Templates

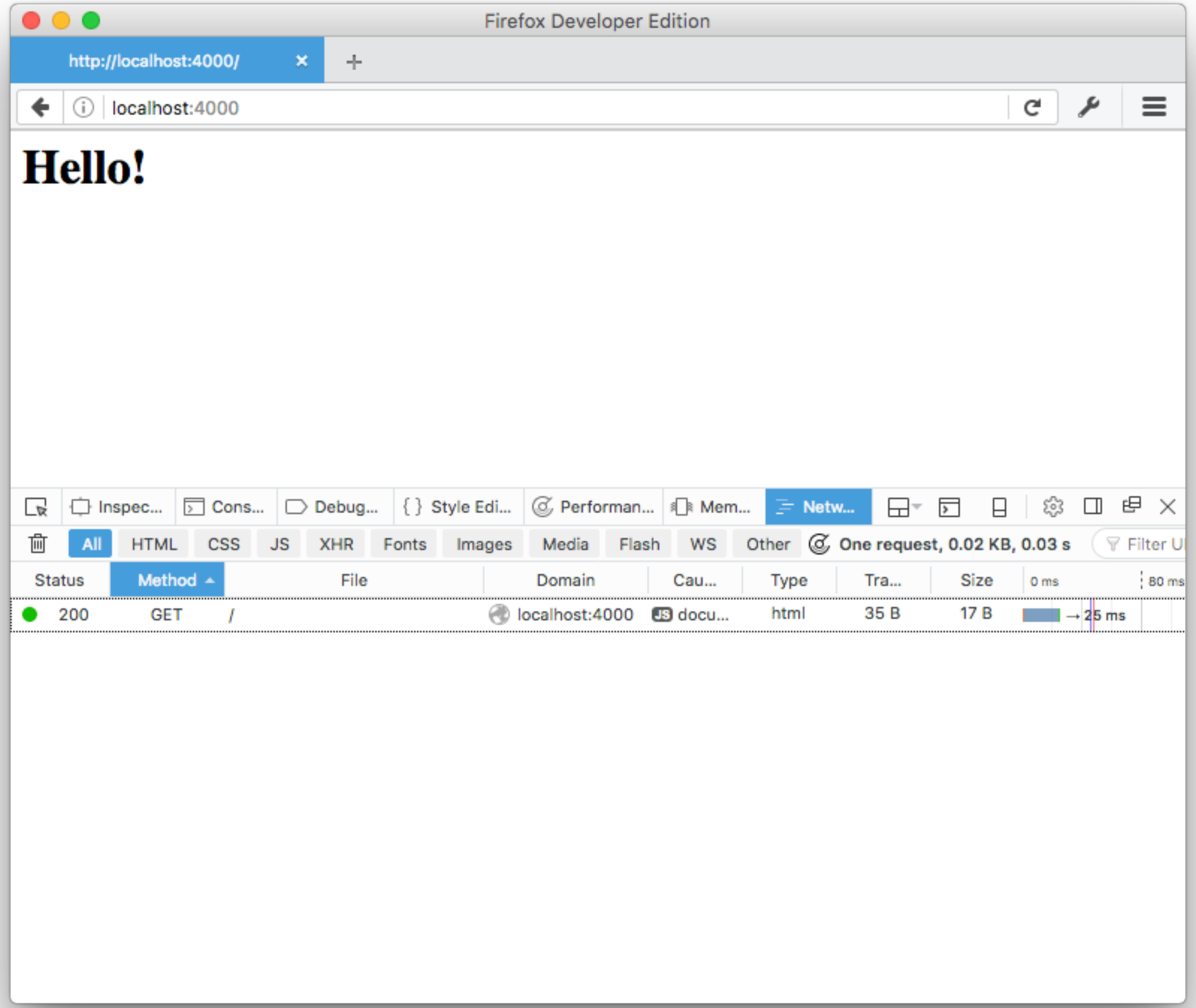
Agenda

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

reply

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

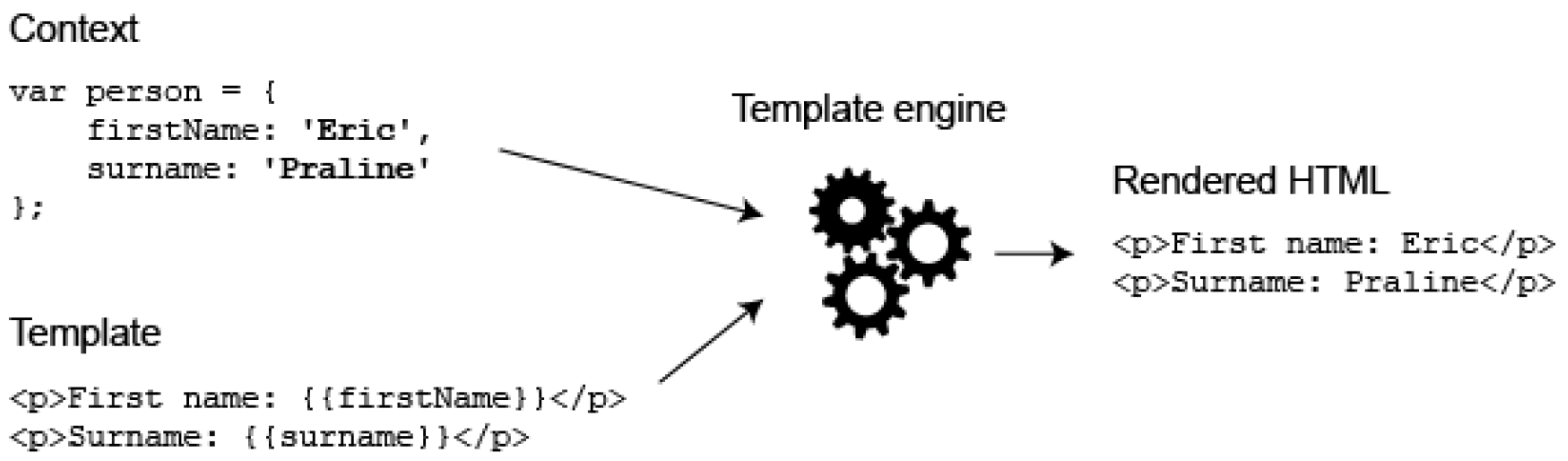
```
exports.index = {

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

};
```



One request, 0.02 KB, 0.03 s

Status	Method	File	Domain	Cau...	Type	Tra...	Size	0 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>
```

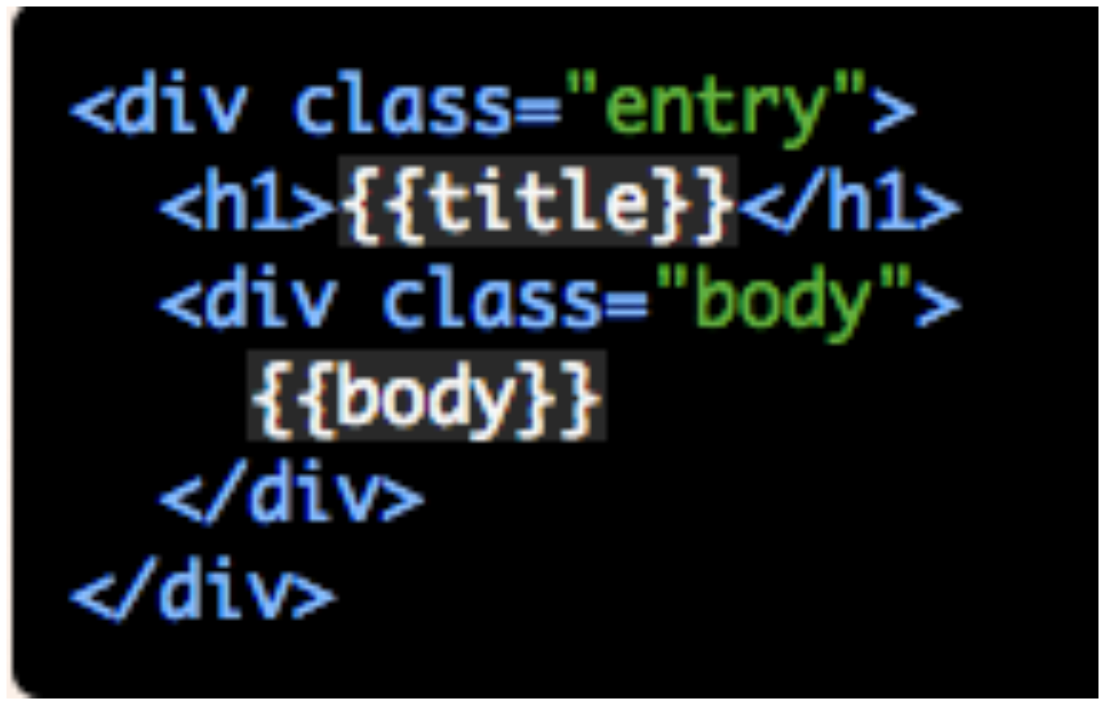
Rendered HTML

```
<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.

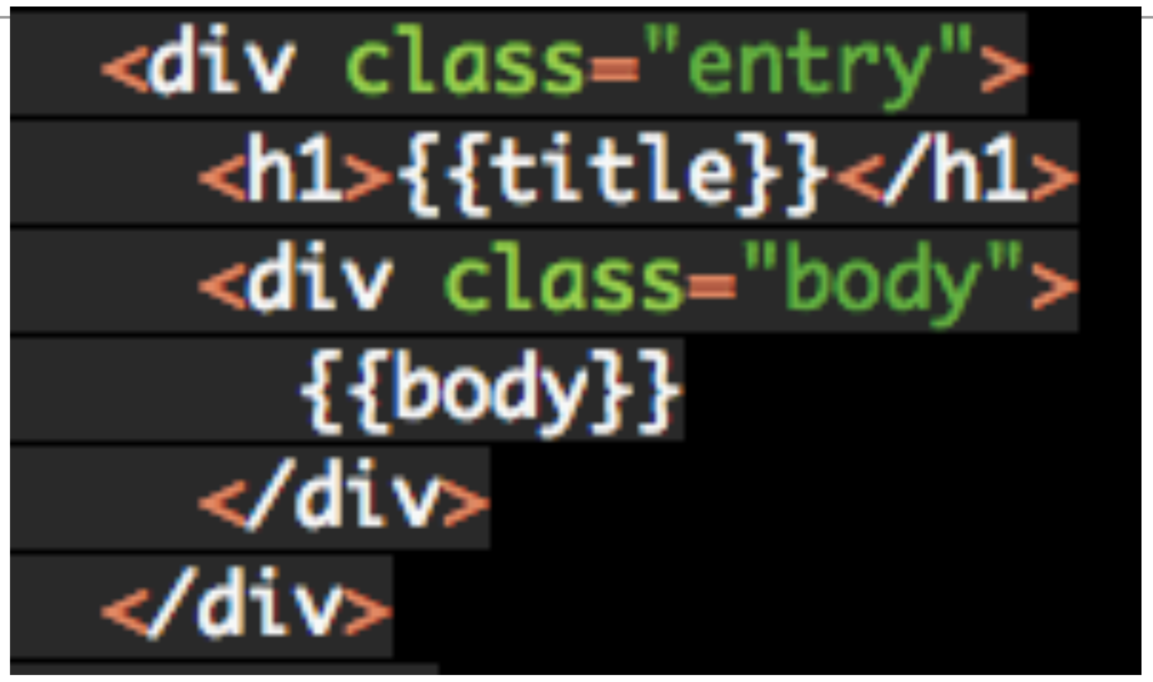
Handlebars is largely compatible with Mustache templates. In most cases it is possible to swap out Mustache with Handlebars and continue using your current templates..”



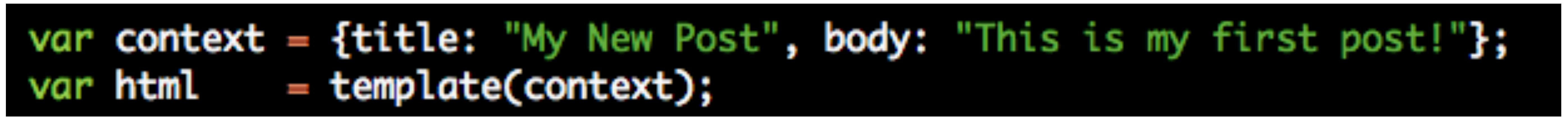
```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```

Template Expressions

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



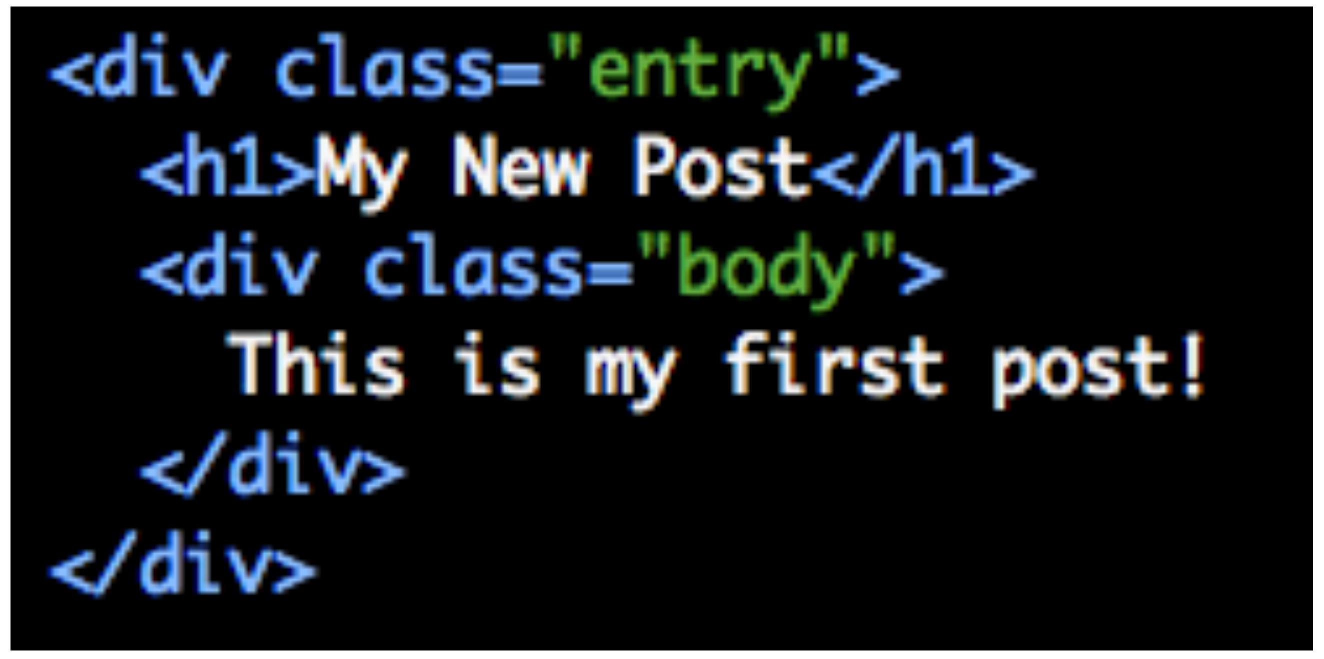
```
<div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
    {{body}}
  </div>
</div>
```



```
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

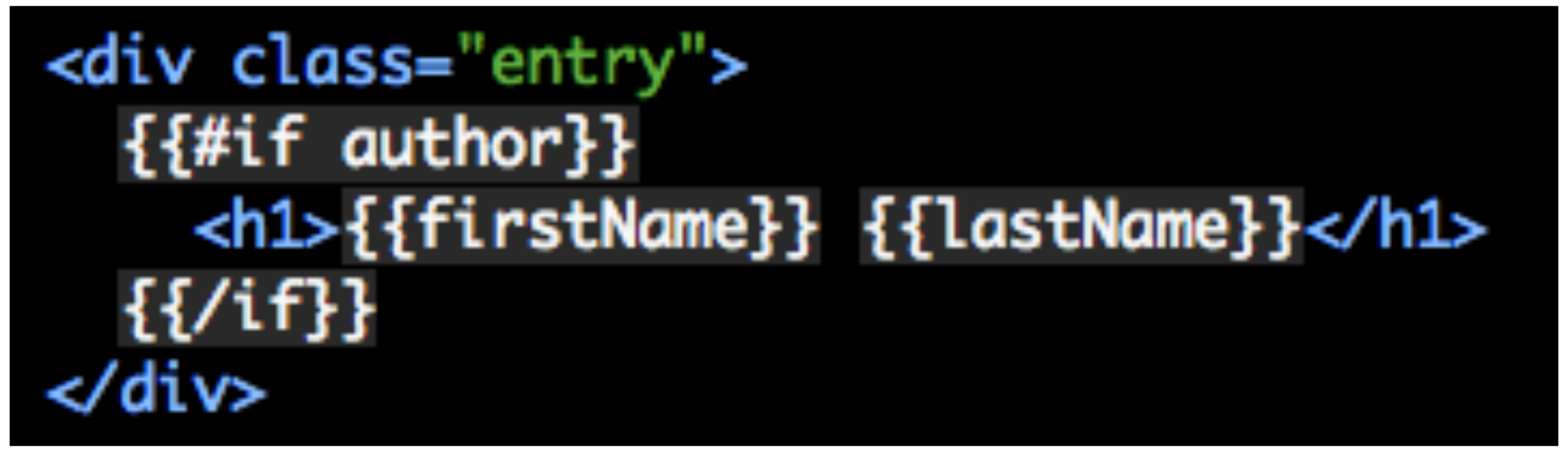
- Precompilation
- Execution

integrated into Hapi by ‘views’ plugin

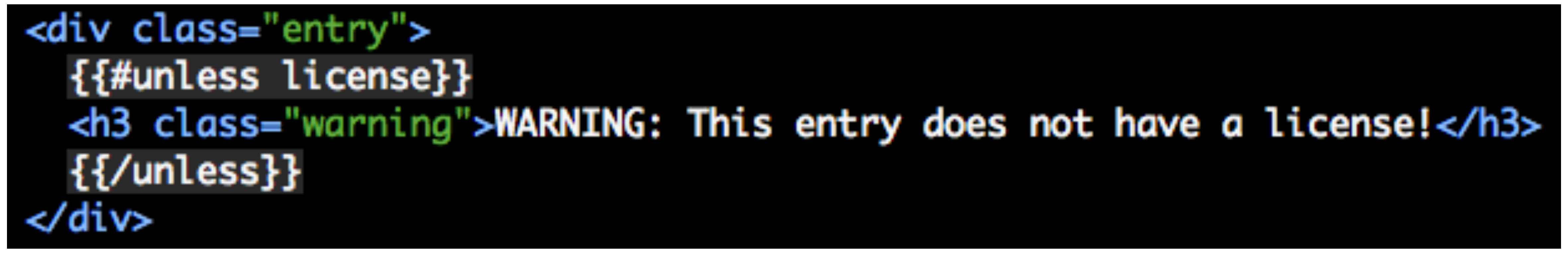
Helpers

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.

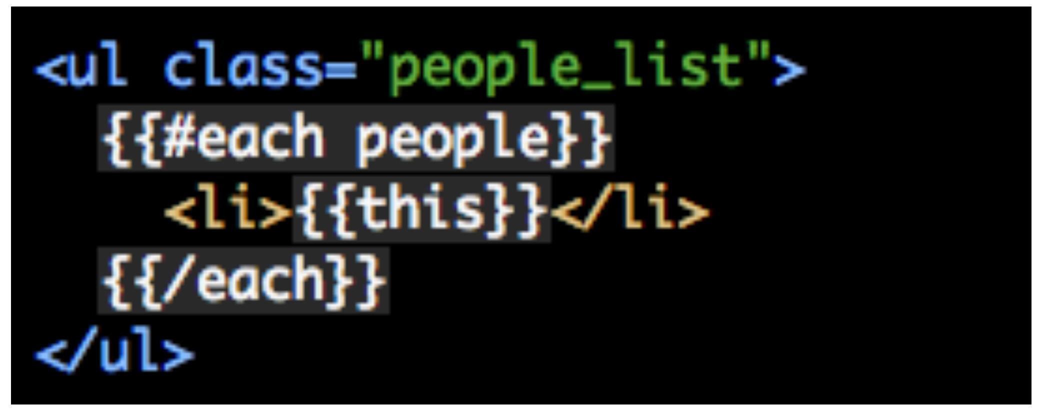


```
<div class="entry">
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{/if}}
</div>
```

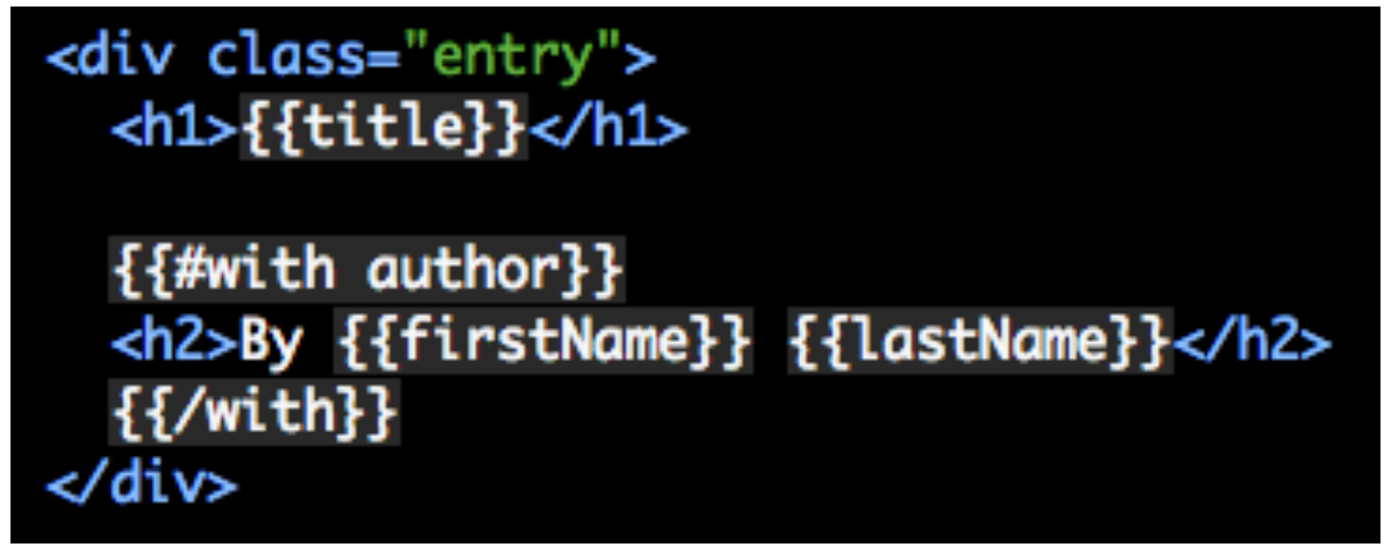


```
<div class="entry">
  {{#unless license}}
  <h3 class="warning">WARNING: This entry does not have a license!</h3>
  {{/unless}}
</div>
```

- if
- unless



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

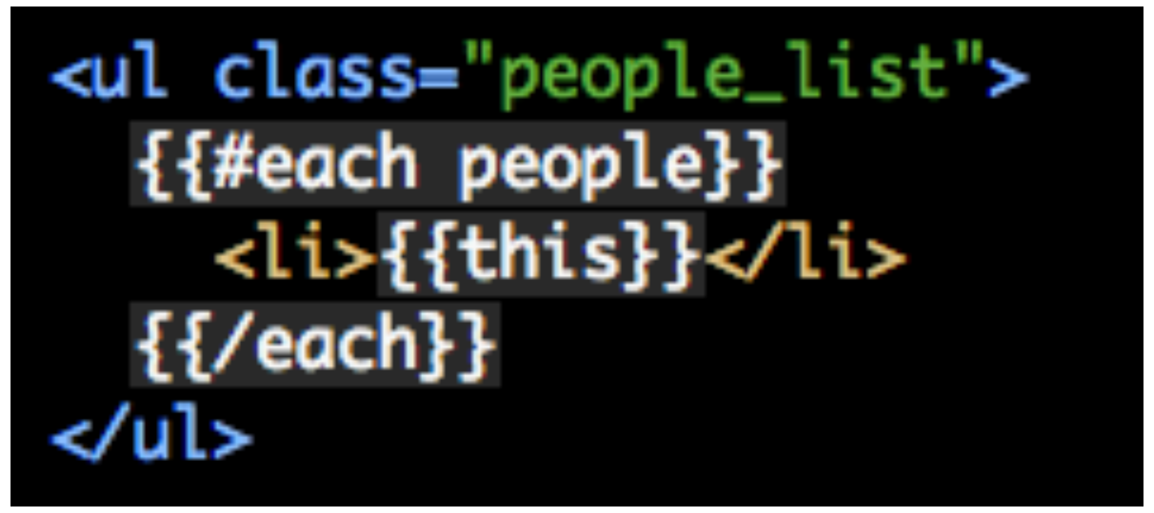


```
<div class="entry">
  <h1>{{title}}</h1>

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

- each
- with
- lookup
- log

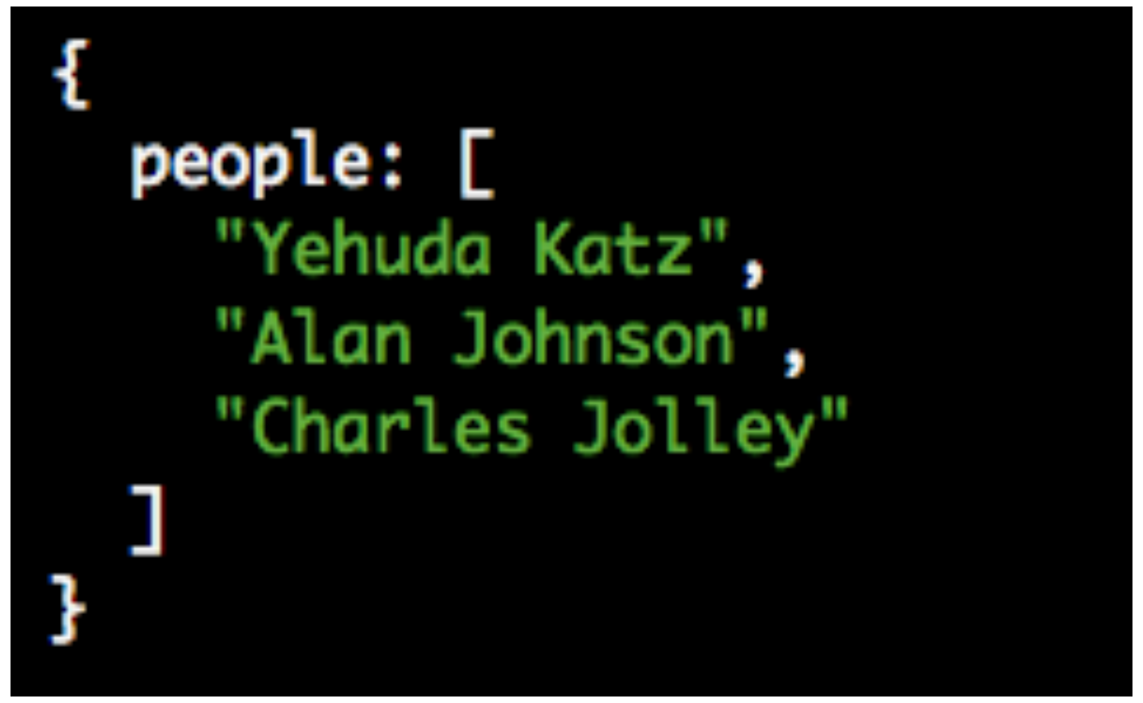
each helper

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.



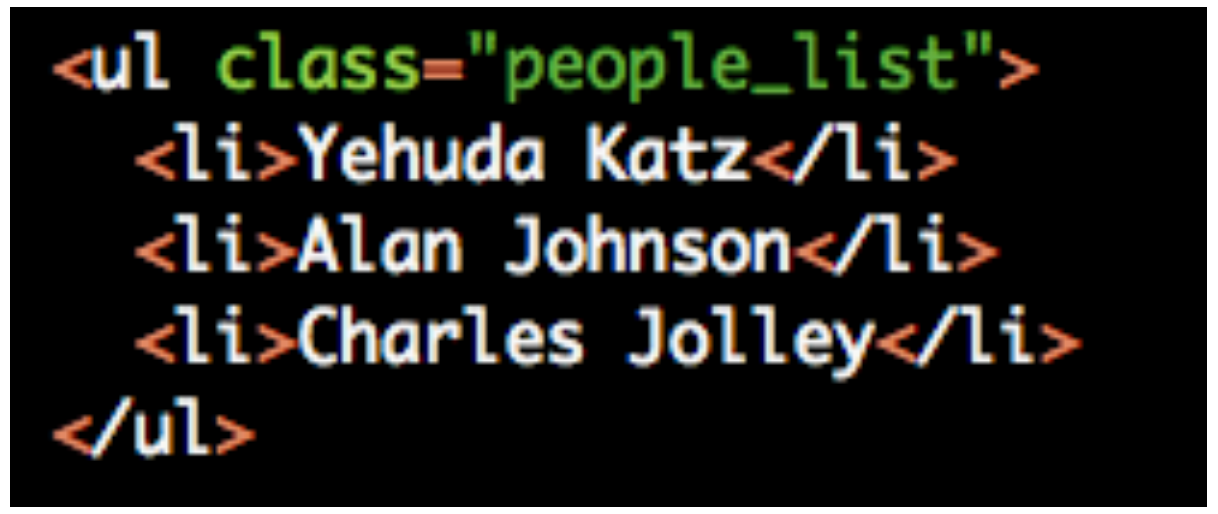
```
<ul class="people_list">
  {{#each people}}
    <li>{{this}}</li>
  {{/each}}
</ul>
```

when used with this context:



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

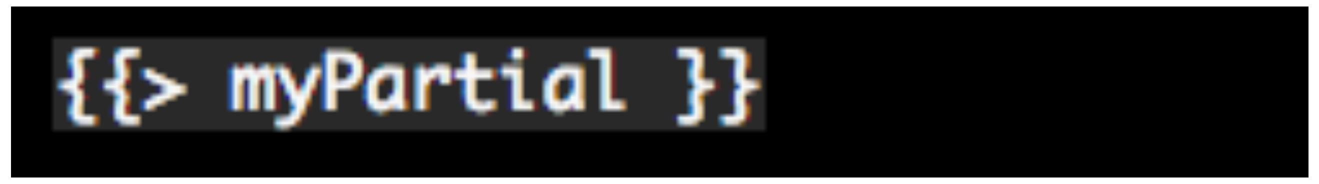
will result in:



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

Partials

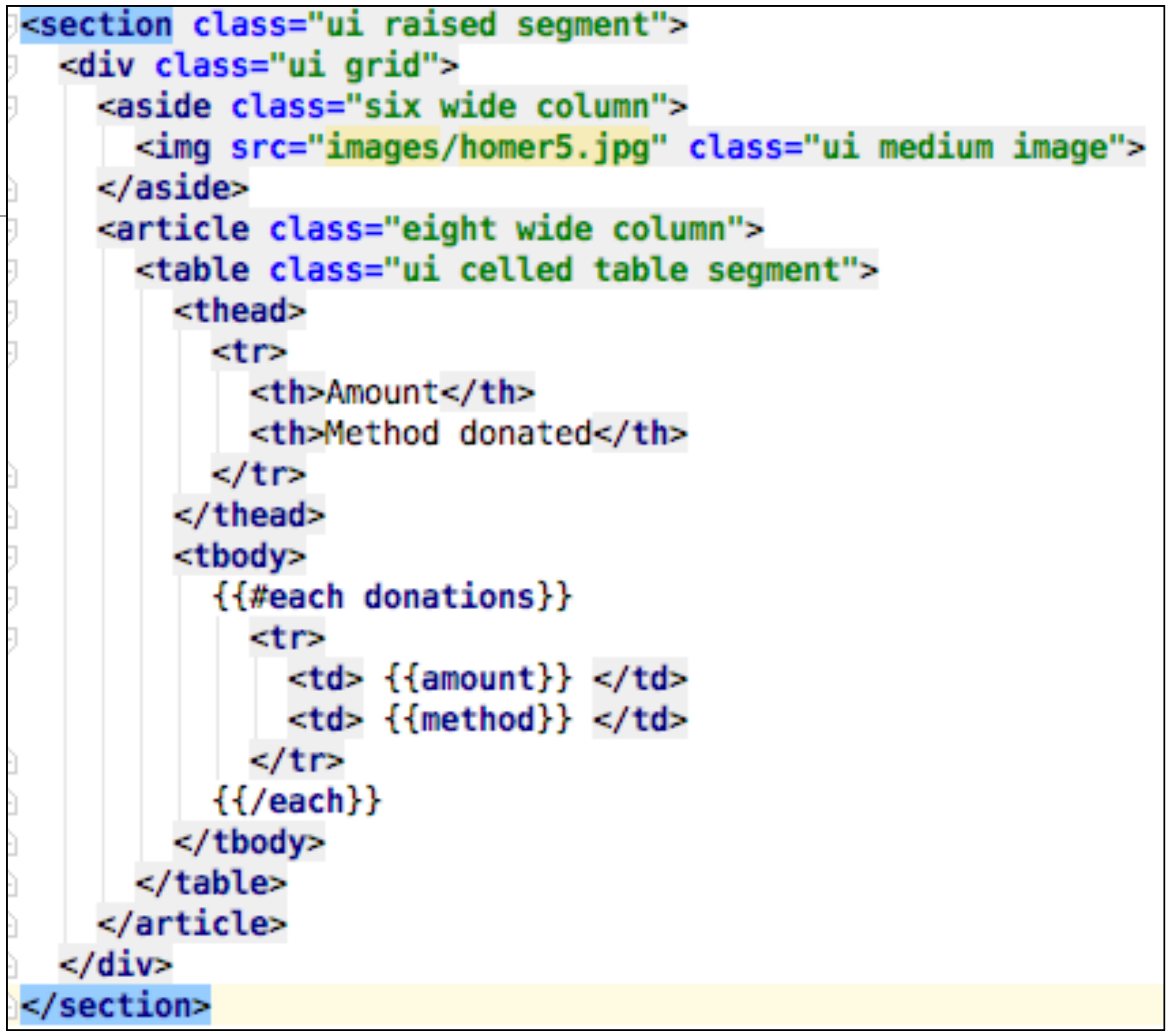
- Handlebars partials allow for code reuse by creating shared templates.
- Calling the partial is done through the partial call syntax:



```
{{> myPartial }}
```

- Will render the partial named myPartial. When the partial executes, it will be run under the current execution context.

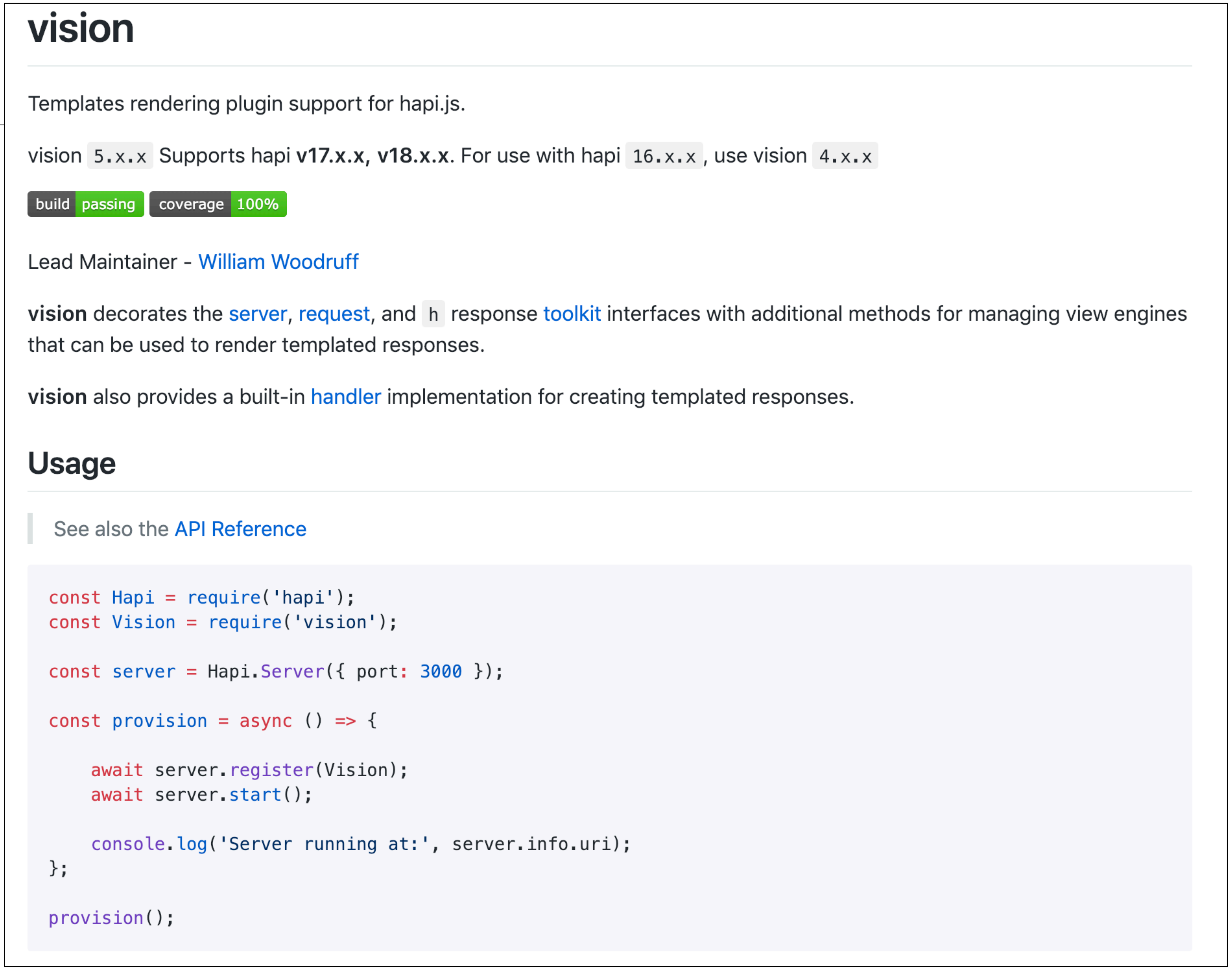
myPartial.hbs



```
<section class="ui raised segment">
  <div class="ui grid">
    <aside class="six wide column">
      <img src="images/homer5.jpg" class="ui medium image">
    </aside>
    <article class="eight wide column">
      <table class="ui celled table segment">
        <thead>
          <tr>
            <th>Amount</th>
            <th>Method donated</th>
          </tr>
        </thead>
        <tbody>
          {{#each donations}}
            <tr>
              <td> {{amount}} </td>
              <td> {{method}} </td>
            </tr>
          {{/each}}
        </tbody>
      </table>
    </article>
  </div>
</section>
```

Handlebars in Hapi

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



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`

build passing coverage 100%

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();
```

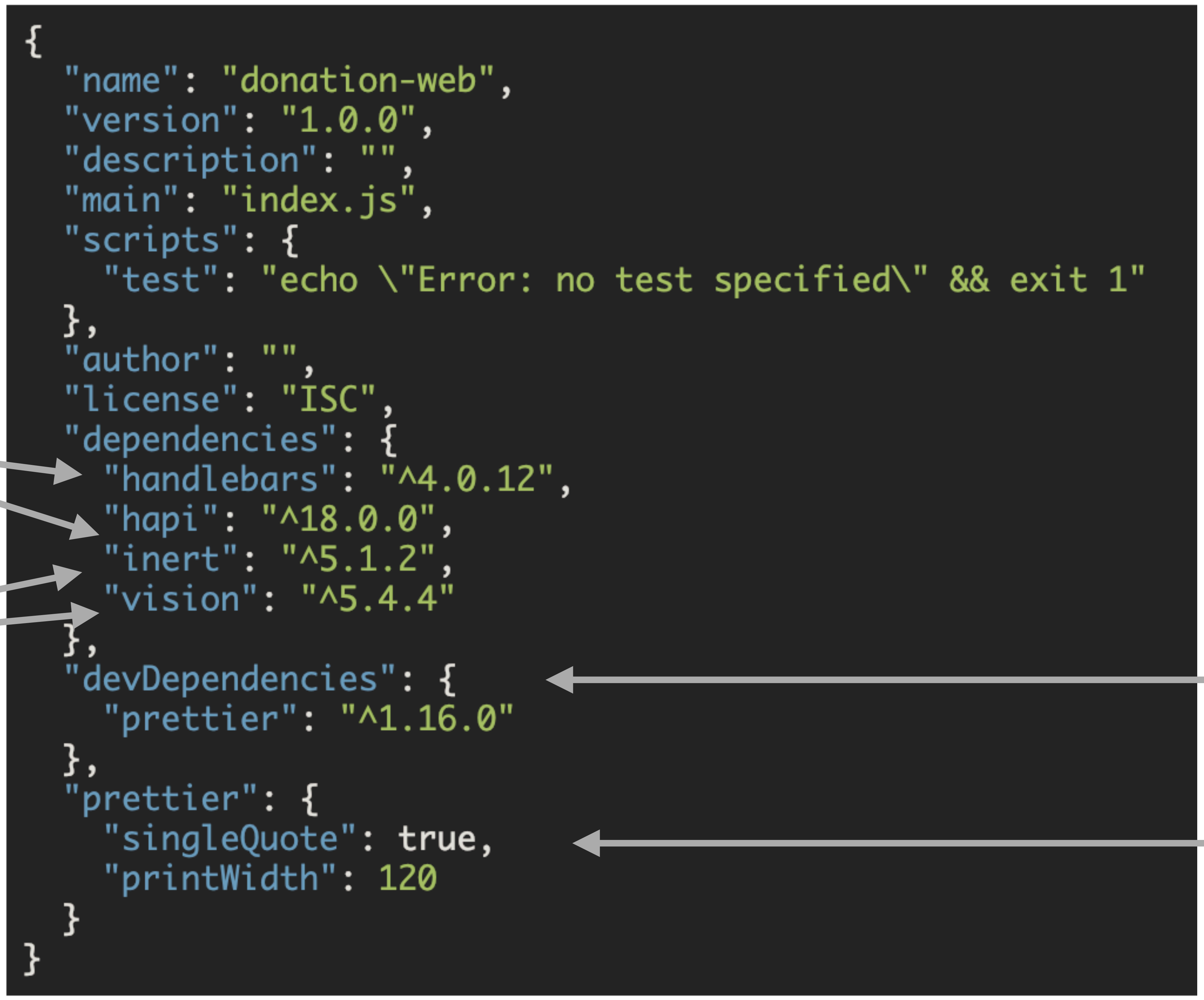
Plugin Install

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

```
npm install vision
```

```
npm install handlebars
```

package.json



```
{
  "name": "donation-web",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "handlebars": "^4.0.12",
    "hapi": "^18.0.0",
    "inert": "^5.1.2",
    "vision": "^5.4.4"
  },
  "devDependencies": {
    "prettier": "^1.16.0"
  },
  "prettier": {
    "singleQuote": true,
    "printWidth": 120
  }
}
```

general purpose node modules

Hapi plugin node modules

modules used in development only

Additional settings for specific module

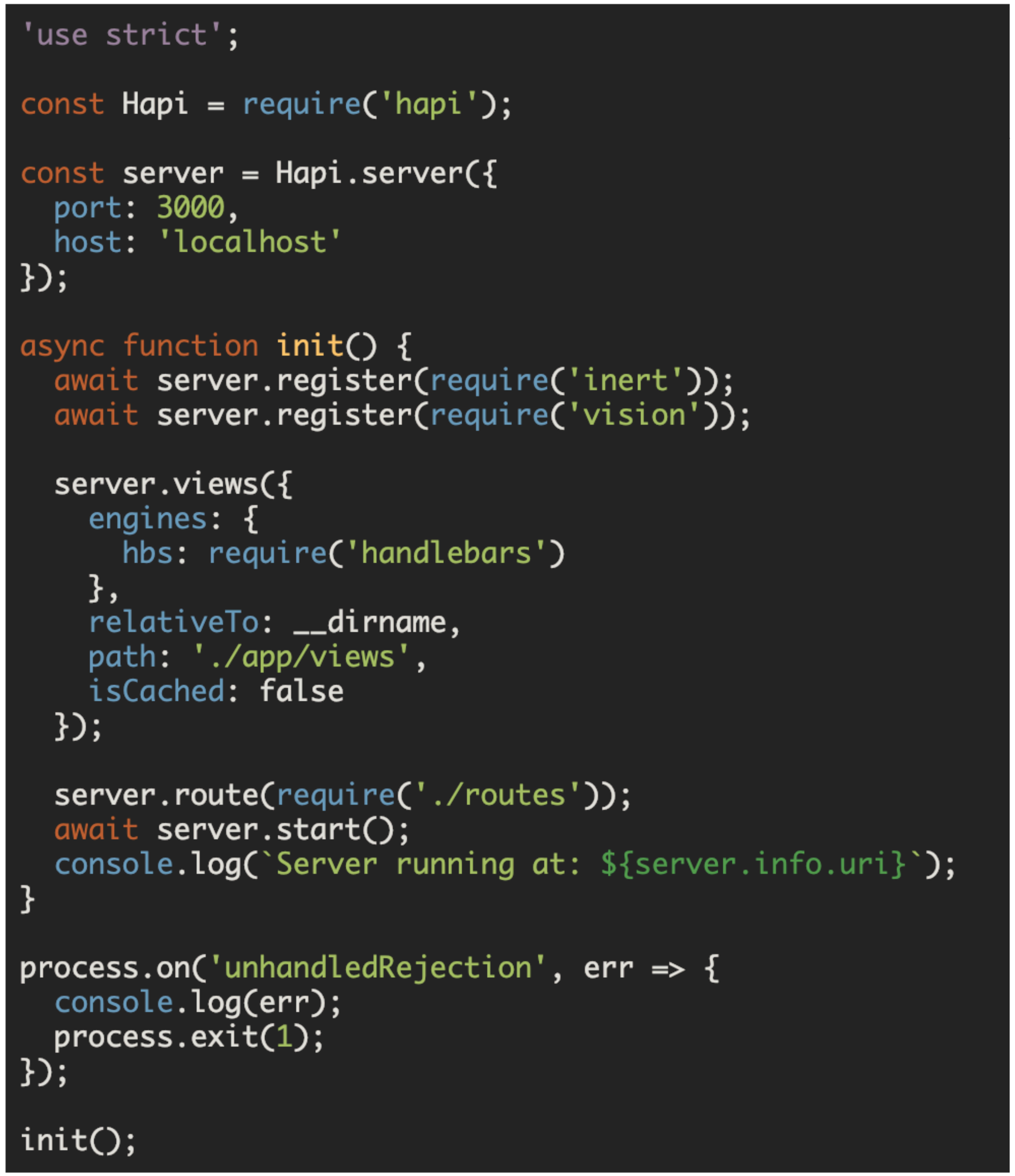
Register the Plugin

Import & register the plugin

Initialise to use Handlebars engine

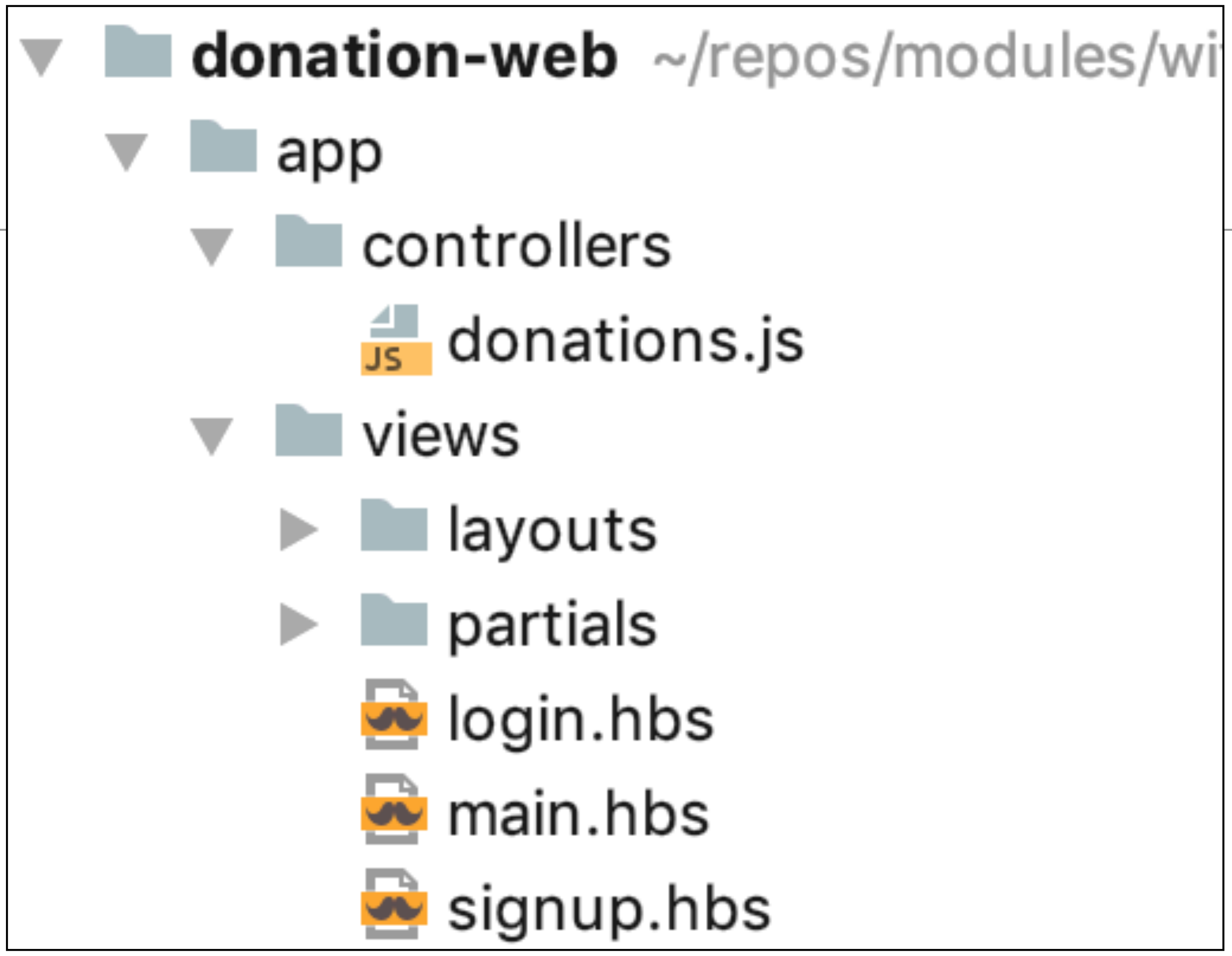
Define template locations and cache settings

index.js



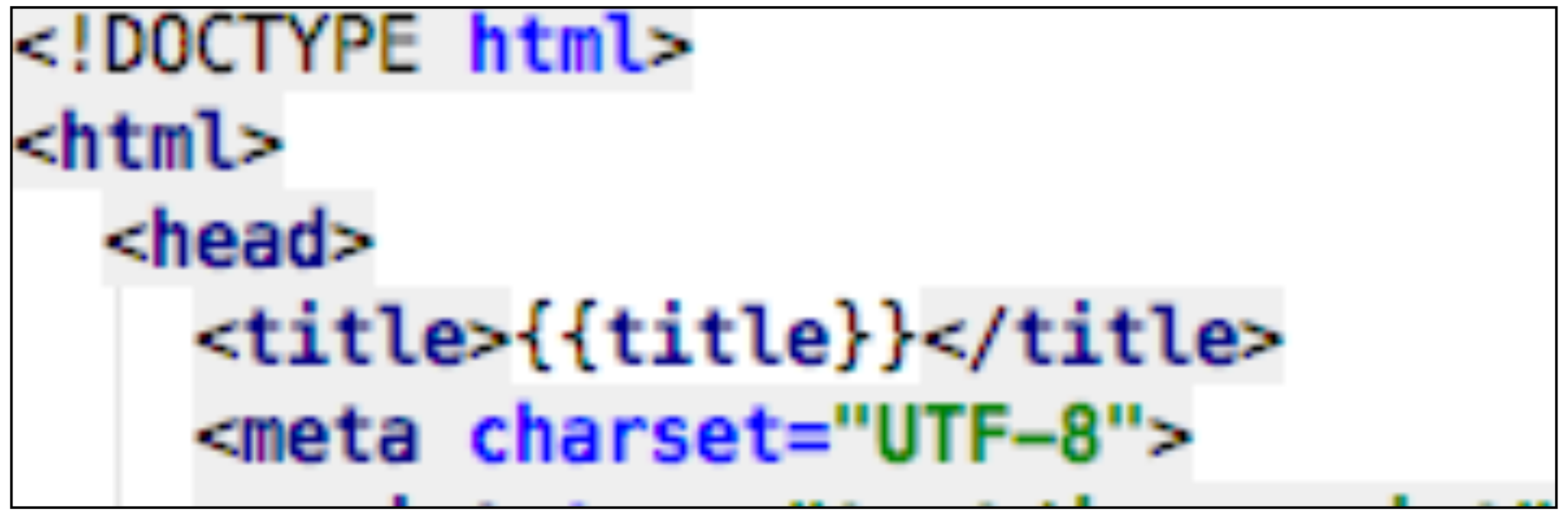
```
'use strict';

const Hapi = require('hapi');

const server = Hapi.server({
  port: 3000,
  host: 'localhost'
});

async function init() {
  await server.register(require('inert'));
  await server.register(require('vision'));

  server.views({
    engines: {
      hbs: require('handlebars')
    },
    relativeTo: __dirname,
    path: './app/views',
    isCached: false
  });

  server.route(require('./routes'));
  await server.start();
  console.log(`Server running at: ${server.info.uri}`);
}

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

init();
```

Rename views to ‘.hbs’



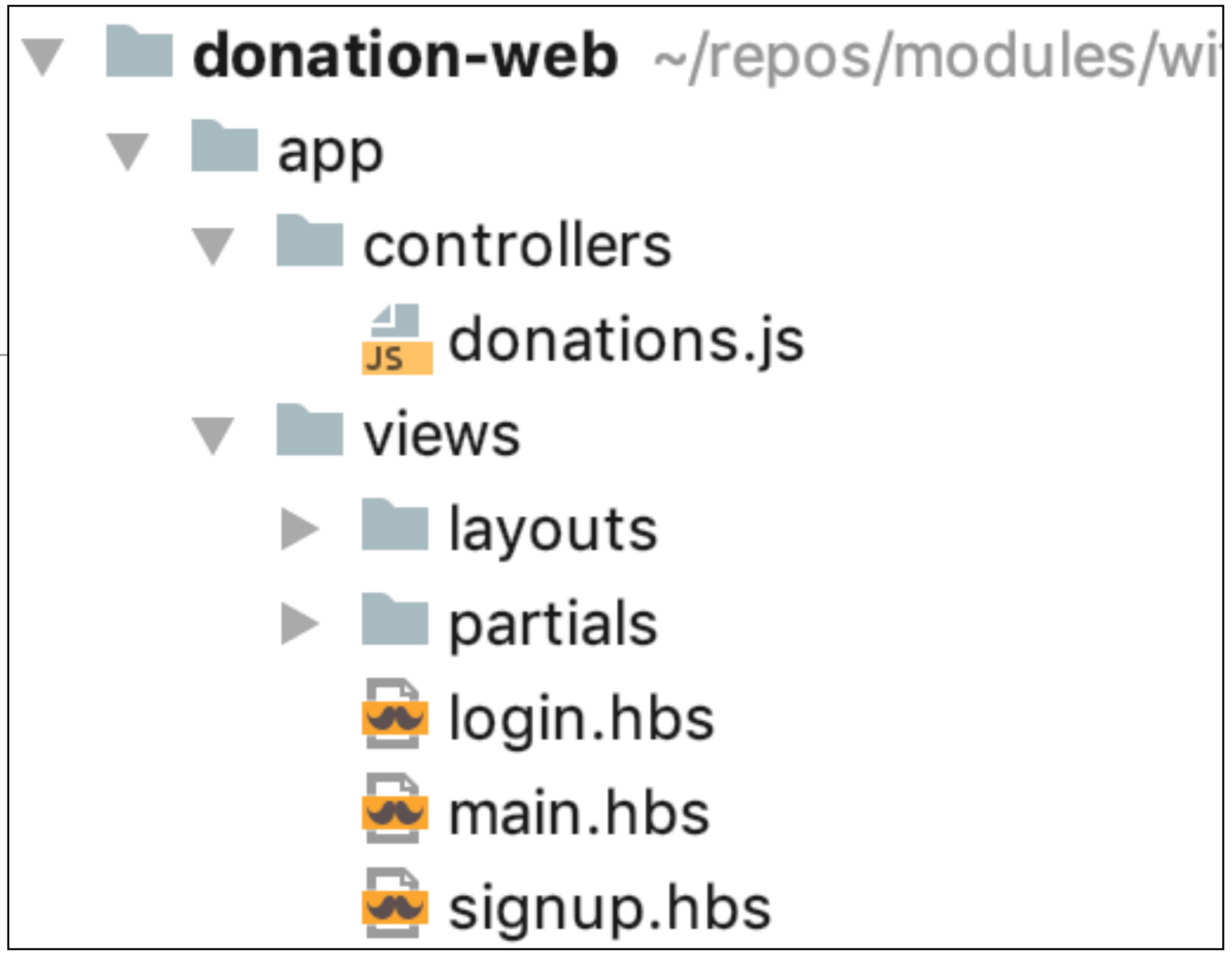
```
donation-web ~/repos/modules/wi
  app
    controllers
      donations.js
    views
      layouts
      partials
      login.hbs
      main.hbs
      signup.hbs
```

- View can now add new handlebars expressions, which will be interpolated if the correct context is provided



```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <meta charset="UTF-8">
```

Rendering the Context



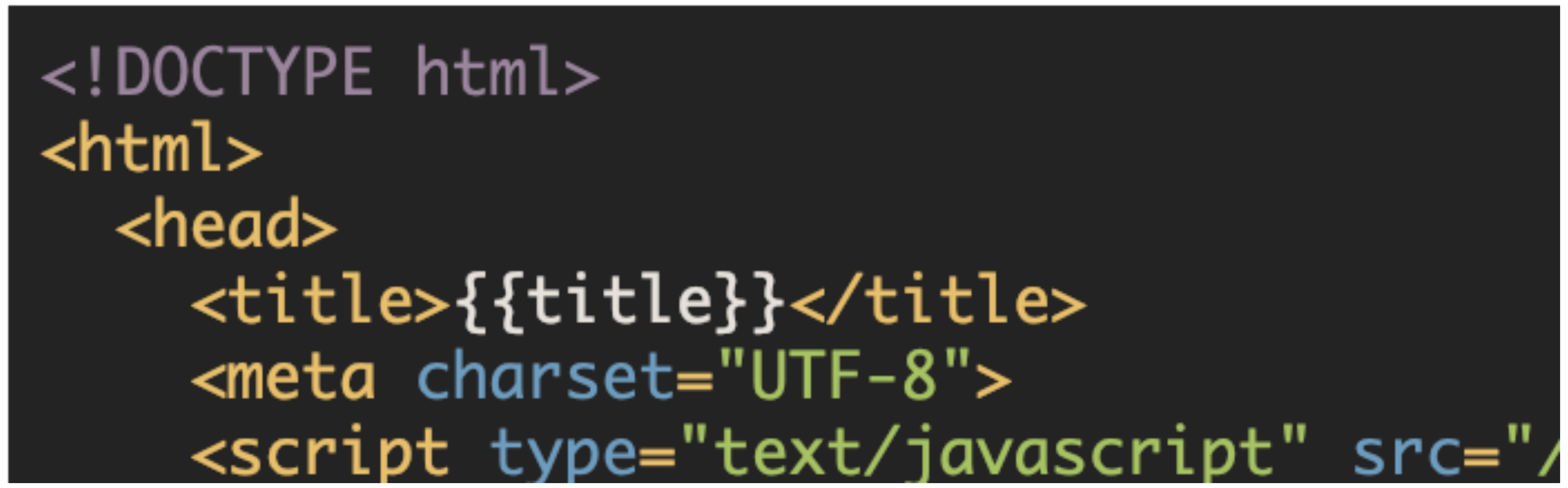
```
donation-web ~/repos/modules/wi
  app
    controllers
      donations.js
    views
      layouts
      partials
      login.hbs
      main.hbs
      signup.hbs
```

- view function accepts a ‘context’ object

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

- ‘main’ template loaded

- handlebars expressions retrieve information from the ‘context’



```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <meta charset="UTF-8">
    <script type="text/javascript" src="/
```

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

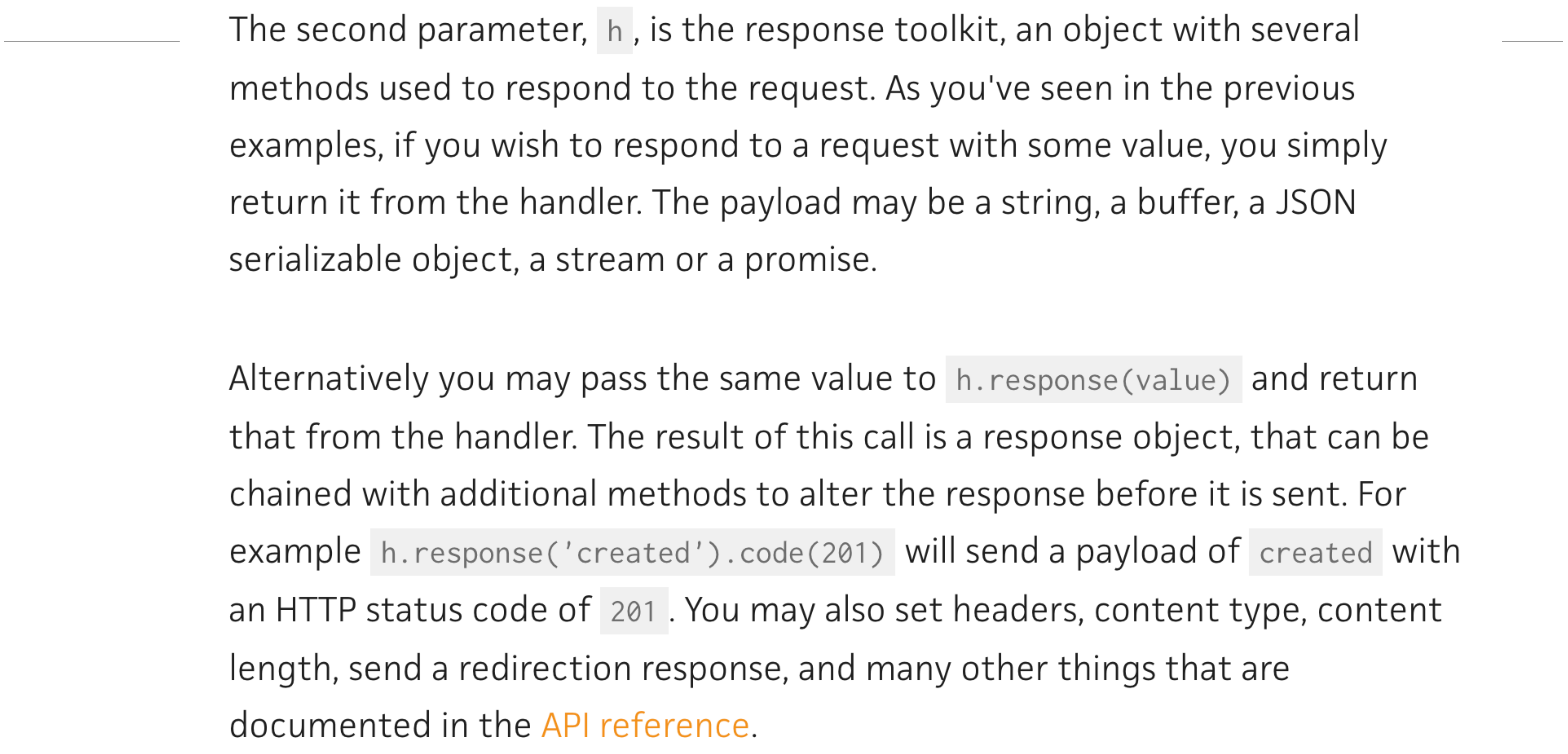
Handler method

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://hapijs.com/tutorials/routing

```
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://hapijs.com/tutorials/routing

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

```
...
server.views({
  engines: {
    hbs: require('handlebars'),
  },
  relativeTo: __dirname,
  path: './app/views',
  layoutPath: './app/views/layout',
  partialsPath: './app/views/partials',
  layout: true,
  isCached: false,
});
...
```

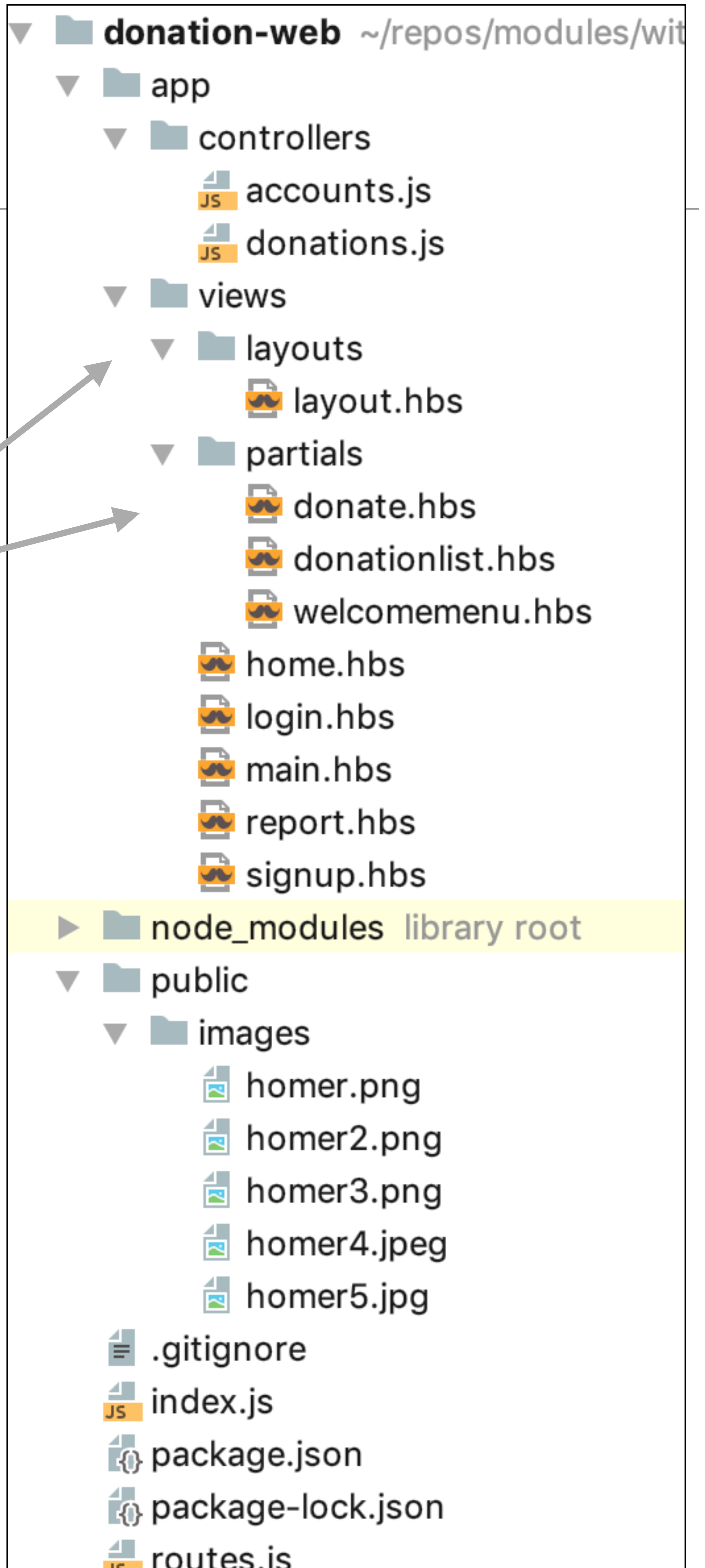
partials & layouts directories in project

Revised Project Layout

```
...
server.views({
  engines: {
    hbs: require('handlebars'),
  },
  relativeTo: __dirname,
  path: './app/views',
  layoutPath: './app/views/layout',
  partialsPath: './app/views/partials',
  layout: true,
  isCached: false,
});
...
```

partials & layouts directories in project

- Templates can now assume these folders part of the rendering pipeline



```
▼ donation-web ~/repos/modules/wit
  ▼ app
    ▼ controllers
        accounts.js
        donations.js
    ▼ views
      ▼ layouts
          layout.hbs
      ▼ partials
          donate.hbs
          donationlist.hbs
          welcomemenu.hbs
        home.hbs
        login.hbs
        main.hbs
        report.hbs
        signup.hbs
  ▶ node_modules library root
  ▼ public
    ▼ images
        homer.png
        homer2.png
        homer3.png
        homer4.jpeg
        homer5.jpg
    .gitignore
    index.js
    package.json
    package-lock.json
    routes.js
```

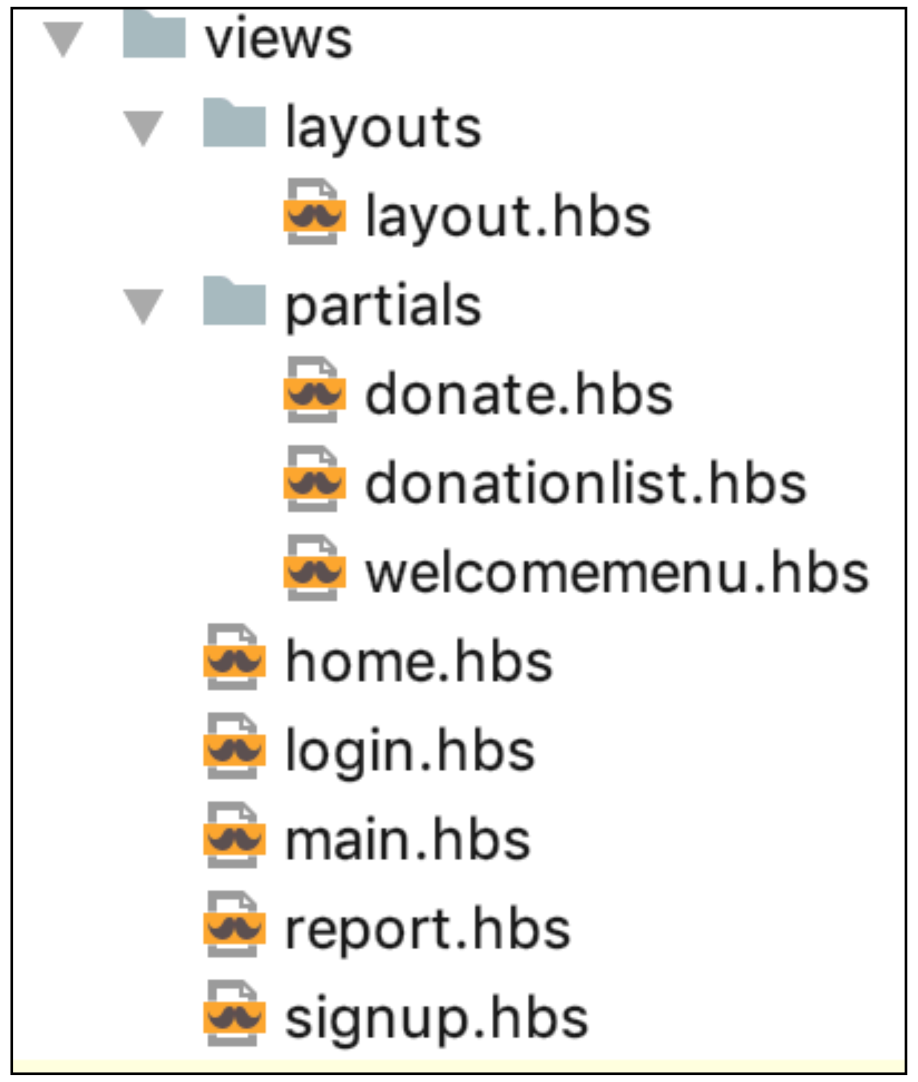
Layouts & Partials in Action

layout.hbs

```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <meta charset="UTF-8">
    ...include stylesheets
  </head>
  <body>
    <section class="ui container">
      {{{content}}}
    </section>
    <script>
  </body>
</html>
```

welcomemenu.hbs

```
<nav class="ui inverted menu">
  <header class="header item"> <a href="/"> Donation </a> </header>
  <div class="right menu">
    <a class="item" href="/signup"> Signup</a>
    <a class="item" href="/login">  Login</a>
  </div>
</nav>
```



```
views
  layouts
    layout.hbs
  partials
    donate.hbs
    donationlist.hbs
    welcomemenu.hbs
  home.hbs
  login.hbs
  main.hbs
  report.hbs
  signup.hbs
```

main.hbs

```
{{> welcomemenu }}

<section class="ui stacked segment">
  <div class="ui grid">
    <aside class="six wide column">
      <img src="images/homer.png" class="ui medium image">
    </aside>
    <article class="ten wide column">
      <header class="ui  header"> Help Me Run Springfield</header>
      <p> Donate what you can now - No Bitcoins accepted! </p>
    </article>
  </div>
</section>
```

- main content is based on layout - replacing {{content}} expression
- welcomemenu is injected into main to provide menu

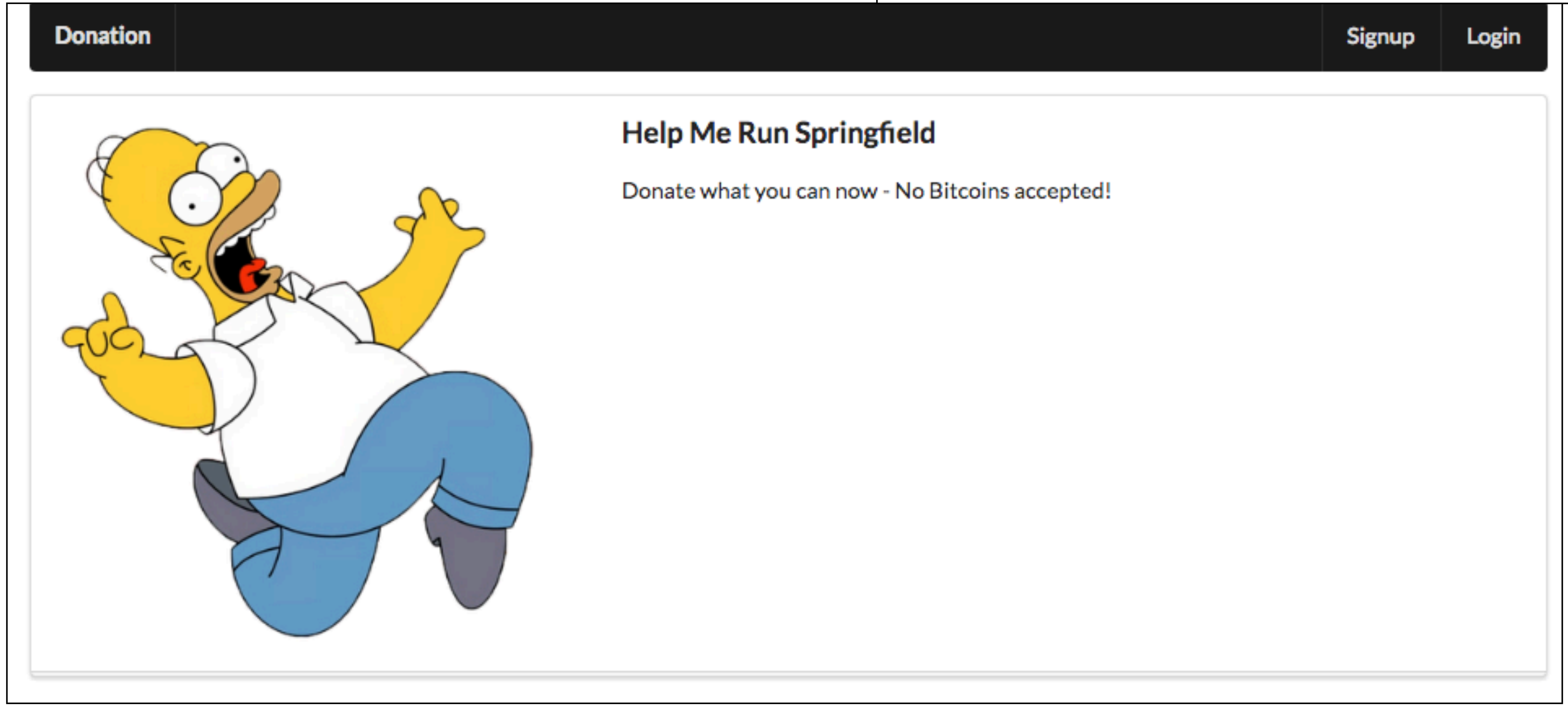
```
<!DOCTYPE html>
<html>
  <head>
    <title>{{title}}</title>
    <meta charset="UTF-8">
    ...include stylesheets
  </head>
  <body>
    <section class="ui container">
      {{{content}}}
    </section>
    <script>
  </body>
</html>
```

layout.hbs

```
<nav class="ui inverted menu">
  <header class="header item"> <a href="/"> Donation </a> </header>
  <div class="right menu">
    <a class="item" href="/signup"> Signup</a>
    <a class="item" href="/login">  Login</a>
  </div>
</nav>
```

welcomemenu.hbs

```
{{> welcomemenu }}

<section class="ui stacked segment">
  <div class="ui grid">
    <aside class="six wide column">
      <img src="images/homer.png" class="ui medium image">
    </aside>
    <article class="ten wide column">
      <header class="ui  header"> Help Me Run Springfield</header>
      <p> Donate what you can now - No Bitcoins accepted! </p>
    </article>
  </div>
</section>
```

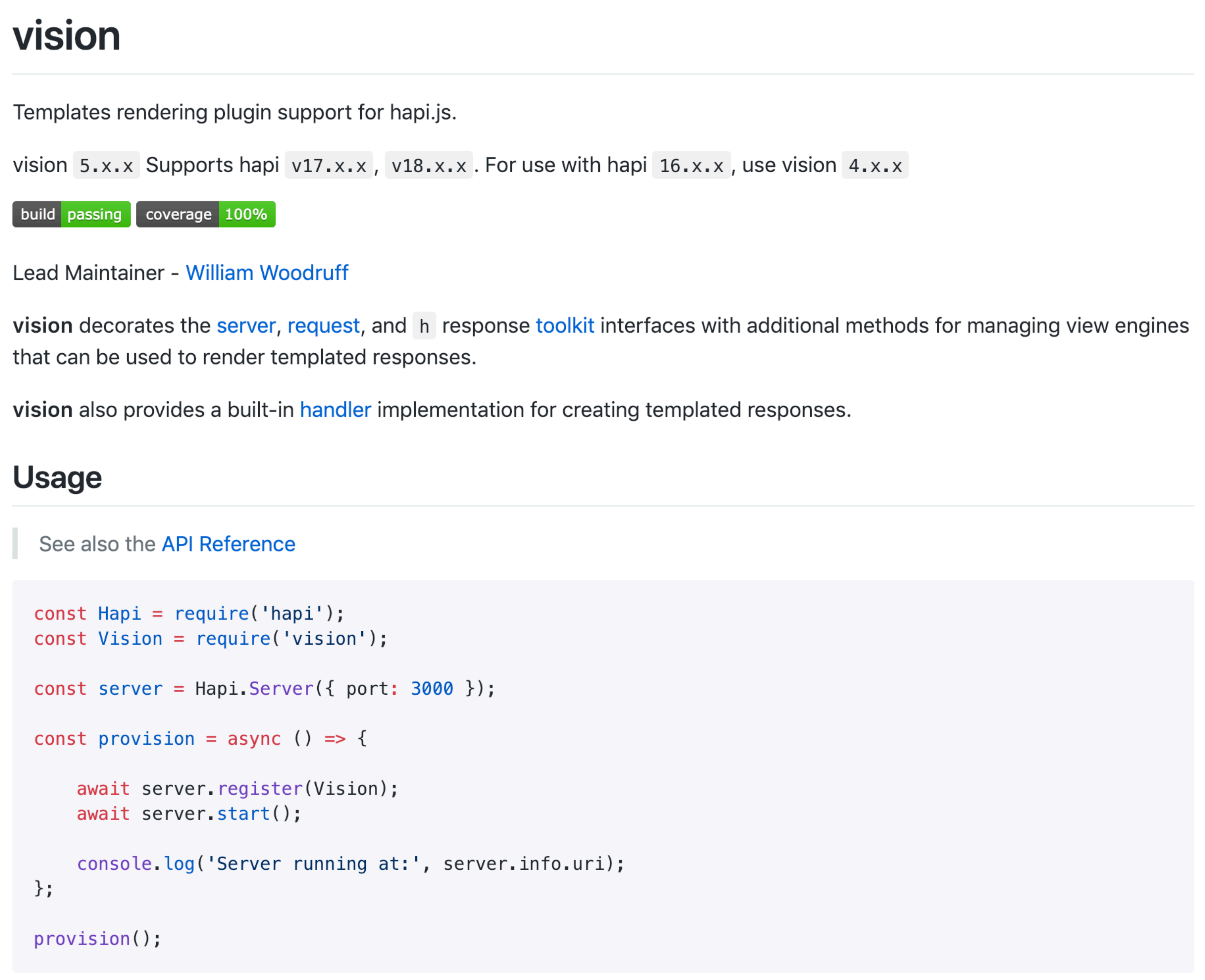


Help Me Run Springfield

Donate what you can now - No Bitcoins accepted!

main.hbs

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

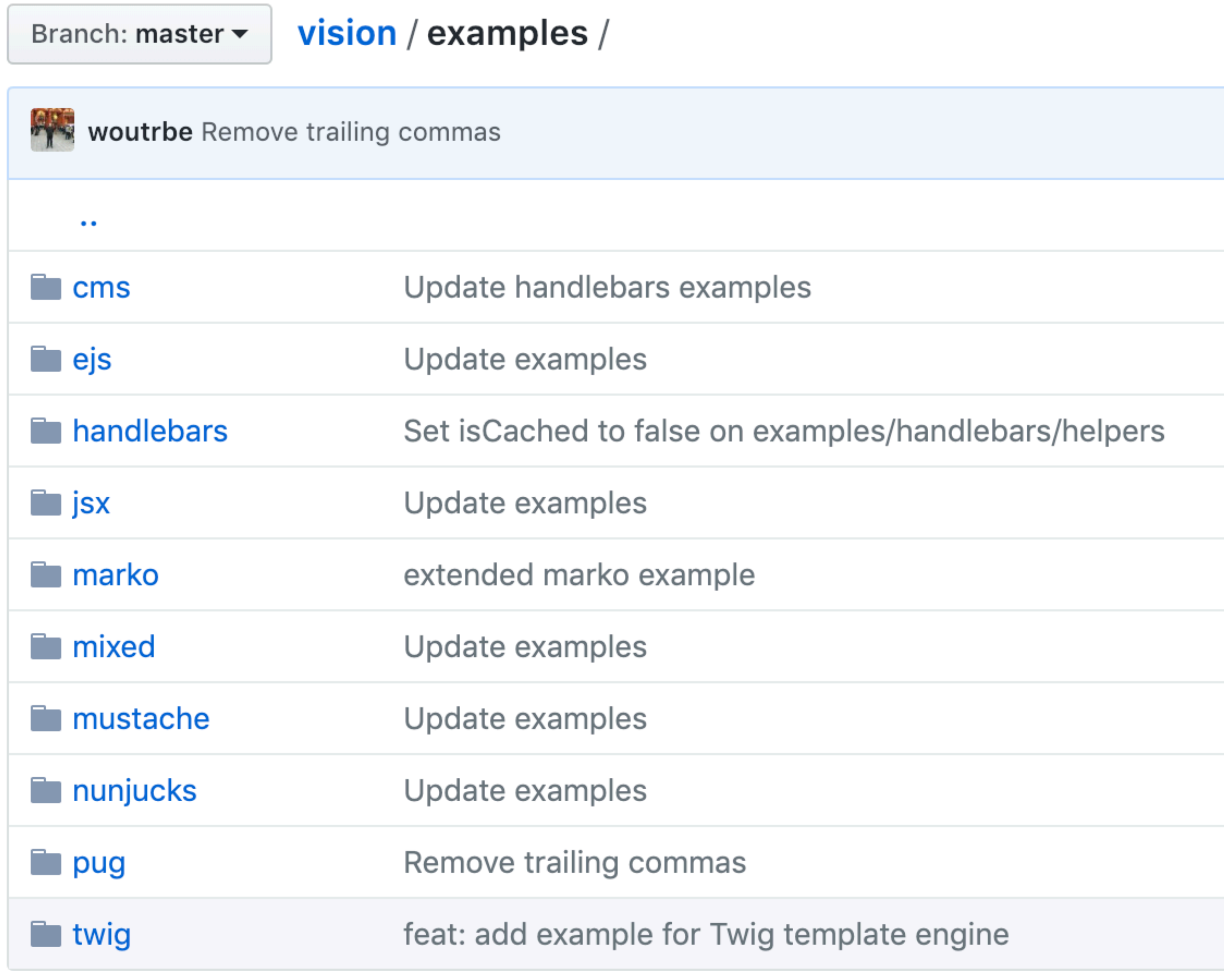
vision decorates the server, request, and `h` response toolkit interfaces with additional methods for managing view engines that can be used to render templated responses.

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 /

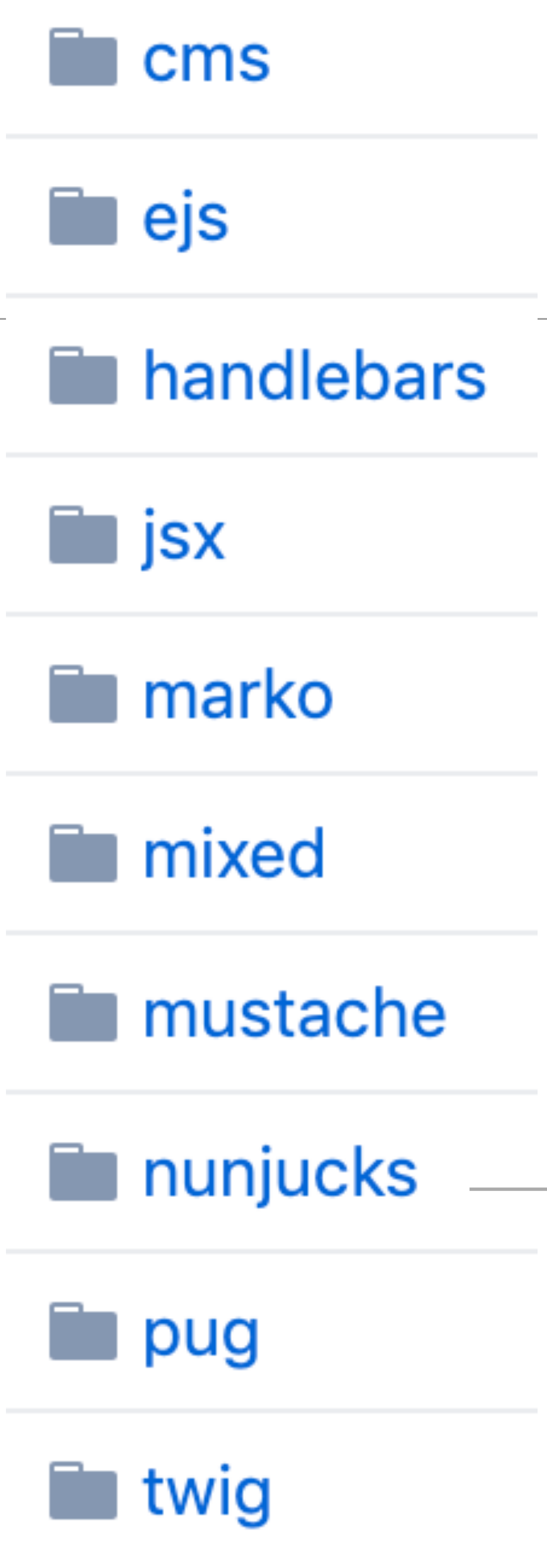
woutrbe Remove trailing commas

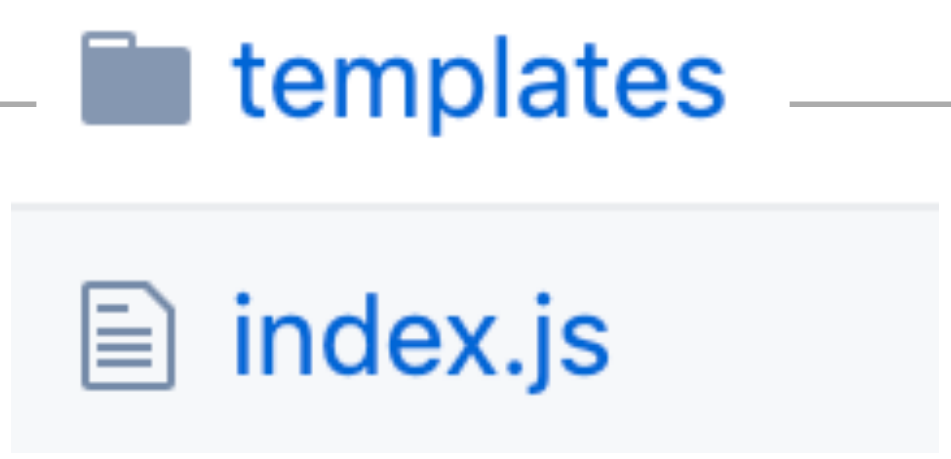
..

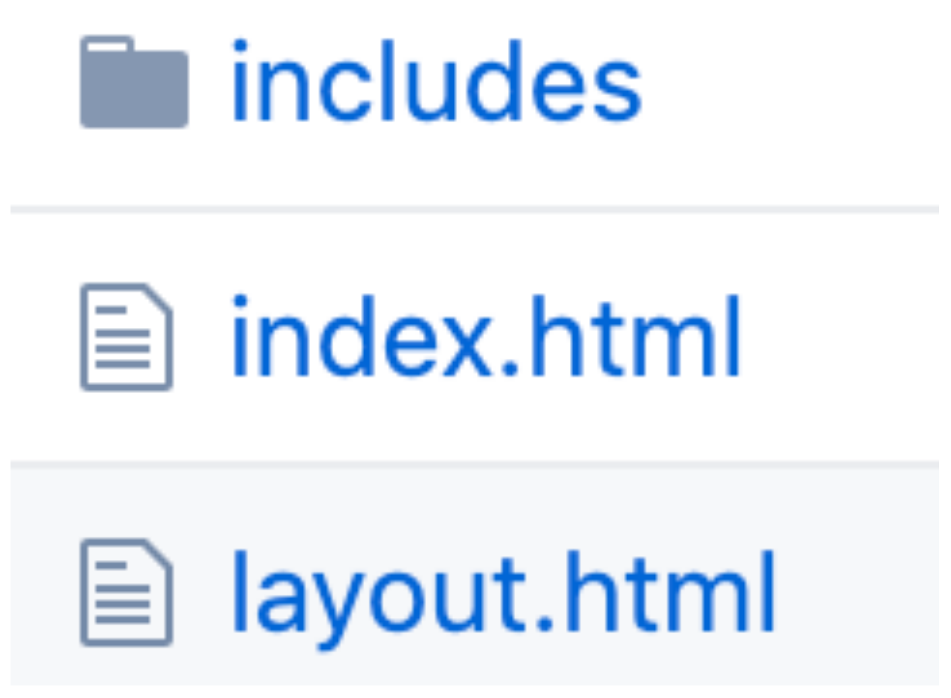
cms	Update handlebars examples
ejs	Update examples
handlebars	Set isCached to false on examples/handlebars/helpers
jsx	Update examples
marko	extended marko example
mixed	Update examples
mustache	Update examples
nunjucks	Update examples
pug	Remove trailing commas
twig	feat: add example for Twig template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

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

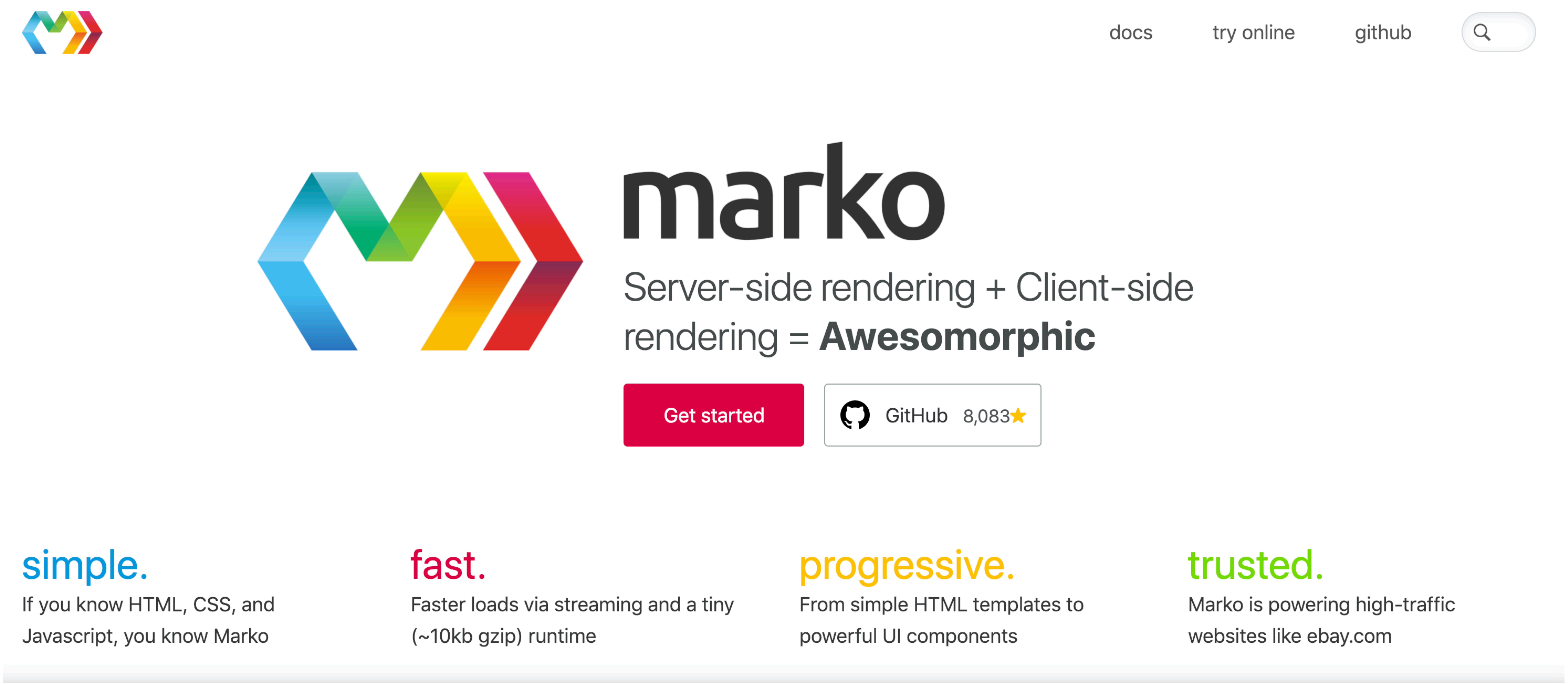
Nunchucks


cms
ejs
handlebars
jsx
marko
mixed
mustache
nunjucks
pug
twig


templates
index.js


includes
index.html
layout.html

https://markojs.com/


docs try online github

marko

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

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

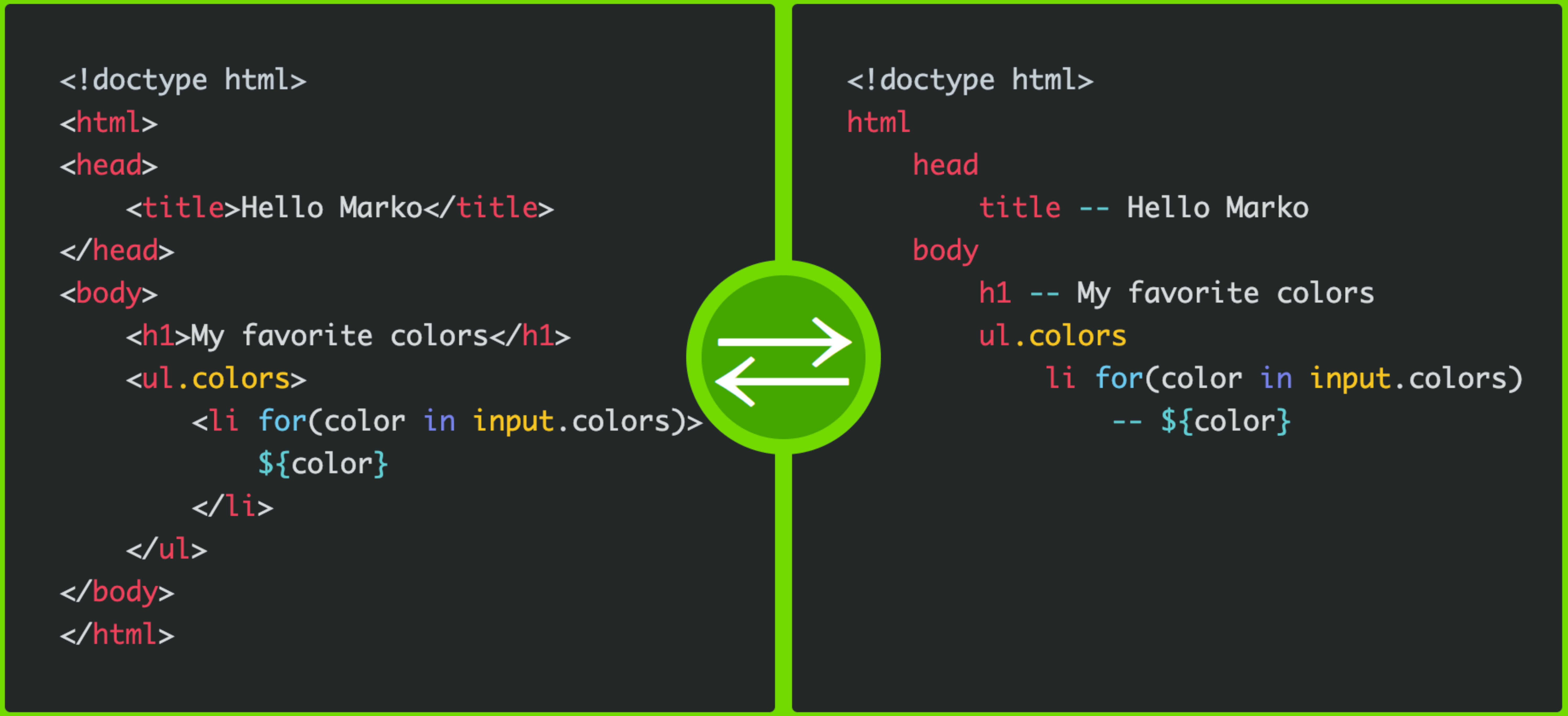
fast.
Faster loads via streaming and a tiny (~10kb gzip) runtime

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}
```