Building The User UI Using HTML5

What Will You Learn Here?

We’ve just implemented the business services of our application, and exposed them through RESTful endpoints. Now we need to implement a flexible user interface that can be easily used with both desktop and mobile clients. After reading this tutorial, you will understand our front-end design and the choices that we made in its implementation. Topics covered include:

- Creating single-page applications using HTML5, JavaScript and JSON
- Using JavaScript frameworks for invoking RESTful endpoints and manipulating page content
- Feature and device detection
- Implementing a version of the user interface that is optimized for mobile clients using JavaScript frameworks such as jQuery mobile

The tutorial will show you how to perform all these steps in JBoss Developer Studio, including screenshots that guide you through.

First, the basics

In this tutorial, we will build a single-page application. All the necessary code: HTML, CSS and JavaScript is retrieved within a single page load. Rather than refreshing the page every time the user changes a view, the content of the page will be redrawn by manipulating the DOM in JavaScript. The application uses REST calls to retrieve data from the server.

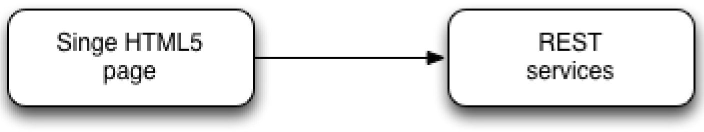


Figure 1. Single page application

Client-side MVC Support

Because this is a moderately complex example, which involves multiple views and different types of data, we will use a client-side MVC framework to structure the application, which provides amongst others:

- routing support within the single page application;
- event-driven interaction between views and data;
- simplified CRUD invocations on RESTful services.

In this application we use the client-side MVC framework "backbone.js".

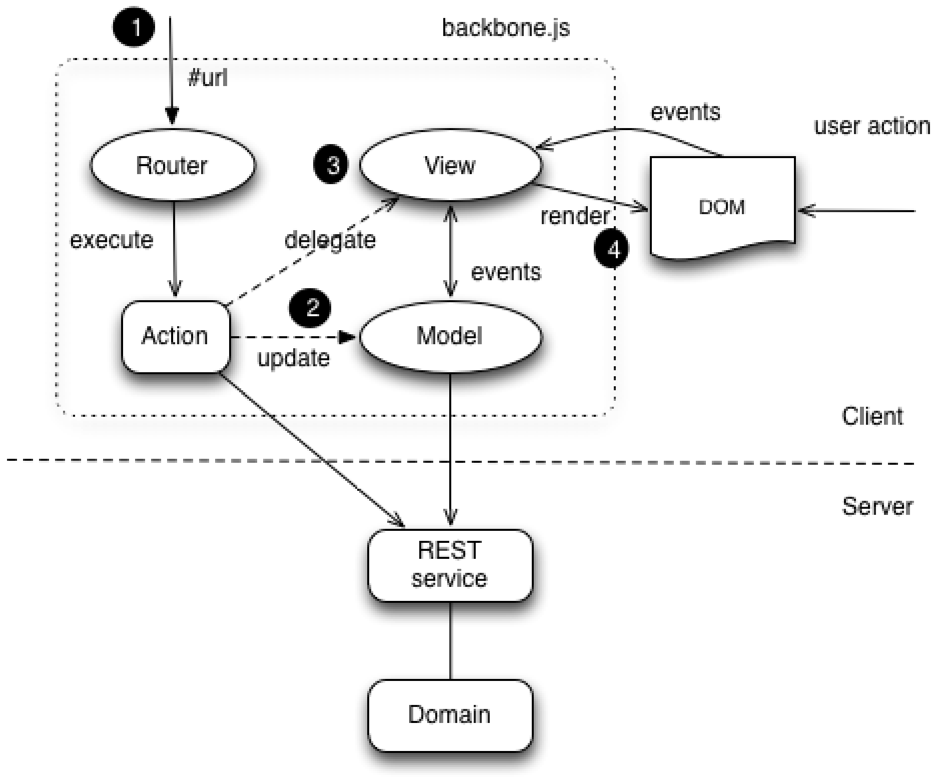


Figure 2. Backbone architecture

Modularity

In order to provide good separation of concerns, we split the JavaScript code into modules. Ensuring that all the modules of the application are loaded properly at runtime becomes a complex task, as the application size increases. To conquer this complexity, we use the Asynchronous Module Definition mechanism as implemented by the "require.js" library.

TIP

Asynchronous Module Definition

The Asynchronous Module Definition (AMD) API specifies a mechanism for defining modules such that the module, and its dependencies, can be asynchronously loaded. This is particularly well suited for the browser where synchronous loading of modules incurs performance, usability, debugging, and cross-domain access problems.

Templating

Instead of manipulating the DOM directly, and mixing up HTML with the JavaScript code, we create HTML markup fragments separately as templates which are applied when the application views are rendered.

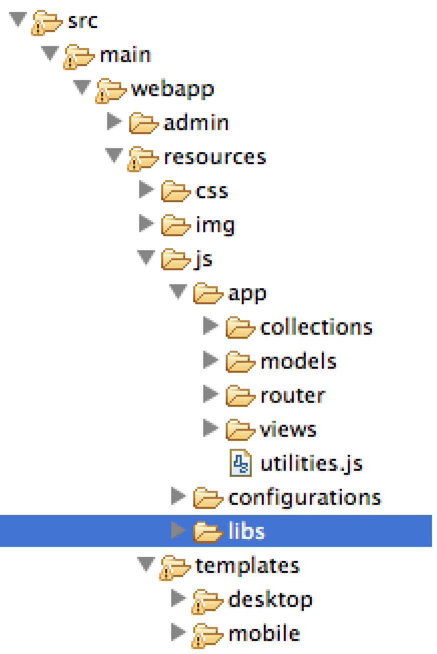
In this application we use the templating support provided by "underscore.js".

Mobile and desktop versions

The page flow and structure, as well as feature set, are slightly different for mobile and desktop, and therefore we will build two variants of the single-page-application, one for desktop and one for mobile. As the application variants are very similar, we will cover the desktop version of the application first, and then we will explain what is different in the mobile version.

Setting up the structure

Before we start developing the user interface, we need to set up the general application structure and add the JavaScript libraries. First, we create the directory structure:



```
src
  main
    webapp
      admin
      resources
        css
        img
        js
          app
            collections
            models
            router
            views
            utilities.js
          configurations
          libs
        templates
          desktop
          mobile
```

Figure 3. File structure for our web application

We put stylesheets in `resources/css` folder, images in `resources/img`, and HTML view templates in `resources/templates`. `resources/js` contains the JavaScript code, split between `resources/js/libs` - which contains the libraries used by the application, `resources/js/app` - which contains the application code, and `resources/js/configurations` which contains module definitions for the different versions of the application - i.e. mobile and desktop. The `resources/js/app` folder will contain the application modules, in subsequent subdirectories, for models, collections, routers and views.

The first step in implementing our solution is adding the stylesheets and JavaScript libraries to the `resources/css` and `resources/js/libs`:

require.js

AMD support, along with the plugin:

- text - for loading text files, in our case the HTML templates

jQuery

general purpose library for HTML traversal and manipulation

Underscore

JavaScript utility library (and a dependency of Backbone)

Backbone

Client-side MVC framework

Bootstrap

UI components and stylesheets for page structuring

Modernizr

JavaScript library for HTML5 and CSS3 feature detection

You can copy these libraries (with associated stylesheets) from the project sources. You can also copy the CSS stylesheet in `screen.css`, since we'll include this stylesheet in the HTML. Addtionally, copy the images from the `src/main/webapp/resources/img` directory in the project sources to the equivalent one in your workspace.

Now, we create the main page of the application (which is the URL loaded by the browser):

src/main/webapp/index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Ticket Monster</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1, user-
scalable=0"/>

    <script type="text/javascript" src="resources/js/libs/modernizr-
2.8.3.min.js"></script>
    <script type="text/javascript" src="resources/js/libs/require.js"
            data-main="resources/js/configurations/loader"></script>
</head>
<body>
</body>
</html>
```

As you can see, the page does not contain much. It loads Modernizr (for HTML5 and CSS3 feature detection) and RequireJS (for loading JavaScript modules in an asynchronous manner). Once RequireJS is loaded by the browser, it will configure itself to use a `baseUrl` of `resources/js/configurations` (specified via the `data-main` attribute on the `script` tag). All scripts loaded by RequireJS will use this `baseUrl` unless specified otherwise.

RequireJS will then load a script having a module ID of `loader` (again, specified via the `data-main` attribute):

src/main/webapp/resources/js/configurations/loader.js

```
//detect the appropriate module to load
define(function () {

    /*
```

```
     A simple check on the client. For touch devices or small-resolution screens)
     show the mobile client. By enabling the mobile client on a small-resolution
screen
     we allow for testing outside a mobile device (like for example the Mobile Browser
     simulator in JBoss Tools and JBoss Developer Studio).
     */

    var environment;

    if (Modernizr.touch || Modernizr.mq("only all and (max-width: 480px)")) {
        environment = "mobile"
    } else {
        environment = "desktop"
    }

    require([environment]);
});
```

This script detects the current client (mobile or desktop) based on its capabilities (touch or not) and loads another JavaScript module (`desktop` or `mobile`) defined in the `resources/js/configurations` folder (aka the `baseUrl`) depending on the detected features. In the case of the desktop client, the code is loaded from `resources/js/configurations/desktop.js`.

src/main/webapp/resources/js/configurations/desktop.js

```
/**
 * Shortcut alias definitions - will come in handy when declaring dependencies
 * Also, they allow you to keep the code free of any knowledge about library
 * locations and versions
 */
requirejs.config({
    baseUrl: "resources/js",
    paths: {
        jquery:'libs/jquery-2.0.3',
        underscore:'libs/underscore',
        text:'libs/text',
        bootstrap: 'libs/bootstrap',
        backbone: 'libs/backbone',
        utilities: 'app/utilities',
        router:'app/router/desktop/router'
    },
    // We shim Backbone.js and Underscore.js since they don't declare AMD modules
    shim: {
        'backbone': {
            deps: ['jquery', 'underscore'],
            exports: 'Backbone'
        },

        'underscore': {
            exports: '_'
        }
    }
});

define("initializer", ["jquery"],
    function ($) {
    // Configure jQuery to append timestamps to requests, to bypass browser caches
    // Important for MSIE
    $.ajaxSetup({cache:false});
    $('head').append('<link rel="stylesheet" href="resources/css/bootstrap.css"
type="text/css" media="all"/>');
    $('head').append('<link rel="stylesheet" href="resources/css/bootstrap-theme.css"
type="text/css" media="all"/>');
    $('head').append('<link rel="stylesheet" href="resources/css/screen.css"
type="text/css" media="all"/>');
    $('head').append('<link href="http://fonts.googleapis.com/css?family=Rokkitt"
rel="stylesheet" type="text/css">');
});

// Now we load the dependencies
// This loads and runs the 'initializer' and 'router' modules.
require([
    'initializer',
    'router'
], function(){
});

define("configuration", {
    baseUrl : ""
});
```

The module loads all the utility libraries, converting them to AMD modules where necessary (like it is the case for Backbone). It also defines two modules of its own - an initializer that loads the application stylesheets for the page, and the `configuration` module that allows customizing the REST service URLs (this will become in handy in a further tutorial).

We also define some utility JavaScript functions that are used in the rest of the front-end in a `utilities` module (also referenced in the `desktop` module above). For convenience, copy the `utilities.js` file from the `src/main/webapp/resources/js/app/` directory in the project sources.

Before we add any functionality, let us create a first landing page. We will begin by setting up a critical piece of the application, the router.

Routing

The router allows for navigation in our application via bookmarkable URLs, and we will define it as follows:

src/main/webapp/resources/js/app/router/desktop/router.js

```
/**
 * A module for the router of the desktop application
 */
define("router", [
    'jquery',
    'underscore',
    'configuration',
    'utilities',
    'text!../templates/desktop/main.html'
],function ($,
            _,
            config,
            utilities,
            MainTemplate) {

    $(document).ready(new function() {
       utilities.applyTemplate($('body'), MainTemplate)
    })

    /**
     * The Router class contains all the routes within the application -
     * i.e. URLs and the actions that will be taken as a result.
     *
     * @type {Router}
     */

    var Router = Backbone.Router.extend({
        initialize: function() {
            //Begin dispatching routes
            Backbone.history.start();
        },
        routes:{
        }
    });

    // Create a router instance
    var router = new Router();

    return router;
});
```

Remember, this is a single page application. You can either navigate using urls such as `http://localhost:8080/ticket-monster/index.html#events` or using relative urls (from within the application, this being exactly what the main menu does). The fragment after the hash sign represents the url within the single page, on which the router will act, according to the mappings set up in the `routes` property.

During the router set up, we load the page template for the entire application. TicketMonster uses a templating library in order to separate application logic from it's actual graphical content. The actual HTML is described in template files, which are applied by the application, when necessary, on a DOM element - effectively populating it's content. So the general content of the page, as described in the `body` element is described in a template file too. Let us define it.

src/main/webapp/resources/templates/desktop/main.html

```
<!--
    The main layout of the page - contains the menu and the 'content' &lt;div/&gt; in
which all the
    views will render the content.
-->
<div id="logo"><div class="wrap"><h1>Ticket Monster</h1></div></div>
<div id="container">
    <div id="menu">
        <div class="navbar">
            <!-- Toggle get grouped for better mobile display -->
            <div class="navbar-header">
                <button type="button" class="navbar-toggle" data-toggle="collapse"
data-target="#navbar-items">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
            </div>

            <!-- Collect the nav links, forms, and other content for toggling -->
            <div id="navbar-items" class="collapse navbar-collapse">
                <ul class="nav navbar-nav">
                    <li><a href="#about">About</a></li>
                    <li><a href="#events">Events</a></li>
                    <li><a href="#venues">Venues</a></li>
                    <li><a href="#bookings">Bookings</a></li>
                    <li><a href="#monitor">Monitor</a></li>
                    <li><a href="admin">Administration</a></li>
                </ul>
            </div>
        </div>
    </div>
    <div id="content" class="container">
    </div>
</div>

<footer style="">
    <div style="text-align: center;"><img src="resources/img/rhjb_eap_logo.png"
alt="HTML5"/></div>
</footer>
```

The actual HTML code of the template contains a menu definition which will be present on all the pages, as well as an empty element named `content`, which is the placeholder for the application views. When a view is displayed, it will apply a template and populate the `content` element.

Setting up the initial views

Let us complete our application setup by creating an initial landing page. The first thing that we will need to do is to add a view component.

src/main/resources/js/app/views/desktop/home.js

```
/**
 * The About view
 */
define([
    'utilities',
    'text!../../../../templates/desktop/home.html'
], function (utilities, HomeTemplate) {

    var HomeView = Backbone.View.extend({
        render:function () {
            utilities.applyTemplate($(this.el),HomeTemplate,{});
            return this;
        }
    });

    return HomeView;
});
```

Functionally, this is a very basic component - it only renders the splash page of the application, but it helps us introduce a new concept that will be heavily used throughout the application views. One main role of a view is to describe the logic for manipulating the page content. It will do so by defining a function named `render` which will be invoked by the application. In this very simple case, all that the view does is to create the content of the splash page. You can proceed by copying the content of `src/main/webapp/resources/templates/desktop/home.html` to your project.

TIP

Backbone Views

Views are logical representations of user interface elements that can interact with data components, such as models in an event-driven fashion. Apart from defining the logical structure of your user interface, views handle events resulting from the user interaction (e.g. clicking a DOM element or selecting an element into a list), translating them into logical actions inside the application.

Once we defined a view, we must tell the router to navigate to it whenever requested. We will add the following dependency and mapping to the router:

src/main/webapp/resources/js/app/router/desktop/router.js

```
/**
```

```
 * A module for the router of the desktop application
 */
define("router", [
    'jquery',
    'underscore',
    'configuration',
    'utilities',
    'app/views/desktop/home',
    'text!../templates/desktop/main.html'
],function ($,
            _,
            config,
            utilities,
            HomeView,
            MainTemplate) {

    ...
    var Router = Backbone.Router.extend({
        ...
        routes : {
            "":"home",
            "about":"home"
        },
        home : function () {
            utilities.viewManager.showView(new HomeView({el:$("#content")}));
        }
    });
    ...
```

We have just told the router to invoke the `home` function whenever the user navigates to the root of the application or uses a `#about` hash. The method will simply cause the `HomeView` defined above to render.

Now you can navigate to `http://localhost:8080/ticket-monster/#about` or `http://localhost:8080/ticket-monster` and see the results.

Displaying Events

The first use case that we implement is event navigation. The users will be able to view the list of events and select the one that they want to attend. After doing so, they will select a venue, and will be able to choose a performance date and time.

The Event model

We define a Backbone model for holding event data. Nearly all domain entities (booking, event, venue) are represented by a corresponding Backbone model:

src/main/webapp/resources/js/app/models/event.js

```
/**
 * Module for the Event model
 */
define([
    'configuration',
    'backbone'
], function (config) {
    /**
     * The Event model class definition
     * Used for CRUD operations against individual events
     */
    var Event = Backbone.Model.extend({
        urlRoot: config.baseUrl + 'rest/events' // the URL for performing CRUD
operations
    });
    // export the Event class
    return Event;
});
```

The `Event` model can perform CRUD operations against the REST services we defined earlier.

TIP

Backbone Models

Backbone models contain data as well as much of the logic surrounding it: conversions, validations, computed properties, and access control. They also perform CRUD operations with the REST service.

The Events collection

We define a Backbone collection for handling groups of events (like the events list):

src/main/webapp/resources/js/app/collections/events.js

```
/**
 * Module for the Events collection
 */
define([
    // The collection element type and configuration are dependencies
    'app/models/event',
    'configuration'
], function (Event, config) {
    /**
     *  Here we define the Bookings collection
     *  We will use it for CRUD operations on Bookings
     */
    var Events = Backbone.Collection.extend({
        url: config.baseUrl + "rest/events", // the URL for performing CRUD operations
        model: Event,
        id:"id", // the 'id' property of the model is the identifier
        comparator:function (model) {
            return model.get('category').id;
        }
    });
    return Events;
});
```

By mapping the model and collection to a REST endpoint you can perform CRUD operations without having to invoke the services explicitly. You will see how that works a bit later.

TIP

Backbone Collections

Collections are ordered sets of models. They can handle events which are fired as a result of a change to a individual member, and can perform CRUD operations for syncing up contents against RESTful services.

The EventsView view

Now that we have implemented the data components of the example, we need to create the view that displays them.

src/main/webapp/resources/js/app/views/desktop/events.js

```
define([
    'utilities',
    'bootstrap',
    'text!../../../../templates/desktop/events.html'
], function (
    utilities,
    bootstrap,
    eventsTemplate) {

    var EventsView = Backbone.View.extend({
        events:{
            "click a":"update"
        },
        render:function () {
            var categories = _.uniq(
                _.map(this.model.models, function(model){
                    return model.get('category')
                }), false, function(item){
                    return item.id
                });
            utilities.applyTemplate($(this.el), eventsTemplate,
{categories:categories, model:this.model})
            $(this.el).find('.item:first').addClass('active');
            $(".carousel").carousel();
            $("a[rel='popover']").popover({trigger:'hover',container:'body'});
            return this;
        },
        update:function () {
            $("a[rel='popover']").popover('hide')
        }
    });

    return  EventsView;
});
```

As we explained, earlier, the view is attached to a DOM element (the `el` property). When the `render` method is invoked, it manipulates the DOM and renders the view. We could have achieved this by writing these instructions directly in the method, but that would make it hard to change the page design later on. Instead, we create a template and apply it, thus separating the HTML view code from the view implementation. Note the dependency on the Bootstrap module - we initialize the Bootstrap carousel and popover components when this view is rendered.

src/main/webapp/resources/templates/desktop/events.html

```
<div class="row">
    <div class="col-md-3 col-md-offset-1">
        <div class="panel" id="itemMenu">

            <%
            _.each(categories, function (category) {
            %>
            <div class="panel panel-default">
                <div class="panel-heading">
                    <a class="panel-toggle"
                       data-target="#category-<%=category.id%>-collapsible" data-
toggle="collapse"
                       data-parent="#itemMenu"><%= category.description %></a>
                </div>
                <div id="category-<%=category.id%>-collapsible" class="panel-collapse
collapse">
                    <div id="category-<%- category.id%>" class="panel-body">

                        <%
                        _.each(model.models, function (model) {
                        if (model.get('category').id == category.id) {
                        %>
                        <p><a href="#events/<%- model.attributes.id%>" rel="popover"
                              data-content="<%- model.attributes.description%>"
                              data-original-title="<%-
model.attributes.name%>"><%=model.attributes.name%></a></p>
                        <% }
                        });
                        %>
                    </div>
                </div>
            </div>
            <% }); %>
        </div>
    </div>

    <div id='itemSummary' class="col-md-8 hidden-sm hidden-xs">
        <div class="carousel-container">
            <div id="eventCarousel" class="carousel slide">
                <!-- Carousel items -->
                <div class="carousel-inner">
                    <%_.each(model.models, function(model) {
                        if( model.get('mediaItem')) {
                    %>
                    <div class="item">
                        <img src='rest/media/<%=model.attributes.mediaItem.id%>'/>

                        <div class="carousel-caption">
                            <div class="row">
                                <div class="col-md-9">
                                    <h4><%=model.attributes.name%></h4>
                                    <p><%=model.attributes.description%></p>
                                </div>
                                <div class="col-md-2">
                                    <a class="btn btn-danger action"
href="#events/<%=model.id%>">Book tickets</a>
                                </div>
                            </div>
                        </div>
                    </div>
                    <%
                        }
                     });
                    %>
                </div>
                <!-- Carousel nav -->
                <a class="carousel-control left" href="#eventCarousel" data-
slide="prev">
                    <span class="glyphicon glyphicon-chevron-left"></span>
                </a>
                <a class="carousel-control right" href="#eventCarousel" data-
slide="next">
                    <span class="glyphicon glyphicon-chevron-right"></span>
                </a>
            </div>
        </div>
    </div>
</div>
```

As well as applying the template and preparing the data that will be used to fill it in (the `categories` and `model` entries in the map), the `render` method also performs the JavaScript calls that are required to initialize the UI components (in this case the Bootstrap carousel and popover).

A view can also listen to events fired by the children of it's root element (`el`). In this case, the `update` method is configured to listen to clicks on anchors. The configuration occurs within the `events` property of the class.

Now that the views are in place, we need to add another routing rule to the application.

src/main/webapp/resources/js/app/router/desktop/router.js

```
/**
 * A module for the router of the desktop application
 */
define("router", [
    ...
    'utilities',
    'app/collections/events',
    'app/views/desktop/home',
    'app/views/desktop/events',
    ...
    'text!../templates/desktop/main.html'
],function ($,
            ...
            utilities,
            Events,
            HomeView,
            EventsView,
            ...
            MainTemplate) {

    var Router = Backbone.Router.extend({
        ...
        routes : {
            ...,
            "events":"events"
        },
        ...,
        events:function () {
            var events = new Events();
            var eventsView = new EventsView({model:events, el:$("#content")});
            events.on("reset",
                function () {
                    utilities.viewManager.showView(eventsView);
                }).fetch({
                    reset : true,
                    error : function() {
                        utilities.displayAlert("Failed to retrieve events from the
TicketMonster server.");
                    }
                });
        }
    });
```

The `events` function handles the `#events` fragment and will retrieve the events in our application via a REST call. We don't manually perform the REST call as it is triggered the by invocation of `fetch` on the `Events` collection, as discussed earlier.

The `reset` event on the collection is invoked when the data from the server is received, and the collection is populated. This triggers the rendering of the events view (which is bound to the `#content` div).

The whole process is event orientated - the models, views and controllers interact through events.

Viewing a single event

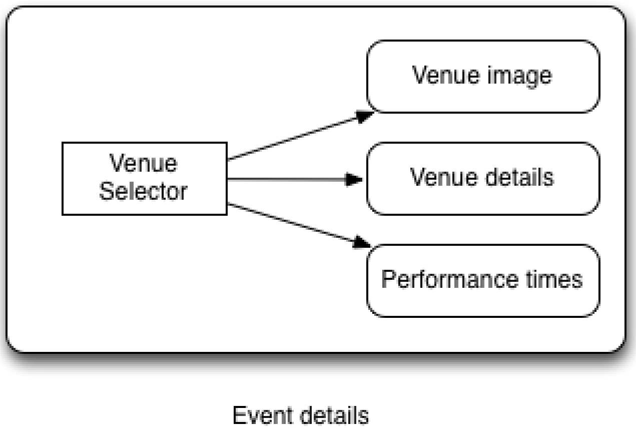
With the events list view now in place, we can add a view to display the details of each individual event, allowing the user to select a venue and performance time.

We already have the models in place so all we need to do is to create the additional view and expand the router. First, we'll implement the view:

src/main/webapp/resources/js/app/views/desktop/event-detail.js

```
define([
    'utilities',
    'require',
    'text!../../../../templates/desktop/event-detail.html',
    'text!../../../../templates/desktop/media.html',
    'text!../../../../templates/desktop/event-venue-description.html',
    'configuration',
    'bootstrap'
], function (
    utilities,
    require,
    eventDetailTemplate,
    mediaTemplate,
    eventVenueDescriptionTemplate,
    config,
    Bootstrap) {

    var EventDetail = Backbone.View.extend({

        events:{
            "click input[name='bookButton']":"beginBooking",
            "change select[id='venueSelector']":"refreshShows",
            "change select[id='dayPicker']":"refreshTimes"
        },

        render:function () {
            $(this.el).empty()
            utilities.applyTemplate($(this.el), eventDetailTemplate,
this.model.attributes);
            $("#bookingOption").hide();
            $("#venueSelector").attr('disabled', true);
            $("#dayPicker").empty();
            $("#dayPicker").attr('disabled', true)
            $("#performanceTimes").empty();
            $("#performanceTimes").attr('disabled', true)
            var self = this
            $.getJSON(config.baseUrl + "rest/shows?event=" + this.model.get('id'),
function (shows) {
                self.shows = shows
                $("#venueSelector").empty().append("<option value='0' selected>Select
a venue</option>");
                $.each(shows, function (i, show) {
                    $("#venueSelector").append("<option value='" + show.id + "'>" +
show.venue.address.city + " : " + show.venue.name + "</option>")
                });
                $("#venueSelector").removeAttr('disabled')
            })
            return this;
        },
        beginBooking:function () {
            require("router").navigate('/book/' + $("#venueSelector
option:selected").val() + '/' + $("#performanceTimes").val(), true)
        },
        refreshShows:function (event) {
            event.stopPropagation();
            $("#dayPicker").empty();

            var selectedShowId = event.currentTarget.value;

            if (selectedShowId != 0) {
                var selectedShow = _.find(this.shows, function (show) {
                    return show.id == selectedShowId
                });
                this.selectedShow = selectedShow;
                utilities.applyTemplate($("#eventVenueDescription"),
eventVenueDescriptionTemplate, {venue:selectedShow.venue});
                var times = _.uniq(_.sortBy(_.map(selectedShow.performances, function
(performance) {
                    return (new Date(performance.date).withoutTimeOfDay()).getTime()
                }), function (item) {
                    return item
                }));
                utilities.applyTemplate($("#venueMedia"), mediaTemplate,
selectedShow.venue)
                $("#dayPicker").removeAttr('disabled')
                $("#performanceTimes").removeAttr('disabled')
                _.each(times, function (time) {
                    var date = new Date(time)
                    $("#dayPicker").append("<option value='" + date.toYMD() + "'>" +
date.toPrettyStringWithoutTime() + "</option>")
                });
                this.refreshTimes()
                $("#bookingWhen").show(100)
            } else {
                $("#bookingWhen").hide(100)
                $("#bookingOption").hide()
                $("#dayPicker").empty()
                $("#venueMedia").empty()
                $("#eventVenueDescription").empty()
                $("#dayPicker").attr('disabled', true)
                $("#performanceTimes").empty()
                $("#performanceTimes").attr('disabled', true)
            }

        },
        refreshTimes:function () {
            var selectedDate = $("#dayPicker").val();
            $("#performanceTimes").empty()
            if (selectedDate) {
                $.each(this.selectedShow.performances, function (i, performance) {
                    var performanceDate = new Date(performance.date);
                    if (_.isEqual(performanceDate.toYMD(), selectedDate)) {
                        $("#performanceTimes").append("<option value='" +
performance.id + "'>" + performanceDate.getHours().toZeroPaddedString(2) + ":" +
performanceDate.getMinutes().toZeroPaddedString(2) + "</option>")
                    }
                })
            }
            $("#bookingOption").show()
        }

    });

    return  EventDetail;
});
```

This view is more complex than the global events view, as portions of the page need to be updated when the user chooses a venue.



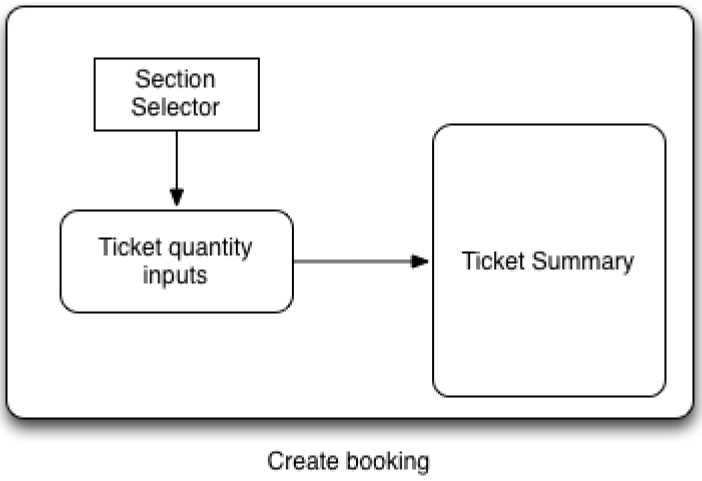


Figure 4. On the event details page some fragments are re-rendered when the user selects a venue

The view responds to three different events:

- changing the current venue triggers a reload of the venue details and the venue image, as well as the performance times. The application retrieves the performance times through a REST call.
- changing the day of the performance causes the performance time selector to reload.
- once the venue and performance date and time have been selected, the user can navigate to the booking page.

The corresponding templates for the three fragments rendered above are:

src/main/webapp/resources/templates/desktop/event-detail.html

```
<div class="row">
    <h2 class="page-header special-title light-font"><%=name%></h2>
</div>
<div class="row">
    <div class="col-md-4">
        <div class="well">
```

```
            <div class="row">
                <h3 class="page-header col-md-6">What?</h3>
                <% if(mediaItem) { %><img width="100"
src='rest/media/<%=mediaItem.id%>'/><% } %>
            </div>
            <div class="row top5">
                <div class="col-md-12"><%= description %></div>
            </div>
        </div>
    </div>
    <div class="col-md-4">
        <div class="well">
            <div class="row">
                <h3 class="page-header col-md-6">Where?</h3>
                <div class="col-md-6" id='venueMedia'/>
            </div>
            <div class="row top5">
                <div class="col-md-12">
                    <select id="venueSelector" class="form-control"/>
                </div>
            </div>
            <div class="row top5">
                <div class="col-md-12">
                    <div id="eventVenueDescription"/>
                </div>
            </div>
        </div>
    </div>
    <div id='bookingWhen' style="display: none;" class="col-md-4">
        <div class="well">
            <div class="row">
                <h3 class="page-header col-md-6">When?</h3>
            </div>

            <div class="row top5">
                <div class="col-md-12">
                    <select class="form-control" id="dayPicker"/>
                </div>
            </div>
            <div class="row top5">
                <div class="col-md-12">
                    <select class="form-control" id="performanceTimes"/>
                </div>
            </div>

            <div id="bookingOption" class="row top5">
                <div class="col-md-6">
                    <input name="bookButton" class="btn btn-primary" type="button"
value="Order tickets">
                </div>
            </div>
        </div>
    </div>
</div>
```

src/main/webapp/resources/templates/desktop/event-venue-description.html

```
<address>
    <p><%= venue.description %></p>
    <p><strong>Address:</strong></p>
    <p><%= venue.address.street %></p>
    <p><%= venue.address.city %>, <%= venue.address.country %></p>
</address>
```

src/main/webapp/resources/templates/desktop/media.html

```
<%if (mediaItem) { %><img width="100" src='rest/media/<%=mediaItem.id%>'/><% } %>
```

Now that the view exists, we add it to the router:

src/main/webapp/resources/js/app/router/desktop/router.js

```
/**
 * A module for the router of the desktop application
 */
define("router", [
    ...
    'app/models/event',
    ...,
    'app/views/desktop/event-detail',
    ...
],function (
            ...
            Event,
            ...
            EventDetailView,
            ...) {

    var Router = Backbone.Router.extend({
        ...
        routes:{
            ...
            "events/:id":"eventDetail",
        },
        ...
        eventDetail:function (id) {
            var model = new Event({id:id});
            var eventDetailView = new EventDetailView({model:model,
el:$("#content")});
            model.on("change",
                function () {
                    utilities.viewManager.showView(eventDetailView);
                }).fetch({
                    error : function() {
                        utilities.displayAlert("Failed to retrieve the event from the
TicketMonster server.");
                    }
                });
        }
    }
    ...
);
```

As you can see, this is very similar to the previous view and route, except that now the application can accept parameterized URLs (e.g. `http://localhost:8080/ticket-monster/index#events/1`). This URL can be entered directly into the browser, or it can be navigated to as a relative path (e.g. `#events/1`) from within the applicaton.

With this in place, all that remains is to implement the final view of this use case, creating the bookings.

Creating Bookings

The user has chosen the event, the venue and the performance time, and must now create the booking. Users can select one of the available sections for the show's venue, and then enter the number of tickets required for each category available for this show (Adult, Child, etc.). They then add the tickets to the current order, which causes the summary view to be updated. Users can also remove tickets from the order. When the order is complete, they enter their contact information (e-mail address) and submit the order to the server.

First, we add the new view:

src/main/webapp/resources/js/app/views/desktop/create-booking.js

```
define([
    'utilities',
    'require',
    'configuration',
    'text!../../../../templates/desktop/booking-confirmation.html',
    'text!../../../../templates/desktop/create-booking.html',
    'text!../../../../templates/desktop/ticket-categories.html',
    'text!../../../../templates/desktop/ticket-summary-view.html',
    'bootstrap'
],function (
    utilities,
    require,
    config,
    bookingConfirmationTemplate,
    createBookingTemplate,
    ticketEntriesTemplate,
    ticketSummaryViewTemplate){


    var TicketCategoriesView = Backbone.View.extend({
        id:'categoriesView',
        intervalDuration : 100,
        formValues : [],
        events:{
            "change input":"onChange"
        },
        render:function () {
            if (this.model != null) {
                var ticketPrices = _.map(this.model, function (item) {
                    return item.ticketPrice;
                });
                utilities.applyTemplate($(this.el), ticketEntriesTemplate,
{ticketPrices:ticketPrices});
            } else {
                $(this.el).empty();
            }
            this.watchForm();
            return this;
        },
        onChange:function (event) {
            var value = event.currentTarget.value;
            var ticketPriceId = $(event.currentTarget).data("tm-id");
            var modifiedModelEntry = _.find(this.model, function (item) {
                return item.ticketPrice.id == ticketPriceId
            });
            // update model
            if ($.isNumeric(value) && value > 0) {
                modifiedModelEntry.quantity = parseInt(value);
            }
            else {
                delete modifiedModelEntry.quantity;
            }
            // display error messages
            if (value.length > 0 &&
                   (!$.isNumeric(value)  // is a non-number, other than empty string
                        || value <= 0 // is negative
                        || parseFloat(value) != parseInt(value))) { // is not an
integer
                $("#error-input-"+ticketPriceId).empty().append("Please enter a
positive integer value");
                $("#ticket-category-fieldset-"+ticketPriceId).addClass("error");
            } else {
                $("#error-input-"+ticketPriceId).empty();
                $("#ticket-category-fieldset-"+ticketPriceId).removeClass("error");
            }
            // are there any outstanding errors after this update?
            // if yes, disable the input button
            if (
               $("div[id^='ticket-category-fieldset-']").hasClass("error") ||
                   _.isUndefined(modifiedModelEntry.quantity) ) {
              $("input[name='add']").attr("disabled", true)
            } else {
              $("input[name='add']").removeAttr("disabled")
            }
        },
        watchForm: function() {
            if($("#sectionSelectorPlaceholder").length) {
                var self = this;
                $("input[name*='tickets']").each( function(index,element) {
                    if(element.value !== self.formValues[element.name]) {
                        self.formValues[element.name] = element.value;
                        $("input[name='"+element.name+"']").change();
                    }
                });
                this.timerObject = setTimeout(function() {
                    self.watchForm();
                }, this.intervalDuration);
            } else {
                this.onClose();
            }
        },
        onClose: function() {
            if(this.timerObject) {
                clearTimeout(this.timerObject);
                delete this.timerObject;
            }
        }
    });

    var TicketSummaryView = Backbone.View.extend({
        tagName:'tr',
        events:{
            "click i":"removeEntry"
        },
        render:function () {
            var self = this;
            utilities.applyTemplate($(this.el), ticketSummaryViewTemplate,
this.model.bookingRequest);
        },
        removeEntry:function () {
            this.model.bookingRequest.tickets.splice(this.model.index, 1);
        }
    });

    var CreateBookingView = Backbone.View.extend({

        intervalDuration : 100,
        formValues : [],
        events:{
            "click input[name='submit']":"save",
            "change select[id='sectionSelect']":"refreshPrices",
            "keyup #email":"updateEmail",
            "change #email":"updateEmail",
            "click input[name='add']":"addQuantities",
            "click i":"updateQuantities"
        },
        render:function () {

            var self = this;
            $.getJSON(config.baseUrl + "rest/shows/" + this.model.showId, function
(selectedShow) {

                self.currentPerformance = _.find(selectedShow.performances, function
(item) {
                    return item.id == self.model.performanceId;
                });

                var id = function (item) {return item.id;};
                // prepare a list of sections to populate the dropdown
                var sections = _.uniq(_.sortBy(_.pluck(selectedShow.ticketPrices,
'section'), id), true, id);
                utilities.applyTemplate($(self.el), createBookingTemplate, {
                    sections:sections,
                    show:selectedShow,
                    performance:self.currentPerformance});
                self.ticketCategoriesView = new TicketCategoriesView({model:{},
el:$("#ticketCategoriesViewPlaceholder") });
                self.ticketSummaryView = new TicketSummaryView({model:self.model,
el:$("#ticketSummaryView")});
                self.show = selectedShow;
                self.ticketCategoriesView.render();
                self.ticketSummaryView.render();
                $("#sectionSelector").change();
                self.watchForm();
            });
            return this;
        },
        refreshPrices:function (event) {
            var ticketPrices = _.filter(this.show.ticketPrices, function (item) {
                return item.section.id == event.currentTarget.value;
            });
            var sortedTicketPrices = _.sortBy(ticketPrices, function(ticketPrice) {
                return ticketPrice.ticketCategory.description;
            });
            var ticketPriceInputs = new Array();
            _.each(sortedTicketPrices, function (ticketPrice) {
                ticketPriceInputs.push({ticketPrice:ticketPrice});
            });
            this.ticketCategoriesView.model = ticketPriceInputs;
            this.ticketCategoriesView.render();
        },
        save:function (event) {
            var bookingRequest = {ticketRequests:[]};
            var self = this;
            bookingRequest.ticketRequests = _.map(this.model.bookingRequest.tickets,
function (ticket) {
                return {ticketPrice:ticket.ticketPrice.id, quantity:ticket.quantity}
            });
            bookingRequest.email = this.model.bookingRequest.email;
            bookingRequest.performance = this.model.performanceId
            $("input[name='submit']").attr("disabled", true)
            $.ajax({url: (config.baseUrl + "rest/bookings"),
                data:JSON.stringify(bookingRequest),
                type:"POST",
                dataType:"json",
                contentType:"application/json",
                success:function (booking) {
                    this.model = {}
                    $.getJSON(config.baseUrl +'rest/shows/performance/' +
booking.performance.id, function (retrievedPerformance) {
                        utilities.applyTemplate($(self.el),
bookingConfirmationTemplate, {booking:booking, performance:retrievedPerformance })
                    });
                }}).error(function (error) {
                    if (error.status == 400 || error.status == 409) {
                        var errors = $.parseJSON(error.responseText).errors;
                        _.each(errors, function (errorMessage) {
                            $("#request-summary").append('<div class="alert alert-
error"><a class="close" data-dismiss="alert">×</a><strong>Error!</strong> ' +
errorMessage + '</div>')
                        });
                    } else {
                        $("#request-summary").append('<div class="alert alert-
error"><a class="close" data-dismiss="alert">×</a><strong>Error! </strong>An error has
occured</div>')
                    }
                    $("input[name='submit']").removeAttr("disabled");
                })

        },
        addQuantities:function () {
            var self = this;
            _.each(this.ticketCategoriesView.model, function (model) {
                if (model.quantity != undefined) {
                    var found = false;
                    _.each(self.model.bookingRequest.tickets, function (ticket) {
                        if (ticket.ticketPrice.id == model.ticketPrice.id) {
                            ticket.quantity += model.quantity;
                            found = true;
                        }
                    });
                    if (!found) {

self.model.bookingRequest.tickets.push({ticketPrice:model.ticketPrice,
quantity:model.quantity});
                    }
                }
            });
            this.ticketCategoriesView.model = null;
            $('option:selected', 'select').removeAttr('selected');
            this.ticketCategoriesView.render();
            this.updateQuantities();
        },
        updateQuantities:function () {
            // make sure that tickets are sorted by section and ticket category
            this.model.bookingRequest.tickets.sort(function (t1, t2) {
                if (t1.ticketPrice.section.id != t2.ticketPrice.section.id) {
                    return t1.ticketPrice.section.id - t2.ticketPrice.section.id;
                }
                else {
                    return t1.ticketPrice.ticketCategory.id -
t2.ticketPrice.ticketCategory.id;
                }
            });

            this.model.bookingRequest.totals =
_.reduce(this.model.bookingRequest.tickets, function (totals, ticketRequest) {
                return {
                    tickets:totals.tickets + ticketRequest.quantity,
                    price:totals.price + ticketRequest.quantity *
ticketRequest.ticketPrice.price
                };
            }, {tickets:0, price:0.0});

            this.ticketSummaryView.render();
            this.setCheckoutStatus();
        },
        updateEmail:function (event) {
            if ($(event.currentTarget).is(':valid')) {
                this.model.bookingRequest.email = event.currentTarget.value;
                $("#error-email").empty();
            } else {
                $("#error-email").empty().append("Please enter a valid e-mail
address");
                delete this.model.bookingRequest.email;
            }
            this.setCheckoutStatus();
        },
        setCheckoutStatus:function () {
            if (this.model.bookingRequest.totals != undefined &&
this.model.bookingRequest.totals.tickets > 0 && this.model.bookingRequest.email !=
undefined && this.model.bookingRequest.email != '') {
                $('input[name="submit"]').removeAttr('disabled');
            }
            else {
                $('input[name="submit"]').attr('disabled', true);
            }
        },
        watchForm: function() {
            if($("#email").length) {
                var self = this;
                var element = $("#email");
                if(element.val() !== self.formValues["email"]) {
                    self.formValues["email"] = element.val();
                    $("#email").change();
                }
                this.timerObject = setTimeout(function() {
                    self.watchForm();
                }, this.intervalDuration);
            } else {
                this.onClose();
            }
        },
        onClose: function() {
            if(this.timerObject) {
                clearTimeout(this.timerObject);
                delete this.timerObject;
            }
            this.ticketCategoriesView.close();
        }
    });

    return CreateBookingView;
});
```

The code above may be surprising! After all, we said that we were going to add a single view, but instead, we added three! This view makes use of two subviews (`TicketCategoriesView` and `TicketSummaryView`) for re-rendering parts of the main view. Whenever the user changes the current section, the list of available tickets is updated. Whenever the user adds the tickets to the booking, the booking summary is re-rendered. Changes in quantities or the target email may enable or disable the submission button - the booking is validated whenever changes to it are made. We do not create separate modules for the subviews, since they are not referenced outside the module itself.

The booking submission is handled by the `save` method which constructs a JSON object, as required by a POST to `http://localhost:8080/ticket-monster/rest/bookings`, and performs the AJAX call. In case of a successful response, a confirmation view is rendered. On failure, a warning is displayed and the user may continue to edit the form.

The corresponding templates for the views above are shown below:

src/main/webapp/resources/templates/desktop/booking-confirmation.html

```
<div class="row">
    <h2 class="special-title light-font">Booking #<%=booking.id%> confirmed!</h2>
</div>
<div class="row">
    <div class="col-md-6">
        <div class="well">
            <h4 class="page-header">Checkout information</h4>
            <p><strong>Email: </strong><%= booking.contactEmail %></p>
            <p><strong>Event: </strong> <%= performance.event.name %></p>
            <p><strong>Venue: </strong><%= performance.venue.name %></p>
            <p><strong>Date: </strong><%= new
Date(booking.performance.date).toPrettyString() %></p>
            <p><strong>Created on: </strong><%= new
Date(booking.createdOn).toPrettyString() %></p>
        </div>
    </div>
    <div class="col-md-6">
        <div class="well">
            <h4 class="page-header">Ticket allocations</h4>
            <table class="table table-striped table-bordered" style="background-color:
#fffffa;">
                <thead>
                <tr>
                    <th>Ticket #</th>
                    <th>Category</th>
                    <th>Section</th>
                    <th>Row</th>
                    <th>Seat</th>
                </tr>
                </thead>
                <tbody>
            <% $.each(_.sortBy(booking.tickets, function(ticket) {return ticket.id}),
function (i, ticket) { %>
                <tr>
                    <td><%= ticket.id %></td>
                    <td><%=ticket.ticketCategory.description%></td>
                    <td><%=ticket.seat.section.name%></td>
                    <td><%=ticket.seat.rowNumber%></td>
                    <td><%=ticket.seat.number%></td>
                </tr>
                <% }) %>
                </tbody>
            </table>
        </div>
    </div>
</div>
<div class="row" style="padding-bottom:30px;">
    <div class="col-md-2"><a href="#" class="highlight-link">Home</a></div>
```

```
</div>
```

src/main/webapp/resources/templates/desktop/create-booking.html

```
<div class="row">
    <div class="col-md-12">
        <h2 class="special-title light-font"><%=show.event.name%>
            <small><%=show.venue.name%>, <%=new
Date(performance.date).toPrettyString()%></p></small>
        </h2>
    </div>
</div>
<div class="row">
    <div class="col-md-4">
        <div class="well">
            <h3 class="page-header">Select tickets</h3>
            <form class="form-horizontal">
                <div id="sectionSelectorPlaceholder">
                    <div class="form-group">
                        <label class="col-md-3 control-label"
for="sectionSelect"><strong>Section</strong></label>
                        <div class="col-md-9">
                            <select id="sectionSelect" class="form-control">
                                <option value="-1" selected="true">Choose a
section</option>
                                <% _.each(sections, function(section) { %>
                                <option value="<%=section.id%>"><%=section.name%> -
<%=section.description%></option>
                                <% }) %>
                            </select>
                        </div>
                    </div>
                </div>
            </form>
            <div id="ticketCategoriesViewPlaceholder"></div>
        </div>
    </div>
    <div id="request-summary" class="col-md-5 col-md-offset-1">
        <div class="well">
            <h3 class="page-header">Order summary</h3>
            <div id="ticketSummaryView" class="row"/>
            <h3 class="page-header">Checkout</h3>
            <div class="row">
                <div class="col-md-12">
                    <form>
                        <div class="form-group">
                            <input type='email' id="email" class="form-control"
placeholder="Email" required/>
                            <p class="help-block error-notification"  id="error-
email"></p>
```

```
                        </div>
                        <div class="form-group">
                            <input type='button' class="btn btn-primary" name="submit"
value="Checkout"
                                   disabled="true"/>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    </div>
</div>
```

src/main/webapp/resources/templates/desktop/ticket-categories.html

```
<% if (ticketPrices.length > 0) { %>
<form class="form-horizontal">
    <% _.each(ticketPrices, function(ticketPrice) { %>
    <div class="form-group" id="ticket-category-fieldset-<%=ticketPrice.id%>">
        <label class="col-md-3 control-
label"><strong><%=ticketPrice.ticketCategory.description%></strong></label>

        <div class="col-md-9">
            <div class="input-group">
                <input class="form-control col-md-6" rel="tooltip" title="Enter value"
                       data-tm-id="<%=ticketPrice.id%>"
                       placeholder="Number of tickets"
                       name="tickets-<%=ticketPrice.ticketCategory.id%>"/>
                <span class="input-group-addon">@ $<%=ticketPrice.price%></span>

                <p class="help-block" id="error-input-<%=ticketPrice.id%>"></p>
            </div>
        </div>
    </div>
    <% }) %>

<p>&nbsp;</p>

<div class="form-group">
    <div class="col-md-offset-2">
        <input type="button" class="btn btn-primary" disabled="true" name="add"
value="Add tickets"/>
    </div>
</div>
</div>
</form>
<% } %>
```

src/main/webapp/resources/templates/desktop/ticket-summary-view.html

```
<div class="col-md-12">
    <% if (tickets.length>0) { %>
    <table class="table table-bordered table-condensed" style="background-color:
#fffffa;">
        <thead>
        <tr>
            <th colspan="7"><strong>Requested tickets</strong></th>
        </tr>
        <tr>
            <th>Section</th>
            <th>Category</th>
            <th>Quantity</th>
            <th>Price</th>
            <th></th>
        </tr>
        </thead>
        <tbody id="ticketRequestSummary">
        <% _.each(tickets, function (ticketRequest, index, tickets) { %>
        <tr>
            <td><%= ticketRequest.ticketPrice.section.name %></td>
            <td><%= ticketRequest.ticketPrice.ticketCategory.description %></td>
            <td><%= ticketRequest.quantity %></td>
            <td>$<%=ticketRequest.ticketPrice.price%></td>
            <td><span class="glyphicon glyphicon-trash" data-index='<%= index
%>'/></td>
        </tr>
        <% }); %>
        </tbody>
    </table>
    <p/>
    <div class="row">
        <div class="col-md-5"><strong>Total ticket count:</strong> <%= totals.tickets
%></div>
        <div class="col-md-5"><strong>Total price:</strong> $<%=totals.price%></div>
    </div>
    <% } else { %>
    No tickets requested.
    <% } %>
</div>
```

Finally, once the view is available, we can add it’s corresponding routing rule:

src/main/webapp/resources/js/app/router/desktop/router.js

```
/**
 * A module for the router of the desktop application
 */
define("router", [
    ...
    'app/views/desktop/create-booking',
    ...
],function (
            ...
            CreateBookingView,
            ...
            ) {

    var Router = Backbone.Router.extend({
        ...
        routes:{
            ...
            "book/:showId/:performanceId":"bookTickets",
        },
        ...
        bookTickets:function (showId, performanceId) {
            var createBookingView =
                new CreateBookingView({
                    model:{ showId:showId,
                            performanceId:performanceId,
                            bookingRequest:{tickets:[]}},
                            el:$("#content")
                           });
            utilities.viewManager.showView(createBookingView);
        }
    }
    ...
);
```

This concludes the implementation of the booking use case. We started by listing the available events, continued by selecting a venue and performance time, and ended by choosing tickets and completing the order.

The other use cases: a booking starting from venues and view existing bookings are conceptually similar, so you can just copy the logic for the following routes from `src/main/webapp/resources/js/app/routers/desktop/router.js`:

- `venues`
- `venues/:id`
- `bookings`
- `bookings/:id`

Finally, copy the following files in the `src/main/webapp/resources/js/app/models`, `src/main/webapp/resources/js/app/collections`, `src/main/webapp/resources/js/app/views/desktop` and `src/main/webapp/resources/templates`:

- `src/main/webapp/resources/js/app/models/booking.js`

- `src/main/webapp/resources/js/app/models/venue.js`
- `src/main/webapp/resources/js/app/collections/bookings.js`
- `src/main/webapp/resources/js/app/collections/venues.js`
- `src/main/webapp/resources/js/app/views/desktop/bookings.js`
- `src/main/webapp/resources/js/app/views/desktop/booking-detail.js`
- `src/main/webapp/resources/js/app/views/desktop/venues.js`
- `src/main/webapp/resources/js/app/views/desktop/venue-detail.js`
- `src/main/webapp/resources/templates/desktop/booking-details.html`
- `src/main/webapp/resources/templates/desktop/booking-table.html`
- `src/main/webapp/resources/templates/desktop/venues.html`
- `src/main/webapp/resources/templates/desktop/venue-detail.html`
- `src/main/webapp/resources/templates/desktop/venue-event-description.html`

Mobile view

The mobile version of the application uses approximately the same architecture as the desktop version. Any differences are due to the functional changes in the mobile version and the use of jQuery mobile.

Setting up the structure

The first step in implementing our solution is to copy the CSS and JavaScript libraries to `resources/css` and `resources/js/libs`:

require.js
: AMD support, along with the plugin:
  - text - for loading text files, in our case the HTML templates

jQuery
: general purpose library for HTML traversal and manipulation

Underscore
: JavaScript utility library (and a dependency of Backbone)

Backbone
: Client-side MVC framework

jQuery Mobile
: user interface system for mobile devices;

(If you have already built the desktop application, some files may already be in place.)

For mobile clients, the main page will display the mobile version of the application, by loading the mobile AMD module of the application. Let us create it.

src/main/webapp/resources/js/configurations/mobile.js

```
/**
 * Shortcut alias definitions - will come in handy when declaring dependencies
 * Also, they allow you to keep the code free of any knowledge about library
 * locations and versions
 */
require.config({
    baseUrl:"resources/js",
    paths: {
        jquery:'libs/jquery-2.0.3',
        jquerymobile:'libs/jquery.mobile-1.4.2',
        text:'libs/text',
        underscore:'libs/underscore',
        backbone: 'libs/backbone',
        utilities: 'app/utilities',
        router:'app/router/mobile/router'
    },
    // We shim Backbone.js and Underscore.js since they don't declare AMD modules
    shim: {
        'backbone': {
            deps: ['underscore', 'jquery'],
            exports: 'Backbone'
        },

        'underscore': {
            exports: '_'
        }
    }
});

define("configuration", function() {
    if (window.TicketMonster != undefined && TicketMonster.config != undefined) {
        return {
            baseUrl: TicketMonster.config.baseRESTUrl
        };
    } else {
        return {
            baseUrl: ""
        };
    }
});

define("initializer", [
    'jquery',
    'utilities',
    'text!../templates/mobile/main.html'
], function ($,
             utilities,
             MainTemplate) {
    // Configure jQuery to append timestamps to requests, to bypass browser caches
    // Important for MSIE
    $.ajaxSetup({cache:false});
    $('head').append('<link rel="stylesheet" href="resources/css/jquery.mobile-
1.4.2.css"/>');
    $('head').append('<link rel="stylesheet" href="resources/css/m.screen.css"/>');
    // Bind to mobileinit before loading jQueryMobile
    $(document).bind("mobileinit", function () {
        // Prior to creating and starting the router, we disable jQuery Mobile's own
routing mechanism
        $.mobile.hashListeningEnabled = false;
        $.mobile.linkBindingEnabled = false;
        $.mobile.pushStateEnabled = false;

        // Fix jQueryMobile header and footer positioning issues for iOS.
        // See: https://github.com/jquery/jquery-mobile/issues/4113 and
        // https://github.com/jquery/jquery-mobile/issues/5532
        $(document).on('blur', 'input, textarea, select', function() {
            setTimeout(function() {
            window.scrollTo(document.body.scrollLeft, document.body.scrollTop);
            }, 0);
        });

        utilities.applyTemplate($('body'), MainTemplate);
    });
    // Then (load jQueryMobile and) start the router to finally start the app
    require(['router']);
});

// Now we declare all the dependencies
// This loads and runs the 'initializer' module.
require(['initializer']);
```

In this application, we combine Backbone and jQuery Mobile. Each framework has its own strengths; jQuery Mobile provides UI components and touch support, whilst Backbone provides MVC support. There is some overlap between the two, as jQuery Mobile provides its own navigation mechanism which we disable.

We also define a `configuration` module which allows the customization of the base URLs for RESTful invocations. This module does not play any role in the mobile web version. We will come to it, however, when discussing hybrid applications.

We also define a special initializer module (`initializer`) that, when loaded, adds the stylesheets and applies the template for the general structure of the page in the `body` element. In the initializer module we make customizations in order to get the two frameworks working together - disabling the jQuery Mobile navigation. Let us add the template definition for the template loaded by the initializer module.

src/main/webapp/resources/templates/mobile/main.html

```
<!--
    The main layout of the page - contains the menu and the 'content' &lt;div/&gt; in
which all the
    views will render the content.
-->
<div id="container" data-role="page" data-ajax="false"></div>
```

Copy over the `m.screen.css` file referenced in the `initializer` module, from the project sources, to the appropriate location in the workspace.

Next, we create the application router.

src/main/webapp/resources/js/app/router/mobile/router.js

```
/**
 * A module for the router of the mobile application.
 *
 */
define("router",[
    'jquery',
    'jquerymobile',
    'underscore',
    'utilities'
],function ($,
            jqm,
            _,
            utilities) {

    /**
     * The Router class contains all the routes within the application - i.e. URLs and
the actions
     * that will be taken as a result.
     *
     * @type {Router}
     */
    var Router = Backbone.Router.extend({
        initialize: function() {
            //Begin dispatching routes
            Backbone.history.start();
        },
        execute : function(callback, args) {
            $.mobile.loading("show");
            window.setTimeout(function() {
                if (callback) {
                    callback.apply(this, args);
                }
                $.mobile.loading("hide");
            }, 300);
        }
    });

    // Create a router instance
    var router = new Router();

    return router;
});
```

In the router code we add the `execute` method to the router for handling transitions between routes. Here, we will display the jQuery Mobile loader widget before displaying any Backbone view, and then hide it once the view is rendered.

Next, we need to create a first page.

The landing page

The first page in our application is the landing page. First, we add the template for it:

src/main/webapp/resources/templates/mobile/home-view.html

```
<div data-role="header">
    <h3>Ticket Monster</h3>
</div>
<div class="ui-content">
    <img src="resources/img/rhjb_eap_logo.png" />
    <h4>Find events</h4>
    <ul data-role="listview">
        <li>
            <a href="#events">By Category</a>
        </li>
        <li>
            <a href="#venues">By Location</a>
        </li>
    </ul>
</div>
```

Now we have to add the page to the router:

src/main/webapp/resources/js/app/router/mobile/router.js

```
/**
 * A module for the router of the mobile application.
 *
 */
define("router",[
    ...
    'text!../templates/mobile/home-view.html'
],function (
        ...
        HomeViewTemplate) {

    ...
    var Router = Backbone.Router.extend({
        ...
        routes:{
            "":"home"
        },
        ...
        home:function () {
            utilities.applyTemplate($("#container"), HomeViewTemplate);
            $("#container").enhanceWithin();
        }
    });
    ...
});
```

Because jQuery Mobile navigation is disabled, we must tell jQuery Mobile explicitly to enhance the page content in order to create the mobile view. Here, we enhance the page using the `enhanceWithin` method, to ensure that the page gets the appropriate look and feel.

The events view

First, we display a list of events (just as in the desktop view). Since mobile interfaces are more constrained, we will just show a simple list view:

src/main/webapp/resources/js/app/views/mobile/events.js

```
define([
    'utilities',
    'text!../../../../templates/mobile/events.html'
], function (
    utilities,
    eventsView) {

    var EventsView = Backbone.View.extend({
        render:function () {
            var categories = _.uniq(
                _.map(this.model.models, function(model){
                    return model.get('category');
                }), false, function(item){
                    return item.id;
                });
            utilities.applyTemplate($(this.el), eventsView,  {categories:categories,
model:this.model});
            $(this.el).enhanceWithin();
            return this;
        }
    });
    return EventsView;
});
```

As you can see, the view is very similar to the desktop view, the main difference being the explicit hint to jQuery mobile through the `pagecreate` event invocation.

Next, we add the template for rendering the view:

src/main/webapp/resources/templates/mobile/events.html

```
<div data-role="header">
    <a href="#" class="ui-btn ui-icon-home ui-btn-icon-left">Home</a>
    <h3>Categories</h3>
</div>
<div class="ui-content">
    <div id="itemMenu" data-role="collapsible-set" data-inset="false">
        <%
        _.each(categories, function (category) {
        %>
        <div data-role="collapsible">
            <h2><%= category.description %></h2>
            <ul id="categoryMenu" data-role="listview" data-inset="true">
            <%
            _.each(model.models, function (model) {
                if (model.get('category').id == category.id) {
                %>
                <li>
                    <a
href="#events/<%=model.attributes.id%>"><%=model.attributes.name%></a>
                </li>
                <% }
            });
            %>
            </ul>
        </div>
        <% }); %>
    </div>
</div>
```

And finally, we need to instruct the router to invoke the page:

src/main/webapp/resources/js/app/router/mobile/router.js

```
/**
 * A module for the router of the desktop application.
 *
 */
define("router",[
    ...
    'app/collections/events',
    ...
    'app/views/mobile/events'
    ...
],function (
    ...,
    Events,
    ...,
    EventsView,
    ...) {

    ...
    var Router = Backbone.Router.extend({
        ...
        routes:{
            ...
            "events":"events"
            ...
        },
        ...
        events:function () {
            var events = new Events;
            var eventsView = new EventsView({model:events, el:$("#container")});
            events.on("reset", function() {
                utilities.viewManager.showView(eventsView);
            }).fetch({
                reset : true,
                error : function() {
                    utilities.displayAlert("Failed to retrieve events from the
TicketMonster server.");
                }
            });
        }
        ...
    });
    ...
});
```

Just as in the case of the desktop application, the list of events will be accessible at `#events` (i.e. `http://localhost:8080/ticket-monster/#events`).

Displaying an individual event

Now, we create the view to display an event:

src/main/webapp/resources/js/app/views/mobile/event-detail.js

```
define([
    'utilities',
    'require',
    'configuration',
    'text!../../../../templates/mobile/event-detail.html',
    'text!../../../../templates/mobile/event-venue-description.html'
], function (
    utilities,
    require,
    config,
    eventDetail,
    eventVenueDescription) {

    var EventDetailView = Backbone.View.extend({
        events:{
            "click a[id='bookButton']":"beginBooking",
            "change select[id='showSelector']":"refreshShows",
            "change select[id='performanceTimes']":"performanceSelected",
            "change select[id='dayPicker']":'refreshTimes'
        },
        render:function () {
            $(this.el).empty()
            utilities.applyTemplate($(this.el), eventDetail, _.extend({},
this.model.attributes, config));
            $(this.el).enhanceWithin();
            $("#bookButton").addClass("ui-disabled");
            var self = this;
            $.getJSON(config.baseUrl + "rest/shows?event=" + this.model.get('id'),
function (shows) {
                self.shows = shows;
                $("#showSelector").empty().append("<option data-
placeholder='true'>Choose a venue ...</option>");
                $.each(shows, function (i, show) {
                    $("#showSelector").append("<option value='" + show.id + "'>" +
show.venue.address.city + " : " + show.venue.name + "</option>");
                });
                $("#showSelector").selectmenu('refresh', true)
                $("#dayPicker").selectmenu('disable')
                $("#dayPicker").empty().append("<option data-placeholder='true'>Choose
a show date ...</option>")
                $("#performanceTimes").selectmenu('disable')
                $("#performanceTimes").empty().append("<option data-
placeholder='true'>Choose a show time ...</option>")
            });
            $("#dayPicker").empty();
            $("#dayPicker").selectmenu('disable');
            $("#performanceTimes").empty();
            $("#performanceTimes").selectmenu('disable');
            $(this.el).enhanceWithin();
            return this;
        },
        performanceSelected:function () {
            if ($("#performanceTimes").val() != 'Choose a show time ...') {
                $("#bookButton").removeClass("ui-disabled")
            } else {
                $("#bookButton").addClass("ui-disabled")
            }
        },
        beginBooking:function () {
            require('router').navigate('book/' + $("#showSelector
option:selected").val() + '/' + $("#performanceTimes").val(), true)
        },
        refreshShows:function (event) {

            var selectedShowId = event.currentTarget.value;

            if (selectedShowId != 'Choose a venue ...') {
                var selectedShow = _.find(this.shows, function (show) {
                    return show.id == selectedShowId
                });
                this.selectedShow = selectedShow;
                var times = _.uniq(_.sortBy(_.map(selectedShow.performances, function
(performance) {
                    return (new Date(performance.date).withoutTimeOfDay()).getTime()
                }), function (item) {
                    return item
                }));
                utilities.applyTemplate($("#eventVenueDescription"),
eventVenueDescription, _.extend({},{venue:selectedShow.venue},config));
                $("#detailsCollapsible").show()
                $("#dayPicker").removeAttr('disabled')
                $("#performanceTimes").removeAttr('disabled')
                $("#dayPicker").empty().append("<option data-placeholder='true'>Choose
a show date ...</option>")
                _.each(times, function (time) {
                    var date = new Date(time)
                    $("#dayPicker").append("<option value='" + date.toYMD() + "'>" +
date.toPrettyStringWithoutTime() + "</option>")
                });
                $("#dayPicker").selectmenu('refresh')
                $("#dayPicker").selectmenu('enable')
                this.refreshTimes()
            } else {
                $("#detailsCollapsible").hide()
                $("#eventVenueDescription").empty()
                $("#dayPicker").empty()
                $("#dayPicker").selectmenu('disable')
                $("#performanceTimes").empty()
                $("#performanceTimes").selectmenu('disable')
            }


        },
        refreshTimes:function () {
            var selectedDate = $("#dayPicker").val();
            $("#performanceTimes").empty().append("<option data-
placeholder='true'>Choose a show time ...</option>")
            if (selectedDate) {
                $.each(this.selectedShow.performances, function (i, performance) {
                    var performanceDate = new Date(performance.date);
                    if (_.isEqual(performanceDate.toYMD(), selectedDate)) {
                        $("#performanceTimes").append("<option value='" +
performance.id + "'>" + performanceDate.getHours().toZeroPaddedString(2) + ":" +
performanceDate.getMinutes().toZeroPaddedString(2) + "</option>")
                    }
                })
                $("#performanceTimes").selectmenu('enable')
            }
            $("#performanceTimes").selectmenu('refresh')
            this.performanceSelected()
        }

    });

    return EventDetailView;
});
```

Once again, this is very similar to the desktop version. Now we add the page templates:

src/main/webapp/resources/templates/mobile/event-detail.html

```
<div data-role="header" data-position="fixed">
    <a href="#" class="ui-btn ui-icon-home ui-btn-icon-left">Home</a>
    <h3>Book tickets</h3>
</div>
<div class="ui-content">
    <h3><%=name%></h3>
    <img width='100px' src='<%=baseUrl%>rest/media/<%=mediaItem.id%>'/>
    <p><%=description%></p>
    <div class="ui-field-contain">
        <label for="showSelector"><strong>Where</strong></label>
        <select id='showSelector' data-mini='true'/>
    </div>

    <div data-role="collapsible" data-content-theme="c" style="display: none;"
         id="detailsCollapsible">
        <h3>Venue details</h3>

        <div id="eventVenueDescription">
        </div>
    </div>

    <div data-role='fieldcontain'>
        <fieldset data-role='controlgroup'>
            <legend><strong>When</strong></legend>
            <label for="dayPicker">When:</label>
            <select id='dayPicker' data-mini='true'/>
            <label for="performanceTimes">When:</label>
            <select id="performanceTimes" data-mini='true'/>

        </fieldset>
    </div>

</div>
<div data-role="footer" data-position="fixed">
    <div class="ui-grid-b">
        <div class="ui-block-a"></div>
        <div class="ui-block-b"></div>
        <div class="ui-block-c">
            <a id='bookButton' class="ui-btn ui-btn-b ui-icon-check ui-btn-icon-left
block-btn">Book</a>
        </div>
    </div>
</div>
```

src/main/webapp/resources/templates/mobile/event-venue-description.html

```
<img width="100" src="<%=baseUrl%>rest/media/<%=venue.mediaItem.id%>"/></p>
<%= venue.description %>
<address>
    <p><strong>Address:</strong></p>
    <p><%= venue.address.street %></p>
    <p><%= venue.address.city %>, <%= venue.address.country %></p>
</address>
```

Finally, we add this to the router, explicitly indicating to jQuery Mobile that a transition has to take place after the view is rendered - in order to allow the page to render correctly after it has been invoked from the listview.

src/main/webapp/resources/js/app/router/mobile/router.js

```
/**
 * A module for the router of the desktop application.
 *
 */
define("router",[
    ...
    'app/models/event',
    ...
    'app/views/mobile/event-detail'
    ...
],function (
    ...,
    Event,
    ...,
    EventDetailView,
    ...) {

    ...
    var Router = Backbone.Router.extend({
        ...
        routes:{
            ...
            "events/:id":"eventDetail",
            ...
        },
        ...
        eventDetail:function (id) {
            var model = new Event({id:id});
            var eventDetailView = new EventDetailView({model:model,
el:$("#container")});
            model.on("change",
                function () {
                    utilities.viewManager.showView(eventDetailView);
                    $("body").pagecontainer("change", "#container",
{transition:'slide', changeHash:false});
                }).fetch({
                    error : function() {
                        utilities.displayAlert("Failed to retrieve the event from the
TicketMonster server.");
                    }
                });
        }
        ...
    });
    ...
});
```

Just as the desktop version, the mobile event detail view allows users to choose a venue and a performance time. The next step is to allow the user to book some tickets.

Booking tickets

The views to book tickets are simpler than the desktop version. Users can select a section and enter the number of tickets for each category however, there is no way to add or remove tickets from an order. Once the form is filled out, the user can only submit it.

First, we create the views:

src/main/webapp/resources/js/app/views/mobile/create-booking.js

```
define([
    'utilities',
    'configuration',
    'require',
    'text!../../../../templates/mobile/booking-details.html',
    'text!../../../../templates/mobile/create-booking.html',
    'text!../../../../templates/mobile/confirm-booking.html',
    'text!../../../../templates/mobile/ticket-entries.html',
    'text!../../../../templates/mobile/ticket-summary-view.html'
], function (
    utilities,
    config,
    require,
    bookingDetailsTemplate,
    createBookingTemplate,
    confirmBookingTemplate,
    ticketEntriesTemplate,
    ticketSummaryViewTemplate) {

    var TicketCategoriesView = Backbone.View.extend({
        id:'categoriesView',
        events:{
            "change input":"onChange"
        },
        render:function () {
            var views = {};

            if (this.model != null) {
                var ticketPrices = _.map(this.model, function (item) {
                    return item.ticketPrice;
                });
                utilities.applyTemplate($(this.el), ticketEntriesTemplate,
{ticketPrices:ticketPrices});
            } else {
                $(this.el).empty();
            }
            return this;
        },
        onChange:function (event) {
            var value = event.currentTarget.value;
            var ticketPriceId = $(event.currentTarget).data("tm-id");
            var modifiedModelEntry = _.find(this.model, function(item) { return
item.ticketPrice.id == ticketPriceId});
            if ($.isNumeric(value) && value > 0) {
                modifiedModelEntry.quantity = parseInt(value);
            }
            else {
                delete modifiedModelEntry.quantity;
            }
        }
    });

     var TicketSummaryView = Backbone.View.extend({
        render:function () {
            utilities.applyTemplate($(this.el), ticketSummaryViewTemplate,
this.model.bookingRequest)
```

```
        }
    });

    var CreateBookingView = Backbone.View.extend({

        currentView: "CreateBooking",
        intervalDuration : 100,
        formValues : [],
        events:{
            "click a[id='confirmBooking']":"checkout",
            "change select":"refreshPrices",
            "change input[type='number']":"updateForm",
            "change input[name='email']":"updateForm",
            "click a[id='saveBooking']":"saveBooking",
            "click a[id='goBack']":"back",
            "click a[data-action='delete']":"deleteBooking"
        },
        render: function() {
            if (this.currentView === "CreateBooking") {
                this.renderCreateBooking();
            } else if(this.currentView === "ConfirmBooking") {
                this.renderConfirmBooking();
            }
            return this;
        },
        renderCreateBooking:function () {

            var self = this;

            $.getJSON(config.baseUrl + "rest/shows/" + this.model.showId, function
(selectedShow) {
                self.model.performance = _.find(selectedShow.performances, function
(item) {
                    return item.id == self.model.performanceId;
                });
                self.model.email = self.model.email || "";
                var id = function (item) {return item.id;};
                // prepare a list of sections to populate the dropdown
                var sections = _.uniq(_.sortBy(_.pluck(selectedShow.ticketPrices,
'section'), id), true, id);

                utilities.applyTemplate($(self.el), createBookingTemplate, {
show:selectedShow,
                    performance:self.model.performance,
                    sections:sections,
                    email:self.model.email});
                $(self.el).enhanceWithin();
                self.ticketCategoriesView = new TicketCategoriesView({model:{},
el:$("#ticketCategoriesViewPlaceholder") });
                self.model.show = selectedShow;
                self.ticketCategoriesView.render();
                $('a[id="confirmBooking"]').addClass('ui-disabled');
                $("#sectionSelector").change();
                self.watchForm();
            });

        },
        refreshPrices:function (event) {
            if (event.currentTarget.value != "Choose a section") {
                var ticketPrices = _.filter(this.model.show.ticketPrices, function
(item) {
                    return item.section.id == event.currentTarget.value;
                });
                var ticketPriceInputs = new Array();
                _.each(ticketPrices, function (ticketPrice) {
                    var model = {};
                    model.ticketPrice = ticketPrice;
                    ticketPriceInputs.push(model);
                });
                $("#ticketCategoriesViewPlaceholder").show();
                this.ticketCategoriesView.model = ticketPriceInputs;
                this.ticketCategoriesView.render();
                $(this.el).enhanceWithin();
            } else {
                $("#ticketCategoriesViewPlaceholder").hide();
                this.ticketCategoriesView.model = new Array();
                this.updateForm();
            }
        },
        checkout:function () {
            var savedTicketRequests = this.model.bookingRequest.tickets =
this.model.bookingRequest.tickets || [];
            _.each(this.ticketCategoriesView.model, function(newTicketRequest){
                var matchingRequest = _.find(savedTicketRequests,
function(ticketRequest) {
                    return ticketRequest.ticketPrice.id ==
newTicketRequest.ticketPrice.id;
                });
                if(newTicketRequest.quantity) {
                    if(matchingRequest) {
                        matchingRequest.quantity += newTicketRequest.quantity;
                    } else {
                        savedTicketRequests.push(newTicketRequest);
                    }
                }
            });
            this.model.bookingRequest.totals =
this.computeTotals(this.model.bookingRequest.tickets);
            this.currentView = "ConfirmBooking";
            this.render();
        },
        updateForm:function () {
            var valid = true;
            this.model.email = $("input[type='email']").val();
            $("input[type='number']").each(function(idx,element) {
                var quantity = $(this).val();
                if(quantity.length > 0 &&
                    (!$.isNumeric(quantity)  // is a non-number, other than empty
string
                        || quantity <= 0 // is negative
                        || parseFloat(quantity) != parseInt(quantity))) {
                    $("#error-" + element.id).empty().append("Should be a positive
number.");
                    valid = false;
                } else {
                    $("#error-" + element.id).empty();
                }
            });
            try {
                var validElements = document.querySelectorAll(":valid");
                var $email = $("#email");
                var emailElem = $email.get(0);
                var validEmail = false;
                for (var ctr=0; ctr < validElements.length; ctr++) {
                    if (emailElem === validElements[ctr]) {
                        validEmail = true;
                    }
                }
                if(validEmail) {
                    this.model.email = $email.val();
                    $("#error-email").empty();
                } else {
                    $("#error-email").empty().append("Please enter a valid e-mail
address");
                    delete this.model.email;
                    valid = false;
                }
            }
            catch(e) {
                // For browsers like IE9 that do fail on querySelectorAll for CSS
pseudo selectors,
                // we use the regex defined in the HTML5 spec.
                var emailRegex = new RegExp("[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-
Z0-9-]+(?:\.[a-zA-Z0-9-]+)*");
                var emailValue = $("#email").val();
                if(emailRegex.test(emailValue)) {
                    this.model.email = emailValue;
                    $("#error-email").empty();
                } else {
                    $("#error-email").empty().append("Please enter a valid e-mail
address");
                    delete this.model.email;
                    valid = false;
                }
            }
            var totals = this.computeTotals(this.ticketCategoriesView.model);
            if (totals.tickets > 0 && valid) {
                $('a[id="confirmBooking"]').removeClass('ui-disabled');
            } else {
                $('a[id="confirmBooking"]').addClass('ui-disabled');
            }
        },
        computeTotals: function(ticketRequestCollection) {
            var totals = _.reduce(ticketRequestCollection, function (partial, model) {
                if (model.quantity != undefined) {
                    partial.tickets += model.quantity;
                    partial.price += model.quantity * model.ticketPrice.price;
                    return partial;
                } else {
                    return partial;
                }
            }, {tickets:0, price:0.0});
            return totals;
        },
        renderConfirmBooking:function () {
            utilities.applyTemplate($(this.el), confirmBookingTemplate, this.model);
            this.ticketSummaryView = new TicketSummaryView({model:this.model,
el:$("#ticketSummaryView")});
            this.ticketSummaryView.render();
            $(this.el).enhanceWithin();
            if (this.model.bookingRequest.totals.tickets > 0) {
                $('a[id="saveBooking"]').removeClass('ui-disabled');
            } else {
                $('a[id="saveBooking"]').addClass('ui-disabled');
            }
            return this;
        },
        back:function () {
            this.currentView = "CreateBooking";
            this.render();
        },
        saveBooking:function (event) {
            var bookingRequest = {ticketRequests:[]};
            var self = this;
            _.each(this.model.bookingRequest.tickets, function (model) {
                if (model.quantity != undefined) {

bookingRequest.ticketRequests.push({ticketPrice:model.ticketPrice.id,
quantity:model.quantity})
                }
            });

            bookingRequest.email = this.model.email;
            bookingRequest.performance = this.model.performanceId;
            $.ajax({url:(config.baseUrl + "rest/bookings"),
                data:JSON.stringify(bookingRequest),
                type:"POST",
                dataType:"json",
                contentType:"application/json",
                success:function (booking) {
                    utilities.applyTemplate($(self.el), bookingDetailsTemplate,
booking);
                    $(self.el).enhanceWithin();
                }}).error(function (error) {
                    try {
                        var response = JSON.parse(error.responseText);
                        var displayMessage = "";
                        if(response && response.errors) {
                            var errors = response.errors;
                            for(var idx = 0; idx < errors.length; idx++) {
                                displayMessage += errors[idx] + "\n";
                            }
                            alert(displayMessage);
                        } else {
                            alert("Failed to perform the bookng.");
                        }
                    } catch (e) {
                        alert("Failed to perform the bookng.");
                    }
                });
        },
        deleteBooking: function(event) {
            var deletedIdx = $(event.currentTarget).data("ticketpriceid");
            this.model.bookingRequest.tickets =
_.reject(this.model.bookingRequest.tickets, function(ticketRequest) {
                return ticketRequest.ticketPrice.id == deletedIdx;
            });
            this.model.bookingRequest.totals =
this.computeTotals(this.model.bookingRequest.tickets);
            this.renderConfirmBooking();
            return false;
        },
        watchForm: function() {
            if($("#sectionSelect").length) {
                var self = this;
                $("input").each( function(index,element) {
                    if(element.value !== self.formValues[element.id]) {
                        self.formValues[element.id] = element.value;
                        $("input[id='"+element.id+"']").change();
                    }
                });
                this.timerObject = setTimeout(function() {
                    self.watchForm();
                }, this.intervalDuration);
            } else {
                this.onClose();
            }
        },
        onClose: function() {
            if(this.timerObject) {
                clearTimeout(this.timerObject);
                delete this.timerObject;
            }
        }
    });
    return CreateBookingView;
});
```

The views follow the structure the desktop application, except that the summary view is not rendered inline but after a page transition.

Next, we create the page fragment templates. First, the actual page:

src/main/webapp/resources/templates/mobile/create-booking.html

```
<div data-role="header" data-position="fixed">
    <a href="#" class="ui-btn ui-icon-home ui-btn-icon-left">Home</a>
    <h1>Book tickets</h1>
</div>
<div class="ui-content">
    <p>
       <h3><%=show.event.name%></h3>
    </p>
    <p>
      <%=show.venue.name%>
    <p>

    <p>
      <small><%=new Date(performance.date).toPrettyString()%></small>
    </p>
    <div id="sectionSelectorPlaceholder">
        <div class="ui-field-contain">
            <label for="sectionSelect">Section</label>
            <select id="sectionSelect">
                <option value="-1" selected>Choose a section</option>
                <% _.each(sections, function(section) { %>
                <option value="<%=section.id%>"><%=section.name%> -
<%=section.description%></option>
                <% }) %>
            </select>
        </div>

    </div>
    <div id="ticketCategoriesViewPlaceholder" style="display:none;"></div>

    <div class="fieldcontain">
```

```
        <label>Contact email</label>
        <input type='email' id='email' name='email' required placeholder="Email"
value="<%=email%>" />
        <span id="error-email" class="error" />
    </div>
</div>

<div data-role="footer" data-position="fixed">
    <div class="ui-grid-a">
        <div class="ui-block-a"><a href="#" class="ui-btn ui-icon-delete ui-btn-icon-
left block-btn">Cancel</a></div>
        <div class="ui-block-b"><a id="confirmBooking" class="ui-btn ui-btn-b ui-icon-
check ui-btn-icon-left block-btn" disabled>Checkout</a></div>
    </div>
</div>
```

Next, the fragment that contains the input form for tickets, which is re-rendered whenever the section is changed:

src/main/webapp/resources/templates/mobile/ticket-entries.html

```
<% if (ticketPrices.length > 0) { %>
    <form name="ticketCategories" id="ticketCategories">
    <h4>Select tickets by category</h4>
    <% _.each(ticketPrices, function(ticketPrice) { %>
      <div id="ticket-category-input-<%=ticketPrice.id%>"/>

      <fieldset class="ui-field-contain">
         <label for="ticket-
<%=ticketPrice.id%>"><%=ticketPrice.ticketCategory.description%>($<%=ticketPrice.price
%>)</label>
         <input id="ticket-<%=ticketPrice.id%>" data-tm-id="<%=ticketPrice.id%>"
type="number" min="0" placeholder="Enter value"
               name="tickets"/>
         <span id="error-ticket-<%=ticketPrice.id%>" class="error" />
      </fieldset>
   <% }) %>
   </form>
<% } %>
```

Before submitting the request to the server, the order is confirmed:

src/main/webapp/resources/templates/mobile/confirm-booking.html

```
<div data-role="header" data-position="fixed">
    <a href="#" class="ui-btn ui-icon-home ui-btn-icon-left">Home</a>
    <h1>Confirm order</h1>
</div>
<div class="ui-content">
    <h3><%=show.event.name%></h3>
    <p><%=show.venue.name%></p>
    <p><small><%=new Date(performance.date).toPrettyString()%></small></p>
    <p><strong>Buyer:</strong>  <emphasis><%=email%></emphasis></p>
    <div id="ticketSummaryView"/>

</div>

<div data-role="footer" data-position="fixed">
    <div class="ui-grid-b">
        <div class="ui-block-a"><a id="cancel" href="#" class="ui-btn ui-icon-delete
ui-btn-icon-left block-btn">Cancel</a></div>
        <div class="ui-block-b"><a id="goBack" class="ui-btn ui-icon-back ui-btn-icon-
left block-btn">Back</a></div>
        <div class="ui-block-c"><a id="saveBooking" class="ui-btn ui-btn-b ui-icon-
check ui-btn-icon-left block-btn">Buy!</a></div>
    </div>
</div>
```

The confirmation page contains a summary subview:

src/main/webapp/resources/templates/mobile/ticket-summary-view.html

```
<ul data-role="listview" data-split-icon="delete" data-split-theme="c" data-
inset="true">
    <% _.each(tickets, function(model) { %>
    <li>
        <a class="readonly-listview">
            <p><strong>Section</strong> <%= model.ticketPrice.section.name %></p>
            <p><strong>Category</strong> <%=
model.ticketPrice.ticketCategory.description %></p>
            <p><strong>Price</strong> <%= model.ticketPrice.price %></p>
            <p><strong>Quantity</strong> <%= model.quantity %></p>
        </a>
        <a href="#" data-action="delete" data-ticketpriceid="<%= model.ticketPrice.id
%>"></a>
    </li>
    <% }); %>
</ul>
<div>
    <h4>Totals</h4>
    <p><strong>Total tickets: </strong><%= totals.tickets %></p>
    <p><strong>Total price: $ </strong><%= totals.price %></p>
</div>
```

Finally, we create the page that displays the booking confirmation:

src/main/webapp/resources/templates/mobile/booking-details.html

```
<div data-role="header" data-position="fixed">
    <a href="#" class="ui-btn ui-icon-home ui-btn-icon-left">Home</a>
    <h1>Booking complete</h1>
</div>
<div class="ui-content">
    <table id="confirm_tbl">
        <thead>
        <tr>
            <td colspan="5" align="center"><strong>Booking <%=id%></strong></td>
        <tr>
        <tr>
            <th>Ticket #</th>
            <th>Category</th>
            <th>Section</th>
            <th>Row</th>
            <th>Seat</th>
        </tr>
        </thead>
        <tbody>
        <% $.each(_.sortBy(tickets, function(ticket) {return ticket.id}), function (i,
ticket) { %>
        <tr>
            <td><%= ticket.id %></td>
            <td><%=ticket.ticketCategory.description%></td>
            <td><%=ticket.seat.section.name%></td>
            <td><%=ticket.seat.rowNumber%></td>
            <td><%=ticket.seat.number%></td>
        </tr>
        <% }) %>
        </tbody>
    </table></div>
<div data-role="footer" data-position="fixed">

    <div class="ui-block-b">
        <a id="back" href="#" class="ui-btn ui-icon-back ui-btn-icon-left block-
btn">Back</a>
    </div>

</div>
```

The last step is registering the view with the router:

src/main/webapp/resources/js/app/router/mobile/router.js

```
/**
 * A module for the router of the desktop application
 */
define("router", [
    ...
    'app/views/mobile/create-booking',
    ...
],function (
            ...
            CreateBookingView
            ...) {

    var Router = Backbone.Router.extend({
        ...
        routes:{
            ...
            "book/:showId/:performanceId":"bookTickets",
            ...
        },
        ...
        bookTickets:function (showId, performanceId) {
            var createBookingView = new CreateBookingView({model:{showId:showId,
performanceId:performanceId, bookingRequest:{tickets:[]}}, el:$("#container")});
            utilities.viewManager.showView(createBookingView);
        },
        ...
        });
    ...
});
```

The other use case: a booking starting from venues is conceptually similar, so you can just copy the rest of the logic from `src/main/webapp/resources/js/app/routers/mobile/router.js`, and the rest of the files files in the `src/main/webapp/resources/js/app/views/mobile` and `src/main/webapp/resources/templates/mobile` directories.