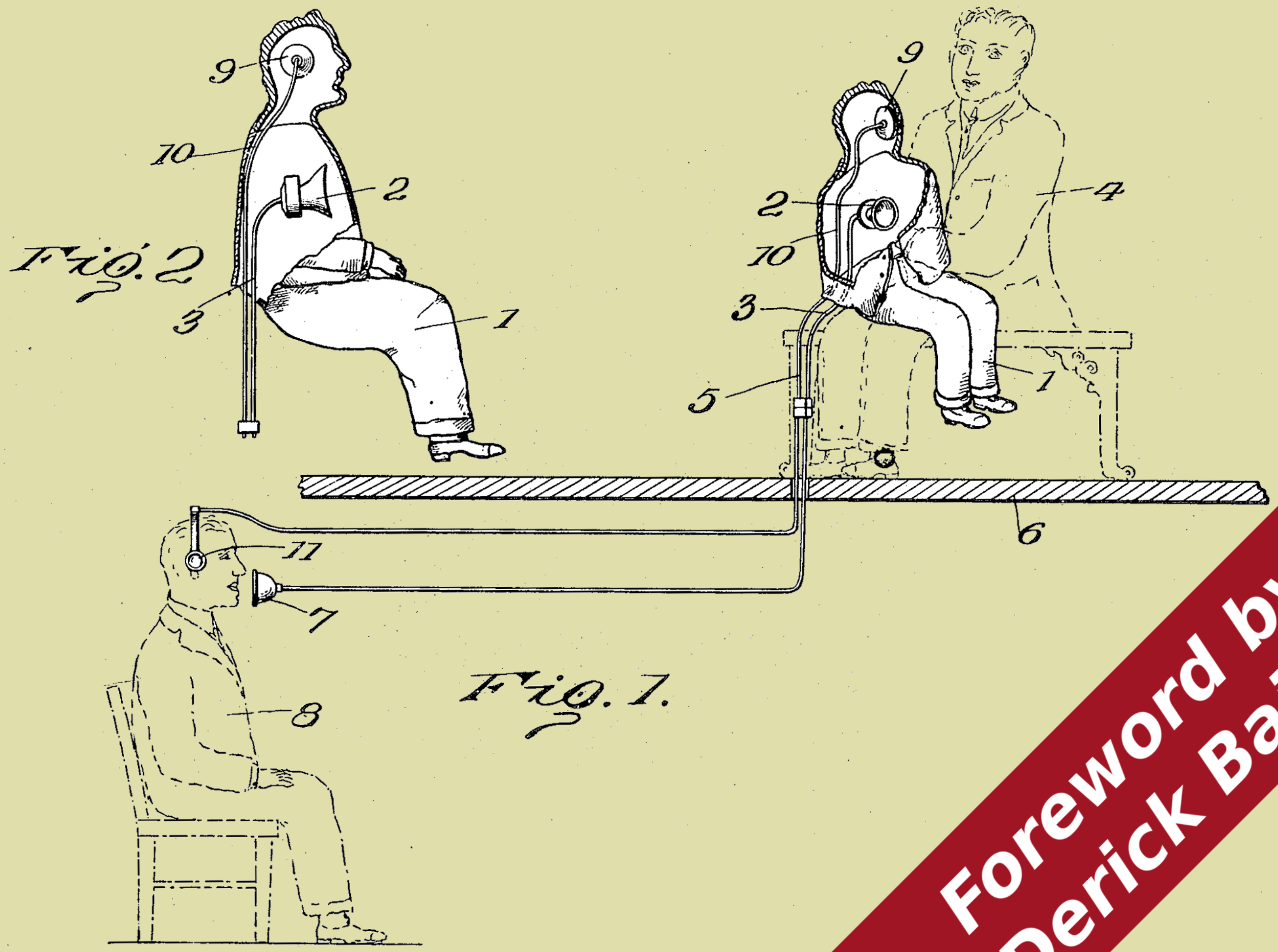
Build a Marionette.js app, one step at a time

Backbone Marionette.js



Foreword by Derick Bailey

A Gentle Introduction

by David Sulc

Backbone.Marionette.js: A Gentle Introduction

Build a Marionette.js app, one step at a time

David Sulc

Contents

Displaying a Model

Now that we've covered displaying static content, let's move on to displaying content containing data from a model. As you may know, one of Backbone's selling points is the possibility to structure javascript applications with a Model-View-Controller[1] (MVC) pattern. In this pattern, we use so-called *models* to interact with our data, passing them onto views for rendering the information they contain. You can learn more about models in Backbone's documentation[2].

So let's declare a model within our javascript block, above our view declaration:

```
ContactManager.Contact = Backbone.Model.extend({});
```

That wasn't very hard. What did we do? We simply declared a model named `Contact` and attached it to our `ContactManager` app. As you can see, this model extends Backbone's model definition and inherits various methods from it. When we extend Backbone's base model like this, we provide a javascript object (which is empty in our case) that can contain additional information pertaining to our model (we'll get back to that later).

Same as before, we'll need a template and a view definition before we can display anything in the browser. Let's replace our previous template and `StaticView` with the following:

```
<script type="text/template" id="contact-template">
  <p><%- firstName %> <%- lastName %></p>
</script>

ContactManager.ContactView = Marionette.ItemView.extend({
  template: "#contact-template"
});
```

The template will be included within the HTML `body`, but **outside** of the script block containing our application code. Refer to the full `index.html` included below if you're unsure where this code gets inserted.

You'll notice that we've got some special `<%- %>` tags in there. These serve the same purpose as in many templating languages (ERB in Rails, PHP, JSP, etc.): they allow the templating engine to interpret them and include the resulting output within the rendered result (while escaping the

[1]http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
[2]http://backbonejs.org/#Model

contained HTML). By default, Marionette uses Underscore's templating engine[3] where `<%- %>` means output will be displayed (and escaped), and `<% %>` tags which allow arbitrary javascript to be executed (such as an `if` condition), and `<%= %>` which displays the data after interpreting (without escaping any HTML). Since the model is serialized and passed on to the view template, writing `<%- firstName %>` means the model's `firstName` attribute will be displayed. For security reasons, **when in doubt, use the `<%- %>` alternative**.

So how do we display our view with model information? With our definitions written, we still need to create instances of a model and view, then display the view. All of this will happen within the `start` handler:

```
 1  ContactManager.on("start", function(){
 2    var alice = new ContactManager.Contact({
 3      firstName: "Alice",
 4      lastName: "Arten",
 5      phoneNumber: "555-0184"
 6    });
 7
 8    var aliceView = new ContactManager.ContactView({
 9      model: alice
10    });
11
12    ContactManager.regions.main.show(aliceView);
13  });
```

First, we create a model instance with data on lines 2-6: you'll notice we specify various model attributes and their respective values within a javascript object. Then, we create a new view instance and provide the model instance as an attribute on lines 8-10.

> Remember how we discussed passing options to the view when it gets instantiated? That's exactly what we're doing here: when we use the contact view, we'll always be using the same template (and have indicated it in the view definition for convenience), but the model we'll want to display will change. Therefore, we leave the model attribute out of the view's definition, and we specify which model to use each time we instantiate a new view instance.

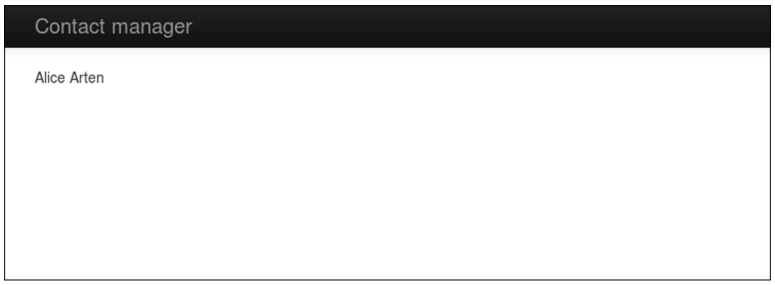
And all that's left to do after that is to display the view within the region (line 12), same as before.

Here's what our `index.html` looks like at this stage:

[3] http://underscorejs.org/#template

```
<div id="main-region" class="container">
  <p>Here is static content in the web page. You'll notice that it gets
  replaced by our app as soon as we start it.</p>
</div>

<script type="text/template" id="contact-template">
  <p><%- firstName %> <%- lastName %></p>
</script>

<!-- The javascript includes are here -->

<script type="text/javascript">
  var ContactManager = new Marionette.Application();

  ContactManager.Contact = Backbone.Model.extend({});

  ContactManager.ContactView = Marionette.ItemView.extend({
    template: "#contact-template"
  });

  ContactManager.on("before:start", function(){
    var RegionContainer = Marionette.LayoutView.extend({
      el: "#app-container",

      regions: {
        main: "#main-region"
      }
    });

    ContactManager.regions = new RegionContainer();
  });

  ContactManager.on("start", function(){
    var alice = new ContactManager.Contact({
      firstName: "Alice",
      lastName: "Arten",
      phoneNumber: "555-0184"
    });

    var aliceView = new ContactManager.ContactView({
      model: alice
    });

    ContactManager.regions.main.show(aliceView);
  });

  ContactManager.start();
</script>
```

And the visual result:



Displaying a model

Git commit to display our basic model view:

a5a36d3ffc6e3a71c29bee95082ba0269b70927e[4]

Using Model Defaults

What if our contact didn't have a first name? We don't want our app to break if the `firstName` attribute is missing: the template would be trying to retrieve an attribute that doesn't exist on the model. How can we manage this case? The functionality we're looking for is default values for model attributes.

To declare default attribute values, simply add a `defaults` object to the main object provided to our model definition:

```
ContactManager.Contact = Backbone.Model.extend({
  defaults: {
    firstName: ""
  }
});
```

If we now declare the following model instance

[4]https://github.com/davidsulc/marionette-gentle-introduction/commit/a5a36d3ffc6e3a71c29bee95082ba0269b70927e

```
var contact = new ContactManager.Contact({
  lastName: "Arten",
  phoneNumber: "555-0184"
});
```

and we try to display the missing `firstName` attribute, the empty string we defined as the default value will be shown instead.

Note that this code is included only to demonstrate default model attributes. It will not be part of our application's code: later on, we will add model validations to manage missing attribute values.

Introducing Events

Let's enrich our view slightly: we've got a phone number for Alice, so let's display it in an alert when her name is clicked.

Marionette views inherit all of Backbone's functionality, among which the ability to define events and their associated handlers. Here's what they look like:

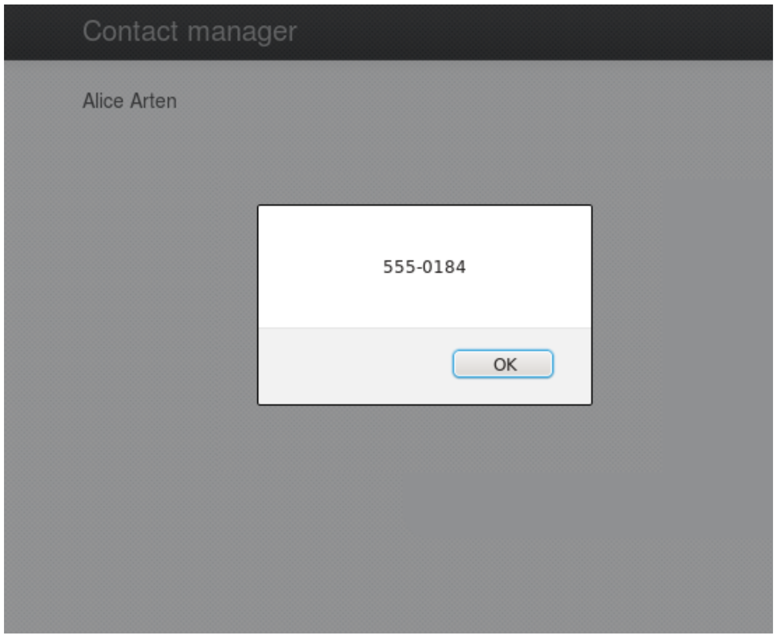
```
events: {
  "click p": "alertPhoneNumber"
}
```

This event translates as "when the user `clicks` the `p` tag that can be found in this view, call the `alertPhoneNumber` function". If you've used jQuery, you'll recognize it's essentialy an event name followed by a selector (which could contain class names, etc.). Let's use this feature in our view to display Alice's phone number, by modifying our view declaration:

```
ContactManager.ContactView = Marionette.ItemView.extend({
    template: "#contact-template",

    events: {
      "click p": "alertPhoneNumber"
    },

    alertPhoneNumber: function(){
      alert(this.model.escape("phoneNumber"));
    }
  });
```

Backbone models' `escape`[5] works the same way as `get`[6]: they both return the value of the attribute provided as an argument, but `escape` will escape HTML content, protecting you from XSS attacks if you're displaying user-provided data within the HTML.

If you now refresh the page and click on Alice's name, you'll see her phone number displayed. Pretty straightforward, right? You'll notice that since we're in the view definition when we're writing our `alertPhoneNumber` function, we have access to the view's model instance via `this.model`, even though *which* model instance will be used isn't known yet (it's provided when we instantiate the view, remember?).



Displaying an alert when clicking a contact

This code is not going to be included in our app, so you won't see it going forward.

Exercise

Displaying a Contact With No Phone Number

Add a default phone number of "No phone number!". Then create a new model without a phone number, and click on it. Make sure that "No phone number!" is displayed in the alert. You can see the exercise solution at the end of the book.

[5]http://backbonejs.org/#Model-escape
[6]http://backbonejs.org/#Model-get