Lesson 2 Exercises

L2- Set 1

In this set of exercises you will be demonstrating your ability to install dependencies for a Node.js project. Good luck!

1) Double check node is installed by checking the node version in the terminal.

```
node --version
```

2) Install the `cors` package from the terminal.

```
npm install cors
```

3) Install the `body-parser` package from the terminal

```
npm install body-parser
```

L2- Set 2

In this set of exercises you will focus on setting up project dependencies in a file for server side code. You can do it!

1) Install the 'express' package from the terminal. Include the 'express' package in your project and create an instance of an express app in the file named `starter.js` .

```
// Express to run server and routes
const express = require('express');

// Start up an instance of app
const app = express();
```

2) Include the 'cors' package in your project and connect it to the express app instance.

Once there is an instance of the express app saved to variable, the `use()` method is useful for connecting project dependencies:

```
const cors = require('cors');
app.use(cors());
```

L2- Set 3

In this set of exercises you will demonstrate creating and starting a server with Node and Express. Your serve!

1) Create a server with a callback function. Inside the callback function, log the port number so you can see it is working.

2) Test your server by running the `starter.js` file from the terminal. *Make sure you have saved your file before running the terminal command.*

Note that you can open a second terminal in the toolbar with `File>New>Terminal` . You can use the `curl` command with `localhost:{your port}` to see if your server returns what you expect.

```
const server = app.listen(port, listening);
function listening(){
    // console.log(server);
    console.log(`running on localhost: ${port}`);
};
```

```
node starter.js
```

Lesson 3 Requests & Routes

This exercise focuses on GET routes. You GOT it!

L3- Set 1

1) In the file `server.js` , create a GET route that uses the url `/all` and returns the JavaScript object named `projectData` .

This example uses a callback function:

```
app.get('/all', sendData);


function sendData (request, response) {
  response.send(projectData);
};
```

You could also just pass a function in as the second parameter to `.get()` like this:

```
app.get('/all', function (request, response) {
  response.send(projectData);
});
```

Or even with an arrow function like this:

```
app.get('/all', (request, response)=> {
  response.send(projectData);
});
```

L3- Set 2

In this set of exercises you will tangle with POST routes and your favorite animal.

1) In the file, `server.js`, create a POST route that uses the url `/add` and sends the response `POST received` when used to make a request.

The parameters used here, `req` and `res`, are shortened versions of and equivalent to `request` and `response`, which are also often used for naming the parameters of route methods.

```
app.post('/add', callBack);

function callBack(req,res){
  res.send('POST received')
}
```

2) Add a POST route for adding a favorite animal via the path `'/animal'` to an array named `data`. You will need to create the array as well.

```
const data = [];

app.post('/animal', addAnimal);

function addAnimal (req,res){
    data.push(req.body);
};
```

You're getting there! This stuff can be hard to pick up at first, but your brain is starting to untangle this, I promise!

L3- Set 3

In this exercise you will start to understand how a route setup on the server side, is used on the client side with a request.

> This is meant to be a starting place for you to see how things work a bit, so don't worry too much about code you don't understand yet-- we're building up to it...

1) In the file named `app.js`, which is located in the `website` directory of this project, call the function `postData` with the url `/addAnimal` and the name of your favorite animal to create a POST request that uses the POST route you setup in `server.js`.

```
postData('/animal', {animal:'lion'})
```

See?! That wasn't so bad, right?

Lesson 4 Asynchronous JavaScript

In this exercise you will essentially write the function that was provided for you in the last exercise of Lesson 3, so it's there somewhere in your brain...you got this!

L4- Set 1

1) In the file `getPost.js` (located in the `website` directory of this project), write an `async` function to make a POST request that has two arguments: a url to make the POST to, and a JavaScript object holding the data to post.

```
const postData = async ( url = '', data = {})=>{

    const response = await fetch(url, {
    method: 'POST',
    credentials: 'same-origin',
    headers: {
        'Content-Type': 'application/json',
    },
    body: JSON.stringify(data), // body data type must match "Content-Typ
e" header
  });

    try {
      const newData = await response.json();
      return newData
    }catch(error) {
    console.log("error", error);
    }
}
```

L4- Set 2

In this exercise you will GET async with it...

1) In the file `getPost.js`, write an `async` function to make a GET request that has one argument: a url to make the GET request to.

Hint - Fetch is your friend.

```
const retrieveData = async (url='') =>{
  const request = await fetch(url);
  try {
  // Transform into JSON
  const allData = await request.json()
  }
  catch(error) {
    console.log("error", error);
    // appropriately handle the error
  }
}
```

L4 Set 3

In this exercise, you will bring it all together by chaining async requests.

1) In the file `getPost.js`, write a function that chains together the two `async` functions you have previously written, so that you make a POST request to the `/animal` route, and then retrieve the data with a GET request to the `/all` path.

You should pass in a data object of your favorite animal as the second argument for the POST request.

Call your function as the last line of code in the file named `getPost.js`.

```
function postGet(){
  postData('/animal', {fav:'lion'})
    .then(function(data){
      retrieveData('/all')
    })
}

postGet()
```