Introduction to Handlebars.js templates

By now you're probably wondering if it is really okay to put HTML code inside your `render` function. Yes, it is not okay however with some really simple markup it is still acceptable. Suppose, however, that our markup becomes more complex – having it inside JavaScript function is not a great idea. Therefore we are going to employ **templates** that allow you to extract portions of HTML code and later render it.

BackboneJS does not instruct you which templating engine to use – many developers stick with a built-in one, provided by Underscore. However I'd like to show you **HandlerbarsJS templating engine** which is very popular as well and provides many features. Visit http://handlebarsjs.com/ to learn more.

```
<script id="entry-template" type="text/x-handlebars-template">
    <div class="entry">
        <h1>{{title}}</h1>
        <div class="body">
            {{body}}
        </div>
    </div>
</script>
```

This is the Handlebars template and it mostly looks like a simple HTML. The piece of code in the curly brackets is the HandlebarsJS expression. The idea behind those expressions is really simple: we take the template and provide values for `title` and `body`. In the simplest case these templates are being placed right into your HTML page surrounded with a `script` tag. Note that I am assigning a unique id to this tag because later I am going to use it inside JS code. Also note that it has the type of `text/x-handlebars-template` therefore browser does not process it as regular JavaScript.

Using Handlebars.js templates

This is how this template is used:

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

```
var data = {title: 'My title', body: 'My body'}
var source = $('#enty-template').html()
var template = Handlebars.compile(source);
$('body').html( template(data) );
```

`data` is going to be interpolated into the template. Then I we template's HTML. After that use special `compile` function provided by Handlerbars. **Compilation** in this case is the process of converting the template to a JavaScript function. After that we can render the template anywhere on the page by passing an object with data to this function. The function returns plain HTML.

More complex examples

```
var data = {entries: [
    {title: "Title 1", body: "Body 1"},
    {title: "Title 2", body: "Body 2"}
]};

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

In this case we use each helper provided by Handlebars to cycle through an array of entries.

Handlebars also supports conditional statements:

```
var data = {title: 'My title', body: 'My body', visible: true}

{{#if visible}}
    <div class="entry">
        <h1>{{title}}</h1>
        <div class="body">
            {{body}}
        </div>
    </div>
{{else}}
    <p>This entry is hidden.</p>
{{/if}}

{{#unless visible}}
    <p>This entry is hidden.</p>
{{/unless}}
```

There are much more features that Handlebars offers like partial templates and creating custom helpers so be sure to check its documentation to learn more.

Include handlebars to project

Now let's include Handlebars into our project, create template and use it!

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/handlebars.js/3.0.1/handlebars.js"></script>
```

Don't use **runtime** version of Handlebars because this requires your templates to be precompiled with a special program and included into the project as JavaScript file. We are going to discuss this in the last step of the lesson.

Now at the bottom of the file create a new template for a single event:

```
<script id="event-template" type="text/x-handlebars-template">
  {{title}}
  <a href='#' class='btn btn-danger'>remove</a>
</script>
```

On to the `render` function. First of all we have to grab the template and use Handlebars' special `compile` method:

```
var template = Handlebars.compile($('#event-template').html());
```

Now we can render this template by passing data to it:

```
this.$el.html( template(this.model) );
```

Remember that `this.model` contains a JavaScript object with a single key `title`, so inside the template we are simply using `{{title}}`.