Getting started with Hapi JS

https://itnext.io/getting-started-with-hapi-js-e841724da924

Part 1: starting the server and basic routing

I know it seems like the only framework for Node is Express, but I think it's worth it for new programmers to give hapi a shot. Hapi is extremely light weight and built around a plugin system that keeps everything modular. Plus, its docs are great and they have a nice little tutorials section. In this series we'll go over the basics, starting with getting your server going and adding some routes.

Here is the GitHub to see the finished project

Hapi 16 vs 17

A *quick* side note about which version you should use. There was a shift in the framework from 16 to 17, where it went from using callbacks to async/await. This tutorial, and most of the professional world, will use hapi 17 and up. If you're working

on older code that has to be updated, I highly recommend using this hapi conversion article as a guide. That's a great site just to poke around as well, I've loved going through their tutorials.

Getting started: installation

Run `npm install hapi` and you'll be good to go. You should also install nodemon globally. That will take care of restarting the server on file changes. Or, you can just clone my GitHub, which uses the magic of npx to run a locally installed nodemon.

Up and running with the server.js file

Here's what our `server.js` file looks like so far:

```
const Hapi = require('hapi');
const server = new Hapi.Server({
    host: 'localhost',
    port: 3101,
});
const launch = async () => {
    try {
        await server.start();
    } catch (err) {
        console.error(err);
        process.exit(1);
```

```
    };
    console.log(`Server running at $
{server.info.uri}`);
}
launch();
```

If you run `npm start` and go to http://localhost:3101/, you'll see that our server *is* up and running, you're just getting a 404 because we haven't made a route handler yet. But before we do, let's talk about the two main parts here:

The Server method and its config object

As you can see, we create our server by calling hapi's `Server method`. It takes a config object and returns a `server` object that we can work with. So much stuff *can* go into the config object, but for now all you need is a host and port, a string and an integer, respectively. Also, `'localhost'` is fine if you're just puttering around, but save yourself the aggravation and set your host to `'0.0.0.0'` if you're using Docker.

The "launch" method

Hapi has a built-in async method on the server called `start()`, and you'd think that's all you need. But soon, we'll do a lot of things to our server before it actually starts, and it'll be helpful to keep

them all in one place. That's why it's common to create a function of your own. I like to name mine 'launch,' but you can call it whatever. It's also going to handle any errors by printing them to the console and exiting out of the node process. After we define the function, we call it at the end of our file, which finally starts our server. Now, let's add some routes!

Adding basic Routes

You add routes by using the server object's `route()` method. The method takes either a config object or an array of objects. Here's what the most basic route looks like:

```
server.route({
    method: 'GET',
    path: '/',
    handler: (request, h) => {
        return 'I am the home route'
    }
});
```

Now if you go to http://localhost:3101/, there's a string printed out to the screen instead of a 404. Progress! As for the config object, we'll give it the minimum three properties for now: `method` , `path` , and `handler` .

method

This is the http verb that will trigger the route, it

can be a string or an array of strings (way more common to see a string). Capitalization doesn't matter, but I personally think all caps looks neater.

path

This is the actual route string path that you would put into the browser, and where you would specify your parameters.

handler function

This is a function that determines what gets sent back to the client. It has access to both the initial `request` object, as well as something called the `h response toolkit`. They're great for more complex responses, but a simple `return` is all that's needed to send back .json or strings.

Adding routes to your server in context

All put together, this is what your final `server.js` file looks like this:

```
const Hapi = require('hapi');
// create a server with a host and port
const server = new Hapi.Server({
    host: 'localhost',
    port: 3101
});
// add each route
server.route([
```

```
    {
        method: 'GET',
        path: '/',
        handler: (request, h) => {
            return 'I am the home route';
        },
    },
    {
        method: 'GET',
        path: '/example',
        handler: (request, h) => {
            return { msg: 'I am json' };
        },
    }
]);
// define server start function
const launch = async () => {
    try {
        await server.start();
    } catch (err) {
        console.error(err);
        process.exit(1);
    };
    console.log(`Server running at $
{server.info.uri}`);
}
// start your server
launch();
```

Congratulations! You've built a tiny server with hapi! We'll go over path parameters and how to use the response toolkit in the next installment, but I encourage you to check out the tutorials and docs in the mean time.