Templating with Handlebars

Learning Objectives

- Understand the difference between server and client side HTML rendering
- Understand what handlebars is used for
- Understand how to configure handlebars with express

Material

Server-Side vs Client-Side Rendering

When we visit a website like our Twitter homepage, what we're looking at is a GUI representation of an HTML file. For static websites, such as a website for a local business, this HTML file may simply be stored in its complete form on a web server and sent as is when our browser requests it. However, for a website such as Twitter, this page needs to be constructed, or *rendered*, on the fly with the specific data for the user making the request, for example, including the HTML to display the tweets made recently by users they follow.

This bespoke HTML page can be rendered either on the server or on the client.

Server-Side Rendering

For server-side rendering, when we make a request for a webpage from our server, it performs any logic it needs to (e.g. look our user up in the database) then builds a complete, bespoke HTML page and sends it to the client to be displayed by the browser.

Client-Side Rendering

With client-side rendering, the HTML is instead constructed within the user's browser by JavaScript which has also been sent by the server. The server will usually send the client the raw data it needs to build the page (for example, it might send a JSON array of tweets). The JavaScript running in the browser will then loop through this data and create HTML elements to add to the page.

There are advantages and disadvantages to both approaches. Server-side rendered pages are argued to be more performant and SEO-friendly, but modern JavaScript frameworks such as React tend to use client-side rendering by default.

Handlebars

Handlebars is a templating engine.

If we start with data like this:

```
{
    "restaurants": [
        {
            "name": "Theo's Pizza",
            "image": "http://theos.image.jpg"
        },
        {
            "name": "Patsies Pastries",
            "image": "http://patsies.image.jpg"
        }
    ]
}
```

We have an array of data here that we want to render in HTML. The way of doing this with Handlebars is to create a template that will enable us to iterate over our array of data and repeatedly add the same parsed HTML block to our HTML. This is how dynamic lists of content are rendered.

```
<section>
```

```
    {{#each restaurants}}
        <article>
            <img src="{{this.image}}"/>
            <h2>{{this.name}}</h2>
        </article>
    {{/each}}
</section>
```

The `{{ }}` sections mark the boundaries of a piece of dynamic content that is to be slotted into that location from the data we are passing the template. The `#each` syntax functions like a for loop: producing a copy of the nested `article` for each restaurant in the array.

These handlebar templates are like functions that take an HTML template and some data, and return the completed HTML. The above example will return the following HTML.

```
<section>
    <article>
        <img src="http://theos.image.jpg"/>
        <h2>Theo's Pizza</h2>
    </article>
    <article>
        <img src="http://patsies.image.jpg"/>
        <h2>Patsies Pastries</h2>
    </article>
</section>
```

Handlebars with Express

Handlebars can be used to render HTML on both the client and the server. We're going to use Handlebars to render our templates on the **server** and send the completed HTML to the client.

First, we need to install a few dependencies:

```
npm install handlebars express-handlebars
@handlebars/allow-prototype-access
```

Next, we need to create a views directory at the base of our Express app.

```
├── app.js
└── views
    ├── home.handlebars
```

```
└── layouts
    └── main.handlebars
```

The layouts directory will contain our main HTML template that all the other templates will be plugged into:

```
<!--main.handlebars-->
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Example App</title>
</head>
<body>

    {{{body}}}

</body>
</html>
```

Our home.handlebars file will be used to render the homepage - in this case, we'll display a list of cities:

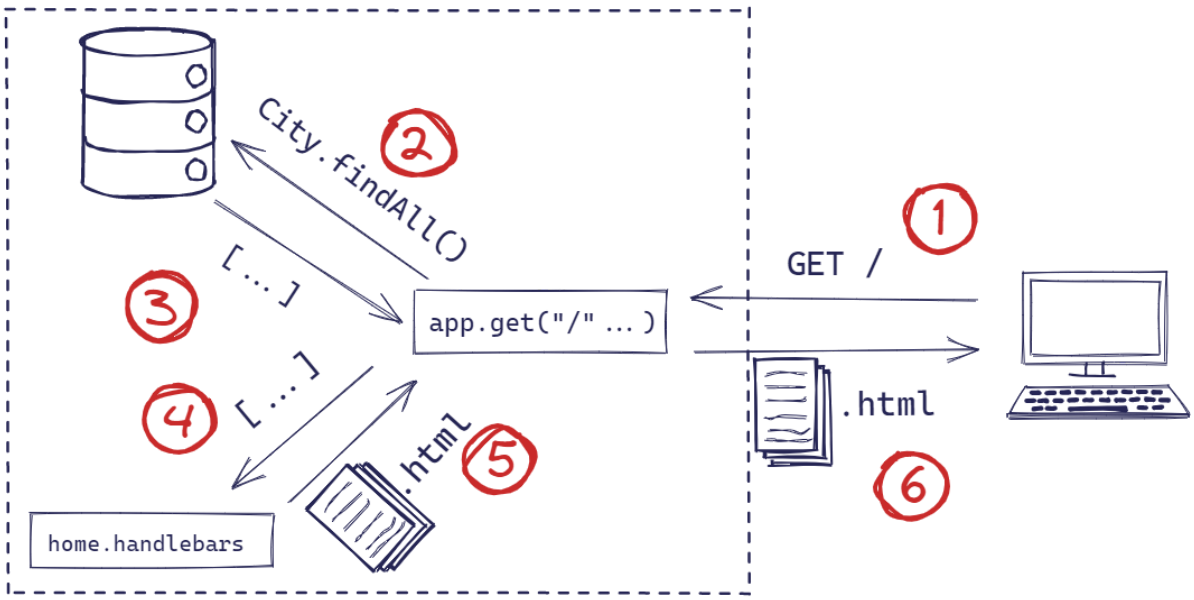
```
<!--home.handlebars-->
<h1>Cities</h1>
{{#each cities}}
    <article>
        <label>
            Name:
            <span>{{this.name}}</span>
        </label>
        <label>
            Population:
            <span>{{this.population}}</span>
        </label>
    </article>
{{/each}}
```

We then need to import our dependencies and configure Express' view engine. The setup is a bit complicated (feel free to copy and paste) but all we're doing is telling Express to use handlebars when we call `res.render`.

```
// app.js
const express = require("express");
```

```
const path = require("path");
const Handlebars = require("handlebars");
const expressHandlebars = require("express-handlebars");
const {
    allowInsecurePrototypeAccess,
} = require("@handlebars/allow-prototype-access");

const app = express();

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

// setup our templating engine
const handlebars = expressHandlebars({
    handlebars: allowInsecurePrototypeAccess(Handlebars),
});
app.engine("handlebars", handlebars);
app.set("view engine", "handlebars");
app.set('views', path.join(__dirname, 'views')

app.get("/", async (req, res) => {
    const cities = await City.findAll();
    res.render("home", { cities });
});
```

The final `app.get` is just here as an example of an endpoint which renders a template and sends the HTML back to the user. We call res.render and pass in the name of the template ("home" refers to "home.handlebars", in this case) and the data to plug into the template.



Core Assignments

Restaurant Handlebars

We're going to add a UI to our restaurant chain application. When a user visits our website (e.g. http://localhost:4000/) they should see a list of all the companies' names and logos. When a user clicks a particular company, it should open that company's page and display a list of information about the company's menus and locations.

You'll need to create a new GET endpoint for each of these two pages, and a corresponding handlebars template to render the HTML to send back to the user.

(This assignment requires the *RESTaurant API* assignment to be completed)

Extension Assignments

Link some CSS to your handlebars template to style your website.

Additional Resources

Cities GitHub Example