

Handlebars 1

Website Development 2

What are JavaScript Templates?

- Nowadays the majority of the Web consists of *dynamic applications* in which the data keeps changing frequently.
- As a result, there is a continuous need to update the data rendered on the browser.
- This is where JavaScript templating engines come to the rescue and become so useful.
- They simplify the process of manually updating the view and at the same time they improve the structure of the application by allowing developers to separate the business logic from the rest of the code.

What are JavaScript Templates?

- JavaScript templates are a method of separating HTML structure from the content contained within.
- Templating systems generally introduce some new syntax but are usually very simple to work with.
- Implementation is as simple as including our chosen library, fetching our template and rendering it alongside some data.

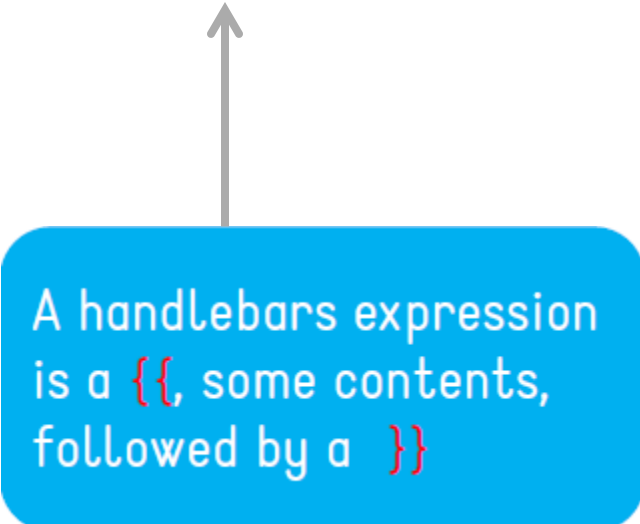
What is handlebars?

- **Handlebars** is a templating engine for JavaScript.
- It is a JavaScript library that you include in your page just as you include any other JavaScript file.
- With it, you can add templates to your HTML page that will be parsed and interpolated (values of properties inserted in place) with the values from the data (JSON object) you passed to the Handlebars function.

Example of Template

```
<script id="template" type="text/x-handlebars-template">
  <p>Use the <strong>{{power}}</strong>, {{name}}!</p>
</script>
```

```
const person = {
  name: "Luke",
  power: "force"
};
```



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

Example of Template

```
//Compile the template
let compiled_template = Handlebars.compile($('#template').html());

//Render the data into the template
let rendered = compiled_template(person);

//Overwrite the contents of #target with the rendered HTML
$('#placeholder').html(rendered);
```

Result: Use the **force**, Luke!

Template Context

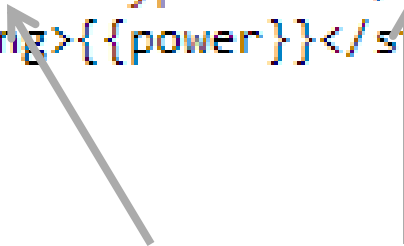
- A context is a JavaScript object used to populate a template.

```
const person = {  
  name: "Luke",  
  power: "force"  
};
```

Compiling a Template

- Templates are defined within a `<script>` tag or in external files.

```
<script id="template" type="text/x-handlebars-template">
  <p>Use the <strong>{{power}}</strong>, {{name}}!</p>
</script>
```



- Each template has an **id** and **type**.

Compiling a Template

- **Handlebars.compile** is used to compile a template.

```
//Compile the template  
let compiled_template = Handlebars.compile($('#template').html());
```

- Compiling = obtaining a JS object representing the template.

Obtaining the final HTML code

- You have to execute a template with a context in order to get its corresponding HTML code (i.e. map the template with the JSON object/array).

```
//Render the data into the template  
let rendered = compiled_template(person);
```



Expressions

- The simplest expression is a simple identifier.

```
<h1>{ {name} }</h1>
```

- This expression means "look up the name property in the current context".

Expressions

- Handlebars expressions can also be dot-separated paths.

```
<h1>{{user.username}}</h1>
```

- This expression means "look up the user property in the current context, then look up the username property in the result".

Values Escaping

- Handlebars HTML-escapes all the values returned by an `{{expression}}`.
- If you don't want Handlebars to escape a value, use the "triple-stash".

```
<script id="template" type="text/x-handlebars-template">  
  <p>Use the <strong>{{{power}}}</strong>, {{name}}!</p>  
</script>
```

Built-in Helpers

- **Each**

- To iterate over a list inside the block, you can use *this* to reference the element being iterated.

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

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



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

Built-in Helpers

- **With**
 - It shifts the context for a section of a template.

```
{ title: "My first post!",  
  author: { firstName: "Charles", lastName: "Jolley" }  
}
```

```
<div class="entry">  
<h1>{{title}}</h1>  
{{#with author}}  
<h2>By {{firstName}} {{lastName}}</h2>  
{{/with}}  
</div>
```



```
<div class="entry">  
<h1>My first post!</h1>  
<h2>By Charles Jolley</h2>  
</div>
```


Built-in Helpers

- **If - Else**
 - To conditionally render a block .
 - It will render the block if its argument is not equal to false, undefined, null, [].

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

The unless helper is the inverse of if

Example (From a Book)

Strawberry, Mango & Coconut Smoothie:

5-6 strawberries, frozen or fresh

1.5 cup mangoes, frozen or fresh

(1/2) cup coconut milk,

up to (1/2) cup water

Combine all ingredients in your blender and blend until smooth and creamy.

Example (From a Book)

```
const recipe = {
  "smoothie": "Strawberry, Mango & Coconut",
  "ingredients" : [
    {"qty": "5-6", "name": "strawberries frozen or fresh"},
    {"qty": "1.5 cup", "name": "mango frozen or fresh"},
    {"qty": ".5 cup", "name": "coconut milk, from the can"},
    {"qty": ".5 cup", "name": "water"}
  ],
  "instructions": [
    "Combine all ingredients in your blender", "Blend until smooth and creamy"
  ]
}
```

Example (produced with handlebars & JSON object)

Strawberry, Mango & Coconut

- 5-6 strawberries frozen or fresh
 - 1.5 cup mango frozen or fresh
 - .5 cup coconut milk, from the can
 - .5 cup water
1. Combine all ingredients in your blender
 2. Blend until smooth and creamy