AngularJS Deployment Essentials

Whether you are brand new to deploying AngularJS applications or a seasoned veteran, this book will provide you with the skills you need to optimally structure, develop, and deploy an AngularJS application.

You’ll learn how to set up your development environment, select the tools that will make up your workflow, optimize your code for production environments, and deploy your application to the leading hosting environments. By sequentially working through the steps in each chapter, you will quickly master your deployment strategy to facilitate a bulletproof workflow. With the help of illustrations that offer a play-by-play account of how to deploy your optimized application, you’ll be able to successfully deploy your next AngularJS application.

By the end of this tutorial, you will have mastered the ability to deploy your AngularJS application to everywhere your users expect to see your product.

Who this book is written for
If you are a web developer, this handy guide will empower you to quickly learn the fundamentals of AngularJS development and deployment.

What you will learn from this book
- Form a bulletproof deployment workflow that can be repeated on future updates
- Deploy your application as a mobile AngularJS app using the Ionic Framework
- Build powerful tool stacks that work together to optimize your workflow
- Employ Grunt to automate repetitive tasks such as image optimization, code minification, and packaging
- Automatically deploy your application from GitHub.com to your Apache server
- Configure your production environments to optimally serve your AngularJS applications
- Scale and enhance your Node.js server using both the Heroku Dashboard and Toolbelt
- Package your application as a Google Chrome app that can be run online and offline

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Our App and Tool Stack'
- A synopsis of the book’s content
- More information on AngularJS Deployment Essentials

About the Author

Zachariah Moreno is a 23-year-old web developer from Sacramento, California, who works for the state government and enjoys contributing to open source projects of the web variety. Zach completed an internship with Google in 2012 and graduated with his BS in web development from the Art Institute of California—Sacramento, in the following year. He has published an article entitled AngularJS Tool Stack in Software Developer’s Journal. Zach can typically be found on Google+ discussing design, developer tools, workflow, camping, golf, and his English bulldog, Gladstone.
AngularJS Deployment Essentials

Where is AngularJS? Since AngularJS's inception, it has been deployed into a multitude of different environments, mostly due to its flexibility and extensibility. Parallel to Angular's rise are developer tools such as Apache Cordova and Chrome packaged apps. These have given developers the opportunity to deploy web applications on platforms that were previously dominated by languages such as Java and C. With these advancements, AngularJS apps can now be found in the Chrome Web Store, Google Play Store, Apple App Store, PlayStation Store, and Firefox Marketplace, all in addition to traditional in-browser web applications. To see examples of other members of the AngularJS community that are built on AngularJS, https://builtwith.angularjs.org/ is a phenomenal place. As AngularJS and its community mature, I suspect that the aforementioned question will become "Where isn't AngularJS?"

Without a talented and thriving developer community, AngularJS would not be the framework that it is today. The most apparent metric I found to measure this phenomenon is the rate at which the new versions of AngularJS are released. At its apex, it reaches 3 minor releases in 1 day and 5 minor releases per month in March 2014. To help the community and contributors keep up with the pace of development, the AngularJS core team maintains the website https://dashboard.angularjs.org/, which provides a real-time snapshot into the current build status of AngularJS's core. Because of its extensible design, contributors have also been able to enhance AngularJS's core through the development of third-party modules, services, filters, and directives. Some examples are the popular REST Angular module for consuming RESTful APIs, the Angular-UI frontend framework, Ionic Framework for mobile UIs, Firebase real-time data storage and syncing, and Yoeman generators for Angular.

What This Book Covers

Chapter 1, Our App and Tool Stack, explains the tooling process, which is one of the hottest topics in today's web development community, and AngularJS has a suite of tools that enhances your developer experience. You will learn about Angular's tool stack, how they fit together, and most notably, how they can help you deploy and maintain future applications.

Chapter 2, Deploying to Apache, explains why Apache, being the most popular web server environment, is a solid platform for hosting most Angular applications. You will get an opportunity to enhance Apache's default configuration to best support your Angular application. We will also discuss best practices to determine whether Apache is suitable for your type of application or whether an alternate infrastructure should be considered.
Chapter 3, *Deploying to Heroku*, discusses Heroku, which is one of the best platforms as a service. Its infrastructure empowers developers to spin up, deploy to, and scale their own Node.js servers at a minimal cost. Deploying to Heroku is a positive developer experience because its tool speed and efficiency are second to none.

Chapter 4, *Deploying to Firebase Hosting*, explains why Firebase has only been offering PaaS for a few months, but the promise of hosting your application code in the same environment as your real-time database has created a lot of interest around the Firebase services. You will learn how Firebase is preconfigured to optimally serve the AngularJS applications and how to further tailor Firebase Hosting to your application's needs.

Chapter 5, *Deploying a Mobile App*, explains why hybrid mobile apps is not a new idea, but the quality and tooling around it has made it a strong contender when building a mobile application. We will take our application's code and build it for Android/iOS using PhoneGap, Apache Cordova, and Cordova Chrome Apps.

Chapter 6, *Deploying as Chrome Apps*, explains why Chrome OS's adoption is growing at an exponential rate and the need for developers to support this platform is parallel in importance. Angular thrives in this environment, and deploying our application as a Chrome App for consumption on the Chrome Web Store/Chrome OS is an exciting frontier for both developers and consumers.

Chapter 7, *Postdeployment*, explains why deployment is no exception to the marketing adage of "rinse, lather, repeat", because it is never finished. Optimizing your workflow with best practices is necessary to ensure the longevity of your application. We will enhance our application and grok how to safely deploy the enhancement into production.

Chapter 8, *Conclusion – AngularJS Deployment Essentials*, explains that harnessing the tools will allow you to deploy a single AngularJS application to new platforms and new customers. We will discuss a few tips that will help you get the best out of these tools.
Before NASA or Space X launches a vessel into the cosmos, there is a tremendous amount of planning and preparation involved. The guiding principle when planning for any successful mission is similar to minimizing efforts and resources while retaining maximum return on the mission. Our principles for development and deployment are no exception to this axiom, and you will gain a firmer working knowledge of how to do so in this chapter.

In this chapter, you will learn how to do the following:

- Minimize efforts and maximize results using a tool stack optimized for AngularJS development
- Access the krakn app via GitHub for deployment in future chapters
- Scaffold an Angular app with Yeoman, Grunt, and Bower
- Set up a local Node.js development server
- Read through krakn’s source code

The right tools for the job

Web applications can be compared to buildings; without tools, neither would be a pleasure to build. This makes tools an indispensable factor in both development and construction. When tools are combined, they form a workflow that can be repeated across any project built with the same stack, facilitating the practices of design, development, and deployment. The argument can be made that it is just as paramount to document workflow as an application’s source code or API.
Our App and Tool Stack

Along with grouping tools into categories based on the phases of building applications, it is also useful to group tools based on the opinions of a respective project—in our case, Angular, Ionic, and Firebase. I call tools grouped into opinionated workflows tool stacks. For example, the remainder of this chapter discusses the tool stack used to build the application that we will deploy across environments in this book. In contrast, if you were to build a Ruby on Rails application, the tool stack would be completely different because the project's opinions are different. Our app is called krakn, and it functions as a real-time chat application built on top of the opinions of Angular, the Ionic Framework, and Firebase.

You can find all of krakn's source code at github.com/zachmoreno/krakn.

Version control with Git and GitHub

Git is a command-line interface (CLI) developed by Linus Torvalds, to use on the famed Linux kernel. Git is mostly popular due to its distributed architecture making it nearly impossible for corruption to occur. Git's distributed architecture means that any remote repository has all of the same information as your local repository. It is useful to think of Git as a free insurance policy for my code.

You will need to install Git using the instructions provided at www.git-scm.com/book/en/Getting-Started-Installing-Git for your development workstation's operating system.

www.GitHub.com has played a notable role in Git's popularization, turning its functionality into a social network focused on open source code contributions. With a pricing model that incentivizes Open Source contributions and licensing for private, GitHub elevated the use of Git to heights never seen before.
If you don't already have an account on GitHub, now is the perfect time to visit www.github.com to provision a free account. I mentioned earlier that krakn's code is available for forking at www.github.com/zachmoreno/krakn. This means that any person with a GitHub account has the ability to view my version of krakn, and clone a copy of their own for further modifications or contributions. In GitHub's web application, forking manifests itself as a button located to the right of the repository's title, which in this case is ZachMoreno/krakn. When you click on the button, you will see an animation that simulates the hardcore forking action. This results in a cloned repository under your account that will have a title to the tune of YourName/krakn.

**Node.js**

Node.js, commonly known as Node, is a community-driven server environment built on Google Chrome's V8 JavaScript runtime that is entirely event driven and facilitates a nonblocking I/O model. According to www.nodejs.org, it is best suited for:

"Data-intensive real-time applications that run across distributed devices."

So what does all this boil down to? Node empowers web developers to write JavaScript both on the client and server with bidirectional real-time I/O. The advent of Node has empowered developers to take their skills from the client to the server, evolving from frontend to full stack (like a caterpillar evolving into a butterfly). Not only do these skills facilitate a pay increase, they also advance the Web towards the same functionality as the traditional desktop or native application.

For our purposes, we use Node as a tool; a tool to build real-time applications in the fewest number of keystrokes, videos watched, and words read as possible. Node is, in fact, a modular tool through its extensible package interface, called **Node Package Manager (NPM)**. You will use NPM as a means to install the remainder of our tool stack.
NPM

The NPM is a means to install Node packages on your local or remote server. NPM is how we will install the majority of the tools and software used in this book. This is achieved by running the $ npm install -g [PackageName] command in your command line or terminal. To search the full list of Node packages, visit www.npmjs.org or run $ npm search [Search Term] in your command line or terminal as shown in the following screenshot:

Yeoman's workflow

Yeoman is a CLI that is the glue that holds your tools into your opinionated workflow. Although the term opinionated might sound off-putting, you must first consider the wisdom and experience of the developers and community before you who maintain Yeoman. In this context, opinionated means a little more than a collection of the best practices that are all aimed at improving your developer's experience of building static websites, single page applications, and everything in between. Opinionated does not mean that you are locked into what someone else feels is best for you, nor does it mean that you must strictly adhere to the opinions or best practices included. Yeoman is general enough to help you build nearly anything for the Web as well as improving your workflow while developing it. The tools that make up Yeoman's workflow are Yo, Grunt.js, Bower, and a few others that are more-or-less optional, but are probably worth your time.

Yo

Apart from having one of the hippest namespaces, Yo is a powerful code generator that is intelligent enough to scaffold most sites and applications. By default, instantiating a yo command assumes that you mean to scaffold something at a project level, but yo can also be scoped more granularly by means of sub-generators. For example, the command for instantiating a new vanilla Angular project is as follows:

$ yo angular radicalApp
Yo will not finish your request until you provide some further information about your desired Angular project. This is achieved by asking you a series of relevant questions, and based on your answers, yo will scaffold a familiar application folder/file structure, along with all the boilerplate code. Note that if you have worked with the angular-seed project, then the Angular application that yo generates will look very familiar to you. Once you have an Angular app scaffolded, you can begin using sub-generator commands. The following command scaffolds a new route, radicalRoute, within radicalApp:

$ yo angular:route radicalRoute

The :route sub-generator is a very powerful command, as it automates all of the following key tasks:

- It creates a new file, radicalApp/scripts/controllers/radicalRoute.js, that contains the controller logic for the radicalRoute view
- It creates another new file, radicalApp/views/radicalRoute.html, that contains the associated view markup and directives
- Lastly, it adds an additional route within, radicalApp/scripts/app.js, that connects the view to the controller

Additionally, the sub-generators for yo angular include the following:

:controller
:directive
:filter
:service
:provider
:factory
:value
:constant
:decorator
:view

All the sub-generators allow you to execute finer detailed commands for scaffolding smaller components when compared to :route, which executes a combination of sub-generators.
Our App and Tool Stack

Installing Yo

Within your workstation's terminal or command-line application type, insert the following command, followed by a return:

$ npm install -g yo

If you are a Linux or Mac user, you might want to prefix the command with `sudo`, as follows:

$ sudo npm install -g yo

Grunt

Grunt.js is a task runner that enhances your existing and/or Yeoman's workflow by automating repetitive tasks. Each time you generate a new project with `yo`, it creates a `/Gruntfile.js` file that wires up all of the curated tasks. You might have noticed that installing Yo also installs all of Yo's dependencies. Reading through `/Gruntfile.js` should incite a fair amount of awe, as it gives you a snapshot of what is going on under the hood of Yeoman's curated Grunt tasks and its dependencies.

Generating a vanilla Angular app produces a `/Gruntfile.js` file, as it is responsible for performing the following tasks:

- It defines where Yo places Bower packages, which is covered in the next section
- It defines the path where the `grunt build` command places the production-ready code
- It initializes the watch task to run:
  - JSHint when JavaScript files are saved
  - Karma's test runner when JavaScript files are saved
  - Compass when SCSS or SASS files are saved
  - The saved `/Gruntfile.js` file
- It initializes LiveReload when any HTML or CSS files are saved
- It configures the `grunt server` command to run a Node.js server on `localhost:9000`, or to show test results on `localhost:9001`
- It autoprefixes CSS rules on LiveReload and `grunt build`
- It renames files for optimizing browser caching
- It configures the `grunt build` command to minify images, SVG, HTML, and CSS files or to safely minify Angular files
Let us pause for a moment to reflect on the amount of time it would take to find, learn, and implement each dependency into our existing workflow for each project we undertake. Ok, we should now have a greater appreciation for Yeoman and its community.

For the vast majority of the time, you will likely only use a few Grunt commands, which include the following:

$ grunt server
$ grunt test
$ grunt build

**Bower**

If Yo scaffolds our application's structure and files, and Grunt automates repetitive tasks for us, then what does Bower bring to the party? Bower is web development's missing package manager. Its functionality parallels that of Ruby Gems for the Ruby on Rails MVC framework, but is not limited to any single framework or technology stack. The explicit use of Bower is not required by the Yeoman workflow, but as I mentioned previously, the use of Bower is configured automatically for you in your project's /Gruntfile.js file.

How does managing packages improve our development workflow? With all of the time we've been spending in our command lines and terminals, it is handy to have the ability to automate the management of third-party dependencies within our application. This ability manifests itself in a few simple commands, the most ubiquitous being the following command:

$ bower install [PackageName] --save

With this command, Bower will automate the following steps:

1. First, search its packages for the specified package name
2. Download the latest stable version of the package if found
3. Move the package to the location defined in your project's /Gruntfile.js file, typically a folder named /bower_components
4. Insert dependencies in the form of `<link>` elements for CSS files in the document's `<head>` element, and `<script>` elements for JavaScript files right above the document's closing `</body>` tag, to the package's files within your project's /index.html file
Our App and Tool Stack

This process is one that web developers are more than familiar with because adding a JavaScript library or new dependency happens multiple times within every project. Bower speeds up our existing manual process through automation and improves it by providing the latest stable version of a package and then notifying us of an update if one is available. This last part, "notifying us of an update if … available", is important because as a web developer advances from one project to the next, it is easy to overlook keeping dependencies as up to date as possible. This is achieved by running the following command:

$ bower update

This command returns all the available updates, if available, and will go through the same process of inserting new references where applicable.

Bower.io includes all of the documentation on how to use Bower to its fullest potential along with the ability to search through all of the available Bower packages.

Searching for available Bower packages can also be achieved by running the following command:

$ bower search [SearchTerm]

If you cannot find the specific dependency for which you search, and the project is on GitHub, consider contributing a bower.json file to the project's root and inviting the owner to register it by running the following command:

$ bower register [ThePackageName] [GitEndpoint]

Registration allows you to install your dependency by running the next command:

$ bower install [ThePackageName]

The Ionic framework

The Ionic framework is a truly remarkable advancement in bridging the gap between web applications and native mobile applications. In some ways, Ionic parallels Yeoman where it assembles tools that were already available to developers into a neat package, and structures a workflow around them, inherently improving our experience as developers.

If Ionic is analogous to Yeoman, then what are the tools that make up Ionic's workflow? The tools that, when combined, make Ionic noteworthy are Apache Cordova, Angular, Ionic's suite of Angular directives, and Ionic's mobile UI framework.
Batarang

An invaluable piece to our Angular tool stack is the Google Chrome Developer Tools extension, Batarang, by Brian Ford. Batarang adds a third-party panel (on the right-hand side of Console) to DevTools that facilitates Angular’s specific inspection in the event of debugging. We can view data in the scopes of each model, analyze each expression's performance, and view a beautiful visualization of service dependencies all from within Batarang. Because Angular augments the DOM with ng-attributes, it also provides a Properties pane within the Elements panel, to inspect the models attached to a given element's scope. The extension is easy to install from either the Chrome Web Store or the project’s GitHub repository and inspection can be enabled by performing the following steps:

1. Firstly, open the Chrome Developer Tools.
2. You should then navigate to the AngularJS panel.
3. Finally, select the Enable checkbox on the far right tab.

Your active Chrome tab will then be reloaded automatically, and the AngularJS panel will begin populating the inspection data. In addition, you can leverage the Angular pane with the Elements panel to view Angular-specific properties at an elemental level, and observe the $scope variable from within the Console panel.

Sublime Text and Editor integration

While developing any Angular app, it is helpful to augment our workflow further with Angular-specific syntax completion, snippets, go to definition, and quick panel search in the form of a Sublime Text package. Perform the following steps:

1. If you haven’t installed Sublime Text already, you need to first install Package Control. Otherwise, continue with the next step.
3. Then, you need to select the Package Control: Install Package option.
4. Finally, type angularjs and press Enter on your keyboard.

In addition to support within Sublime, Angular enhancements exist for lots of popular editors, including WebStorm, Coda, and TextMate.
Krakn
As a quick refresher, krakn was constructed using all of the tools that are covered in this chapter. These include Git, GitHub, Node.js, NPM, Yeoman's workflow, Yo, Grunt, Bower, Batarang, and Sublime Text. The application builds on Angular, Firebase, the Ionic Framework, and a few other minor dependencies.

The workflow I used to develop krakn went something like the following. Follow these steps to achieve the same thing. Note that you can skip the remainder of this section if you'd like to get straight to the deployment action, and feel free to rename things where necessary.

Setting up Git and GitHub
The workflow I followed while developing krakn begins with initializing our local Git repository and connecting it to our remote master repository on GitHub. In order to install and set up both, perform the following steps:

1. Firstly, install all the tool stack dependencies, and create a folder called krakn.
2. Following this, run `$ git init`, and you will create a README.md file.
3. You should then run `$ git add README.md` and commit README.md to the local master branch.
4. You then need to create a new remote repository on GitHub called ZachMoreno/krakn.
5. Following this, run the following command:
   ```
   $ git remote add origin git@github.com:[YourGitHubUserName] / krakn.git
   ```
6. Conclude the setup by running `$ git push –u origin master`.

Scaffolding the app with Yo
Scaffolding our app couldn't be easier with the `yo ionic` generator. To do this, perform the following steps:

1. Firstly, install Yo by running `$ npm install -g yo`.
2. After this, install `generator-ionicjs` by running `$ npm install -g generator-ionicjs`.
3. To conclude the scaffolding of your application, run the `yo ionic` command.
Development

After scaffolding the folder structure and boilerplate code, our workflow advances to the development phase, which is encompassed in the following steps:

1. To begin, run `grunt server`.
2. You are now in a position to make changes, for example, these being deletions or additions.
3. Once these are saved, LiveReload will automatically reload your browser.
4. You can then review the changes in the browser.
5. Repeat steps 2-4 until you are ready to advance to the predeployment phase.

Views, controllers, and routes

Being a simple chat application, krakn has only a handful of views/routes. They are login, chat, account, menu, and about. The menu view is present in all the other views in the form of an off-canvas menu.

The login view

The default view/route/controller is named login. The login view utilizes the Firebase's Simple Login feature to authenticate users before proceeding to the rest of the application. Apart from logging into krakn, users can register a new account by entering their desired credentials. An interesting part of the login view is the use of the `ng-show` directive to toggle the second password field if the user selects the register button. However, the `ng-model` directive is the first step here, as it is used to pass the input text from the view to the controller and ultimately, the Firebase Simple Login. Other than the Angular magic, this view uses the `ion-view` directive, grid, and buttons that are all core to Ionic.

Each view within an Ionic app is wrapped within an `ion-view` directive that contains a `title` attribute as follows:

```html
<ion-view title="Login">
</ion-view>
```

The login view uses the standard input elements that contain a `ng-model` attribute to bind the input's value back to the controller's `$scope` as follows:

```html
<input type="text" placeholder="you@email.com" ng-model="data.email" />

<input type="password" placeholder="embody strength" ng-model="data.pass" />
```
Our App and Tool Stack

<input type="password" placeholder="embody strength" ng-model="data.confirm" />

The Log in and Register buttons call their respective functions using the ng-click attribute, with the value set to the function's name as follows:

<button class="button button-block button-positive" ng-click="login()" ng-hide="createMode">Log In</button>

The Register and Cancel buttons set the value of $scope.createMode to true or false to show or hide the correct buttons for either action:

<button class="button button-block button-calm" ng-click="createMode = true" ng-hide="createMode">Register</button>
<button class="button button-block button-calm" ng-show="createMode" ng-click="createAccount()">Create Account</button>
<button class="button button-block button-assertive" ng-show="createMode" ng-click="createMode = false">Cancel</button>

$scope.err is displayed only when you want to show the feedback to the user:

<p ng-show="err" class="assertive text-center">{{err}}</p>

</ion-view>

The login controller is dependent on Firebase's loginService module and Angular's core $location module:

controller('LoginCtrl', ['$scope', 'loginService', '$location',
  function($scope, loginService, $location) {

Ionic's directives tend to create isolated scopes, so it was useful here to wrap our controller's variables within a $scope.data object to avoid issues within the isolated scope as follows:

$scope.data = {
  "email"   : null,
  "pass"    : null,
  "confirm" : null,
  "createMode" : false
}
The `login()` function easily checks the credentials before authentication and sends feedback to the user if needed:

```javascript
$scope.login = function(cb) {
    $scope.err = null;
    if( !$scope.data.email ) {
        $scope.err = 'Please enter an email address';
    } else if( !$scope.data.pass ) {
        $scope.err = 'Please enter a password';
    }

    if( credentials are sound, we send them to Firebase for authentication, and when we receive a success callback, we route the user to the chat view using `$location.path()` as follows:

```javascript
    else { 
        loginService.login($scope.data.email, 
        $scope.data.pass, function(err, user) {
            $scope.err = err? err + '' : null;
            if( !err ) {
                cb && cb(user);
                $location.path('krakn/chat');
            }
        });
    }
};
```

The `createAccount()` function works in much the same way as `login()`, except that it ensures that the users don't already exist before adding them to your Firebase and logging them in:

```javascript
$scope.createAccount = function() {
    $scope.err = null;
    if( assertValidLoginAttempt() ) {
        loginService.createAccount($scope.data.email, 
        $scope.data.pass, 
        function(err, user) {
            if( !err ) {
                $scope.err = err? err + '' : null;
            } else {
                // must be logged in before I can write to my profile
                $scope.login(function() {
```
Our App and Tool Stack

loginService.createProfile(user.uid, user.email);
$location.path('krakn/account');
});
}
});
}

The `assertValidLoginAttempt()` function is a function used to ensure that no errors are received through the account creation and authentication flows:

```javascript
function assertValidLoginAttempt() {
  if (!$scope.data.email) {
    $scope.err = 'Please enter an email address';
  } else if (!$scope.data.pass) {
    $scope.err = 'Please enter a password';
  } else if ($scope.data.pass !== $scope.data.confirm) {
    $scope.err = 'Passwords do not match';
  }
  return !$scope.err;
}
```

The chat view

Keeping vegan practices aside, the meat and potatoes of krakn's functionality lives within the chat view/controller/route. The design is similar to most SMS clients, with the input in the footer of the view and messages listed chronologically in the main content area. The `ng-repeat` directive is used to display a message every time a message is added to the messages collection in Firebase. If you submit a message successfully, unsuccessfully, or without any text, feedback is provided via the placeholder attribute of the message input.

There are two filters being utilized within the chat view: `orderByPriority` and `timeAgo`. The `orderByPriority` filter is defined within the `firebase` module that uses the Firebase object IDs that ensure objects are always chronological.

The `timeAgo` filter is an open source Angular module that I found. You can access it at [www.jsfiddle.net/i_woody/cnL5T/](http://www.jsfiddle.net/i_woody/cnL5T/).
The `ion-view` directive is used once again to contain our chat view:

```html
<ion-view title="Chat">

Our list of messages is composed using the `ion-list` and `ion-item` directives, in addition to a couple of key attributes. The `ion-list` directive gives us some nice interactive controls using the `option-buttons` and `can-swipe` attributes. This results in each list item being swipable to the left, revealing our `option-buttons` as follows:

```html
<ion-list option-buttons="itemButtons" can-swipe="true" ng-show="messages">

Our workhorse in the chat view is the trusty `ng-repeat` directive, responsible for persisting our data from Firebase to our service to our controller and into our view and back again:

```html
<ion-item ng-repeat="message in messages | orderByPriority" item="item" can-swipe="true">

Then, we bind our data into vanilla HTML elements that have some custom styles applied to them:

```html
<h2 class="user">{{ message.user }}</h2>

The third-party `timeago` filter converts the time into something such as, "5 min ago", similar to Instagram or Facebook:

```html
<small class="time">{{ message.receivedTime | timeago }}</small>
<p class="message">{{ message.text }}</p>
```

A vanilla input element is used to accept chat messages from our users. The input data is bound to `$scope.data.newMessage` for sending data to Firebase and `$scope.feedback` is used to keep our users informed:

```html
<input type="text" class="{{ feeling }}" placeholder="{{ feedback }}" ng-model="data.newMessage" />
```

When you click on the `send/submit` button, the `addMessage()` function sends the message to your Firebase, and adds it to the list of chat messages, in real time:

```html
<button type="submit" id="chat-send" class="button button-small button-clear" ng-click="addMessage()"><span class="ion-android-send"></span></button>
```

```html
</ion-view>
The ChatCtrl controller is dependant on a few more modules other than our LoginCtrl, including syncData, $ionicScrollDelegate, $ionicLoading, and $rootScope:

```javascript
controller('ChatCtrl', ['$scope', 'syncData', '$ionicScrollDelegate', '$ionicLoading', '$rootScope',
  function($scope, syncData, $ionicScrollDelegate, $ionicLoading, $rootScope) {

The userName variable is derived from the authenticated user's e-mail address (saved within the application's $rootScope) by splitting the e-mail and using everything before the @ symbol:

```javascript
var userEmail = $rootScope.auth.user.email
userName = userEmail.split('@');
```

Avoid isolated scope issue in the same fashion, as we did in LoginCtrl:

```javascript
$scope.data = {
  newMessage : null,
  user : userName[0]
}
```

Our view will only contain the latest 20 messages that have been synced from Firebase:

```javascript
$scope.messages = syncData('messages', 20);
```

When a new message is saved/synced, it is added to the bottom of the ng-repeated list, so we use the $ionicScrollDelegate variable to automatically scroll the new message into view on the display as follows:

```javascript
$ionicScrollDelegate.scrollBottom(true);
```

Our default chat input placeholder text is something on your mind?:

```javascript
$scope.feedback = 'something on your mind?';
// displays as class on chat input placeholder
$scope.feeling = 'stable';
```

If we have a new message and a valid username (shortened), then we can call the $add() function, which syncs the new message to Firebase and our view is as follows:

```javascript
$scope.addMessage = function() {
  if( $scope.data.newMessage
    && $scope.data.user ) {
    // new data elements cannot be synced without adding them to FB Security Rules
    $scope.messages.$add({
      text : $scope.data.newMessage,
```
user : $scope.data.user,
        receivedTime : Number(new Date())
    });
    // clean up
    $scope.data.newMessage = null;

On a successful sync, the feedback updates say Done! What's next?, as shown in the following code snippet:

```javascript
$scope.feedback = 'Done! What\'s next?';
$scope.feeling = 'stable';
} else {
    $scope.feedback = 'Please write a message before sending';
    $scope.feeling = 'assertive';
}
```

```javascript
$ionicScrollDelegate.scrollBottom(true);
```

**The account view**

The account view allows the logged in users to view their current name and e-mail address along with providing them with the ability to update their password and e-mail address. The input fields interact with Firebase in the same way as the chat view does using the syncData method defined in the firebase module:

```html
<ion-view title="Account" left-buttons="leftButtons">
    The $scope.user object contains our logged in user's account credentials, and we bind them into our view as follows:

    ```html
    <p>{{ user.name }}</p>
    ...
    <p>{{ user.email }}</p>
    ```

    The basic account management functionality is provided within this view; so users can update their e-mail address and or password if they choose to, using the following code snippet:

    ```html
    <input type="password" ng-keypress="reset()" ng-model="oldpass" />
    ...
    <input type="password" ng-keypress="reset()" ng-model="newpass" />
    ```
Both the `updatePassword()` and `updateEmail()` functions work in much the same fashion as our `createAccount()` function within the `LoginCtrl` controller. They check whether the new e-mail or password is not the same as the old, and if all is well, it syncs them to Firebase and back again:

```html
<button class="button button-block button-calm" ng-click="updatePassword()">update password</button>
...
<p class="error" ng-show="err">{{err}}</p>
<p class="good" ng-show="msg">{{msg}}</p>
...
<input type="text" ng-keypress="reset()" ng-model="newemail"/>
...
<input type="password" ng-keypress="reset()" ng-model="pass"/>
...
<button class="button button-block button-calm" ng-click="updateEmail()">update email</button>
...
<p class="error" ng-show="emailerr">{{emailerr}}</p>
<p class="good" ng-show="emailmsg">{{emailmsg}}</p>
...
</ion-view>

The menu view

Within krakn/app/scripts/app.js, the menu route is defined as the only abstract state. Because of its abstract state, it can be presented in the app along with the other views by the `ion-side-menus` directive provided by Ionic. You might have noticed that only two menu options are available before signing into the application and that the rest appear only after authenticating. This is achieved using the `ng-show-auth` directive on the chat, account, and log out menu items. The majority of the options for Ionic's directives are available through attributes making them simple to use. For example, take a look at the `animation="slide-left-right"` attribute. You will find Ionic's use of custom attributes within the directives as one of the ways that the Ionic Framework is setting itself apart from other options within this space.

The `ion-side-menu` directive contains our menu list similarly to the one we previously covered, the `ion-view` directive, as follows:

```html
<ion-side-menus>
  <ion-pane ion-side-menu-content>
    <ion-nav-bar class="bar-positive">
      ...
    </ion-nav-bar>
  </ion-pane>
</ion-side-menus>
```
Our back button is displayed by including the `ion-nav-back-button` directive within the `ion-nav-bar` directive:

```html
<ion-nav-back-button class="button-clear"><i class="icon ion-chevron-left"></i> Back</ion-nav-back-button>
</ion-nav-bar>
```

Animations within Ionic are exposed and used through the `animation` attribute, which is built atop the `ngAnimate` module. In this case, we are doing a simple animation that replicates the experience of a native mobile app:

```html
<ion-nav-view name="menuContent" animation="slide-left-right"></ion-nav-view>
</ion-pane>

<ion-side-menu side="left">
    <header class="bar bar-header bar-positive">
        <h1 class="title">Menu</h1>
    </header>
    <ion-content class="has-header">
        A simple `ion-list` directive/element is used to display our navigation items in a vertical list. The `ng-show` attribute handles the display of menu items before and after a user has authenticated. Before a user logs in, they can access the navigation, but only the `About` and `Log In` views are available until after successful authentication.

```html
<ion-list>
    <ion-item nav-clear menu-close href="#/app/chat" ng-show-auth='"login"'>
        Chat
    </ion-item>

    <ion-item nav-clear menu-close href="#/app/about">
        About
    </ion-item>

    <ion-item nav-clear menu-close href="#/app/login" ng-show-auth=['logout','error']>
        Log In
    </ion-item>
</ion-list>
```

The `Log Out` navigation item is only displayed once logged in, and upon a click, it calls the `logout()` function in addition to navigating to the login view:

```html
<ion-item nav-clear menu-close href="#/app/login" ng-click="logout()" ng-show-auth='"login"'>
    Log Out
</ion-item>
```
The MenuCtrl controller is the simplest controller in this application, as all it contains is the toggleMenu() and logout() functions:

```javascript
controller("MenuCtrl", ['$scope', 'loginService', '$location', '$ionicScrollDelegate', function($scope, loginService, $location, $ionicScrollDelegate) {
  $scope.toggleMenu = function() {
    $scope.sideMenuController.toggleLeft();
  };

  $scope.logout = function() {
    loginService.logout();
    $scope.toggleMenu();
  }]
```}

The about view
The about view is 100 percent static, and its only real purpose is to present the credits for all the open source projects used in the application.

Global controller constants
All of krakn's controllers share only two dependencies: ionic and ngAnimate. Because Firebase's modules are defined within /app/scripts/app.js, they are available for consumption by all the controllers without the need to define them as dependencies. Therefore, the firebase service's syncData and loginService are available to ChatCtrl and LoginCtrl for use.

The syncData service is how krakn utilizes three-way data binding provided by www.krakn.firebaseio.com. For example, within the ChatCtrl controller, we use syncData('messages', 20) to bind the latest twenty messages within the messages collection to $scope for consumption by the chat view. Conversely, when a ng-click user clicks the submit button, we write the data to the messages collection by use of the syncData.$add() method inside the $scope.addMessage() function:

```javascript
$scope.addMessage = function() {
  if(...) { $scope.messages.$add({ ... });
  }
};
```
Models and services

The model for krakn is www.krakn.firebaseio.com. The services that consume krakn's Firebase API are as follows:

- The **firebase** service in krakn/app/scripts/service.firebase.js
- The **login** service in krakn/app/scripts/service.login.js
- The **changeEmail** service in krakn/app/scripts/changeEmail.firebase.js

The **firebase** service defines the **syncData** service that is responsible for routing data bidirectionally between krakn/app/bower_components/angularfire.js and our controllers. Please note that the reason I have not mentioned angularfire.js until this point is that it is basically an abstract data translation layer between firebaseio.com and Angular applications that intend on consuming data as a service.
Predeployment

Once the majority of an application's development phase has been completed, at least for the initial launch, it is important to run all of the code through a build process that optimizes the file size through compression of images and minification of text files. This piece of the workflow was not overlooked by Yeoman and is available through the use of the $ grunt build command. As mentioned in the section on Grunt, the /Gruntfile.js file defines where built code is placed once it is optimized for deployment. Yeoman's default location for built code is the /dist folder, which might or might not exist depending on whether you have run the grunt build command before.

Summary

In this chapter, we discussed the tool stack and workflow used to build the app that we will deploy in the forthcoming chapters. Together, Git and Yeoman formed a solid foundation for building krakn. Git and GitHub provided us with distributed version control and a platform for sharing the application's source code with you and the world. Yeoman facilitated the remainder of the workflow: scaffolding with Yo, automation with Grunt, and package management with Bower. With our app fully scaffolded, we were able to build our interface with the directives provided by the Ionic Framework, and wire up the real-time data synchronization forged by our Firebase instance. With a few key tools, we were able to minimize our development time while maximizing our return.

Next, we will take our freshly constructed real-time chat app, krakn, and deploy it to an Apache web server for consumption over the Web. Once manually deployed to Apache, we will discuss automating future deployments and optimizing the environment for an Angular application.
Where to buy this book

You can buy AngularJS Deployment Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.