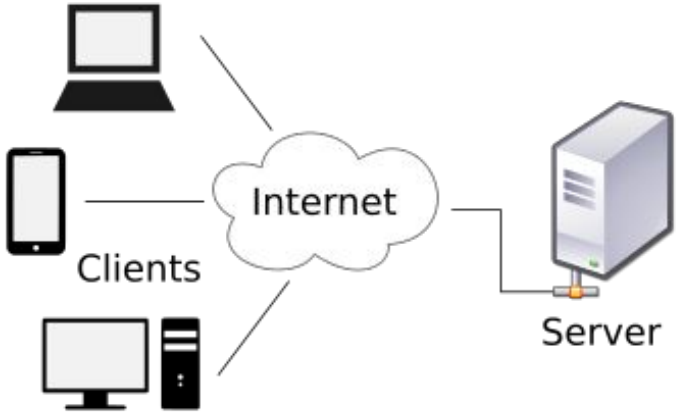
Week 9 Assignment

Introduction to Node

Server Side Programming

Previously, we have written code that is served to the user and run in the browser. Javascript listens for user behavior and executes code that manipulates the DOM as a result.

Server side programs are never given to users. They run constantly on the server, waiting for requests the same way JS listens for events. They will then run code and provide a response back to the client.



Node.js

We will be using Node to run Javascript as server side code. Using modules like Express, we will be making forward-facing APIs like the ones you have been using with AJAX fetch.

We will define API endpoints, and then provide the code that should be run if someone requests from that endpoint. In doing so, we can read their query parameters and respond with relevant data.

It is now absolutely critical that you have Node properly installed.

Basic app.js structure

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


// define all endpoints here


app.use(express.static('public'));
const PORT = process.env.PORT || 8000;
app.listen(PORT);
```

Our first endpoint

```
app.get('/hello', function (req, res) {
  res.type('text');
  res.send('Hello World!');
});
```

`app.get(...)`	Method to handle GET requests
`app.post(...)`	Method to handle POST requests
`req`	Request object, holds items like the request parameters
`res`	Response object, has methods to send data to the client
`res.type(...)`	Sets the "content-type". Always set either "text" or "json" with your response
`res.send(response)`	Sends the response to the client
`res.json(response)`	Does the same as `res.send`, but with a JSON object

Setting up the project and running the server

- Assuming your project has the correct directory structure, run the command `npm init` at the root
- You will need to install any non-core modules using npm. Run the command `npm install <package-name>` to do so.
- Open a terminal in the directory with the server and enter `nodemon`. `nodemon` is a tool that restarts the server if you make changes to the JS code in order to reflect the changes.
  - To stop the server, enter `ctrl+c` in the terminal.
- Access your page in the browser. Since the server is being hosted locally on your machine, use the URL `localhost:8000/hello`. 8000 is the port we specified in `hello.js`.
- Since we told the server to serve files in the "public" directory, we can access our website with the url `localhost:8000/hello.html`.

Route parameters

You may recognize this from the number trivia API we used at the beginning of AJAX. We can accept parameters directly inside of the URL with node.JS by putting them in the endpoint we pass into `app.get:`

Node.JS: `app.get('/hello/name/:first/:last', ...)`
Request: `localhost:8000/hello/name/dwayne/johnson`

The colon before the param name implies that it is a key, whose value is passed in by the user when they make the request. In your code, you can access these parameters in a similar way to query parameters:

Node.JS: `let firstName = req.params['first'];`

You do not need to check that these are defined, because in order for them to use this endpoint, they **must** have the parameters.

Query Parameters

In Node.JS, we can access user supplied GET parameters as follows:

Request: `localhost:8000/hello?key=value`
Node.JS: `let value = req.query['key']`

It is important, though, to make sure that the user actually supplied the parameter, since we will get `undefined` otherwise. **Always check if parameters are defined before accessing them**

Node.JS: `if (req.query['key']) { ... }`

Setting Error Codes

- In the case that you encounter an error, you'll want to send a request with an error code. If you just send a regular response, it will not cause `checkStatus` to fail, nor will it communicate that it is an error
- Remember to always set the type of the response - even when handling errors
- We can use the `status` method of the response object to set the code.

```
res.type('text').status(400).send('Error, Bad Request!');
```

Example endpoint

```
'use strict';

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

app.get('/hello', function(req, res) {
  res.type('text');
  res.send('Hello World!');
});

const PORT = process.env.PORT || 8000;
app.listen(PORT);
```

Exercise 1: Splendid Circles

Add a new GET endpoint, `/math/circle/:r`, which takes a radius as a URL parameter. It should then respond in JSON with the area and circumference.

```
{"area": 3.14, "circumference": 6.28}
```

The area of a circle is `PI * r * r`, and the circumference is equal to `PI * 2r`. You can access PI with `Math.PI`.

app.js -- copy this file and add to it for your exercises

```
'use strict';

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

// define endpoint for exercise 1 here
// define endpoint for exercise 2 here


const PORT = process.env.PORT || 8000;
app.listen(PORT);
```

Exercise 2: Hello, you!

Add a new GET endpoint, `/hello/name,` which takes as query parameters a `first` and `last` parameter. It should then respond in plain text with "Hello *firstName lastName*" replacing *firstName* with the value of `first` and lastName with the value of `last`

If they fail to provide the necessary GET parameters, respond with a 400 status code, and the error message "Missing Required GET parameters: first, last". If they provide one, but not the other, your error message should only have the missing parameter listed.

Turn in your app.js