Writing Your First Route

9 min

Once the Express server is listening, it can respond to any and all

Preview: Docs Loading link description

requests

. But how does it know what to do with these requests? To tell our server how to deal with any given request, we register a series of *routes*. Routes define the control flow for requests based on the request's *path* and HTTP verb.

For example, if your server receives a GET request at /monsters, we will use a route to define the appropriate functionality for that HTTP verb (GET) and path (/monsters).

The path is the part of a request URL after the hostname and port number, so in a request to localhost:4001/monsters, the path is /monsters (in this example, the hostname is localhost, the port number is 4001).

The HTTP verb is always included in the request, and it is one of a finite number of options used to specify expected functionality. GET requests are used for retrieving resources from a server, and we will discuss additional request types in later exercises.

Express uses app.get() to register routes to match GET requests. Express routes (including app.get()) usually take two arguments, a path (usually a string), and a callback function to handle the request and send a response.

```
const moods = [{ mood: 'excited about express!'}, { mood: 'route-tastic!' }];
app.get('/moods', (req, res, next) => {
  // Here we would send back the moods array in response
});
```

The route above will match any GET request to '/moods' and call the callback function, passing in two

Preview: Docs Loading link description

objects

as the first two arguments. These objects represent the request sent to the server and the response that the Express server should eventually send to the client.

If no routes are matched on a client request, the Express server will handle sending a 404 Not Found response to the client.

Instructions

1. Checkpoint 1 Enabled

1.

Now that your server starting code should be working properly, you can start up the Express Yourself machine. Start your server from the terminal window with node app.js. Once it logs that it is running, you can refresh the browser window currently displaying Not Found.

Inside **app.js**, create a route handler to handle a GET request to '/expressions'. For now, give it a req, res, next callback. For now, log the req object inside the callback. Verify that the route works and logs the request by starting your server and clicking the Refresh Expressions button which will send a GET /expressions request.

We will complete this route in the next exercise and finish the first round of functionality to the Express Yourself machine.

You may notice that there's a line with the command app.use(express.static('public'));. This is used to make sure that once the server is started, you can reload the browser and see the Express Yourself machine.

app.js

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

const app = express();


const PORT = process.env.PORT || 4001;

// Use static server to serve the Express Yourself Website

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


// Open a call to `app.get()` below:

app.get('/expressions', (req, res, next) => {

  console.log(req)

})


app.listen(PORT, () => {

  console.log(`Listening on port ${PORT}`);

});
```