Ionic in Action

by Jeremy Wilken

Chapter 7

brief contents

7 Advanced techniques for professional apps

This chapter covers

- Customizing Ionic styles using Sass variables
- Handling gestures and events
- Storing and persisting data between app uses
- Modifying your app to adapt to different platforms
- Configuring Ionic default behaviors and settings

This chapter focuses on some advanced techniques that you could incorporate into most apps. As Ionic developers dig deeper into the platform, they'll likely discover that the core components, while useful, can't provide everything well-designed apps need. There should be an element of uniqueness for every app. Just using Ionic's components out of the box without any customization or creativity isn't the best approach for quality apps.

Using these various techniques, you'll be able to design apps that take the strengths of Ionic and extend it for a unique experience. You'll be able to mold an app to adapt its design and behavior for different platforms and improve the user experience through events and using storage.

7.1 Set up chapter project

This chapter is a bit different from the previous chapters where you built a full-scale app. Here the examples are minimized to focus on just the features we're discussing at the time. You can either download the chapter examples or use Git to check out a copy for yourself.

The examples are organized into folders for each section. I'll let you know which folder to look at when you're working with it. For each folder, you should only have to use `ionic serve` from that folder to preview the app in the browser. For some folders, you might consider running it in an emulator or on a device.

7.1.1 Get the code

To get the latest copy of the chapter example, you can download the completed files or check out the repository using GitHub. The following link will download the chapter example as a zip file, which you can extract and then view: https://github.com/ionic-in-action/chapter7/archive/master.zip. To check out the chapter examples, use the following command to clone the repository (this chapter uses the master branch, and doesn't have tags to check out each step):

```
$ git clone https://github.com/ionic-in-action/chapter7.git
$ cd chapter7
```

7.2 Custom Ionic styling using Sass

Ionic comes with a beautiful set of default colors and styles for every component. The examples so far have used very little custom styling and have relied heavily on Ionic's defaults. This is great for learning and shows the power of Ionic, but typically you'll want to customize the design for your needs in some way.

It's best practice to customize the display of your app for your own needs. This is particularly true regarding colors, because you want to give your app its own design and branding. It usually takes some time to consider what works best for your app, and boils down to your vision for the app branding and styling.

I want to reiterate that you shouldn't try to modify the default Ionic CSS file. This is bad practice, and will cause problems when you want to update Ionic. Also, if you try to add new rules to change Ionic styling, it might cause you stress in the long term to maintain the list, especially if you're trying to change the default colors on every component.

In this section you'll use the example inside of the sass directory of your code project. You can refer to it for a working example of how Sass is configured. Let's use Sass to customize Ionic's styling for your own purposes.

7.2.1 Setting up Sass

Sass (Syntactically Awesome Stylesheets) is a CSS preprocessor. Sass is a superset of CSS, which means you can write regular CSS, and Sass understands it. Sass compiles down to CSS, so there's nothing special that the browser needs. But Sass provides a number of features that CSS doesn't (such as variables, nesting, and inheritance) that are very useful for customizing styling. You can learn more about Sass at http://sass-lang.com/.

Ionic has written its styling using Sass, and has used variables extensively. These variables can be declared once and used in multiple places. This makes it possible to change the variable for a color once, and have the color update anywhere it was used. There are hundreds of variables that control the primary color styles, fonts, padding, borders, and more. You can override any of the variables, and then regenerate the CSS with your new values.

First you need to set up your app to be ready for Sass. You need to make sure you've installed your Node dependencies for the project, and then run the `ionic setup` command, which will update a few parts of your app:

```
$ npm install -g gulp
$ ionic setup sass
```

The first command will install Gulp, which is a build tool. Ionic uses Gulp to run tasks, such as converting Sass files into CSS. Gulp uses the gulpfile.js file that Ionic created in your project when you first began. The file is used to manage the Gulp tasks. You can modify (or may have already) the Gulp file with additional tasks that you wish to run, but by default Ionic only creates tasks related to building Sass.

The second command handles setting up a few things required by Sass. It will install any dependencies required to run using the Node package manager (npm), and then check that your Gulp file has a Sass task. Assuming it finds one (it should unless you deleted it sometime), it will run the task and build the CSS for the first time. It will also add a few notes to the ionic.project file. Lastly, it updates the index .html file with a reference to the new customized compiled CSS file (www/css/ionic .app.css). You should verify the new file is correctly linked in the www/index.html file.

It's easiest to do this right away when you start a new project, instead of waiting to do it later. Now let's take a look at how to modify the default variables to customize Ionic.

7.2.2 Customize Ionic with Sass variables

Ionic has hundreds of default variables for different parts of the Ionic styling. The most obvious and useful to change are the nine default color options. The exact number of variables may change as Ionic updates, but you can find the complete list in the www/lib/ionic/scss/_variables.scss file. But don't change them in that file! Just use it as a reference to find the variables you need to customize, and you'll override them in another location.

To customize these variables, you'll modify the sass/scss/ionic.app.scss file. Inside there are some comments, but really there are two commands:

```
// The path to ionicons font files, relative to the built CSS in www/css
$ionicons-font-path: "../lib/ionic/fonts" !default;

// Include all of Ionic
@import "www/lib/ionic/scss/ionic";
```

The first is a variable that correctly links to the font icon directory because this file is in a different place than the default files. The second is an `@import` command, which will import the file found at www/lib/ionic/scss/ionic.scss, which then imports the rest of the Sass files. Any variables that you set before the `@import` command will override the default variables, and this is where you'll assign the new values. Any time you add variables, you'll need to rebuild the Sass files.

Imagine you want to change the default Ionic color to be colors set forth by Google's material design standard. Add the variables before the `@import` command with values to set a new default, as shown in the following listing.

Listing 7.1 Sass variables (sass/scss/ionic.app.scss)

```
$light: #FAFAFA;
$stable: #EEE;
$positive: #3F51B5;
$calm: #2196F3;
$balanced: #4CAF50;
$energized: #FFC107;
$assertive: #F44336;
$royal: #9C27B0;
$dark: #333;

// Include all of Ionic
@import "www/lib/ionic/scss/ionic";

@import "www/scss/app";
```

Sets default variables according to your requirements

Imports Ionic library Sass files; your variables will override existing Ionic ones

Imports a Sass file from app

Anywhere you have used Ionic color classes will now default to the new color, such as the bar-positive or tabs-positive color presets. You must regenerate the CSS first, and you can do so by running the Gulp task:

```
$ gulp sass
```

The file should rebuild in less than a second, and update the www/css/ionic.app.css file with the new color preset. This is pretty awesome, but it can get annoying to remember to always rerun the Gulp task every time you change some styling. There's also a watch task that will automatically rebuild any time you save changes. It has to run in its own command-line window, so it can be running continuously in the background. Open a new command-line window or tab, and run the `gulp watch` command:

```
$ gulp watch
```

Alternatively, when you use `ionic serve` and you set up Sass, the serve command will automatically rebuild the CSS when you change the files and then refresh the CSS in the browser without reloading, so you'll get to see your changes instantly.

Sometimes the `gulp watch` or `ionic serve` commands will hit an error and stop running. Depending on your command line, the serve command might alert you, but if you notice that changes don't seem to be appearing as you make them, verify that the serve command is still running correctly. Occasionally syntax errors in your code can cause the serve command to fail.

7.2.3 Using Sass for your own styling

You can use Sass for your customizations beyond just changing Ionic variables. It's a good idea to write all of your custom styling using Sass as well. There are many features you can use to help, but you can write CSS if that's what you prefer. I personally recommend it, even if you aren't sure that you'll need the extra features of Sass. At a minimum, it will tell you about syntax errors as soon as you try to save the file.

The easiest way to start is to create new files inside of the scss directory and write your styles there. You'll need to add import statements in the ionic.app.scss file to load your scss files, just like it imports the Ionic styles. Note that you'll want to do this after importing the Ionic library. Here's an example of the syntax to import:

```
@import "customizations"
```

You can leave off the file extension if the files are named with .scss. By default, the Gulp task watches any Sass file in the scss directory, so it will start to automatically rebuild when you make changes to any of your styling as long as the watcher is running.

I like to keep my styles in the www directory. I've described before how I have my JavaScript, CSS, and HTML for the same view located in the same folder. This isn't a problem, because you can still use the ionic.app.scss file as the main app file, and then import the files from the www directory. By default, Ionic's Gulp task assumes you'll put all of your Sass files in the scss directory, so the watcher task doesn't look at the www directory for changes. You can change this easily by updating gulpfile.js, where you see the `paths.sass` property defined. This property takes an array of paths (which can include wildcards or glob patterns to match), and this example will add support to watch the www directory as well:

```
var paths = {
  sass: ['./scss/**/*.scss', './www/**/*.scss']
};
```

That's a simple little improvement that allows you to keep your styles together with the HTML and JavaScript for the view. You can organize your code however you like, but it's best to keep it consistent.

7.3 How to support online and offline mode

In all three examples from chapters 4 through 6, you assumed the device is online with an internet connection so you could load data into the app. But in the mobile world, internet connections can be spotty, or users might manually disable it (such as airplane mode for flying). You can do some things to check for the online status of a device and handle the situation when the device is offline:

- Use a Cordova plugin that can ask the device for the current connection status.
- Listen for online and offline events.

Let's demonstrate the second option. The Cordova plugin shouldn't be necessary to detect online or offline status. This section is only concerned about dealing with the presence or absence of a connection. You can investigate the Cordova Network Connection Plugin API to get more details about the type of connection (Wi-Fi verses cellular data, for example) if your use case requires it.

Browsers have support for determining if a browser has a connection or not to a server. The basic code to determine if you're online or offline is fairly simple. The challenge is in designing the application to properly handle both situations in the best way possible.

The following listing adds two event listeners to the application and checks the default status of the network connection to offline/www/js/app.js.

Listing 7.2 Listening for online and offline events (offline/www/js/app.js)

```
angular.module('App', ['ionic'])
.run(function($rootScope, $window) {

  alert($window.navigator.onLine);

  $window.addEventListener('offline', function() {
    alert('offline');
    $rootScope.$digest();
  });

  $window.addEventListener('online', function() {
    alert('online');
    $rootScope.$digest();
  });

})
```

1 Shows alert with online status on load

2 Listens for offline event and shows alert

3 Listens for online event and shows alert

The application example here is only to demonstrate how to create the event listener and check the online status on load. The `$window.navigator.onLine` value **1** returns either `true` or `false`, depending on if the browser has a network connection. Then you add two event listeners to the window **2**, **3** that listen for online or offline state changes. These will only fire when the status changes, but not on load. There's a `$digest()` call because changes that happen inside of a native event listener don't get registered with the Angular digest loop. If you changed something with your Angular application in the event callback, you'd have to end it with a call to `$digest()` to propagate those changes through the app.

To test this, you also need to realize this only tells you if you have a network connection, which may not always be what you want to check. For example, if you use `ionic serve` with live reload, the browser recognizes the Ionic live reload server as a network connection and therefore will not appear offline, even if you disconnect from the primary computer's network connection. The best way to test this is to emulate this without the live reload option, and then disable your computer's network connection to trigger the state change:

```
$ ionic platform add ios
$ ionic emulate ios
```

Once the app has launched in the emulator, you can toggle the connection and you should get an alert with the changed state. While this example is simplistic, it serves to introduce the means to detect changes.

7.4 Handling gesture events in Ionic

Sometimes you'll need to build your own component or interface and you'll want to handle user gestures and events such as swipes and drags. Ionic has several options for you to use to build this support.

Very few apps can be built without creating customized interface elements. Some apps require very unique touch experiences to interact with the elements. I advise against creating complex gestures or relying on users learning specific gestures, because most users have a low threshold for learning an app. If your custom interface doesn't make sense or provide enough contextual information about how to use it, then users are likely to abandon your app. Nobody likes to feel dumb or confused, so be considerate in how you build these interactions by favoring simplicity.

The two main ways Ionic provides support for gestures are with a set of directives to listen for events, or by adding event listeners programmatically into your controllers.

7.4.1 Listen for events with Ionic event directives

The Ionic event directives are a collection that will listen for a particular event and call an expression or function when the event fires. These events include hold, tap, drag, and swipe. The exact timings for these events to fire are listed in the documentation. This section uses the events directory of the project, and figure 7.1 shows the output.

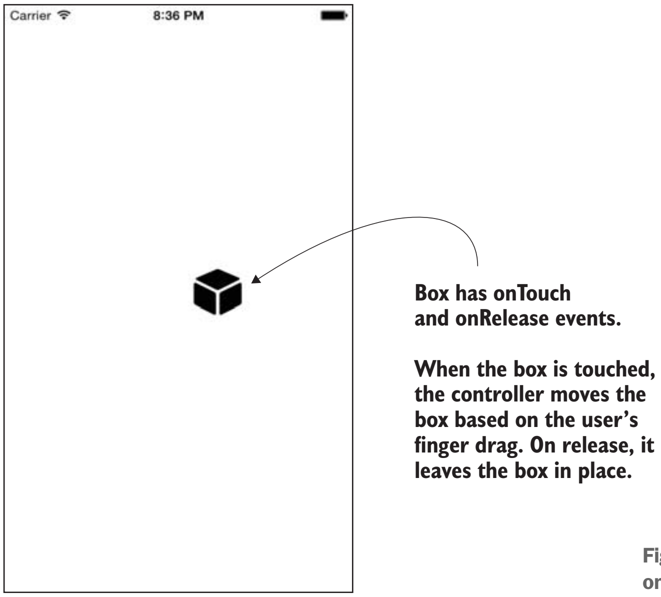
Carrier
8:36 PM
Box has onTouch and onRelease events.
When the box is touched, the controller moves the box based on the user's finger drag. On release, it leaves the box in place.

Figure 7.1 Box that can be moved on touch using event directives

These event directives are the easiest way to listen for events. Let's take a look at an example of how to use them. Listing 7.3 shows a directive that has a combination of events that lets users drag an icon around the screen, and it will also log to the console the number of milliseconds from the time a user touched the item until it's released. The code in the following listing has the directive, which for simplicity is added into the app.js file.

Listing 7.3 Box directive (events/www/js/app.js)

```
angular.module('App', ['ionic'])
.directive('box', function () {
  return {
    restrict: 'E',
    link: function (scope, element) {
      var time = 0, boxX = 0, boxY = 0;
      var leftBound = window.innerWidth - 50;
      var bottomBound = window.innerHeight - 50;
      scope.top = 0;
      scope.left = 0;

      scope.startTouch = function (event) {
        time = event.timeStamp;
      };

      scope.endTouch = function (event) {
        console.log('You held the box for ' +
     (event.timeStamp - time) + 'ms');
        boxX = scope.left;
        boxY = scope.top;
      };

      scope.drag = function (event) {
        var left = boxX + Math.round(event.gesture.deltaX);
        var top = boxY + Math.round(event.gesture.deltaY);

        if (left > leftBound) {
          scope.left = leftBound;
        } else if (left < 0) {
          scope.left = 0;
        } else {
          scope.left = left;
        }
        if (top > bottomBound) {
          scope.top = bottomBound;
        } else if (top < 0) {
          scope.top = 0;
        } else {
          scope.top = top;
        }
      };
    },

    template: '<div id="box" class="icon ion-cube" on-
     touch="startTouch($event)" on-release="endTouch($event)" on-
```

Links function for box directive to add listeners

Sets up some variables to track positions

Touch event handler; tracks start time of drag

Release event handler; tracks total time of drag and logs to console

Drag event handler; moves position of box based on drag and limits boundaries to edge

```
      drag="drag($event)" ng-style="{top: top + \'px\', left: left +
      \'px\'}"></div>'
    }
  })
```

Inline template; box is an icon with events and styles updated by drag

This example creates an icon that can be moved around the screen. It checks that the icon doesn't go outside of the window space; otherwise, it will go anywhere the user drags it. The `onTouch` and `onRelease` event handlers are used to track the total time the user touches the icon, and the `onDrag` event handler does the work to move it around by changing scope variables for the top and left positions that `ngStyle` updates.

To use this, just add a box element to the app. Here you'll add the single box to the app and it will allow the user to begin dragging:

```
<body ng-app="App">
  <box></box>
</body>
```

There's also some CSS required for the positioning to work. CSS rules allow you to position an element absolutely by giving it top and left position values, which come from the drag event:

```
#box {
  position: absolute;
  width: 50px;
  height: 50px;
  font-size: 50px;
  text-align: center;
}
```

There are many different ways you could accomplish similar tasks, but this example highlights how event directives can be used to react to user gestures. I placed this example into a directive because best practice is to use a directive when you need to manipulate the DOM. But you could also have used the directives in a more standard template and put the event handler methods on the controller.

7.4.2 Listen for events with $ionicGesture service

Another way to listen for events is to use the `$ionicGesture` service. This allows you to listen for a wider range of events, but requires a more programmatic approach. The example for this section is found in the gestures directory of the project.

The `$ionicGesture` service needs to be injected into the controller, and then you can declare which events to listen to. You also have to declare the element to which the listener will attach itself. This makes it even more important to use the `$ionicGesture` service inside of a directive when possible, so you have easy access to the element.

You'll use the service to build a simple card that can be swiped off the screen, like you see in figure 7.2. While the user swipes the card to the right or left, it will animate in that direction, and once the user releases, if the card is far enough it will be

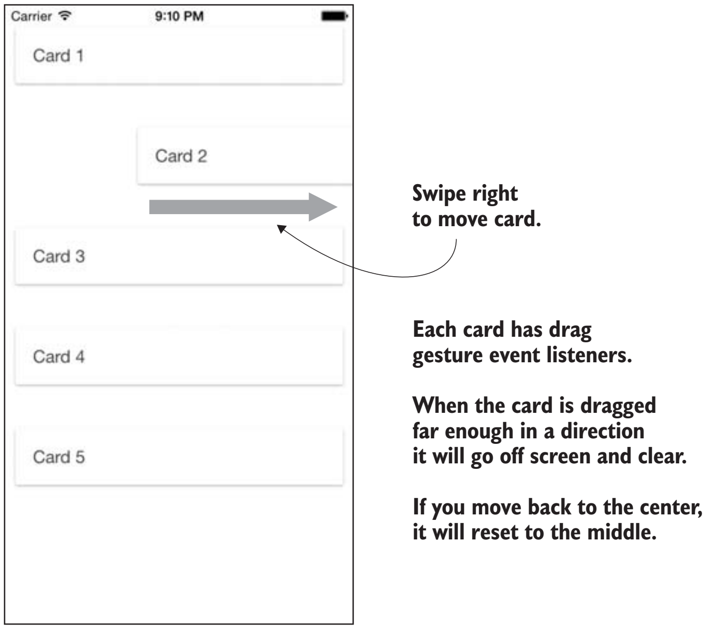
Carrier
9:10 PM
Card 1
Card 2
Card 3
Card 4
Card 5
Swipe right to move card.
Each card has drag gesture event listeners.
When the card is dragged far enough in a direction it will go off screen and clear.
If you move back to the center, it will reset to the middle.

Figure 7.2 Gesture event listeners to swipe cards off the screen

removed from the screen, or it will reset to the center. This is shown in the following listing and found in the chapter 7 project inside of the gestures directory.

Listing 7.4 `$ionicGesture` service (gestures/www/js/app.js)

```
angular.module('App', ['ionic'])
.directive('card', function () {
  return {
    scope: true,
    controller: function ($scope, $element, $ionicGesture, $interval) {
      $scope.left = 0;

      $ionicGesture.on('drag', function (event) {
        $scope.left = event.gesture.deltaX;
        $scope.$digest();
      }, $element);

      $ionicGesture.on('dragend', function (event) {
        if (Math.abs($scope.left) > (window.innerWidth / 3)) {
          $scope.left = ($scope.left < 0) ? -window.innerWidth :
            window.innerWidth;
          $element.remove();
        } else {
          var interval = $interval(function () {
            if ($scope.left < 5 && $scope.left > -5) {
              $scope.left = 0;
              $interval.cancel(interval);
```

Injects $ionicGesture service into controller

Listens for drag event and moves card horizontally while card is dragging

Listens for dragend event and determines if card should be removed or reset

If card is over 33% offscreen, removes it

If card is still near middle, animates it back to center by moving five pixels every five milliseconds

```
          } else {
            $scope.left = ($scope.left < 0) ? $scope.left + 5 :
             $scope.left - 5;
          }
        }, 5);
      }
      $scope.$digest();
    }, $element);
  },
  transclude: true,
  template: '<div class="list card" ng-style="{left: left + \'px\'}"><div
   class="item" ng-transclude>Swipe Me</div></div>'
 }
})
```

If card is still near middle, animates it back to center by moving five pixels every five milliseconds

This card directive attaches two event listeners for `drag` and `dragend`. Technically you're listening for the drag events here, because swipe events don't fire until the swipe has occurred. If you listened for the swipe event, the cards wouldn't move until after the user had already completed the swipe, so it would have a visual delay that might confuse the user. The directive uses the controller to inject the service. When you attach an event listener using the `on` method, you have to pass it at least three things: one of the predefined event names, a callback function to fire when the event is triggered, and the element it should attach to. Because you use a directive here, it has access to the special `$element` service; otherwise, in a controller you'd have to use `angular.element()` to locate the proper element to attach the listener.

For this example to work, you just need to add one line of CSS to www/css/styles.css:

```
.card { position: relative; left: 0; }
```

Then you can add any number of these card directives to your app, and each can be individually swiped off the screen:

```
<body ng-app="App">
  <card>Card 1</card>
     <card>Card 2</card>
  <card>Card 3</card>
  <card>Card 4</card>
  <card>Card 5</card>
</body>
```

The contents of the card element are transcluded inside of the card, which is an Angular feature available to directives. Transclude essentially copies all of the HTML content inside of the directive tag, and places it into the directive template where `ngTransclude` is declared.

This approach is more flexible and is able to support more gesture events than the event directives you looked at earlier. But gesture events require a little more work to set up. In the end, both of them accomplish the same task, so the choice ultimately is preference over style.

7.4.3 Available gesture events

There are a lot of gesture events that you can listen for. Table 7.1 gives a list of the possible gestures, the event name, the directive (if available), and notes about what triggers the gesture event.

Table 7.1 Supported gestures, JavaScript event names, use notes, and possible directives (if available)

Gesture	Event	Directive	Notes
Hold	`hold`	`on-hold`	Touch an element for at least 500 ms
Tap	`tap`	`on-tap`	Touch an element for less than 250 ms
Double tap	`doubletap`		Two touches on same place, within 300 ms
Touch	`touch`	`on-touch`	Fires when a touch is detected
Release	`release`	`on-release`	Fires when a touch is released
Drag	`drag`	`on-drag`	Long touch while moving in any direction, generic
Drag start	`dragstart`		Fires when drag is first detected
Drag end	`dragend`		Fires when drag is released
Drag up	`dragup`	`on-drag-up`	Drag up on *y* axis
Drag down	`dragdown`	`on-drag-down`	Drag down on *y* axis
Drag left	`dragleft`	`on-drag-left`	Drag left on *x* axis
Drag right	`dragright`	`on-drag-right`	Drag right on *x* axis
Swipe	`swipe`	`on-swipe`	Quick touch and flick in any direction, generic
Swipe up	`swipeup`	`on-swipe-up`	Swipe up on *y* axis
Swipe down	`swipedown`	`on-swipe-down`	Swipe down on *y* axis
Swipe left	`swipeleft`	`on-swipe-left`	Swipe left on *x* axis
Swipe right	`swiperight`	`on-swipe-right`	Swipe right on *x* axis
Transform	`transform`		Two fingers touch and move, generic
Transform start	`transformstart`		Fires when a transform is first detected
Transform end	`transformend`		Fires when a transform is released

Table 7.1 Supported gestures, JavaScript event names, use notes, and possible directives (if available) *(continued)*

Gesture	Event	Directive	Notes
Rotate	`rotate`		Two fingers rotating
Pinch	`pinch`		Two fingers pinch and slide together or apart
Pinch in	`pinchin`		Two fingers pinch and slide together
Pinch out	`pinchout`		Two fingers pinch and slide apart

7.5 Storing data for persistence

In the examples from chapters 4 through 6, every time you loaded the app it would reset any changes you had made and start as if it were the first time the app was used. This is obviously annoying and a bad experience for users. For example, users expect that if they mark an item as a favorite, it will stay a favorite. Wouldn't it be great if the app could remember things and pick up where a user left off? The good news is that there are several ways to do this, and I'll show you the primary way that doesn't require any additional plugins.

Because Ionic apps are web applications, apps have the ability to use some of the built-in storage features of the web platform. They have support for `localStorage` for key-value pairs and either Web SQL, IndexedDB, or SQLite for a more robust database.

The general approach for either option is that you'll need to store data, and when the app resumes, the first task will be to load the data from storage. Any app that has the ability to log in will retain some kind of session and user information in storage to properly communicate with a back-end service.

Apps with persistent data should also be designed to handle the situation where data is cleared from the cache. Never assume that stored data will remain indefinitely.

When you store data on a device, you should take precautions against storing anything that users shouldn't be able to see. Anything on a device is potentially viewable by the device owner using debugging tools, but it's reasonable to store private data for that user (such as an OAuth token). Anything that should never be shown to the user should be stored in a server environment (such as a private API key for a web service).

7.5.1 Using localStorage

`localStorage` is a very simple storage option for your app that stores values in the browser cache directory. It's essentially a key-value-pair storage system, or you can think of it like a JavaScript object with the ability to create only one level of properties, which must always be strings. I turn to `localStorage` any time I can because it's the easiest way to store data. In a browser, users can clear the data from `localStorage` any time, but in a hybrid app they aren't able to clear the data unless they're using debugging tools.

`localStorage` is very easy to use but has two major limitations. First, values are stored as a string, regardless of what data type they were before. This means an integer will be turned into a string during storage. This can cause problems if you try to compare a string to a number using strict comparators (such as `"1" === 1`, which is false). Second, there are size limits on the total data you can store, which isn't standardized between browsers. You should consult the documentation for the platforms to see what the current capacity is (Android Browser 4.3 has 2 MB, Safari has 5 MB, and Chrome has 10 MB at time of writing), or visit http://mng.bz/7J3R for a good summary of many storage type limitations. This is a lot of space, but if you exceed it you'll get errors. It can become difficult to manage over time if you're working with a lot of data.

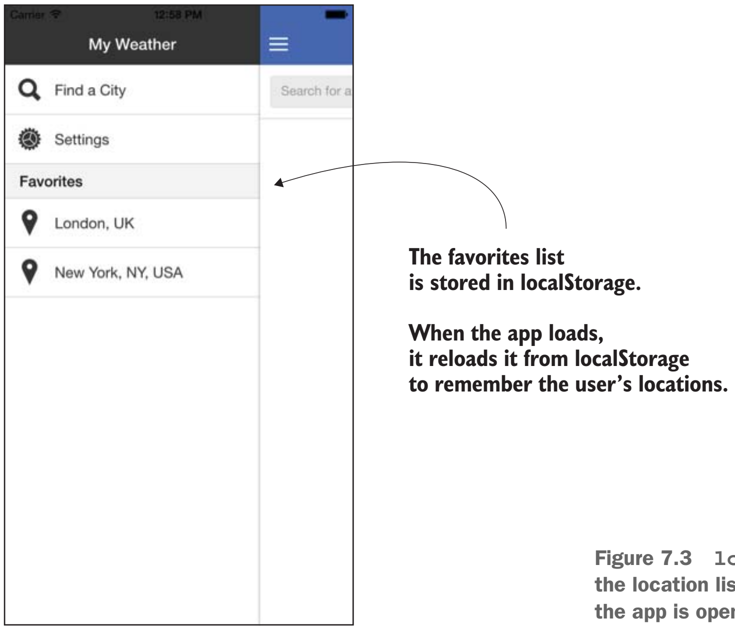
Carrier
12:58 PM
My Weather
Find a City
Settings
Favorites
London, UK
New York, NY, USA
Search for a
The favorites list is stored in localStorage.
When the app loads, it reloads it from localStorage to remember the user's locations.

Figure 7.3 `localStorage` **stores the location list and reloads it when the app is opened.**

But if your storage needs are simple, `localStorage` is often the best solution. Figure 7.3 shows the updated weather app example from chapter 6 to store the locations list in `localStorage` and load it back into the app when it's reloaded. For this section you'll use the code inside of the storage directory. The only change from the chapter 6 example is in the `Locations` service in the storage/www/js/app.js file, shown in the following listing with changes in bold.

Listing 7.5 Save and load from `localStorage` (storage/www/js/app.js)

Creates store() method to handle saving JSON string data into localStorage

```
.factory('Locations', function ($ionicPopup) {
  function store () {
    localStorage.setItem('locations', angular.toJson(Locations.data));
  }
  var Locations = {
    data: [],
    getIndex: function (item) {
      var index = -1;
      angular.forEach(Locations.data, function (location, i) {
        if (item.lat == location.lat && item.lng == location.lng) {
          index = i;
        }
      });
      return index;
    },
    toggle: function (item) {
      var index = Locations.getIndex(item);
      if (index >= 0) {
        $ionicPopup.confirm({
          title: 'Are you sure?',
          template: 'This will remove ' + Locations.data[index].city
        }).then(function (res) {
          if (res) {
            Locations.data.splice(index, 1);
          }
        });
      } else {
        Locations.data.push(item);
        $ionicPopup.alert({
          title: 'Location saved'
        });
      }
      store();
    },
    primary: function (item) {
      var index = Locations.getIndex(item);
      if (index >= 0) {
        Locations.data.splice(index, 1);
        Locations.data.splice(0, 0, item);
      } else {
        Locations.data.unshift(item);
      }
      store();
    }
  };

  try {
    var items = angular.fromJson(localStorage.getItem('locations')) || [];
    Locations.data = items;
  } catch (e) {
    Locations.data = [];
  }

  return Locations;
})
```

Calls store() method after toggling a location from list

Calls store() method after setting a new primary location

When app starts, tries to load data from localStorage or else sets a blank array

You can see `localStorage` is available globally in JavaScript because it's part of the primary JavaScript APIs. You create a `store()` function to abstract the logic because there are multiple places you want to store the data into `localStorage`. When `store()` is called, it takes the array of locations and stores them in `localStorage`, but first converts the array into a JSON string (because `localStorage` only handles strings). Then when the list of locations changes, the app calls the `store()` method to update the cached data.

In the `try-catch` statement, the app attempts to load the data from `localStorage`, and then parse the JSON if it was set. If nothing is set in `localStorage` or if there's an error loading data from `localStorage`, then the location list is set to an empty array.

Now the app will always try to load the list of stored locations and use that instead of starting with a blank list. This is obviously a very good improvement for users, and `localStorage` is very easy to implement.

You can inspect the `localStorage` values inside of the browser developer tools and should see an item with the list there. `localStorage` is app-specific, so the data you store is safe from other apps. But `localStorage` can be inspected by developers, meaning you can't safely store anything that you absolutely cannot allow others to see.

7.5.2 Using Web SQL, IndexedDB, and SQLite

Web SQL, IndexedDB, and SQLite are different types of browser-based databases. Like `localStorage`, the data is stored in the browser's cache system. These options are best for larger amounts of data, or data that you want to be able to directly query. But they're all more difficult to use, and support for them varies across platforms.

Web SQL is similar to a full-featured database with the ability to use SQL to query tables. It allows you to use SQL statements like `SELECT`, `UPDATE`, and so forth. The challenge is that the specification for Web SQL was abandoned back in 2010 when browser vendors couldn't come to agreement on the standard. At the time of writing, iOS and Android both support Web SQL, but it's possible that over time this support may be removed.

IndexedDB is an object store that's somewhere between Web SQL and `localStorage`. It uses a key-value storage like `localStorage`, but items have fields with specific data types and the ability to limit results by requesting certain fields with a given value. At the time of writing, IndexedDB isn't supported by iOS and Android.

SQLite is similar to Web SQL with a SQL-like syntax for loading data in and out of a local database. It also suffers from being abandoned by browsers and standardization bodies for first-class support in browsers. Now most support for SQLite comes through Cordova plugins.

Like `localStorage`, the data from these databases can't be viewed by other apps, but can still be viewed by developers when they debug using their device.

At the time of writing, Web SQL is the option that both iOS and Android support fully with the help of Cordova, and IndexedDB isn't properly supported. But Web SQL has been deprecated since 2010 and likely will be removed in the future, so in time support will likely shift to IndexedDB. But to be doubly sure of what's supported or not, you can check the Cordova storage documentation at http://mng.bz/1UYx.

Check that you're looking at the same version of the documentation that fits the project's Cordova version—you can run `cordova info` to find the version. You can also do a quick test yourself by running one of the following commands to tell you if it's supported or not on a given platform:

```
alert('WebSQL: ' + ((window.openDatabase) ? 'yes' : 'no'));
alert('IndexedDB: ' + ((window.indexedDB) ? 'yes' : 'no'));
```

You need to run these commands on an emulator or device to get the proper message. Just add these lines at the start of your JavaScript to alert you of support for the two options.

7.5.3 Other options from Cordova plugins

Cordova provides plugins to allow you to access additional features on a device. We'll look at some plugins in chapter 8 in depth, but you should know there are many options in the Cordova plugin repository for storage.

The options are varied and ever-changing. Some are able to bring IndexedDB or Web SQL support to all devices, others support different storage systems like SQLite, and others are designed to allow you to store entire files. You can discover storage plugins at http://plugins.cordova.io/npm/index.html.

7.6 Building one app for multiple platforms

One of the best features of building apps with Ionic is the ability to target multiple devices and platforms with one app. But sometimes you need to tweak behavior or design for a particular device or platform.

There are different situations where you need to think about different experiences for different platforms. Ionic provides some of this built into its core. For example, tabs on Android appear differently than they do on iOS. The reason is the Ionic developers wanted to be able to provide the same behavior (tabs) but make it look and feel native to that platform (styling). The tabs do the same thing, just the appearance varies slightly.

There are two main ways to target a platform: change the appearance or change the behavior. Before we look at them, let's dig a bit more into why you should bother building apps that adapt to different platforms.

7.6.1 One size doesn't always fit all

As an app developer, you should consider what makes the app best for your users, not what makes the app the easiest for you to build. Building apps with the exact same behavior on Android and iOS may not always work out for your users, and you should consider this carefully. This is especially true in cases where users are accustomed to certain interactions.

Android and iOS have many differences in their appearance and interaction behaviors. Even different versions of iOS and Android can differ greatly, and over time you can only assume that will continue. Ionic is committed to supporting the modern versions of these platforms, and as the mobile platforms continue to evolve, Ionic will adapt.

You must remember that Ionic is only able to do so much for app developers. Ultimately you're responsible for ensuring that the apps you build work and make sense on different platforms. It's worth spending time with the official native style guides for iOS and Android to familiarize yourself with the differences. Then when you're designing your app, you'll be able to consider the best design for each platform and if you need to design anything specific to a platform. Here's where you can find the official style guides:

- Android style guide: http://developer.android.com/design/style/index.html
- iOS style guide: https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

7.6.2 Adapt styling for a platform or device type

Ionic provides you a simple way to determine which platform or device you're using so you can adapt your app styling as necessary. Ionic determines what platform you're using, and adds a number of classes to the body element:

- `platform-ios` for iOS
- `platform-android` for Android
- `platform-browser` for browsers

These classes give you insight into what type of platform is used. You can also find other classes based on the version number of the platform, for example, `platform-ios-ios7`. In some cases you might need to target a specific version; the version class can provide you that information.

The two major reasons you'll need to use this technique are for providing platform-specific styling, and to address possible display bugs present only on a particular platform. In general, you probably will want to limit the amount of platform-specific design because it will add to the amount of testing you need to do.

You'll use the code from the adaptive-style directory for this section. This is a simple app that just shows the Ionic logo on a background, but depending on the platform, a different background color will display, as shown in figure 7.4. The template for the app is shown in the following listing and the CSS is shown in listing 7.7.

Listing 7.6 Adaptive styling template (adaptive-style/www/index.html)

```
<body ng-app="App">
  <ion-pane>
    <ion-content>
      <span class="icon ion-ionic"></span>
    </ion-content>
  </ion-pane>
</body>
```

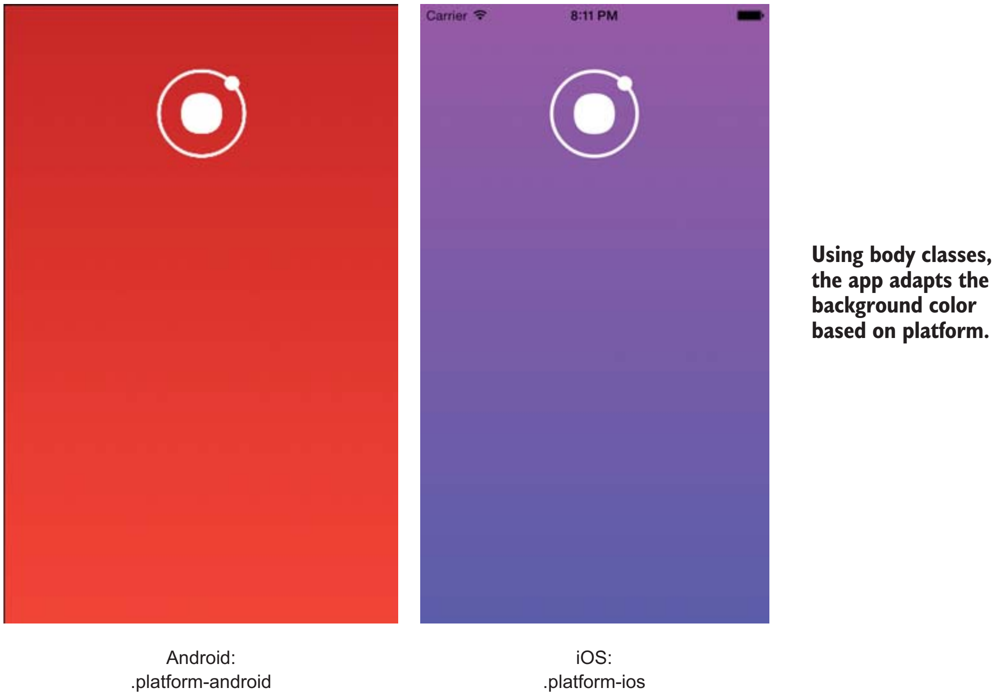
Carrier
8:11 PM
Using body classes, the app adapts the background color based on platform.
Android:
.platform-android
iOS:
.platform-ios

Figure 7.4 Platform-specific styling for Android (left) and iOS (right)

Listing 7.7 Adaptive styling CSS (adaptive-style/www/css/style.css)

```
.scroll {
  text-align: center;
  padding-top: 50px;
}
.ion-ionic {
  font-size: 100px;
  color: #fff;
}
.pane {
  background: #333;
}
.platform-ios .pane {
  background: #C644FC;
  background: -webkit-linear-gradient(top, #C644FC 0%,#5856D6 100%);
  background: linear-gradient(to bottom, #C644FC 0%,#5856D6 100%);
}
.platform-android .pane {
  background: #C62828;
  background: -webkit-linear-gradient(top, #C62828 0%,#F44336 100%);
  background: linear-gradient(to bottom, #C62828 0%,#F44336 100%);
}
```

CSS selector to target iOS only

CSS selector to target Android only

By prefixing your CSS rules with the platform `body` class, you can see the different colors in the background by platform.

7.6.3 Adapt behavior for a platform or device type

You can also adapt the behaviors of the app for a particular platform. For example, you may want to use an action sheet component on iOS but a popover on Android to better fit in with the platform. Ionic can detect which platform is in use, and then modify behaviors accordingly.

The `ionic.Platform` service is available to provide you this information. It provides a list of methods such as `isIOS()` and `isAndroid()` to return a Boolean value if the platform is active, and you can also use the `platform()` method to return the name of the current platform.

In a fairly simple example shown in figure 7.5, pressing the more button (the icon with three dots) will behave differently depending on the platform. You'll check if it's iOS, and show the action sheet; otherwise, show the popover for Android, as in listing 7.8.

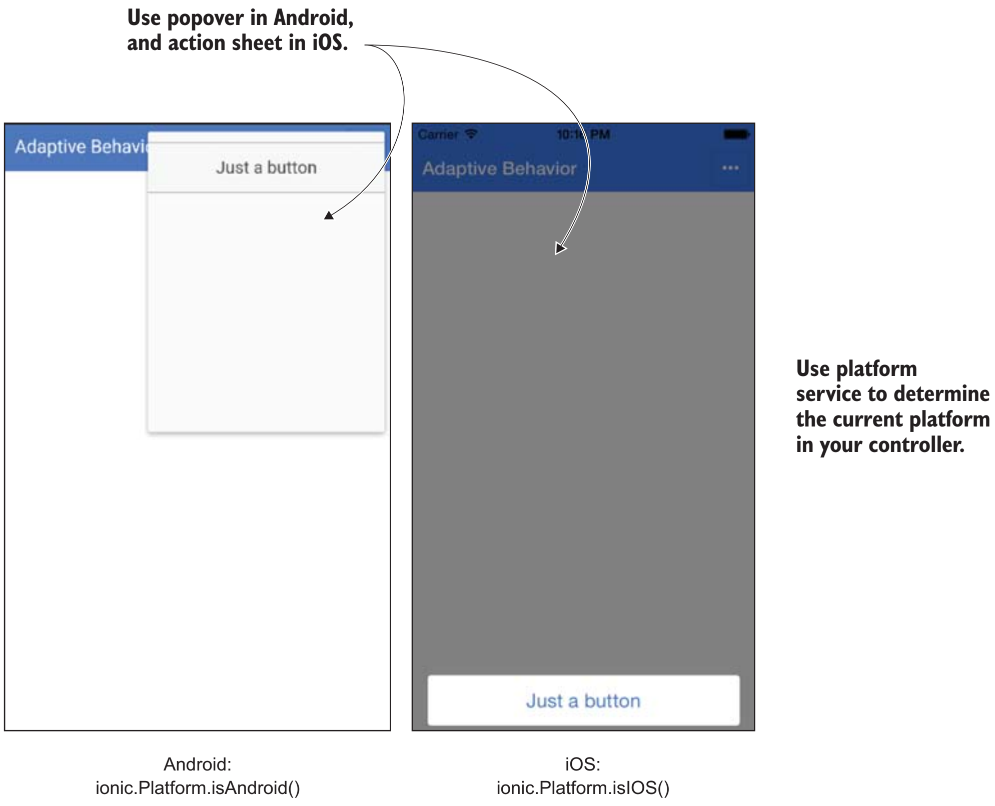
Use popover in Android, and action sheet in iOS.
Adaptive Behavi
Just a button
Carrier
10:1_ PM
Adaptive Behavior
Just a button
Use platform service to determine the current platform in your controller.
Android: ionic.Platform.isAndroid()
iOS: ionic.Platform.isIOS()

Figure 7.5 Based on the platform, you can change the behavior of the button in iOS or Android.

Listing 7.8 Adapting behavior by platform (adaptive-behavior/www/js/app.js)

```
angular.module('App', ['ionic'])
.controller('Controller', function ($scope, $ionicActionSheet,
     $ionicPopover) {
  $scope.more = function (event) {

    if (ionic.Platform.isIOS()) {
      $ionicActionSheet.show({
        buttons: [
          {text: 'Just a button'}
        ],
        buttonClicked: function (index) {
          return true;
        }
      });

    } else {
      var popover = $ionicPopover.fromTemplate('<ion-popover-view>
     <button class="button button-full">Just a button</button>
     </ion-popover-view>');
      popover.show(event);
    }
  }
})
```

Creates controller and injects services

more() method to be called by ngClick

Uses ionic .Platform to determine if iOS

If iOS, shows action sheet with dummy button

Otherwise, shows popover with dummy button

Here you create a controller with a single method that checks if the device is running iOS or not. The `ionic.Platform` service isn't an Angular service, so you don't need to inject it. There's an `$ionicPlatform` service, but it's intended for use with Cordova plugins and doesn't provide information about the current platform.

Once the platform is determined, you choose to show the action sheet or popover. The markup for this example is shown in the following listing.

Listing 7.9 Adaptive behavior template (adaptive-behavior/www/index.html)

```
<body ng-app="App">
  <ion-header-bar align-title="left" class="bar-positive" ng-
     controller="Controller">
    <h1 class="title">Adaptive Behavior</h1>
    <div class="buttons">
      <button class="button" ng-click="more($event)"><span class="icon
     ion-more"></span></button>
    </div>
  </ion-header-bar>
</body>
```

Uses ngClick to call more() method, and passes event for popover

The `ionic.Platform` service is able to provide current information about the platform. It also has a few methods to modify the app behavior, such as the ability to set the app to full screen or exit the app programmatically.

7.7 Modify default behaviors with $ionicConfigProvider

Ionic has a way to modify a number of default behaviors. You were able to modify the default styling using custom Sass variables, and this is the same idea, except you can modify behaviors such as transition types or default navbar title alignment.

The defaults for Ionic are designed to be focused on the correct platform. For example, in a navbar the title will align to the left on Android and will center on iOS to match the style guidelines. But you can force Ionic to render the titles the same regardless of the platform.

The complete list of configurable items is provided in the documentation. You'll build one example to modify the default tabs styling so that it's always striped, and you want the tabs on top. All of the configuration options can be modified in the same manner as you see in this example. In figure 7.6 you can see the result of updated defaults for the tabs.

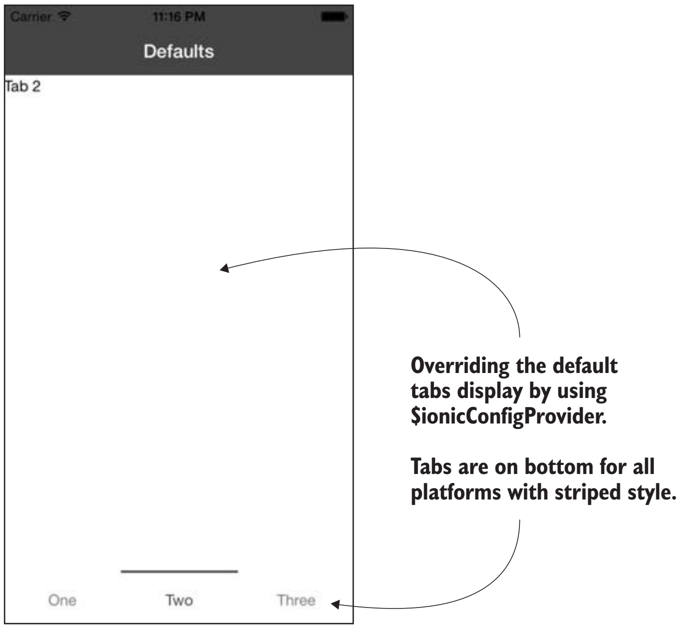
Carrier
11:16 PM
Defaults
Tab 2
One
Two
Three
Overriding the default tabs display by using $ionicConfigProvider.
Tabs are on bottom for all platforms with striped style.

Figure 7.6 Overriding Ionic default values for tabs

The configuration defaults are set in the module `config()` method, in the same method that the states are declared. The following listing has the default configuration changes set for tabs.

Listing 7.10 Updating default configuration (config/www/js/app.js)

```
angular.module('App', ['ionic'])
.config(function($ionicConfigProvider) {
  $ionicConfigProvider.tabs.style('striped').position('bottom');
})
```

Injects $ionicConfigProvider into configuration

Calls tabs settings, and they can be chained in some cases

The `$ionicConfigProvider` is the special service provider for Ionic's configuration, and you're able to update values by calling its methods and passing arguments. In this example you also can chain the two tabs methods together, but if you were changing the default for another aspect unrelated to tabs, chaining wouldn't work. This code will set tabs to be striped and on the bottom, which isn't the default behavior for tabs.

The configurations can still be overridden in the tabs implementation using classes. It might not be necessary to change the default for some things like the tabs display because you can still set the CSS classes on the tabs instance to modify its display. But some of the configurations can't be changed elsewhere, particularly the caching views information.

7.8 Summary

This chapter has given you additional tools and insights into how to build Ionic apps. Let's review the major topics we covered:

- How to build a custom version of the Ionic styles for your app using Sass, and the build processes that Ionic uses with Gulp
- Support for events and gestures, using both event directives and the `$ionicGesture` service
- `localStorage` for persisting data in the app, and other options such as Web SQL and IndexedDB
- Modifying app behavior and display based on the current platform of the device running the app to provide specialized experiences per platform
- Changing the default Ionic configuration to set global parameters for different parts of Ionic

In the next chapter we'll dig deeper into Cordova, and you'll learn how to use the ecosystem of plugins with your Ionic apps.