

Build an API with Hapi.js



Hello! I'm Ryan

- Product Owner at Auth0
- Google Developer Expert
- Angularcasts.io

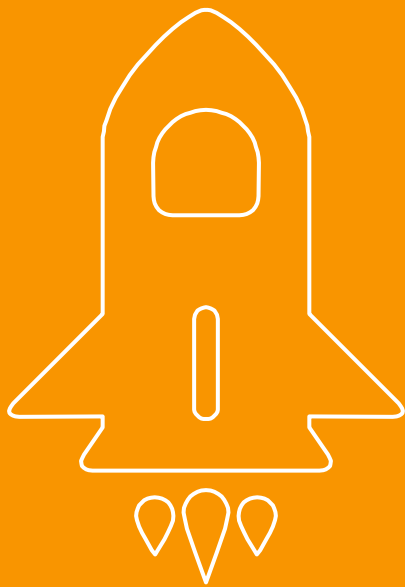


@ryanchenkie



chenkie





WHAT WE'LL BUILD

LET'S MAKE AN API

We've got an Angular app to manage FEM instructors

Now we need an API

- Get all instructors
- Get one instructor
- Add a new instructor

FemInstructorsApp

Ryan

localhost:4200/instructors/ryan-chenkie

FEM Instructors App


Add Instructor

Ryan Chenkie

Lukas Ruebbelke

Kyle Simpson

Brian Holt



Ryan Chenkie

ryanchenkie@gmail.com

chenkie

@ryanchenkie

Courses

- Build an API with Hapi.js
- Authentication for Single Page Apps

FemInstructorsApp

Ryan

localhost:4200/instructors/new

FEM Instructors App

Add Instructor

Ryan Chenkie

Lukas Ruebbelke

Kyle Simpson

Brian Holt

Name

Email

Github

Twitter

Courses

Add Course

Submit

Cancel

What we'll need

1. Node.js and npm
2. Postman
3. The Angular app

GET THE ANGULAR APP

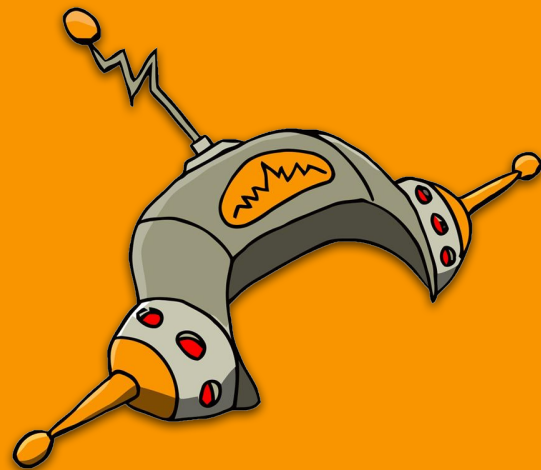
1. Clone the repo `bit.ly/fem-frontend`
2. Install the Angular CLI `npm install @angular/cli`
3. Install the dependencies `npm install`
4. Run the app `npm start`

GET THE HAPI API

1. Clone the repo `bit.ly/fem-backend`
2. Install the dependencies `npm install`
3. Switch between branches `git checkout branch-name`
4. Run the API `npm start`

1.

What is
Hapi.js All
About?



Hapi.js

“A rich framework for building applications and services”

Hapi.js History

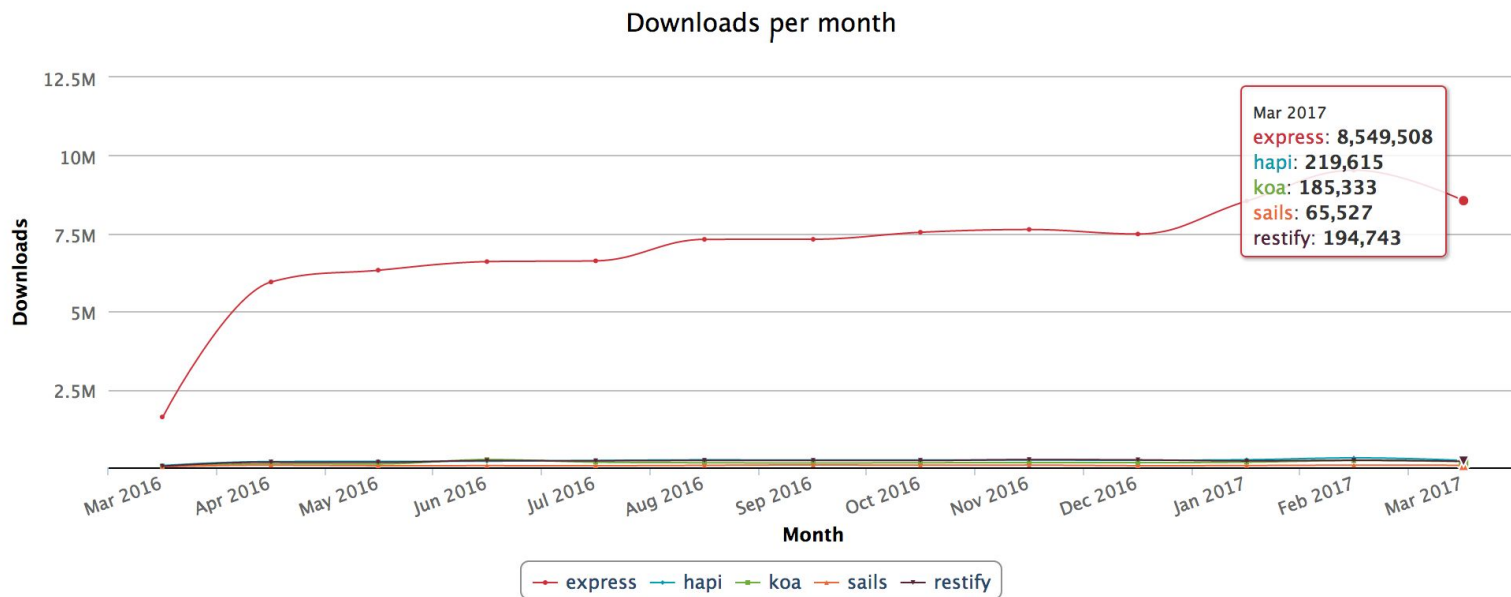
- Originally created by Walmart Labs
- Lead maintainer: Eran Hammer

What is Hapi.js used for?

- Data APIs
- Services
- Regular websites

HOW DOES HAPI STACK UP?

Not the most popular kid



WHY HAPI?

1. Batteries included

Error objects, validation, sessions, CORS, logging

2. Ecosystem

3. Simplicity & Organization

The Hapi.js Ecosystem



Error Objects



Validation



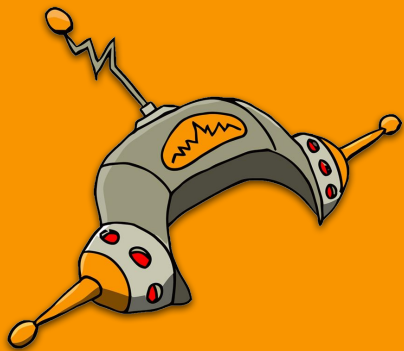
HTTP Client



Utilities



Process Monitoring



What does a Hapi.js app look like?

CREATE A SERVER INSTANCE

Wiring up a server is dead simple

```
const Hapi = require('hapi');  
const server = new Hapi.Server();
```

DEFINE A ROUTE

To make it do something, define a route

```
server.route({  
  method: 'GET',  
  path: '/api/stuff',  
  handler: (request, reply) => { ... }  
});
```

START THE SERVER

Tell Hapi which port to listen on and start the server

```
server.connection({ port: 3001 });  
server.start(err => {  
  if (err) throw err;  
});
```

THE REPLY INTERFACE

Send a response to the client using `reply`

...

```
handler: (request, reply) => {  
    reply('Hey there');  
}
```

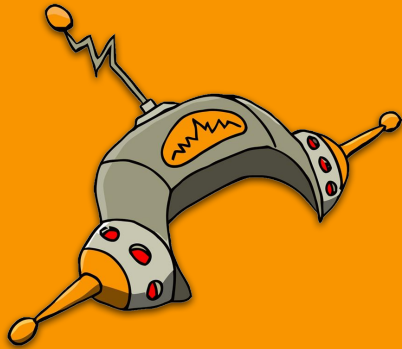
Challenge #1

WIRE UP A SIMPLE APP

- `git checkout module-1-start`
- Create a Hapi.js server that listens on port 3001
- Create a single route which responds to a `GET` request
- When the server starts, use the `server.info` object to log the URI out to the console

2.

Routes and Route Handlers



When it comes down to it, the web
is all about request and response

REST API Principles Review

A FEW CORE PRINCIPLES

- REST describes how to make resources available in a client-server relationship
- Data should be organized around resources
- API should respond to common HTTP verbs
`GET, POST, PUT, PATCH, DELETE`
- Server should be stateless

REST API Principles Review

OPERATIONS ON A SINGLE RESOURCE

Given some `contacts` resource, we might want multiple endpoints to have full CRUD operations

- Get all contacts `GET /contacts`
- Get one contact `GET /contacts/42`
- Create a contact `POST /contacts`
- Edit a contact `PUT /contacts/42`
- Delete a contact `DELETE /contacts/42`

ROUTE CONFIGURATION

Routes are configured with an options object

At a minimum, a route needs three things:

`method, path, handler`

```
server.route({  
  method: 'GET',  
  path: '/api/stuff',  
  handler: (request, reply) => { ... }  
});
```

THE ROUTE HANDLER

The route handler is where the magic happens

When we send a request to an endpoint, we want some action to take place

Example: save to a database, calculate a value, fetch an external resource

THE ROUTE HANDLER

For the route handler to do its job, it uses a `request` object and a `reply` interface

The Request Object

For each incoming request, a `request` object is created

The `request` object has useful information about the request

- Query string
- Params string
- Payload
- Headers
- Auth
- Route prerequisite

DATA ON THE REQUEST OBJECT

GET /api/contacts/42?profile=true

```
path: '/api/contacts/{id}'
```

```
handler: (request, reply) => {
```

```
  const id = request.params.id; // 42
```

```
  const profile = request.query.profile // true
```

```
}
```

The Reply Interface

A function argument to respond to requests

Used as a callback interface and a response generator depending on where it is used

- When used as a callback, returns control to Hapi
- When used as a response generator, sends a response to the client

THE REPLY INTERFACE

Send a response to the client using `reply`

...

```
handler: (request, reply) => {  
    reply('Hey there');  
}
```

Challenge #2

RETURN SOME DATA

- git checkout module-2-start
- When a GET request is made to `/api/instructors`, return all the instructors data
- When a GET request is made to `/api/instructors/{slug}`, return the specific instructor
- For the single instructor route, send back only the `id`, `name`, and `slug`

Challenge #2

NOTES

- We're storing the data in-memory
- We can use simple JavaScript functions like `map` and `find` to fake out database queries
- Make sure to check whether the resource exists before sending anything back

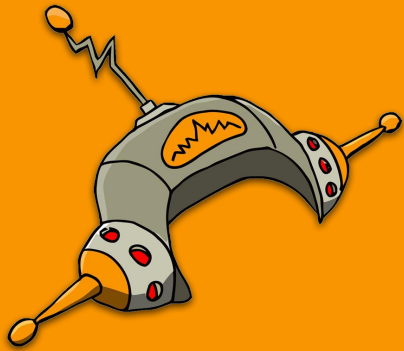
Challenge #2

BONUS

- Provide a way to sort the data based on a `sortKey` and `sortDirection`
- Use the query string to specify these options
- Hint: use Lodash to do the sorting

3.

Routes
Prerequisites

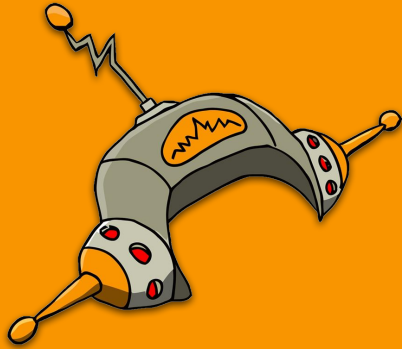


APIs don't operate in a vacuum

COMMON API TASKS

- Pull data out of a database and save new data to it
- Call other APIs and services
- Run complex, resource intensive tasks (calculations)

Regardless of the task, it needs to complete before a reply can be sent to the client



Typical Approach:
Everything in the Handler

ROUTE PREREQUISITES

A better way: specify some action to happen before the route handler is reached

- Prerequisite functions are run before anything else
- Values can be picked up in the handler
- Async operations complete before the handler is reached

ROUTE PREREQUISITES

What does a route prerequisite look like?

- Array of objects on `config.pre`
- At a minimum, specify a `method`

Route Prerequisites

Methods need to have the same `request` and `reply` signature found in handlers

Same access to `request` information

Calling `reply` passes control back to the framework

Assigning a value will put it on `request.pre` in the handler (optional)

Route Prerequisites

By default, methods will be called in succession

Putting methods in another array will run them in parallel

BENEFITS

- Small, reusable units
- Better organization
- Don't need to worry about handling async

Challenge #3

POST SOME DATA

- git checkout module-3-start
- When a `POST` request is made to `/api/instructors`, add a new instructor to the array
- Create a route prerequisite which checks whether the instructor already exists
- Create a route prerequisite which creates a `slug` for the instructor and assigns it to a key for use in the handler

Challenge #3

NOTES

- Github API requires a user agent
- { "User-Agent" : "whatever" }

Challenge #3

BONUS

- Use the Wreck library to make a call to the Github API to get the instructor's avatar URL
 - Demonstrates that route prerequisites work asynchronously
- This should be done in the `GET /instructors/{slug}` route

Challenge #3

NOTES

- Instructor ID can be incremented based on the length of the array
- When an instructor is added, reply with the newly created object

4.

Smart Error Objects with Boom

ERROR HANDLING

When something goes wrong in the request, we need to handle the error

ERROR HANDLING

What could go wrong?

- Resource doesn't exist
- User isn't authorized to access the resource
- Client is making too many requests

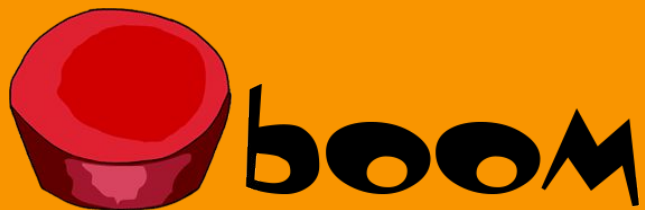
Handling Errors

We need to respond with an error code, usually something in the 400s

Provide a message to the user about what went wrong

Doing this manually is a pain

- How should the returned object be shaped?
- What if we want to return data or specify headers?
- What's that error code again?



Boom simplifies error responses

ERRORS WITH BOOM

With Boom, just provide a message

```
const Boom = require('boom');  
  
handler: (request, reply) => {  
  if (!contacts) {  
    reply(Boom.notFound('No contacts found!'));  
  }  
}
```

ERRORS WITH BOOM

Call any HTTP error reason

```
reply(Boom.<error_reason>(message));
```

Handling Errors with Boom

BENEFITS

- Error objects are easy to create
- Consistency
- Alignment with Hapi and ecosystem libraries which use Boom internally

Challenge #4

RETURN ERRORS WITH BOOM

- In the resources we've created, use Boom to return an error if something goes wrong
 - No instructor(s) found
 - Instructor already exists

5.

Validation with Joi

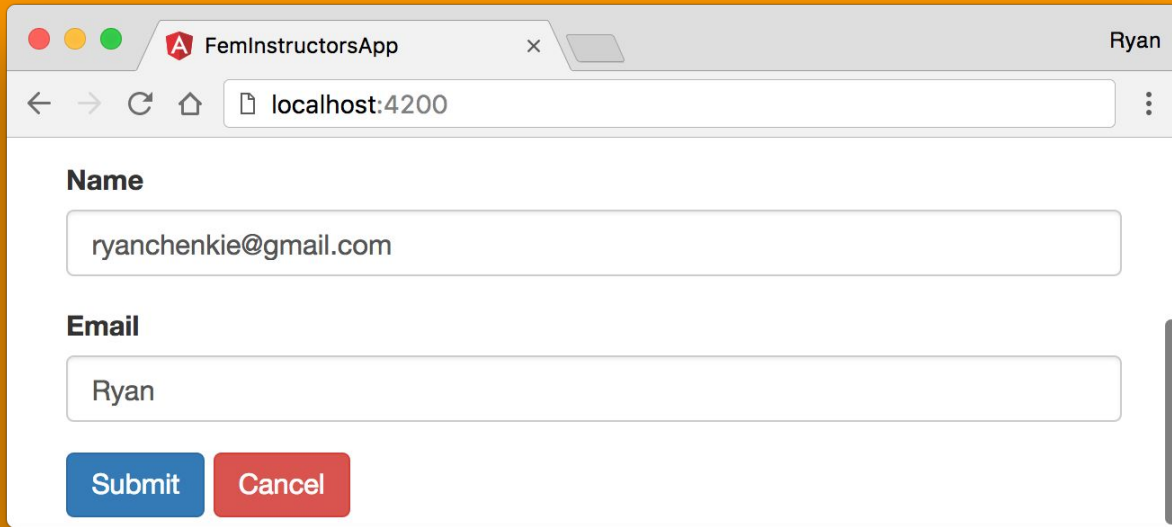
ROUTE VALIDATION

When it comes to payloads, queries, and parameters,
we shouldn't accept just anything

- Data validity
- Data consistency
- Security


```
<input type="text" placeholder="Name">
```

```
<input type="email" placeholder="Email">
```



A screenshot of a web browser window titled "FemInstructorsApp" with the URL "localhost:4200". The browser shows a form with two input fields. The first field, labeled "Name", contains the text "ryanchenkie@gmail.com". The second field, labeled "Email", contains the text "Ryan". Below the input fields are two buttons: a blue "Submit" button and a red "Cancel" button.

Validating Route Data

Validating incoming payloads, params, and queries by hand would be cumbersome

- What should the schema validation look like?
- How do we respond when something fails?
- Do we have all edge cases covered?

At the end of the day, we'd be writing a library to do this



Joi simplifies object schema validation

VALIDATION WITH JOI

With Joi, describe what the payload, params, or query should look like

```
const Joi = require('joi');  
const paramsValidator = Joi.object({  
  slug: Joi.string()  
});
```

VALIDATION WITH JOI

Be as specific or generic as you want

```
const payloadValidator = Joi.object({  
  name: Joi.string().required(),  
  skills: Joi.string().valid(['javascript', 'node'])  
});
```

VALIDATION WITH JOI

Use built-in utilities to ease validation

```
const payloadValidator = Joi.object({  
  email: Joi.string().email().required(),  
  title: Joi.string().regex(/[A-Za-z0-9]/)  
});
```

VALIDATION WITH JOI

How do we use it?

```
config: {  
  validate: {  
    payload: payloadValidator  
  }  
}
```

Object Schema Validation with Joi

What can be validated? Lot's of things.

- Strings
- Arrays
- Numbers
- Objects
- Date
- Boolean

Object Schema Validation with Joi

Common use cases

- Required
- Min/Max
- Email
- Credit Card
- Regex
- Valid (whitelisting)

VALIDATION WITH JOI

What happens when validation fails?

```
error: "Bad Request"
```

```
message: "child \"name\" fails because [\"name\" is not  
allowed to be empty]"
```

```
statusCode: 400
```

Challenge #5

ADD VALIDATION FOR OUR ROUTES

- git checkout module-5-start
- Query for GET /api/instructor
- Payload for POST /api/instructor
- Params for GET /api/instructor/{slug}

Challenge #5

NOTES

- We know what the only acceptable values for `sortDirection` and `sortKey` are
- For the `POST` route, `name` and `email` are required

Challenge #5

BONUS

- Twitter handles require an @ symbol and are limited to 15 characters - provide a regex to check for this

6.

Plugins

Plugin Ecosystem

Libraries like Boom and Joi are part of the “extended Hapi universe”

Many other (often smaller) libraries outside of the Hapi lineup are provided by the community

Plugins serve a number of needs

- Authentication
- Localization
- Documentation
- Templating
- Utility

USING PLUGINS

After installation, plugins need to be registered

```
const Plugin = require('plugin');  
server.register(Plugin, () => {});
```


USING PLUGINS

Most often, we'll want to register some options

```
server.register({  
  register: Plugin,  
  options: {  
    someOption: true  
  }  
}, err => ... );
```

Challenge #6

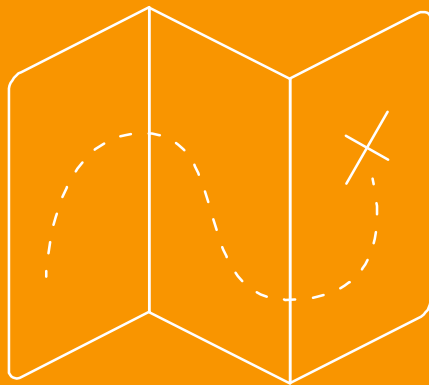
ADD SOME PLUGINS

- To get a feel for Hapi plugins, install Inert and hapi-geo-locate
- Inert is used for serving static files. Make the `GET /api/images/{name}` route respond with the image in the `public` directory
- hapi-geo-locate is used to get the user's IP address. Simply log out the IP address when a request is made for an image

Challenge #6

NOTES

- Check the readmes for these two plugins to learn about how to configure them



WRAPPING UP

Hapi.js Offers a Lot

1. Does a lot out-of-the-box
2. Rich ecosystem of libraries and plugins
3. Great approach to organization
4. Modularity
5. Enjoyable to build with

Where to go from here

1. Explore the Hapi ecosystem
2. Server methods
3. Authentication
4. Go and build



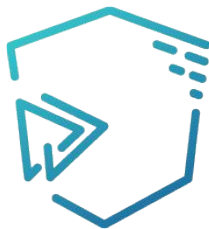
Drop me a line



@ryanchenkie



chenkie



angularcasts

THANK YOU!

