Node/Express

BACKEND DEVELOPMENT

CSIS 3380 *FULL STACK DEVELOPMENT WITH JAVASCRIPT*

ANU GUPTA

guptaa10@douglascollege.ca

Outline

Introduction to Backend

Node/Express

Native Modules – Local File System

External Modules

NPM - NPM init and NPM install

Express

- Creating our First Server
- Handling Requests and Responses
- GET and POST requests
- Understanding and working with routes
- Examples

Back End Development

Front end technologies, like HTML, CSS and JavaScript, as well as jQuery and Bootstrap, enable us

- to design our web site and
- also allow for user interaction on the client side, in the browser for example.

Back end technologies like Node.js, PHP, Ruby on Rails, ASP.Net, Python, Java and many more enable us

- to interact with databases
- have business logic on a server and a whole lot more.

Full stack basically is Front end development + Back end development.

Back End Development

On the back end there's a lot of technologies that are commonly used , and these include PHP, Ruby, Java, C#, Node.js amongst many others.

Now, in addition, there are frameworks, such asCakePHP, Ruby on Rails, Spring, ASP. Net, Express.

Frameworks **speed up development using the particular back end technology you choose.**

The main job of these frameworks is to simply **reduce the amount of repetitive work that a developer has to do.**

Back End Development

A Framework provides pre-built components and structures that can be used to build an application without the need for us to write every single line of code from scratch.

We're going to be learning about Node.js as well as the most common framework that's used with it, which is **Express**.

- No need to learn a brand new programming language to do it, because Node.js runs on Javascript.

This is why Node.js is one of the most popular back end languages currently, because it runs on Javascript.

Node.js

Node.js is used to create a backend using Javascript.

Node is also superfast and will allow us to create really **scalable and fast running web apps**.

And we're not the only ones choosing Node.

Web sites such as LinkedIn, Twitter, Netflix, all use Node.js in their backend, and more and more companies are joining their ranks and starting to use Node.

Node.js

Create a new file index.js and type:

- **Console.log(“Hello World!!”);**

Up till now, we've only been able to run this code if we incorporated it into an HTML file and opened it inside our browser.

But now we're going to run this code using our computer, and we can do that by simply being inside the folder where that file exists. Type:

- **node index.js**

And now, if you hit enter, you can see that the result of that file “Hello world!!”

Node.js API

When you install Node, it already comes bundled with a whole bunch of built**in modules**.

These are essentially **libraries of code** that the Node team wrote so that it is easier for us to create applications, especially on the server side.

For example, we can **use Node.js to get access directly to the local files of the computer.**

If you head over to nodejs.org/api, then you can see all of the native modules and the documentation of how you can use it.

File System

Now if you read the documentation, there's a module called the**file system.**

It is the native node module that helps interact with the**local file system/access local storage**.

- There are methods and properties that you can tap into, but in order to use the module **we first have to require it.**

Using **fs** module

In order to use libraries of code, or modules, or packages, inside our project, we have to require the module.

- **const fs = require (“fs”)**

So now that **we've required fs, that means we've incorporated this file system module into our project**, and we're now able to use it in our project.

Using fs module

What are the sort of things that you can do using file system?

open files

change files

access the local file system

read and write to the local file system.

Using **fs** module

So inside the current folder, create a new file, **file1.txt.**

So this is just a pure simple text file that contains some text **“I am file 1.”,** hit save,

Type: **fs.copyFileSync("file1.txt", "file2.txt");**

So, this line of code will look within the current directory, it will look for a file that's called file1.txt, and then it will copy it to something called file2.txt.

Run the following command:

node index.js

We will have two files, file1, which still has the same content as before, and also file2, which now has the copied content from file1.

So if file2 already existed, then this copyFileSync **will simply replace the existing content**.

Using **fs** module

Explore fs.writeFile() and readFile() methods.

It allow us to take a message that may be user inputs and write it into a file.

```
fs.writeFile("message.txt", "Hello from node.js", (err) => {
    if (err) throw err;
    console.log("The file has been saved.");
})
```

Using external modules with NPM

External modules: These are not the modules that come bundled with Node but actually these whole bunch of libraries that other people have written which we can use through **NPM.**

NPM is a package manager for external modules.

NPM itself actually stands for **Node Package Manager**, and it's currently the world's largest collection of these packages of code.

- It's open source and you can search to find all the packages that other developers have created

NPM is prebundled with Node, so when we installed node, we already installed NPM.

Creating a node.js project

Use **npm init** to initialize a new Node.js project.

When you run this command, npm will interactively ask you a series of questions to gather information about your project, such as its name, version, description, entry point, and more.

It will then generate a package.json file based on your responses.

The **package.json** file contains information about the project dependencies.

npm init

Create a folder (week3) and initialize npm in it.

week3 > npm init

Complete the steps.

A new file called package.json will be created as part of the process.

Now we call start installing the npm packages.

Check https://www.npmjs.com/

Examples

```
const superheroes = require("superheroes")

const pokemon = require('pokemon');

const yesNoWords = require('yes-no-words');


console.log(superheroes.random());

console.log(pokemon.random());

console.log(pokemon.all()); //=> ['Bulbasaur', …]

console.log(yesNoWords.yesRandom());
```

Express

Express is a JavaScript framework that allows us to create backends for our websites.

It simplifies the process of building web applications and APIs by providing a set of pre-built features and middleware.

Benefits:

- Better readability
- Write less code
- Ability to add middleware

Create our first server with node.js and Express

1. Create a new folder, call it **my-express-server**
2. Open the command prompt and navigate to that folder.
3. Then inside that project folder, create a new file called **server.js**
4. Initialize npm. → npm init
5. Set the **starting point as server.js** and complete the remaining steps.
6. **install Express package (also check the Express documentation at expressjs.com)**
7. Start the server

First Node/Express server

In the code snippet below, the constant app is assigned to the value returned by express() function.

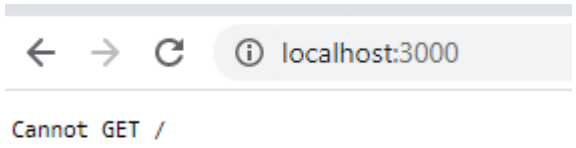
The express() function returns a new instance of an Express application.

```
import express from "express";
const app = express();

app.listen(3000, () => {
    console.log("server is up and listening on port 3000.");
})
```

Handling Requests and Responses: the GET Request

Open the browser window and type the url http://localhost:3000/, you see



What does it mean??

http requests

we make these http requests, there are five main words that you'll come across:

Get – request a resource from the server

Post – sending resource to the server (e.g., submitting form data)

Put – update a resource by replacing it completely

Patch – update a part of the resource (patch up a resource)

Delete - delete a resource

Making get requests

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

This is a method that's provided by Express that allows us to specify what should happen when a browser makes a get request.

The first parameter specifies the route that the server is going to respond to.

The second parameter is the callback.

nodemon

nodemon is a tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected.

npm i nodemon

Endpoints

Endpoint: endpoints are specific routes that your app or API reveals in order to handle incoming HTTP requests.

An API **endpoint** is a specific location within an API that accepts requests and sends back responses.

It's a way for different systems and applications to communicate with each other, by sending and receiving information and instructions via the endpoint

Understanding and working with Routes

If we create another app.get method, and, instead of targeting the home route, target maybe the contact route or about route, or so on.

```
app.get("/about", (req, res) => {
    res.send("<h1>My name is Anu!!</h1>");
})
app.get("/contact", (req, res) => {
    res.send("<h1>My email is anu@gmail.com</h1><h2>My phone number is 1234567890</h2>");
})
app.get("/hobbies", (req, res) => {
    res.send("<ul><li>Music</li><li>Travelling</li></ul>");
})
app.get("*", (req, res) => {
    res.send("The route is unknown")
})
```

HTTP response status codes

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

1. Informational responses (`100` – `199`)
2. Successful responses (`200` – `299`)
3. Redirection messages (`300` – `399`)
4. Client error responses (`400` – `499`)
5. Server error responses (`500` – `599`)

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Making other types of http requests

Test the response in Postman or install ThunderClient VS code extension.

```
//test in postman
app.post("/register", (req, res) => {
    // res.send("A post request has been made");
    res.sendStatus(201);
})
app.put("/user/login", (req, res) => {
    res.sendStatus(200);
})
app.patch("/user/login", (req, res) => {
    res.sendStatus(200);
})
app.delete("/user/login", (req, res) => {
    res.sendStatus(200);
})
```

Express Middleware

Express middleware refers to a set of functions that execute during the processing of HTTP requests received by an Express application.

Middleware functions access the HTTP request and response objects. They either terminate the HTTP request or forward it for further processing to another middleware function.

We can also use middleware for authentication. So before we let the request through to our back end handlers, we can see if that request is actually coming from a client that is authorized to make that request.

Middleware can also process any errors in the requests, identifying them and handling them before they go through to the handlers as well.

Body-Parser

Commonly used Middleware which parses incoming request bodies.

It is available under req.body property.

Commonly used to handle form data.

Build a calculator app

1. Make a new folder called Calculator on your Desktop
2. Change Directory to this new folder
3. Inside the Calculator folder, create a new file called **calculator.js**
4. Set up a new NPM package
5. Open the project folder in VS Code
6. Using NPM install the **express and body-parser** modules
7. Setup express
8. Create a root route get method with app.get() that sends a file (calculator form) to the browser.
9. User fills in the data and send it to the server in a POST request
10. Handle post request at server and send the results back to client.

__dirname constant

The constant __dirname in a Node script gives you the path of the folder where the current JavaScript file resides.

How to use it inside an ES module?

```
import path from 'path';

import { fileURLToPath } from 'url';

const __filename = fileURLToPath(import.meta.url);

const __dirname = path.dirname(__filename);
```

bodyParser.urlencoded({ extended: true }))

bodyParser.urlencoded([options])

Returns middleware that only parses urlencoded bodies and only looks at requests where the Content-Type header matches the type option. This parser accepts only UTF-8 encoding of the body.

A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body). This object will contain key-value pairs, where the value can be a string or array (when extended is false), or any type (when extended is true).

The app. use() function is used to mount the specified middleware function(s) at the path which is being specified. It is mostly used to set up middleware for your application. Syntax: app.use(path, callback)