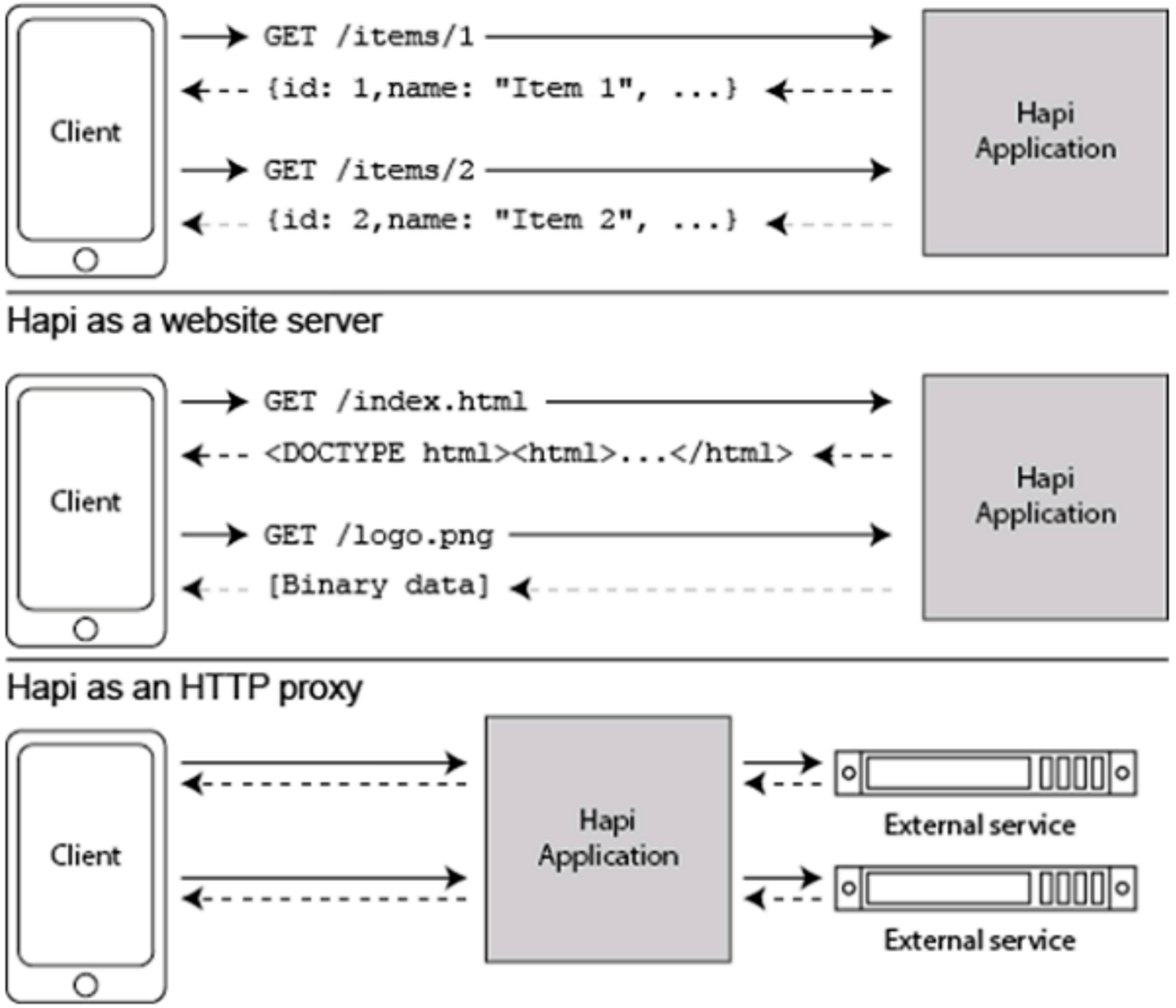
Views

Full Stack Web Development

Agenda

- Need for a tempting engine
- Handlebars
- Handlebars in Hapi

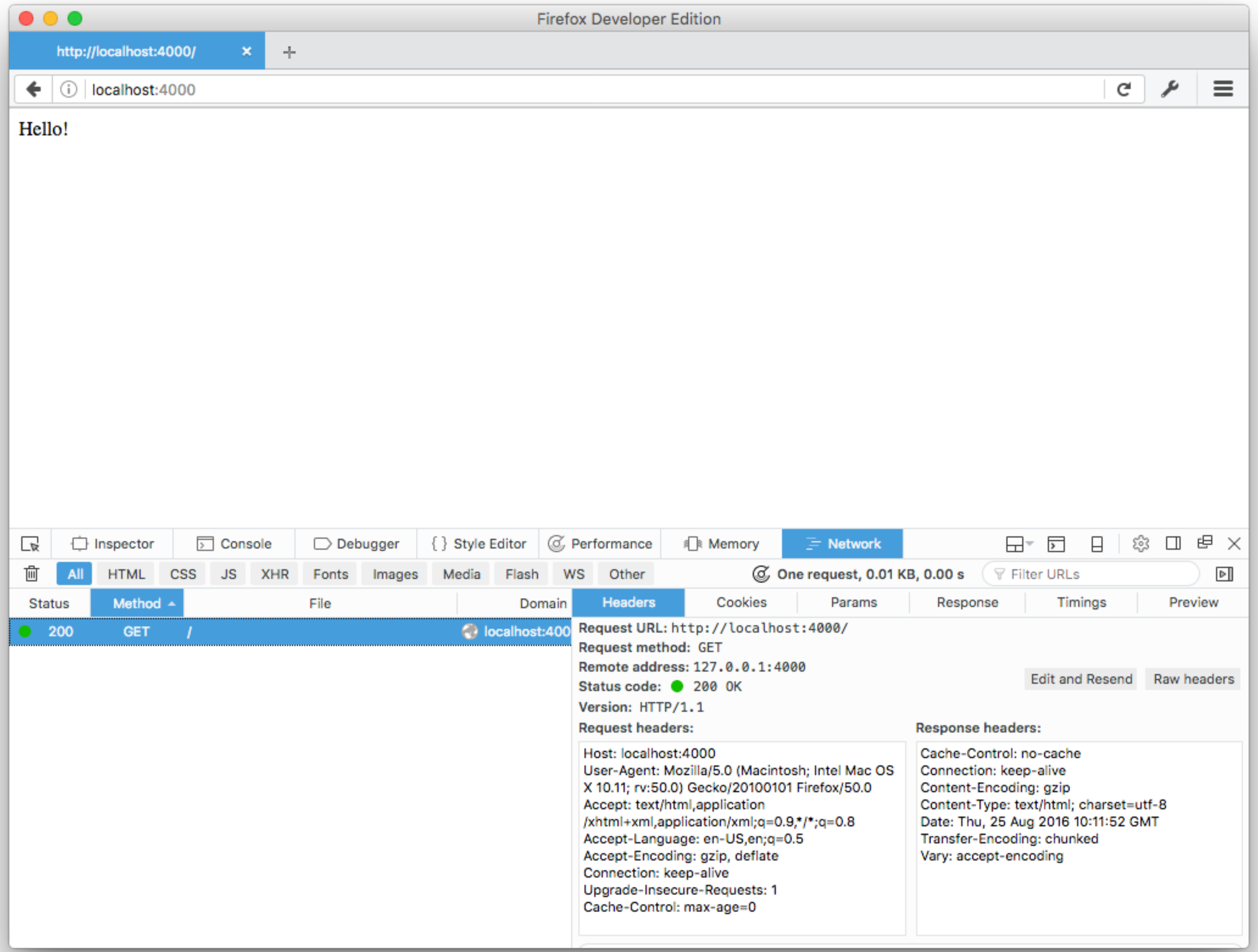
Hapi Application Types



return

- Responds to the browser with a simple string.

```
exports.index = {
  handler: function(request, h) {
    return 'Hello!';
  }
};
```



Hello!

One request, 0.01 KB, 0.00 s

```
200  GET  /  localhost:4000
```

```
Request URL: http://localhost:4000/
Request method: GET
Remote address: 127.0.0.1:4000
Status code: ● 200 OK
Version: HTTP/1.1
```

Request headers:

```
Host: localhost:4000
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS
X 10.11; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application
/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

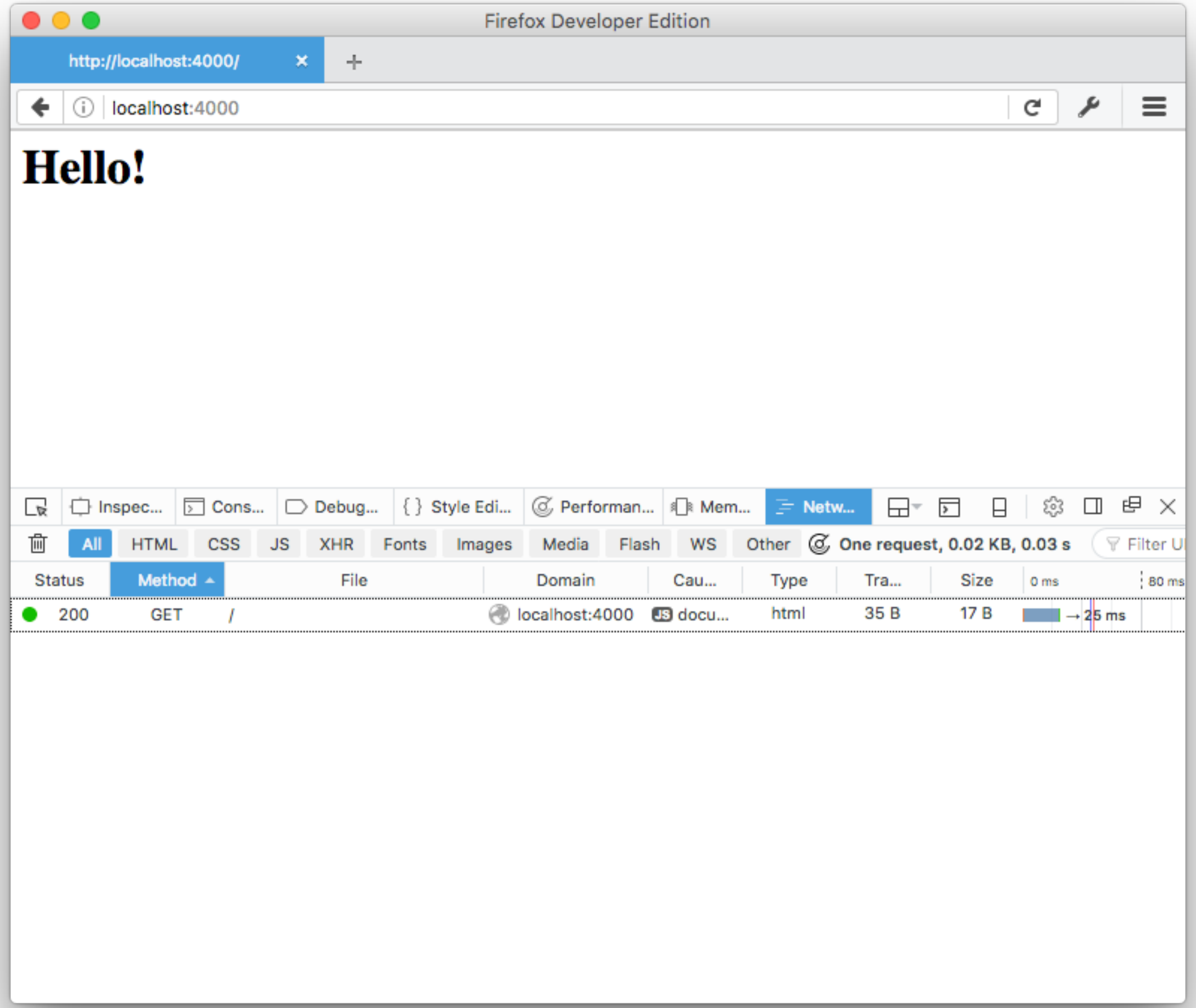
Response headers:

```
Cache-Control: no-cache
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Thu, 25 Aug 2016 10:11:52 GMT
Transfer-Encoding: chunked
Vary: accept-encoding
```

reply

- In order to render web pages we could pass html content
- This would become very unwieldy and unmaintainable

```
exports.index = {

  handler: function (request, h) {
    return('<h1> Hello! </h1>');
  }

};
```



Firefox Developer Edition

http://localhost:4000/

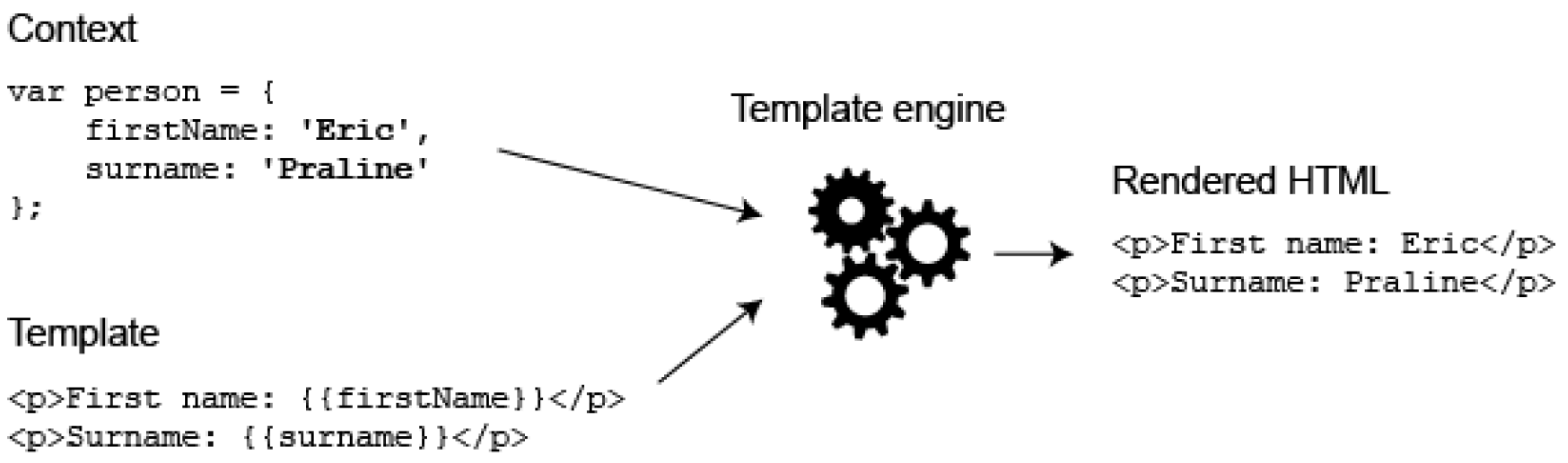
localhost:4000

Hello!

One request, 0.02 KB, 0.03 s

Status	Method	File	Domain	Cau...	Type	Tra...	Size	0 ms	80 ms
● 200	GET	/	localhost:4000	docu...	html	35 B	17 B	→ 25 ms	

Tempting Engine


Context

```
var person = {
    firstName: 'Eric',
    surname: 'Praline'
};
```

Template

```
<p>First name: {{firstName}}</p>
<p>Surname: {{surname}}</p>
```

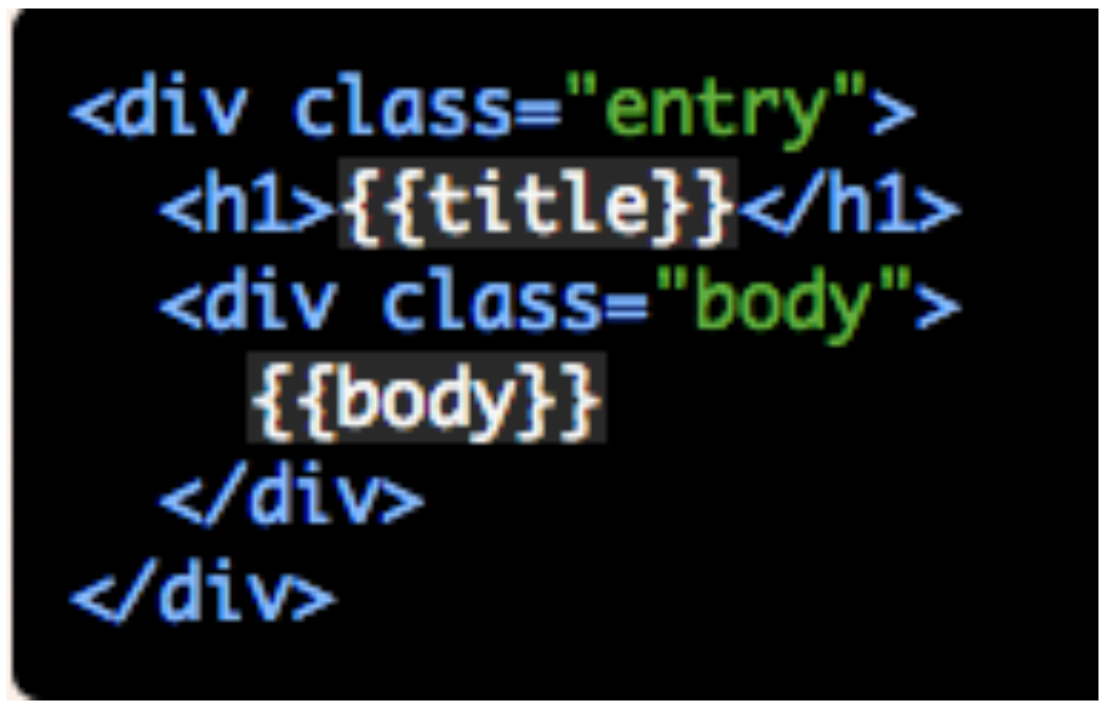
Rendered HTML

```
<p>First name: Eric</p>
<p>Surname: Praline</p>
```

Template Engines: Handlebars

"Handlebars provides the power necessary to let you build semantic templates effectively with no frustration.

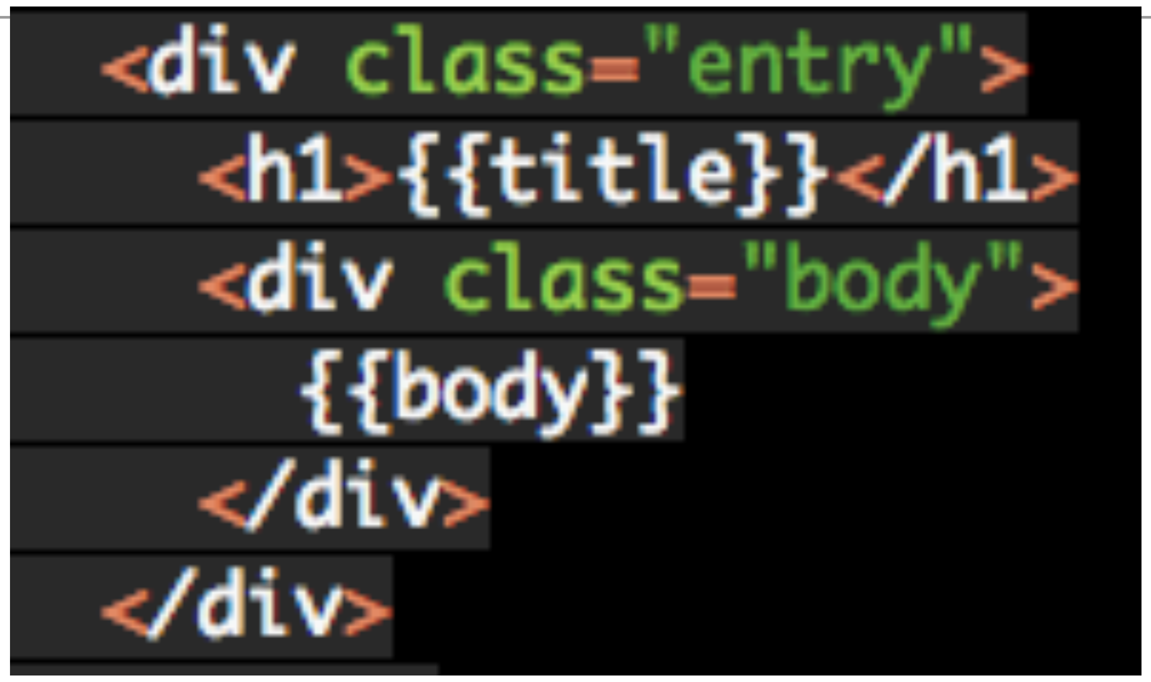
Handlebars is largely compatible with Mustache templates. In most cases it is possible to swap out Mustache with Handlebars and continue using your current templates.."



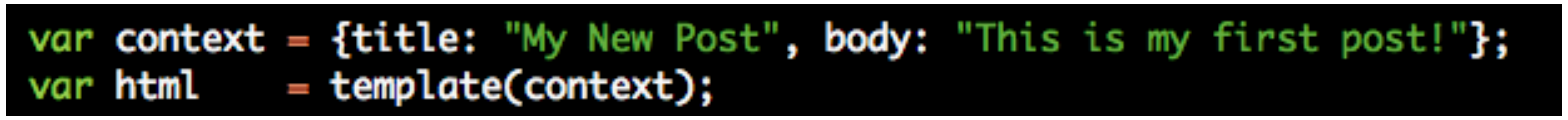
```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

Template Expressions

- A handlebars expression is a {{, some contents, followed by a }}



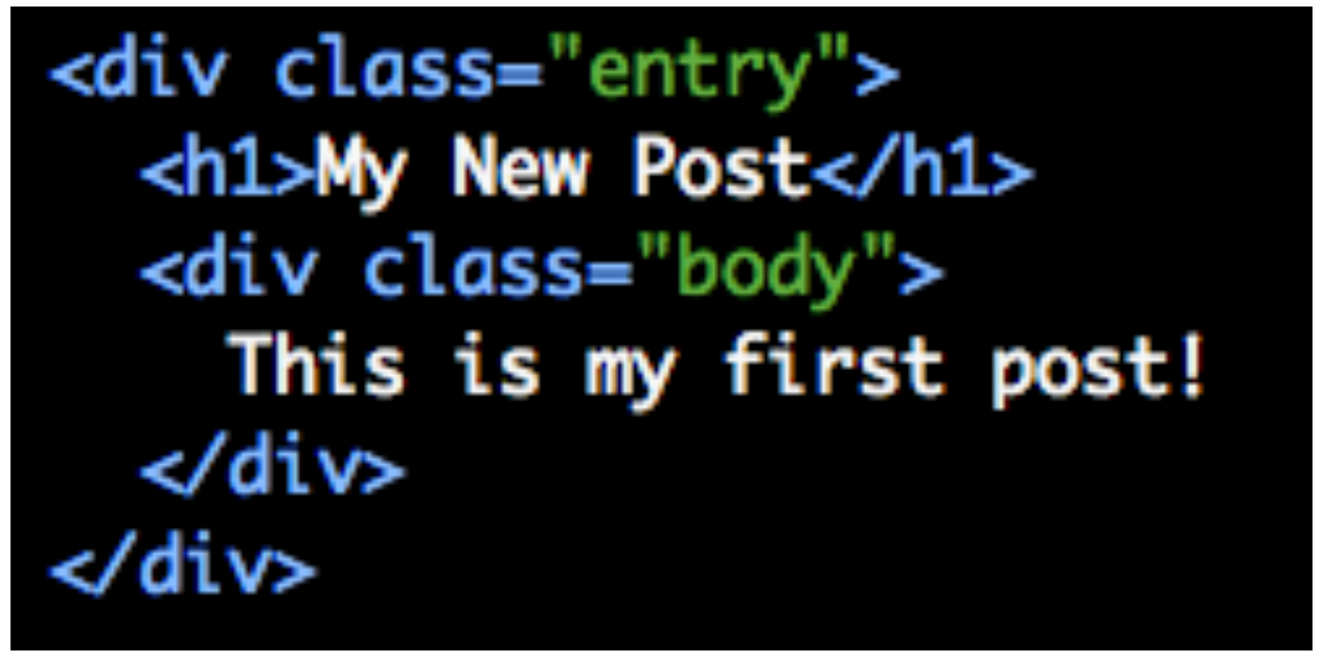
```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```



```
var context = {title: "My New Post", body: "This is my first post!"};
var html    = template(context);
```

- In Javascript, create an object literal with matching properties

- When rendered, the properties replace the handlebars expressions



```
<div class="entry">
  <h1>My New Post</h1>
  <div class="body">
    This is my first post!
  </div>
</div>
```

Handlebars Features

- Expressions
- Helpers
- Partials

must be mastered by developer

http://handlebarsjs.com/expressions
http://handlebarsjs.com/builtin_helpers
http://handlebarsjs.com/helpers

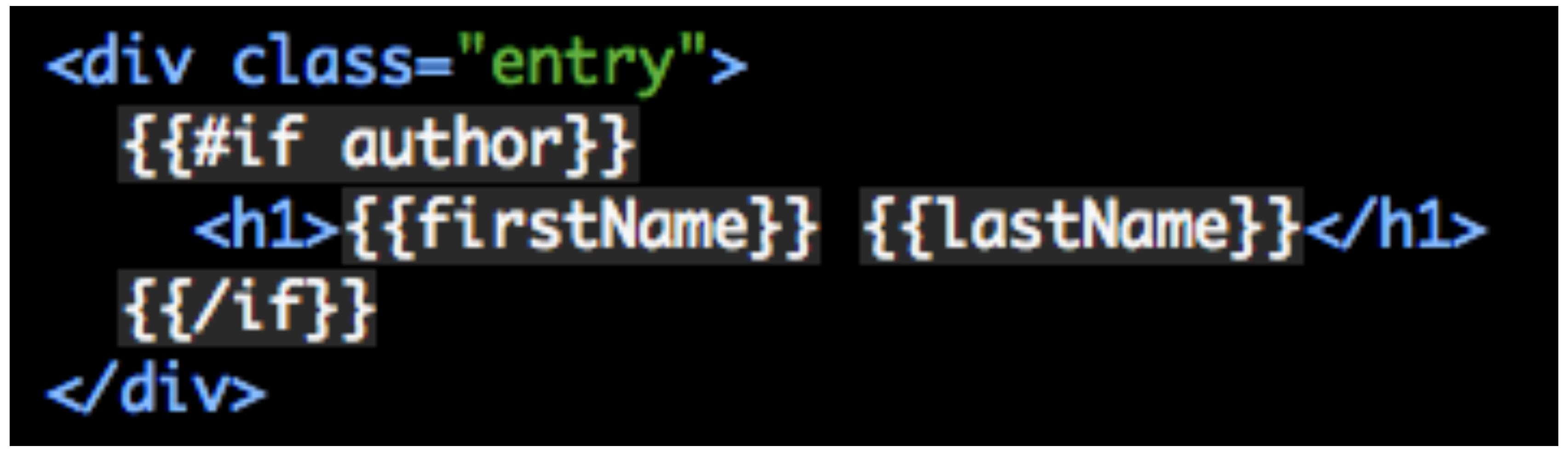
- Precompilation
- Execution

integrated into Hapi by ‘views’ plugin

Helpers

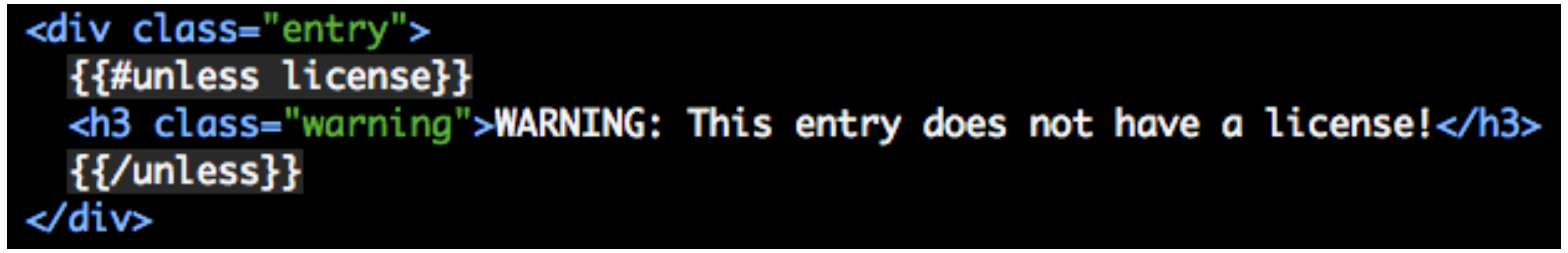
http://handlebarsjs.com/builtin_helpers

- Block expressions allow you to define helpers that will invoke a section of your template with a different context than the current.
- These block helpers are identified by a # preceeding the helper name and require a matching closing mustache, /, of the same name.

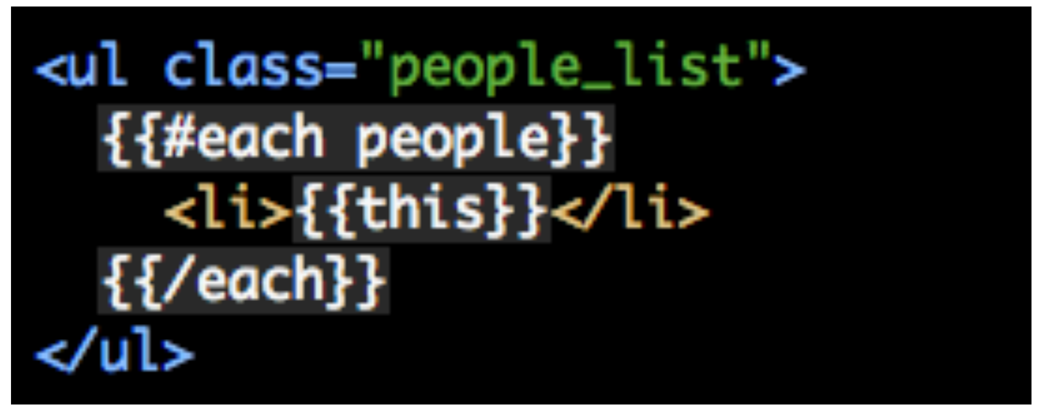


```
<div class="entry">
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{/if}}
</div>
```

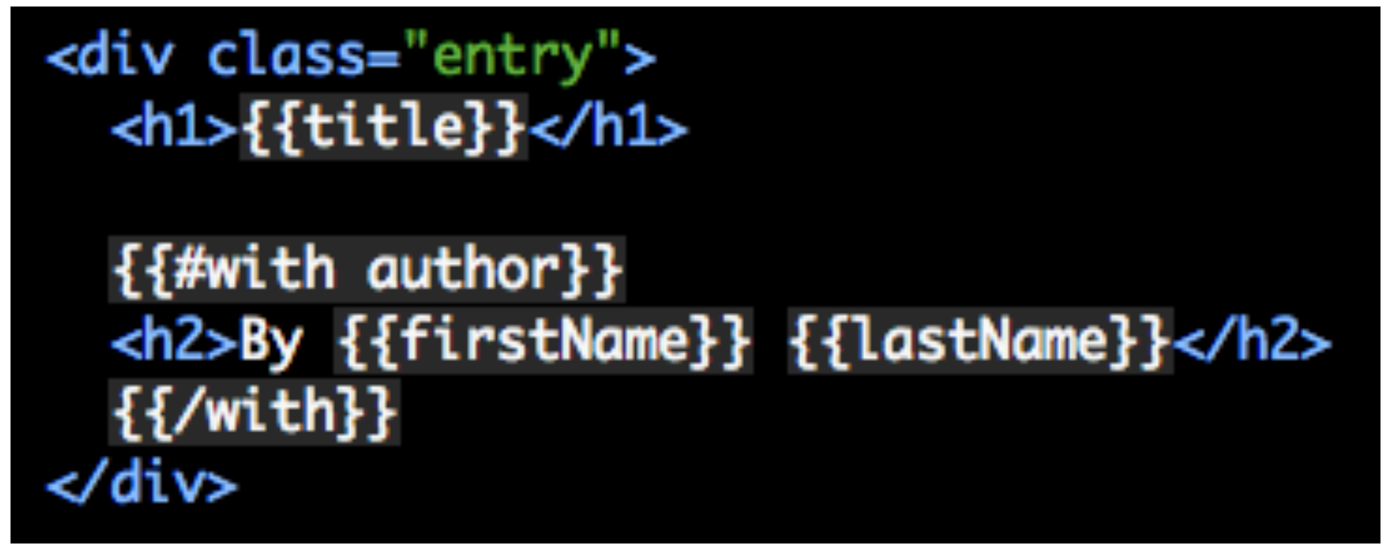
- if
- unless



```
<div class="entry">
  {{#unless license}}
  <h3 class="warning">WARNING: This entry does not have a license!</h3>
  {{/unless}}
</div>
```



```
<ul class="people_list">
  {{#each people}}
    <li>{{this}}</li>
  {{/each}}
</ul>
```

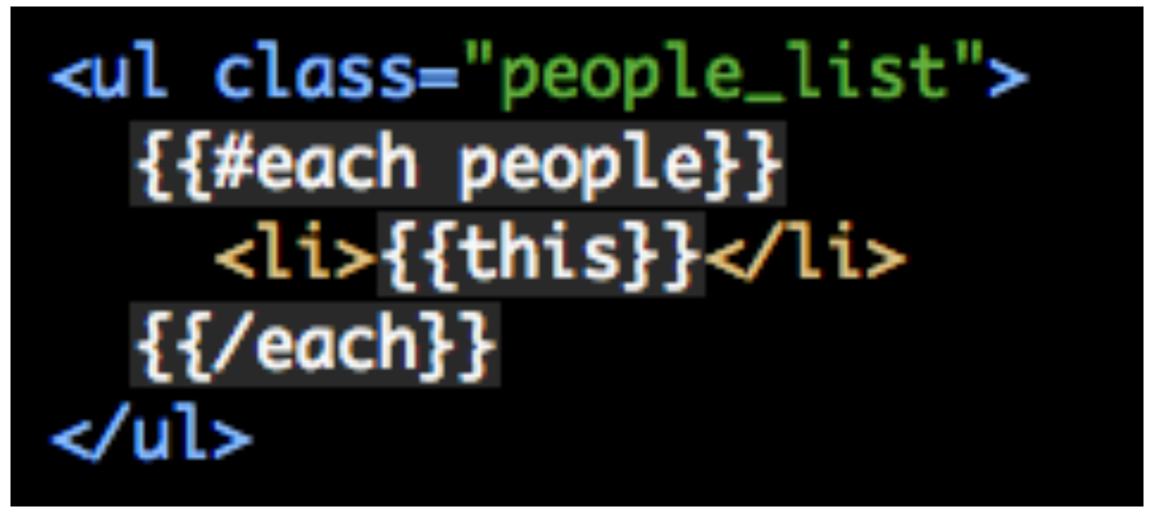


```
<div class="entry">
  <h1>{{title}}</h1>

  {{#with author}}
  <h2>By {{firstName}} {{lastName}}</h2>
  {{/with}}
</div>
```

- each
- with
- lookup
- log

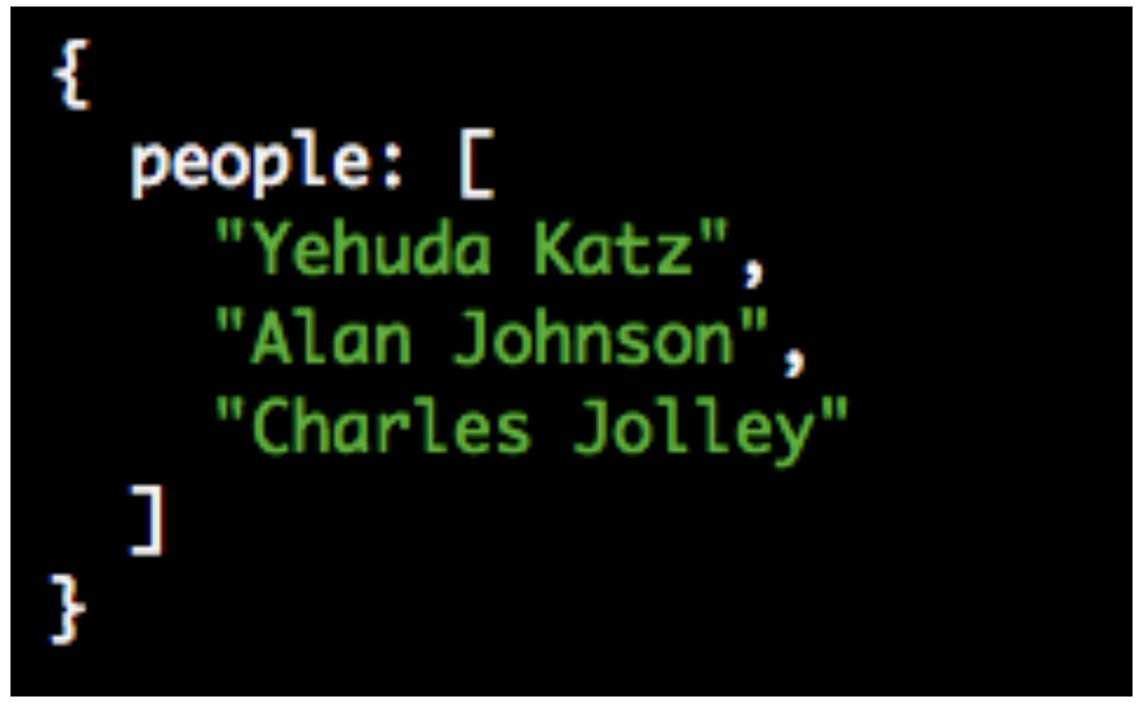
each helper

You can iterate over a list using the built-in each helper. Inside the block, you can use this to reference the element being iterated over.



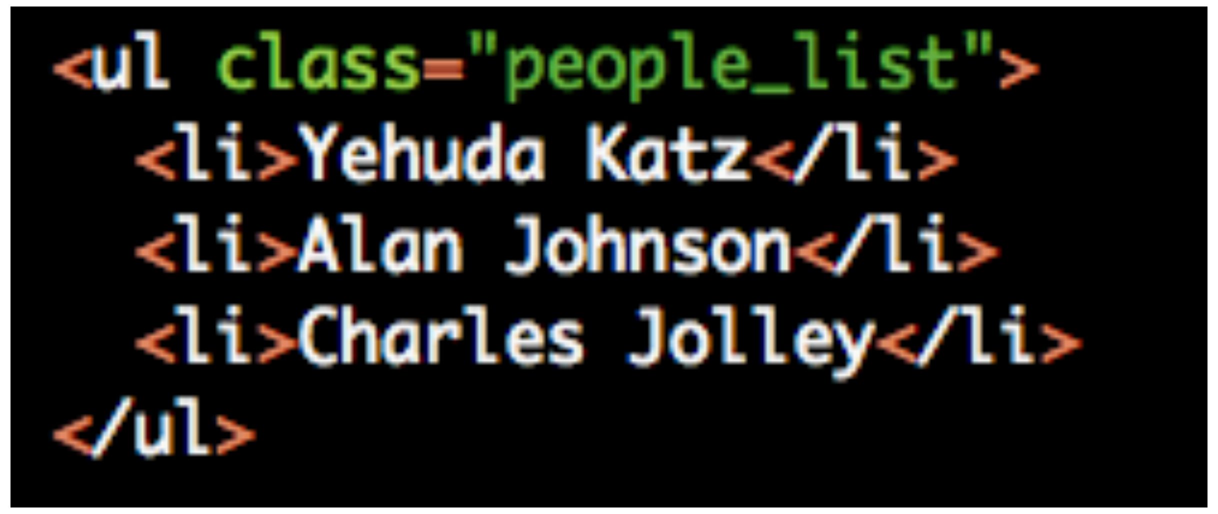
```
<ul class="people_list">
  {{#each people}}
    <li>{{this}}</li>
  {{/each}}
</ul>
```

when used with this context:



```
{
  people: [
    "Yehuda Katz",
    "Alan Johnson",
    "Charles Jolley"
  ]
}
```

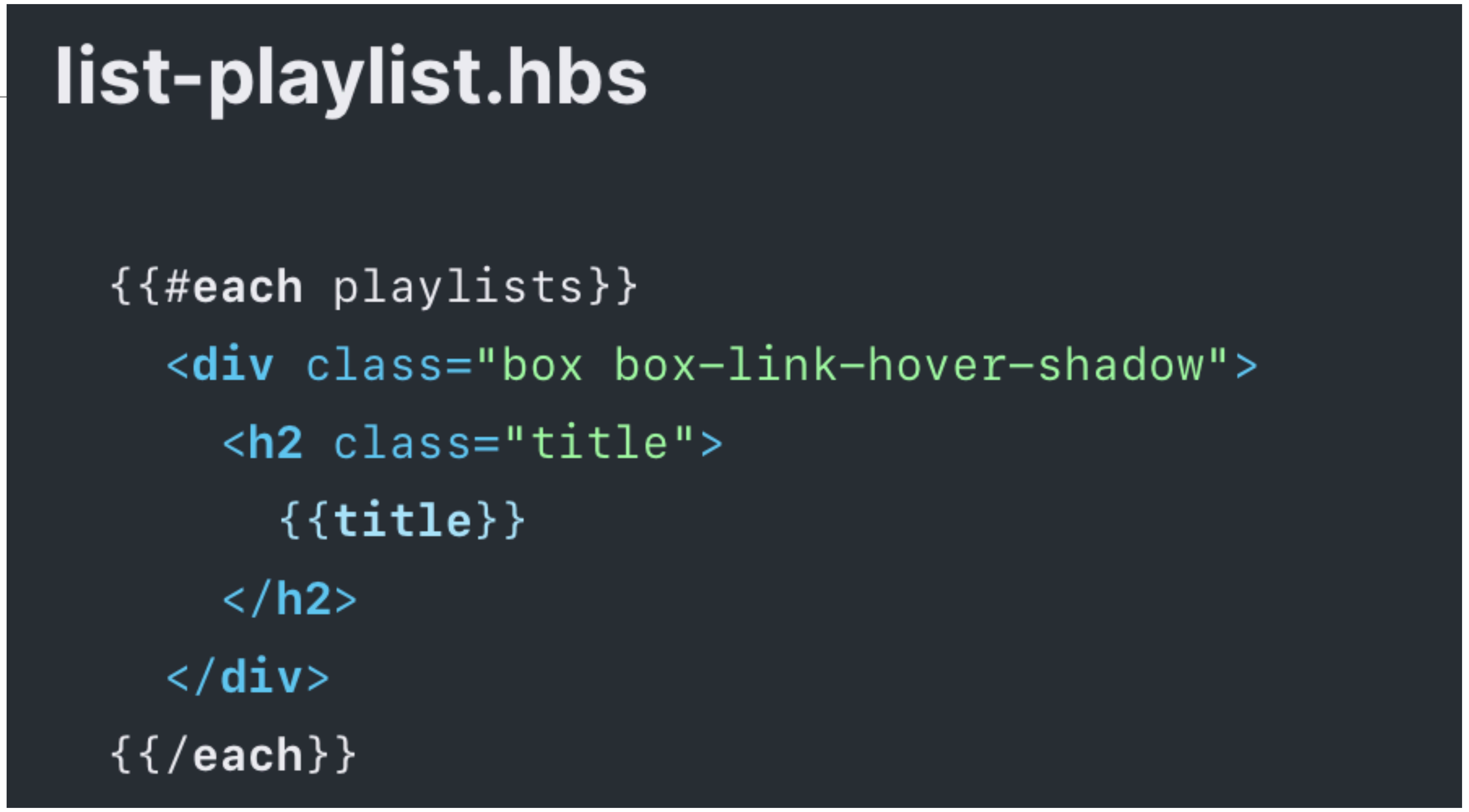
will result in:



```
<ul class="people_list">
  <li>Yehuda Katz</li>
  <li>Alan Johnson</li>
  <li>Charles Jolley</li>
</ul>
```

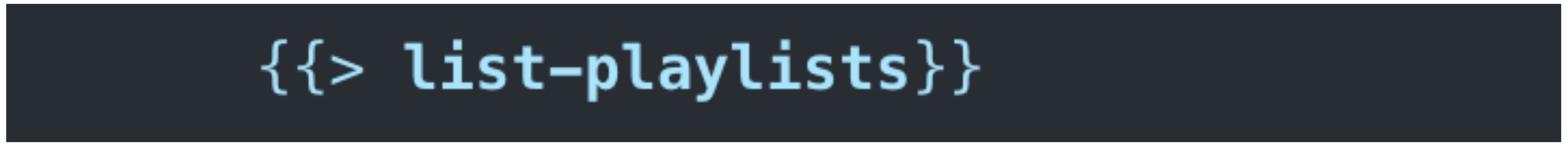
Partials

- Handlebars partials allow for code reuse by creating shared templates.
- Calling the partial is done through the partial call syntax
- Will render the partial named myPartial. When the partial executes, it will be run under the current execution context.



list-playlist.hbs

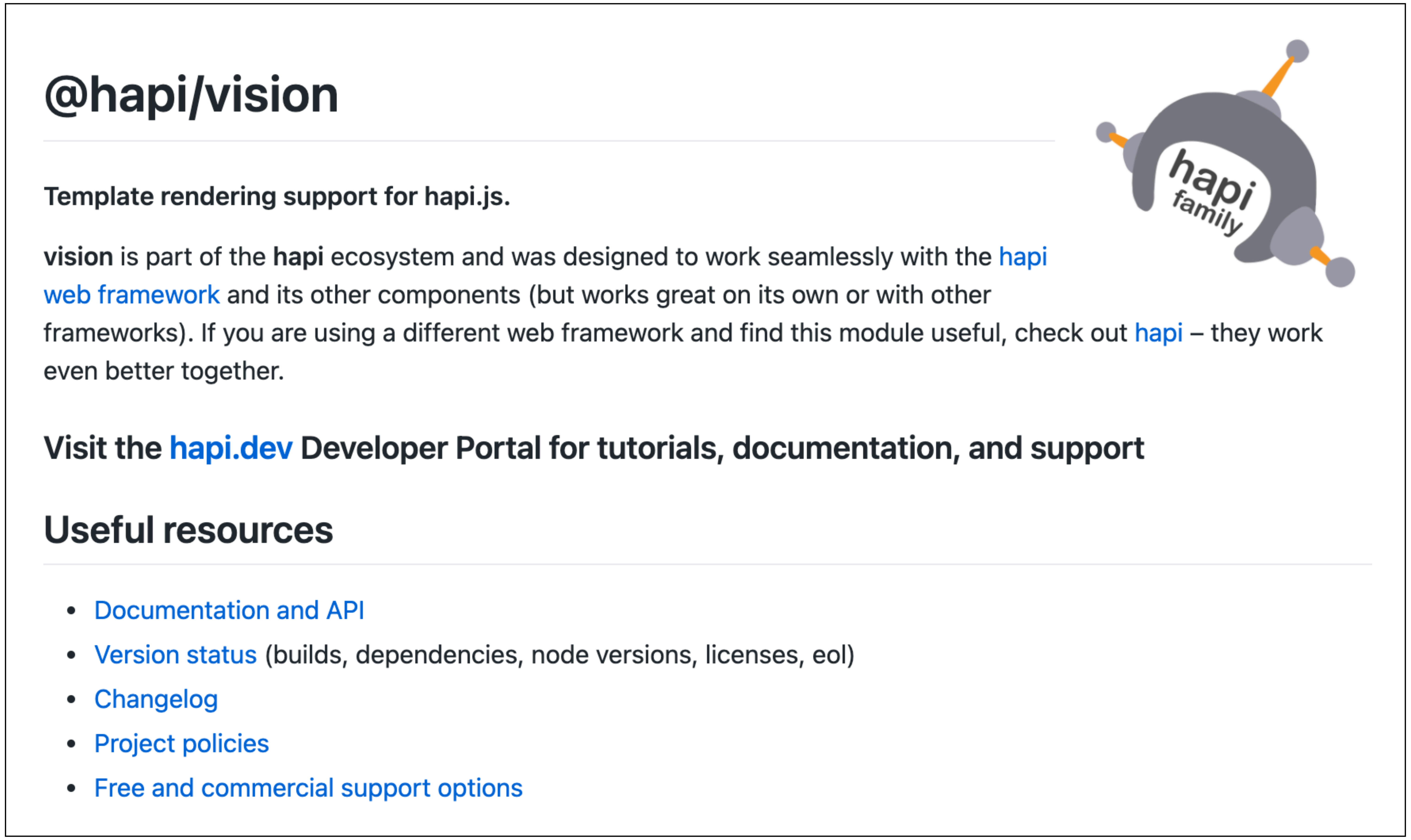
```
{{#each playlists}}
  <div class="box box-link-hover-shadow">
    <h2 class="title">
      {{title}}
    </h2>
  </div>
{{/each}}
```



```
{{> list-playlists}}
```

Handlebars in Hapi

- Vison Plugin loads and manages a templating engine
- Supports a range of tempting languages


@hapi/vision

Template rendering support for hapi.js.

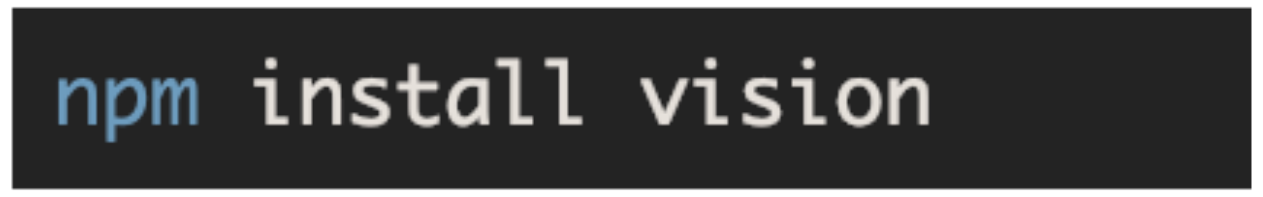
vision is part of the **hapi** ecosystem and was designed to work seamlessly with the hapi web framework and its other components (but works great on its own or with other frameworks). If you are using a different web framework and find this module useful, check out hapi – they work even better together.

Visit the hapi.dev Developer Portal for tutorials, documentation, and support

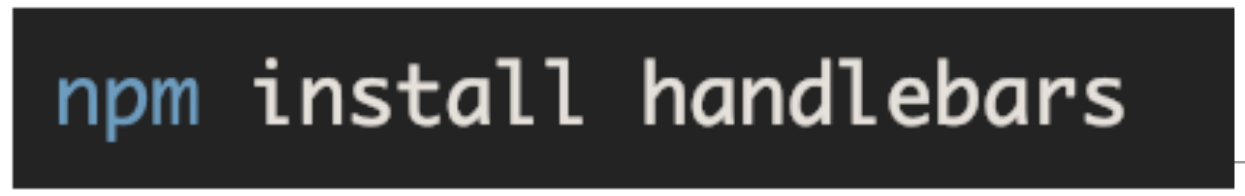
Useful resources

- Documentation and API
- Version status (builds, dependencies, node versions, licenses, eol)
- Changelog
- Project policies
- Free and commercial support options

Plugin Install

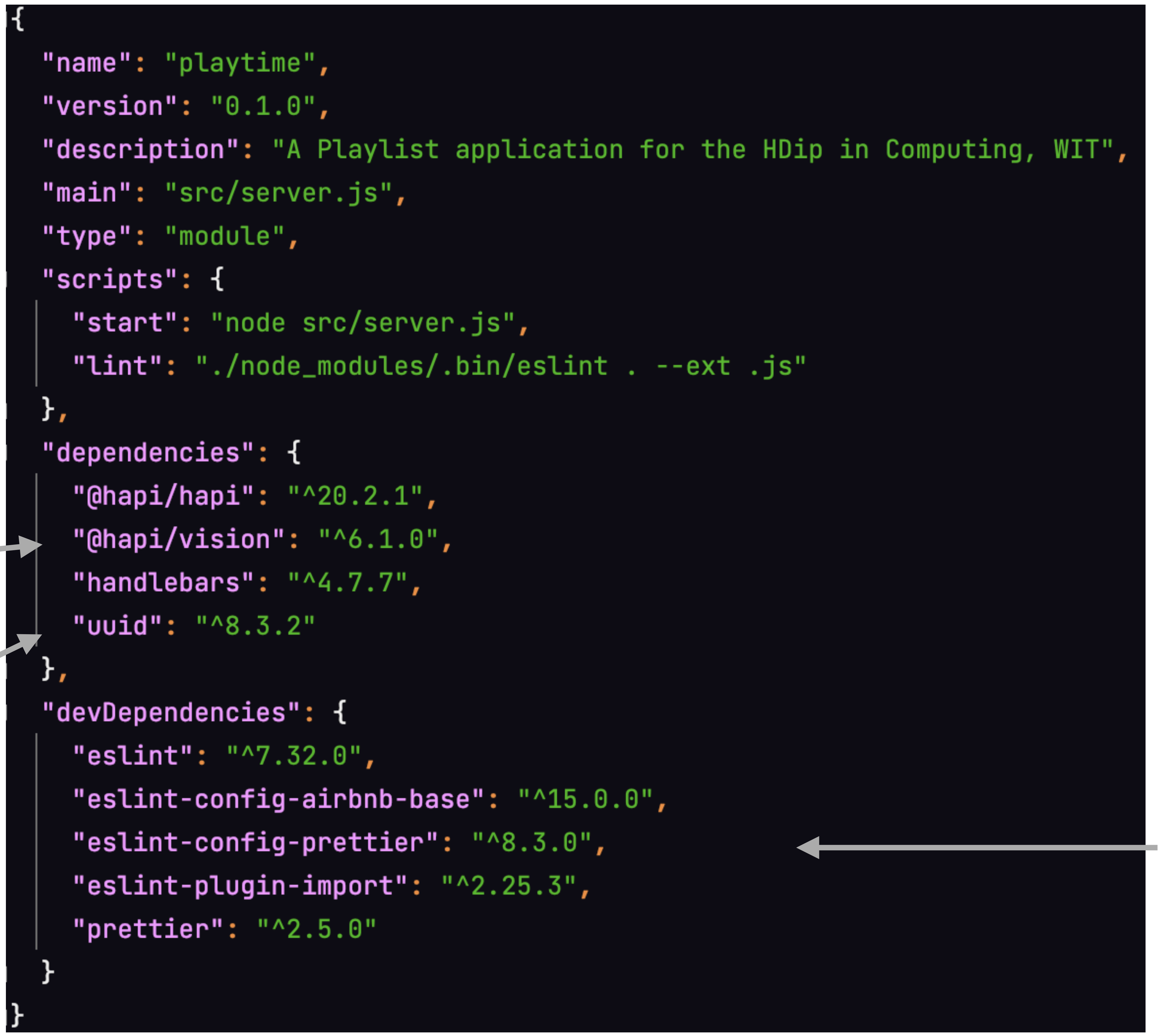


```
npm install vision
```



```
npm install handlebars
```

- Install the Vision plugin + the specific tempting engine you wish to use



```
{
  "name": "playtime",
  "version": "0.1.0",
  "description": "A Playlist application for the HDip in Computing, WIT",
  "main": "src/server.js",
  "type": "module",
  "scripts": {
    "start": "node src/server.js",
    "lint": "./node_modules/.bin/eslint . --ext .js"
  },
  "dependencies": {
    "@hapi/hapi": "^20.2.1",
    "@hapi/vision": "^6.1.0",
    "handlebars": "^4.7.7",
    "uuid": "^8.3.2"
  },
  "devDependencies": {
    "eslint": "^7.32.0",
    "eslint-config-airbnb-base": "^15.0.0",
    "eslint-config-prettier": "^8.3.0",
    "eslint-plugin-import": "^2.25.3",
    "prettier": "^2.5.0"
  }
}
```

Hapi + plugin modules

general purpose node modules

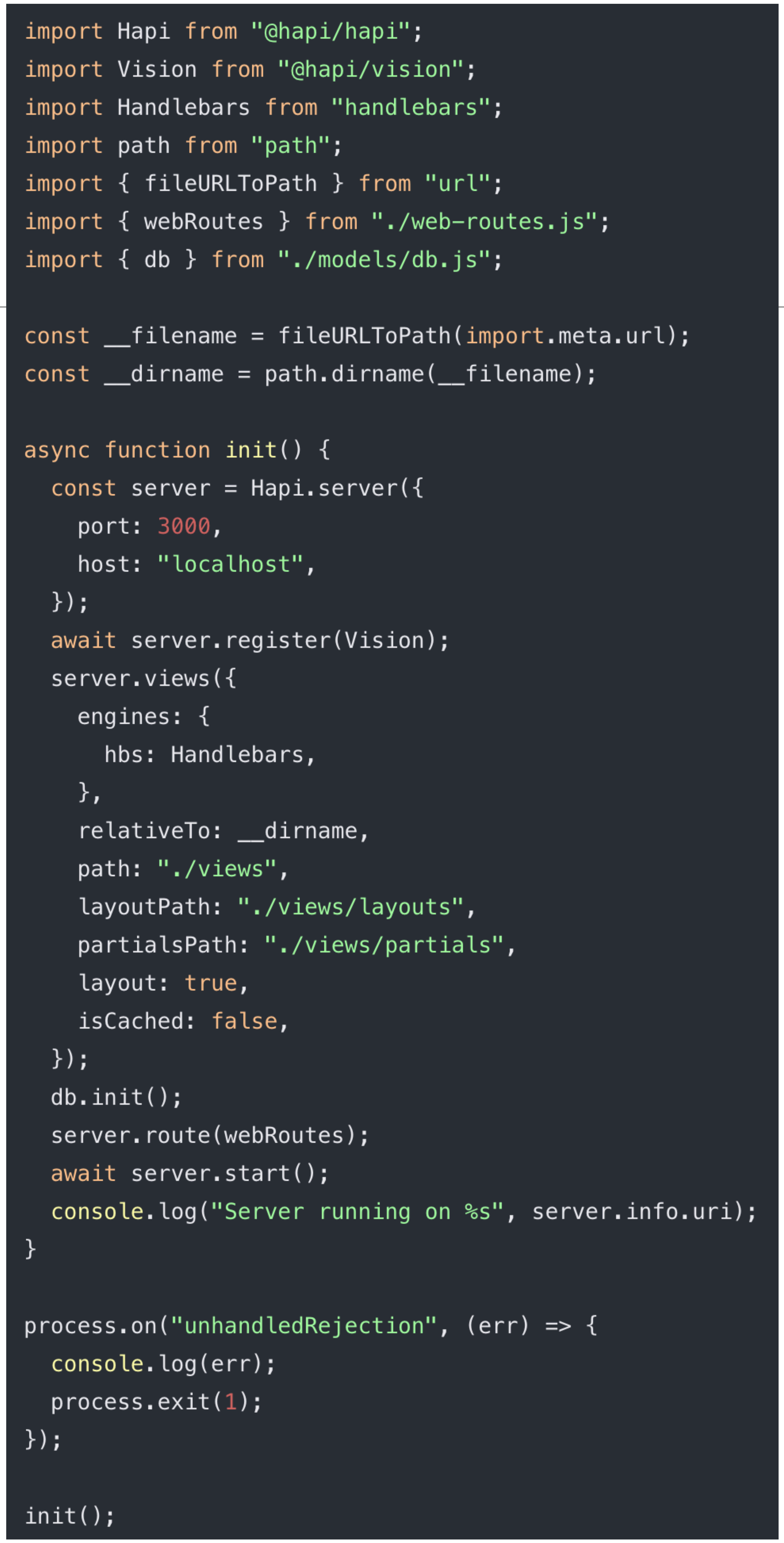
modules used in development only

Register the Plugin

Import & register plugin

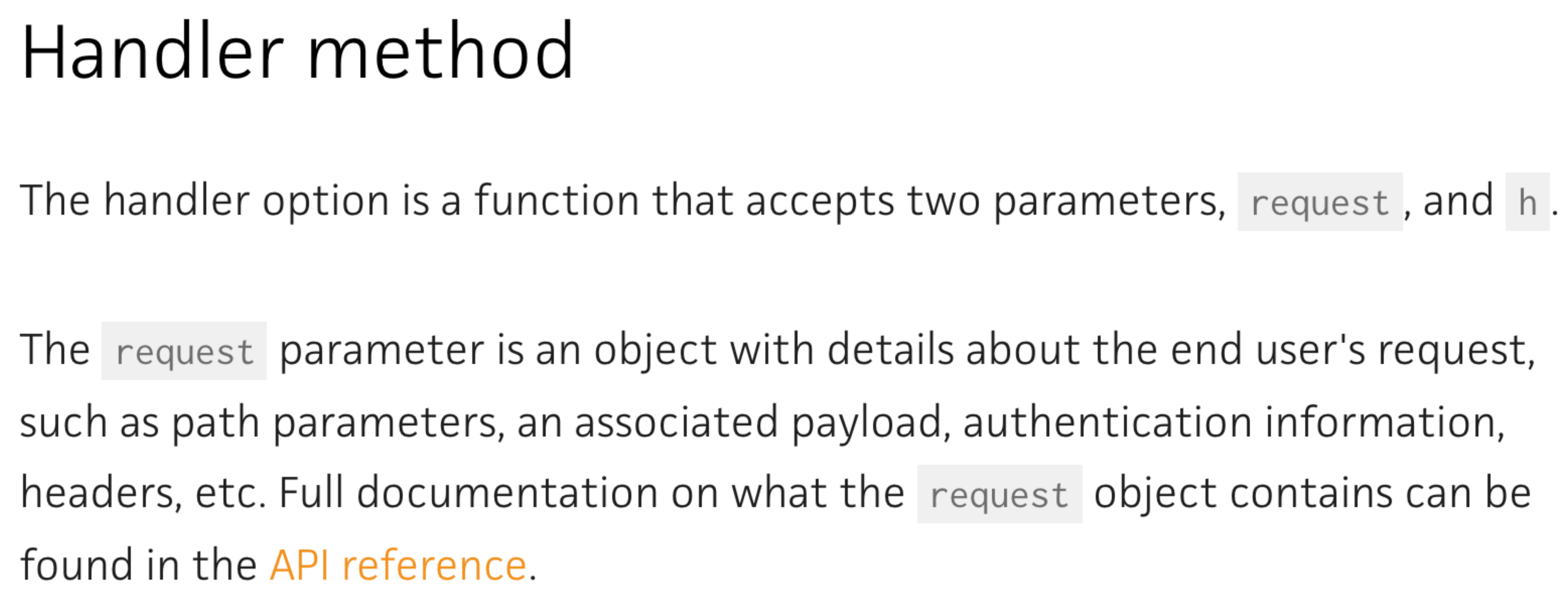
Initialise to use Handlebars engine

Define template locations and cache settings



```javascript
import Hapi from "@hapi/hapi";
import Vision from "@hapi/vision";
import Handlebars from "handlebars";
import path from "path";
import { fileURLToPath } from "url";
import { webRoutes } from "./web-routes.js";
import { db } from "./models/db.js";

const __filename = fileURLToPath(import.meta.url);
const __dirname = path.dirname(__filename);

async function init() {
  const server = Hapi.server({
    port: 3000,
    host: "localhost",
  });
  await server.register(Vision);
  server.views({
    engines: {
      hbs: Handlebars,
    },
    relativeTo: __dirname,
    path: "./views",
    layoutPath: "./views/layouts",
    partialsPath: "./views/partials",
    layout: true,
    isCached: false,
  });
  db.init();
  server.route(webRoutes);
  await server.start();
  console.log("Server running on %s", server.info.uri);
}

process.on("unhandledRejection", (err) => {
  console.log(err);
  process.exit(1);
});

init();
```

```
handler: function(request, h) {
  return h.view('main', { title: 'Welcome to Donations' });
}
```



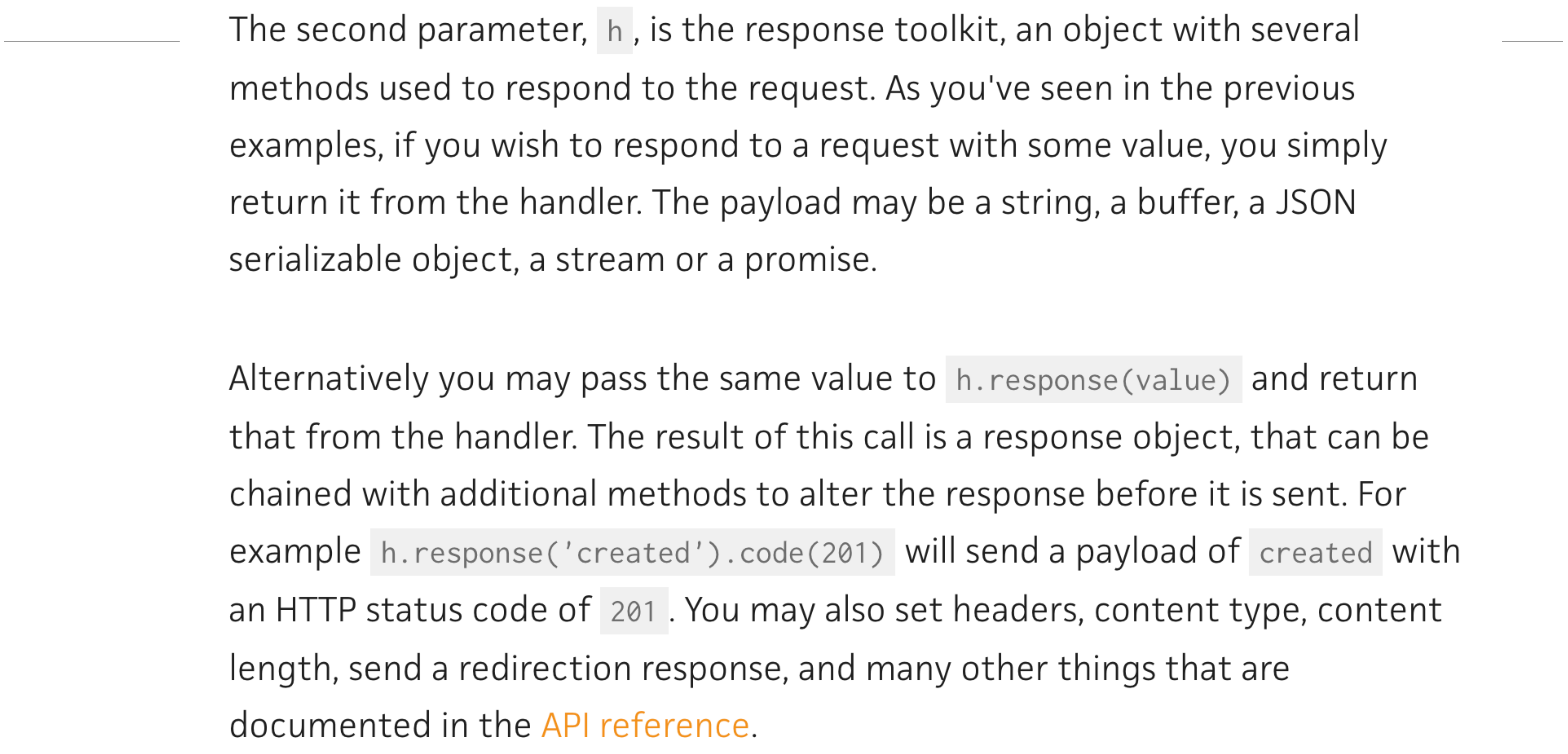
Handler method

The handler option is a function that accepts two parameters, `request`, and `h`.

The `request` parameter is an object with details about the end user's request, such as path parameters, an associated payload, authentication information, headers, etc. Full documentation on what the `request` object contains can be found in the API reference.

https://hapi.dev/tutorials/routing/?lang=en_US

```
handler: function(request, h) {
  return h.view('main', { title: 'Welcome to Donations' });
}
```



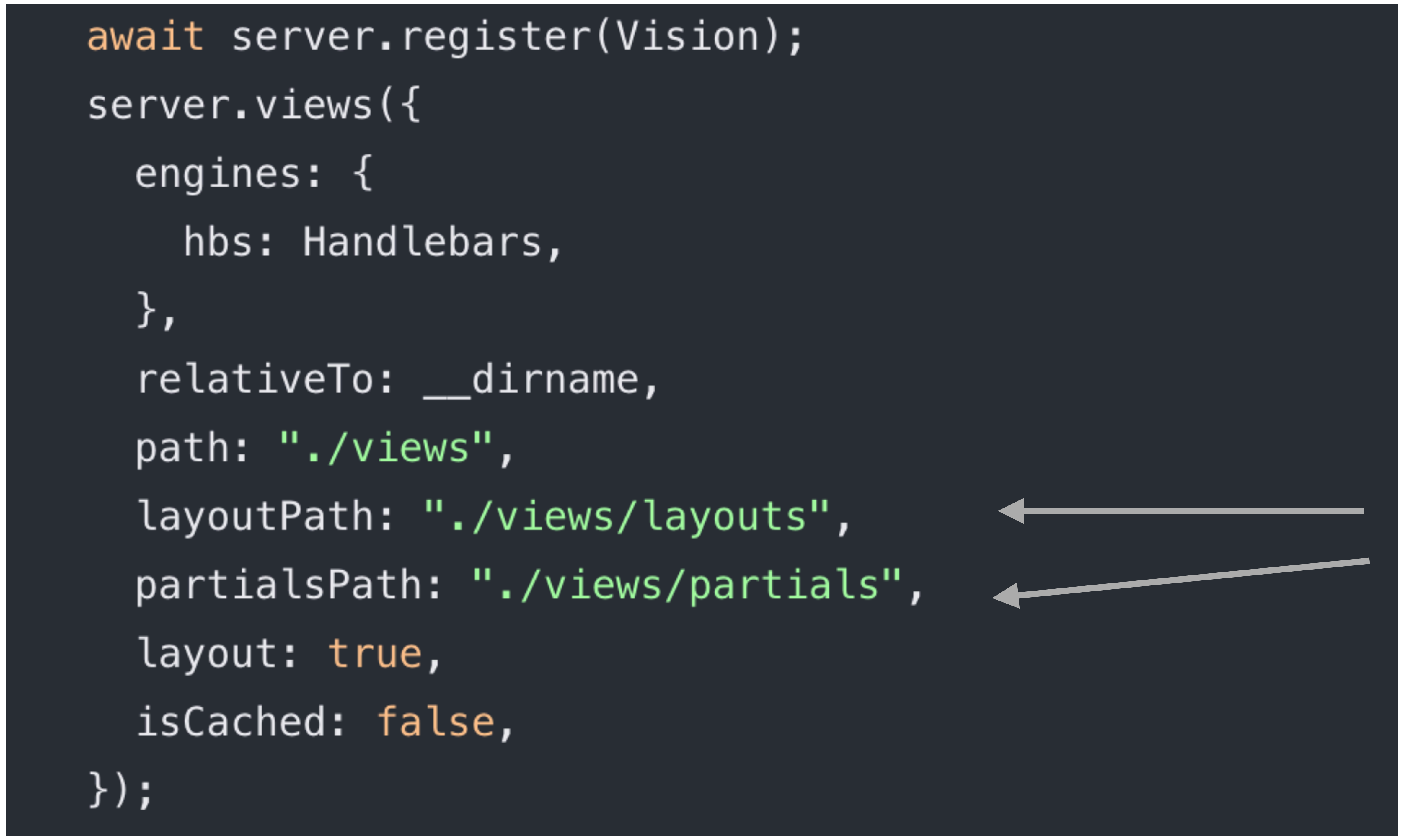
The second parameter, `h`, is the response toolkit, an object with several methods used to respond to the request. As you've seen in the previous examples, if you wish to respond to a request with some value, you simply return it from the handler. The payload may be a string, a buffer, a JSON serializable object, a stream or a promise.

Alternatively you may pass the same value to `h.response(value)` and return that from the handler. The result of this call is a response object, that can be chained with additional methods to alter the response before it is sent. For example `h.response('created').code(201)` will send a payload of `created` with an HTTP status code of `201`. You may also set headers, content type, content length, send a redirection response, and many other things that are documented in the API reference.

https://hapi.dev/tutorials/routing/?lang=en_US

Partials & Layouts

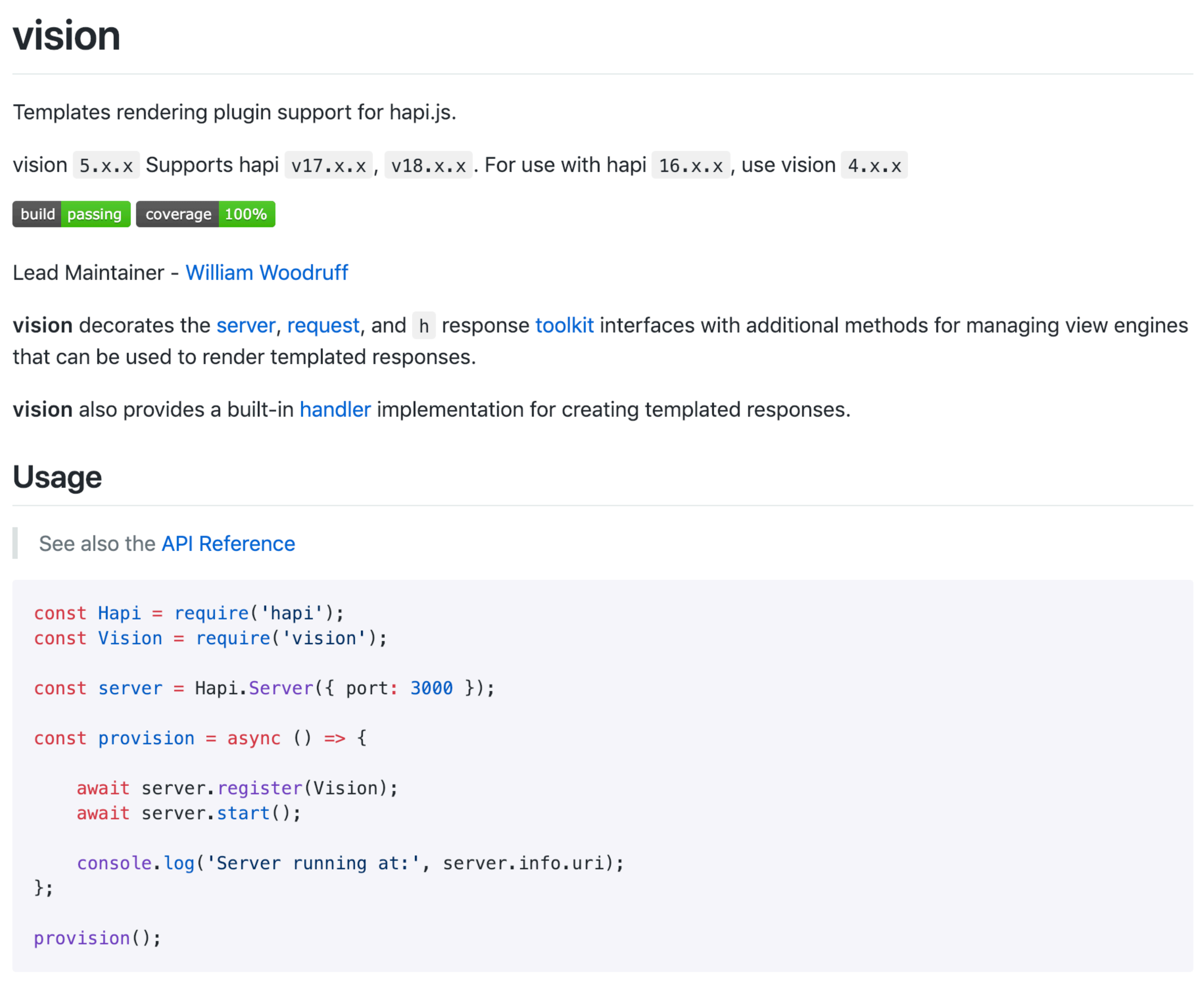
- Partials & Layouts play a prominent role in enabling DRY (Dont Repeat Yourself) principles
  - Partials: Reusable templates
  - Layouts: Reusable Page Structure
- These features must be explicitly enabled



```
await server.register(Vision);
server.views({
  engines: {
    hbs: Handlebars,
  },
  relativeTo: __dirname,
  path: "./views",
  layoutPath: "./views/layouts",
  partialsPath: "./views/partials",
  layout: true,
  isCached: false,
});
```

partials & layouts directories in project

Alternatives to Handlebars



vision

Templates rendering plugin support for hapi.js.

vision `5.x.x` Supports hapi `v17.x.x`, `v18.x.x`. For use with hapi `16.x.x`, use vision `4.x.x`

Lead Maintainer - William Woodruff

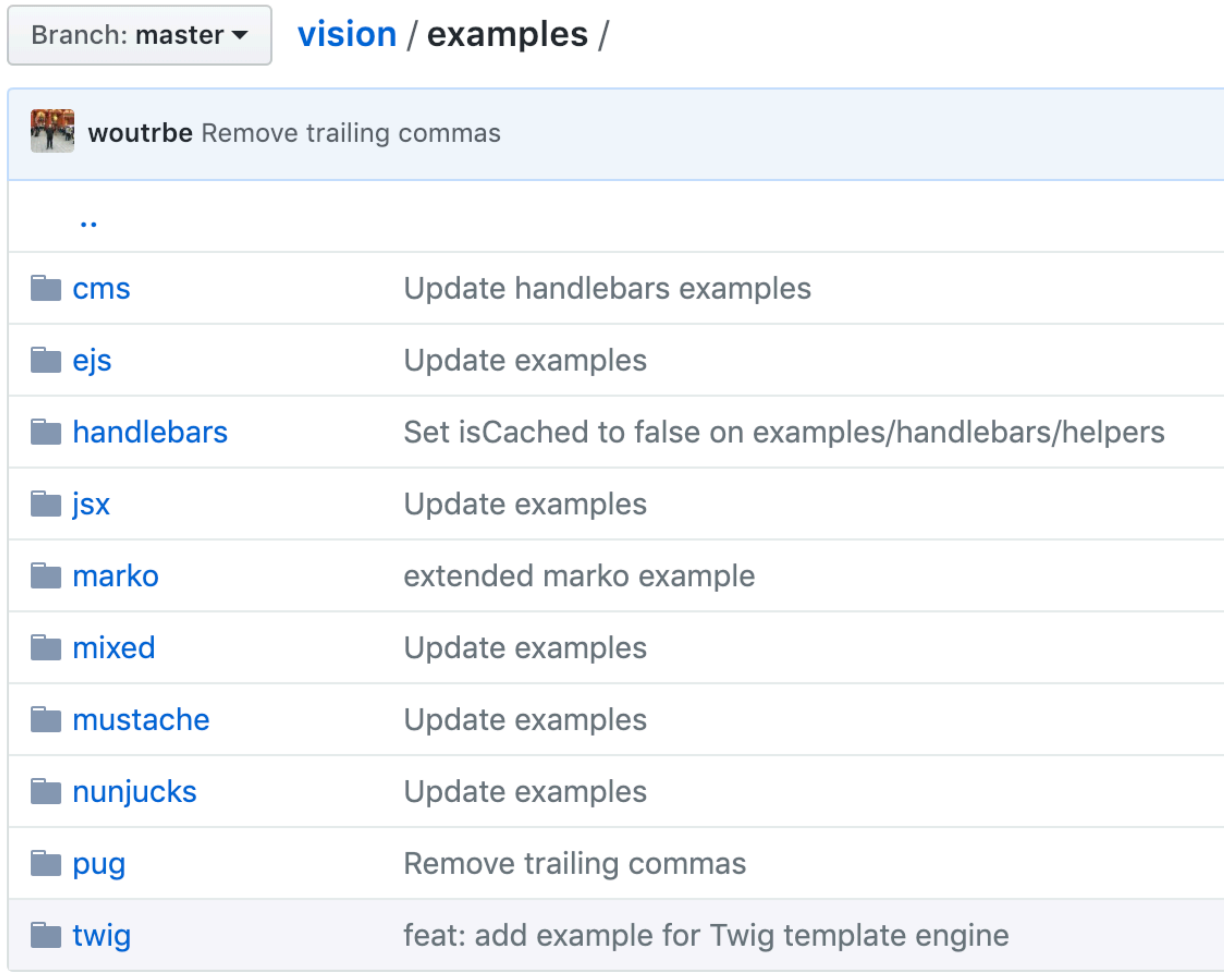
vision decorates the server, request, and `h` response toolkit interfaces with additional methods for managing view engines that can be used to render templated responses.

vision also provides a built-in handler implementation for creating templated responses.

Usage

> See also the API Reference

```
const Hapi = require('hapi');
const Vision = require('vision');

const server = Hapi.Server({ port: 3000 });

const provision = async () => {

    await server.register(Vision);
    await server.start();

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

provision();
```

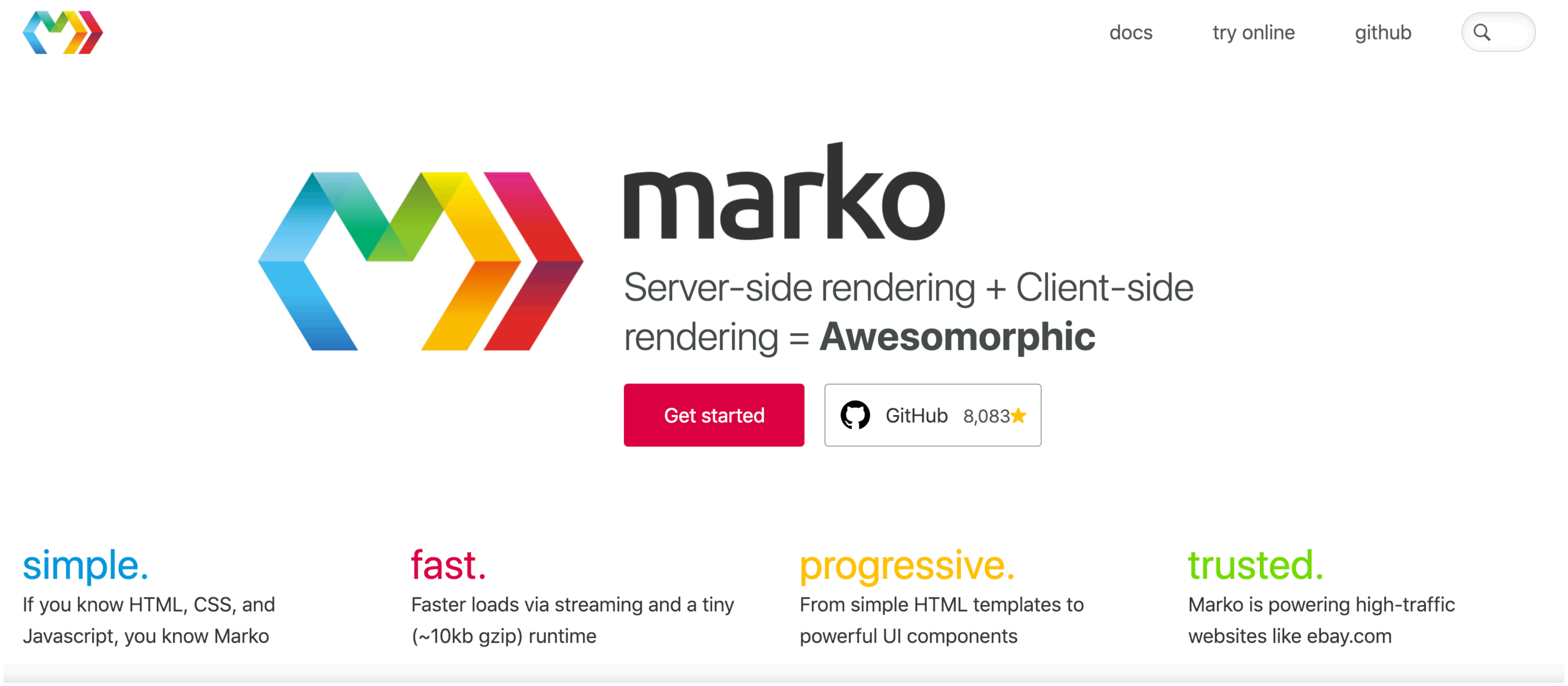


Branch: master ▾ vision / examples /

woutrbe Remove trailing commas

..	
cms	Update handlebars examples
ejs	Update examples
handlebars	Set isCached to false on examples/handlebars/helpers
jsx	Update examples
marko	extended marko example
mixed	Update examples
mustache	Update examples
nunjucks	Update examples
pug	Remove trailing commas
twig	feat: add example for Twig template engine

https://github.com/hapijs/vision/tree/master/examples https://markojs.com/



marko

Server-side rendering + Client-side rendering = **Awesomorphic**

simple.
If you know HTML, CSS, and Javascript, you know Marko

fast.
Faster loads via streaming and a tiny (~10kb gzip) runtime

progressive.
From simple HTML templates to powerful UI components

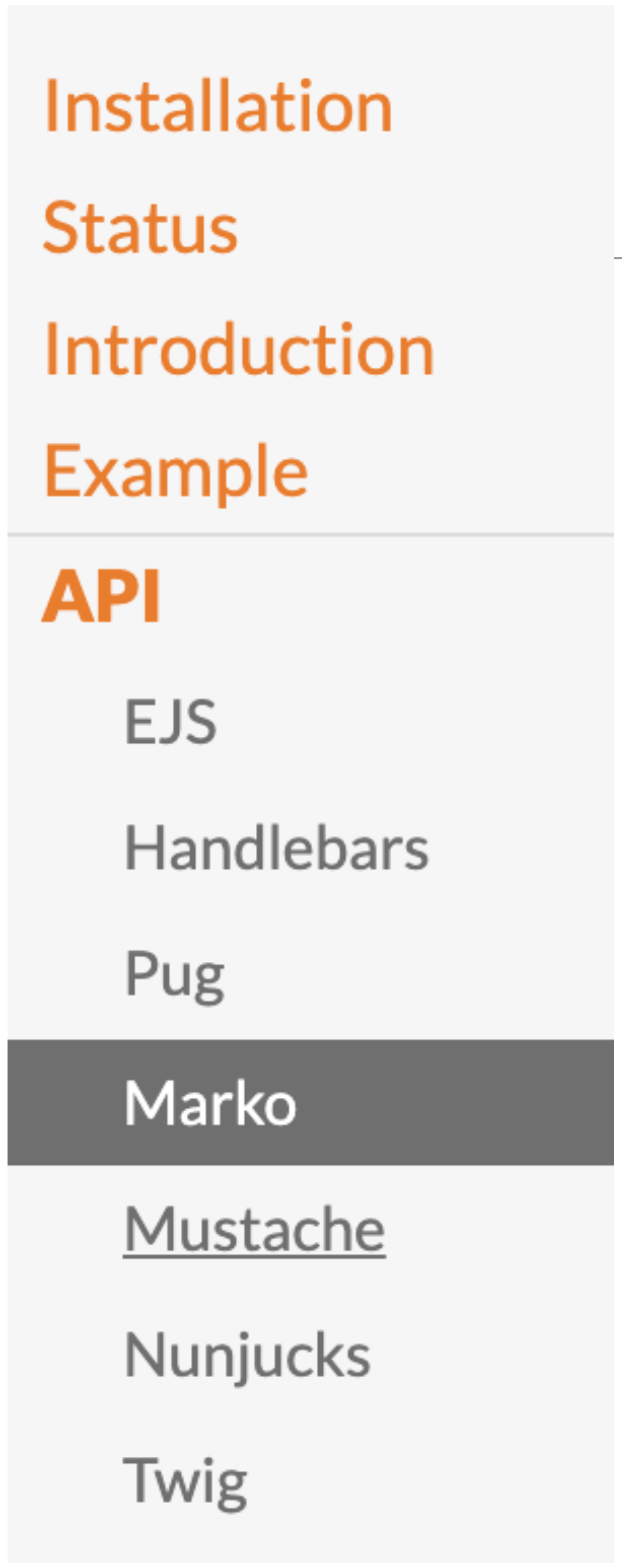
trusted.
Marko is powering high-traffic websites like ebay.com

Choose a syntax

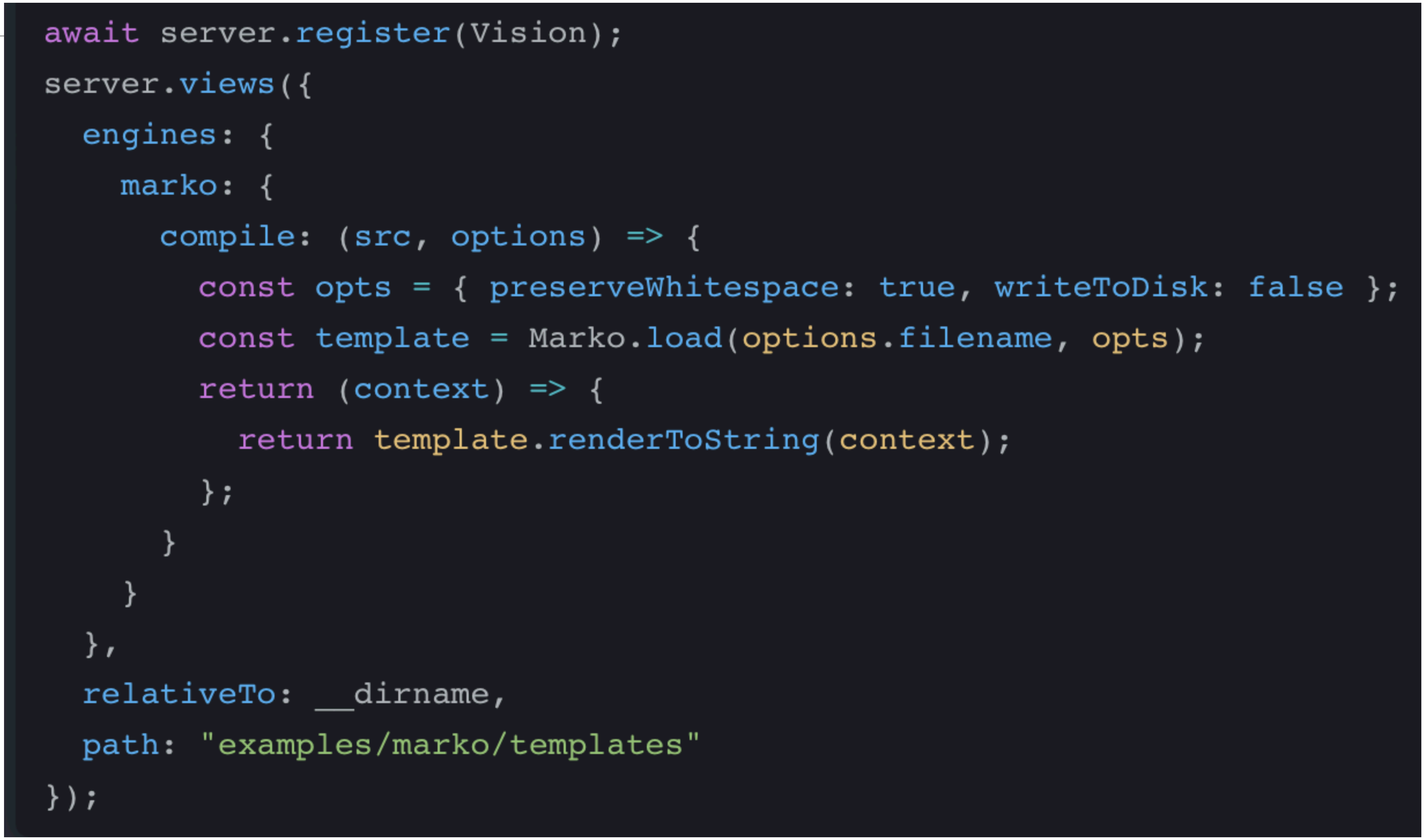
Write in a familiar HTML-like style or drop the angle brackets and use Marko's concise syntax

```
<!doctype html>
<html>
<head>
    <title>Hello Marko</title>
</head>
<body>
    <h1>My favorite colors</h1>
    <ul.colors>
        <li for(color in input.colors)>
            ${color}
        </li>
    </ul>
</body>
</html>
```

```
<!doctype html>
html
    head
        title -- Hello Marko
    body
        h1 -- My favorite colors
        ul.colors
            li for(color in input.colors)
                -- ${color}
```



Vision Configuration



```
await server.register(Vision);
server.views({
  engines: {
    marko: {
      compile: (src, options) => {
        const opts = { preserveWhitespace: true, writeToDisk: false };
        const template = Marko.load(options.filename, opts);
        return (context) => {
          return template.renderToString(context);
        };
      }
    }
  },
  relativeTo: __dirname,
  path: "examples/marko/templates"
});
```

https://hapi.dev/module/vision

Views

Full Stack Web Development