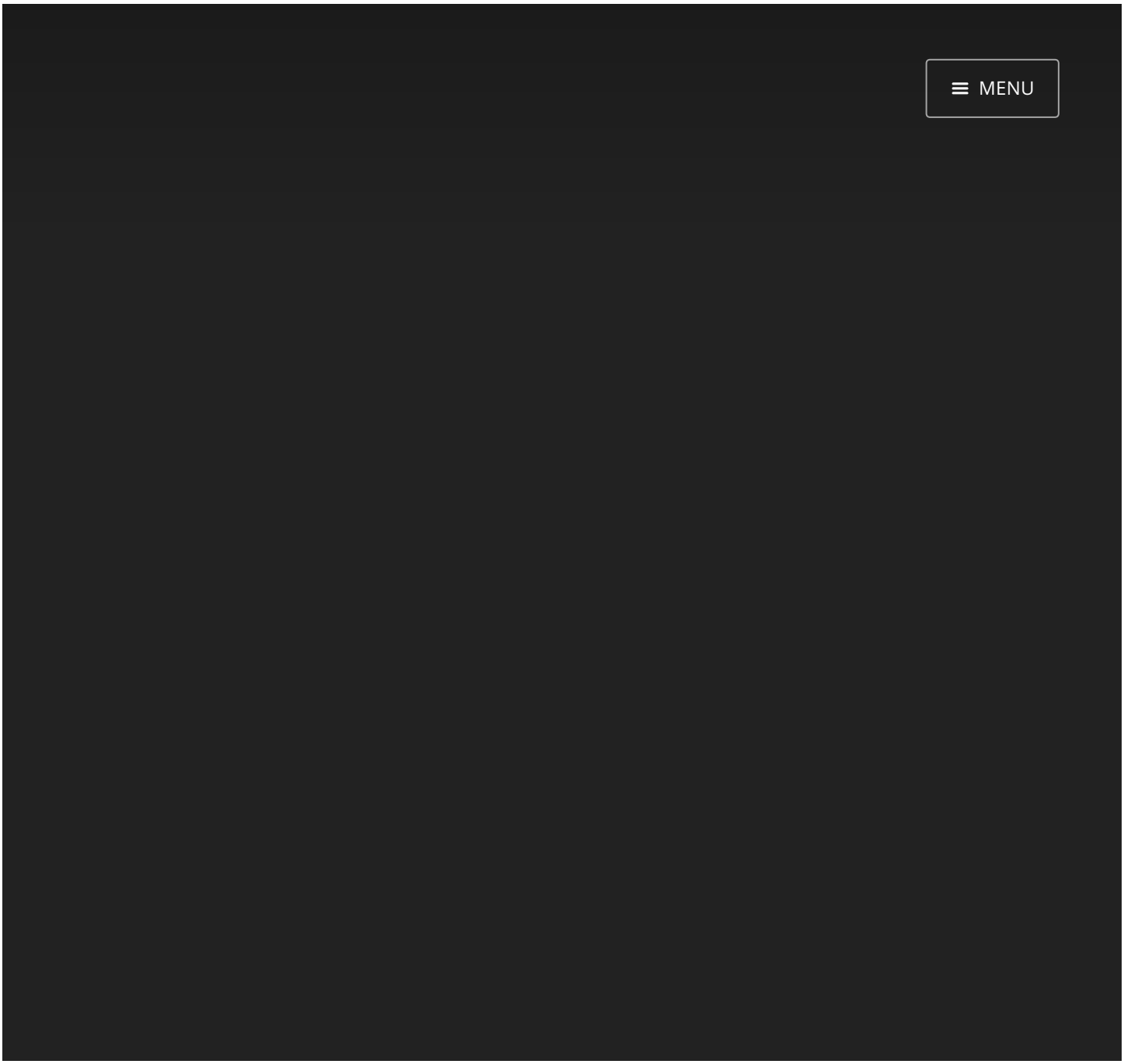

Why I like Hapi more than Express

21 OCTOBER 2015

When I first started with Node.js, my first HTTP servers were built using Express. Almost all the tutorials use Express as their server, which I think helps its dominance.

But Hapi beats it hands down. Hapi, out of the box has:

1. Easier setup with better defaults
2. Better error handling
3. More powerful reply interface
4. Better testability

And that's just scratching the surface.

Let's dive in to some code (GitHub). All of my work involves JSON API's so that will be my focus.

> *Note: I'm using ES6 and HTTPie because it's my blog and I can do what I want.*

Hapi

```
const Hapi = require('hapi')

const port = process.env.PORT || 8080
const server = new Hapi.Server()

server.connection({ port: port })

server.route({
    method: 'POST',
    path: '/hello',
    handler: (req, reply) => {
        reply({ hello: req.payload.name })
    }
})

server.start( (err) => {
```

```
    console.log('Server running at:', server.info.uri,
err ? err : '');
})
```

Express

```
const express = require('express')
const bodyParser = require('body-parser')

const port = process.env.PORT || 8080
const app = express()

app.use(bodyParser.urlencoded({ extended: true }))
app.use(bodyParser.json())

const router = express.Router()

router.post('/hello', (req, res) => {
    res.json({ hello: req.body.name })
})

app.use(router)

app.listen(port)
```

Easier Setup

These are pretty similar. But already I see things I don't like in Express. It requires an additional dependency to parse out my JSON body, which is annoying. What else would I be POSTing? A form? That also requires the `body-parser` dependency!

Hapi, on the other hand, parses both out of the box just fine.

Express's middleware mentality has caused me to have over ten middleware plugins for my app (example). I don't really want to deal with middleware for serving a favicon.

Better Error Handling

Let's start our Hapi Server:

```
$ npm start

> hapi-ethanmick@1.0.0 start
/Users/ethan/Documents/ethanmick/hapi-vs-express/hapi
> node --es_staging index.js

Server running at: http://Ethan-Micks-MacBook-
Pro.local:8080
```

Awesome.

Let's start another one! (In a new shell):

```
$ npm start

> hapi-ethanmick@1.0.0 start
/Users/ethan/Documents/ethanmick/hapi-vs-express/hapi
> node --es_staging index.js

Server running at: http://Ethan-Micks-MacBook-
Pro.local:8080 { [Error: listen EADDRINUSE 0.0.0.0:8080]
  code: 'EADDRINUSE',
  errno: 'EADDRINUSE',
  syscall: 'listen',
```

```
  address: '0.0.0.0',
  port: 8080 }
```

How kind! It uses the `err` object in the callback to inform me of an issue with starting the server.

Express?

```
$ npm start

> express-ethanmick@1.0.0 start
/Users/ethan/Documents/ethanmick/hapi-vs-express/express
> node --es_staging index.js

events.js:141
      throw er; // Unhandled 'error' event
      ^

Error: listen EADDRINUSE :::8080
    at Object.exports._errnoException (util.js:837:11)
    at exports._exceptionWithHostPort (util.js:860:20)
    at Server._listen2 (net.js:1231:14)
    at listen (net.js:1267:10)
    at Server.listen (net.js:1363:5)
    at EventEmitter.listen
(/Users/ethan/Documents/ethanmick/hapi-vs-
express/express/node_modules/express/lib/application.js:
617:24)
    at Object.<anonymous>
(/Users/ethan/Documents/ethanmick/hapi-vs-
express/express/index.js:23:5)
    at Module._compile (module.js:434:26)
```

```
    at Object.Module._extensions..js (module.js:452:10)
    at Module.load (module.js:355:32)
```

No such kindness. It throws an exception. Express's `listen` method is just a simple passthrough to the HTTP's listen method, which doesn't pass in an error in the callback. To fix this, we need to change the code to:

```
app.listen(port, () => {
    console.log('Server running at:', port)
}).on('error', err => { console.log('Err!', err) } )
```

Alright, fine, slightly different syntax. Whatever. Let's fire off 404 request to Express:

```
$ http 'http://localhost:8080/random'
HTTP/1.1 404 Not Found
Connection: keep-alive
Content-Length: 19
Content-Type: text/html; charset=utf-8
Date: Tue, 20 Oct 2015 20:35:41 GMT
X-Content-Type-Options: nosniff
X-Powered-By: Express

Cannot GET /random
```

Woh! What!? I want a *JSON API* not a "return anything you want" API! Express, by default, returns a lovely `text/html` response on a 404. **Ugh**.

Hapi?

```
$ http 'http://localhost:8080/random'
HTTP/1.1 404 Not Found
Connection: keep-alive
Date: Tue, 20 Oct 2015 20:35:29 GMT
Transfer-Encoding: chunked
cache-control: no-cache
content-encoding: gzip
content-type: application/json; charset=utf-8
vary: accept-encoding

{
    "error": "Not Found",
    "statusCode": 404
}
```

Oh, thank you. (Hapi uses Boom for its HTTP errors, which is an excellent package).

Moving on, let's look at the startup options. In Express, you can set some options when you create the server:

```
const app = express({
    dotfiles: 'ignore',
    redirect: false
})
```

Same with Hapi. But what happens if I mistype some?

```
const app = express({
    dotfile: 'ignore',
    redirect: 'ok'
})
```

```
$ npm start

> express-ethanmick@1.0.0 start
/Users/ethan/Documents/ethanmick/hapi-vs-express/express
> node --es_staging index.js

Server running at: 8080
```

Not a peep.

Let's try this in Hapi. Our server now looks like:

```
const server = new Hapi.Server({
    connections: {
        router: {
            isCaseSensitive: false
        }
    },
    debug: {
        log: ['error']
    }
})
```

Okay, let's make some honest mistakes:

```
const server = new Hapi.Server({
    connections: {
        routers: {
            isCaseSensitive: 'yes'
        }
```

```
    },
    debug: {
        log: 'error'
    }
})
```

```
$ npm start

> hapi-ethanmick@1.0.0 start
/Users/ethan/Documents/ethanmick/hapi-vs-express/hapi
> node --es_staging index.js

/Users/ethan/Documents/ethanmick/hapi-vs-
express/hapi/node_modules/hapi/node_modules/hoek/lib/ind
ex.js:731
    throw new Error(msgs.join(' ') || 'Unknown error');
    ^

Error: Invalid server options {
  "connections": {
    "routers" [1]: {
      "isCaseSensitive": "yes"
    }
  },
  "debug": {
    "log": "error"
  }
}

[1] "routers" is not allowed
```

Wow! You got really angry. But I love it! It even shows you *where* the issue was, marked by the [1]. As you fix each issue in your configuration, the error message catches each one:

> *[1] "log" must be an array*
> *[1] "isCaseSensitive" must be a boolean*

This stops you from easily passing in an Object with many extraneous keys (such as a large generic configuration object), but it will **always, always** catch you mistyping a key, or setting an invalid value.

Again, behind the scenes, they are using their own powerful Joi module, which you should definitely take a look at.

Replying

Express favors a response object that has many methods on it, such as `.json` or `.sendFile`. Hapi does two things here. One, their reply interface is first and foremost a function.

```
reply({hello: 'Everyone!'})
```

Objects and Arrays are returned as JSON, strings are returned as plain text. Just what I would expect. But Hapi goes further, and you can return a whole bunch of clever things:

1. null
2. undefined
3. string
4. number
5. boolean

6. Buffer object
7. Error object (Formatted to a 500 unless it's a Boom error)
8. Stream object
9. Promise object
10. any other object or array

Nice! You can return a Stream and it'll stream the data back to the user. You can also return a Promise, and when fulfilled, it will send the result to the user. It makes for some beautiful and elegant code.

The Reply interface can be enhanced with some Plugins (This is annoying, they used to be built into Hapi), and you can serve files and directories, or Proxy requests.

Proxying is especially powerful. You can redefine the entire URI and Headers in your proxy method, which allows you do awesome things, such as:

1. Hide S3 resources behind your own authentication
2. Add OAuth Authentication information to requests and pass them to upstream services

Testing

Lastly, I want to touch on testing. Tests are extremely important, and one thing I've found very hard is to test the *controller* part of my code. I have all my logic separated into a model layer, but I still want to ensure my controllers have code coverage and work as expected.

How can I do that in Express? Maybe this? Or this? I've used Supertest in the past, but it's clunky and inelegant.

Hapi to the rescue. Change the end of our file to:

```
//server.start( (err) => {
//    console.log('Server running at:', server.info.uri,
err ? err : '');
//})

var req = {
    method: 'POST',
    url: '/hello',
    payload: JSON.stringify({name: 'Ethan'})
};

server.inject(req, res => {
    console.log('Tested!', res.statusCode, res.result);
});
```

Run it and get: `Tested! 200 { hello: 'Ethan' }`

Hapi has a built in `inject` method that allows you to, well, inject an HTTP request into the server. It gets routed like a normal request to the correct route handler, which then executes as normal. The result object in our test has the information that we can validate in a test. You can see how I fully test a server in my example hapi-starter.

Last Thoughts

There are some other things which I really like about the Hapi Project. I like how aggressively they move forward. They just

released version 11. They follow semantic versioning, but aren't *afraid* to release a major new version with breaking changes.

They supported IO.js from the very beginning and when Node 4 came out they released Hapi 10:

> *hapi v10.0.0 contains no breaking changes. It is published to indicate the transition from node v0.10 to node v4.0. Moving to v10.0.0 only requires upgrading your node version to the latest v4. Future releases of the hapi v10 branch will include internal changes to take advantage of the new features available in node v4 and those will break under node v0.10.*

I love the versioning, and their unabashed movement toward a better API. On the other side, you have Express, which released it's latest major version on on Apr 9, 2014.

So there you have it. I've been using Hapi for over 2 years now and always have been impressed with it. It's definitely worth a try for your next project!

Code
Discuss on Hacker News

Ethan Mick

Read more posts by this author.

Share this post

READ THIS NEXT

Decrypt all PDF's in a Directory

I recently had to decrypt a bunch of PDF's. I knew the password, but didn't want to do them...

YOU MIGHT ENJOY

A Revolution Down on the Farm

I just finished reading A Revolution Down on the Farm, by Paul K. Conkin. I wanted to read a...