Using Templates

Relevant Links

- Template literals[1]
- Handlebars[2]
- underscorejs templates[3]

Notes

Templates in General

HTML Templates are used to create an outline for the kind of HTML you want, with placeholders to fill in based on an object's parameters.

So here is a simple example of a template:

```
<div class="task" data-id="{{id}}">
  <h3>{{title}}<h3>
  <p>{{text}}</p>
</div>
```

So in this template there are three placeholders, one for a task's title and another for task's text. And a third placeholder inside the "data-id" attribute. Overall using this template would go as follows:

1. Load the template into a string templString.
2. "Compile" that template string: templFunc = Handlebars.compile(templString). This gives us back a function.
3. Call that function with an object to get a "filled in" string: templFunc({ id: 2, title: "yey!", text: "bo
4. Use that string with jquery's html-type methods to insert this string as HTML.

This is in general the idea: You separate the structure of the HTML from the creation of the data that is to be used to fill it in. This makes it very easy to completely change the look and feel of the program without having to change the code that generates it, by simply editing the template instead.

The power of templates stems from the functionalities they provide:

- Ways to iterate over a list of items
- Ways to conditionally insert a component based on a condition
- Ways to control whether something will go through HTML escaping (so that a less-than symbol is not interpreted as opening a tag) or not
- Ways to provide your own "builder" methods for performing smart tasks

[1] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals
[2] http://handlebarsjs.com/
[3] http://underscorejs.org/#template

- Ways to delve deeper into an object via nested paths

Different template libraries offer more or fewer of these features.

There are broadly speaking two kinds of "templates" you can use in Javascript:

- ES6 *Template Strings/Literals*. They offer some simple capabilities, and they are built in to modern Javascript engines.
- Third-part template libraries. They are quite powerful with control structures, helper functions and other features. The downside is that you need to include the libraries to your project.

We will take a brief look at ES6 template literals, and then focus on the Handlebars library.

ES6 Template literals

In ES6 a new syntax was introduced for defining template literals. It looks something like this:

```
let name = "Haris";
let language = "Javascript";
let s = `Here is a template. I am ${name} and I like ${language}!.
And we can even span multiple lines with this!
Btw, did you know that ${name} has ${name.length} characters?`;
```

The part between the backticks is the template literal. When that line is executed, the parts like `${name}` are evaluated in the current scope and substituted. You see that you can have some fairly elaborate logic inside the `${...}` part.

You can read more about template literals here[4].

Templates via Handlebars

The Handlebars library defines a Handlebars global, or you can use it as part of an module loading system (we will learn of those later). There is also a command-line utility that you can use to pre-compile your templates into AMD or other format. More details at the project's GitHub page[5].

For now we focus on the kind of functionality it offers:

{{aKey}} Inserts the value stored in the key `aKey` on the passed object. This value will undergo HTML escaping (so if it contains HTML tags, they will not be interpreted as such).

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Template_literals
[5] https://github.com/wycats/handlebars.js/

{{{aKey}}} Inserts the value stored in the key aKey *as is*. Any HTML present will result in HTML inserted.

{{aKey.withinAKey}} This will access a property within the object defined by a property of the main passed object. You can also use "../" to evaluate something relative to the "parent" context.

{{#each aKey}}...{{/each}} Meant to be used with a key that returns an array of values. Will repeat the contained block once for each value in the array, using that value as the context.

{{#if aKey}}...{{/if}} Only conditionally include that block depending on aKey's value.

{{! ...}} / {{!– –}} Comments.

{{helper arg1 arg2}} Custom helpers can be created by defining functions in a particular area. When they are used as the first term in a Handlebars placeholder, that function will be called. More details to follow.

Helper functions have their this set to the current context.

Some built-in helpers[6] are provided.

{{> partialName}} Can be used to insert other "partial" templates. So you can break your template into smaller reusable chunks.

Here's a reference to the complete Handlebars API[7]. Study it!

You can see an example of using a Handlebars template[8] in our TodoApp. It is loaded via a helper javascript file[9].

[6] http://handlebarsjs.com/builtin_helpers.html
[7] http://handlebarsjs.com/reference.html
[8] https://github.com/skiadas/WebAppsTodo/tree/master/app/template.handlebars
[9] https://github.com/skiadas/WebAppsTodo/tree/master/app/template.js