Intro to Node Lab

In the last lab we created an environment that lints, transpiles, bundles and minifies our JavaScript code which would run in a browser. But as of now we have no way to get it to a browser. We need a web server.

In this lab we'll learn about node by creating a bona-fide web server with Node and Express. These two are usually used together.

Let's start with the node/express web server.

Creating your node web server script

1. In your existing project, run this command:

```
npm install --save-dev express
```

2. In the root of your project (not in src) create a new file called server.js and put this in it:

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

   These are pulling in needed libraries.

3. Add some lines to process http requests:

```
const app = express();
const port = process.env.PORT || 5000;
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.get('/marco', function (req, res) {
  return res.send("polo");
});
app.listen(port, () => console.log(`Listening on port ${port}`));
```

4. Let's test it! Open a new terminal window and run

```
node ./server.js
```

It will tell you that it is waiting for a request on port 5000.

5. Open a browser and navigate to http://localhost:5000. It should give you a 404 Not found.
6. Now navigate to http://localhost:5000/marco. Did it respond with "Polo"? If so, we now have a web server listening for web requests and serving responses. But "Marco" and "Polo" are kind of limiting. Let's tell it to serve real pages.

Serving real pages

7. Edit server.js. Add this before the app.listen:

```
app.use(express.static(path.join(__dirname, 'dist')));
```

   This tells node to look in the "dist" directory for any files asked for and serve them.

8. Stop the server and re-start it. Navigate to http://localhost:5000/index.html. You should be seeing the index.html page you created earlier.

Creating a development environment

Think you might get tired of having to stop and re-start the server each time you make changes? It would be cool if we could set up the system so that every time you save a change to the server it re-starts automatically. That's what nodemon does.

9. Install nodemon via npm.

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

10. Edit package.json. Create a new "script" entry. Make it look like this:

```
"server": "nodemon server.js"
```

11. Open a terminal window and type in

```
npm run server
```
Assuming that there were no errors, you should be able to browse to localhost:5000 and see index.html in your browser.

12. Make a change to server.js. Any change will do. If you want a suggestion, you can change the line where you listen for "marco". Have it respond with something other than "polo".
13. When you hit save, notice that the node web server automatically restarts.

14. Let's do the same for our client-side source code with webpack. Edit package.json and add a new script entry:

```
"build-dev": "eslint src --ext .js && webpack --watch",
```

15. Save. Then run "npm run build-dev". Browse to your dist/index.html page.
16. While it is running, make a change to index.js and save it. Notice that webpack sees the change and re-compiles!

Now this is cool and all but npm can't run both node and webpack in watch mode without a little assist. We're going to give it that assist with a final library called "concurrently".

17. First, install concurrently using npm:

```
npm install concurrently --save-dev
```

18. Then add this as a script:

```
"dev": "concurrently --kill-others \"npm run build-dev\" \"npm run server\""
```

19. Save. Then run this:

```
npm run dev
```

20. Your client-side source code should compile and your webserver should start up. Any change to any source code file should trigger a re-compile and restart. Try it out!

Kind of cool, right? Once you've got them both running, you can be finished with this lab.