PhoneGap for Enterprise

This book covers the necessary concepts you'll need to build useful, rich, secure enterprise apps for mobile devices, and also explains portions of an example enterprise app development so that you have a working demonstration of the concepts in the book.

The book focuses heavily on security concepts such as authentication, SSL and certificates, source code protection, local data store encryption, and so on, as ensuring a secure channel of communication between the user and the enterprise server is critical. The book explores the entire app's design, covering the data model and business logic you'll use for your app. It then moves on to cover how to build middle-tier communication between the middle tier and the mobile device, handling application and network events (such as a network going offline), and culminates by showing you how to work with various rich presentation technologies.

This book is invaluable for anyone looking to learn how to create secure mobile apps for enterprise environments in an easy way with lots of useful examples.

Who this book is written for

This book is intended for developers who wish to use PhoneGap to develop useful, rich, secure mobile applications for their enterprise environment. The book assumes you have working knowledge of PhoneGap, HTML5, CSS3, and JavaScript, and a reasonable understanding of networking and n-tier architectures.

What you will learn from this book

- Develop cross-platform mobile enterprise apps using PhoneGap
- Understand the PhoneGap system and app architecture
- Communicate between the mobile device and the backend using XHR and WebSockets for data transmission
- Build a middle tier using Node.js to connect to the data store
- Discover how PhoneGap allows you to whitelist trusted domains
- Handle online/offline states by learning how to handle application and network status events

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 7 “Push Notifications”
- A synopsis of the book’s content
- More information on PhoneGap for Enterprise

About the Author

Kerri Shotts has worked with computers for nearly 25 years. Her love for technology and programming started when she was introduced to her first computer, which was a Commodore 64. She obtained a degree in computer science at college and eventually became a software test engineer. Later on, she worked as an Oracle Database Administrator for several years. Now, she owns her own company and works as a technology consultant. Kerri is responsible for creating, designing, and maintaining custom applications (both desktop and mobile), websites, graphics, and logos for her clients. She has been actively involved with PhoneGap for several years, and has written several native and hybrid applications. You can find her blog posts at http://www.photokandy.com. Kerri is also active on the Google Group for PhoneGap and Stack Overflow. When she isn't coding or writing, she enjoys photography, music, and fishkeeping. She is the author of three other books published by Packt Publishing, including PhoneGap 3.x Mobile Application Development Hotshot.
PhoneGap for Enterprise

Mobile devices are virtually ubiquitous among the enterprise workforce, and employees are becoming increasingly mobile. This book discusses how the enterprise workforce can take advantage of Apache Cordova/Adobe PhoneGap applications to create cross-platform hybrid mobile applications that can serve the varied needs of the user and the enterprise.

This book will cover the various aspects of typical hybrid app architecture, from the backend database and web servers all the way to the hybrid app that resides on the employee's device. As the security of the enterprise data is paramount, a large amount of time is focused on ensuring that the communication between the backend and the user's device is secure. The book also covers how to respond to changes in network connectivity and application state.

A code package is also available for this book. Inside, you'll find a complete backend database and web server as well as a frontend hybrid application called Tasker. The entire project is presented as an example that you can learn from and apply to other projects. The link to the code package is given later on in this section.

What This Book Covers

Chapter 1, PhoneGap and Enterprise Mobility, discusses the history of Apache Cordova/Adobe PhoneGap applications, presents reasons why Cordova/PhoneGap is a good fit for the enterprise, and how Cordova facilitates cross-platform development for Cordova and hybrid application architecture.

Chapter 2, Building the Data Store and Business Logic, introduces you to the typical backend server architecture, as well as designing the data models and business logic. You will also get introduced to Tasker, the demonstration app for this book.

Chapter 3, Securing PhoneGap Apps, shows the importance of ensuring the security of enterprise data. This chapter also covers backend security, general security, and issues that relate directly to Cordova/PhoneGap apps.

Chapter 4, Building the Middle Tier, discusses RESTful-like APIs and HATEOAS (Hypertext As The Engine Of Application State). A sample Node.js server is included as a demonstration of these topics, including examples of connecting to databases, executing queries, and generating appropriate responses.

Chapter 5, Communicating Between Mobile and the Middle Tier, demonstrates how to utilize XMLHttpRequest (XHR), SSL Certificate fingerprints, and third-party Cordova/PhoneGap plugins in order to facilitate secure communication between the mobile application and the backend servers.
Chapter 6, *Application Events and Storage*, briefly discusses how to respond to changes in network events and application state. This chapter also introduces a third-party SQLite plugin for persistent data storage and an additional plugin to access the iOS Keychain for secure data storage.

Chapter 7, *Push Notifications*, covers typical Push Notification architecture, and introduces you to Boxcar.io, a service that provides an HTTP API to send pushes. It also introduces a PhoneGap API to respond to these push notifications.

Chapter 8, *Building the Presentation Tier*, covers many of the different patterns used to build a mobile application, including mock-up tools, various libraries and frameworks, common patterns (MVC, data binding, templates, and so on), forms and validation, and data visualization.

Appendix, *Useful Resources*, provides web links to resources that the reader might find useful for further learning. These are split chapterwise so that they can relate to a specific chapter in this book.
Apps that rely on dynamic content often need a way to alert the users of the app that existing content has changed or if new content is available. Although it is possible for an app to repeatedly poll a server for changes or to create a web socket so that changes can be sent in real time from the server, neither option works when the app is in the background. Furthermore, polling is bad for battery life. Push Notifications come to the rescue; these are OS-level services that can display notifications even when our app isn't running, and they can do so in a way that doesn't drain batteries as much as continuous polling. Most platforms also provide some degree of fault tolerance: if the device isn't able to receive a Push Notification, the last notice is usually stored in the cloud until the device is in a situation where it can receive the notice.

In this chapter we'll cover the following topics:

- Push Notification architecture
- Boxcar Universal Push Notifications platform
- Integrate Boxcar into PhoneGap/Cordova
- Send Push via Boxcar
- Receive Pushes from Boxcar
Delving into Push Notification architecture

Push Notifications are an implementation of the publish/subscribe pattern. Clients (mobile devices) can subscribe to a specific channel, and data can be published in these channels from the backend infrastructure. Often, the request is regarding a specific piece of data, but it's also possible to publish more general data, such as a notification that all users of the system need to see.

Typically, Push Notification architectures look something like the following figure:

Push Notifications originate from the backend (although pushes can be generated from clients as well), and these notifications are often routed through a gateway that forwards the notification to the appropriate provider (Apple or Google). The gateway can be a service running on a local server in your data center, or it can be a service provided by a third party. In our case, we'll use a service named Boxcar (http://www.boxcar.io), but there are many other options available, including Parse (http://www.parse.com) and push-notify (https://www.npmjs.org/package/push-notify).
Technically, a gateway isn't required; the Push Notification package mentioned earlier will function just fine if it was included directly within our Node.js server application. The benefit of having a gateway, on the other hand, is that it can take care of the platform-specific implementation of Push Notifications, and anything that needs to send a Push Notification in our app (or even the enterprise) only needs to send a request to the gateway.

It is important to recognize that there is no guarantee that the clients will receive any Push Notifications that are sent. Your app should be able to work properly without push; push should only augment the app's experience.

Boxcar Universal Push Notifications Platform

While it's entirely possible to rely only on internal software for Push Notifications, it does simplify development by utilizing a third-party Push Notification gateway service. We've chosen Boxcar for our demo app not only because it's fast and scalable, but also because it offers an HTTP API that allows us to generate pushes from anything capable of generating an HTTP request (which includes JavaScript, Node.js, and our Oracle database server). Boxcar also provides a PhoneGap SDK that is easy to embed without having to worry about a lot of platform-specific code. Finally, Boxcar also provides a free service tier of up to 200 pushes per minute and a device limit of 100 Android devices and iOS devices. After these limits are reached, the price for the service is based solely on the number of pushes sent per minute. The following steps show how to sign up and begin with your project:

2. Once you’ve got an account, you need to create a new project:

![Project creation interface](image1)

3. At this point, you'll be asked for a project name as well as an optional icon:

![Project creation interface](image2)

After you create a project, you'll be presented with a dashboard that allows you to set up notifications for each platform you want to target. Boxcar supports several platforms, but for our purposes, we'll only worry about iOS and Android.

**Setting up Apple Push Notifications with Boxcar**

In order to set up Apple Push Notifications, click on the **Apple** logo below the **Devices Configuration** heading.
The Apple Push Notification service has the concept of a development and production environment. The certificates and configuration for the development environment are different from the production environment. Therefore, in the following steps, we've highlighted where you'll need to repeat steps when moving your app into production.

Perform the following steps:

1. Under Apple Push Certificates, click on Create Your SSL Certificate? You'll want to use the button in the development row for now, but when you move to production, you'll need to repeat these steps for the production row.

   If you already have a certificate previously generated, you can upload it to Boxcar instead.

2. A new dialog box will appear; click on Download a CSR file to download a certificate request that you will send to Apple.

3. After the CSR is downloaded, click on the Apple link in the dialog box, or navigate to https://developer.apple.com/account/overview.action and click on Identifiers. It's best to do this in a new browser tab.

4. Click on App IDs to display the application identifiers associated with your account.

5. Click on the + icon to add a new app ID.

6. Assign a human-readable name to the ID; we used Tasker PGE CH7 for this chapter.

7. Make sure that Explicit App ID is selected (it should be selected by default).

8. Enter a unique reverse domain application ID in the Bundle ID field; you should use the same ID you specified when creating your Cordova/PhoneGap app. We used com.packtpub.pgech7 for this chapter.

9. Scroll down to App Services and check Data Protection (make sure Complete is marked) and Push Notifications.

10. Click on Continue, and when the next screen appears, click on Submit after you've reviewed everything is correct.

11. Click on App IDs in the sidebar again and click on your new application ID. The row should expand with all the services associated with the app ID; there should be an Edit button near the bottom of this row, click on it.
Push Notifications

12. Under the Push Notifications section, click on **Create Certificate**...; be sure to use the same section as you used in step one (**development/production**).
13. A new dialog box will appear (indicating that you need to create a CSR). Boxcar has already done this for us, so just click on **Continue**.
14. Select **Choose File**... and locate the CSR you downloaded from Boxcar and click on **Choose/OK**.
15. Once the file has been uploaded, click on **Generate** to create the actual certificate.
16. Wait a few seconds. When **Your Certificate is Ready** appears, click on the **Download** button to download the certificate to your computer.
17. Return to Boxcar's browser tab and click on **Step 2, Upload the CER file**.
18. Upload the certificate you downloaded from Apple.
19. You should receive an email indicating that Boxcar attempted to connect to Apple using the certificate provided and that the attempt was successful.
20. Make a note of the **Access Key** and **Secret Key** in this page; you'll need it so that iOS devices can receive Push Notifications.

**Setting up Google Cloud Messages with Boxcar**

In order to target Android, you'll need to click on the **Android** icon on the project's settings page and perform the following steps:

1. In a new tab, open your Google Developer's Console (**https://console.developers.google.com**) and sign in using your Google Developer credentials. (If you don't have an account, you'll need to create one.)
2. Click on the **Create Project** button.
3. In the new dialog, enter the project's name and ID. The ID is not something you can change later, so double check that you've entered it correctly. For our sample, we used **Tasker CH7** as the name and **taskerch7** as the project ID.
4. Read the Google Cloud Platform terms of service and then check the appropriate box to indicate your agreement.
5. Click on **Create**.
6. Next, click on **Enable an API** on the resulting dashboard.
7. Scroll down to **Google Cloud Messaging for Android** and click on the **OFF** button (this will enable the service).
8. A new dialog box will appear. Read the terms of service and check the appropriate box and click on Accept.
9. In the sidebar, click on Credentials. Then, click on Create New Key.
10. Click on Server key in the resulting dialog box.
11. If you want, you can limit what servers can send notifications by entering their IP addresses in the Accept Requests From These Server IP Addresses box. If you don't want to limit who can send notifications, leave this blank.
12. Click on Create.
13. An API KEY will appear; copy this to your clipboard.
14. Return to the Boxcar browser tab.
15. Under the Android Push Credentials heading, paste the key you just copied in the API Key field.
16. Click on Update Client.
17. Make a note of the Access Key and Secret Key from this page; you'll need them so that Android devices can receive Push Notifications.
18. Return to your Google Developer's Console and return to your project's overview page.
19. Take note of the Project ID as you'll need this number later.

Downloading the Boxcar PhoneGap SDK

Now that you've configured Boxcar, you should download the SDK for PhoneGap/Cordova. At the bottom of your Boxcar dashboard, there should be an SDK & Docs link, click on it. A new screen will appear with several SDKs, as shown in the following screenshot. Click on the download button in the PhoneGap section:
Integrating Boxcar with PhoneGap/Cordova

At this point, we can create a PhoneGap/Cordova project and integrate it with Boxcar. We've got a sample project at /ch7 in the code package; all it does is respond to Push Notifications by displaying an alert. You're welcome to use our code as a base, or perform the following steps to integrate Boxcar with your app:

1. Create your project using the Cordova CLI as normal (however, use the app ID you used when setting up Push Notifications):
   ```
   cordova create ./ch7 com.packtpub.pgech7 TaskerCH7
   ```

2. Add the iOS and Android platforms:
   ```
   cordova platform add ios android
   ```

3. Add the necessary plugins; Boxcar requires the following two plugins:
   ```
   cordova plugin add org.apache.cordova.device
   cordova plugin add https://github.com/boxcar/PushPlugin
   ```

4. Copy the `Boxcar.js` file from the Boxcar SDK and place it in your project's www directory.

   We had to make several changes to the `Boxcar.js` file in order to function as needed for our purposes. We've submitted these features and bug fixes to Boxcar, but if they haven't updated their SDK to support these changes, you might want to utilize our custom copy of `Boxcar.js` in the /boxcar folder in the code package for this book.

5. Load the `Boxcar.js` file and the `PushNotification.js` file using script tags in your index.html file:
   ```html
   <script type="text/javascript" src="PushNotification.js"></script>
   <script type="text/javascript" src="Boxcar.js"></script>
   ```

   If you're paying attention to your directory structure, you might wonder where `PushNotification.js` is coming from; it's installed by the `PushPlugin` we added in step three, but it resides in the root directory of your platform-specific www directory. This only occurs when the project is prepared or built, so it's not immediately apparent where this file lives. In your code, you can just assume it's in the root directory where your index.html file lives.
The preceding code works well if you aren't using **RequireJS** or another module loader. If you are, however, you might want to place `Boxcar.js` in a library folder within your directory structure and then configure your module loader appropriately. For example, since our demo uses RequireJS, we placed `Boxcar.js` in `/www/js/app/lib/Boxcar.js` and used the following configuration in `www/js/app.js`:

```javascript
requirejs.config({
    baseUrl: '../js/lib',
    paths: { ...
        'Boxcar': '../app/lib/Boxcar',
        'PushNotifications': '../.../PushNotification'
    },
    shim: { ...
        "PushNotifications": {
            exports: "PushNotification"
        },
        "Boxcar": {
            exports: "Boxcar",
            deps: [ "PushNotifications" ]
        }
    }
});
```

At this point, you can begin to write code that uses Boxcar to listen to Push Notifications. However, before you can do so, you need to initialize Boxcar with your Access Keys and Secret Keys (this must be done after `deviceready` has been fired):

```javascript
APIKeys = {
    ios: {
        clientKey: "Access Key obtained from Boxcar for iOS client",
        secret: "Secret obtained from Boxcar for iOS client"
    },
    android: {
        clientKey: "Access Key obtained from Boxcar for Android",
        secret: "Secret obtained from Boxcar for Android client",
        androidSenderID: "Project ID obtained from Google"
    }
};
Boxcar.init({
    server: "https://boxcar-api.io",
    richUrlBase: "https://boxcar-push.s3.amazonaws.com",
    ios: APIKeys.ios,
    android: APIKeys.android
});
```
Push Notifications

Once `init` has been called, you're ready to start listening to Push Notifications, but you need to perform a couple of configuration steps first for each platform.

**Additional iOS configuration steps**

Before your app will respond to Push Notifications, you need to download the appropriate entitlements from Apple. To do this, follow these steps:

1. Launch Xcode and open the project we just created (the Xcode project will be located under `/platforms/ios` with a `.xcodeproj` extension).
2. Open the Xcode menu and click on Preferences.
3. Click on the Accounts tab.
4. Next, click on your Apple Developer account if it isn't already selected. Then, click on View Details…
5. Click on the refresh icon in the lower-left corner of the dialog box and wait for a few seconds while Xcode downloads your entitlements. When done, you should have a new entitlement for Push Notifications.
6. Now that you've downloaded your entitlements, exit out of the dialog box and then build your project once. Although it won't work yet, doing this now will ensure that you can utilize the Cordova CLI to build and deploy this project in the future rather than using Xcode.

**Additional Android Configuration steps**

According to Boxcar's SDK documentation, you also need to add an entry to the `/platforms/android/res/xml/config.xml` file after adding the plugin using the Cordova CLI. Find the `<widget>` element and add the following code:

```xml
<plugin name="Storage" value="org.apache.cordova.Storage" />
```

**Receiving Pushes from Boxcar**

Although it might seem counterintuitive to work on the code that listens to Push Notifications before we work on the code to send them, it's really not. Boxcar allows you to send any number of pushes from their dashboard, which means that we can initially focus on receiving pushes before we worry about how to send them.
In order to receive pushes, our app needs to register the device with Boxcar. When we register our device, we can specify what kind of Push Notifications we are interested in using **tags**. Tags enable us to target pushes to specific consumers without the need to know the specific device information (such as their UUID). Tags are akin to channels, if that helps.

Our demo app just registers for two specific tags, but when we put everything together in the next chapter, we'll actually listen using a tag based on the authenticated user's username. This will enable our backend to send a Push Notification to a username, and it will reach them on any of the devices they have logged in on.

If the app subscribes to a tag that doesn't exist, Boxcar will create the tag for us. There is a mechanism in the Boxcar console that allows you to create tags ahead of time, if you so choose, but this isn't required.

Along with the tags we're interested in, we also send along some device information in our demo app: the device's unique identifier and an alias. This alias can be anything we want; when we put everything together, it will look like BSMITH iPad 3,1 iOS 7.1, which is easy for humans to read, so even though the alias is optional, it's better to send it. The unique identifier is actually not required by Boxcar. You can send it, if you like, but it is completely optional. It's also important to recognize that this unique identifier might not actually be something that can identify the device, but only be unique to the app. For more information regarding the UUID in Cordova, see https://github.com/apache/cordova-plugin-device/blob/master/doc/index.md#deviceuuid).

Here's what our demo code looks like:

```javascript
Boxcar.registerDevice( {
  mode: "development", // or production
  onsuccess: notificationSuccess.bind( undefined, "Device Registered" ),
  onerror: notificationError,
  onalert: notificationReceived,
  onnotificationclick: notificationClicked,
 .udid: device.uuid, // optional
  alias: [ "BSMITH", device.platform, device.model, device.version ].join(" "), // optional
  tags: [ "_general", "BSMITH" ]
});
```
Push Notifications

When the device is successfully registered, the onsuccess handler will be called. In our case, it's a partially applied function designed to write Device Registered to the console and alert this to the user as well:

```javascript
function notificationSuccess( msgToLog ) {
    console.log( msgToLog );
    alert( msgToLog );
}
```

If an error occurs during registration, the onerror handler is called instead. Again, in our demo, it logs the error to the console and alerts the user, but in a real application, you can take additional steps in an attempt to resolve the problem:

```javascript
function notificationError( err ) {
    console.log( JSON.stringify( err ) );
    alert( JSON.stringify( err ) );
}
```

When a notification is received, the onalert handler will be called. If the user interacted with the notification as displayed by the operating system (while the app wasn't running or was backgrounded), the onnotificationclick handler will also be called.

The data sent to the handler is an object that includes the following properties:

- id: This is the unique identifier of the notification
- time: This is the the time when the notification was received
- sound: This is the sound that should be played when the notification is received
- badge: This is the number of unread messages (matters only for iOS)
- richPush: This indicates whether the Push Notification includes rich content
- url: This is an optional URL for the notification
- seen: This indicates whether the notification has been marked as received
- json: This is the complete notification data; use JSON.parse to convert it to a proper JSON object so that you can access any custom properties (such as a task ID)
Our demo handler doesn't do anything special with the incoming data other than to write it to the console and alert the user, but it does notify Boxcar that the message has been seen. It also resets the badge count (for iOS, this will also remove the notifications from the notification center as well):

```javascript
function notificationReceived( data ) {
    console.log( JSON.stringify( data ) );
    alert( JSON.stringify( data ) );
    // indicate that we've seen the alert
    Boxcar.markAsReceived( {
        onsuccess: notificationSuccess.bind( undefined, "Notification marked as seen." ),
        onerror: notificationError,
        id: data.id
    } );
    Boxcar.resetBadge( {
        onsuccess: notificationSuccess.bind( undefined, "Badge reset." ),
        onerror: notificationError
    } );
}
```

`markAsReceived` should always be called to let Boxcar know that we've seen the notification. We also send a `resetBadge` request so that the badge on iOS is cleared (otherwise it will simply increase until we reset it later). It depends on the nature of your app whether you will want to call `resetBadge` at any other time.

The `onnotificationclick` handler simply alerts the user that the notification was clicked; this only occurs on Android, and although the handler is required, chances are good that you won't worry much about any code in this handler for typical Push Notifications.

Whenever the user is about to log out, you should unregister the device so that the device no longer receives notifications. This can be done, as shown in the following code:

```javascript
Boxcar.unregisterDevice( {
    onsuccess: notificationSuccess.bind( undefined, "Device unregistered" ),
    onerror: notificationError
} );
```

Normally, you wouldn't notify the user of this action, but it is useful to do so during debugging.
Push Notifications

Background vs foreground Push Notifications
When the device receives a Push Notification, it first checks to see whether your app is in the foreground. If it is, the notification is passed directly to the app for display and handling. If the app isn't running or isn't in the foreground, the operating system displays the text of the Push Notification to the user. If the user interacts with the Push Notification, your app is launched (or resumed) and receives the same data so that it can handle the notification appropriately. In our demo app, the notification is redisplayed, but in the final app, we'll use the onnotificationclick handler to automatically navigate to the appropriate task (based on the Push Notification received).

If you need to determine whether the Push Notification was received while the app was in the foreground, you can use the json field of the incoming data object to do so:

```javascript
var notification = JSON.parse(data);
if (notification.foreground) { // our app was in the foreground }
else { // our app wasn't in the foreground }
```

Sending a Push via Boxcar
Sending a Push Notification can be done quite easily from the Boxcar console, but this isn't useful when we want these pushes to be sent automatically. Thankfully, Boxcar utilizes an HTTP API, which means we can generate a push from anything that can generate HTTP requests.

You'll need to obtain the Publisher Access Key and Publisher Secret Key to send a Push Notification programmatically. To do this, follow these steps:

1. On the Dashboard, click on the gear icon for our Boxcar project.
2. Click on Publishers.
3. Click on Tasker API and you should be presented with a screen that contains the necessary keys as well as some example codes you can use. We have modified the Python example and placed it in /ch7/sendPush.
Sending a push boils down to this process:

1. A push object is created, as follows:

   ```json
   push = {
       "taskId": "21",  // custom data
       "f": "0",        // indicates if has rich content 0=no
       "i": "17327",    // id for push
       "badge": "auto", // badge (iOS) "auto" or a number
       "sound": "1.caf", // sound to play (optional)
       "priority": "normal", // priority
       "aps": {
           "alert": "Hello World!" // text to display
       },
       "expires": 1409525432, // optional epoch time when the
       push expires
       "tags": ["_general"] // tag/channel to target
   }
   ```

2. The object is converted to a string.

3. A signature is computed based on the following input:

   ```
   POST
   boxcar-api.io
   /api/push
   <string representation of push object>
   ```

4. The signature is an SHA1 hash using the **Publisher Secret Key** as provided by Boxcar.

5. A URL request is formed like the following code (which includes all the properties from the preceding object):

   ```
   https://boxcar-api.io/api/push/?publishkey=ACCESS_KEY
   &signature=COMPUTED_SIGNATURE&taskId=21&f=0&i=17327&...
   ```

6. The request is sent. A very basic `curl` request is shown:

   ```
   curl 'https://boxcar-api.io/api/push?publishkey=ACCESS_KEY
   &signature=d9cde77daa93b1b372c67c92f0b76bf6b73998d
   c0' -H Content-Type:application/json -d
   '{"id":111,"aps":{"alert":"Hello
   world!"},"expires":1409525432,"tags":[]}'
   ```

Keep your **Publisher Access Key** and **Publisher Secret key** a secret! The only keys that should be present in the code that runs on your user devices should be the iOS and Android client keys and not the publisher keys. If these keys aren't kept secret, anyone can send messages to your app, not just you.
Push Notifications

In our final app, sending this Push Notification will actually occur from the Oracle database server using PL/SQL to send the HTTP request. You can, however, send push notifications from Node.js, Python, or any other platform that supports sending HTTP requests.

Summary

We’ve covered quite a bit in this chapter. We discussed the typical architecture for Push Notifications and gateways, and we covered how to set up Push Notifications, using Boxcar, Apple, and Google Cloud Messaging. We also wrote code to send and receive Push Notifications.

In the next chapter, we will put everything together from the last several chapters to create one proper application that your users might actually want to use. We’ll cover various presentation techniques (charts, templates, and so on) and frameworks, as well as how to wire everything together (models, view controllers, routes, and so on).
Where to buy this book

You can buy PhoneGap for Enterprise 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.