



Red Hat JBoss Web Framework Kit 2.7 Explore the Examples

Examining the source code of the examples

Red Hat Customer Content Services

Red Hat JBoss Web Framework Kit 2.7 Explore the Examples

Examining the source code of the examples

Red Hat Customer Content Services

Legal Notice

Copyright © 2015 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Red Hat JBoss Web Framework Kit can be used to build and deploy example applications to observe how different frameworks are used. This guide illustrates two examples and explores the code they use.

Table of Contents

Chapter 1. Introduction to Red Hat JBoss Web Framework Kit	2
1.1. About Red Hat JBoss Web Framework Kit	2
1.2. About the JBoss Web Framework Kit Tiers	2
1.3. About the JBoss Web Framework Kit Distribution	3
Chapter 2. Explore the JBoss Web Framework Examples	4
2.1. About Explore the Examples	4
2.2. Explore the Examples System Requirements	4
Chapter 3. The contacts-mobile-basic Example	5
3.1. Running the Example	6
3.2. Exploring the Source Code	7
Chapter 4. The kitchensink-angularjs-bootstrap Example	17
4.1. Running the Example	17
4.2. Exploring the Source Code	18
Chapter 5. Next	30
5.1. Further Reading	30
5.2. Additional Examples	30
Revision History	31

Chapter 1. Introduction to Red Hat JBoss Web Framework Kit

1.1. About Red Hat JBoss Web Framework Kit

Red Hat JBoss Web Framework Kit is a set of enterprise-ready versions of popular open source frameworks. Together, these frameworks provide a solution for developing light and rich Java-based web applications.

JBoss Web Framework Kit comprises enterprise distributions of JBoss community web application frameworks and tested third-party frameworks. These leading frameworks support fast and easy client-side and server-side application development and testing, with frameworks including RichFaces, jQuery, jQuery Mobile, Hibernate Search, Spring, and Arquillian.

The breadth of JBoss Web Framework Kit frameworks provides choice for Java application development. Each framework has been tested and certified for use in applications deployed to Red Hat JBoss Enterprise Application Platform, Red Hat JBoss Web Server, or Red Hat OpenShift JBoss EAP cartridge. Inclusion in JBoss Web Framework Kit ensures stable versions of frameworks are available over long-term enterprise product life cycles, with regular releases for fixes and nonintrusive feature updates. Further, Red Hat JBoss Developer Studio (an Eclipse-based development environment) provides integrated project templates, quickstarts and tooling for many of the JBoss Web Framework Kit frameworks.

For the complete list of the frameworks composing JBoss Web Framework Kit and the certified platform and framework configurations, see <https://access.redhat.com/site/articles/112543> on the Red Hat Customer Portal.

[Report a bug](#)

1.2. About the JBoss Web Framework Kit Tiers

The frameworks composing JBoss Web Framework Kit are categorized in four distinct tiers. A description of each tier and the associated Red Hat support is detailed here.

Tier 1 - Included Components

These are components that are based wholly or partly on open source technologies that support broad collaboration and where Red Hat maintains a leadership role; as such Red Hat is able to support these components and provide upgrades and fixes under our standard support terms and conditions.

Tier 2 - Tested Frameworks

These are third-party frameworks where Red Hat does not have sufficient influence and does not provide upgrades and fixes under our standard support terms and conditions. Commercially reasonable support is provided by Red Hat Global Support Services for these frameworks.

Tier 3 - Frameworks in Tested Examples

These are third-party frameworks where Red Hat does not have sufficient influence and does not provide upgrades and fixes under our standard support terms and conditions. Red Hat supports the examples these frameworks are used in and the generic use cases that these examples intend to demonstrate.

Tier 4 - Confirmed Frameworks

These are third-party frameworks that do not receive any support from Red Hat, but Red Hat verifies that the frameworks run successfully on Red Hat JBoss Enterprise Application Platform. Frameworks and versions not listed here have not been explicitly tested and certified, and thus may be subject to support limitations.

For a list of JBoss Web Framework Kit frameworks by tier, see <https://access.redhat.com/site/articles/112543> on the Red Hat Customer Portal.

[Report a bug](#)

1.3. About the JBoss Web Framework Kit Distribution

The frameworks composing JBoss Web Framework Kit are distributed from a range of sources and in a variety of formats:

- The component frameworks are available from the Red Hat Customer Portal. They are distributed in two alternative formats: a binary for each framework or together as one Maven repository. In addition, the source code for each framework is provided for inspection.
- The third party frameworks are not distributed by Red Hat and each must be obtained from its own source.

A number of defined Maven JBoss stacks are provided as part of the JBoss Web Framework Kit distribution. All of the BOMs defining the JBoss stacks are available in the Maven repository `.zip` file available to download from the Red Hat Customer Portal or from <http://www.jboss.org/developer-materials/> on the JBoss Developer Framework website.

An extensive set of examples are also provided as part of the JBoss Web Framework Kit distribution:

- TicketMonster is a moderately complex application demonstrating a number of the JBoss Web Framework Kit frameworks working together.
- Quickstarts and Maven archetypes illustrate subsets of the JBoss Web Framework Kit frameworks used to create simple applications.
- RichFaces, Snowdrop and Seam demonstrations showcase the power of each framework in web application development.

All of these examples are available from the Red Hat Customer Portal, with TicketMonster, the quickstarts, and the Maven archetypes also available from <http://www.jboss.org/developer-materials/> on the JBoss Developer Framework website.

[Report a bug](#)

Chapter 2. Explore the JBoss Web Framework Examples

2.1. About Explore the Examples

2.1.1. Exploring the Examples Overview

This guide explains how to configure and run the **contacts-mobile-basic** example and the **kitchensink-angularjs-bootstrap** example. **contacts-mobile-basic** uses jQuery Mobile. **kitchensink-angularjs-bootstrap** uses AngularJS and Bootstrap.

This guide explains how to acquire, build and deploy the examples to a JBoss EAP server, and explores how the frameworks have been used in each example to achieve the look and functionality.

[Report a bug](#)

2.2. Explore the Examples System Requirements

2.2.1. System Requirements

Install and configure the following on your system:

1. **Red Hat JBoss Enterprise Application Platform** (version specific to this JBoss Web Framework Kit release)
2. **JBoss EAP Maven repository** (version matching installed JBoss EAP). Download this from the Red Hat Customer Portal and extract its contents.
3. **JBoss Web Framework Kit Maven repository** (version matching this release). Download this from the Red Hat Customer Portal and extract its contents.
4. **Java**. The instance of Java installed on your system must satisfy the requirements of Red Hat JBoss Enterprise Application Platform.

A list of supported browsers, versions and configurations of Red Hat JBoss Web Framework Kit is available at <https://access.redhat.com/site/articles/112543>.

[Report a bug](#)

Chapter 3. The contacts-mobile-basic Example

The **contacts-mobile-basic** example creates a simple mobile web application that stores contacts. It offers CRUD (create, read, update, delete) functionality.

jQuery Mobile is the main framework used to create the user interface of the application. The framework is composed of widgets that enable interactive application pages for mobile devices, and includes items such as headers, buttons and forms.

This chapter examines how to build and deploy the **contacts-mobile-basic** example, how it uses the jQuery Mobile framework, and how widgets are used.

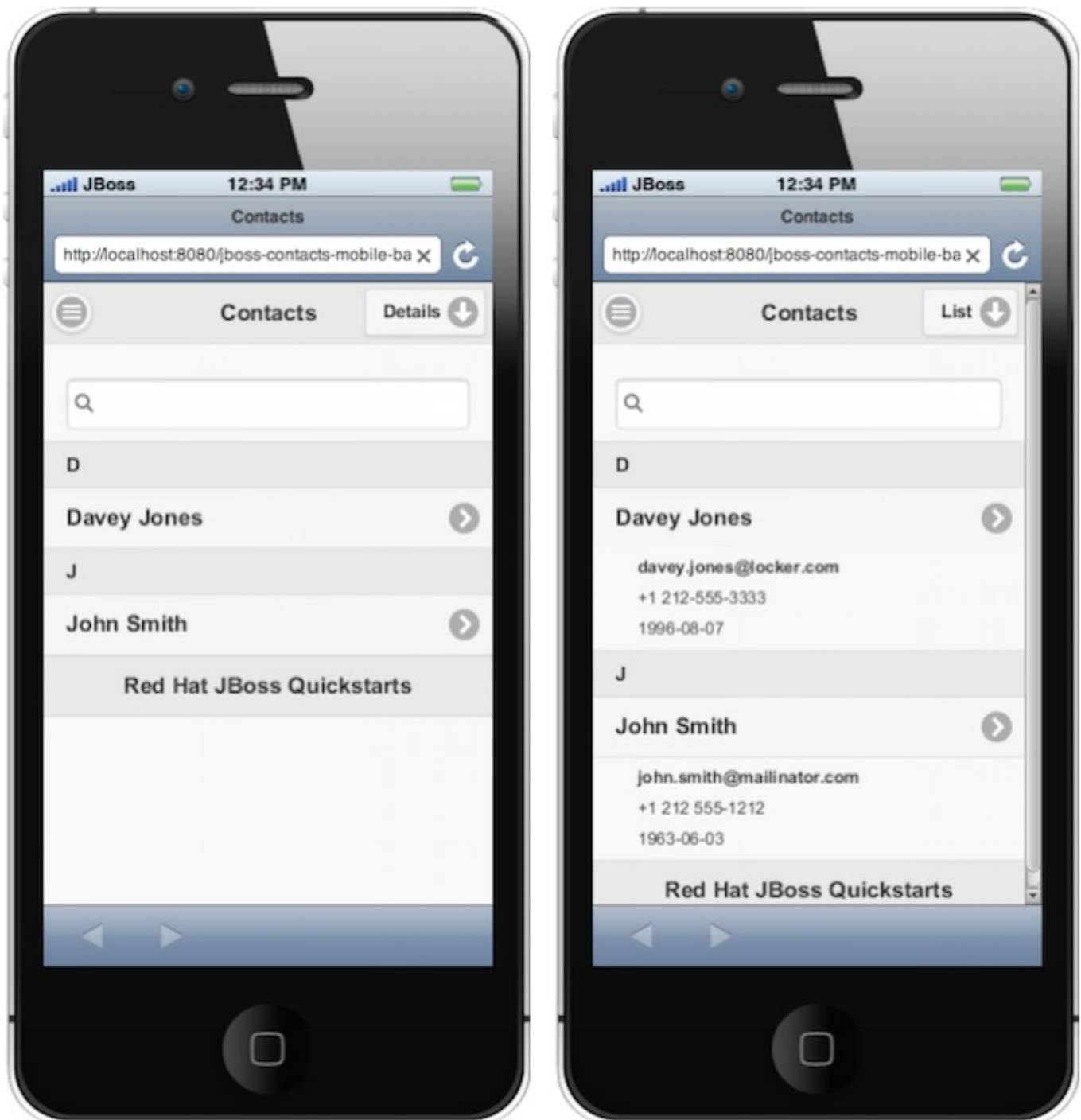


Figure 3.1. The contacts-mobile-basic Example: List and Details views



Figure 3.2. The contacts-mobile-basic Example: Sidebar and Add Contact views

[Report a bug](#)

3.1. Running the Example

The **contacts-mobile-basic** example is provided as part of the Red Hat JBoss Web Framework Kit distribution. The example can be downloaded from the Red Hat Customer Portal. This section explains how to download, build, and deploy the example.

[Report a bug](#)

3.1.1. Obtaining the Example

The **contacts-mobile-basic** example is available from the Red Hat Customer Portal. It must be downloaded and extracted into a local directory before it can be built.

Procedure 3.1. Obtaining the Example from the Red Hat Customer Portal

1. Log into the Red Hat Customer Portal at <http://access.redhat.com>.
2. Locate the downloads for this release version of Red Hat JBoss Web Framework Kit.
3. Download the **.zip** file for Red Hat JBoss Web Framework Kit quickstarts, which contains the **contacts-mobile-basic** example.
4. Extract the contents of the downloaded **.zip** file to the location where you want to keep the quickstarts source code.

[Report a bug](#)

3.1.2. Building the Example

Use **Maven** to build **contacts-mobile-basic** locally.

Procedure 3.2. Building contacts-mobile-basic with Maven

1. On the command line, navigate to the directory containing the **contacts-mobile-basic** example.
2. Run the following command to build the example:

```
$ mvn clean package
```

[Report a bug](#)

3.1.3. Deploying the Example

Deploy the example to the JBoss EAP server. After deploying the example to the JBoss EAP server, you can view the example in your browser.

Procedure 3.3. Deploying contacts-mobile-basic to the Server

1. In the directory containing JBoss EAP, run the following command to start the server:

```
$ ./bin/standalone.sh
```

2. Open a new command line instance and navigate to the directory containing the **contacts-mobile-basic** example.
3. Enter the following command to deploy the **contacts-mobile-basic** example:

```
$ mvn jboss-as:deploy
```

4. Open a web browser and access the example at <http://localhost:8080/jboss-contacts-mobile-basic/>.

[Report a bug](#)

3.2 Exploring the Source Code

3.2. Exploring the Source Code

Different frameworks are used for different aspects of the **contacts-mobile-basic** example. The application functionality is generated with Java EE. Client-side data validation is provided by jQuery and some custom JavaScript. The user interface is created with HTML5, CSS and jQuery Mobile.

This section explains the role jQuery Mobile plays in the **contacts-basic-mobile** example. The jQuery Mobile source code for the application is contained in **src/main/webapp/index.html**.

[Report a bug](#)

3.2.1. About jQuery Mobile

jQuery Mobile is a framework that uses JavaScript for building web applications.

jQuery Mobile is enriched with mobile support and uses HTML5 and CSS3 to render stylesheets on mobile devices. jQuery Mobile uses Ajax to pull data directly from the server and load it onto the webpage when the data is accessed.

jQuery Mobile is supported by most mobile devices and desktop browsers. Applications built with jQuery Mobile are accessible from both mobile devices and desktop browsers. Users with a basic understanding of how jQuery works as a JavaScript library can leverage this information to learn how to create applications with jQuery Mobile.

jQuery Mobile detects when a user accesses a website from a mobile devices and renders the website accordingly. jQuery Mobile scales the resolution to match the screen's resolution and loads the mobile version of the website.

Widgets (interactive objects in the user interface) are touchscreen-friendly. Buttons, lists, and toolbars are kinds of widgets. Widgets are added to the HTML body of the application, and display according to the style employed.

Styles are defined with the swatch elements in jQuery Mobile libraries. Styles customize the appearance of applications. Color schemes are defined at the beginning of the HTML body and inherited in the rest of the document. jQuery Mobile libraries also contain default buttons, headers, panels, and menus. The contents of these libraries makes it possible to use ready-made graphics, which reduces the cost of development and makes the development of similar content unnecessary.

[Report a bug](#)

3.2.2. The jQuery Mobile Declarations

jQuery Mobile libraries must be declared before jQuery Mobile can be used in a body of HTML.

In the **contacts-mobile-basic** example, these library declarations must be added to the `<head>` section of the **index.html** file. The example uses two declarations: one for the CSS library and one for the JavaScript library:

```
<head>
  ...
  <link rel="stylesheet" href="css/jquery.mobile-1.4.2.css"
type="text/css" media="all" />
  ...
  <script src="js/libs/jquery.mobile-1.4.2.js"></script>
</head>
```

The contents of these libraries are briefly described in the table below.

Table 3.1. jQuery Mobile Declarations

Name	Description
jquery.mobile-1.4.2.css	Contains the data for implementing style sheets. The file includes information on style elements, such as button edges, field separators, and link-enabled buttons. These provide the visual elements of the application.
jquery.mobile-1.4.2.js	Contains pre-written JavaScript for adding features such as transition animations, drop-down lists, and visual effects. It also contains uncompressed JavaScript used for debugging. It is added to the <head> section last to apply mobile mark-ups on launch.

[Report a bug](#)

3.2.3. The Pages

The **contacts-mobile-basic** example has five pages. These pages provide the main content of the application. Each page has:

1. an initial page statement
2. common content such as headers and footers
3. unique page content

The jQuery Mobile widgets used to implement each of these are discussed in this section.

[Report a bug](#)

3.2.3.1. Page Statement

Each page in the **contacts-mobile-basic** example is implemented by including the page widget in the <body> of **index.html**:

```
<body>
  <div data-role="page" id="contacts-list-page" class="ui-page-theme-a"
data-url="/jboss-contacts-mobile-basic/"> ... </div>
  <div data-role="page" id="contacts-detail-list-page" class="ui-page-
theme-a" > ... </div>
  <div data-role="page" id="about-page" class="ui-page-theme-a" > ...
</div>
  <div data-role="page" id="contacts-add-page" class="ui-page-theme-a" >
... </div>
  <div data-role="page" id="contacts-edit-page" class="ui-page-theme-a"
> ... </div>
  ...
</body>
```

Each page widget is defined in a <div> tag with data-role value **page**.

Page widgets have several attributes. Some page widgets have unique values and some have shared values. By default, the first page listed in the <body> displays when a user navigates to the web application URL. This is the page widget for the first page of the application:

```
<div data-role="page" id="contacts-list-page" class="ui-page-theme-a"
data-url="/jboss-contacts-mobile-basic/"> ... </div>
```

Table 3.2. Attributes

Name	Description
data-role	Indicates what kind of object displays. In the example above, this attribute declares that this is a page.
id	A unique name given for identification and storage of the page in the application.
class	Indicates the jQuery Mobile CSS color swatch used for the page. This example uses “a”, which corresponds to a default light theme.
data-url	The location of the example directory. The page at the top of the <body> loads as the homepage.

[Report a bug](#)

3.2.3.2. Common Page Content

The majority of the pages in the **contacts-mobile-basic** example contain widgets for rendering a header and a footer. These are nested within the page widget, as demonstrated here for the first page:

```
<div data-role="page" id="contacts-list-page" class="ui-page-theme-a"
data-url="/jboss-contacts-mobile-basic/">
  ...
  <div data-role="header" data-position="fixed" > ... </div>
  ...
  <div data-role="footer" > ... </div>
</div>
```

The header and footer widgets are distinguished by their data-role values.

Table 3.3. Attributes

Name	Description
data-position	Indicates the position of the widget. If set to fixed , the widget stays in one place on the screen. Because the first widget is defined as a header, it displays at the top of the screen. Because the second widget is defined as a footer, it displays at the bottom of the screen.

The headers in the **contacts-mobile-basic** example typically contain a title and button widgets to enable navigation between the various application pages. Some of the more detailed footers also contain button widgets. The button widgets are simply links to other pages in the application represented with icons and text, as demonstrated here for the first page:

```
<div data-role="header" data-position="fixed" >
  <a href="#contacts-list-page-menu-panel" id="contacts-list-page-menu-
button" class="ui-btn ui-corner-all ui-shadow ui-icon-bars ui-btn-icon-
notext ui-mini ui-btn-inline">Menu</a>
  <h1>Contacts</h1>
```

```
<a href="#contacts-detail-list-page" id="contacts-detail-list-page-
button" class="ui-btn ui-corner-all ui-shadow ui-icon-arrow-d ui-btn-
icon-right ui-mini ui-btn-inline">Details</a>
</div>
```

This page header has two buttons,  with icon **ui-icon-bars** and no text to open the **#contacts-list-page-menu-panel** and  with icon **ui-icon-arrow-d** and text **Details** to open the **#contacts-detail-list-page**.

Table 3.4. Button Attributes

Name	Description
ui-btn	Indicates that this is a button widget.
ui-corner-all	Adds rounded corners to the element.
ui-shadow	Adds an item shadow around the element.
ui-icon-...	Indicates the icon to be used for the button: bars, arrow-d (↓), carat-r (>), and many more.
ui-btn-icon-...	Indicates where the icon appears in relation to the button text: top, bottom, right, left. Alternatively, no text suppresses the button text and the result is a circular button the size of the icon.
ui-mini	Reduces the font size and scales down paddings proportionally to produce a miniature version of the element for use in toolbars and tight spaces.
ui-btn-inline	Displays the button inline. This means that it will only consume as much space as is needed for the label. This allows you to place buttons side by side, flowing with the text.

[Report a bug](#)

3.2.3.3. Unique Page Content

The **Contacts List** and **Detailed List** pages are similar in structure, both displaying an alphabetized list of contacts but the latter with the contact details expanded. As such, the implementation of the pages is also very similar. The implementation for the **Contacts List** page is given here:

```
<div data-role="page" id="contacts-list-page" class="ui-page-theme-a"
data-url="/jboss-contacts-mobile-basic/">
...
<div role="main" class="ui-content">
  <form id="filter-form-list-page" class="ui-filterable">
    <input id="filter-input-list-page" data-type="search">
  </form>
  <ul data-role="listview" id="contacts-display-listview"
class="sortedList" data-autodividers="true" data-filter="true" data-
input="#filter-input-list-page">
    <!-- The list of Contacts gets inserted here. -->
  </ul>
</div>
...
</div>
```

Table 3.5. Contacts List Attributes

Name	Description
data-role="listview"	Adds a list containing the contacts. These are added in place of <!-- The list of contacts gets inserted here. --> . This list inherits the style swatch specified at the beginning of the page. In this case, the style swatch specified at the beginning of the page is "a".
id="contacts-display-listview"	The unique name given to the list, used for identification and storage.
class="sortedList"	Sorts each list entry. By default, these entries are sorted in alphabetical order.
data-autodividers="true"	Adds graphical bars that separate each entry on the list from the other. In this case, each name is presented as text in its own blue-colored horizontal bar.
data-filter="true"	Adds a search field to the list. In this case, the search field is located above the list of names. The user types the search term into the box and presses enter or clicks the magnifying glass icon to have the results "filtered" back to them.
data-input="#filter-input-list-page"	Specifies that the list of contacts serves as the data over which the search field operates.

Likewise, the **Add Contact** and **Edit Contact** pages are similar in structure, both displaying a five-field form but the latter with contact details already populating the form fields. As such, the implementation of the pages is also very similar. The implementation for the **Edit Contact** page is given here:

```
<div data-role="page" id="contacts-edit-page" class="ui-page-theme-a" >
  ...
  <div role="main" class="ui-content">
    <form name="contacts-edit-form" id="contacts-edit-form"
class="contact_info" method="post" data-ajax="false">
      <div>
        <label for="contacts-edit-input-firstName">First Name:</label>
        <input name="firstName" id="contacts-edit-input-firstName"
class="name" data-clear-btn="true" value="" placeholder="Enter a first
name." type="text"/>
      </div>
      <div>
        <label for="contacts-edit-input-lastName">Last Name:</label>
        <input name="lastName" id="contacts-edit-input-lastName"
class="name" data-clear-btn="true" value="" placeholder="Enter a last
name." type="text"/>
      </div>
      <div>
        <label for="contacts-edit-input-tel">Phone:</label>
        <input name="phoneNumber" id="contacts-edit-input-tel"
class="phoneNumber" data-clear-btn="true" value="" type="tel"/>
      </div>
      <div>
        <label for="contacts-edit-input-email">Email:</label>
        <input name="email" id="contacts-edit-input-email" class="email"
data-clear-btn="true" value="" placeholder="name@company.domain"
type="email"/>
      </div>
      <div>
        <label for="contacts-edit-input-date">Birth Date:</label>
```



```

        <input name="birthDate" id="contacts-edit-input-date"
class="birthDate" data-clear-btn="true" value="" placeholder="YYYY-MM-
DD" type="date" min="1900-01-01"/>
    </div>
    <div>
        <input name="id" id="contacts-edit-input-id" value=""
type="hidden"/>
    </div>
    <input id="submit-edit-btn" data-inline="true" type="submit"
value="Save" />
    <input id="clear-edit-btn" data-inline="true" type="reset"
value="Clear" />
    <input id="cancel-edit-btn" data-inline="true" type="reset"
value="Cancel" />
    </form>
</div>
...
</div>

```

Table 3.6. Form Widget Attributes

Name	Description
form name="contacts-edit-form"	The name of the form being added.
id	Specifies which form is being added. In this case, the "contacts-edit-form" is being added.
class="contact_info"	Declares the class properties.
method="post"	Specifies how any data typed into the form is sent. In this example, data is sent as a post transaction. This means that the data displays in the body of the request.
data-ajax="false"	Disables automatic page transitions when forms are submitted. This keeps forms on screen and prevents applications from redirecting to other pages.

Table 3.7. First Name Field Attributes

Name	Description
label for="contacts-edit-input-firstName"	Specifies the part of the form defined in the quotation marks. In this case, it specifies the First Name text field.
input name="firstName"	Specifies a text-box field. In this case, it is the field for entering the first name of a contact.
id	A unique name for referring to the text-box field. It is stored locally for quick access.
class="name"	Specifies the class name for the text box. The jQuery Mobile CSS file adds a stylized graphical visualization of the text box.
data-clear-btn="true"	Adds to the layout a button that clears data. Click this button to remove all user-entered information from the text box.
placeholder="Enter a first name."	Adds grayed-out text to the text box. In this case, it reads "Enter a first name". The text indicates that the user can type into the field. The grayed-out text disappears when the user enters input into the field.
type="text"	Indicates the type of input that is accepted. In this case, text entries are accepted as input.

The **About** page contains a list containing two links for learning more about JBoss EAP. The `` attribute indicates that the links are list items:

```
<div data-role="page" id="about-page" class="ui-page-theme-a" >
  ...
  <div role="main" class="ui-content" >
    <p>Learn more about Red Hat JBoss Enterprise Application Platform 6.
  </p>
    <ul>
      <li><a
href="https://access.redhat.com/site/documentation/JBoss_Enterprise_Appli
cation_Platform/">Documentation</a></li>
      <li><a href="http://red.ht/jbeap-6">Product Information</a></li>
    </ul>
  </div>
</div>
```

[Report a bug](#)

3.2.4. The Panel

The panel stores additional options without cluttering the screen. Clicking on the Menu button in the header expands this panel. The panel displays a list of links.

The **contacts-mobile-basic** example has 5 panels, 1 panel per page. The panel for the first page is different, containing four links: Add Contact, Detailed List, About, and Themes. The panels for the other four pages of the application are the same and have three links: Add Contact, List view, and About.

[Report a bug](#)

3.2.4.1. Panel Statement

Each panel is generated by a panel widget. The panel widget is nested inside each page widget, as demonstrated here for the first page:

```
<div data-role="page" id="contacts-list-page" class="ui-page-theme-a"
data-url="/jboss-contacts-mobile-basic/">
  <div data-role="panel" id="contacts-list-page-menu-panel"> ... </div>
  ...
</div>
```

The panel widget has only the data-role and id attributes. No other attributes are defined in the panel widget.

[Report a bug](#)

3.2.4.2. Panel Content

The panel contains a listview widget with links, as demonstrated here for the panel of the first page:

```
<div data-role="panel" id="contacts-list-page-menu-panel">
  <ul data-role="listview" id="contacts-list-page-menu-panel-listview">
    <li><a href="#contacts-add-page">Add Contact</a></li>
    <li><a href="#contacts-detail-list-page">Detailed List</a></li>
```

```

    <li><a href="#about-page">About</a></li>
    <li><a href="#contacts-theming-dialog" data-rel="popup" data-
position-to="window">Themes</a></li>
  </ul>
</div>

```

Table 3.8. Panel Content Attributes

Name	Description
	Specifies that the link is an item on a list.

In this example, the listview widget has only "data-role" and "id" as defined attributes.

[Report a bug](#)

3.2.5. The Dialog

On the **Edit Contact** screen, there is a Delete button in the header. The Delete button removes the contact from the application. When a user clicks the button, a window opens to confirm the deletion.

[Report a bug](#)

3.2.5.1. Dialog Statement

The dialog is implemented with the popup widget. This widget is added at the bottom of the **contacts-edit-page**:

```

<div data-role="page" id="contacts-edit-page" class="ui-page-theme-a" >
  ...
  <div data-role="popup" id="contact-delete-dialog" data-overlay-
theme="b"> ... </div>
</div>

```

Table 3.9. Dialog Attributes

Name	Description
data-role	Indicates the type of information that displays, in this case, a popup window.
id	The identifying name for the dialog that opens when the Delete button is activated, in this case, "contact-delete-dialog".

[Report a bug](#)

3.2.5.2. Dialog Content

The dialog widget contains a header and a content widget. The content widget contains a header **Are you sure you want to Delete?** and two button widgets corresponding to **Yes** and **No**.

```

<div data-role="popup" id="contact-delete-dialog" data-overlay-
theme="b">
  <div data-role="header">
    <h1>Delete Contact</h1>
  </div>

```

```
<div role="main" class="ui-content">
  <!-- PUT and DELETE are not supported in an HTML form. Please see
  http://www.w3.org/TR/2010/WD-html5-diff-20101019/#changes-2010-06-24 That
  is why we are using a dialog with simple Yes/No. The ID of the Contact
  to be deleted will programatically be pulled in from the Update form. --
  >
  <h3>Are you sure you want to Delete?</h3>
  <a href="#contacts-list-page" id="confirm-delete-button" class="ui-
  btn ui-corner-all ui-shadow ui-btn-inline ui-btn-b">Yes</a>
  <a href="#contacts-list-page" id="cancel-delete-button" class="ui-btn
  ui-corner-all ui-shadow ui-btn-inline">No</a>
</div>
</div>
```

Table 3.10. Dialog Content Attributes

Name	Attribute
ui-content	Indicates the type of information that displays, in this case, the dialog content.
ui-btn-b	Sets the color swatch for the button, here "b. Use a single letter from a-z that maps to the swatches included in your theme.

[Report a bug](#)

Chapter 4. The kitchensink-angularjs-bootstrap Example

The **kitchensink-angularjs-bootstrap** example creates an application to register members. The name, email address, and phone number of a new member is entered with a webform. After registration, those details are added to a members list for future reference. This example is designed primarily for mobile devices.

This example makes use of two frameworks: AngularJS and Bootstrap. These frameworks add objects to the user interface, store JavaScript files, and match the layout various screen sizes. This section examines the layout, features, and coding of the example.

[Report a bug](#)

4.1. Running the Example

The **kitchensink-angularjs-bootstrap** example requires the Red Hat JBoss Web Framework Kit distribution. The example can be downloaded from the Red Hat Customer Portal. This section explains how to download, build, and deploy the example.

[Report a bug](#)

4.1.1. Obtaining the Example

The **kitchensink-angularjs-bootstrap** example is available from the Red Hat Customer Portal. It must be downloaded and extracted into a local directory before it can be built.

Procedure 4.1. Downloading the Example from the Red Hat Customer Portal

1. Log into the Red Hat Customer Portal at <http://access.redhat.com>.
2. Locate the downloads for this version of Red Hat JBoss Web Framework Kit.
3. Download the **.zip** file for Red Hat JBoss Web Framework Kit quickstarts, which contains the **kitchensink-angularjs-bootstrap** example.
4. Extract the contents of the downloaded **.zip** file to the location where you want to keep the quickstarts source code.

[Report a bug](#)

4.1.2. Building the Example

Use **Maven** to build **kitchensink-angularjs-bootstrap** locally.

Procedure 4.2. Building kitchensink-angularjs-bootstrap with Maven

1. On the command line, navigate to the directory containing the **kitchensink-angularjs-bootstrap** example.
2. Run the following command to build the example:

```
$ mvn clean package
```

[Report a bug](#)

4.1.3. Deploying the Example

Deploy the example to the JBoss EAP server. After deploying the example to the JBoss EAP server, you can view the example in your browser.

Procedure 4.3. Deploying `kitchensink-angularjs-bootstrap` to the Server

1. In the directory containing JBoss EAP, run the following command to start the server:

```
$ ./bin/standalone.sh
```

2. Open a new command line instance and navigate to the directory containing the `kitchensink-angularjs-bootstrap` example.
3. Enter the following command to deploy the `kitchensink-angularjs-bootstrap` example:

```
$ mvn jboss-as:deploy
```

4. Open a web browser and access the example at <http://localhost:8080/jboss-kitchensink-angularjs-bootstrap/>.

[Report a bug](#)

4.2. Exploring the Source Code

The user interface of the `kitchensink-angularjs-bootstrap` example is built with two main frameworks: **AngularJS** and **Bootstrap**. These create the look and functionality of the application.

This section explains the roles Bootstrap and AngularJS play in the `kitchensink-angularjs-bootstrap` example. The user interface source code of the example is located primarily in `src/webapp/index.html`. The user interface is generated with CSS and JavaScript, the libraries for which are located in `kitchensink-angularjs-bootstrap/src/main/webapp/css` and `kitchensink-angularjs-bootstrap/src/main/webapp/js/libs` respectively.

[Report a bug](#)

4.2.1. Bootstrap

4.2.1.1. About Bootstrap

Bootstrap is a framework used to create responsive design applications. In this example, it is used for its function that changes the size of the user interface, so that the user interface fits on the screen of the target device. Mobile screens come in various sizes, and bootstrap ensures that your application is delivered to each in a format appropriate to the device.

You need only to create the primary application, and Bootstrap will ensure that it fits the resolution of the target screen. This makes it unnecessary to write a version of the application for each device you intend to deliver the application to. It removes problems that arise when attempting to create a standard-sized layout when there is no standard screen size across all mobile device models.

The Bootstrap framework consists of JavaScript and CSS file libraries that used in mobile applications. These libraries are inserted in the header. Because the libraries are pre-loaded, users do not have to wait for website information to download from the server.

Bootstrap uses a grid system that automatically scales webpages to the size of the screen. User-

interface objects are contained in the rows of columns that make up the grid. These columns fix to the size of the screen, changing the position of the objects where necessary to create a consistent layout. A maximum of twelve columns is allowed, which will stretch or skew to the size of the screen. The default is three columns of equal size. This ensures that they do not overlap or distort, no matter the screen size.

[Report a bug](#)

4.2.1.2. The Bootstrap Declarations

The file libraries necessary to invoke this example are declared in the **index.html** file, located in **WEB-INF**. These library declarations occur in the <head> section. The CSS and JavaScript libraries are declared:

```
<head>
...
<link rel="stylesheet" href="css/bootstrap.css" type="text/css" />
<link rel="stylesheet" href="css/bootstrap-theme.css" type="text/css" />
...
<!-- Load Bootstrap JS components -->
<script src="js/libs/bootstrap.js"></script>
...
</head>
```

The contents of these libraries are briefly described in the table below.

Table 4.1. Bootstrap Declarations

Name	Description
bootstrap.css	Contains the columns and rows that Bootstrap requires to create a grid. Also contains information on margin sizes, coloring, and fonts. index.html uses the bootstrap.css stylesheet to generate columns (<col-*>) in the container section of the body.
bootstrap-theme.css	Contains graphical and theme elements for the user interface, such as text shadows, buttons, and navigation bars. index.html uses this to generate the appearance of the buttons. (For example, class="btn btn-default" .)
bootstrap.js	Contains all the Bootstrap JavaScript plugins. This includes transition effects, button functions, and pop-up text. In this example, JavaScript implementations are performed by AngularJS .

[Report a bug](#)

4.2.1.3. The Layout

The layout of the application provides an interface which objects can be added to. It contains several items:

- ✳ A header that displays a JBoss EAP graphic and welcome message
- ✳ A footer that displays the text “This project was generated from a Maven archetype from JBoss.”
- ✳ Space to insert the form

The content of the layout changes position depending on the size of the screen. When opened in a desktop browser, the application displays horizontally. The side panel sits on the right of the page next to the form, with the JBoss EAP graphic aligned to the right. When opened on a mobile device, the application displays vertically. The objects are stacked to preserve the layout, preventing overlapping and clutter. The JBoss EAP graphic is pushed to the top of the screen with the form directly under it, followed by the side panel. Bootstrap defines the aspect ratio of application elements using a grid system. The information that is used to produce columns and rows comes from the Bootstrap.css file.

The following code in the index.html file is used to correctly position the graphic and welcome message:

```
<section id="container" class="container">
  <div class="col-xs-12 col-sm-12 col-md-12">
    <div class="dualbrand img-responsive pull-right">
      
    </div>
  </div>
```

Table 4.2. Layout Attributes

Name	Description
section id	A unique identifier given to refer to the object. (In this case, "container".)
class	Defines what sort of object is being inserted. In this case, it is a container. The container is used to nest columns and rows.
<div class="col-xs-12 col-sm-12 col-md-12">	Defines the size of the container. This is determined by columns, referenced to as col-xs (extra small devices), col-sm (small devices), and col-md (medium devices). All these columns are followed by a 12, indicating that they are 12 columns wide. Depending on the size of the screen (extra small, small or medium), the container scales the columns to fit.
<div class="dualbrand img-responsive pull-right">	Makes the layout responsive to the device it is being viewed on. The dualbrand variable is a style class. The img-responsive variable scales the image (in this case, rhjb_eap_logo.png) to fit the size of the screen. The pull-right variable ensures the container floats to the right of the screen. This prevents it from distorting the layout when the user views and moves about the application.

[Report a bug](#)

4.2.1.4. The Content

The page content consists of a web form generated by **AngularJS**. Bootstrap provides a section for it in **index.html**:

```
<div id="content" class="col-xs-12 col-sm-12 col-md-8" ng-view>
  <!-- This div will be templated by AngularJS -->
  [Template content will be inserted here]
</div>
```

Table 4.3. Content Attributes

Name	Description
<code>id="content"</code>	A unique identifier given to refer to the object. (In this case, "content".)
<code>class="col-xs-12 col-sm-12 col-md-8"</code>	Determines the scaling of the content. In this case, the web form is the content. Screens defined as "extra small" and "small" display the web form over the width of 12 columns. A "medium-sized" screen displays the web form over 8 columns. In a desktop browser, the web form displays horizontally. On a mobile device, the web form displays vertically.

[Report a bug](#)

4.2.1.5. The Side Bar

The side bar contains links to the JBoss EAP documentation. The side bar displays in a gray box that changes position depending upon the size of the screen.

```
<aside id="aside" class="col-xs-12 col-sm-12 col-md-4">
  <div class="row">
    ...
  </div>
</aside>
```

Table 4.4. Side Bar Attributes

Name	Description
<code>aside</code>	Creates the side bar. In a web browser, this bar displays on the top right of the homepage. When scaled down to adhere to a mobile device screen, the bar moves to the bottom of the homepage.
<code>id</code>	A unique identifier that refers to the object. (In this case, "aside".)
<code>class</code>	Defines the size of the side bar on extra small, small, and medium screens (12, 12, and 4 columns wide respectively). This data is drawn from Bootstrap.css . The col-xs and col-sm variables in this example display content over 12 columns, so as to take up the whole width of the layout on a mobile device. The col-md variable displays information over 4 columns, which exhausts the layout width unconsumed by content on medium-sized screens.
<code><div class="row"></code>	Creates a row for the panel's content. Each row contains a group of columns and is nested inside a container. Adding another row adds another group of columns to which content may be added.

[Report a bug](#)

4.2.2. AngularJS

4.2.2.1. About AngularJS

The AngularJS framework is used in the creation of single-page applications. All input and validation processes on one screen, and content updates dynamically. AngularJS can be used as a standalone framework or added to the body of a HTML document. Because of its instant response time, it is useful in creating mobile applications.

As all the information in the application displays on one page, users do not have to navigate between different tabs and sections. This decreases loading time and ensures that data is easily accessible. Everything loads on launch. If a user makes an error when typing information, a warning displays immediately. The application does not spend time validating the input before producing an error message. This ensures that information is not lost when switching between forms and validation pages.

The AngularJS framework uses JavaScript libraries that are declared in the header of the **index.html** file. These libraries load when the application launches. The libraries do not wait for a response from the server before loading. AngularJS objects are defined with `<ng>` tags.

AngularJS uses directives to dictate the behavior of Document Object Model (DOM) elements. This is achieved through data-binding, which automatically updates the application when changes are made. Forms are instantly validated. AngularJS objects can be embedded in other frameworks to create multi-technology mobile applications.

[Report a bug](#)

4.2.2.2. The AngularJS Declarations

The necessary JavaScript libraries are declared in the **index.html** file. The folders are located in **src/main/webapp/js/libs**. These library declarations occur in the "`<head>`" section of the example and apply to the entire application:

```
<head>
  ...
  <!-- Load angularjs -->
  <script src="js/libs/angular.js"></script>
  <!-- Load angularjs-route, which allows us to use multiple views
displayed through application routes -->
  <script src="js/libs/angular-route.js"></script>
  <!-- Load angularjs-resource, which allows us to interact with REST
resources as high level services -->
  <script src="js/libs/angular-resource.js"></script>
  ...
</head>
```

The contents of these libraries are briefly described in the table below.

Table 4.5. AngularJS Declarations

Name	Description
angular.js	Contains the AngularJS JavaScript plugins and directives (attributes). In this example, it contains the elements needed to generate the form. This is the primary library.
angular-resource.js	Contains JavaScript for implementing RESTful data sources. In this example, it is used when generating the list of contacts displayed at the bottom of the web form.
angular-route.js	Contains the JavaScript elements used to respond to input by directly updating the layout. It implements the <code>ngView</code> directive that renders the layout.

[Report a bug](#)

4.2.2.3. The Member Registration Form

When the application opens, a webform, the "member registration form", displays. The user enters the contact details of the person they wish to register as a member. When submitted, these contact details are saved in the application and given RESTful URLs. The member registration form provides a simple method of adding and monitoring contact details and can be modified on the fly.

The form contains text fields, buttons, and a validation option that are generated by AngularJS. These objects are contained in the AngularJS libraries and appear in the body of the **home.html** file. They are responsive to user input and have been designed with mobile devices in mind. The properties and functions of these elements are discussed in detail in this section.

[Report a bug](#)

4.2.2.3.1. Form Statement

The webform is a standard HTML form which uses an AngularJS template. It is contained in the **home.html** file in **src/main/webapp/partials**. This file loads the form, which is inserted in the body of the application:

```
<!-- This is a partial, and is templated by AngularJS -->
<div>
  <!-- Standard HTML form, with the submit function bound to the
  register() function defined in the the controller -->
  <form name="regForm" class="form-horizontal" ng-submit="register()">
    <h2>Member Registration</h2>
    <fieldset>
      ...
    </fieldset>
  </form>
  ....
</div>
```

Table 4.6. Form Attributes

Name	Description
name="regForm"	Identifies the object is a form. A unique name is entered to identify the form in the layout. In this case, it is named regForm .
class="form-horizontal"	Ensures that the form fills the entire width of the layout and that the form uses all available space. A "form-vertical" attribute would ensure that the form takes up the entire length of the available space.
ng-submit="register()"	Reloads the form using AngularJS when a member is registered. It refreshes instantly instead of pulling data from the server.

All of the form fields, buttons, and validation are nested within the `<form>` element.



Member Registration

Register a member:

Name:

required

Email:

required

Phone #:

required

Figure 4.1. Form Generated with AngularJS

[Report a bug](#)

4.2.2.3.2. Form Fields

The form has three text fields: "Name", "Email", and "Phone #". They must all be completed before the form is submitted. These fields are declared in the <fieldset> of the form. Each field shares several common attributes, but also contains unique attributes.

```
<fieldset>
    <legend>Register a member:</legend>
    <!-- Each input is bound to a property on the prototype
newMember object -->
    <div class="form-group">
        <label for="name" class="col-sm-2 control-label">Name:
</label>
        <div class="col-sm-10">
            <input type="text" name="name" id="name" ng-
model="newMember.name" placeholder="Your Name" required autofocus/>
            <span class="error help-block" ng-
show="regForm.name.$error.required">required</span>
            <span class="error help-block" ng-show="errors.name">
{{errors.name}}</span>
        </div>
    </div>
    <div class="form-group">
        <label for="email" class="col-sm-2 control-label">Email:
</label>
        <div class="col-sm-10">
            <input type="email" name="email" id="email" ng-
model="newMember.email" placeholder="Your Email" required />
            <span class="error help-block" ng-
show="regForm.email.$error.required">required</span>
            <span class="error help-block" ng-
show="regForm.email.$error.email">not a valid email</span>
            <span class="error help-block" ng-
show="errors.email">{{errors.email}}</span>
        </div>
    </div>
    <div class="form-group">
        <label for="phoneNumber" class="col-sm-2 control-
label">Phone #:</label>
        <div class="col-sm-10">
            <input type="tel" name="phoneNumber"
id="phoneNumber" ng-model="newMember.phoneNumber" ng-pattern="/^[0-9]
{10,12}$/" placeholder="Your Phone #" required />
            <span class="error help-block" ng-
show="regForm.phoneNumber.$error.required">required</span>
            <span class="error help-block" ng-
show="regForm.phoneNumber.$error.pattern">must be between 10 and 12
digits</span>
            <span class="error help-block" ng-
show="errors.phoneNumber">{{errors.phoneNumber}}</span>
        </div>
    </div>
    ....
</fieldset>
```

Table 4.7. Common Attributes

Name	Description
input type	Specifies the kind of data that can be added to the form, in this case, text.
name	Where the title of the field is specified. These fields are names name , email , and phoneNumber respectively.
ng-model	Creates a two-way binding process model. It is triggered by the input, in this case, when a user enters text. When the input is invalid, it alerts the user with a warning that displays beneath the name field.
placeholder	Adds grayed-out text to indicate the field required input, for example, Your Name . The text disappears when the user starts typing.
ng-show	Displays events when triggered. In this example, it is triggered when the required input is missing, resulting in a red error message displaying beneath the field.
required	Requires the field to contain input before it is submitted. The form will not validate if the field is left blank.

Table 4.8. Unique Attributes

Name	Description
autofocus	The Name field contains an autofocus attribute. This activates the field when the application deploys. In this example, it adds a flashing text cursor to the first field of the form to prompt the user to add input.
ng-pattern	The Phone # field contains the ng-pattern directive. This defines which characters are allowed. In this example, it allows only umerical characters ([0-9]) that are 10-12 numbers long ({10,12}).

[Report a bug](#)

4.2.2.3.3. Form Buttons

After filling out the form, users can choose to either submit it or cancel. This is done with two buttons labeled "Register" and "Cancel". They are located at the bottom of the form and are specified after the validation messages in the **home.html** file:

```
<fieldset>
  ...
  <div class="form-group">
    <div class="col-sm-offset-2 col-sm-10">
      <input type="submit" ng-disabled="regForm.$invalid" id="register"
value="Register" class="btn btn-primary" />
      <input type="button" ng-click="reset()" name="cancel" id="cancel"
value="Cancel" class="btn btn-default" />
    </div>
  </div>
</fieldset>
```

Table 4.9. Button Attributes

Name	Description
------	-------------

Name	Description
input type	Defines which of the elements displays. In this example, both elements are buttons: the first is used to submit the form and the second is used to cancel it.
ng-disabled	Stops the browser from submitting the form if the input is incorrect. In this case, it disables the validation button if the registration form is invalid.
ng-click	Specifies which action the button takes when it is clicked. In this case, it contains a reset command. Clicking the Cancel button resets the form to its default state.



Figure 4.2. Form Buttons

[Report a bug](#)

4.2.2.3.4. Form Validation

The validation messages are inserted after the form fields in the home.html file and display underneath the Phone # field in the application. When the user submits the form, either a success or an error message displays.

```
<fieldset>
  ...
  <!-- We output a list of success messages (appropriately
  styled!) for the form. -->
  <ul ng-hide="!successMessages" class="success">
    <li ng-repeat="message in successMessages">{{message}}
  </li>
  </ul>

  <!-- Output the list of error messages if any. -->
  <ul ng-hide="!errorMessages" class="error">
    <li ng-repeat="message in errorMessages">{{message}}</li>
  </ul>
  ...
</fieldset>
```

Table 4.10. Validation Attributes

Name	Description
ng-hide	Hides data where specified. If there is no error message, the error message dialogue is hidden and not appear on screen.
ng-repeat	Repeats a set of data from a specified location. In this case, it replicates the list of the messages in the defined attribute (successMessages or errorMessages).

[Report a bug](#)

4.2.2.4. The Members List

The members list contains contact details for every member that has been registered using the form. In the application, the members list displays in a box beneath the form with the title “Members”. When a new name is submitted, it is instantly added to this list which generates an ID and REST URL for it. Beneath the list is a REST URL that contains the details of all the members in the list. These URLs do not point to a link on the web. These URLs link to the location within the application where the details are stored. **AngularJS** adds new entries to this list automatically. Click the refresh button under the table to manually reload the entries.

Members

Id	Name	Email
0	John Smith	john.smith@m

REST URL for all members:

/rest/members

Refresh

Figure 4.3. Members List

[Report a bug](#)

4.2.2.4.1. List Statement

After a new member is added, the details of that member display in the members list below the form. The members list is the second heading in the **home.html** file:

```
<!-- This is a partial, and is templated by AngularJS -->
<div>
  ...
  <!-- A list of registered members -->
  <h2>Members</h2>
  <!-- If there are no members registered, instead of showing the
table, we show a simple message. -->
  <em ng-show="members.length == 0">No registered members.</em>
  <div class="table-responsive">
    <table ng-hide="members.length == 0" class="table table-striped">
      <thead>
        <tr>
          <th>Id</th>
          <th>Name</th>
          <th>Email</th>
          <th>Phone #</th>
          <th>REST URL</th>
        </tr>
      </thead>
      <!-- The table is built using the AngularJS repeat function,
```



```

iterating over the members variable, and ordering by the property
specified in the orderBy variable -->
    <tr ng-repeat="member in members | orderBy:orderBy">
        <td>{{member.id}}</td>
        <td>{{member.name}}</td>
        <td>{{member.email}}</td>
        <td>{{member.phoneNumber}}</td>
        <td><a
href="rest/members/{{member.id}}">rest/members/{{member.id}}</a>
        </td>
    </tr>
</table>
</div>
...
</div>

```

Table 4.11. List Attributes

Name	Description
ng-show	Displays a message when triggered. In this case it has been defined to display a “No registered members” message when no members have been added.
ng-hide	Does not display the specified data if certain requirements have not been met. If there are no registered members, it hides the members list so it does not appear on screen.
ng-repeat	Reiterates the list of members that have been submitted so they display in the list. It contains a orderBy attribute, which lists the members in alphabetical order by default.

[Report a bug](#)

4.2.2.4.2. Refresh Button

The refresh button is located under the list. Click the refresh button to manually reload the members list. The design of the button is rendered with CSS.

```

<!-- This is a partial, and is templated by AngularJS -->
<div>
    ...
    <div>
        <!-- The table has a button to manually refresh the values, in
case, for example, someone else adds a member -->
        <input type="button" ng-click="refresh()" name="refresh"
            id="refresh" value="Refresh" class="btn btn-default" />
    </div>
</div>

```

Table 4.12. Refresh Button Attributes

Name	Description
ng-click	Specifies which action the button takes when it is clicked. In this case it contains a refresh command. Clicking the refresh button reloads the values dynamically in the members list.

[Report a bug](#)

Chapter 5. Next

5.1. Further Reading

Each example is packaged with a **README.md** file. This file contains instructions for building and deploying the example.

Additional information about the requirements and configurations of the server can be found in the Red Hat Enterprise Application Server documentation on the Red Hat Customer Portal at <http://access.redhat.com>.

For further demonstrations of jQuery Mobile, visit <http://jquerymobile.com/>.

Additional information on AngularJS and related downloads can be found at <https://angularjs.org/>.

For further information and downloads related to Bootstrap, visit <http://getbootstrap.com>.

To learn more about DeltaSpike, Cordova, and related frameworks, visit the Apache website at <http://www.apache.org/>

[Report a bug](#)

5.2. Additional Examples

This guide has examined two basic examples shipped with Red Hat JBoss Web Framework Kit. There are many other examples for different frameworks. These include demonstrations that use DeltaSpike, Cordova, Backbone, and others. These can be downloaded from the Red Hat Customer Portal at <http://access.redhat.com>.

[Report a bug](#)

Revision History

Revision 2.7.0-1	Fri Jan 16 2015	Zac Dover
Generated for 2.7.0 release		