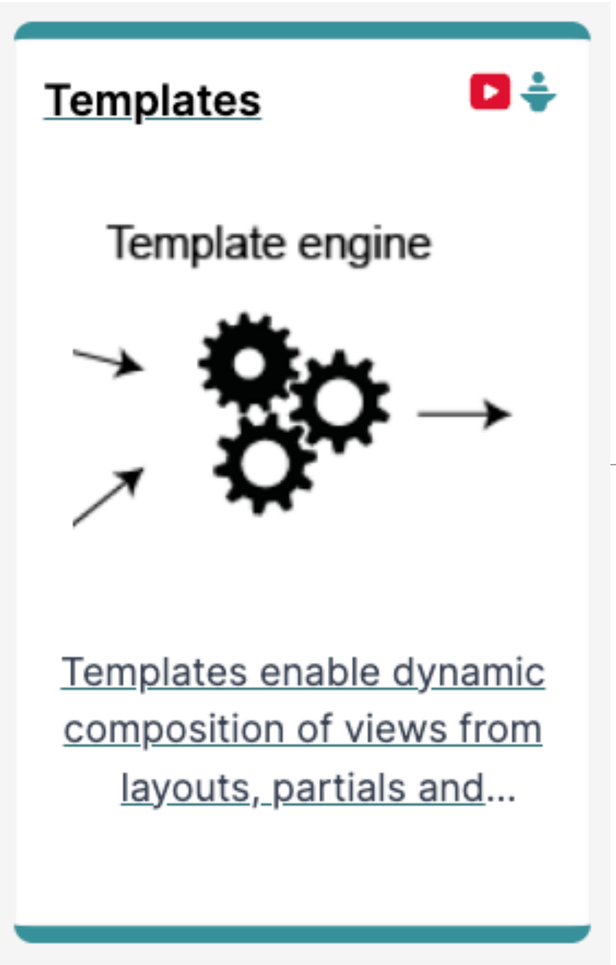
Templates


Templates
Template engine
Templates enable dynamic composition of views from layouts, partials and...

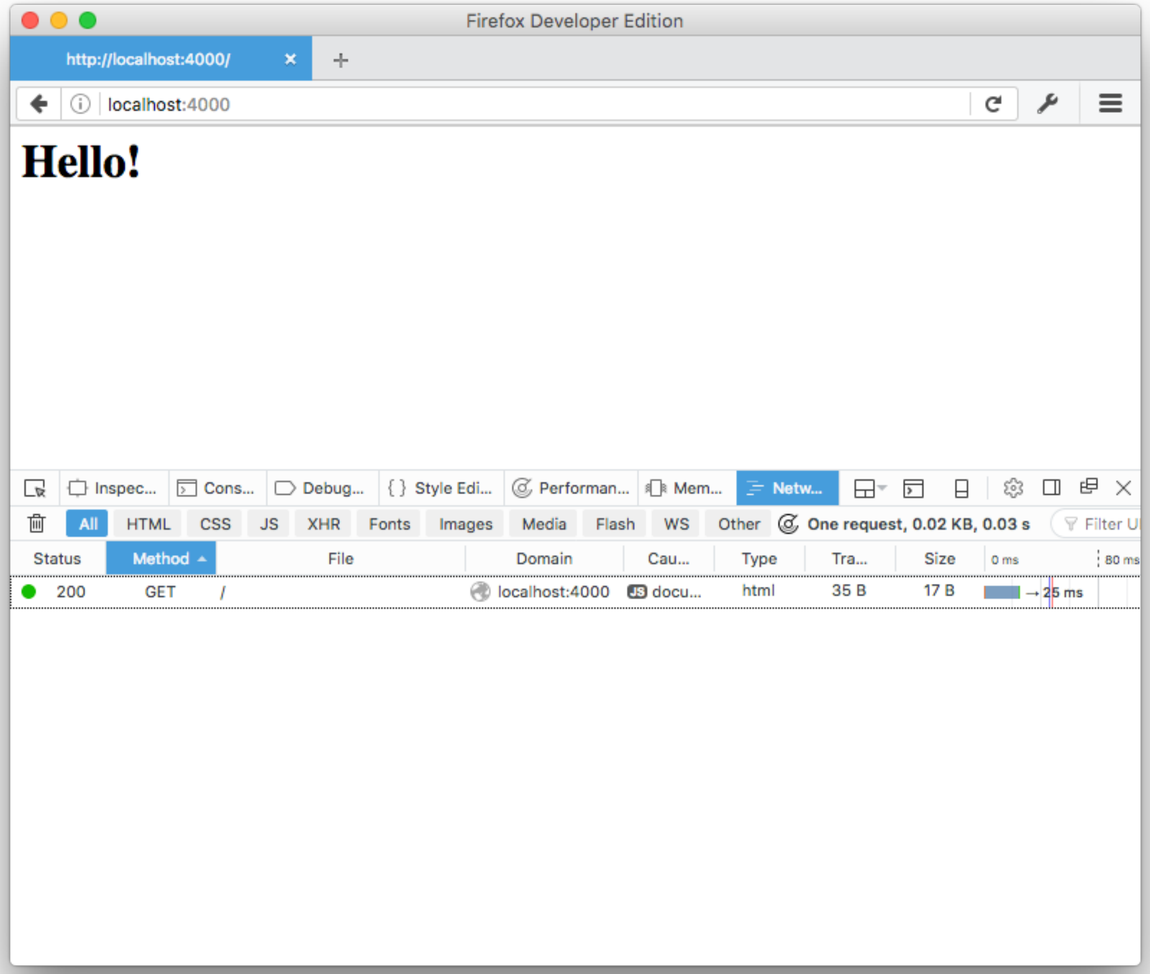
response.send

- In order to render web pages we could pass html content

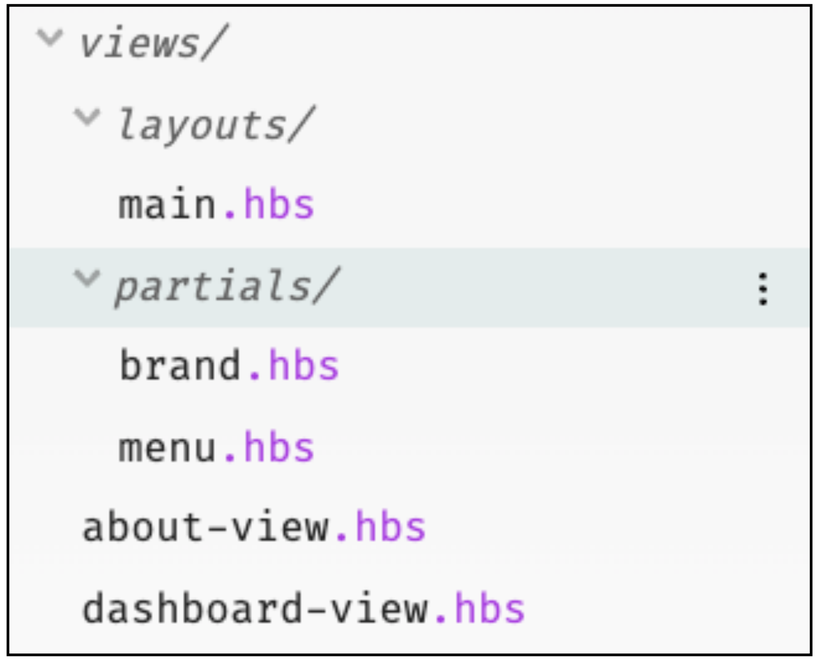
- This would become very unwieldy and unmaintainable

start.js

```
const start = {
  index(request, response) {
    logger.info('start rendering');
    response.send('<h1> Hello </h1>');
  },
};
```



Front-end



- All written in HTML + handlebars
- Handlebars: Templating language
- Similar to the templating we've seen, it supports:
  - **Layouts**
  - **Partials**
  - **Views**
- These are very similar to Eleventy equivalents that we worked with nunjucks

Partials & Layouts

- Partials & Layouts play a prominent role in enabling DRY (Dont Repeat Yourself) principles
  - Layouts: Reusable Page Structure
  - Partials: Reusable templates

```
<!DOCTYPE html>
<html lang="en-IE">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device
  <title>{{title}}</title>
  <link rel="stylesheet" href="https://cdn.js
</head>

<body>
  <div class="container">
    {{{body}}}
  </div>
</body>

</html>
```

```
{{> menu active="dashboard"}}

<section class="box has-text-centered">
  <h3 class="is-3 title">
    Main Dashboard
  </h3>
  <div class="section">
    <p>
      This is the main dashboard view
    </p>
  </div>
</section>
```

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

```
<nav class="navbar mb-6">
  <div class="navbar-brand">
    {{> brand}}
  </div>
  <div class="navbar-menu" id="navMenu">
    <div class="navbar-end">
      <div class="navbar-item">
        <div class="buttons">
          <a id="dashboard" class="button" href="/dashboard"> Dashboard </a>
          <a id="about" class="button" href="/about"> About </a>
        </div>
      </div>
    </div>
  </div>
</nav>

<script>
  document.getElementById("{{active}}").classList.add("is-primary");
</script>
```

Layout

- All views will be based on structure laid down in **main.hbs.**
- Includes Bulma CSS library
- View content will be inserted into {{{body}}}

```
<!DOCTYPE html>
<html lang="en-IE">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device
  <title>{{title}}</title>
  <link rel="stylesheet" href="https://cdn.js
</head>

<body>
  <div class="container">
    {{{body}}}
  </div>
</body>

</html>
```

Template Expressions

- In addition to layouts + partials, tempting also support **Template Expressions**
- These expressions enable external information to be incorporated into a page.
- This information will be delivered via Javascript Objects


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

Tempting Engine

Context

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

Template

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

Template engine

Rendered HTML

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

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!"};
```

- 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>
```

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>
```

About Controller -> About View

about-controller.js

```
export const aboutController = {
  index(request, response) {
    const viewData = {
      title: "About This Application",
    };
    console.log("about rendering");
    response.render("about-view", viewData);
  },
};
```

about-view.hbs

```
{{> menu active="about"}}

<section class="box is-3 has-text-centered">
  <h3 class="title">
    Template Application
  </h3>
  <div class="section">
    Description of the app here...
  </div>
</section>
```

- **response.render** locates the named template and sends it to the browser

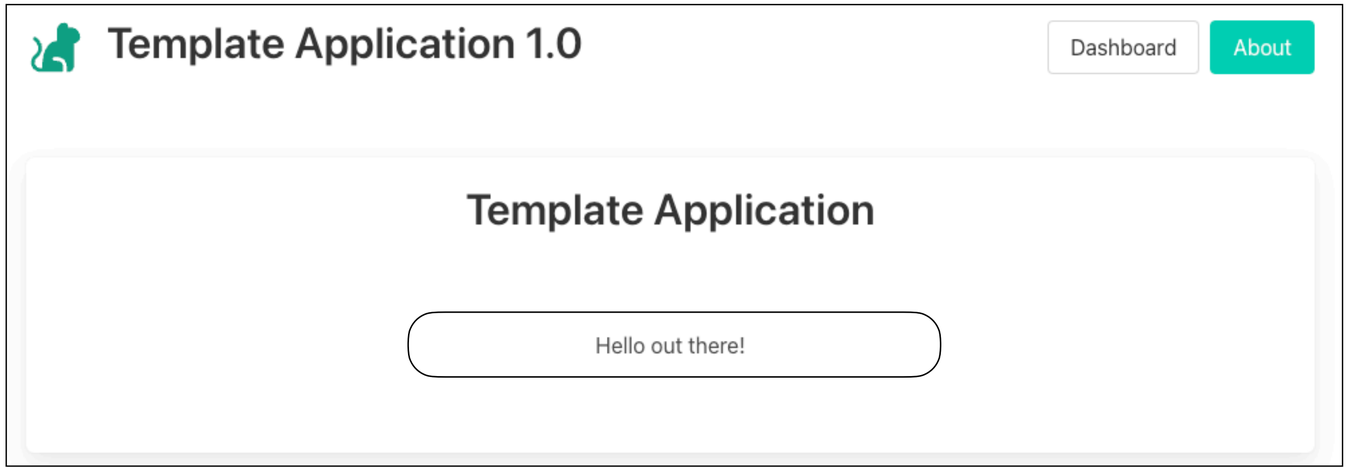
- It also passes the **viewData** object to the to the view

- The View may or may not use the data in this object (not used in above example)

About Controller -> About View

about-controller.js

```
export const aboutController = {
  index(request, response) {
    const viewData = {
      title: "About This Application",
      greeting: "Hello out there!"
    };
    console.log("about rendering");
    response.render("about-view", viewData);
  },
};
```



- We can pass simple and complex data to the views

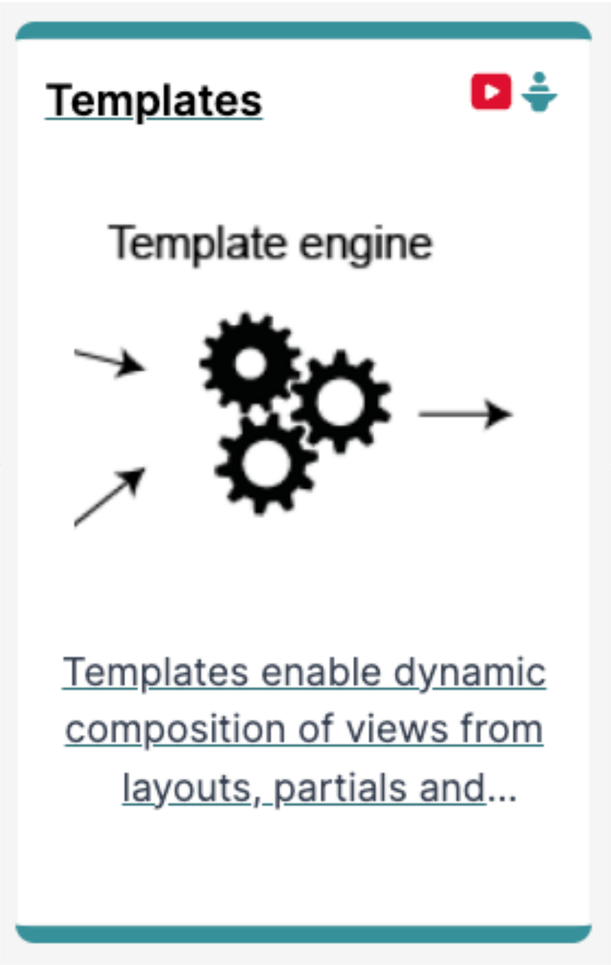
about-view.hbs

```
{{> menu active="about"}}

<section class="box is-3 has-text-centered">
  <h3 class="title">
    Template Application
  </h3>
  <div class="section">
    {{greeting}}
  </div>
</section>
```

- {{greeting}} replaced with the value in the viewData object called ‘greeting’

Templates


Templates
Template engine
Templates enable dynamic composition of views from layouts, partials and...