DronaHQ Docs Documentation

Release v6.3

February 22, 2017

Contents

This documentation is organized into the following sections:

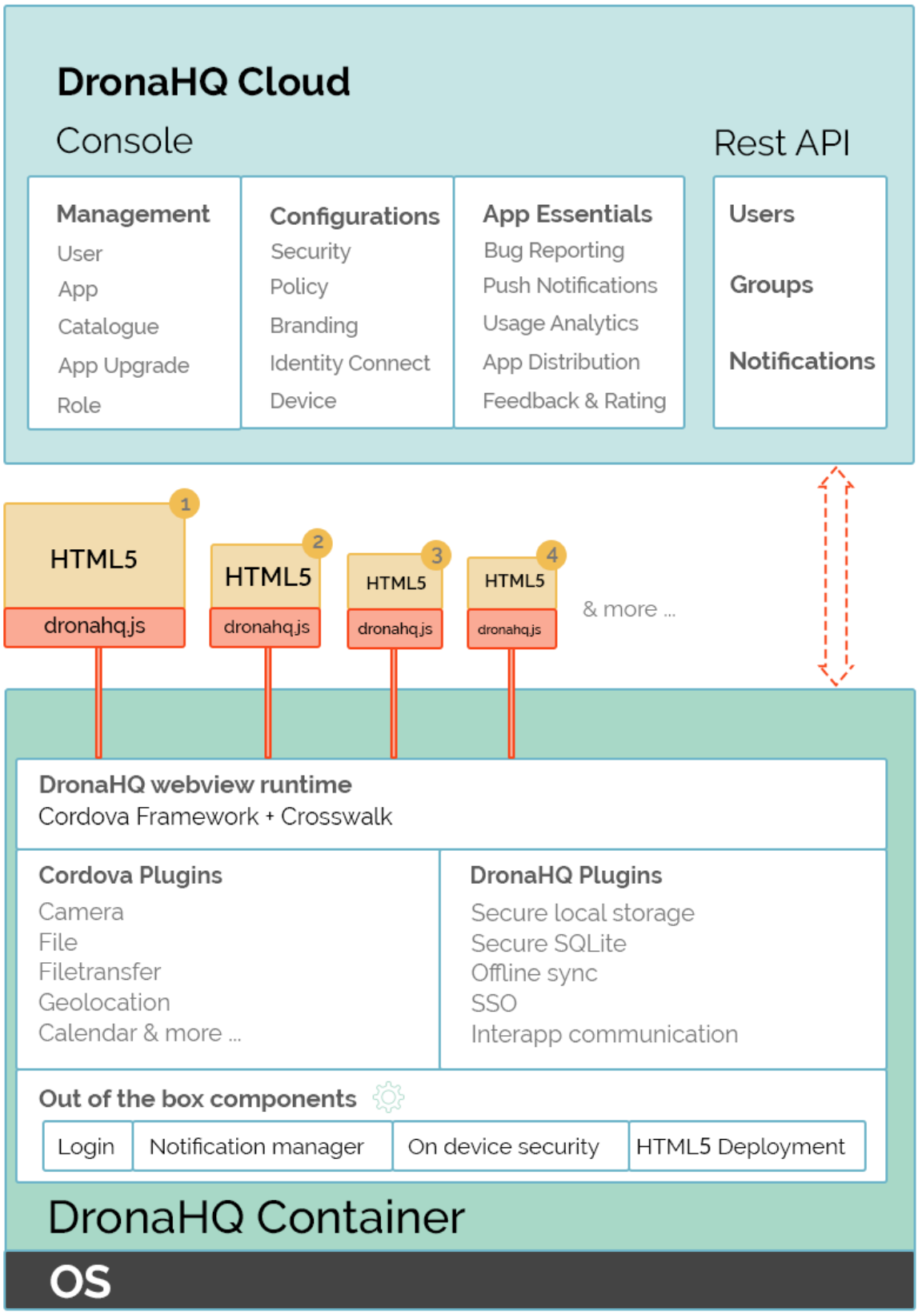
CHAPTER 1

System Architecture

The DronaHQ platform comprises of four major components.

- *The DronaHQ Container App*
- *The DronaHQ Micro-apps*
- *The DronaHQ Admin Console*
- *The DronaHQ SDK*

The following diagram shows a high-level overview of the DronaHQ system architecture.


DronaHQ Cloud
Console
Rest API
Management
User
App
Catalogue
App Upgrade
Role
Configurations
Security
Policy
Branding
Identity Connect
Device
App Essentials
Bug Reporting
Push Notifications
Usage Analytics
App Distribution
Feedback & Rating
Users
Groups
Notifications
1
HTML5
dronahq.js
2
HTML5
dronahq.js
3
HTML5
dronahq.js
4
HTML5
dronahq.js
& more ...
DronaHQ webview runtime
Cordova Framework + Crosswalk
Cordova Plugins
Camera
File
Filetransfer
Geolocation
Calendar & more ...
DronaHQ Plugins
Secure local storage
Secure SQLite
Offline sync
SSO
Interapp communication
Out of the box components
Login
Notification manager
On device security
HTML5 Deployment
DronaHQ Container
OS

The DronaHQ Container App

The DronaHQ Container App is the main app that gets deployed to the user's device. The DronaHQ Container App is supported on iOS, Android, and Windows Phone devices. For users that are on devices other than iOS, Android, and Windows Phone, we also provide a Web platform that could be accessed via any HTML5 compliant browser on the user's device.

The DronaHQ platform comes with a number of pre-built modules, so that you can focus on quickly building a highly functional and great looking micro-apps for your end users.

- **User Management**: The user management module handles the user authentication process via DronaHQ IdP or your own identity provider using SAML 2.0 or OAuth 2.0 protocol. With the user management module, you also get a basic profile screen for users where they can manage their personal information such as a profile image, designation, etc.
- **Micro-app Management**: The DronaHQ platform treats all apps and content deployed in the native container as a micro-app. The micro-app management module helps with the deployment and upgrade of any app/content that is made available to users within the container app. The micro-app manager also has a capability to update any micro-app on a user's device in the background, so that the users are always up-to-date and they have a seamless experience while using their micro-apps.
- **Notification Manager**: The notification manager handles any push notifications sent to the user from the micro-apps. The container app also comes with a pre-built unified inbox of all such notifications where users can easily access all their current and previous notifications.
- **WebView Runtime**: A key component of the DronaHQ container app is a high-performance runtime (based on WebView) to run your micro-apps. This runtime provides similar experience to all your users irrespective of the OS and their versions. The WebView runtime also supports remote debugging of the micro-apps for developers.
- **Security Manager**: The security manager is responsible for adding the layer of security to the container app. The security manager supports all enterprise grade security features such as passcode and fingerprint (touch id) authentication for preventing any unauthorized access. It also provides functionality for remote data wipe in case of device loss. The security manager also provides a secure storage module for the micro-apps, which can be utilized via the DronaHQ SDK.

The DronaHQ Micro-apps

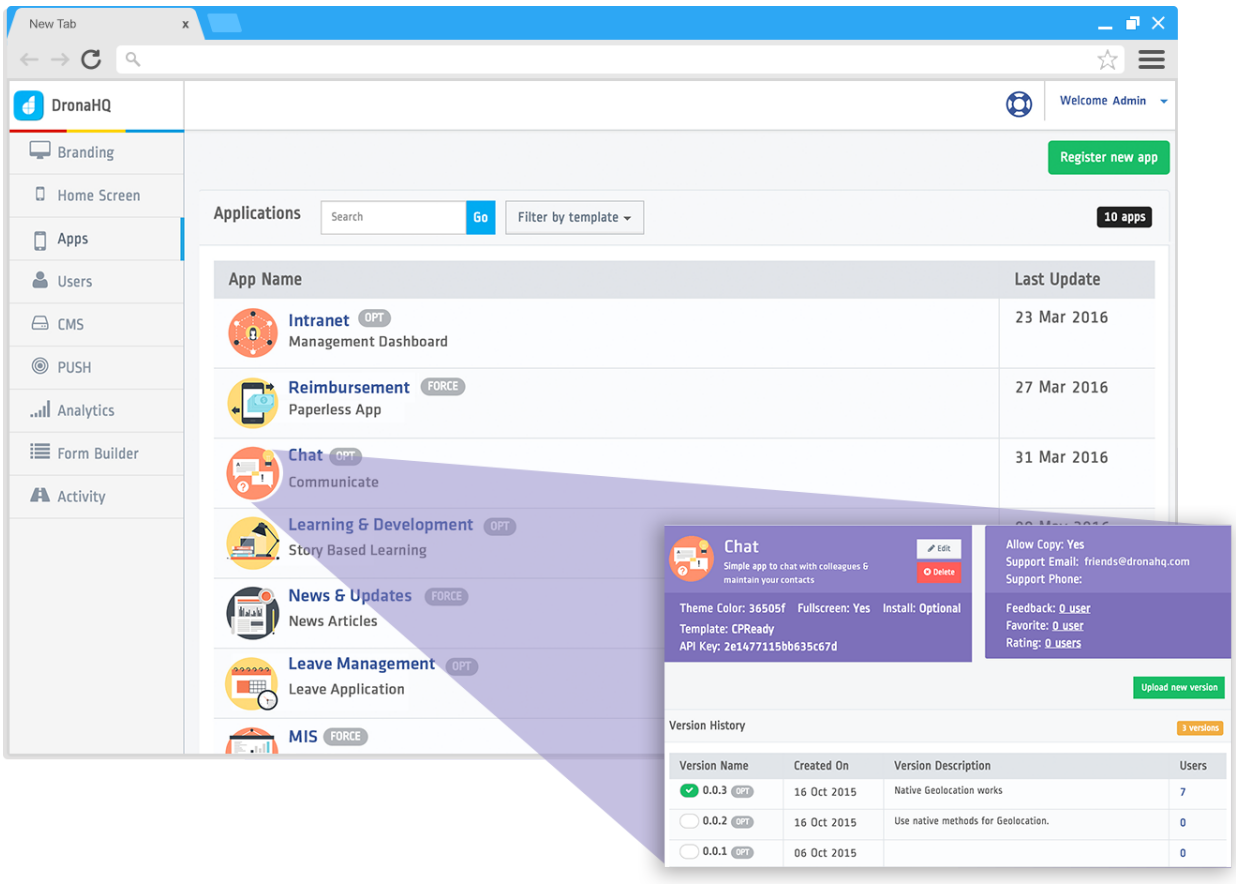
The DronaHQ Micro-apps are the applications that run inside the DronaHQ native containers. Once deployed, the micro-apps are delivered to the container app as a web application or pure content. The micro-apps can be deployed as a zipped package of your application code base, or you could simply host your application on an application server and provide a web URL to the DronaHQ management console. Note that micro-apps that are deployed as zipped packages need a file (index.html) that references all the CSS, JavaScript, image, or any other resources necessary to run the application. The micro-apps are executed in the WebView runtime of the DronaHQ container app.

DronaHQ also supports integration with other native apps already installed on the user's device. Due to technology limitations, only a basic integration with external natives apps is possible as of now.

The DronaHQ Admin Console

The DronaHQ Admin Console is a centralized management console that helps deploy, manage, and analyze micro-apps and content on user's devices. The console provides you with features the ability to:

- Deploy and manage micro-apps
- Create and manage users and groups
- Configure and customize the container app
- Provision and revoke access to micro-apps
- Analyze and review usage statistics



The key sections of the management console include:

- **App Management**: to deploy and manage all your micro-apps. It also provides a per-app usage analysis.
- **User Management**: to manage the end-users. It allows you to invite, activate/deactivate users, and analyze their usage patterns.
- **Catalog Management**: to create and manage your micro-app catalogs, which can then be assigned directly to individual users or a group of users.
- **Settings & Policies**: to help set corporate policies such as screenshot/copy prevention, per-user device limits, etc. The settings area also provides easy-to-use features for customizing home screens, adding logos, and the likes for your custom branding needs.

The DronaHQ SDK

The DronaHQ platform provides a powerful JavaScript SDK and set of REST APIs to help developers build micro-apps faster using HTML5 and associated technologies. The DronaHQ JavaScript SDK is built on top of , a popular mobile framework that allows cross-platform development, and provides access to device native features using JavaScript APIs. So if you are familiar with Apache Cordova, you are already a few steps ahead.

The purpose of DronaHQ SDK is to provide the core building blocks that help developers build applications (aka micro-apps) faster. Developers own the look and feel, the UI interactions, and the business logic of the micro-app, and you can decide to use any HTML5 framework/library for it. The DronaHQ platform takes care of the security, and overall management your container app, micro-apps, and the users.

The DronaHQ SDK has two main components:

- **dronahq.js**: A JavaScript library built on top of Cordova, which combines a core Cordova device APIs and a set of well known Cordova plugins, along with DronaHQ platform specific methods that help you make your app secure and allow you to use native device features.
- **REST APIs**: A set of DronaHQ platform APIs that allows developers to program their micro-apps for operations such as sending notifications, getting a list of users, or other similar functions detailed out in their respective sections.

CHAPTER 2

DronaHQ SDK Overview

The DronaHQ SDK comprises of two major segments. Each segment contains powerful resources to help you build engaging applications on the DronaHQ platform.

- *Device APIs*
- *REST APIs*

Device APIs

The Device APIs portion of the SDK provides a set of APIs that allow micro-apps to interact with the DronaHQ Client Container App as well as the device hardware features such as Camera, File System, etc.

Following Device APIs allow micro-apps to interact with the DronaHQ Client Container App:

- - allows micro-apps to get the profile of the user currently logged into a Client App.
- - allows a micro-apps to access push notifications received by the Client App.
- - allows micro-apps to interact with their own instance.
- - allows micro-apps to submit data in background by leveraging offline data upload capability of the Client App.
- - allows micro-apps to leverage the secure local key-value storage of the Client App as an alternative to the HTML5 local-storage.
- - provides native interface to SQLite Cordova plugin for Android, iOS, and Windows Phone, with APIs similar to HTML5 or .

Following Device APIs allow micro-apps to access device hardware via the Cordova plugins integrated into our Client App:

- - defines a global `navigator.camera` object, which provides an API for taking pictures and for choosing images from the device's image library.
- - implements a File API allowing read/write access to files residing on the device.
- - allows download and upload of files to and from the device.
- - provides a web browser view using `cordova.InAppBrowser.open()` method.

If you need to use a Cordova plugin outside the above list for your micro-apps, please contact our sales team, and we will be happy to assist.

REST APIs

The REST APIs portion of the DronaHQ help micro-apps to interact with the DronaHQ platform for features such as getting user list, sending notifications, etc. DronaHQ makes sure that all **HTTP** request to the REST APIs are authenticated via a **token key** which is either scoped to a channel/account or to a particular micro-app. To generate and learn more about the scope of a **token key**, read our guide.

The REST APIs allow the following actions:

- - can be used by micro-apps to get a list of users for the given **token key**, ordered chronologically.
- - can be used by micro-apps to get the user object for a provided **user id** or **email**.
- - can be used by micro-apps to create users with pre-registered **password** and pre-assigned **groups** based on the scope of a **token key**.
- - can be used by micro-apps to assign a list of group to a user, and also removes a list of group for a user based on the scope of the **token key**.
- - can be used by micro-apps to assign a list of users to a group, and also remove a list of users from a group based on the scope of a **token key**.
- - can be used by micro-apps to activate users based on the scope of the **token key**.
- - can be used by micro-apps to delete users based on the scope of the **token key**.
- - can be used by micro-apps to send notifications to one or more users.
- - can be used by micro-apps to delete notifications on the basis of **notification id** and **content id**.

CHAPTER 3

Quick Start

The purpose of Quick Start is to help you build a very simple HTML5 web application and deploy it inside the DronaHQ Client App container. This helps developers establish a basic foundation for developing and deploying apps on the DronaHQ platform.

As mentioned before, in the guide, then DronaHQ platform's SDK consists of two major segments:

- Device APIs
- REST APIs.

While DronaHQ Rest APIs are to be consumed via HTTP requests, using the Device APIs requires your application to include dronahq.js in the header section of your HTML page.

```
<script src="js/dronahq.js"></script>
```

You can get the latest copy of from our GitHub repository, and place it anywhere in your application's resources.

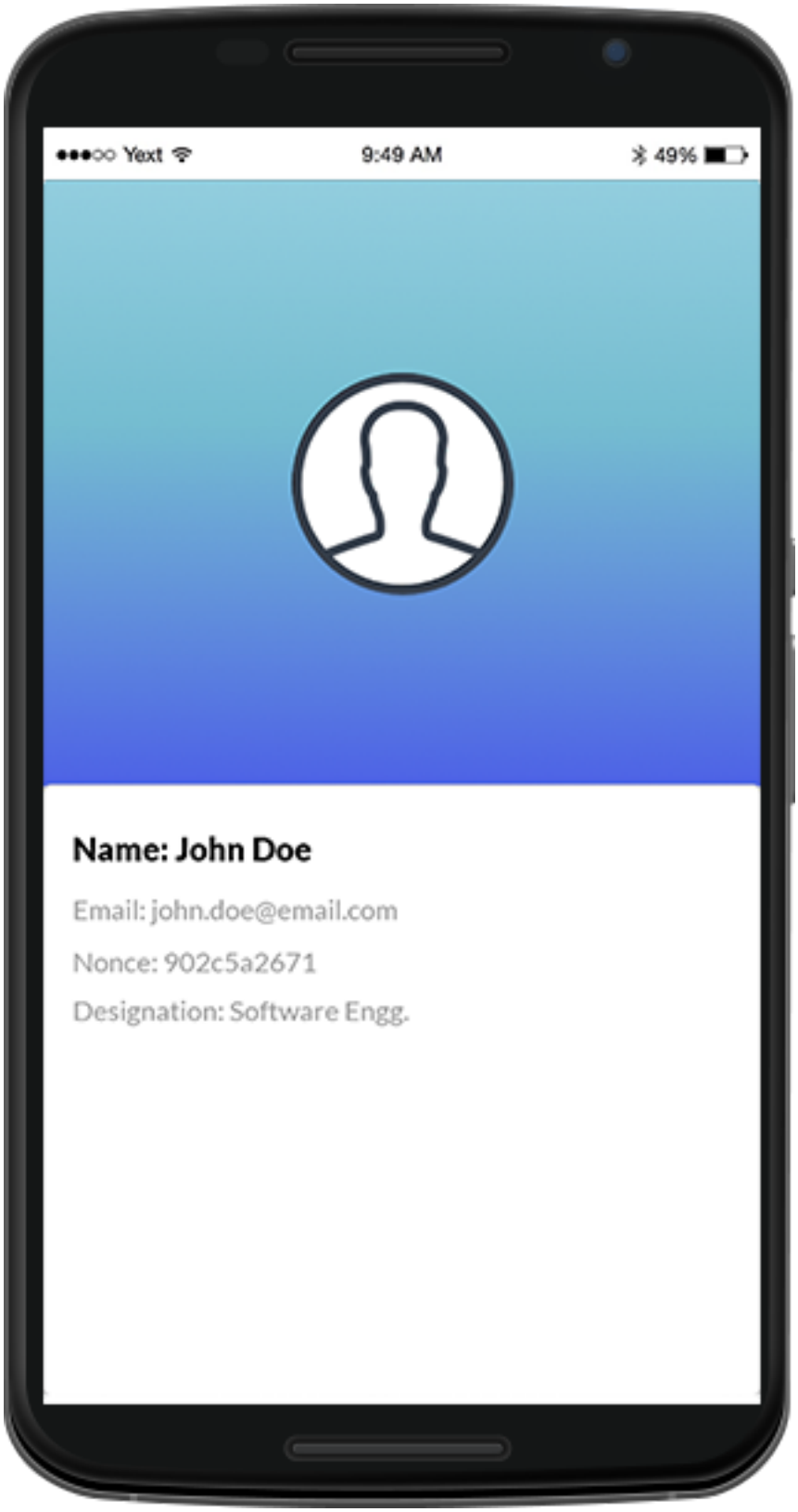
The following properties allow you to detect what platform your micro-app is running on, if you need to add platform specific features/content in your micro-app.

- **DronaHQ.onIos:** returns *true* if the micro-app is running inside DronaHQ container on iOS.
- **DronaHQ.onAndroid:** returns *true* if micro-app is running inside DronaHQ container on Android.
- **DronaHQ.onWindowsPhone:** returns *true* if micro-app is running inside DronaHQ container on Windows Phone.
- **DronaHQ.onWeb:** returns *true* if micro-app is running inside DronaHQ Web App on an HTML5 compliant browser.

Note that the DronaHQ Client must be fully initialized before making any Device API calls.

```
document.addEventListener('deviceready', function(){
        //Intialize your app
}, false);
```

With this in mind, lets start by making a simple application that displays the details of on signed-in user.



Steps to Build

You can follow the steps mentioned below or get the from the github repository.

- Step 1: Create the app project folder "the-user-sso". This will be the application's root directory.
- Step 2: Add index.html, the web page that hosts the application, to the application's root directory.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
                <title>User profile - DronaHQ</title>
                <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scal
                <link rel="stylesheet" href="css/semantic.min.css" type="text/css" />
                <link rel="stylesheet" href="css/app.css" type="text/css" />
        </head>
        <body>
                <div class="ui card">
                        <div id="top">
                                <div class="image">
                                        <img id="imgUserProfile" src="" />
                                </div>
                        </div>
                        <div id="bottom">
                                <div class="ui card centered user-profile">
                                        <div class="content">
                                                <div class="header meta padding-t-10">
                                                        Name: <span id="spUserName" class="">{{u
                                                </div>
                                                <div class="meta padding-t-10">
                                                        Email: <span id="spUserEmail" class="ema
                                                </div>
                                                <div class="meta padding-t-10">
                                                        Nonce: <span id="spUserNonce">{{nonce}}<
                                                </div>
                                                <div id="userDesg" class="meta padding-t-10 hide
                                                        Designation: <span id="spUserDesg">{{use
                                                </div>
                                        </div>
                                </div>
                        </div>
                </div>
        <script src="js/vendor/dronahq.js"></script>
        <script src="js/vendor/jquery.min.js"></script>
        <script src="js/vendor/semantic.min.js"></script>
        <script src="js/vendor/pnglib.js"></script>
        <script src="js/vendor/identicon.js"></script>

        <!-- App -->
        <script src="js/app.js"></script>
        </body>
</html>
```

- Step 3: Add the directories named 'js' and 'css' to the application's root directory. These will contain all your javascript files and style-sheets respectively.
  - Add app.js in the directory 'js'

```
/* global $, DronaHQ, Identicon */

var App = function () {
        var _getDefaultImage = function (inputHash) {
                // set up options
                var hash = 'myUnicodeUsername!' // Any unicode string
```

```
                var options = {
                        background: [255, 255, 255, 255], // rgba white
                        margin: 0.2, // 20% margin
                        size: 290 // 420px square
                }

                // create a base64 encoded PNG
                var data = new Identicon(hash, options).toString()

                return ' ' + data;
        }

        var _initUser = function () {
                DronaHQ.user.getProfile(function (uData) {
                        console.log('User ID: ' + uData.uid)
                        $('#spUserName').text(uData.name)
                        $('#spUserEmail').text(uData.email)
                        if (uData.designation) {
                                $('#userDesg').removeClass('hide')
                                $('#spUserDesg').text(uData.designation)
                        }
                        if (uData.profile_image) {
                                $('#imgUserProfile').attr('src', uData.profile_image)
                        } else {
                                $('#imgUserProfile').attr('src', _getDefaultImage(uData.uid))
                        }
                        $('#spUserNonce').text(uData.nonce)
                })
        }

        return {
                init: function () {
                        _initUser()
                }
        }
}

$(document).on('deviceready', function () {
        var objApp = new App()
        objApp.init()
})
```

- Step 4: Include all files in the application's root directory to a .ZIP file.
- Step 5: as a .ZIP package named "**MyQS**".

Now open the client app, and on the homescreen a micro-app icon named "**MyQS**" would be available. Click the micro-app to view your application.

You can also get started by trying out more of our .

CHAPTER 4

Getting Started

DronaHQ provides a robust set of REST APIs and Device APIs that allow mobile developers to quickly add capabilities such as push notifications, or access to device hardware for apps that run inside the DronaHQ Client Container. DronaHQ's APIs are framework independent and can be used by any web application regardless of HTML5 framework(s) that the application is written with.

With DronaHQ you not only can create new applications, but also integrate any existing HTML5 mobile applications to deploy within the DronaHQ platform.

Whether you are creating a new application on DronaHQ or integrating an existing HTML5 application with DronaHQ , you need to -

- have a registered account.
- add , the DronaHQ javascript SDK, to your application.

WHY A REGISTERED ACCOUNT?

The DronaHQ Administrator Console allows you to create, deploy and manage your application along with many other admin features. So, in order to deploy and manage your applications, you must have a registered DronaHQ account.

A part of the DronaHQ SDK comes in the form of Web APIs. Authentication of these APIs is done via a **token key**. The **token key** can either be of global/account scope or scoped to the limits of a plugin and can be generated only through the DronaHQ Administrator Console.

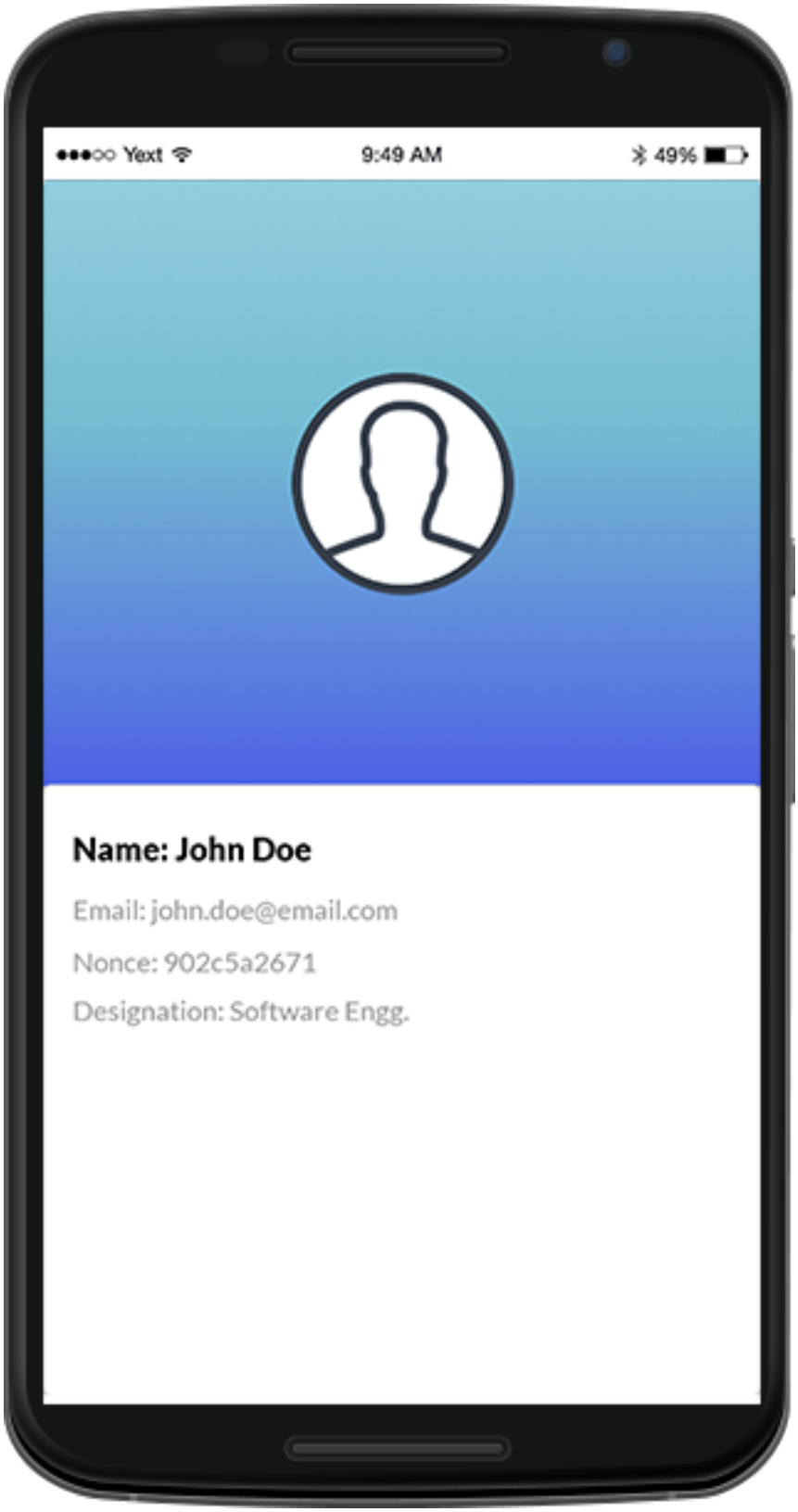
about the scope of your **token key**.

WHY DO I NEED DRONAHQ.JS?

The dronahq.js provides a set of APIs that allow an application to interact exclusively with the DronaHQ Container App and also with the device hardware, which makes its inclusion necessary in your application.

Quick start with a new application

- by creating a new application that displays the detail of the signed-in user -



- You can also try out more of our .

CHAPTER 5

Sample Micro-apps

DronaHQ provides a number of in our GitHub repository to help developers learn the micro-app development for the DronaHQ platform, or even use the sample apps as is if they fit their requirements.

Secure Storage (Key-Value Storage)

This sample app demonstrates how to use DronaHQ key-value storage to store, retrieve & delete data. Check our to learn more about key-value storage.

Link:

Image Capture & Upload

This sample app demonstrates how to use Apache Corodva's camera plugin to capture image from Camera or Gallery. This app also uses file-transfer plugin to upload the chosen image to Cloudinary. Check our to learn more about camera & file-transfer plugins.

Link: https://github.com/dronahq/samples/tree/master/the-picker

Offline Submission

This sample app demonstrates how to use DronaHQ Sync methods to submit data while user is offline.

Link: https://github.com/dronahq/samples/tree/master/the-offline-form

Push Notifications

This sample app demonstrates how to use DronaHQ REST APIs to send push notifications to users. This app also uses DronaHQ Notification methods to retrieve the notification data on the device.

Link: https://github.com/dronahq/samples/tree/master/the-messenger

Single Sign-On

This sample app demonstrates how to use DronaHQ User method to get profile of currently logged-in user. Check our to learn more about DronaHQ APIs.

Link: https://github.com/dronahq/samples/tree/master/the-user-sso

CHAPTER 6

Device API

User

This family of device APIs provided by dronahq.js helps in interacting with the user currently signed-in to Container App.

It has the following method(s) -

- *getprofile()*

getprofile()

```
DronaHQ.user.getProfile(fnSuccess, fnError);
```

This method gives the profile object of the user logged into the app.

USER PROFILE OBJECT -

Parameter	Type	Description
uid	integer	user id of the user currently signed-in to the container app
name	string	Full name of the user currently signed-in to the container app
email	object	Email id of the user currently signed-in to the container app
designation	string	Designation of the user currently signed-in to the container app
profile_image	object	Url of the profile image of the user currently signed-in to the container app
nonce	function	A randrom string

```
var fnSuccess = function(uData){
        console.log('User ID: ' + uData.uid);
        console.log('User Name: ' + uData.name);
        console.log('User Email: ' + uData.email);
        console.log('User Designation: ' + uData.designation);
        console.log('User Profile Image: ' + uData.profile_image);
        console.log('User Nonce: ' + uData.nonce);
};

var fnError = function(err){
        console.error('Failed to load user info. Error: ' + err);
};

DronaHQ.user.getProfile(fnSuccess, fnError);
```

Notification

This family of device APIs provided by dronahq.js can be used to interact with the notification data received by a micro-app.

All notifications are received in the inbox of the container app by default. On clicking a notification the micro-app is invoked and the unique id of the notification is appended as query sting, which can be used to perform actions such as fetching the relevant notification data.

It has the following method(s) -

- *getanotification()*
- *getallnotification()*

getnotification()

This method gets notification based on a notification id.

```
//Get the notification id from query string

function getQParameter(name) {
        name = name.replace(/[\[]/, "\\[").replace(/[\]]/, "\\]");
        var regex = new RegExp("[\\?&]" + name + "=([^&#]*)"),
        results = regex.exec(location.search);
        return results === null ? "" : decodeURIComponent(results[1].replace(/\+/g, " "));
}

var notiId = getQParameter('dm_noti_id');

var fnSuccess = function(notiData){
        //notiData contains the JSON object
        //that was sent using platform notification API
        console.log(notiData);

};
var fnError = function(e){
        if(e.code == '1'){
                console.log('Invaid Notification ID');
        }
};

DronaHQ.notification.getNotification(notid, fnSuccess, fnError);
```

getallnotification()

This method generates a list of all unread notifications. Any notification that is not retrieved by getNotification() is to be considered as unread.

```
var fnSuccess = function(data){
        var unreadNotiCount = data.notifications.length;
};

var fnError = function(e){};

DronaHQ.notification.getAllNotification(fnSuccess, fnError);
```

App

The App API provides an easy way interact with a particular instance of a micro-app.

It has the following method(s) -

- *exitapp()*

exitapp()

Using this method allows you to exist the current instance of the micro-app.

```
DronaHQ.app.exitApp();
```

Sync

This family of device APIs provided by dronahq.js allows a micro-app an easy way to submit data in background via http requests.

It has the following method(s) -

- *upload()*
- *refresh()*
- *getpendinguploadcount()*

It also provide the following event(s) -

- *uploadcomplete*

upload()

This method uploads JSON data to a remote URL. Optionally you can also upload an image file using this request. The JSON data is sent in the request body. By invoking this method, the micro-app will keep trying the data upload until the server returns an http 2XX response. For failed requests, the app will keep trying to upload the data in every 1 hour.

Please note that calling this method only submits the request to the Container App. Actual upload happens when the sync service is triggered the next time. You can call the refresh method to trigger the sync service.

REQUEST PARAMETERS

Parameter	Type	Description
remoteurl	string	Required. http(s) URL of your API where the data should be submitted
request-method	string	Required. HTTP method type for the request. Supported values are 'POST' & 'PUT'
requestdata	object	*Required* A valid JSON object which will be sent in the request body.
image-filepath	string	*Optional* File Path URI for the image which should also get uploaded along with the request.
options	object	*Optional* A object with more request parameters like header & timeout.
fnsuccess	function	*Optional* Callback function which will be called after request has been added to the queue.
fnerror	function	*Optional* Callback function which will be called when request could not be added to the queue.

```
var remUrl = 'https://yourserver.com/api/users/134/action/updatedesignation';
var dronaHQReqMethod = 'POST';
var dronaHQReqData = {designation_code : 'X601'};
var reqImgURI = '';
var headers = [{ Authorization: 'Auth a2FuY2hhbkBkcm9uYW1vYmlsZS5jb206bWFpbEAxMjM0' }];

var options = {};
options.header = headers;

DronaHQ.sync.upload(remURL, dronaHQReqMethod, dronaHQReqData, reqImgURI, options);
```

refresh()

Calling this method will wake up the sync service and triggers any pending upload.

```
//refreshType = 'upload' for submitting any pending uploads

DronaHQ.sync.refresh(refreshType);
```

getpendinguploadcount()

This method gives a count of all the pending upload requests.

```
DronaHQ.sync.getPendingUploadCount(function(count){}, function(e){});
```

uploadcomplete

```
DronaHQ.sync.uploadcomplete
```

This event is triggered whenever the sync service has finished processing all pending requests, even if a few requests have failed to complete successfully. The failed requests are retried next time the service runs.

```
document.addEventListener('dronahq.sync.uploadcomplete', function(){
        //Refresh task is complete.
});
```

Key-value storage

This family of device APIs allow micro-apps to leverage of the Container App's secure local storage API. DronaHQ Container App's secure local storage provides an alternative to the localStorage feature of HTML5. Any value assigned to a key must be of the type string.

Method(s)

- *setItem()*
- *getItem()*
- *removeItem()*
- *clear()*

Set Item

This method allows micro-apps to set a string value to a key.

```
DronaHQ.KVStore.setItem(keyName, keyValue, function() {
        //success callback
}, function() {
        //fail callback
});
```

Get Item

This method allows micro-apps to get the string value of a key.

```
DronaHQ.KVStore.getItem(keyName, function(data) {
        //success callback
        var value = data.value;
}, function() {
        //fail callback
});
```

Remove Item

This method allows micro-apps to remove a key and its associated string value.

```
DronaHQ.KVStore.removeItem(keyName, function() {
        //success callback
}, function() {
        //fail callback
});
```

Clear All

This method allows micro-apps to remove all key with their associated string value.

```
DronaHQ.KVStore.clear(function() {
        //success callback
}, function() {
        //fail callback
});
```

SQlite storage adapter

Interface to sqlite in a dronahq microapp, with API similar to HTML5/.

Status

- SQLite version 3.8.10.2 is supported for all supported platforms Android/iOS/Windows Phone.
- In case of memory issues please use smaller transactions.

Highlights

- Drop-in replacement for HTML5/: the only change should be to replace the static window.openDatabase()factory call with window.sqlitePlugin.openDatabase(), with parameters as documented below.
- Failure-safe nested transactions with batch processing optimizations (according to HTML5/)
- As described in :
  - No 5MB maximum, more information at: http://www.sqlite.org/limits.html

Usage

General

- Drop-in replacement for HTML5/: the only change should be to replace the static window.openDatabase()factory call with window.sqlitePlugin.openDatabase(), with parameters as documented below. Some other known deviations are documented below. Please report if you find any other possible deviations.

NOTE: If a sqlite statement in a transaction fails with an error, the error handler must return false in order to recover the transaction. This is correct according to the HTML5/ standard. This is different from the WebKit implementation of Web SQL in Android and iOS which recovers the transaction if a sql error hander returns a non-true value.

For some basic examples, see the

Opening a database

To open a database access handle object:

var db = window.sqlitePlugin.openDatabase({name: 'my.db'}, successcb, errorcb);

IMPORTANT: Please wait for the 'deviceready' event, as in the following example:

```
// Wait for dronahq to load
document.addEventListener('deviceready', onDeviceReady, false);
// DronaHQ is ready
function onDeviceReady() {
        var db = window.sqlitePlugin.openDatabase({name: 'my.db'});
        // ...
}
```

The successcb and errorcb callback parameters are optional but can be extremely helpful in case anything goes wrong. For example:

```
window.sqlitePlugin.openDatabase({name: 'my.db'}, function(db) {
        db.transaction(function(tx) {
                // ...
        }, function(err) {
                console.log('Open database ERROR: ' + JSON.stringify(err));
        });
});
```

If any sql statements or transactions are attempted on a database object before the openDatabase result is known, they will be queued and will be aborted in case the database cannot be opened. OTHER NOTES:

- It is possible to open multiple database access handle objects for the same database.

- The database handle access object can be closed as described below.

Web SQL replacement tip: To overwrite window.openDatabase:

```
window.openDatabase = function(dbname, ignored1, ignored2, ignored3) {
        return window.sqlitePlugin.openDatabase({name: dbname});
};
```

SQL transactions

The following types of SQL transactions are supported by this version:

- Single-statement transactions
- SQL batch query transactions
- Standard asynchronous transactions

Single-statement transactions

Sample with INSERT:

```
db.executeSql('INSERT INTO MyTable VALUES (?)', ['test-value'], function (resultSet) {
        console.log('resultSet.insertId: ' + resultSet.insertId);
        console.log('resultSet.rowsAffected: ' + resultSet.rowsAffected);
}, function(error) {
        console.log('SELECT error: ' + error.message);
});
```

Sample with SELECT:

```
db.executeSql("SELECT LENGTH('tenletters') AS stringlength", [], function (resultSet) {
        console.log('got stringlength: ' + resultSet.rows.item(0).stringlength);
}, function(error) {
        console.log('SELECT error: ' + error.message);
});
```

NOTE/minor bug: The object returned by resultSet.rows.item(rowNumber) is not immutable. In addition, multiple calls toresultSet.rows.item(rowNumber) with the same rowNumber on the same resultSet object return the same object. For example, the following code will show Second uppertext result: ANOTHER:

```
db.executeSql("SELECT UPPER('First') AS uppertext", [], function (resultSet) {
        var obj1 = resultSet.rows.item(0);
        obj1.uppertext = 'ANOTHER';
        console.log('Second uppertext result: ' + resultSet.rows.item(0).uppertext);
        console.log('SELECT error: ' + error.message);
});
```

SQL batch query transactions

Sample:

```
db.sqlBatch([
        'DROP TABLE IF EXISTS MyTable',
        'CREATE TABLE MyTable (SampleColumn)',
        [ 'INSERT INTO MyTable VALUES (?)', ['test-value'] ],
], function() {
        db.executeSql('SELECT * FROM MyTable', [], function (resultSet) {
                console.log('Sample column value: ' + resultSet.rows.item(0).SampleColumn);
        });
}, function(error) {
```

```
        console.log('Populate table error: ' + error.message);
});
```

In case of an error, all changes in a sql batch are automatically discarded using ROLLBACK.

Standard asynchronous transactions

Standard asynchronous transactions follow the HTML5/ which is very well documented and uses BEGIN and COMMIT or ROLLBACK to keep the transactions failure-safe. Here is a simple example:

```
db.transaction(function(tx) {
        tx.executeSql('DROP TABLE IF EXISTS MyTable');
        tx.executeSql('CREATE TABLE MyTable (SampleColumn)');
        tx.executeSql('INSERT INTO MyTable VALUES (?)', ['test-value'], function(tx, resultSet) {
                console.log('resultSet.insertId: ' + resultSet.insertId);
                console.log('resultSet.rowsAffected: ' + resultSet.rowsAffected);
        }, function(tx, error) {
                console.log('INSERT error: ' + error.message);
        });
}, function(error) {
        console.log('transaction error: ' + error.message);
}, function() {
        console.log('transaction ok');
});
```

In case of a read-only transaction, it is possible to use readTransaction which will not use BEGIN, COMMIT, or ROLLBACK:

```
db.readTransaction(function(tx) {
        tx.executeSql("SELECT UPPER('Some US-ASCII text') AS uppertext", [], function(tx, resultSet)
                console.log("resultSet.rows.item(0).uppertext: " + resultSet.rows.item(0).uppertext);
        }, function(tx, error) {
                console.log('SELECT error: ' + error.message);
        });
}, function(error) {
        console.log('transaction error: ' + error.message);
}, function() {
        console.log('transaction ok');
});
```

WARNING: It is NOT allowed to execute sql statements on a transaction after it has finished. Here is an example from thePopulating Cordova SQLite storage with the JQuery API post at :

```
// BROKEN SAMPLE:
var db = window.sqlitePlugin.openDatabase({name: "test.db"});
db.executeSql("DROP TABLE IF EXISTS tt");
db.executeSql("CREATE TABLE tt (data)");

db.transaction(function(tx) {
        $.ajax({
                url: 'https://api.github.com/users/litehelpers/repos',
                dataType: 'json',
                success: function(res) {
                        console.log('Got AJAX response: ' + JSON.stringify(res));
                        $.each(res, function(i, item) {
                                console.log('REPO NAME: ' + item.name);
                                tx.executeSql("INSERT INTO tt values (?)", JSON.stringify(item.name))
                        });
                }
        });
```

```
}, function(e) {
        console.log('Transaction error: ' + e.message);
}, function() {
        // Check results:
        db.executeSql('SELECT COUNT(*) FROM tt', [], function(res) {
                console.log('Check SELECT result: ' + JSON.stringify(res.rows.item(0)));
        });
});
```

You can find more details and a step-by-step description how to do this right in the Populating Cordova SQLite storage with the JQuery API post at:

NOTE/minor bug: Just like the single-statement transaction described above, the object returned byresultSet.rows.item(rowNumber) is not immutable. In addition, multiple calls to resultSet.rows.item(rowNumber) with the same rowNumber on the same resultSet object return the same object. For example, the following code will show Second uppertext result: ANOTHER:

```
db.readTransaction(function(tx) {
        tx.executeSql("SELECT UPPER('First') AS uppertext", [], function(tx, resultSet) {
                var obj1 = resultSet.rows.item(0);
                obj1.uppertext = 'ANOTHER';
                console.log('Second uppertext result: ' + resultSet.rows.item(0).uppertext);
                console.log('SELECT error: ' + error.message);
        });
});
```

FUTURE TBD: It should be possible to get a row result object using resultSet.rows[rowNumber], also in case of a single-statement transaction. This is non-standard but is supported by the Chrome desktop browser.

Background processing

The threading model depends on which version is used:

- For Android, one background thread per db;
- For iOS, background processing using a very limited thread pool (only one thread working at a time);
- For Windows, no background processing.

Sample with PRAGMA feature

Creates a table, adds a single entry, then queries the count to check if the item was inserted as expected. Note that a new transaction is created in the middle of the first callback.

```
// Wait for DronaHQ to load
document.addEventListener('deviceready', onDeviceReady, false);
// DronaHQ is ready
function onDeviceReady() {
        var db = window.sqlitePlugin.openDatabase({name: 'my.db'});
        db.transaction(function(tx) {
                tx.executeSql('DROP TABLE IF EXISTS test_table');
                tx.executeSql('CREATE TABLE IF NOT EXISTS test_table (id integer primary key, data te
                // demonstrate PRAGMA:
                db.executeSql("pragma table_info (test_table);", [], function(res) {
                        console.log("PRAGMA res: " + JSON.stringify(res));
                });
```

```
                tx.executeSql("INSERT INTO test_table (data, data_num) VALUES (?,?)", ["test", 100],
                        console.log("insertId: " + res.insertId + " -- probably 1");
                        console.log("rowsAffected: " + res.rowsAffected + " -- should be 1");

                        db.transaction(function(tx) {
                                tx.executeSql("select count(id) as cnt from test_table;", [], functio
                                        console.log("res.rows.length: " + res.rows.length + " -- shou
                                        console.log("res.rows.item(0).cnt: " + res.rows.item(0).cnt +
                                });
                        });
                }, function(e) {
                        console.log("ERROR: " + e.message);
                });
        });
}
```

NOTE: PRAGMA statements must be executed in executeSql() on the database object (i.e. db.executeSql()) and NOT within a transaction.

Sample with transaction-level nesting

In this case, the same transaction in the first executeSql() callback is being reused to run executeSql() again.

```
// Wait for DronaHQ to load
document.addEventListener('deviceready', onDeviceReady, false);
// DronaHQ is ready
function onDeviceReady() {
        var db = window.sqlitePlugin.openDatabase({name: 'my.db'});
        db.transaction(function(tx) {
                tx.executeSql('DROP TABLE IF EXISTS test_table');
                tx.executeSql('CREATE TABLE IF NOT EXISTS test_table (id integer primary key, data te
                tx.executeSql("INSERT INTO test_table (data, data_num) VALUES (?,?)", ["test", 100],
                        console.log("insertId: " + res.insertId + " -- probably 1");
                        console.log("rowsAffected: " + res.rowsAffected + " -- should be 1");
                        tx.executeSql("select count(id) as cnt from test_table;", [], function(tx, re
                                console.log("res.rows.length: " + res.rows.length + " -- should be 1"
                                console.log("res.rows.item(0).cnt: " + res.rows.item(0).cnt + " -- sh
                        });
                }, function(tx, e) {
                        console.log("ERROR: " + e.message);
                });
        });
}
```

This case will also works with Safari (WebKit), assuming you replace window.sqlitePlugin.openDatabase withwindow.openDatabase.

Close a database object

This will invalidate all handle access handle objects for the database that is closed:

```
db.close(successcb, errorcb);
```

It is OK to close the database within a transaction callback but NOT within a statement callback. The following example is OK:

```
db.transaction(function(tx) {
        tx.executeSql("SELECT LENGTH('tenletters') AS stringlength", [], function(tx, res) {
                console.log('got stringlength: ' + res.rows.item(0).stringlength);
        });
}, function(error) {
        // OK to close here:
        console.log('transaction error: ' + error.message);
        db.close();
}, function() {
        // OK to close here:
        console.log('transaction ok');
        db.close(function() {
                console.log('database is closed ok');
        });
});
```

The following example is NOT OK:

```
// BROKEN:db.transaction(function(tx) {
        tx.executeSql("SELECT LENGTH('tenletters') AS stringlength", [], function(tx, res) {
                console.log('got stringlength: ' + res.rows.item(0).stringlength);
                // BROKEN - this will trigger the error callback:
                db.close(function() {
                        console.log('database is closed ok');
                }, function(error) {
                console.log('ERROR closing database');
                });
        });
});
```

BUG: It is currently NOT possible to close a database in a db.executeSql callback. For example:

```
// BROKEN DUE TO BUG:db.executeSql("SELECT LENGTH('tenletters') AS stringlength", [], function (res)
        var stringlength = res.rows.item(0).stringlength;
        console.log('got stringlength: ' + res.rows.item(0).stringlength);

        // BROKEN - this will trigger the error callback DUE TO BUG:
        db.close(function() {
                console.log('database is closed ok');
        }, function(error) {
                console.log('ERROR closing database');
        });
});
```

SECOND BUG: When a database connection is closed, any queued transactions are left hanging. All pending transactions should be errored when a database connection is closed. NOTE: As described above, if multiple database access handle objects are opened for the same database and one database handle access object is closed, the database is no longer available for the other database handle objects. Possible workarounds:

- It is still possible to open one or more new database handle objects on a database that has been closed.
- It should be OK not to explicitly close a database handle since database transactions are compliant and the app's memory resources are cleaned up by the system upon termination.

FUTURE TBD: dispose method on the database access handle object, such that a database is closed once all access handle objects are disposed.

Delete a database

```
window.sqlitePlugin.deleteDatabase({name: 'my.db'}, successcb, errorcb);
```

BUG: When a database is deleted, any queued transactions for that database are left hanging. All pending transactions should be errored when a database is deleted.

Database schema versions

The transactional nature of the API makes it relatively straightforward to manage a database schema that may be upgraded over time (adding new columns or new tables, for example). Here is the recommended procedure to follow upon app startup:

- Check your database schema version number (you can use db.executeSql since it should be a very simple query)
- If your database needs to be upgraded, do the following within a single transaction to be failure-safe:
  - Create your database schema version table (single row single column) if it does not exist (you can check thesqlite_master table as described at:)
  - Add any missing columns and tables, and apply any other changes necessary

IMPORTANT: Since we cannot be certain when the users will actually update their apps, old schema versions will have to be supported for a very long time.

Use with Ionic/ngCordova/Angular

It is recommended to follow the tutorial at:

A sample is provided at: litehelpers/Ionic-sqlite-database-example

Documentation at:

Some known deviations from the Web SQL database standard

- The window.sqlitePlugin.openDatabase static factory call takes a different set of parameters than the standard Web SQLwindow.openDatabase static factory call. In case you have to use existing Web SQL code with no modifications please see the Web SQL replacement tip below.
- This plugin does not support the database creation callback or standard database versions. Please read the Database schema versions section below for tips on how to support database schema versioning.
- This plugin does not support the synchronous Web SQL interfaces.
- Error reporting is not 100% compliant, with some issues described below.
- In case of a transaction with an sql statement error for which there is no error handler, the error handler does not returnfalse, or the error handler throws an exception, the plugin will fire more sql statement callbacks before the transaction is aborted with ROLLBACK.
- Known issues with handling of certain ASCII/UNICODE characters as described below.
- This plugin supports some non-standard features as described below.

Known issues

- iOS version does not support certain rapidly repeated open-and-close or open-and-delete test scenarios due to how the implementation handles background processing
- As described below, auto-vacuum is NOT enabled by default.
- Memory issue observed when adding a large number of records due to the JSON implementation.
- A stability issue was reported on the iOS version when in use together with client such as at the same time (see). The workaround is to call sqlite functions and client functions in separate ticks (using setTimeout with 0 timeout).
- If a sql statement fails for which there is no error handler or the error handler does not return false to signal transaction recovery, the plugin fires the remaining sql callbacks before aborting the transaction.
- In case of an error, the error code member is bogus on Android and Windows Phone.
- Possible crash on Android when using Unicode emoji characters due to , which should be fixed in Android 6.x
- Close/delete database bugs described below.
- When a database is opened and deleted without closing, the iOS version is known to leak resources.
- Incorrect or missing insertId/rowsAffected in results for INSERT/UPDATE/DELETE SQL statements with extra semicolon(s) in the beginning for Android in case the androidDatabaseImplementation: 2 (built-in android.database implementation) option is used.
- Within a readTransaction the plugin executes SQL write statements that start with extra semicolon(s).
- Unlike the HTML5/ this plugin handles executeSql calls with too few parameters without error reporting and the iOS version handles executeSql calls with too many parameters without error reporting.

Other limitations

- The db version, display name, and size parameter values are not supported and will be ignored.- (No longer supported by the API)
- Absolute and relative subdirectory path(s) are not tested or supported.
- This plugin will not work before the callback for the 'deviceready' event has been fired, as described in Usage.
- This version will not work within a web worker (not properly supported by the Cordova framework).
- In-memory database db=window.sqlitePlugin.openDatabase({name: ':memory:', ...}) is currently not supported.
- The Android version cannot work with more than 100 open db files (due to the threading model used).
- UNICODE u2028 (line separator) and u2029 (paragraph separator) characters are currently not supported and known to be broken in iOS version due to . There may be a similar issue with certain other UNICODE characters in the iOS version (needs further investigation).
- BLOB type is not supported in this version.
- UNICODE u0000 (same as 0) character not working in Android or Windows Phone
- Case-insensitive matching and other string manipulations on Unicode characters, which is provided by optional ICU integration in the sqlite source and working with recent versions of Android, is not supported for any target platforms.
- iOS version uses a thread pool but with only one thread working at a time due to "synchronized" database access
- Large query result can be slow, also due to JSON implementation
- ATTACH to another database file is not supported by this version.

- UPDATE/DELETE with LIMIT or ORDER BY is not supported.
- WITH clause is not supported by older Android versions in case the androidDatabaseImplementation: 2 (built-in android.database implementation) option is used.
- User-defined savepoints are not supported and not expected to be compatible with the transaction locking mechanism used by this plugin. In addition, the use of BEGIN/COMMIT/ROLLBACK statements is not supported.
- Problems have been reported when using this plugin with Crosswalk (for Android). It may help to install Crosswalk as a plugin instead of using Crosswalk to create the project.
- Does not work with since the window.sqlitePlugin object is not properly exported (ES5 feature). It is recommended to use for SQLite database access with React Native Android/iOS instead.

Some tips and tricks

- If you run into problems and your code follows the asynchronous HTML5/ transaction API, you can try opening a test database using window.openDatabase and see if you get the same problems.
- In case your database schema may change, it is recommended to keep a table with one row and one column to keep track of your own schema version number. It is possible to add it later. The recommended schema update procedure is described below.

Common pitfall(s)

- It is NOT allowed to execute sql statements on a transaction that has already finished, as described below. This is consistent with the HTML5/Web SQL API (http://www.w3.org/TR/webdatabase/).
- The plugin class name starts with "SQL" in capital letters, but in Javascript the sqlitePlugin object name starts with "sql" in small letters.
- Attempting to open a database before receiving the 'deviceready' event callback.
- Inserting STRING into ID field
- Auto-vacuum is NOT enabled by default. It is recommended to periodically VACUUM the database.

Angular/ngCordova/Ionic-related pitfalls

- Angular/ngCordova/Ionic controller/factory/service callbacks may be triggered before the 'deviceready' event is fired

Support

What information is needed for help

Please include the following:

- Which platform(s) Android/iOS/Windows Phone
- Clear description of the issue
- A small, complete, self-contained program that demonstrates the problem, preferably as a Gith

Cordova Plugins

DronaHQ comes with few popular cordova plugins which will allow to you to interact with the native APIs like Camera, Geolocation, File system etc.

Camera

This plugin defines a global `navigator.camera` object, which provides an API for taking pictures and for choosing images from the device's image library.

- **Documentation:**

File

Implements a File API allowing read/write access to files residing on the device.

- **Documentation:**

File Transfer

Allows download and upload of files to and from the device

- **Documentation:**

InAppBrowser

Provides a web browser view using `cordova.InAppBrowser.open()` method.

- **Documentation:**

Ionic Keyboard

The cordova.plugins.Keyboard object provides functions to make interacting with the keyboard easier, and fires events to indicate that the keyboard will hide/show.

- **Documentation:**

Calendar

Cordova plugin to Create, Change, Delete and Find Events in the native Calendar

- **Documentation:**

CHAPTER 7

REST API

Overview

The DronaHQ Plugin REST APIs provide read/write access to the DronaHQ platform, enabling you to integrate DronaHQ platform with your existing applications or building entirely new micro-apps.

All APIs are -based. Responses to the Plugin REST APIs are available in format, and all errors are reported via standard HTTP error codes in addition to JSON formatted error information available in the HTTP response body of the relevant requests.

The DronaHQ REST APIs use an API key for authentication and authorization purposes. DronaHQ provides restricted access to various resources based on the scope of the API key. An API key will have the scope of either:

- Account
- Micro-app

To learn more about DronaHQ REST API authentication, please read the section.

Authentication

DronaHQ authenticates its REST API and restricts its access to various resources based on the scope of the API **tokenkey**. Any **tokenkey** provided by DronaHQ has a scope of either:

- Account
- Micro-app

Account Scope

A **tokenkey** for the REST API that is scoped to an account provides access to all the resources for that account. You can generate a **tokenkey** with the 'account' scope from the page in the DronaHQ Admin Console. This is very useful for testing the APIs, automating any tasks, or integrating your internally developed applications with the DronaHQ platform.

Please note that whenever you generate a new 'account' scoped **tokenkey**, all previous instances of 'account' scoped tokens are automatically invalidated.

Template Scope

A **tokenkey** with 'micro-app' scope is generated while registering a new micro-app in the DronaHQ admin console. All **tokenkeys** scoped to micro-apps allows access to resources constrained to the template where the micro-app resides. For examples: if an app is created in "Sales - France" template, it can only access users, groups, etc. mapped to that template.

The 'micro-app' scoped keys are essential and useful for fine-grained control over the micro-app level authorization. Please note that a 'micro-app' scoped **tokenkey** cannot be regenerated. So if you have to regenerate the **tokenkey** with a micro-app scope, you will need to re-deploy/re-create the app.

Api Resources

DronaHQ REST API provide the following resources that lives at -

```
https://plugin.api.dronahq.com
```

Users

DronaHQ Plugin REST API provide read/write access to DronaHQ user(s) through the following action(s)-

- *Get all users*
- *Get user*
- *Create users*
- *Assign group(s) to a user*
- *Activate Users*
- *Deactivate Users*

Get all users

This API returns a list of users within the of the *tokenkey*, ordered chronologically.

ENDPOINT

```
GET /users
```

REQUEST PARAMETERS

Query string

Parameter	Value Type	Description
tokenkey	string	*Required.* Your API key. Check for more details
ulimit	integer	*Optional.* The number of users to return. Default is 25.
maxuid	integer	*Optional.* If given, users with user id greater than the *maxuid* are returned. To use this argument for paging, you can use the user_id field in the returned user objects.
gid	integer	*Optional.* If given, users who belong to the group with id *gid* are returned.
search	string	*Optional.* Keyword to search for in user's name. Only users matching with the query will be returned.
showstats	boolean	*Optional.* If *true*, a summary of user's stats will be returned.
listtype	string	*Optional.* A enum with 3 possible values. *active*, *inactive*, *moderate*. If *active*, users with activity status as only "active" are returned.

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an .

On success, the HTTP status code in the response header is 200 OK and the response body contains a list of user object.

User object

Parameter	Value Type	Description
user_id	integer	Unique id of the user
user_name	string	Full name of the user
user_email	string	Email-id of the user
user_desg	string	Designation of the user
user_image_url	string	Profile image url of the user
channel_name	string	Unique name of account where user is registered in
app_name	string	Display name of the account where user is registered in
user_reg_date	string	Registration date of the user in 'yyyy-MM-dd hh:mm:ss' format in UTC time-zone
external_idp	boolean	*true* if user account is connected to a third-party identity provider
external_idp_response	string	Response recieved from the third-party identity provider
is_admin	boolean	*true* if the user is an admin
user_group	array	Array of groups which are assigned to the user

Get user

This API returns a user object for the provided userId or Email within the of the *tokenkey.*

ENDPOINT

```
GET /users/{userIdORuserEmail}
```

REQUEST PARAMETERS

Url segment

Parameter	Value Type	Description
userIdORuserEmail	string	*Required.* The "user id" or "email id" of the user whose profile information you which to retrieve

Query string

Parameter	Value Type	Description
tokenkey	string	*Required.* Your API key. Check for more details
nonce	string	*Optional.* Recieved as SSO parameter from the container app via the dronahq.js
stats	boolean	*Optional.* If *true*, a summary of user's CMS activity will be returned in the response object.

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an

On success, the HTTP status code in the response header is 200 OK and the response body contains user object.

User object

Parameter	Value Type	Description
user_id	integer	Unique id of the user
user_name	string	Full name of the user
user_email	string	Email-id of the user
user_desg	string	Designation of the user
user_image_url	string	Profile image url of the user
channel_name	string	Unique name of account where user is registered in
app_name	string	Display name of the account where user is registered in
user_reg_date	string	Registration date of the user in 'yyyy-MM-dd hh:mm:ss' format in UTC time-zone
external_idp	boolean	*true* if user account is connected to a third-party identity provider
external_idp_response	string	Response recieved from the third-party identity provider
is_admin	boolean	*true* if the user is an admin
user_group	array	Array of groups which are assigned to the user

Create users

This API creates a user account with pre-registered password and group(s) based on the of the *tokenkey.*

ENDPOINT

```
POST /users
```

REQUEST PARAMETERS

Request body data

Parameter	Value Type	Description
token_key	string	*Required.* Your API key. Check for more details
pre_register	bool	*Required.* Set value to *true*
invitee_user	array	*Required.* An array of the invitee object. A maximum of 50 users can be sent in one request.

Invitee object data

Parameter	Value Type	Description
user_name	string	*Required.* Full name of the invitee
user_email	string	*Required.* Email-id of the invitee
user_group_name	array	*Required.* A string array of the group-names
user_password	string	*Required.* Password for the invitee

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an

On success, the HTTP status code in the response header is 200 OK and the response body contains an empty array.

However, even when the HTTP status code in the response header is 200 OK, pre-registration of few/all invitee might fail.

In such cases the response body would contain an array of the *error object of an invitee* whose registration failed.

Error object of an invitee

Parameter	Value Type	Description
user_email	string	Email-id of the invitee
error_code	string	Error code
error_detail	string	Detailed description of the error

Possible error codes

Code	Description
2	Email-id is not in correct format.
3	User already exists in the account.
4	The user was registered to atleast one of the mentioned groups but failed for a few.
5	The user could not be registered to any of the mentioned groups. In this case user will not be added to the channel.
6	User license expired. Contact our support desk for more detail.

Assign group(s) to a user

This API can be used to assign a list of groups to a user and can also be used to removes a list of groups assigned to a user within the of its *tokenkey*.

ENDPOINT

```
PUT /users/{userId}/actions/change_group
```

REQUEST PARAMETER

Url segment

Parameter	Value Type	Description
userId	integer	*Required.* The unique id of the user.

Request body data

Parameter	Value Type	Description
token_key	string	*Required.* Your API key. Check for more details
assign	array	*Optional.* An integer array of unique ids of groups to be assigned to the user.
remove	array	*Optional.* An integer array of unique ids of groups to be assigned to the user.

Please note that either assign or remove should contain at least one **group id** in the request body.

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an . On success, the HTTP status code in the response header is 200 OK and the response body contains a JSON object with a list of groups successfully *assigned or removed for* a user.

-Please note that even when the HTTP status code in the response header is 200 OK, *assigning or removing* groups for a user might fail for other reasons such as the group is not a valid group, the API failed to perform the operation. In such cases, the response body would contain an array of the invalid groups in the invalid_groups field and an array of groups for whom the API operation failed in the failed_groups field.

Success response data

Parameter	Value Type	Description
user_id	integer	The unique id of the user.
assigned_to	array	An integer array of unique group ids to which the user was successfully assigned.
removed_from	array	An integer array of unique group ids from where the user was successfully removed.
invalid_groups	array	An integer array of unique group ids that are not valid.
failed_groups	array	An integer array of unique id of groups on which the operation of assigning/removing failed. Retry again with these groups, if problem persists contact our support desk.

Activate Users

This API activates registered users based on the of the *tokenkey.*

ENDPOINT

```
PUT /users
```

REQUEST PARAMETER

Request Body Data

Parameter	Value Type	Description
token_key	string	*Required.* Your API key. Check for more details
list_user_email	array	*Required.* A string array of the users email. A maximum of 50 IDs can be sent in one request.

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an . On success, the HTTP status code in the response header is 200 OK and the response body contains a JSON object.

Success Response Data

Parameter	Value Type	Description
users_success	integer	A string array of user email whose account has been successfully activated.
users_invalid	array	A string array of invalid user email.
user_not_exist	array	A string array of user not found in the API key scope.
users_failed	array	A string array of user email on which the operation of activation failed.

Deactivate Users

This API deactivates users based on the of the *tokenkey.*

ENDPOINT

```
DELETE /users
```

REQUEST PARAMETER

Request Body Data

Parameter	Value Type	Description
token_key	string	*Required.* Your API key. Check for more details
list_user_email	array	*Required.* A string array of the users email. A maximum of 50 IDs can be sent in one request.

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an . On success, the HTTP status code in the response header is 200 OK and the response body contains a JSON object.

Success Response Data

Parameter	Value Type	Description
users_success	integer	A string array of user email whose account has been successfully deactivated.
users_invalid	array	A string array of invalid user email.
user_not_exist	array	A string array of user not found in the API key scope.
users_failed	array	A string array of user email on which the operation of deactivation failed.

Groups

DronaHQ Plugin REST API provide read/write access to DronaHQ group(s) through the following action(s)-

- *Put user(s) to group*

Put user(s) to group

This API can be used to assign a list of users to a group and can also be used to removes a list of users assigned to a group within the of its **tokenkey**.

ENDPOINT

```
POST /groups/{groupId}/actions/assign_user
```

REQUEST PARAMETERS

Url segment

Parameter	Value Type	Description
groupId	integer	*Required.* The unique id of the group in concern

Request body bata

Parameter	Value Type	Description
token_key	string	*Required.* Your API key. Check for more details
assign	array	*Optional.* An integer array of unique ids of users to be assigned to the group.
remove	array	*Optional.* An integer array of unique ids of users to be removed from the group.

Please note that either assign or remove should contain at least one user id.

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an . On success, the HTTP status code in the response header is 200 OK and the response body contains a JSON object with a list of users successfully *assigned to or removed from* a group.

Please note that even when the HTTP status code in the response header is 200 OK, *assigning or removing* users from a group might fail for other reasons such as the user is not a valid user, or the API itself failed to perform the operation. In such cases, the response body would contain an array of the invalid users in the invalid_users field and an array of users for whom the API operation failed in the failed_users field.

Success response data

Parameter	Value Type	Description
group_id	integer	The unique id of the group in concern.
assigned_users	array	An integer array of unique user ids successfully assigned to the group.
removed_users	array	An integer array of unique user ids successfully removed from the group.
invalid_users	array	An integer array of unique user ids that are not valid.
failed_users	array	An integer array of unique id of users on which the operation of assigning/removing failed. Retry again with these users, if problem persists contact our support desk.

Notification

DronaHQ Plugin REST API provide access to the DronaHQ notifications and provide methods to perform the following action(s)-

- *Send Notification*
- *Delete Notification*

Send Notification

This API can be used to send notification to one or more users within the of its *tokenkey*.

ENDPOINT

```
PUT /v2/notifications
```

Request Parameters

Request Body Data

Parameter	Value Type	Description
token_key	string	Your API key. Check for more details.
user_id	array	*Required.* A string array of the DronaHQ user ID or user Email ID. A maximum of 50 IDs can be sent in one request.
message	string	*Required.* Text message to be sent as notification. A maximum of 160 characters are accepted as message.
data	string	*Optional.* A custom JSON string to be sent along with the notification. This data can be retrieved using device API of *dronahq.js*.

RESPONSE FORMAT

On error, the header status code is an error code and the response body contains an .

On success, the HTTP status code in the response header is 200 OK and the response body contains an JSON object.

Success Response Data

Parameter	Value Type	Description
noti_id	integer	Unique ID of the sent notification. This ID can be used to later delete the notification from users inbox.
user_success	array	A string array of user IDs to whom the notification has been sent successfully.
user_failed	array	A string array of invalid user IDs to whom notifications could not be delivered.

Delete Notification

This API can be used to delete a notification on the basis of notification id within the of its *tokenkey*.

ENDPOINT

```
DELETE /notifications/{noti_id}
```

REQUEST PARAMETERS

URL Segment

Parameter	Value Type	Description
noti_id	integer	Unique ID of the notification to be deleted.

Query string

Parameter	Value Type	Description
token_key	string	Your API key. Check for more details.

Response Format

On error, the header status code is an error code and the response body contains an .

On success, the HTTP status code in the response header is 204 No Content.

Pagination

The DronaHQ REST APIs contain methods which return a list of content for any given resource. Such lists often grow very large, so there are limits set in terms of how much of content an application may fetch via a single request. Applications must therefore iterate through the resulting lists using pages in order to be complete and accurate.

Because of the sheer volume of data on the DronaHQ platform, standard paging techniques are sometimes not as effective. Below is an example of how DronaHQ developers can adopt a practice that will ensure efficient and accurate processing a pages or lists.

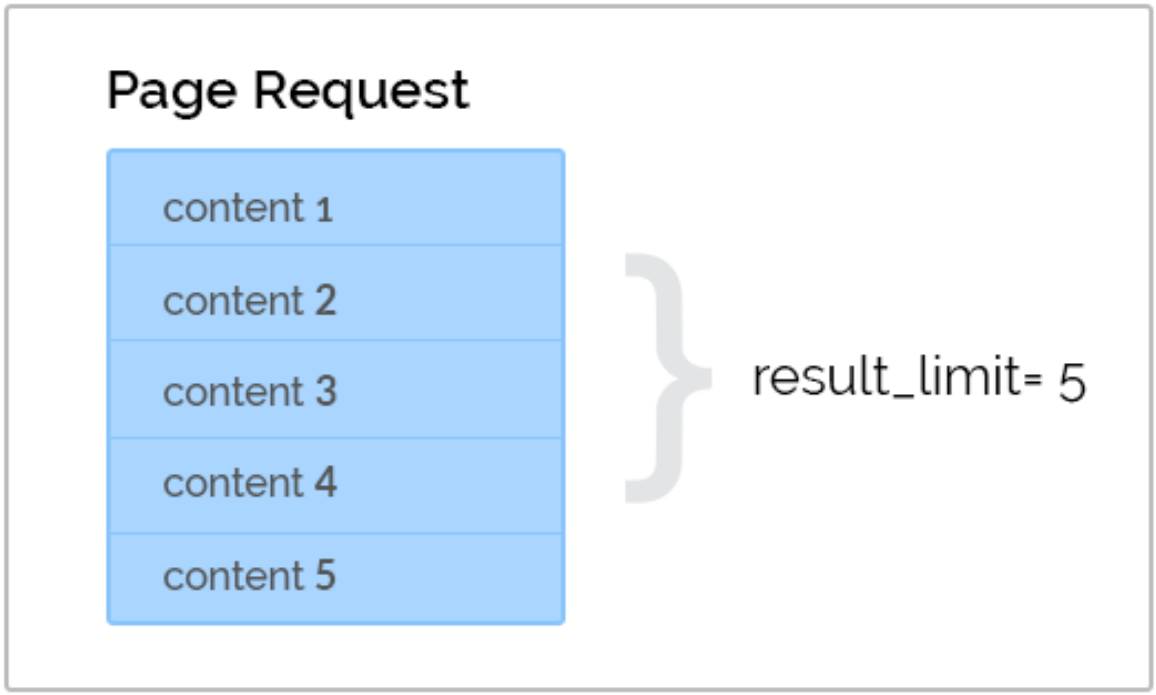
DronaHQ REST API's **list endpoints** help applications to implement pagination through:

- max_id
- result_id

Note that the example below refers to the method in order to demonstrate the pagination concepts.

The 'result_limit' parameter

DronaHQ REST API's **list endpoints** allow an application to set the size of a page in the response through the result_limit parameter.



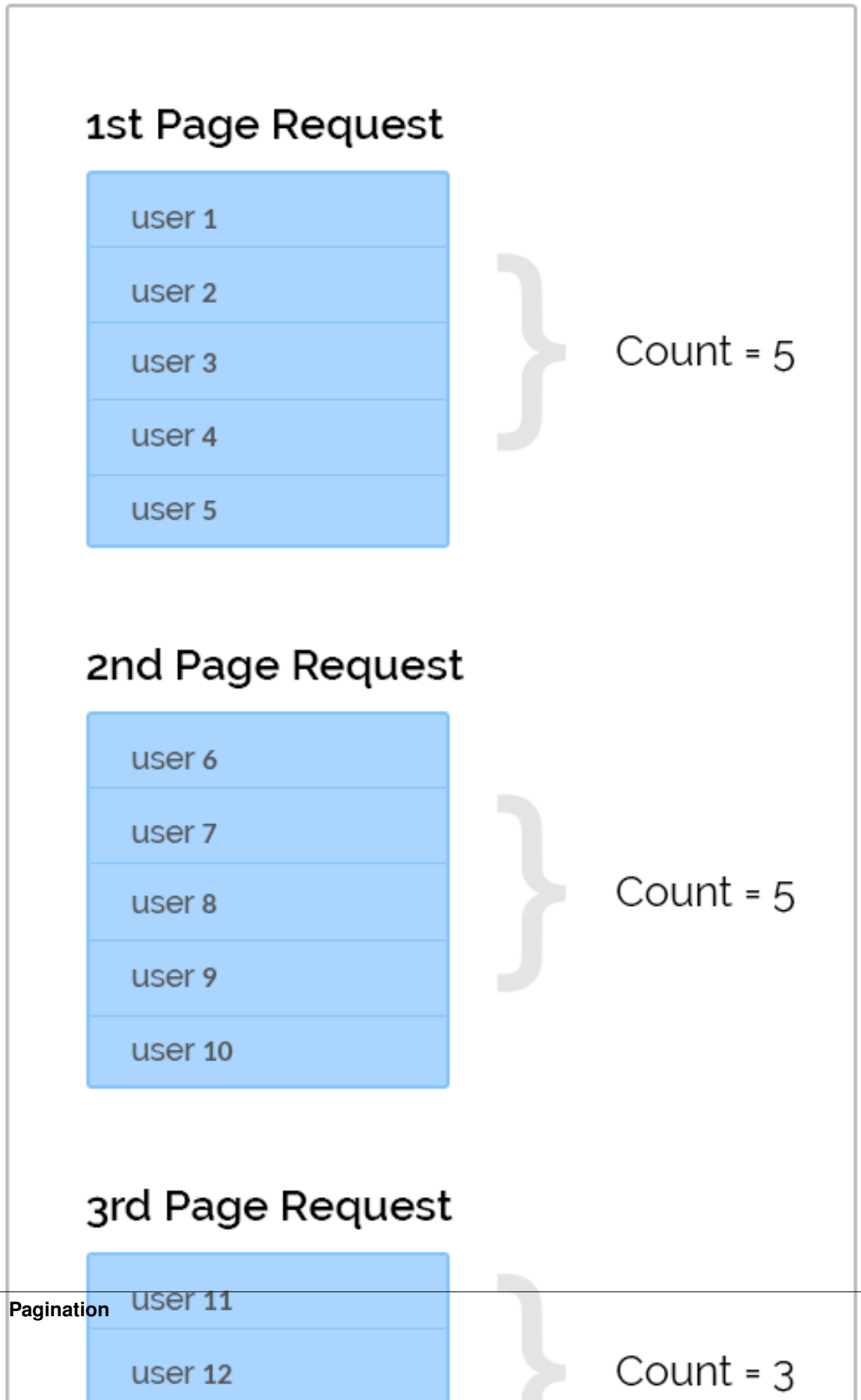
The 'max_id' parameter

All applications must read any list relative to the IDs of the content. This is achieved using the **max_id** request parameter. To use **max_id** correctly, an application's first request to a list endpoint should have the value of **max_id** set as 0 (zero). When processing the request and subsequent responses, the application must keep a track of the highest ID from the contents received. This ID should be passed as the value of the **max_id** parameter for any subsequent requests, which will then return a list of content with IDs higher than the value of the **max_id**.

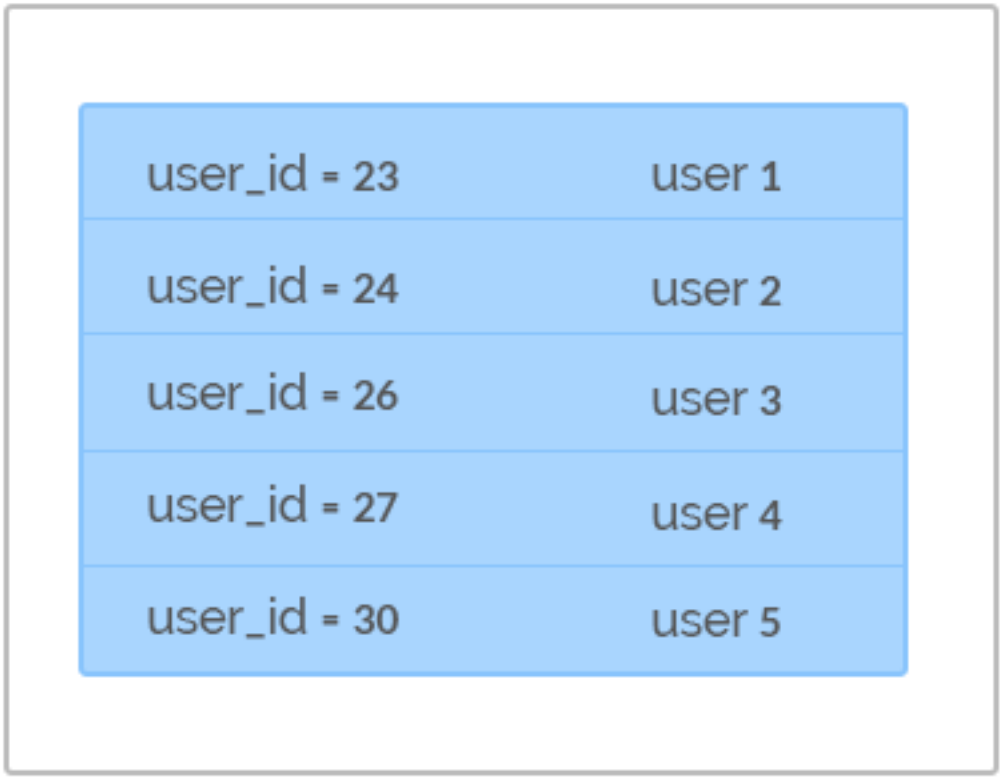
Example

In the API **maxuid** in the query string parameter serves as the **max_id** and ulimit serves as the result_limit of the API.

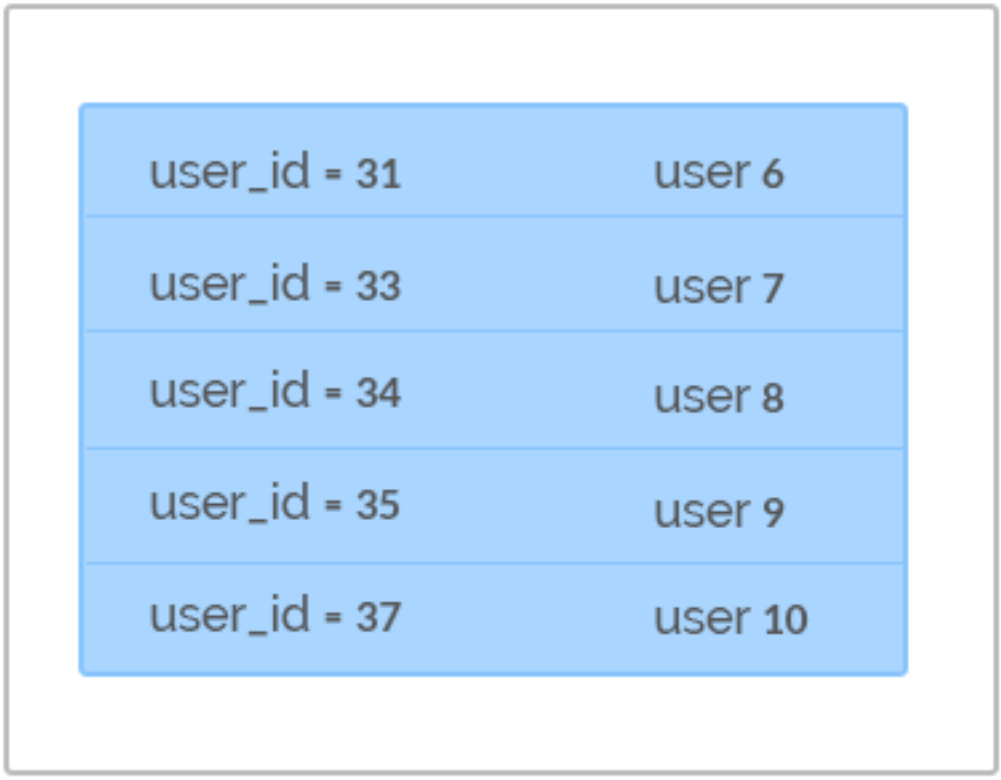
Let's assume an application has set **ulimit = 5**. This implies that on each request the API response would contain a user list with a maximum of 5 users. So for a total user count of 13 the subsequent page response would be as depicted below:


1st Page Request
user 1
user 2
user 3
user 4
user 5
Count = 5
2nd Page Request
user 6
user 7
user 8
user 9
user 10
Count = 5
3rd Page Request
user 11
user 12
Count = 3

Setting **ulimit = 5** the response for the 1st API request would be -



As we can see that the highest ID in the response id 30, the **maxuid** for the second request is set as **maxuid = 30**, which would result in a response like this.



Likewise, the subsequent requests are made by setting the value of **maxuid** to the highest ID of the content already processed.

Error Handling

Response Status Codes

The DronaHQ APIs uses the following HTTP status codes in its response:

Status Code	Description
200	OK - The request has succeeded. The client can read the result of the request in the body.
204	No Content - The request has succeeded but returns no message body.
400	Bad Request - The request could not be understood by the server due to malformed syntax. The message body will contain more information.
401	Unauthorized - The authorization is refused for the resource with the provided API key.
403	Forbidden - The server understood the request, but is refusing to fulfill it.
404	Not Found - The requested resource could not be found. This error can be due to a temporary or permanent condition.
500	Internal Server Error. You should never receive this error because our clever coders catch them all. But if you are unlucky enough to get one, please report it to us through our support desk at .
503	Service Unavailable - The server is currently unable to handle the request due to a temporary condition which will be alleviated after some delay. You can choose to resend the request again.

In case of error response, you can also check the response body to get more information about the error.

Error Response Body

Parameter	Description
error	Error Code
message	Textual description of the error.
reason	Probable cause/reason for the error.

CHAPTER 8

Debugging Apps

DronaHQ provides a specialized instance of DronaHQ Client that enables remote debugging for developers on Android and iOS. DronaHQ Client takes much of the pain out of debugging HTML5 applications across platforms.

We use a custom Chromium-based webview for DronaHQ Client on Android. Any application deployed in DronaHQ on Android 4.0 and above will have the same high-performance, feature-complete web runtime. In addition, developers will have a single version of Chrome to target and be able to debug on any Android 4.x device instead of the highly fragmented environment Android provides today.

Android Debugging

Google has made developing web or hybrid applications on Android difficult for developers. Each version of Android has included a Webkit-based webview that lacked many advanced HTML5 features and had significant performance issues. Using the standard webview, developing a web application for Android involves testing, debugging, and working around Android's many platform incompatibilities and performance limitations. In addition, prior to Android 4.4, the webview did not support remote debugging, making this process particularly challenging.

DronaHQ Client simplifies web application debugging by providing a webview that supports remote debugging on all versions of Android 4.

Because DronaHQ is a security-focused product, we disable debugging for the production DronaHQ Client you can download from the Play Store. Instead, we provide a developer version of the DronaHQ Client for Android that developers can install directly on their Android devices. Fill out this form to request for debuggable client app:

The debug client can be installed on the same device as the version from the Play Store. All data from each version of the app is isolated so it won't impact production data.

Next, if you haven't done any debugging on this device before, follow the instructions to enable remote debugging with Chrome:

Now that you have enabled remote debugging on your device, connect your device to your computer with a USB cable. Open Google Chrome on your computer and navigate to chrome://inspect