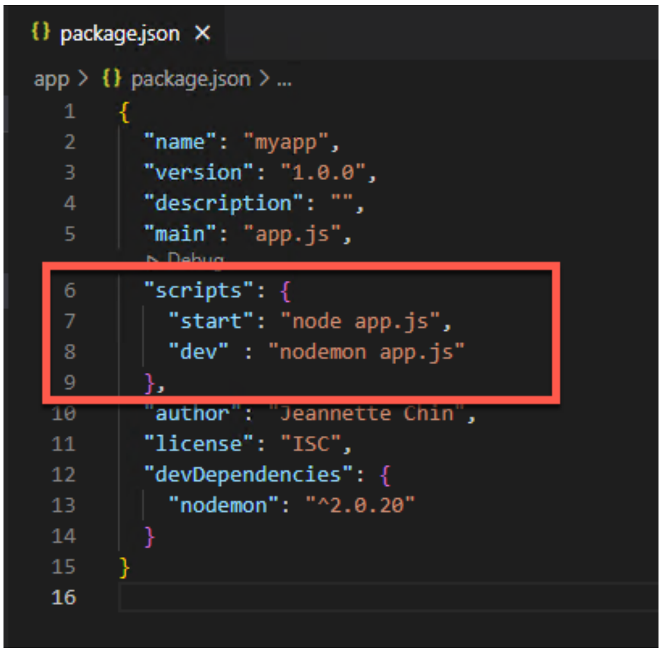
12. Next, we will install some dependencies for this lab exercise:
    i. There is this handy tool called `nodemon` (source) which will help us restart the Node every time we make any changes to the code. This is a very useful tool for project development, otherwise we would have to restart the server manually, which is a tedious job. The command to install this dependency is shown in Figure 12. Type this command on your PowerShell window:



```
PS U:\4011A\week9\app> npm install --save-dev nodemon

added 32 packages, and audited 33 packages in 11s

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

found 0 vulnerabilities
PS U:\4011A\week9\app> _
```

Figure 12. command to install nodemon

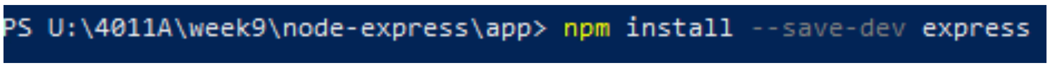
ii. In your text editor, modifiy `package.json` file as shown in Figure 13 – line 7 & 8. After this we can then simply use this command "`npm run dev`" to run our development server later.



```
1   {
2     "name": "myapp",
3     "version": "1.0.0",
4     "description": "",
5     "main": "app.js",
      ▷ Debug
6     "scripts": {
7       "start": "node app.js",
8       "dev" : "nodemon app.js"
9     },
10    "author": "Jeannette Chin",
11    "license": "ISC",
12    "devDependencies": {
13      "nodemon": "^2.0.20"
14    }
15  }
16
```

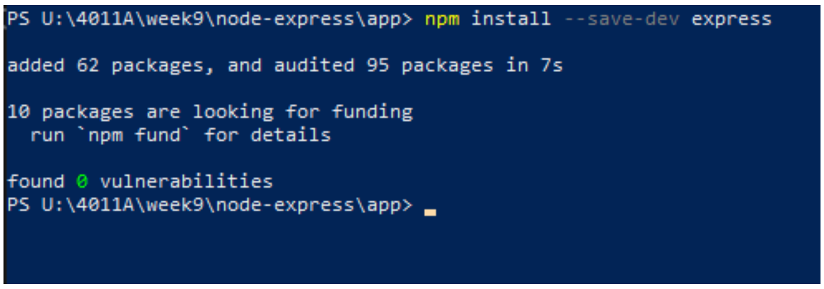
Figure 13. the new code for line 7 and 8 on package.json file

iii. Next, we will install Express. On the PowerShell window, type the command shown in Figure 14, and hit enter



```
PS U:\4011A\week9\node-express\app> npm install --save-dev express
```

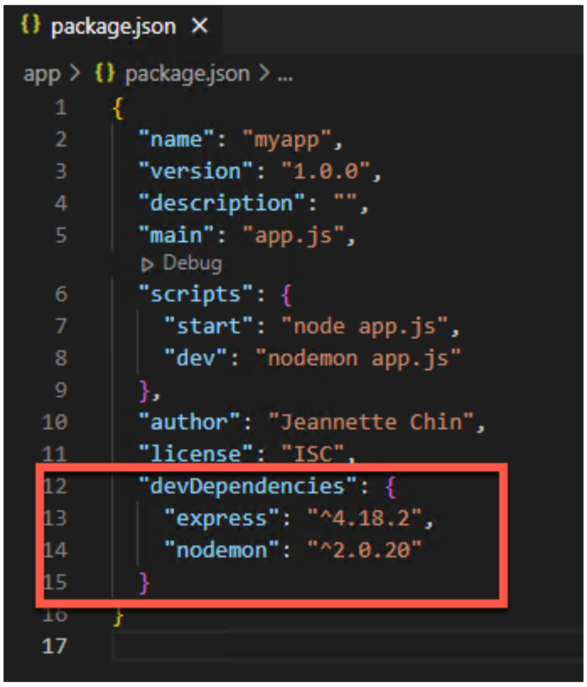
Figure 14. Command to install Express on a Terminal window



```
PS U:\4011A\week9\node-express\app> npm install --save-dev express

added 62 packages, and audited 95 packages in 7s

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

found 0 vulnerabilities
PS U:\4011A\week9\node-express\app> _
```

Figure 14a. The output of the command

iv. Once this is done, examine the output of the dependencies on `package.json`. Figure 15 below shows the output.



```
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  },
  "author": "Jeannette Chin",
  "license": "ISC",
  "devDependencies": {
    "express": "^4.18.2",
    "nodemon": "^2.0.20"
  }
}
```

Figure 15. The dependencies of this lab exercises

13. We will now create our server file `app.js`. In the text editor create this new file inside the app folder, as shown in Figure 16

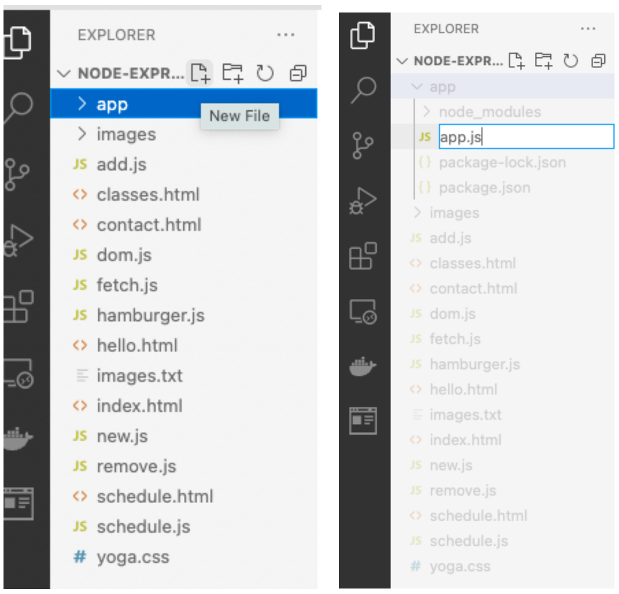


Figure 16. Create app.js file inside the app folder

14. Open this `app.js` file in the text editor – Figure 17

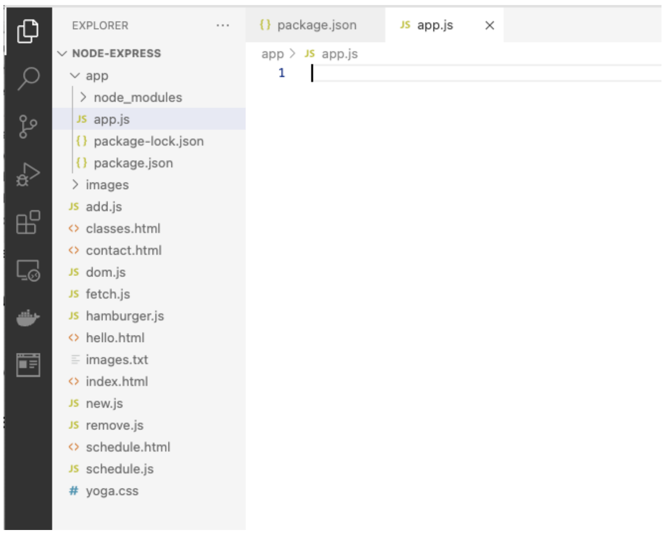
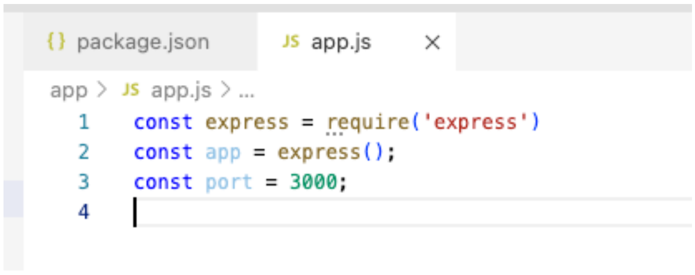


Figure 17. The app.js file

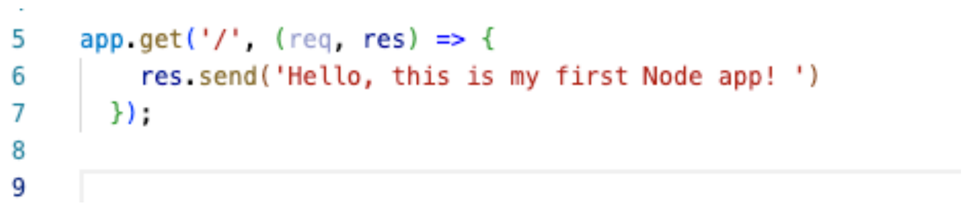
15. First we will import the Express library by using the `require()` function so that it is included in our Node. We will call our Express server "app" and get this server to run on port 3000 – Figure 18



```
const express = require('express')
const app = express();
const port = 3000;
```

Figure 18. Include Express to Node server

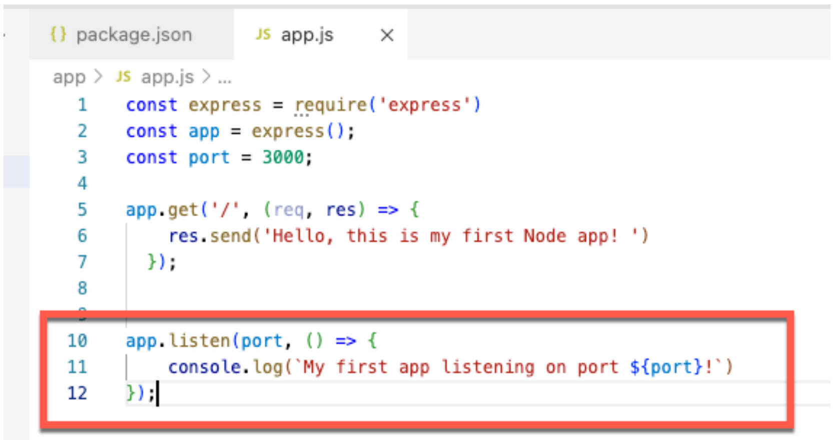
16. We will code our first route, the default route, using HTTP GET ([MDN]) method. Type the code shown in Figure 19 in your `app.js` file. The method takes the path and a `callback` function as parameters. The path indicates the URL string of this route; the `callback` function takes a request (`req`) and respond (`res`) object as parameter. In this method, the server simply responds with a string.



```
app.get('/', (req, res) => {
    res.send('Hello, this is my first Node app! ')
});
```

Figure 19. The default route on app.js

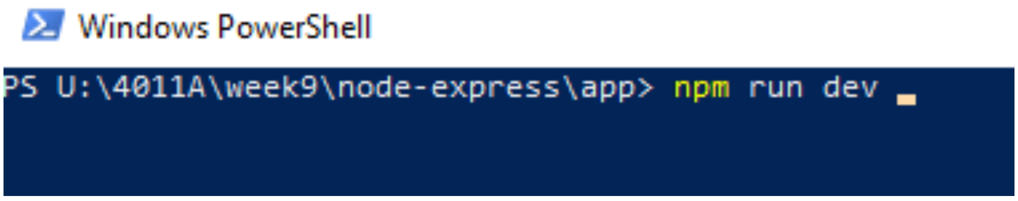
17. To see our route in action we will need to get the app running first. Now type the code shown in Figure 20 in your `app.js`. Notice I use backtick (MDN) in `console.log` method.



```
const express = require('express')
const app = express();
const port = 3000;

app.get('/', (req, res) => {
    res.send('Hello, this is my first Node app! ')
  });

app.listen(port, () => {
    console.log(`My first app listening on port ${port}!`)
});
```

Figure 20. The code on app.js

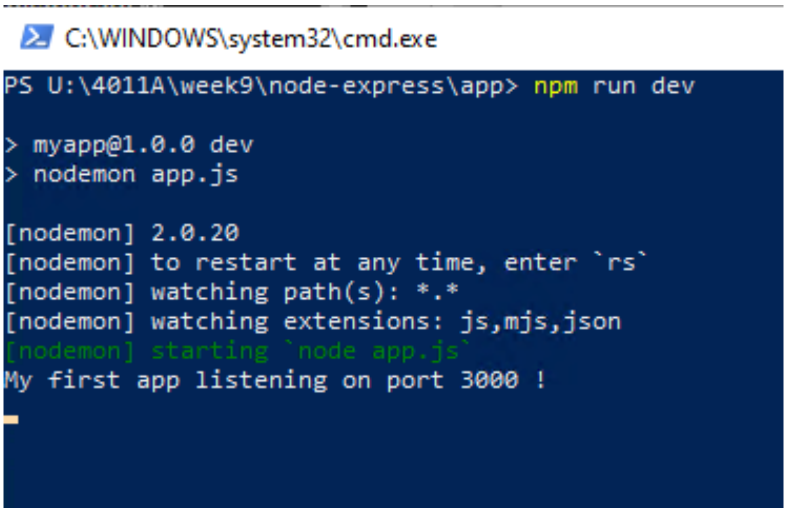
18. To test our first route, type the command shows in Figure 21 to run the development server



```
PS U:\4011A\week9\node-express\app> npm run dev
```

Figure 21. Run the development server

Once you hit enter, you will see the output similar to Figure 22. The development server is running on port 3000



```
PS U:\4011A\week9\node-express\app> npm run dev

> myapp@1.0.0 dev
> nodemon app.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
My first app listening on port 3000 !
```

Figure 22. The development server is running on port 3000

19. To test the route, type localhost:3000 on Chrome (Figure 23)

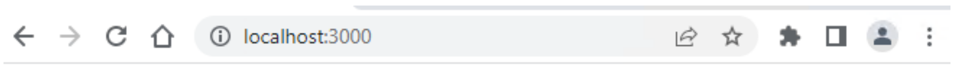
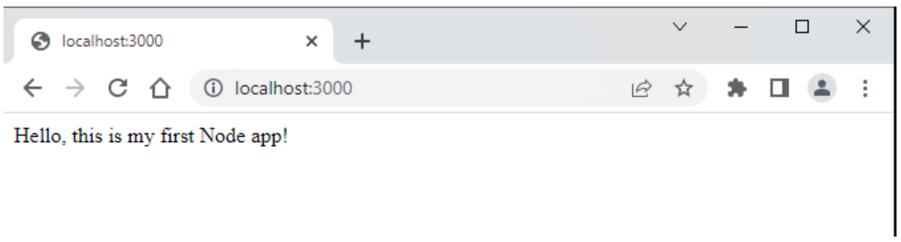


Figure 23. Test the server on Chrome

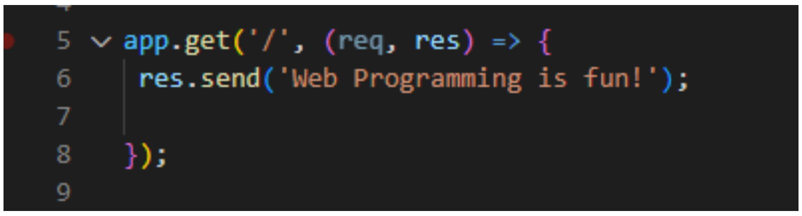
20. The output is shown in Figure 24 below



```
Hello, this is my first Node app!
```

Figure 24. The output of the server

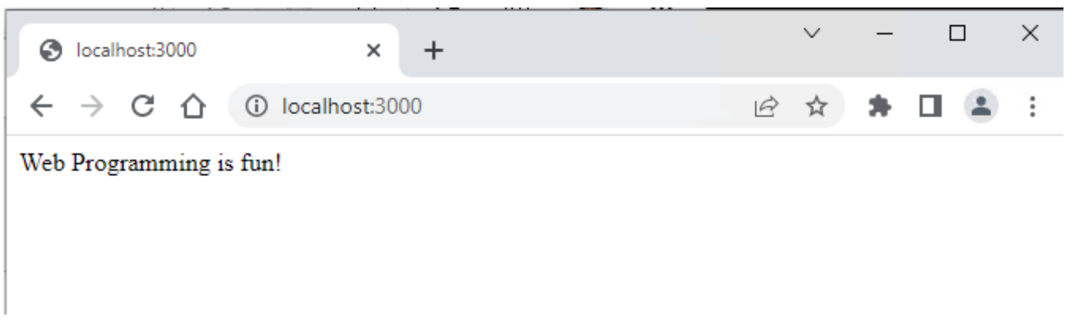
21. Now, on your `app.js`, change the respond value to read “Web Programming is fun!” (Figure 24a)



```
app.get('/', (req, res) => {
  res.send('Web Programming is fun!');

});

```

Figure 24a. Changed the response value

22. Save the file and refresh the page. Your output should look similar to Figure 25

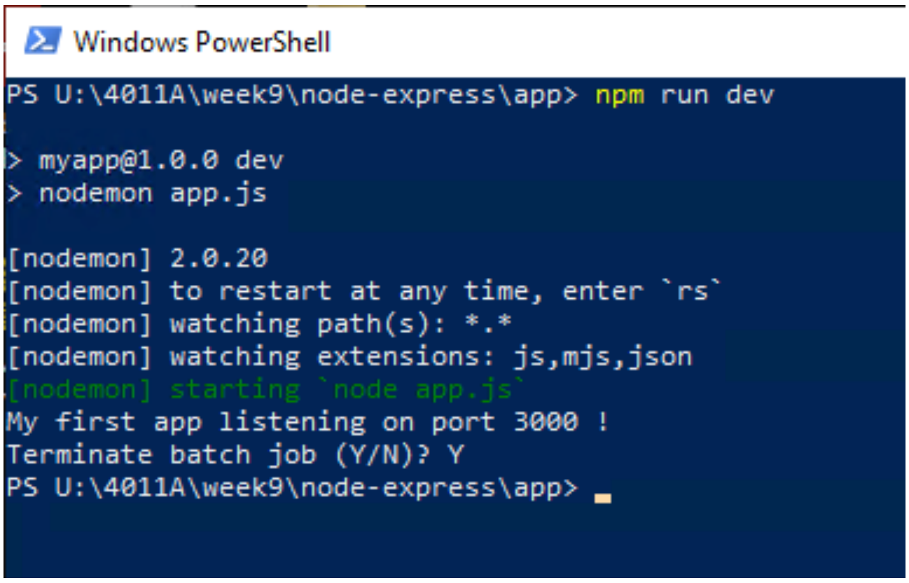


```
Web Programming is fun!
```

Figure 25. The output of the server.

Notice that we don’t need to restart the server but can see the changes. This is because `nodemon` is doing this in the background for us.

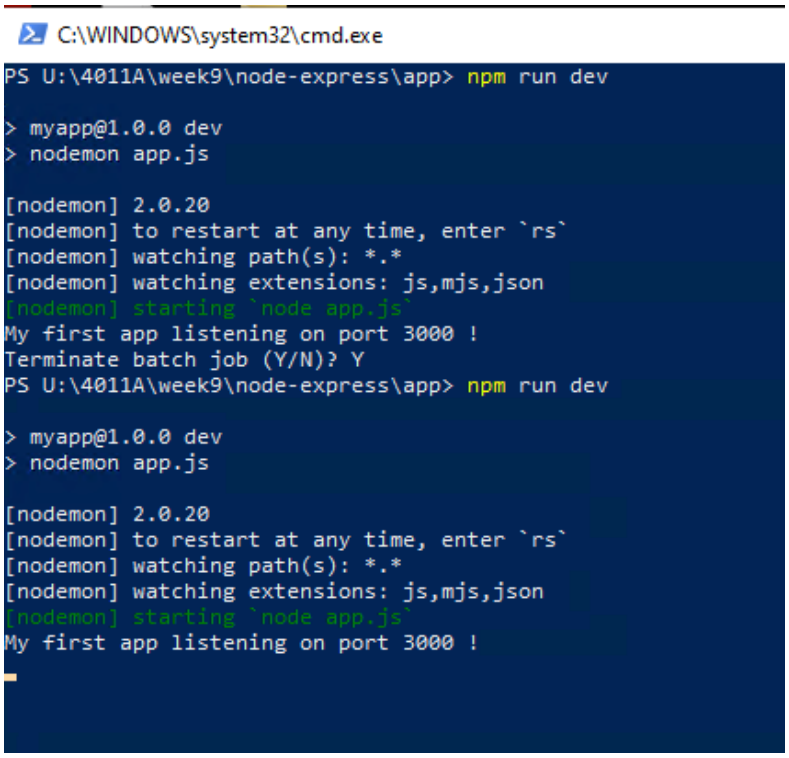
23. Stop the Node server by switching back to the PowerShell that runs the server, and then press the `Ctrl` and `C` keys on the keyboard. When prompt "Terminate batch job?" Enter Y (Figure 26).



```
PS U:\4011A\week9\node-express\app> npm run dev

> myapp@1.0.0 dev
> nodemon app.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
My first app listening on port 3000 !
Terminate batch job (Y/N)? Y
PS U:\4011A\week9\node-express\app>
```

Figure 26. Stop the server using Ctrl C

24. To start the server, press the `Up` arrow key once to show the previous command executed (which should be `npm run dev`), and press `Enter` to start the Node server again (Figure 27).



```
PS U:\4011A\week9\node-express\app> npm run dev

> myapp@1.0.0 dev
> nodemon app.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
My first app listening on port 3000 !
Terminate batch job (Y/N)? Y
PS U:\4011A\week9\node-express\app> npm run dev

> myapp@1.0.0 dev
> nodemon app.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node app.js`
My first app listening on port 3000 !
```

Figure 27. Restart the development server

25. The next task is to get our server to serve web pages instead of responding with a string. We will need a folder with public access to store our HTML pages. On your text editor, create a new folder called `public`. This folder should be placed inside the `app` directory Figure 28

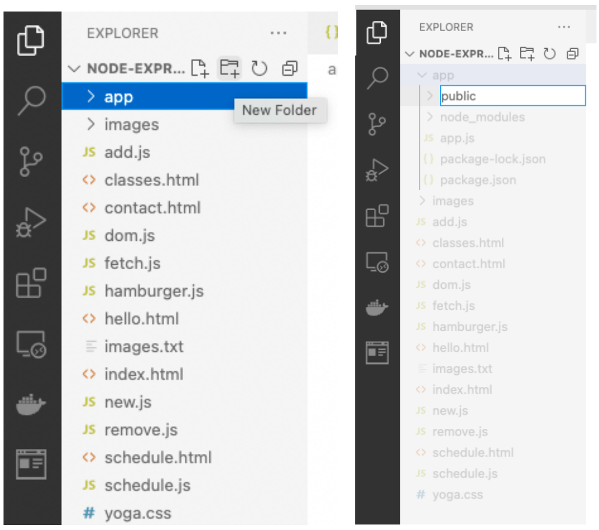
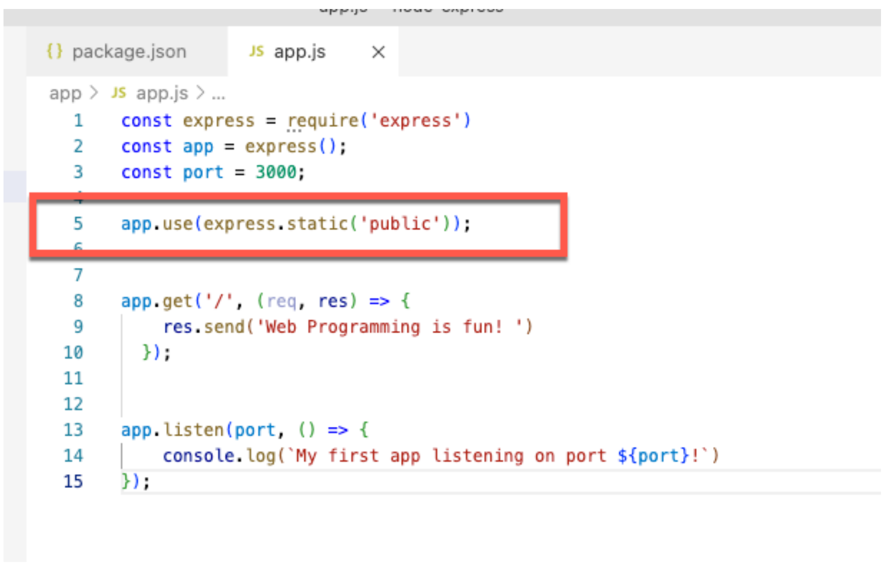


Figure 28. Create a new folder call public

26. Now configure the Node with the code shown in Figure 29, line 5, in your `app.js`. This will tell our Express server the location where we store of our static files.



```
1  const express = require('express')
2  const app = express();
3  const port = 3000;
4
5  app.use(express.static('public'));
6
7
8  app.get('/', (req, res) => {
9      res.send('Web Programming is fun! ')
10   });
11
12
13 app.listen(port, () => {
14     console.log(`My first app listening on port ${port}!`)
15 });
```

Figure 29. Configure static file directory on Node

27. Next, move all the files in the project root directory to this `public` folder – as shown in Figure 30 & 31.

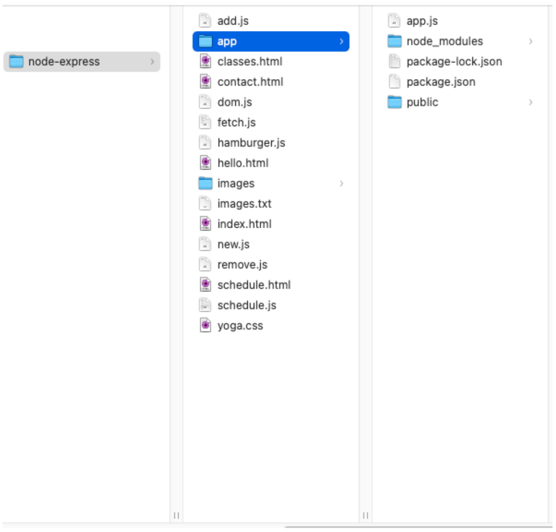


Figure 30. Move files – BEFORE

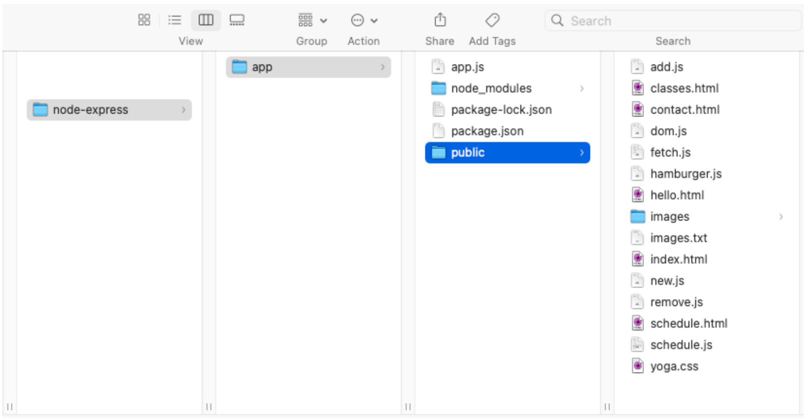
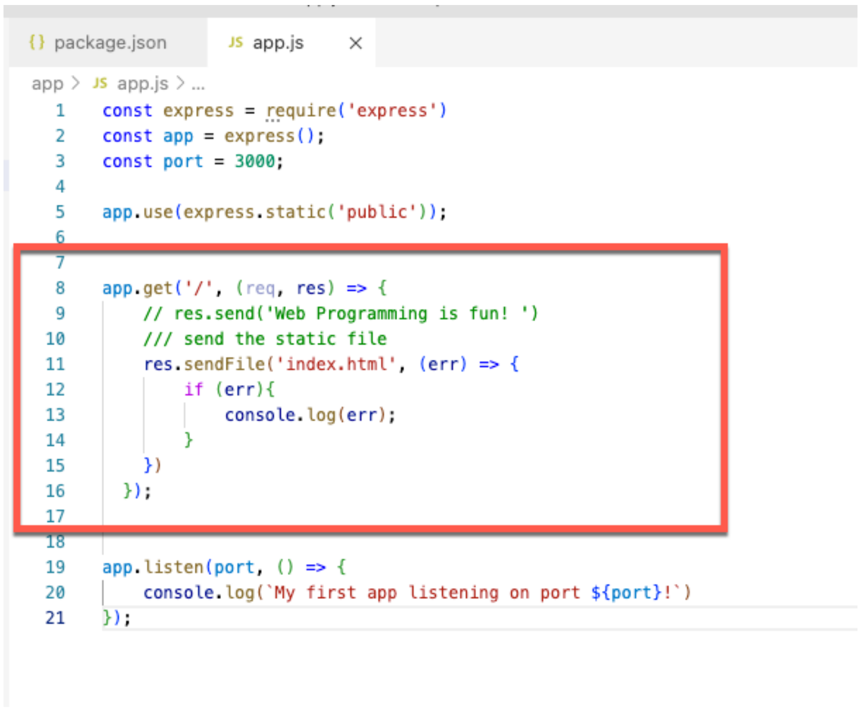


Figure 31. Move files – AFTER

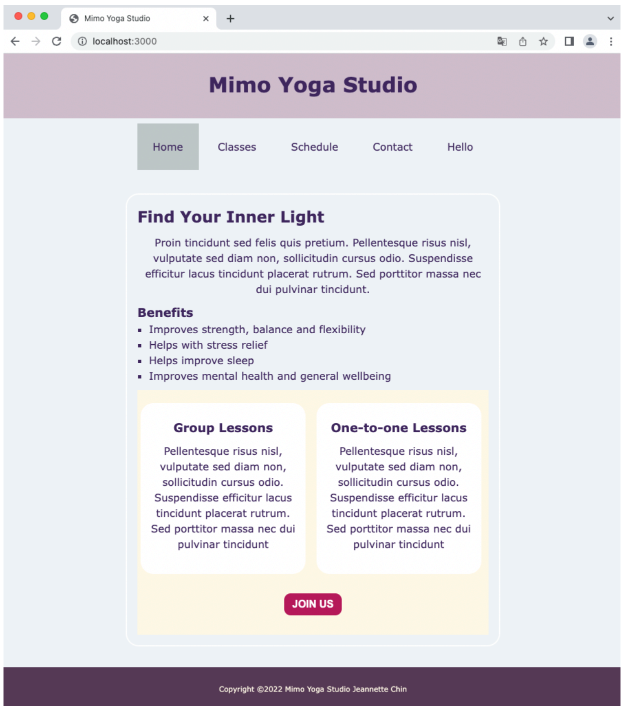
28. When this is done, go back to the `app.js`, modify the response code. We can use the `sendFile` method to send the static file, Figure 32, line 11 to 15



```
const express = require('express')
const app = express();
const port = 3000;

app.use(express.static('public'));


app.get('/', (req, res) => {
    // res.send('Web Programming is fun! ')
    /// send the static file
    res.sendFile('index.html', (err) => {
        if (err){
            console.log(err);
        }
    })
  });


app.listen(port, () => {
    console.log(`My first app listening on port ${port}!`)
});
```

Figure 32. Code on app.js

29. Save the file and refresh the page. Your output should be similar to Figure 33 below



Mimo Yoga Studio

Find Your Inner Light

Proin tincidunt sed felis quis pretium. Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus odio. Suspendisse efficitur lacus tincidunt placerat rutrum. Sed porttitor massa nec dui pulvinar tincidunt.

Benefits
- Improves strength, balance and flexibility
- Helps with stress relief
- Helps improve sleep
- Improves mental health and general wellbeing

Group Lessons

Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus odio. Suspendisse efficitur lacus tincidunt placerat rutrum. Sed porttitor massa nec dui pulvinar tincidunt

One-to-one Lessons

Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus odio. Suspendisse efficitur lacus tincidunt placerat rutrum. Sed porttitor massa nec dui pulvinar tincidunt

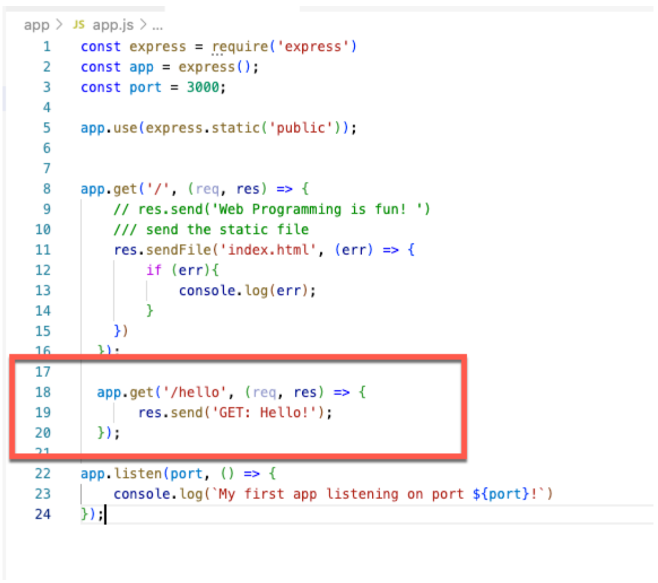
Copyright ©2022 Mimo Yoga Studio Jeannette Chin

Figure 33. The output of landing page

We have now successfully implemented our Node server and using Express to serve static web pages.

30. Now test the navigation links, do they work?

31. Next, we will add route parameters with GET request. Create a GET route with URL ‘hello’ as shown in Figure 34.



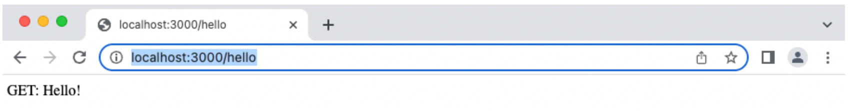
```
app > JS app.js > ...
const express = require('express')
const app = express();
const port = 3000;

app.use(express.static('public'));


app.get('/', (req, res) => {
    // res.send('Web Programming is fun! ')
    /// send the static file
    res.sendFile('index.html', (err) => {
        if (err){
            console.log(err);
        }
    })
});

app.get('/hello', (req, res) => {
    res.send('GET: Hello!');
});

app.listen(port, () => {
    console.log(`My first app listening on port ${port}!`)
});
```

Figure 34. Get route

Save the file.

32. Type this URL on Chrome – http://localhost:3000/hello

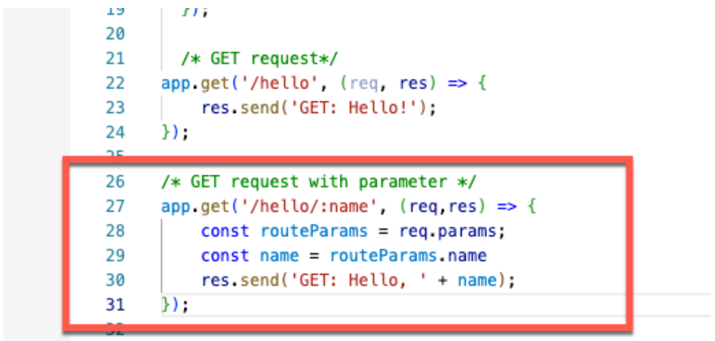
33. Your output should look similar to Figure 35.



```
GET: Hello!
```

Figure 35. The output of /hello request

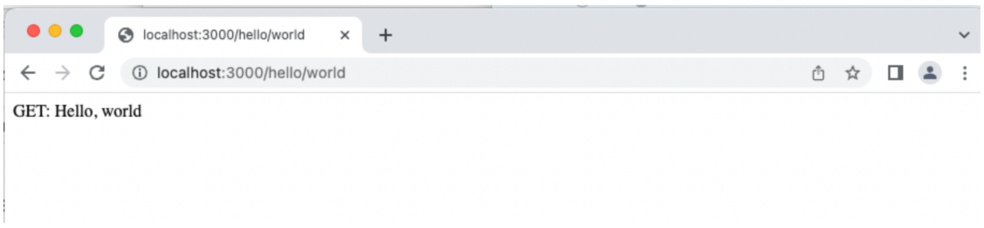
34. Now code another method, this time takes a name as route parameter, as shown in Figure 36, line 27 – 31, save the file.



```
    });

    /* GET request*/
app.get('/hello', (req, res) => {
    res.send('GET: Hello!');
});

/* GET request with parameter */
app.get('/hello/:name', (req,res) => {
    const routeParams = req.params;
    const name = routeParams.name
    res.send('GET: Hello, ' + name);
});
```

Figure 36. Code in app.js

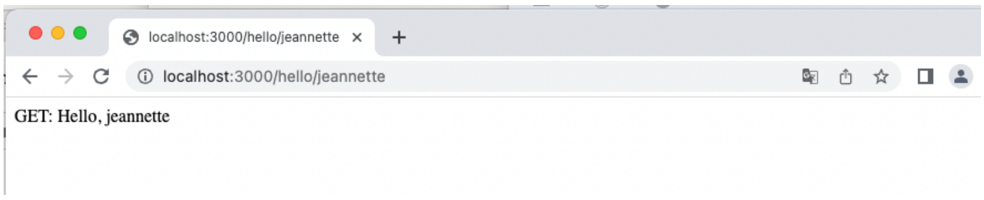
35. The GET request parameters are sent via URL string. We can test this function using Chrome – e.g. http://localhost:3000/hello/world , as shown in Figure 37. In this example the route parameter value is the word "world" using URL string (after /hello/). In the code (Figure 36) the Node function retrieves this value from the request parameters, and simply relay this value back to the client/browser.



```
localhost:3000/hello/world

GET: Hello, world
```

Figure 37. The output of request parameters

36. Now test another value, e.g. using your name. Figure 38 below shows the output of my name:

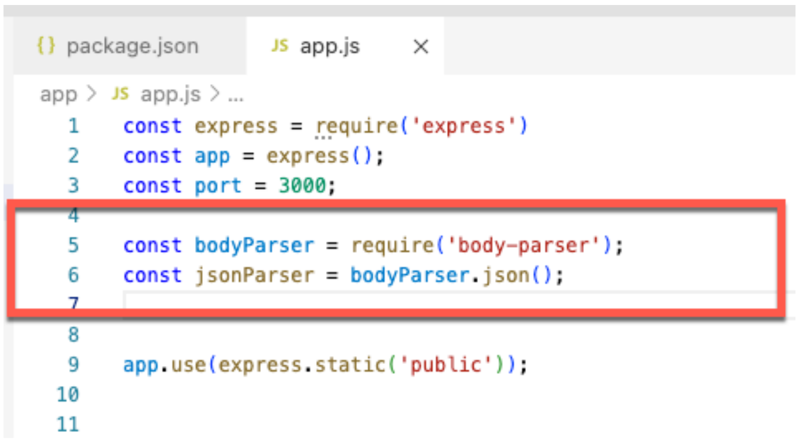


```
localhost:3000/hello/jeannette

GET: Hello, jeannette
```

Figure 38. The output of request parameters.

Using route parameters is relatively straight forward however it is not a secure way of sending data to the server. As we have just seen, the information is visible on the URL string. This method is not suitable for sending Form data. The standard approach of sending data to the server is using POST request.

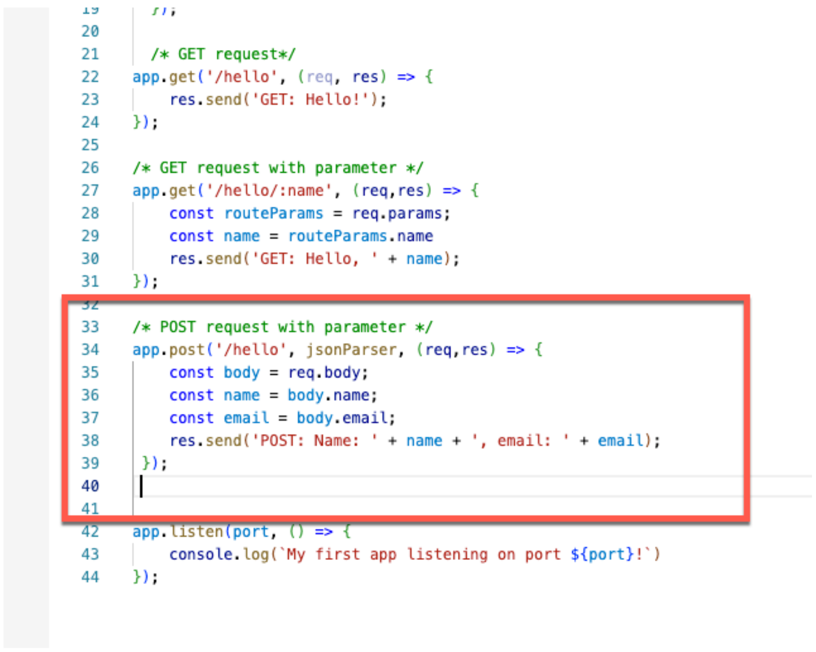
37. Next, we will create a POST request to handle JSON data. Before we code the function, we will use Express body parser library (source) to help us handle the JSON data. NOTE: lab machines have already installed this library. However, you may need to install this on your own machine, follow the instructions on https://expressjs.com/en/resources/middleware/body-parser.html



```javascript
const express = require('express')
const app = express();
const port = 3000;

const bodyParser = require('body-parser');
const jsonParser = bodyParser.json();


app.use(express.static('public'));


```

Figure 39. The code on app.js

38. Next, create a POST request with parameter. The request is sent as JSON data. Type the code shown in Figure 40, lines 34 – 39.



```javascript
19    });
20
21    /* GET request*/
22    app.get('/hello', (req, res) => {
23        res.send('GET: Hello!');
24    });
25
26    /* GET request with parameter */
27    app.get('/hello/:name', (req,res) => {
28        const routeParams = req.params;
29        const name = routeParams.name
30        res.send('GET: Hello, ' + name);
31    });
32
33    /* POST request with parameter */
34    app.post('/hello', jsonParser, (req,res) => {
35        const body = req.body;
36        const name = body.name;
37        const email = body.email;
38        res.send('POST: Name: ' + name + ', email: ' + email);
39     });
40
41
42    app.listen(port, () => {
43        console.log(`My first app listening on port ${port}!`)
44    });
```

Figure 40. The code in app.js

Code explained: The function uses body-parser package to parse the values from the request body. The function relay the values back to the client/browser.

We can’t use a browser to test the POST request because browsers are sending GET requests by default. For this lab exercise, we will write a **client code** to test this request.

39. On your text editor, create a new file called "`hello.js`", and this file should be placed inside the `public` folder – Figure 41

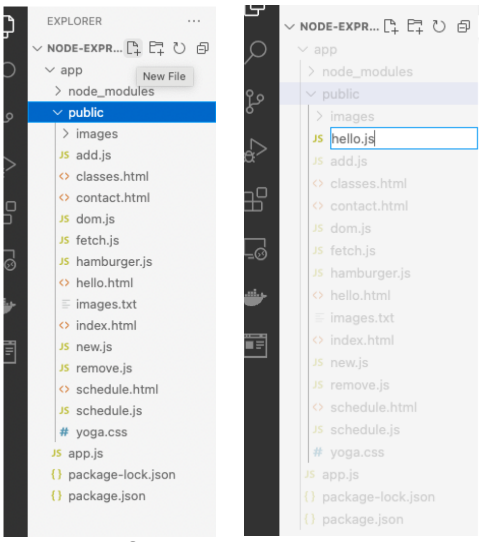
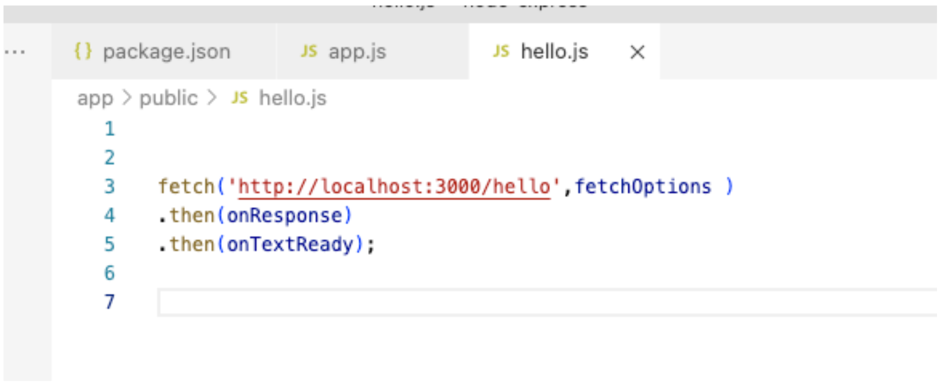


Figure 41. Create hello.js file in public folder

40. On this hello.js file, create a function using fetch API to send a POST request to this URL "http://localhost:3000/hello", as shown in Figure 42.



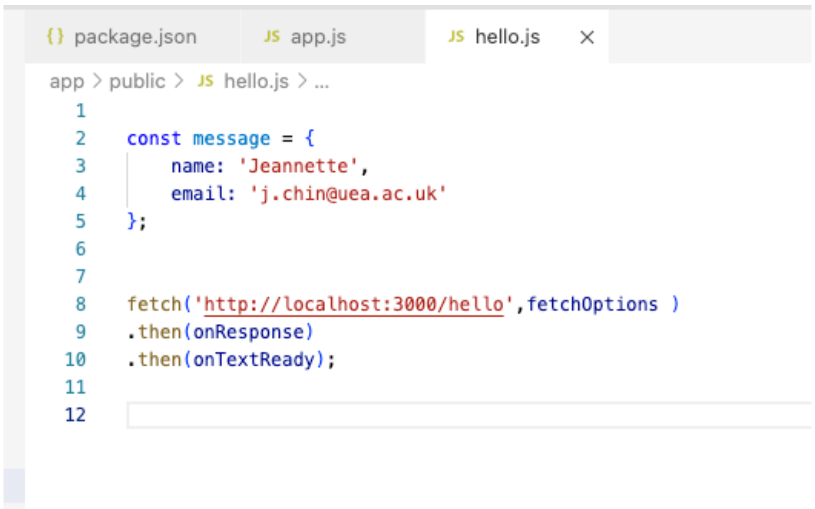
```

fetch('http://localhost:3000/hello',fetchOptions )
.then(onResponse)
.then(onTextReady);

```

Figure 42. The code in hello.js (client code)

Code explained: the fetch API call send a request to URL "http://localhost:3000/hello" with a request data encapsulated in an object called "`fetchOptions`" – which we will code next. The fetch API call will return a Promise which is handled using the `.then()` method, the return from the first Promise is `text()`, which is also a Promise, therefore we can chain this in the second `.then()` method.

Note: the `onResponse` and `onTextReady` handler functions are yet to be implemented. We will code this "`fetchOptions`" first, the JSON data we want to send to our Node server.

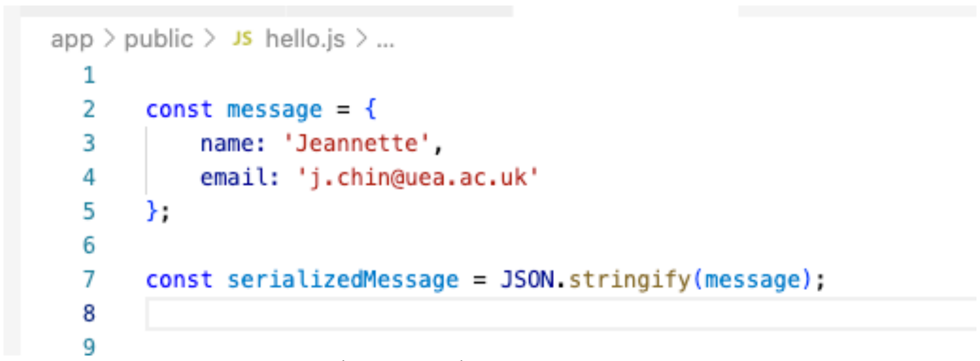
41. Back to `hello.js` file, create a JS object called `message` with the properties and values as shown in Figure 43, line 2 to 5. This is the data we want to send to our Node server. Notice I use my details, change the values appropriately to your name and email address.



```

const message = {
    name: 'Jeannette',
    email: 'j.chin@uea.ac.uk'
};


fetch('http://localhost:3000/hello',fetchOptions )
.then(onResponse)
.then(onTextReady);


```

Figure 43. The code in hello.js (client code)

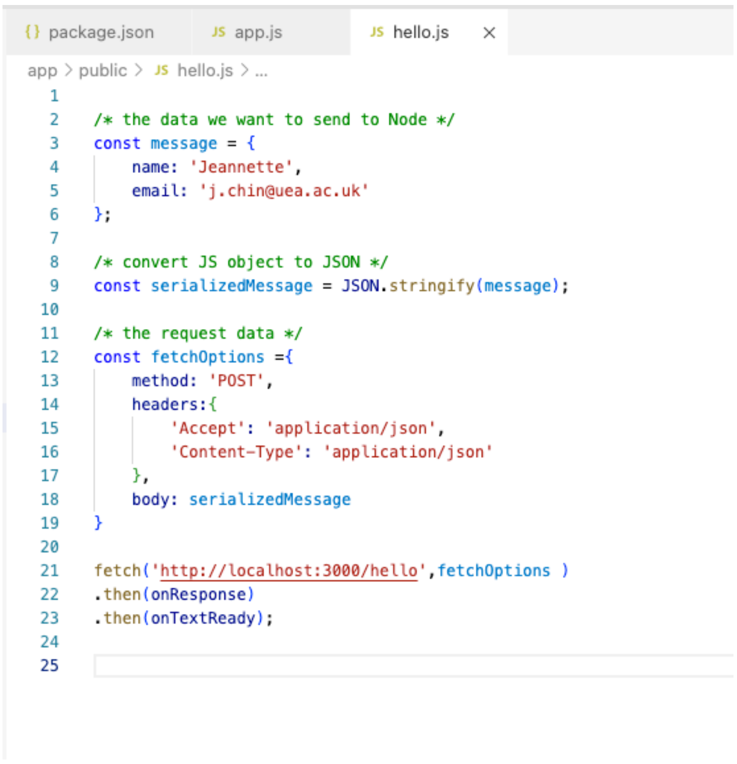
42. Notice that the variable `message` (Figure 43) is coded as **JavaScript object**. However, we want to send **JSON data** to the server. Fortunately we can convert JS object to JSON by using `JSON.stringify` method, as shown in Figure 44 , line 7



```

const message = {
    name: 'Jeannette',
    email: 'j.chin@uea.ac.uk'
};

const serializedMessage = JSON.stringify(message);


```

Figure 44. The code in hello.js (client code)

43. Now we will code our `fetchOptions`, the request data, as shown in Figure 45, line 12 – 19



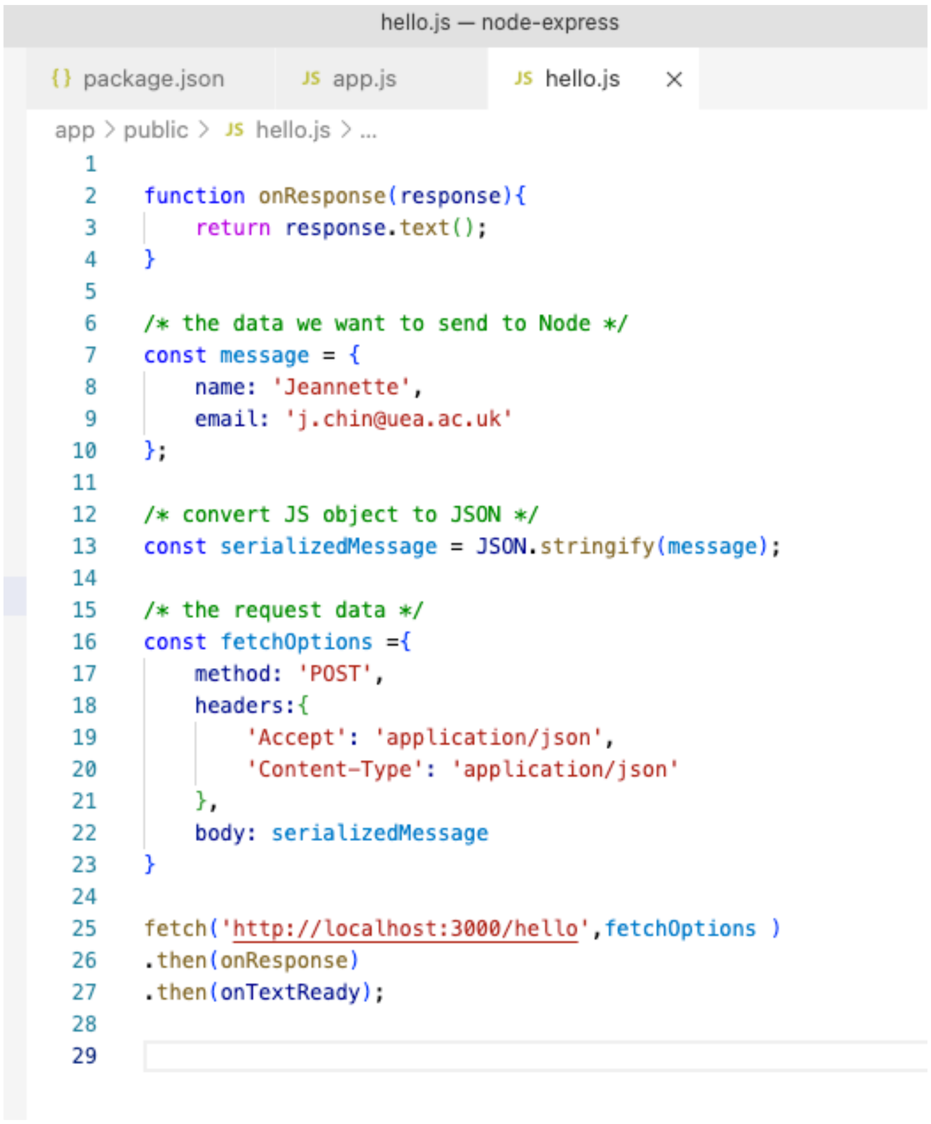
```
/* the data we want to send to Node */
const message = {
    name: 'Jeannette',
    email: 'j.chin@uea.ac.uk'
};

/* convert JS object to JSON */
const serializedMessage = JSON.stringify(message);

/* the request data */
const fetchOptions ={
    method: 'POST',
    headers:{
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    },
    body: serializedMessage
}

fetch('http://localhost:3000/hello',fetchOptions )
.then(onResponse)
.then(onTextReady);
```

Figure 45. The code in hello.js (client code)

Code explained: we use POST method, and header configurations to indicate the data is in JSON format, and the message is sent in the request body.

Now that the request data is completed, we will write the **callback** functions to handle the server response.

44. In `hello.js`, write a function called `onResponse`, the first **callback** function to handle the Promise returned by the server. In this example we are interested in the text returned by the Node server (see Figure 40 above). So the client code simply returns the text – as shown in Figure 46, lines 2 – 4.

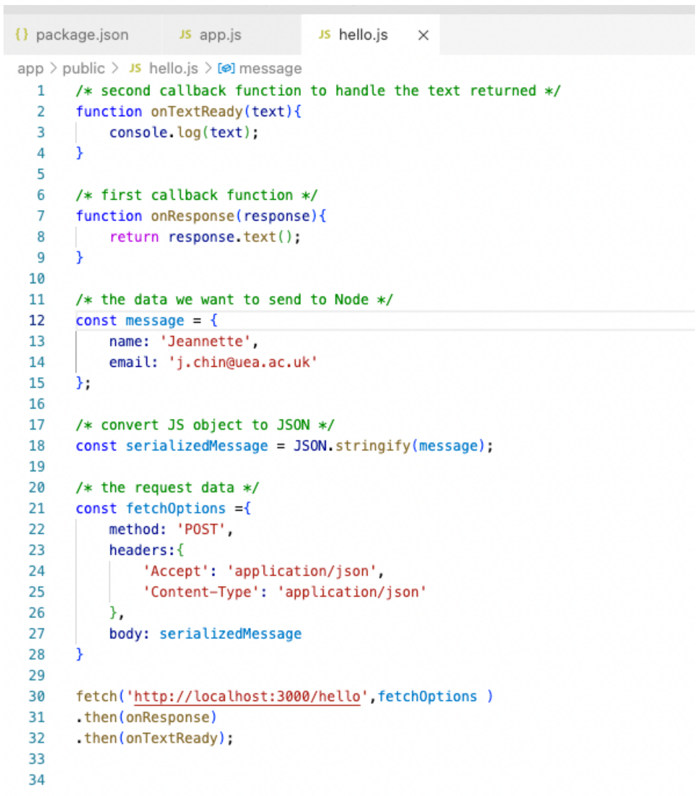


```
 1
 2  function onResponse(response){
 3      return response.text();
 4  }
 5
 6  /* the data we want to send to Node */
 7  const message = {
 8      name: 'Jeannette',
 9      email: 'j.chin@uea.ac.uk'
10  };
11
12  /* convert JS object to JSON */
13  const serializedMessage = JSON.stringify(message);
14
15  /* the request data */
16  const fetchOptions ={
17      method: 'POST',
18      headers:{
19          'Accept': 'application/json',
20          'Content-Type': 'application/json'
21      },
22      body: serializedMessage
23  }
24
25  fetch('http://localhost:3000/hello',fetchOptions )
26  .then(onResponse)
27  .then(onTextReady);
28
29
```

Figure 46. The code in hello.js (client code)

Note: The `response.text()` (line 3) returns a Promise, which is chained in our second `.then()` method in line 27. Next, we will write our second **callback** function "`onTextReady`" to handle the text being returned.

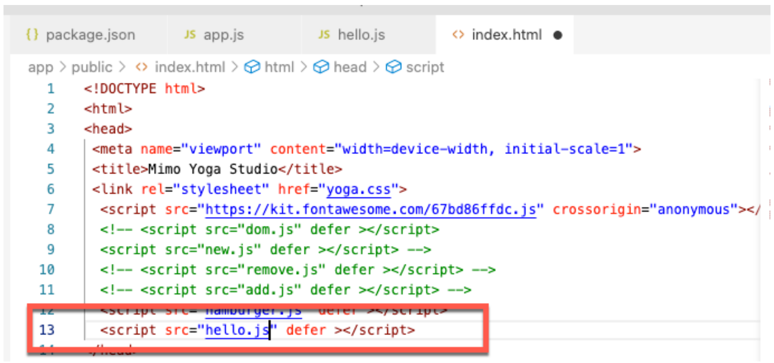
45. Write a function called "`onTextReady`" and takes the text returned as parameter and simply print it on the console, as shown in Figure 47, line 2-4.



```javascript
/* second callback function to handle the text returned */
function onTextReady(text){
    console.log(text);
}

/* first callback function */
function onResponse(response){
    return response.text();
}

/* the data we want to send to Node */
const message = {
    name: 'Jeannette',
    email: 'j.chin@uea.ac.uk'
};

/* convert JS object to JSON */
const serializedMessage = JSON.stringify(message);

/* the request data */
const fetchOptions ={
    method: 'POST',
    headers:{
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    },
    body: serializedMessage
}

fetch('http://localhost:3000/hello',fetchOptions )
.then(onResponse)
.then(onTextReady);
```

Figure 47. The code in hello.js (client code)

46. To test this client code, all we need to do is to add this script to `index.html` file, as shown in Figure 48, and save the file.



```html
<!DOCTYPE html>
<html>
<head>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <title>Mimo Yoga Studio</title>
 <link rel="stylesheet" href="yoga.css">
  <script src="https://kit.fontawesome.com/67bd86ffdc.js" crossorigin="anonymous"></
  <!-- <script src="dom.js" defer ></script>
  <script src="new.js" defer ></script> -->
  <!-- <script src="remove.js" defer ></script> -->
  <!-- <script src="add.js" defer ></script> -->
  <script src="hamburger.js" defer ></script>
  <script src="hello.js" defer ></script>
```

Figure 48. The code in index.html file

47. In Chrome, type this URL:

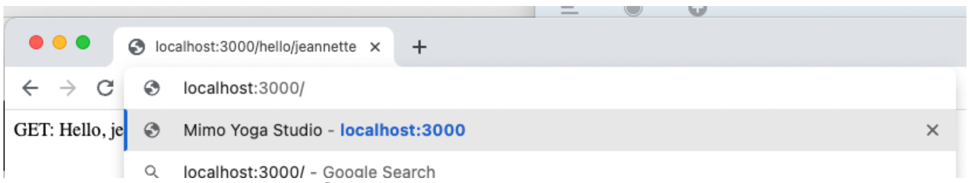

localhost:3000/hello/jeannette
localhost:3000/
GET: Hello, je
Mimo Yoga Studio - localhost:3000
localhost:3000/ - Google Search

Figure 49. Type localhost on Chrome

48. To see the output on the console, right click on the page and select “inspect” option (Figure 50):

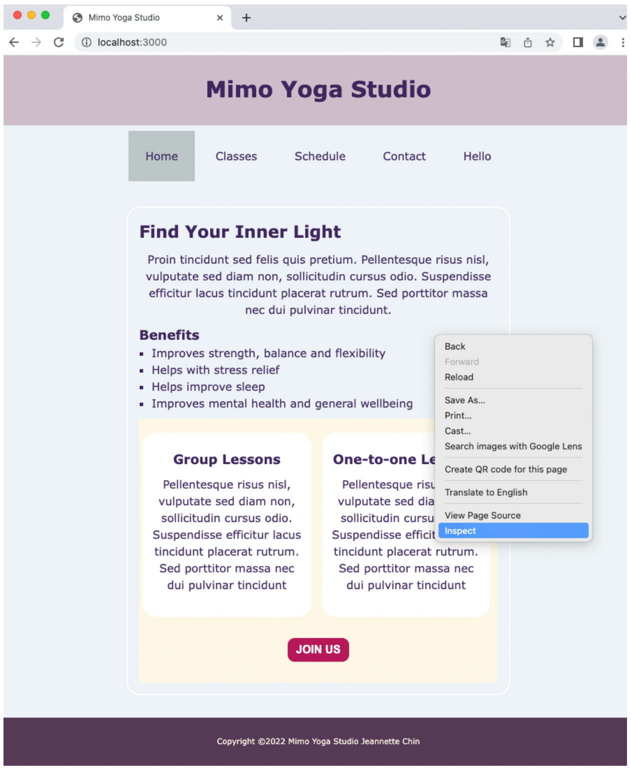
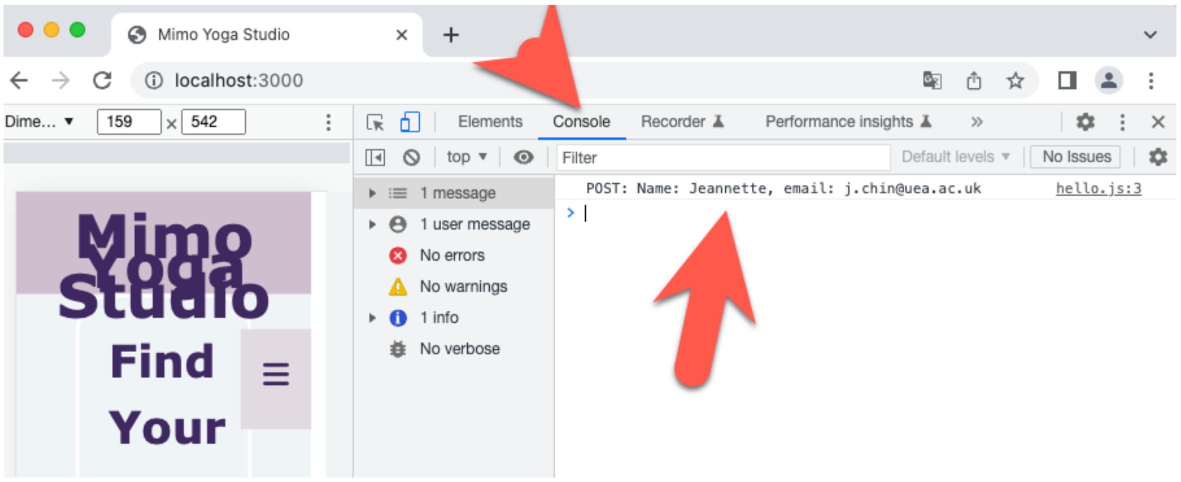


Figure 50. Select “Inspect” option

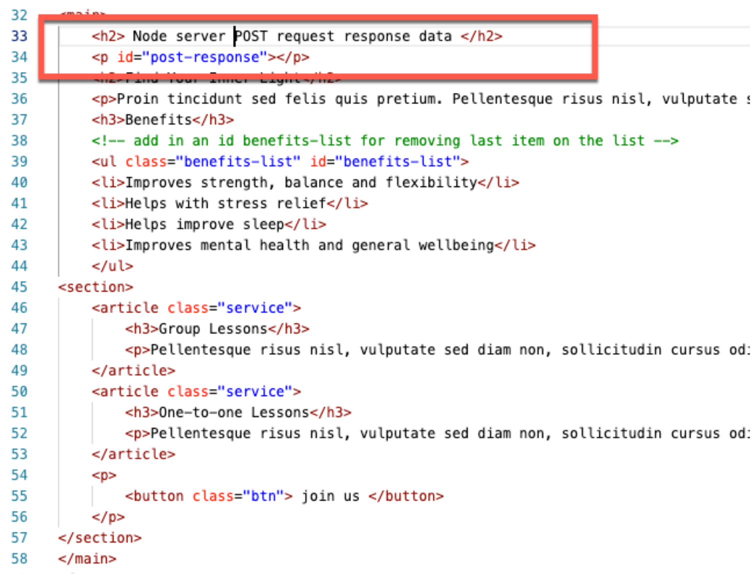
49. Select the Console tab, you should see the server response data here (Figure 51):



```
POST: Name: Jeannette, email: j.chin@uea.ac.uk        hello.js:3
```

Figure 51. View server response data on Console Tab

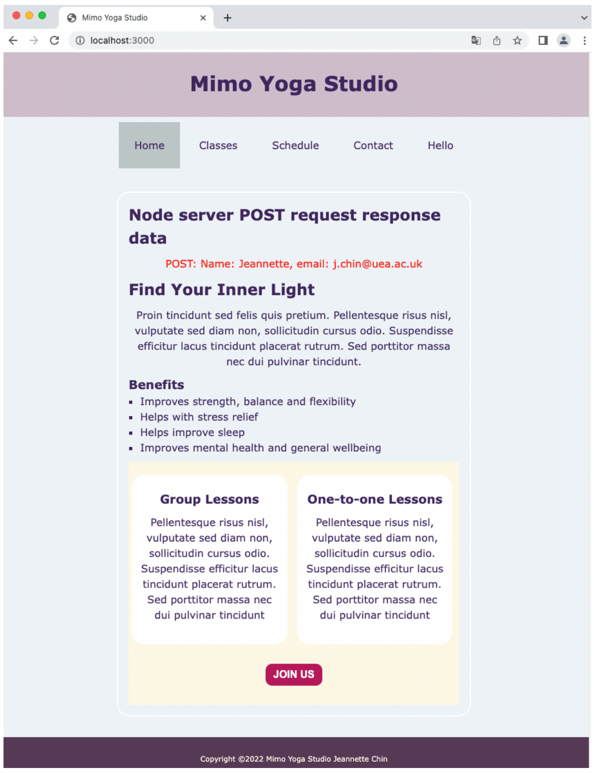
50. Now that our client JS code is working, your task is to render this response data on the web page. In your `index.html` page, create a new `h2` element with text “Node server POST request response data” and `p` element with an ID “post-response”, as shown in Figure 52, line 33 & 34.



```
32    <main>
33        <h2> Node server POST request response data </h2>
34        <p id="post-response"></p>
35        <h2>Find Your Inner Light</h2>
36        <p>Proin tincidunt sed felis quis pretium. Pellentesque risus nisl, vulputate
37        <h3>Benefits</h3>
38        <!-- add in an id benefits-list for removing last item on the list -->
39        <ul class="benefits-list" id="benefits-list">
40        <li>Improves strength, balance and flexibility</li>
41        <li>Helps with stress relief</li>
42        <li>Helps improve sleep</li>
43        <li>Improves mental health and general wellbeing</li>
44        </ul>
45    <section>
46        <article class="service">
47            <h3>Group Lessons</h3>
48            <p>Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus od
49        </article>
50        <article class="service">
51            <h3>One-to-one Lessons</h3>
52            <p>Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus od
53        </article>
54        <p>
55            <button class="btn"> join us </button>
56        </p>
57    </section>
58    </main>
```

Figure 52. The code in index.html

51. Modify your `onTextReady` function in `hello.js` file, so that the data is rendered inside this `p` element with an ID called “post-response”. The text colour should be in red. Once you are done, save the file and refresh the page. Figure 53 below shows my output for you to compare.


Mimo Yoga Studio

localhost:3000

Mimo Yoga Studio

Home Classes Schedule Contact Hello

Node server POST request response data

POST: Name: Jeannette, email: j.chin@uea.ac.uk

Find Your Inner Light

Proin tincidunt sed felis quis pretium. Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus odio. Suspendisse efficitur lacus tincidunt placerat rutrum. Sed porttitor massa nec dui pulvinar tincidunt.

Benefits

- Improves strength, balance and flexibility
- Helps with stress relief
- Helps improve sleep
- Improves mental health and general wellbeing

Group Lessons

Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus odio. Suspendisse efficitur lacus tincidunt placerat rutrum. Sed porttitor massa nec dui pulvinar tincidunt

One-to-one Lessons

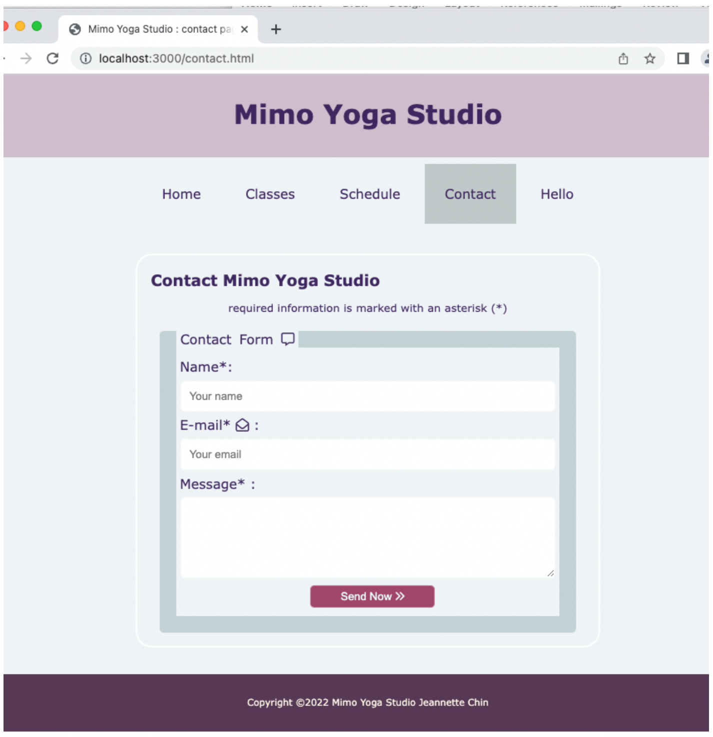
Pellentesque risus nisl, vulputate sed diam non, sollicitudin cursus odio. Suspendisse efficitur lacus tincidunt placerat rutrum. Sed porttitor massa nec dui pulvinar tincidunt

JOIN US

Copyright ©2022 Mimo Yoga Studio Jeannette Chin

Figure 53. The output of index.html

52. We are now ready to handle our Form data. Recall the HTML form in our `contact.html` page (Figure 54), we have 3 text fields.


Mimo Yoga Studio : contact pa

localhost:3000/contact.html

Mimo Yoga Studio

Home Classes Schedule Contact Hello

Contact Mimo Yoga Studio

required information is marked with an asterisk (*)

Contact Form

Name*:

Your name

E-mail* :

Your email

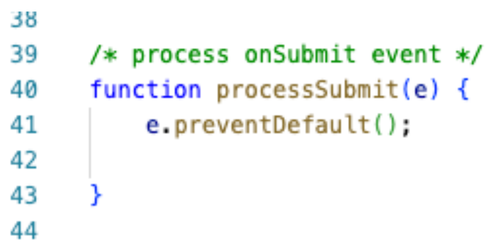
Message* :

Send Now »

Copyright ©2022 Mimo Yoga Studio Jeannette Chin

Figure 54. The output of contact.html

53. Your tasks are:
   a) Write a client-side script called “`form.js`” to capture the form data
      i. Grab the form element using `document.querySelector` method
      ii. We will want to listen to `submit` event (**Hint**: use `addEventListner` method)
      iii. Write some code for the event handler called “`processSubmit`”
      iv. In this `processSubmit` function:
         a. takes an event object `e` as parameter
         b. we will want to prevent the prevent the browser submits the form automatically. We can use `e.preventDefault` method (MDN) for this purpose. The skeleton code is shown in Figure 55 below:

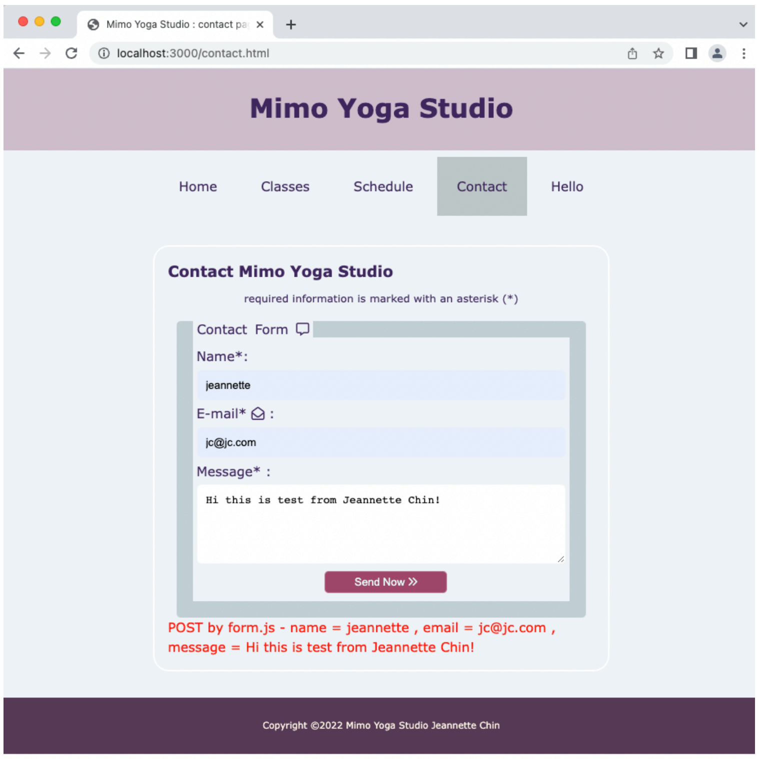


```
/* process onSubmit event */
function processSubmit(e) {
    e.preventDefault();

}

```

Figure 55. The skeleton code for processSubmit handler.

         c. Next, retrieve the form data for each field (i.e, name, email and message) [**Hint**. Use `querySelector` method to grab the element, then use `.value` to access associated value]
         d. Once you got the above working, process the data [**Hint.** Use similar approach as in `hello.js`]
         e. Create fetch option data [**Hint.** Use similar approach as in `hello.js`]
         f. Use fetch API to send the data to Node server using this URL http://localhost:3000/form and handle the returned Promise with 2 call back functions – `onResponse` and `onTextReady` [**Hint:** Use similar approach as in `hello.js`]

      v. The `onTextReady` function should render the returned text to `contact.html` page, below the form (see Figure 56) [**Hint.** Create a `div` element on `contact.html` with an ID such that your JS code can pick up this element]. **Note**: to see the result in Figure 56, you will need to complete the server code first, follow task (b) below.



Mimo Yoga Studio : contact pa
localhost:3000/contact.html

Mimo Yoga Studio

Home Classes Schedule Contact Hello

Contact Mimo Yoga Studio

required information is marked with an asterisk (*)

Contact Form

Name*: jeannette

E-mail* : jc@jc.com

Message* : Hi this is test from Jeannette Chin!

Send Now »

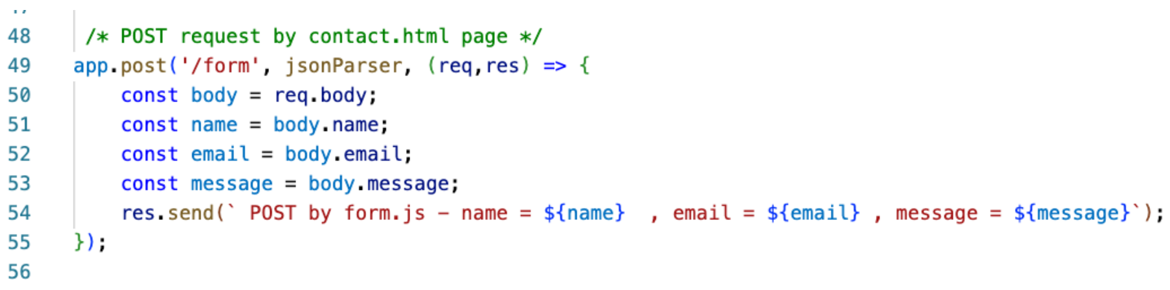
POST by form.js - name = jeannette , email = jc@jc.com , message = Hi this is test from Jeannette Chin!

Copyright ©2022 Mimo Yoga Studio Jeannette Chin

Figure 56. The output of contact.html page

b) On **server side** coding, in your `app.js` file:

   i. create a new **POST** function with this URL **"/form,……"**, using `app.post()` method, and takes a call back function, request `req` and response `res` objects as parameter. Use JSON body parser to help with the data. The code is shown in Figure 57 below.



```
 /* POST request by contact.html page */
app.post('/form', jsonParser, (req,res) => {
    const body = req.body;
    const name = body.name;
    const email = body.email;
    const message = body.message;
    res.send(` POST by form.js - name = ${name}  , email = ${email} , message = ${message}`);
});
```

Figure 57. The skeleton code on app.js

c) Once you have coded the functions, save the files.

d) Test the form submit function on `contact.html` page. [**Hint.** Don't forget to add this `form.js` in your `contact.html`]. Figure 56 above is my output for you to compare.