node.js

The Server-side JavaScript

Joreen Arigye

Recap

- **Node Js**
  - an environment to run JavaScript on the back end, outside the browser
  - Server side Javascript
  - Platform built on Chrome's V8 JavaScript engine
  - Event driven - works on a non-blocking model (LET’s read more about this)
  - Extremely fast and efficient
  - Highly Scalable

Objectives today:

- **All going to write your first node program**
  - In the process learn about:
    - Logging simple output
    - NPM (Node Package Manager)
    - Express
    - Call backs
    - Nodemon
    - Package.json
      - Scripts

Our journey to `Hello World` web app

1. Create a folder for this project in your VS code
   a. Name it whatever you want

2. Navigate into it in your command line, run the `npm init` command
   a. This command creates a `package.json` file which helps you manage libraries/modules that we will keep installing as we learn them.

3. Let's create a file and call it `server.js` to run node with.
   a. Before we write any code in this file and execute it, we want to check that this file will run properly by simply adding `console.log('Hello World')` statement to it

4. Now, run `node server.js` in your command line and you should see the statement you logged — — `` `Hello World` ``

New terminology

What is `NPM`?

- NPM is a package manager for Node.js packages, or modules if you like. www.npmjs.com hosts thousands of free packages to download and use.The NPM program is installed on your computer when you install Node.js

What is a `Package`?

- A package in Node.js contains all the files you need for a module. Modules are JavaScript libraries you can include in your project.

New terminology

What is a `PACKAGE.JSON` file?

- Provides high-level details of the project
- Different ways to run it.
- What libraries it uses.
- The name, description, and version.

What about `console.log`

- Helps show simple output before things get complicated

Our journey to `Hello World` web app

Now that our `server.js` file runs well, let's add some real code

1. In order to render/show some data to the browser where we shall be viewing our app. We need a node.js framework called `Express`

2. We first have to install `Express` before we can use it in our application
   a. All we have to do is run an install command with Node package manager (npm), which comes bundled with Node. Run the `npm install express --save` command in your command line
   b. Once you're done, you should see that npm has saved `Express` as a dependency in package.json
3. Next, we use `express` in `server.js` by requiring it.

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

Our journey to `Hello World` web app

Now that we have our Express app.

1. What we want to do is to create a server where `browsers` can connect. We can do so with the help of a `listen` method provided by Express:

```
app.listen(3000, function() {
console.log('listening on 3000')
            })
```

2. Now, run `node server.js` and navigate to `localhost:3000` on your browser. You should see a message that says "`cannot get /`".
    a. The READ operation is performed by browsers whenever you visit a webpage.
    b. Under the hood, browsers send a `GET` request to the server to perform a READ operation.
    c. The reason we see the "cannot get /" error is because we haven't sent anything back to the browser from our server.

Our journey to `Hello World` web app

Now in that we have a READ operation from the browser.

1. In `Express` we handle a `GET` request with the `get` method:

```
app.get(path, callback)
```

   a. The first argument, `path`, is the path of the `GET` request. It's anything that comes after your domain name.
      i. When we're visiting `localhost:3000`, our browsers are actually looking for `localhost:3000/`. The `path` argument in this case is /
   b. The second argument is a `callback function` that tells the server what to do when the `path` is matched. It takes two arguments, a `request` object and a `response` object:

```
app.get('/', function (req, res) {

   res.send('Hello World')

})
```

New terminology

What is `Express`?

- Express.js is a web application framework for Node.js, released as free and open-source software under the MIT License.
- It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

Some features of `Express:`

- MVC, Routing,Session Support, Middleware etc
- **Research about EXPRESS**

What is a `require`?

- Used include functionality from other packages,modules and files

New terminology

What is a `call back`?

A Callback is simply a function passed as an argument to another function which will then use it (call it back).

Our journey to `Hello World` web app

We are going to start writing in ES6 code:

1. First off, We replace `function()` with an ES6 arrow function. The below code is the same as the above code:

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

2. Now, restart your server by doing the following:
   a. Stop the current server by hitting CTRL + C in the command line.
   b. Run `node server.js` again.
3. Navigate to `localhost:3000` on your browser. You should be able to see a string that says “`Hello World`”.

Our journey to `Hello World` web app

At this point, you probably have realized that you need to restart your `server` whenever you make a change to `server.js`. This is process is incredibly tedious, so let's take a quick detour and streamline it by using a package called nodemon.

1. `Nodemon` restarts the server automatically whenever you save a file that the server uses. We can install Nodemon by using the following command:

   ```
   npm install nodemon --save-dev
   ```

   a. The reason we're using a `--save-dev` flag here is because we're only using Nodemon when we're developing. This flag would save Nodemon as a devDependency in your package.json file.

   b. Nodemon behaves exactly the same as node, which means we can run our server by calling `nodemon server.js`. We do this by adding `nodemon` to the scripts part of the `package.json`

Our journey to `Hello World` web app

```
"scripts": {

 "start": "nodemon server.js"

     }
```

1. Now, you can run `npm start` to trigger nodemon server.js.

Don’t forget to use GITHUB!