Express.js

Learning Objectives

- Understand what Express is
- Understand how to serve static content with Express
- Understand how to server dynamic content with Express

Material

What is Express?

Express is a web framework for Node.js. It allows us to easily create servers using JavaScript.

Creating a Server with Express

To use Express, we first need to install the module using:

```
npm install express
```

Then we create a JavaScript file (in this example, we'll call it `server.js`) and add code to start up a web server listening on a specific port (a port is simply an arbitrary number which labels a particular process running on a server).

```
// server.js
const express = require('express');

const app = express();
const port = 3000;

app.listen(port, () => {
    console.log(`Server listening at http://localhost:${port}`);
});
```

This is our minimal server: we create an `app` object using Express and start our server by calling its `listen` method. If you run this, you'll notice the process doesn't end like a normal JavaScript file, it goes on indefinitely. Our computer is now acting as a server: it will listen for incoming requests on port 3000 until we tell it to stop (e.g. by pressing ctrl+c in the terminal).

Our server currently has nothing to serve. Let's create some static content for our server to return. `index.html` is the default page a web server will respond with when requests are made to the root path of the server (in our case http://localhost:3000). We'll therefore create an `index.html` with the following content and save it to the `public/` folder.

```
<!DOCTYPE html>
<html>
    <head></head>
    <body>
        <h1>Hello World</h1>
    </body>
</html>
```

We need to also tell Express explicitly to serve static files from the public folder:

```
// server.js
const express = require('express');

const app = express();
const port = 3000;

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

app.listen(port, () => {
    console.log(`Server listening at http://localhost:${port}`);
});
```

Our directory now looks like this:

```
app/
├── package.json
├── package-lock.json
├── server.js
├── node_modules/
├── public/
│   ├── index.html
```

If we run server.js and visit localhost:3000 in our browser, we should see our HTML page rendered to the screen.

Serving Dynamic Resources

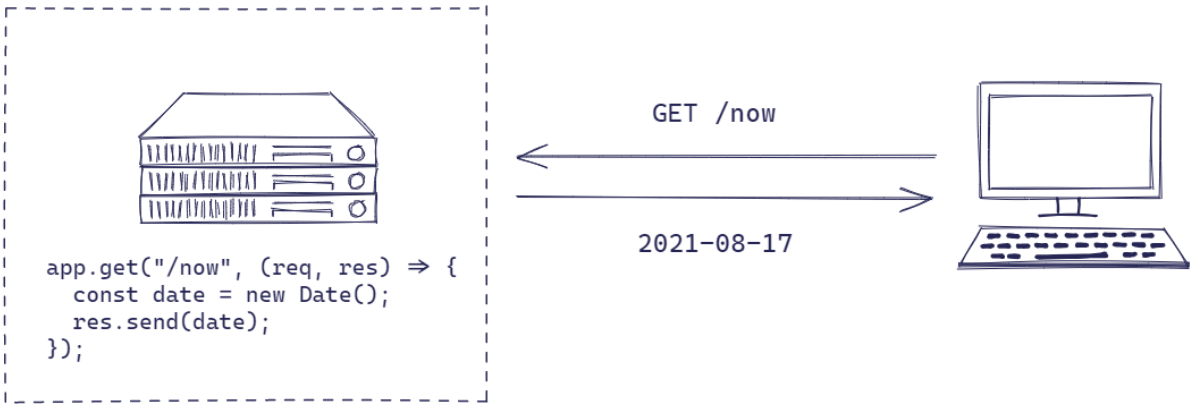
So far we've only used Express to serve a static file. To serve dynamic resources, we need to create an endpoint:

```
// server.js
const express = require('express');

const app = express();
const port = 3000;

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

app.get("/now", (req, res) => {
  const date = new Date();
  res.send(date);
});

app.listen(port, () => {
    console.log(`Server listening at http://localhost:${port}`);
});
```

The app.get is telling Express that this endpoint is listening for HTTP GET requests. Remember, GET is the HTTP method used when you click a link or type in a URL. A request using a different HTTP method e.g. `POST http://localhost:3000/now` would not trigger this endpoint's callback.

The `/now` is telling Express that this endpoint is responsible for requests with a `/now` path. If we decided to change this (say, to `/date`) a client trying to access `http://localhost:3000/now` would no longer hit this endpoint.

The 2nd argument to `app.get` is the callback function that fires when a request comes in which satisfies the endpoint's method+path combination. The `req` object contains information about the incoming request. The `res` object has methods used to send response data back to the client.



Currently, our server only has one endpoint, but we could add more. We call the entire set of endpoints for a service its Application Programming Interface (API). Just like a toaster has a User Interface (UI) - knobs, levers and buttons - which we use if we want to make things happen inside the toaster, so a service has an API which code running outside the service has to use if it wants to make things happen inside the server.

Core Assignments

Express Basics

Create a basic express server.

Create an index.html page and serve this using express.

Create an endpoint to listen for GET requests with the path /flipcoin. This endpoint should randomly respond with the text "heads" or "tails". Verify your code works by starting your server and visiting http://localhost:3000/flipcoin You should see the text displayed. Use Chrome Developer Tools to see the HTTP requests as you refresh the page.

Extension Assignments

Sudoku

Create an express endpoint that solves Sudoku puzzles. Users should be able to send their uncompleted sudokus as JSON and the endpoint should return the completed version.