8

Integrating With Mobile Applications

In this chapter, we will cover the following recipes:

* Throwing an Alert When the Socket Connects
* Pushing Up Data from the Server
* Responding to Tap Events from the Device
* Doing Server-Side Pagination
* Triggering Hot Deploys

# Introduction

Socket.IO not only works on websites, it works in native Cordova and Phonegap applications, too. In this chapter, we will get Socket.IO working in a Cordova application.

# Throwing an Alert When the Socket Connects

If you have never used Apache Cordova before, it is a platform for building native mobile applications using HTML, CSS and JavaScript. It basically wraps your entire application in a web view and exposes certain native events to the application through JavaScript.

In this recipe, we will emit a message from the server using Socket.IO when the application first opens.

## Getting Ready...

For this recipe, you will need to use a Mac. The reason for this is that Cordova will be creating an app that runs using XCode, which is only available for the Macintosh operating system.

## How To Do It...

To throw an alert in a Cordova app when the socket connects, follow these steps:

1. First we will need to install Cordova. There is a command-line interface that can be installed by running npm install cordova –g in your terminal.
2. Next, we will need to create a new application using the Cordova command line interface that we just installed. That can be run with something like this: cordova create app com.throw.alertThrowAlert. The first argument “app” will be the folder where the project gets built into. The second argument “com.throw.alert” is the reverse domain-style identifier. The third argument “ThrowAlert” is the application title.
3. Now, we will add a platform to our app. Cordova is not limited to running iOS emulations, but it is a good starting point. We can run cordova platform add ios to add the iOS platform as a target for our project.
4. Now that we have an iOS target, we can build to it by runningcordova build ios.
5. Now, what we really want is to be able to emulate an iOS device, and for that we will need an iOS simulator. We can install it by running brew install ios-sim
6. Now that we’ve got Cordova ready to go, we’ll need a server to use Socket.IO with. We can create a server.js file like this and start it for access in our app:

var server = require('http').createServer();

vario = require('socket.io')(server);

io.sockets.on('connection', function (socket) {

console.log('App socket connected');

socket.emit('alert', 'This app is connected to Socket.IO!');

});

server.listen(5000);

1. Now, in the Cordova application that you generated, there will be an index.html file located in the /www directory. That will be the file that Cordova uses when you navigate to your app on a device. We can add the following to the index file to start using Socket.IO in our app:

<script type="text/javascript" src="cordova.js"></script>

<script type="text/javascript" src=" js/socket.io/socket.io.js"></script>

<script type="text/javascript" src="js/index.js"></script>

<script type="text/javascript">

app.initialize();

var socket = io('http://localhost:5000');

document.addEventListener('deviceready', function() {

socket.on('connect', function () {

socket.on('alert', function (message) {

alert(message);

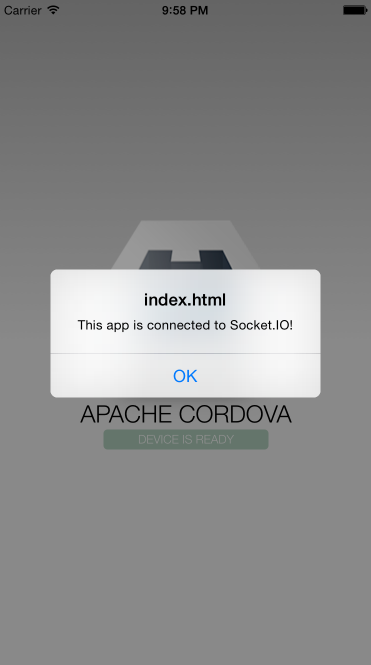
});

});

});

</script>

1. Now, run cordova emulate ios. After the app starts, you should see something like this:



**Insert Image B04893\_08\_01.png**

## How It Works...

Unlike standard web pages, applications wrapped in Cordova are not restricted by a cross-domain origin policy. That means that we are able to load Socket.IO from any domain and will just work. In our example we are starting a server on port 5000, so we simply have to load the script from the same location and call the io() function with our first argument pointing to the server and port we have spun up.

# Pushing Up Data from the Server

Using Socket.IO in a Cordova app is exactly the same as using it on a website. We emit events and listen for events. In the mobile device context, one interesting application is using Socket.IO as an interface over a third-party API. Using Socket.IO to kick off requests to the API, the mobile device doesn’t have to wait for the request to complete; it will simply get a message when that occurs.

In this recipe, we will use Socket.IO as an interface to make request to the openweathermap.org API.

## Getting Ready...

To make request to a third-party API, we will be using the request library. It can be installed by running npm install request in your command line.

## How To Do It...

To create a weather app that interfaces with a third-party API, follow these steps:

1. First, we will need to create a new Cordova application. This can be done by running cordova create app com.push.dataPushData in your command line.
2. Next, we will add the iOS platform by runningcordova platform add ios and build it by running cordova build ios.
3. Now, let’s create our server.js file. This will just need to listen for requests to get the weather and then respond by hitting the API and emitting an event with the freshly fetched weather data:

var server = require('http').createServer(),

io = require('socket.io')(server),

request = require('request');

io.sockets.on('connection', function (socket) {

// The location will only work in Android if we are using the location plugin

socket.on('set-location', function (location) {

request('http://api.openweathermap.org/data/2.5/weather?q=' + location, function (error, response, body) {

if (!error &&response.statusCode == 200) {

socket.emit('weather-change', JSON.parse(body));

}

});

});

});

server.listen(5000);

1. Now, let’s write our app. The index.html file will be fairly simple. We just have a form to request the weather for any location and then a section underneath where the response will eventually be rendered.

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width">

<linkrel="stylesheet" href="http://bootswatch.com/superhero/bootstrap.min.css">

<title>Weather</title>

</head>

<body>

<div class="container">

<h1>Weather</h1>

<form id="submit-weather">

<div class="row">

<div class="col-md-12">

<label for="location">Location:</label>

</div>

<div class="col-md-12">

<input id="location" type="text" name="location" class="form-control" />

</div>

<div class="col-md-12">

<br />

<button type="submit" name="button" class="btnbtn-primary btn-lgbtn-block">Go</button>

</div>

</div>

</form>

<h1 id="location-name"></h1>

<h2 id="weather-main"></h2>

<p id="weather-description"></p>

</div>

<script type="text/javascript" src="cordova.js"></script>

<script type="text/javascript" src="http://localhost:5000/socket.io/socket.io.js"></script>

<script type="text/javascript" src="js/index.js"></script>

</body>

</html>

1. Finally, our client-side index.js file will be responsible for submitting the form value to the server over Socket.IO and listening for an event to render the weather.

var socket = io('http://localhost:5000');

document.getElementById('submit-weather').addEventListener('submit', function (e) {

var location = document.getElementById('location').value;

e.preventDefault();

document.getElementById('location-name').innerHTML = 'Loading...';

socket.emit('set-location', location);

});

socket.on('weather-change', function (data) {

document.getElementById('location-name').innerHTML = data.name + ' ' +

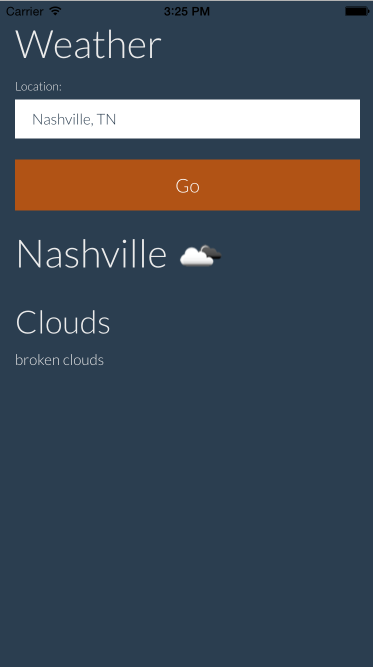
' <imgsrc="http://openweathermap.org/img/w/' + data.weather[0].icon + '.png">';

document.getElementById('weather-main').innerHTML = data.weather[0].main;

document.getElementById('weather-description').innerHTML = data.weather[0].description;

});

1. Now, we can run cordova emulate ios to see our application in action. It will present us with a form to request weather for a location and then render the data as it comes back from the weather API.



**Insert Image B04893\_08\_02.png**

## How It Works...

We are basically using Socket.IO to create a proxy to interface with an API. We can emit a message to the server to start a request to some other server that we are proxying. At that point, the client-side only needs to listen for new data and doesn’t have to concern itself with the implementation details of how the data is retrieved. The server finishes grabbing the data from the API and responds and the client (or multiple clients, if you’d prefer) are able to render the new data.

# Responding to Tap Events from the Device

One of the really cool things about having a central server-side location to emit events from using Socket.IO is that you can use it as an API for multiple applications.

In this recipe, we will emit events to the server every time that a user taps on a button on our app. The server will then emit the tap data to the client-side where it will be displayed on an analytics page.

This technique can be useful for gathering analytics data in real-time and watching the results of A/B testing as they happen.

## How To Do It...

To respond to tap events from the device by updating the counts on our analytics page, follow these steps:

1. First, we will create our new Cordova app by running cordova create app com.client.tapsclientTaps
2. Next, we will add the iOS platform by running cordova platform add ios and build it by running cordova build ios.
3. Now, we can create our server. This will go in server.js and it is only responsible for listening for tap events and emitting the events to any clients that might be listening.

var server = require('http').createServer(),

io = require('socket.io')(server);

io.sockets.on('connection', function (socket) {

socket.on('button-tap', function (btn) {

io.sockets.emit('button-tapped', btn);

});

});

server.listen(5000);

1. Now, we will create a dashboard.html file. This will go outside of our app project because we will just be looking at it in our browser. It will listen for the “button-tapped” event and update the UI when it is received.

<!DOCTYPE html>

<html>

<head>

<linkrel="stylesheet" href="http://bootswatch.com/paper/bootstrap.min.css">

<title>Analytics Dashboard</title>

</head>

<body>

<div class="container">

<h1>Analytics Dashboard</h1>

<div class="row">

<div class="col-md-4">

<h2 class="text-primary">Button 1</h2>

<p><span id="btn-1-clicks">0</span> clicks</p>

</div>

<div class="col-md-4">

<h2 class="text-success">Button 2</h2>

<p><span id="btn-2-clicks">0</span> clicks</p>

</div>

<div class="col-md-4">

<h2 class="text-danger">Button 3</h2>

<p><span id="btn-3-clicks">0</span> clicks</p>

</div>

</div>

</div>

<script type="text/javascript" src="http://localhost:5000/socket.io/socket.io.js"></script>

<script type="text/javascript">

var socket = io('http://localhost:5000');

socket.on('button-tapped', function (i) {

var el = document.getElementById(`btn-${i}-clicks`);

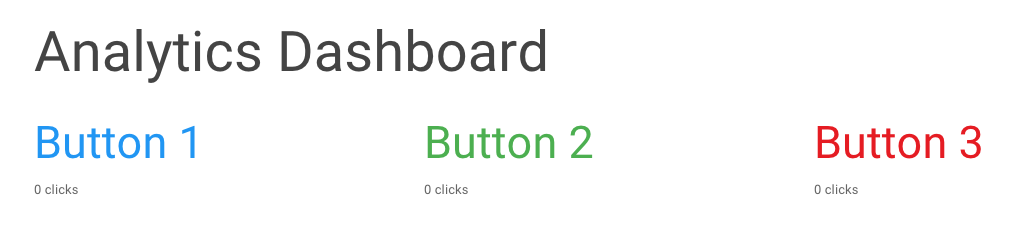
el.innerHTML = parseInt(el.innerHTML) + 1;

});

</script>

</body>

</html>



**Insert Image B04893\_08\_03.png**

1. Now for our index.html file inside our Cordova app. It will just be a list of buttons that can be tapped. When any one of the buttons is tapped, the background color of the app will be changed and we will send the index of the tapped button to the server. The server will emit it and make it visible to our analytics UI.

<!DOCTYPE html>

<html>

<head>

<linkrel="stylesheet" href="http://bootswatch.com/paper/bootstrap.min.css">

<title>Taps</title>

</head>

<body>

<div class="container">

<h1>What is your favorite color?</h1>

<p>

<button data-id="1" class="btnbtn-primary btn-lgbtn-block">Blue</button>

</p>

<p>

<button data-id="2" class="btnbtn-success btn-lgbtn-block">Green</button>

</p>

<p>

<button data-id="3" class="btnbtn-danger btn-lgbtn-block">Red</button>

</p>

<h1 id="location-name"></h1>

<h2 id="weather-main"></h2>

<p id="weather-description"></p>

</div>

<script type="text/javascript" src="cordova.js"></script>

<script type="text/javascript" src="http://localhost:5000/socket.io/socket.io.js"></script>

<script type="text/javascript">

var socket = io('http://localhost:5000');

var buttons = document.querySelectorAll('button');

for (vari = 0; i<buttons.length; i++) {

buttons[i].addEventListener('click', function (e) {

var body = document.querySelector('body'),

index = e.target.getAttribute('data-id'),

backgroundColor;

socket.emit('button-tap', index);

switch (index) {

case '1':

backgroundColor = 'bg-primary';

break;

case '2':

backgroundColor = 'bg-success';

break;

case '3':

backgroundColor = 'bg-danger';

break;

}

body.setAttribute('class', backgroundColor);

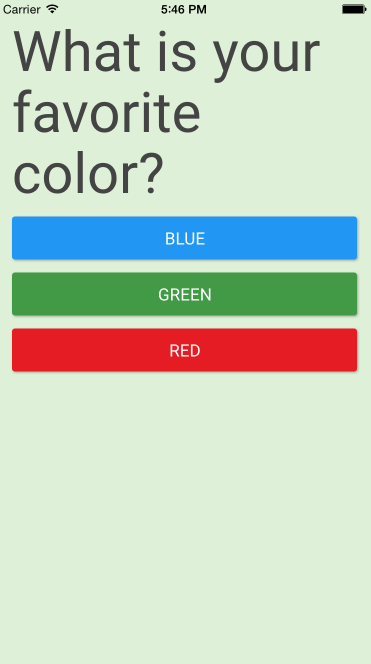
});

}

</script>

</body>

</html>



**Insert Image B04893\_08\_04.png**

1. Now, start your server and open the app using the cordova emulate ios command. The dashboard.html file we created is not being served by our Socket.IO server, so we need to either start a server to serve it up from or open it directly in our file system. Once the app and dashboard are both available, try clicking some buttons on the app. You should see the numbers on the analytics tool increment, as you would expect.

## How It Works...

In this recipe, our server is acting as an intermediary between the mobile app and our analytics tool. When we tap a button on the app, the server is notified and the data is emitted so that the analytics dashboard can see it.

We are not persisting the data. If you refresh your dashboard page, the counts will all be lost. In a production scenario, we would probably be writing the count data to a database so that we could access it beyond a page refresh.

# Doing Server-Side Pagination

When you have a large quantity of data, it is often beneficial to present it in paged format so that you don’t have to load all of it at once. In this recipe, we will create an app with several pages of data that will require pagination emitted from the server-side.

## Getting Ready...

For this recipe, we will be using a library called “Chance” to generate a large set of random data. Chance is composed of various functions that allow us to get all sorts of random data to test with, so it is ideal for mocking up data before you have any. It can be installed by running npm install chance in your command line.

## How To Do It...

To do server-side pagination using Socket.IO, follow these steps:

1. First, we will create our new Cordova app by running cordova create app com.server.paginationserverPagination
2. Next, we will add the iOS platform by running cordova platform add ios and build it by running cordova build ios.
3. Now, we will create our server.js file. We will generate a couple hundred random items when the server starts and selectively provide a subset of the data when the client-side app emits a page change event. We will also emit some information about the pages so that the client can tell how many records are being displayed per page, which page we are currently on and how many total pages there are.

var server = require('http').createServer(),

io = require('socket.io')(server),

Chance = require('chance'),

chance = new Chance(),

cats = [];

// Generate a random cat with chance

functionrandomCat () {

return {

name: chance.name({ prefix: true }),

age: chance.age(),

twitter: chance.twitter(),

email: chance.email(),

website: chance.url(),

image: 'http://lorempixel.com/50/50/cats/' + chance.integer({min: 1, max: 10})

};

}

// Generate a bunch of random cats

for (vari = 0; i< 200; i++) {

cats.push(randomCat());

}

io.sockets.on('connection', function (socket) {

socket.on('get-page', function (data) {

varcatsOnPage = [],

startAt = data.page \* data.per,

endAt = startAt + data.per;

// If there are not enough cats to show in one page

if (cats.length<endAt) {

endAt = cats.length;

}

for (vari = startAt; i<endAt; i++) {

catsOnPage.push(cats[i]);

}

io.sockets.emit('render-page', {

cats: catsOnPage,

pages: {

per: data.per,

page: data.page,

last: parseInt(cats.length / data.per, 10)

}

});

});

});

server.listen(5000);

1. The index.html file inside the app we created will have a few template elements to render our UI into once it comes available.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="format-detection" content="telephone=no" />

<meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-scale=1, minimum-scale=1, width=device-width, height=device-height, target-densitydpi=device-dpi" />

<meta name="msapplication-tap-highlight" content="no" />

<linkrel="stylesheet" href="http://bootswatch.com/slate/bootstrap.min.css" />

<title>My Cats</title>

</head>

<body>

<div class="container">

<h1>My Cats</h1>

<nav>

<ul class="pagination"></ul>

</nav>

<div id="cats-list"></div>

</div>

<script type="text/javascript" src="cordova.js"></script>

<script type="text/javascript" src="http://localhost:5000/socket.io/socket.io.js"></script>

<script type="text/javascript" src="js/index.js"></script>

</body>

</html>

1. Our js/index.js file will be responsible for emitting events to the server every time a page change is requested or on the initial app load. It will also render the pages based on the data that will be emitted from the server.

var socket = io('http://localhost:5000'),

list = document.getElementById('cats-list'),

pagination = document.querySelector('.pagination');

functionrenderPage (cats) {

list.innerHTML = '';

cats.forEach(function (cat) {

varcatElement = document.createElement('div');

catElement.setAttribute('class', 'panel panel-default');

catElement.innerHTML = '<div class="panel-heading">' +

'<h3 class="panel-title"><imgsrc="' + cat.image + '" />&nbsp; ' + cat.name + '</h3>' +

'</div>' +

'<div class="panel-body">' +

'<p><strong>Age:</strong> ' + cat.age + '</p>' +

'<p><strong>Email:</strong> ' + cat.email + '</p>' +

'<p><strong>Twitter:</strong> ' + cat.twitter + '</p>' +

'<p><strong>Website:</strong> ' + cat.website + '</p>' +

'</div>';

list.appendChild(catElement);

});

}

functiongoToPage (page) {

socket.emit('get-page', {

page: page,

per: 25

});

}

functionpageClick (e) {

e.preventDefault();

goToPage(parseInt(e.target.innerHTML, 10));

}

functionrenderPageNumber (i, active) {

var li = document.createElement('li'),

a = document.createElement('a');

if (active) {

li.setAttribute('class', 'active');

}

a.innerHTML = i;

a.setAttribute('href', '#');

a.addEventListener('click', pageClick);

li.appendChild(a);

pagination.appendChild(li);

}

functionrenderPagination (pages) {

pagination.innerHTML = '', pageElements = [];

for (vari = 1; i<pages.last; i++) {

renderPageNumber(i, i === pages.page);

}

}

socket.on('render-page', function (data) {

renderPage(data.cats);

renderPagination(data.pages);

});

document.addEventListener('deviceready', function () {

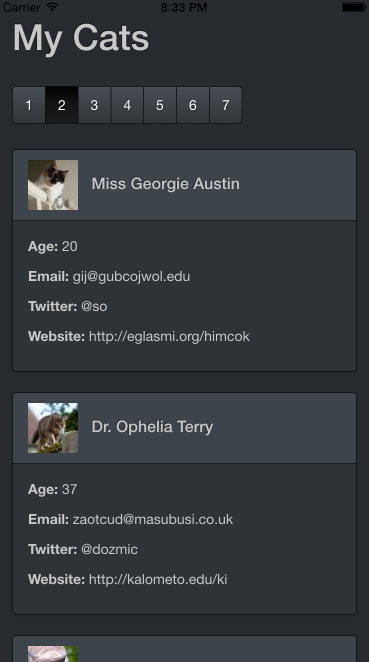
goToPage(1);

});

1. Now we can start our Node server and then run cordova emulate ios in our terminal to see our app.

## How It Works...

When our app boots up, it emits an event to the server asking for the data for the first page. The server computes the data that will go on the first page as well as some metadata to render the page numbers. All of the data is emitted back to the original socket and that socket is able to draw the pages and pagination UI based on the information it receives.



**Insert Image B04893\_08\_05.png**

# Triggering Hot Deploys

One of the more frustrating things about developing mobile apps is that when you find a bug or have a new feature to implement, getting your new app in the hands of your users is not as simple as just deploying it like you would in the web world. Both iOS and Android apps must undergo a review process, which means you could potentially be waiting up to a week for your update to go live.

Fortunately, if we are using JavaScript to write our applications, we emit changes with Socket.IO whenever our code is updated. We can refresh the app in real-time to reflect our new code.

## How To Do It...

To trigger a hot deploy in Cordova using Socket.IO, follow these steps:

1. First, we will create our Cordova app by running cordova create app com.hot.deployHotDeploy
2. We will create a server.js file that will live outside of our Cordova app. This will be responsible for emitting events whenever the myFile.js file changes.

var express = require('express'),

app = express(),

http = require('http'),

fs = require('fs'),

path = require('path'), io, server;

app.use(express.static(\_\_dirname));

server = http.Server(app);

server.listen(5000);

io = require('socket.io')(server);

varfilePath = path.resolve(\_\_dirname, './myFile.js');

fs.watchFile(filePath, function () {

io.sockets.emit('code-change');

});

1. In our Cordova app, we will point to the myFile.js script in our index.html file:

<h1 id="container">

Loading

</h1>

<script type="text/javascript" src="cordova.js"></script>

<script type="text/javascript" src="http://localhost:5000/socket.io/socket.io.js"></script>

<script type="text/javascript" src="js/index.js"></script>

<script type="text/javascript" src="http://localhost:5000/myFile.js"></script>

1. Finally, our myFile.js file will go outside of our Cordova app with our server. We will start it out as an incremental counter, but we will be able to re-save this file and get new results when it hot-reloads the app:

app.initialize();

var socket = io('http://localhost:5000');

socket.on('code-change', function () {

window.location = window.location;

});

vari = 0;

setInterval(function () {

// Change the thing text set in the container to see the app refresh

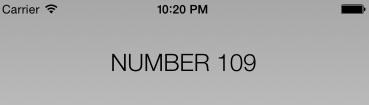
// and display the new code

document.getElementById('container').innerHTML = '#' + i;

i++;

}, 500);

1. Now, open your app by running cordovaemulateios. You will see a counter which increments over time. If you change the code in myFile.js, the app will refresh and the new code will run instead of the old code.



**Insert Image B04893\_08\_06.png**

## How It Works...

In our application, we see an initial counter which increments over time. However, we have a file watcher in our Node application that will trigger a Socket.IO event whenever the file changes. The client-side is listening for the change event to fire and is reacting by reloading the entire page.

Reloading the entire app is not always ideal. Using technology like the WebPack package bundler, we could actually be more selective about the code that we refresh and append code changes a-la-carte instead of blowing away everything each time the app updates.