IBM **MobileFirst**

MobileFirst & Bluemix Apps that Work as Well Offline as they do Online

This file and the accompanying video comes from https://github.com/IBM-Bluemix/ MobileFirst-Offline-Apps/. Be sure to check back at that location for the most up to date information and latest version of this script and related assets.

In the dynamic and ever-changing realm of mobile, context is critical to the success of your applications. Users may be at home sitting on the couch, or they could be on top of a mountain with very limited connectivity. There's no way to predict where someone will be when they're using your app, and as many of us painfully know already, there is never a case when you are always online on your mobile devices.

Well, this doesn't always have to be a problem. Regardless of whether your app is online or offline, it is important that your app does what it needs to do – solve a problem and provide value.

This script will walk you through the process of setting up a MobileFirst app for Bluemix that can handle both online and offline scenarios. Make sure that you have downloaded the code assets for this script.
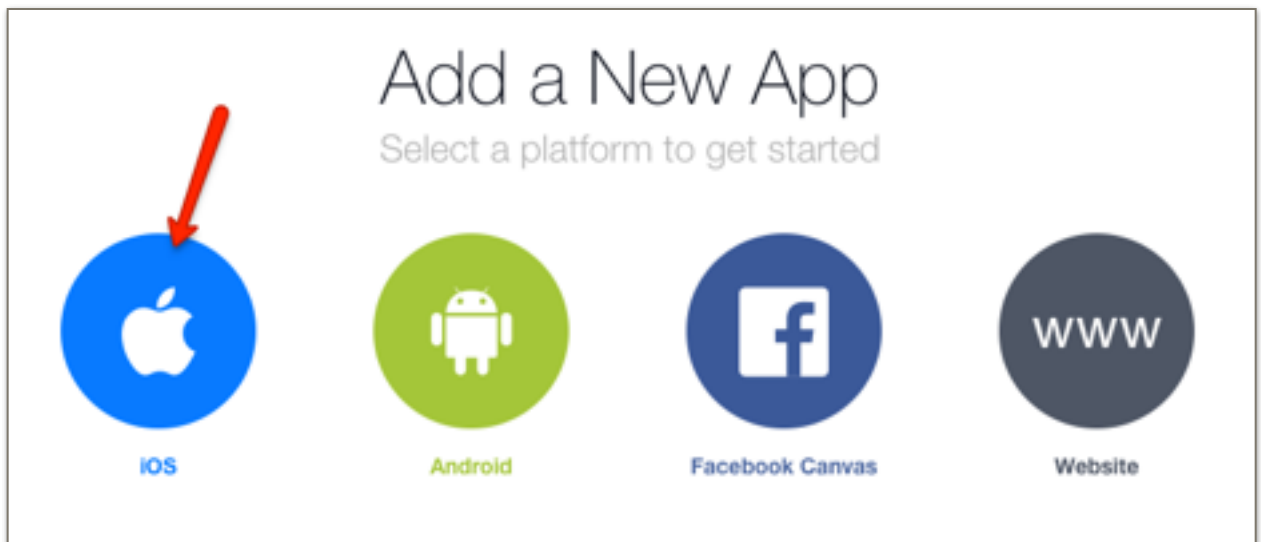
Setup a new Facebook App ID

We are going to use Facebook for authentication within the application, so our first step is to configure a Facebook App authentication at https://developers.facebook.com/

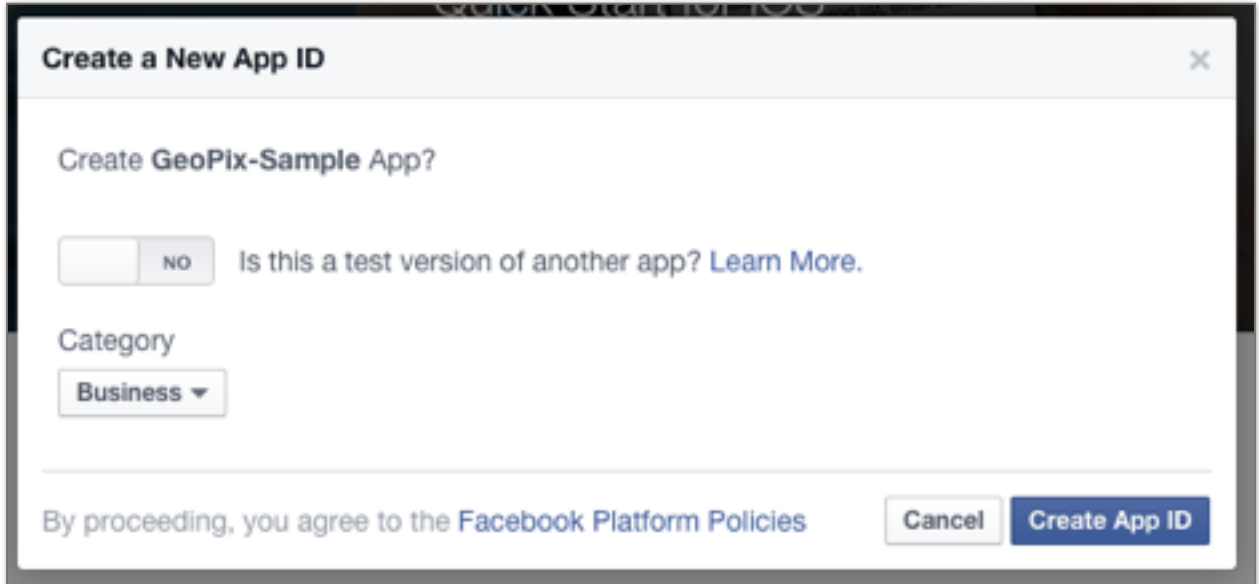1. Go to "My Apps" at: https://developers.facebook.com/apps/

2. Next click the green "Add a New App" button
3. Next, select the "iOS" option

4. Enter the name of your application (this is the name as it will be displayed on Facebook, not necessarily the native App's name in a production environment). I used the value "GeoPix-Sample" Then click the "Create New Facebook App ID" button.

5. Next select a category and press the "Create App ID" button. You can also select whether this App ID is a test/debug version of another app. For the basic demo, this is not required.
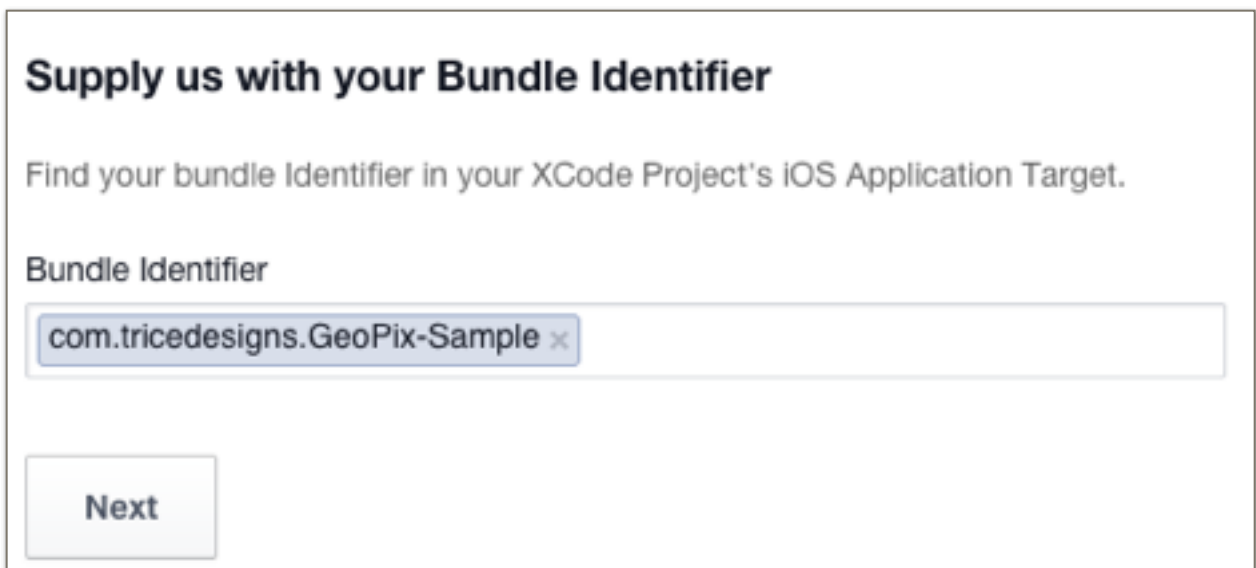
6. At this point Facebook will present you with the Quick Start guide for setting up a new iOS app. You can reference this later if you'd like to learn more. However, it is not necessary, as the sample project has already been configured. Make note of the values for "AppID and App Name - you are going to need them.
7. Scroll down to enter the bundle identifier for your app. This will be the unique id for the native iOS application. **This must be a unique value.** For mine, I will use *com.tricedesigns.GeoPix-Sample*, but yours should be different. Click "Next" to proceed. You can always come back here and add/remove bundle identifiers later.
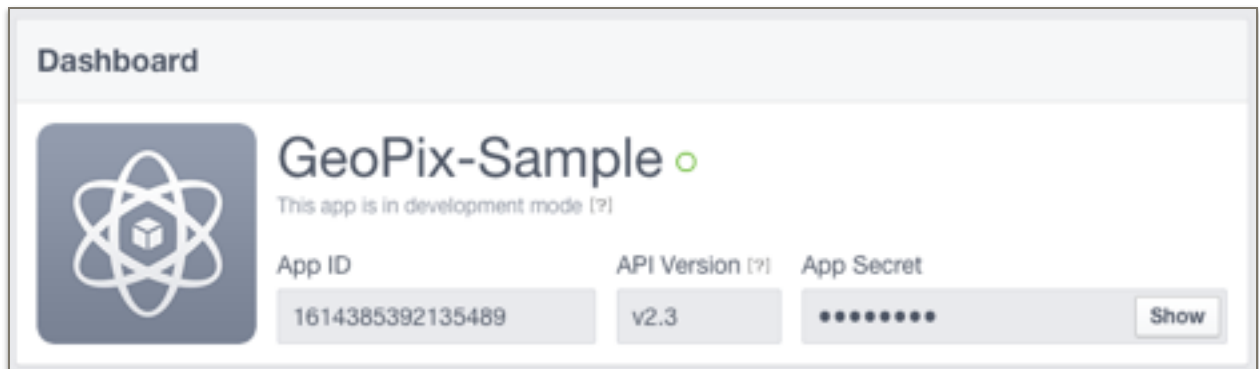
8. Next scroll back to the top and click the "Skip Quick Start" button, and you will be able to see your app dashboard.

Dashboard

GeoPix-Sample ○
This app is in development mode [?]

| App ID	API Version [?]	App Secret	
1614385392135489	v2.3	••••••••	Show

Create a new iOS 8 App on Bluemix

Next we need to setup the backend infrastructure for our mobile application.

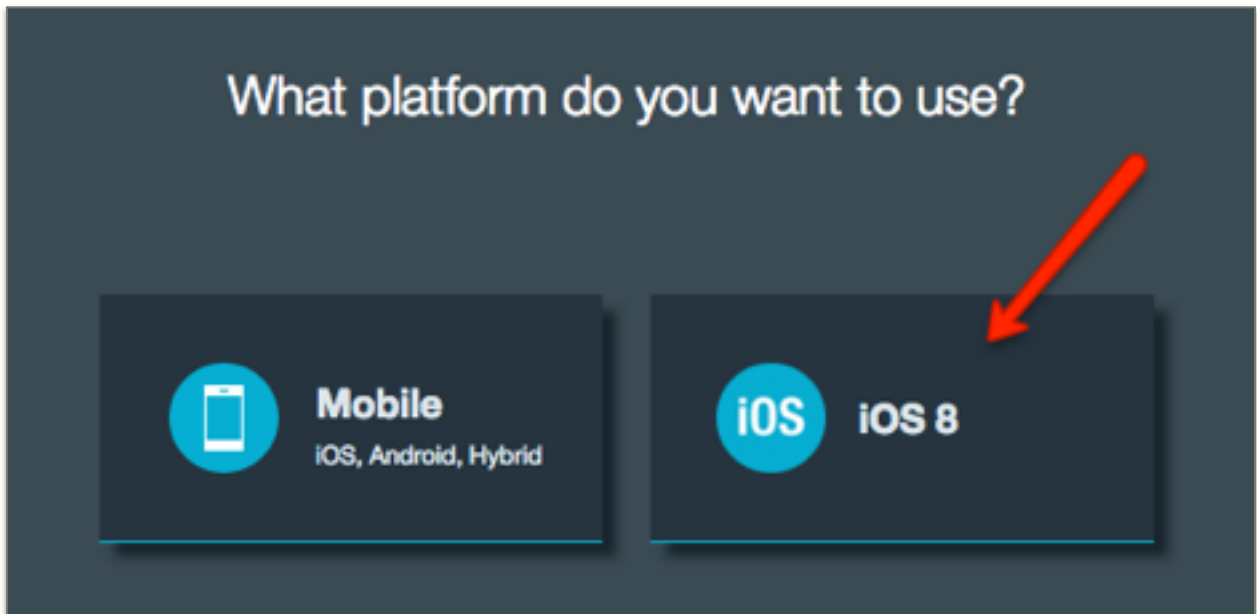1. Log into your IBM Bluemix Dashboard at: https://console.ng.bluemix.net/?ace_base=true/#/resources
2. From the Dashboard Click on the "Create An App" button
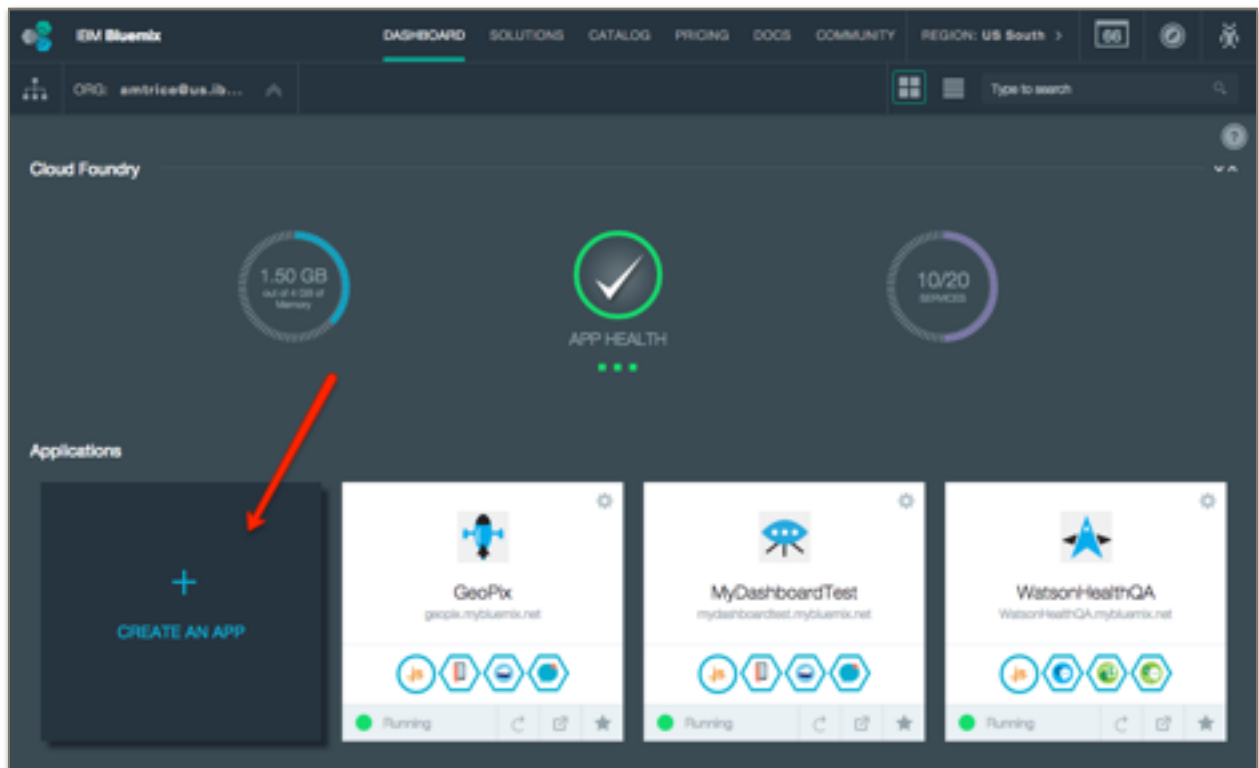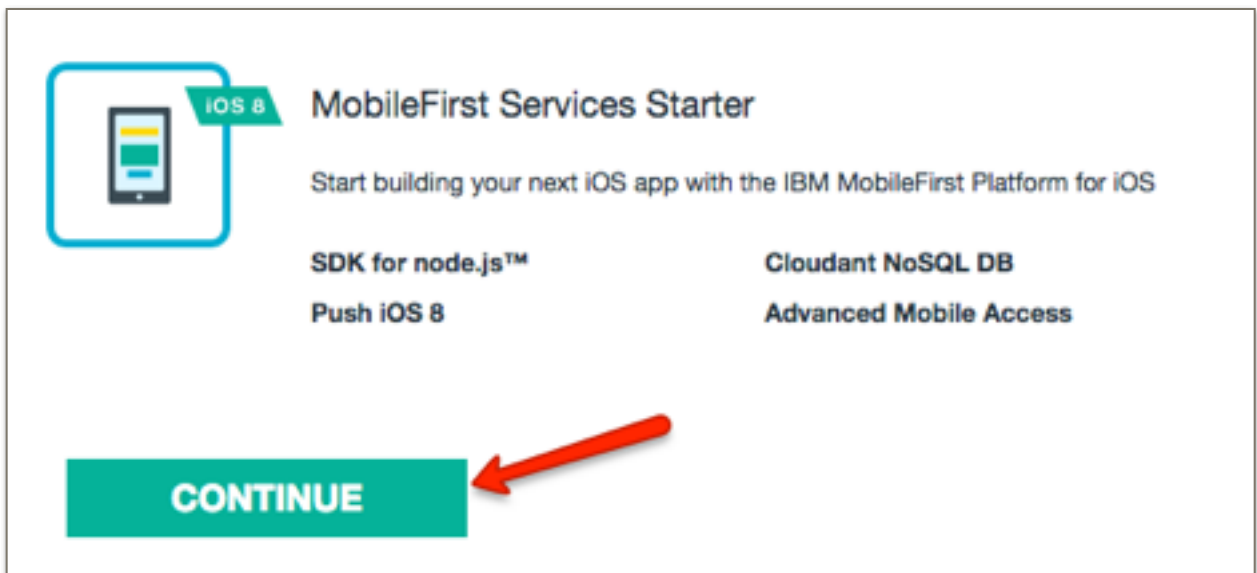

3. You will next be prompted to select the kind of app you are creating. Select the "Mobile" option
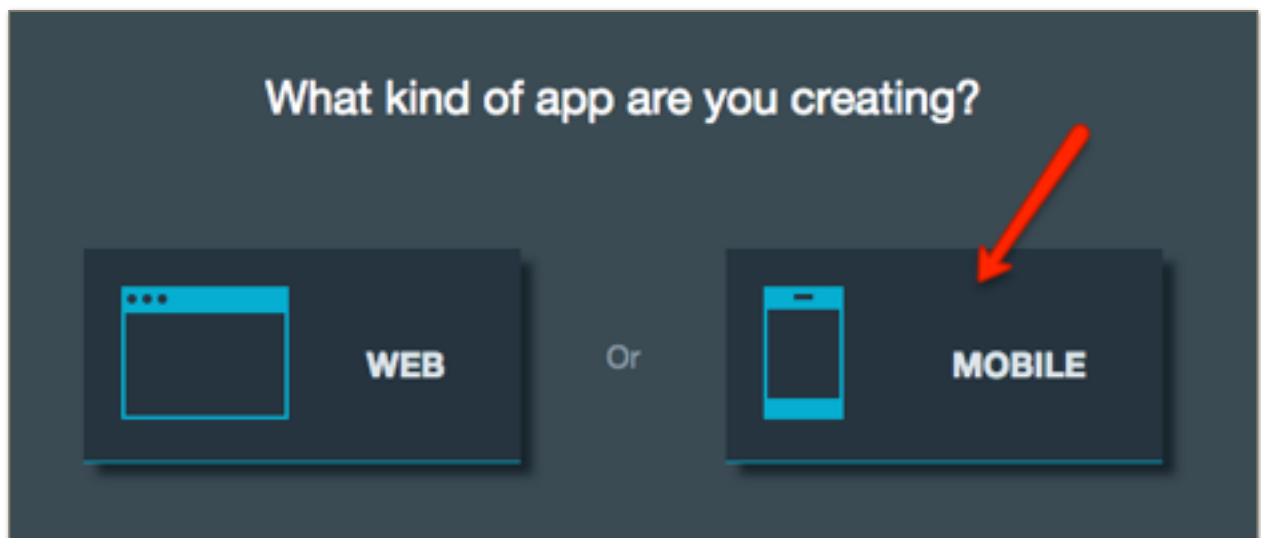
4. Next select the iOS 8 platform template

5. You will be presented with details about the MobileFirst Services Starter template for iOS 8. Click "Continue" to proceed

6. Next enter your app name on Bluemix (this must be unique in Bluemix), and click Finish to proceed

What do you want to name your new app?

APP NAME

GeoPix-Sample

FINISH

7. The backed app infrastructure will now be created and you will be prompted to configure your client Xcode project. Here we will need to Enter the bundle ID and Version number for our app. This bundle ID should match what you entered for the Facebook App ID, and will match that app ID that we will later specify for the native Xcode project.


1 Enter the Bundle ID and Version number   Docs

Bundle ID com.tricedesigns.GeoPix-Sample   Version 1.0

8. If you were setting up a new project you would need to install the Bluemix SDK. Since we are already have the starter app configured, just scroll down and hit the "Done" button on the bottom right of the screen. ***You must do this, or else the client registration will not be completed.

CANCEL   DONE

9. After hitting the "Done" button, you will be brought to the Application Dashboard screen. Here you could add additional services or APIs, or scale up server instances if you need them.

10. Next we want to turn on Facebook user authentication. In the left sidebar menu, select "Advanced Mobile Access"
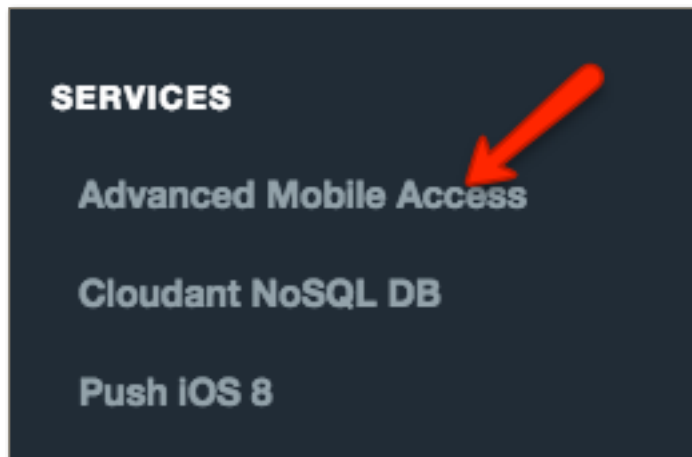
11. Here we can view client information for the Bluemix app.   You can see your platform bundle ID(s) and version(s). Click on the "Allows Unauthenticated access to the backend" link, and we will add authentication to this app id.
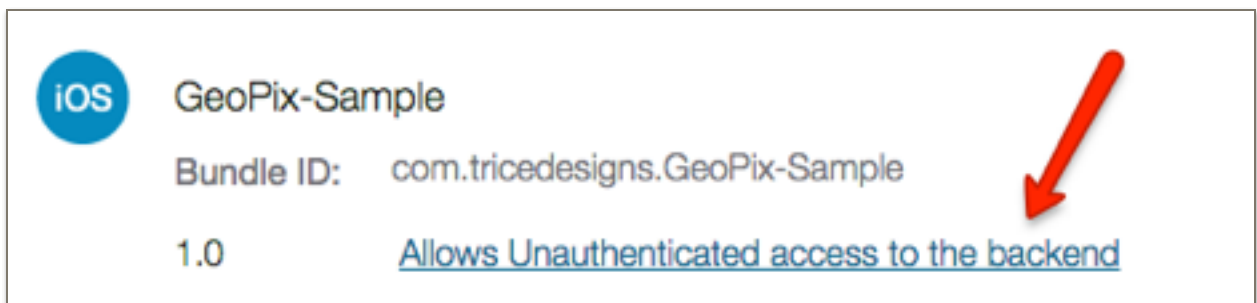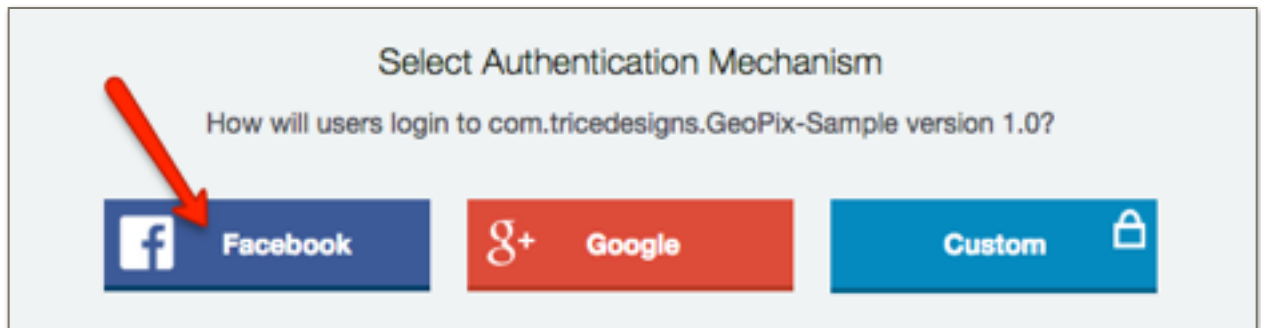

12. Next select the "New Authentication" button
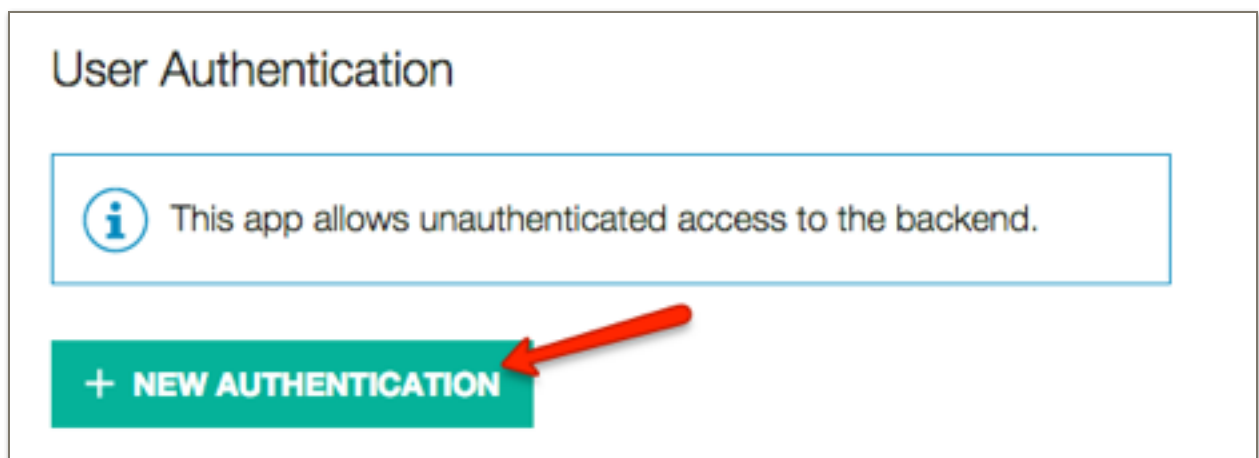
13. Then select Facebook for the authentication mechanism

14. Next we need to specify the Facebook App ID that we created earlier. If you didn't make note of it, you can always go back to Facebook and access the App ID again. Enter your Facebook App ID and click "Save".
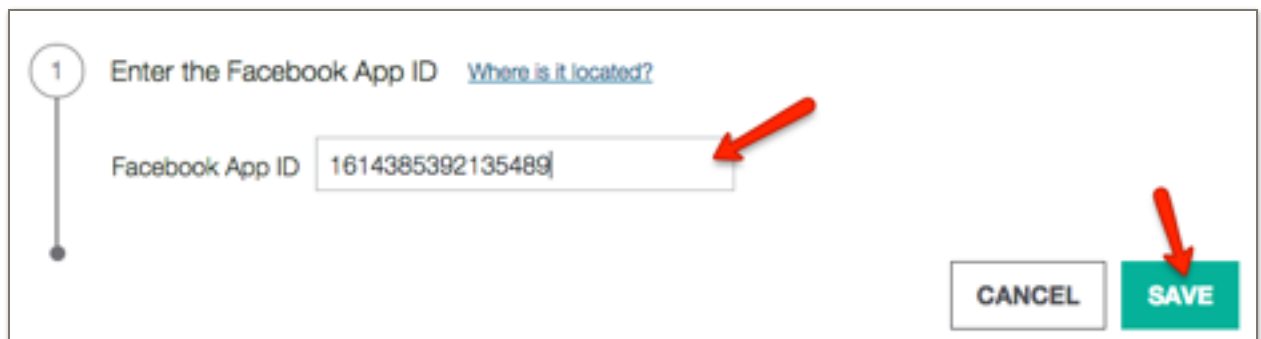
15. Facebook has now been setup as an authentication mechanism for your Bluemix app backend. You will be presented with steps to authenticate the app, but we can ignore those for now since the sample app is already configured.

Setting up the Xcode Workspace

We're now ready to start working on our mobile application in Xcode. This must be done on a Mac, native iOS apps cannot be authored on Windows machines.
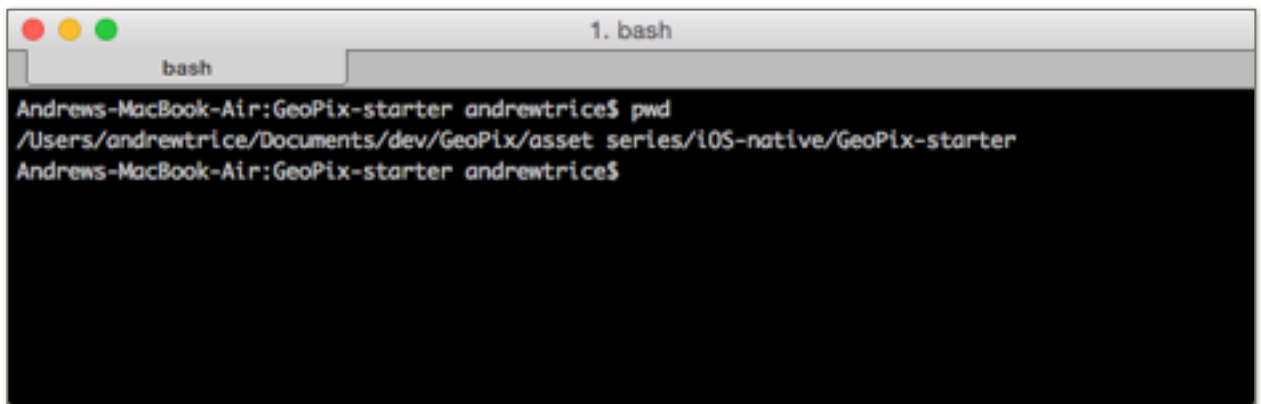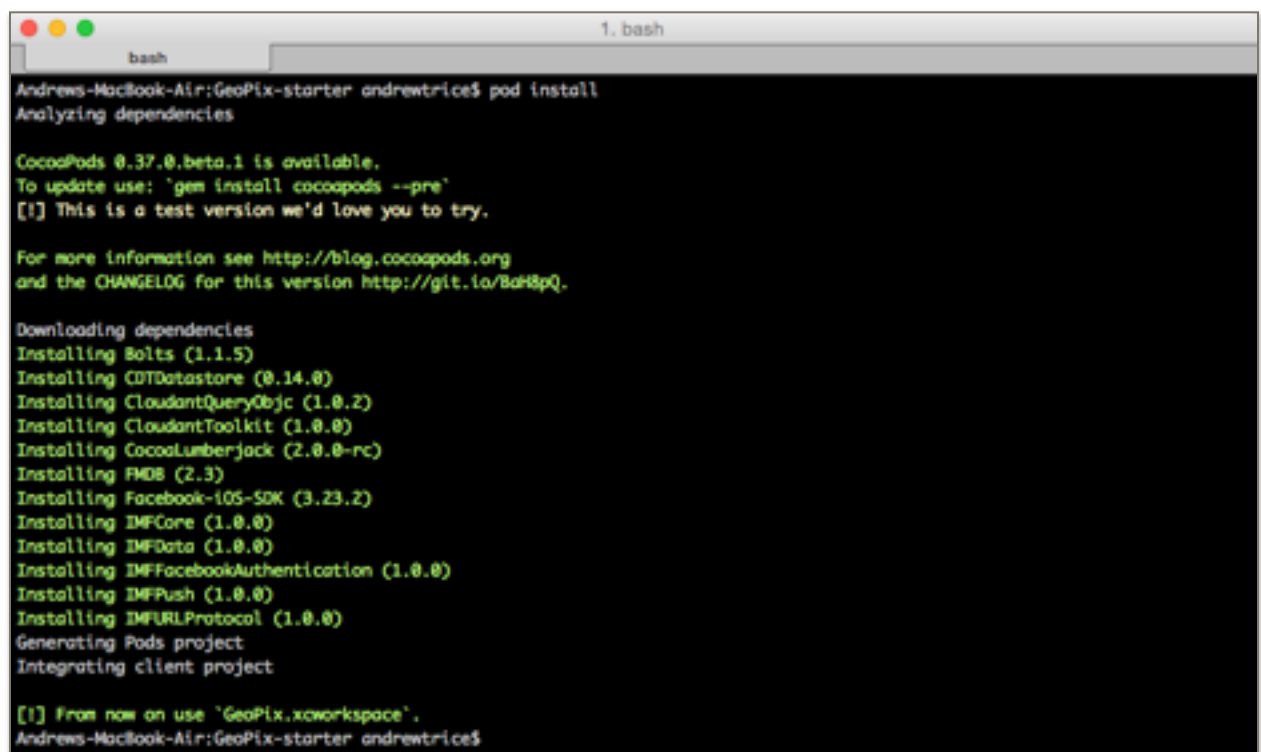
1. Make sure that you have cocoapods installed on your system. You need it to configure the application environment. If you don't have cocoapods, then install it from the instructions at: https://cocoapods.org/

2. Next open up a command line terminal and navigate to the "iOS-native/GeoPix-starter" folder, which you have downloaded and installed from the Git Hub repository at https://github.com/IBM-Bluemix/MobileFirst-Offline-Apps/

```
Andrews-MacBook-Air:GeoPix-starter andrewtrice$ pwd
/Users/andrewtrice/Documents/dev/GeoPix/asset series/iOS-native/GeoPix-starter
Andrews-MacBook-Air:GeoPix-starter andrewtrice$
```

3. Next, run the command "pod install" to download and setup the app environment and dependencies.

```
Andrews-MacBook-Air:GeoPix-starter andrewtrice$ pod install
Analyzing dependencies

CocoaPods 0.37.0.beta.1 is available.
To update use: `gem install cocoapods --pre`
[!] This is a test version we'd love you to try.

For more information see http://blog.cocoapods.org
and the CHANGELOG for this version http://git.io/BaH8pQ.

Downloading dependencies
Installing Bolts (1.1.5)
Installing CDTDatastore (0.14.0)
Installing CloudantQueryObjc (1.0.2)
Installing CloudantToolkit (1.0.0)
Installing CocoaLumberjack (2.0.0-rc)
Installing FMDB (2.3)
Installing Facebook-iOS-SDK (3.23.2)
Installing IMFCore (1.0.0)
Installing IMFData (1.0.0)
Installing IMFFacebookAuthentication (1.0.0)
Installing IMFPush (1.0.0)
Installing IMFURLProtocol (1.0.0)
Generating Pods project
Integrating client project

[!] From now on use `GeoPix.xcworkspace`.
Andrews-MacBook-Air:GeoPix-starter andrewtrice$
```

4. Open Finder and navigate to the "iOS-native/GeoPix-starter" folder and open the **GeoPix.xcworkspace** file. Make sure you open the .xcworkspace file that was just created, not the .xcproject file. The .xcworkspace file contains all of the project dependency information that is required to build the project.

Set configuration variables

Next we need to setup the app configuration so that all of our subsequent steps will work properly.

1. The first thing that you need to do is properly set the app Bundle Identifier to match the Bundle Identifiers entered above. In the sidebar select the GeoPix project, then select the GeoPix target, and update the Bundle Identifier value.
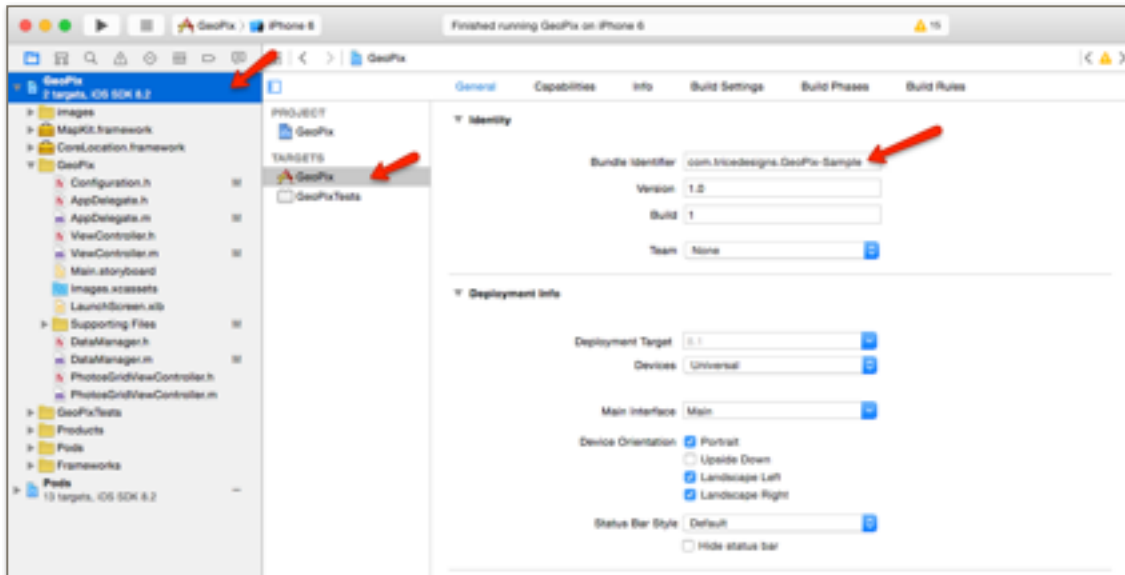
2. Next we need to configure the Info.plist file and enter the values for the Facebook App Auth that were just configured. Open the SupportingFiles/Info.plist file and update the values for FacebookAppID to the Facebook App ID that we created earlier in this process, update FacebookDisplayName to the app name that was created on Facebook, an update the URL types -> Item 0 -> URL Schemes -> Item 0 value to "fb"+ the Facebook App ID. IF we use the values from earlier in these instructions, those values would respectively be: "1614385392135489", "GeoPix-Sample", and "fb1614385392135489".
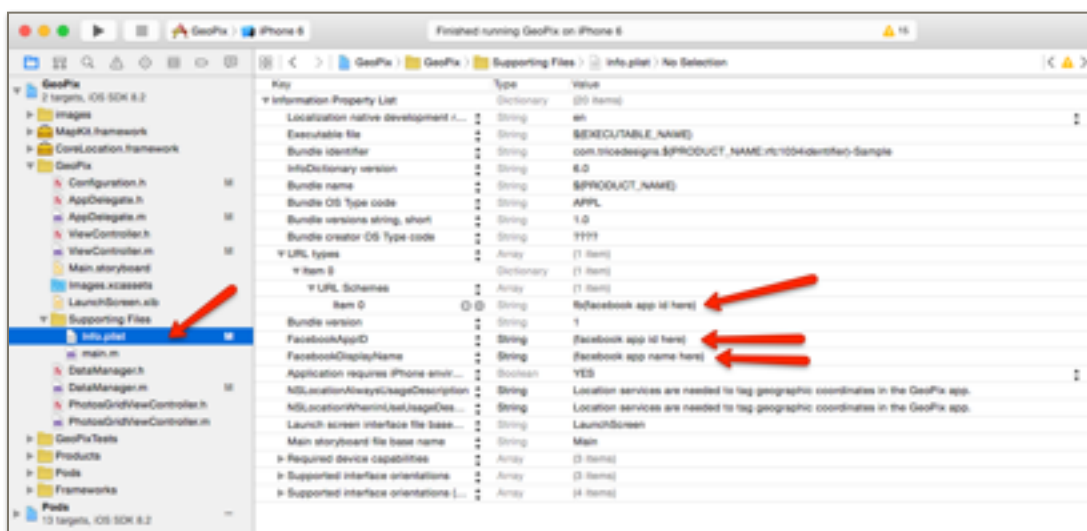
3. Next open the file Configuration.h and update the Bluemix route and clientID. You can get these from the app Dashboard.
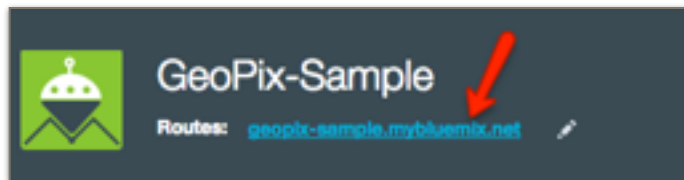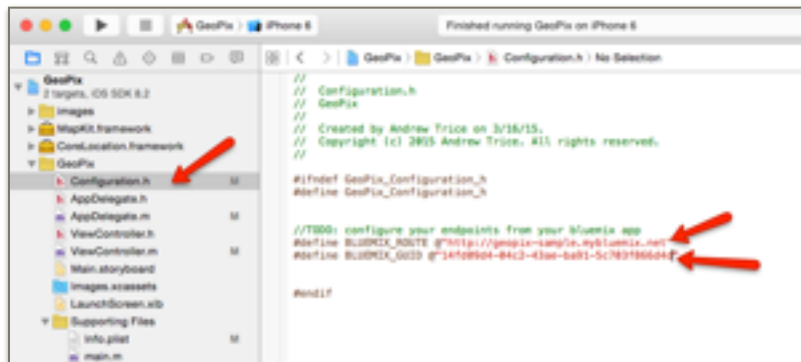
Integrating MobileFirst Platform APIs

The bulk of the application has already been created. This includes capturing images and integrating with location services. Now we're ready to start integrating the MobileFirst platform for operational analytics, logging, and data management.

1. The first thing that we'll do is connect to the MobileFirst backend on Bluemix. Open the AppDelegate.m file, and add the following lines inside of didFinishLaunchingWithOptions to establish the connection to the MobileFirst backend as soon as the application is launched. This will be used to track usage of the app.

```
IMFClient *imfClient = [IMFClient sharedInstance];
[imfClient initializeWithBackendRoute:BLUEMIX_ROUTE
                         backendGUID:BLUEMIX_GUID];
```

2. Next, in the same file & function setup the MobileFirst logger for remote log collection immediately after initializing the MobileFirst client.

```
// capture and record uncaught exceptions (crashes)
[IMFLogger captureUncaughtExceptions];
[IMFLogger setLogLevel:IMFLogLevelTrace];

logger = [IMFLogger loggerForName:NSStringFromClass([self class])];
[logger logDebugWithMessages:@"initializing bluemix client..."];

[IMFLogger send];
```

3. Now let's enable the Facebook app authorization. Immediately after the MobileFirst logging has been setup, use the IMFFacebookAuthenticationHelper class to setup authorization with a single line of code:

```
[[IMFFacebookAuthenticationHandler sharedInstance] registerWithDefaultDelegate];
```

4. Then add the Facebook authentication application: openURL callback function. (This is directly from the user documentation):

```
- (BOOL)application: (UIApplication *)application
            openURL: (NSURL *)url
  sourceApplication: (NSString *)sourceApplication
         annotation: (id)annotation {
    //handle Facebook login
    return [FBAppCall handleOpenURL:url sourceApplication:sourceApplication];
}
```

5. Now add a hook to the Facebook API to track when the application becomes active.

```objectivec
- (void)applicationDidBecomeActive:(UIApplication *)application {

    [FBAppEvents activateApp];
}
```

6. At this point you can run the application to test authentication. No images or captured data will be saved yet. If you launch the application you will be prompted for permission to use location services, and will be redirected to Facebook for OAuth authentication.
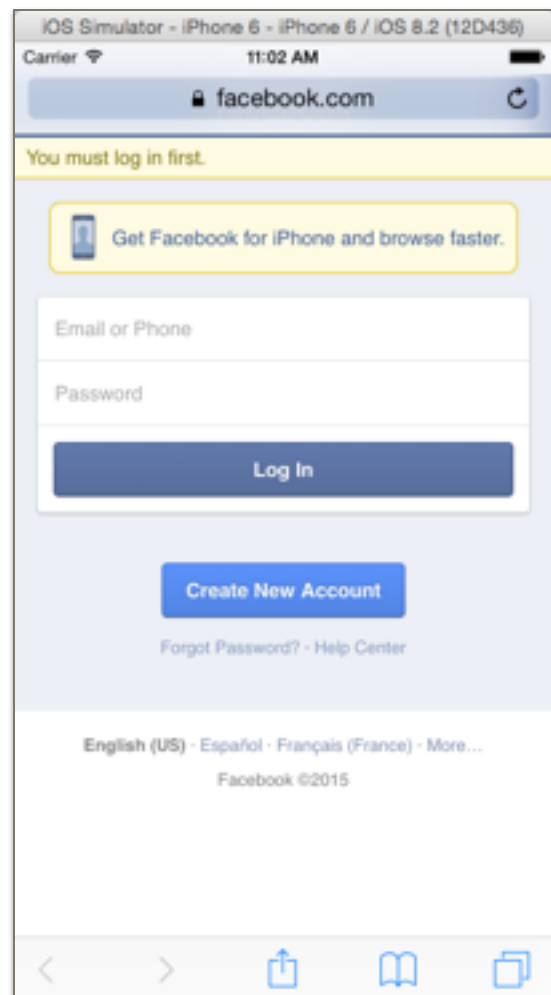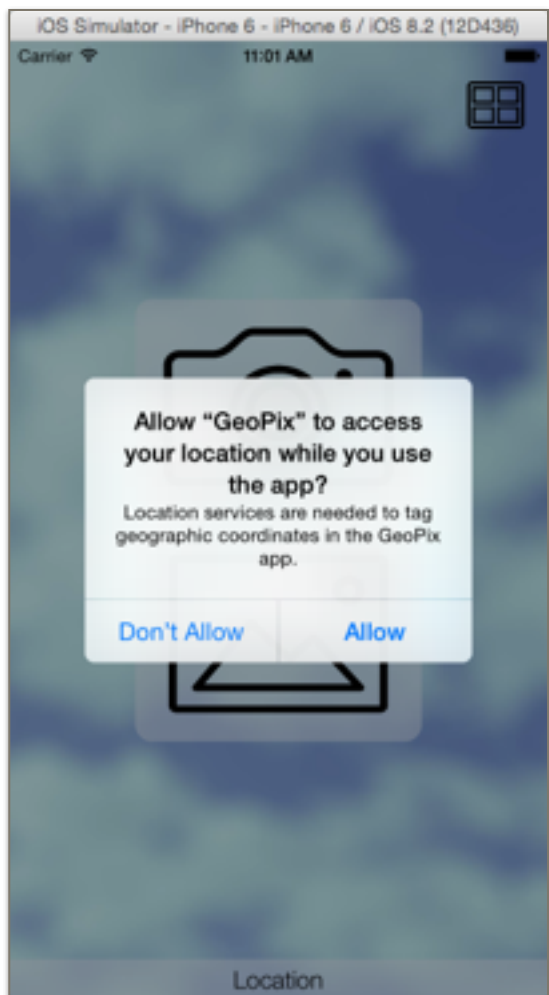
7. Once the user is authenticated you will be presented with a UI for capturing images. The two large buttons are for saving media (take a picture or save an existing picture from the device). The user's location will be displayed at the bottom, and there is a button in the top right that will be used to view existing data that has been captured on this device.

8. All logic for interaction and controlling the user interface resides inside of the ViewController.m file. If you'd like to change any behavior of the UI, you can change it here. Whenever an image is captured (by either camera or existing image), the saveImage: withLocation method on the DataManager is invoked. All logic for saving the image and metadata will be encapsulated within this class. Up next: setting up the data management.

9. The DataManager class is a singleton class (meaning there is only ever a single instance in memory within the application). In this singleton instance resides the logic for managing data, so there is only ever one data store in memory within the application.   Once the DataManager is initialized, the first thing that needs to be done is setup a local data store for saving data locally on the device. Saving to the cloud will come later.   First open up the DataManager.m class and find the init method.

10. Inside of the init method we first need to create a local data store for saving the data locally.

```
// initialize an instance of the IMFDataManager
self.manager = [IMFDataManager sharedInstance];

NSError *error = nil;

//create a local data store
self.datastore = [self.manager localStore:@"geopix" error:&error];

if (error) {
    [logger logErrorWithMessages:@"Error creating local data store %@",error.description];
}
```

11. Next let's look at the saveImage: withLocation method. This method receives the image data and location data as parameters, and will save the data to the local data store.   This method is setup so that all data processing/saving occurs in a background thread so that it does not impace the user experience. In the background thread we will perform 3 steps: 1) create a new document revision (a document is the piece of data that will be stored within the MobileFirst data store based on the Cloudant NoSQL service), 2) save the image as an attachment to the document revision, and 3) save the document revision in the local data store. If you don't already have the saveImage: withLocation method open in Xcode, open it now.

12. First we will create a document revision and set the document body to contain the data that we wish to save. Logically you can think of this as a generic object, kind of like a JSON object. Here we save the location information, plus a timestamp and other metadata.

```
// Create a document
CDTMutableDocumentRevision *rev = [CDTMutableDocumentRevision revision];
rev.body = @{
            @"sort": [NSNumber numberWithDouble:[now timeIntervalSince1970]],
            @"clientDate": dateString,
            @"latitude": [NSNumber numberWithFloat:location.coordinate.latitude],
            @"longitude": [NSNumber numberWithFloat:location.coordinate.longitude],
            @"altitude": [NSNumber numberWithFloat:location.altitude],
            @"course": [NSNumber numberWithFloat:location.course],
            @"type": @"com.geopix.entry"
            };
```

13. Next, we save the image to the local app's temporary directory, and then create a CDTUnsavedFileAttachment containing a reference to the newly saved image. Once we have created the attachment we then assign it to the attachments property of the document revision.

```
//create the image filename and save to a temporary path
NSDate *date = [NSDate date];
NSString *imageName = [NSString stringWithFormat:@"image%f.jpg", [date timeIntervalSince1970]];

NSString *tempDirectory = NSTemporaryDirectory();
NSString *imagePath = [tempDirectory stringByAppendingPathComponent:imageName];

[logger logDebugWithMessages:@"saving image to temporary location: %@", imagePath];

NSData *imageData = UIImageJPEGRepresentation(image, 0.1);
[imageData writeToFile:imagePath atomically:YES];

//create a new attachment
CDTUnsavedFileAttachment *att1 = [[CDTUnsavedFileAttachment alloc]
                                     initWithPath:imagePath
                                     name:imageName
                                     type:@"image/jpeg"];

//add attachment to the document revision
rev.attachments = @{ imageName: att1 };
```

14. Next we save the document revision to the local data store using the save method.

```
//save the attachment to the local data store
[self.datastore save:rev completionHandler:^(id savedObject, NSError *error) {
    if(error) {
        [logger logErrorWithMessages:@"Error creating document: %@", error.localizedDescription];
    }
    [logger logDebugWithMessages:@"Document created: %@", savedObject];
}];
```

15. At this point you could launch the application on your device and start capturing data. You can view debug logs in the Xcode console output to verify that everything is working correctly. Note: You cannot capture images using the camera inside of the Xcode simulator.

16. Next let's setup the capability to view all images that have been captured on the local device. The user interface for displaying local data is encapsulated within the PhotosGridViewController class. In this we will focus on the DataManager class' getLocalData method to query the local data store. In the getLocalData method, a query is performed on the local data store, and then the completionHandler code block is executed - providing a callback to the PhotosGridViewController class once the local query has been performed. A CDTCloudantQuery query instance is created to return all data with type 'com.geopix.entry', and that is used to query the local datastore. You will notice that when we were saving data earlier, we saved the type for all new data as 'com.geopix.entry' intentionally.

```
-(void) getLocalData:(void (^)(NSArray *results, NSError *error)) completionHandler {

    NSPredicate *queryPredicate = [NSPredicate predicateWithFormat:@"(type = 'com.geopix.entry')"];
    CDTCloudantQuery *query = [[CDTCloudantQuery alloc] initWithPredicate:queryPredicate];

    [self.datastore performQuery:query completionHandler:^(NSArray *results, NSError *error) {

        completionHandler( results, error );
    }];
}
```

17. Now if you bring up the view to display local images, you will be able to see media that has been captured and saved locally.
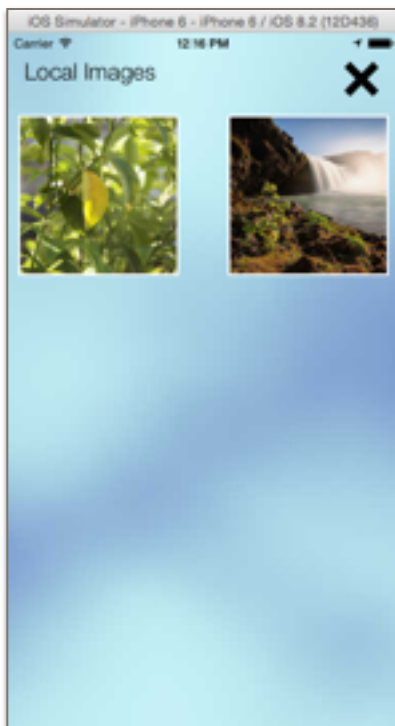
18. Now that we are able to save and retrieve data locally, we can setup replication that will automatically synch local data to the remote data store. Replication can be setup for pushing data to the remote store, pulling data from the remote store, or as a two-way synch. In this case we will setup replication to push all local data to the remote datastore. All of our replication logic will also be contained within DataManager class. Open the DataManager.m class within Xcode and find the init method.

19. Inside the init method we need to create a remote data store to push data into and set user permissions so that we will be able to write data to the remote data store. Note: this will only create a new data store if the remote data store does not already exist - it will not delete any data.

```
//create a remote data store
[self.manager remoteStore:@"geopix" completionHandler:^(CDTStore *store, NSError *error) {
    if (error) {
        [logger logErrorWithMessages:@"Error creating remote data store %@",error.description];
    } else {

        //assign user permissions
        [self.manager setCurrentUserPermissions:DB_ACCESS_GROUP_MEMBERS forStoreName:@"geopix" completionHander:^(BOOL
            success, NSError *error) {
            if (error) {
                [logger logErrorWithMessages:@"Error setting permissions for user with error %@",error.description];
            }
        }];
    }
}];
```

20. Next we need to implement the replication process. Find the "replicate" method inside of the DataManager class.   In here we will setup a CDTPushReplication instance, create a replicator from the remote data manager's replicationFactory method using the push replication, and then trigger the replication process using the replicator's startWithError method. Yes, that is all that we need to do.

```
//create push replication and create the replicator
CDTPushReplication *push = [self.manager pushReplicationForStore: @"geopix"];
self.replicator = [self.manager.replicatorFactory oneWay:push error:&replicationError];

if(replicationError){
    // Handle error if one happens
    [logger logErrorWithMessages:@"An error occurred: %@", replicationError.localizedDescription];
}

self.replicator.delegate = self;

replicationError = nil;
[logger logDebugWithMessages:@"starting replication"];

//start replication
[self.replicator startWithError:&replicationError];
if(replicationError){
    [logger logErrorWithMessages:@"An error occurred: %@", replicationError.localizedDescription];
}else{
    [logger logDebugWithMessages:@"replication start successful"];
}
```

21. To start the replication within our app we can just call the replicate method from inside of the init function. Replication is a "fire and forget" process. We can simply start replication, and any time we make changes to the local data store, those changes will automatically be pushed to the remote store without any additional logic on our part.

22. If you run the app now, all data and images that were captured locally will now be replicated to the remote data store up on Bluemix. However this is a silent process. If you want to provide feedback to the user then you can implement the CDTReplicatorDelegate protocol. This includes methods that are triggered to
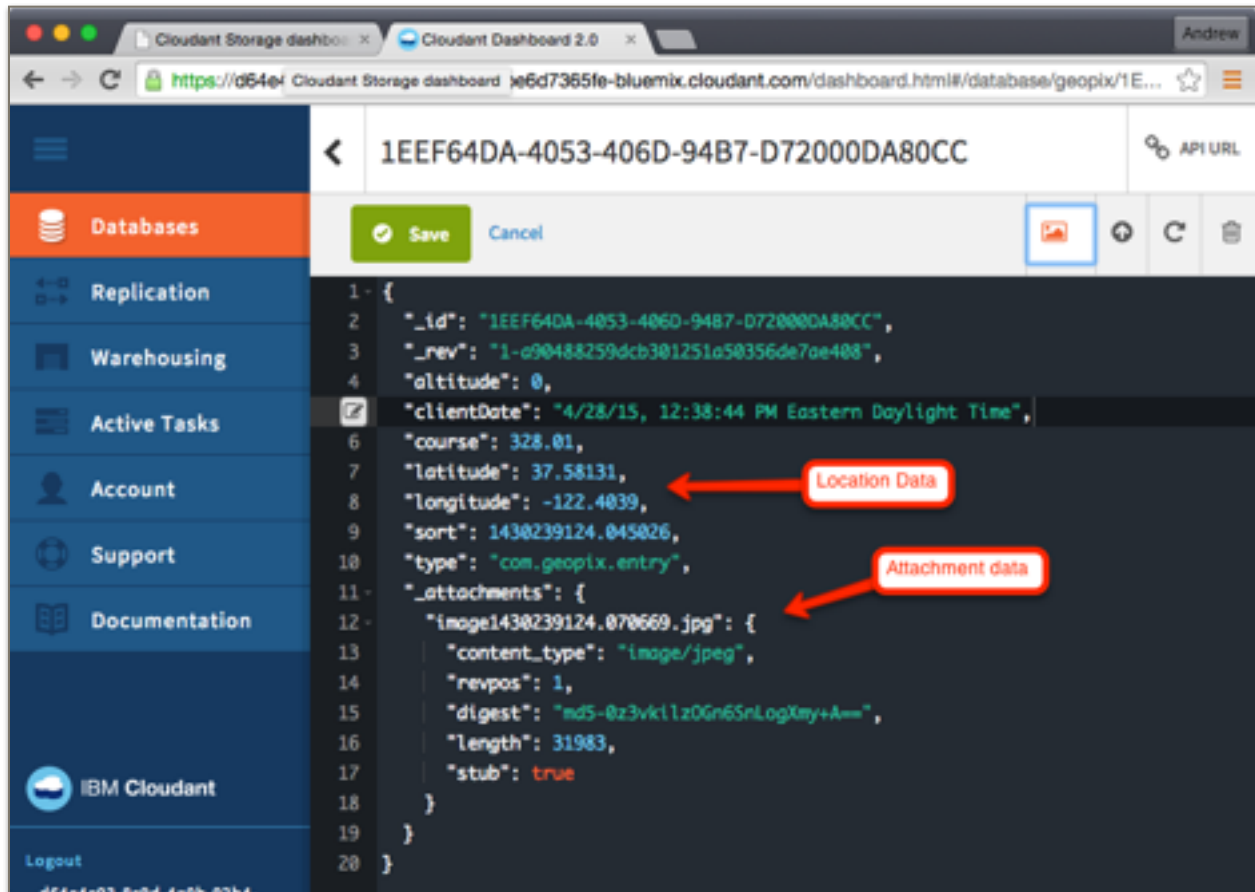
handle completion, error, or progress updates. In the code sample above you can see that the replicator.delegate was assigned to "self" - this is because we have already implemented the protocol methods to provide feedback. Uncomment the the CDTReplicatorDelegate methods (replicatorDidChangeState, replicatorDidChangeProgress, replicatorDidError, replicatorDidComplete) and the application will start displaying replication status in the top left corner of the user interface whenever the replication process is active.

The mobile application is now complete. We are able to capture images, save the images and metadata locally, and leverage the MobileFirst/Cloudant API's replication features to seamlessly save that data up to the cloud. You do not have to worry about online/offline status b/c your data is always saved locally, and replication will occur whenever the application is running and you're online.

Accessing data through the Cloudant Console

We can visit the Bluemix Dashbord and go into your app to see it's status at any time. Select the "Cloudant Dashboard" option to go into the Cloudant web GUI if you'd like to review your data.

Accessing data on the Node.js server
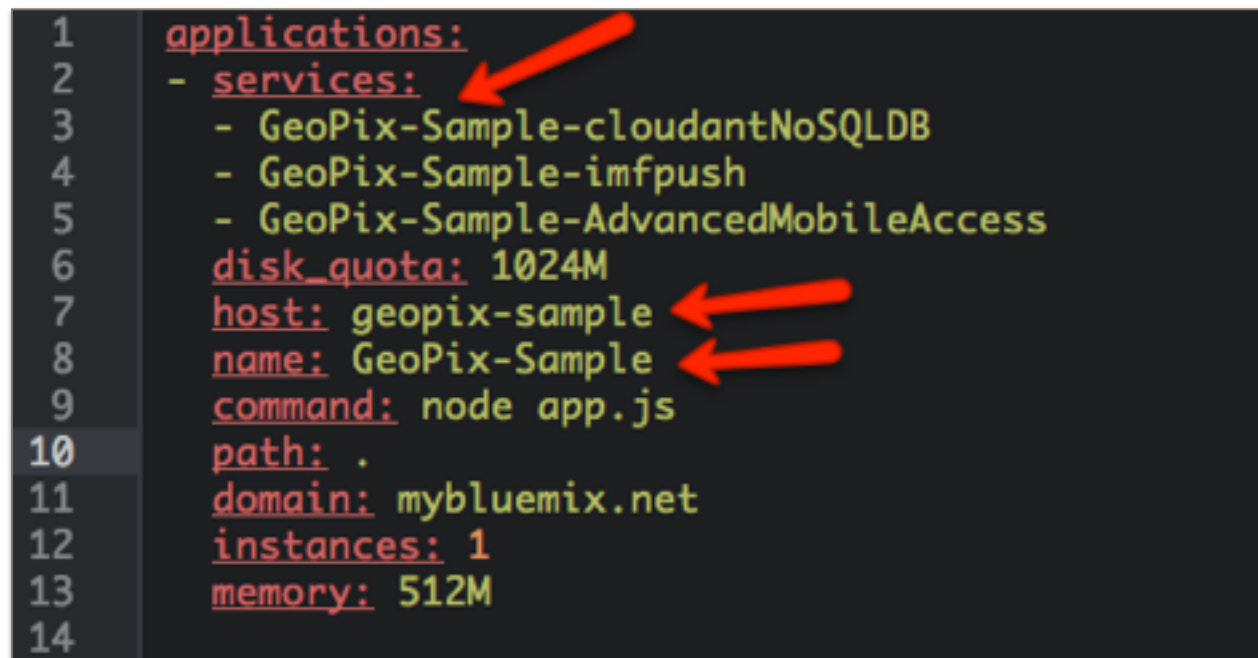
Since we have a Node.js instance with every Bluemix MobileFirst application, we might as well take advantage of it. I have already put together a fairly simple Node.js application that queries the Cloudant NoSQL datastore, and renders it too HTML that can be viewed within the browser.

The web application leverages the Jade templating engine to generate HTML content from the data, and uses the leaflet JavaScript mapping engine with Open Street Map data to display your images on a map.
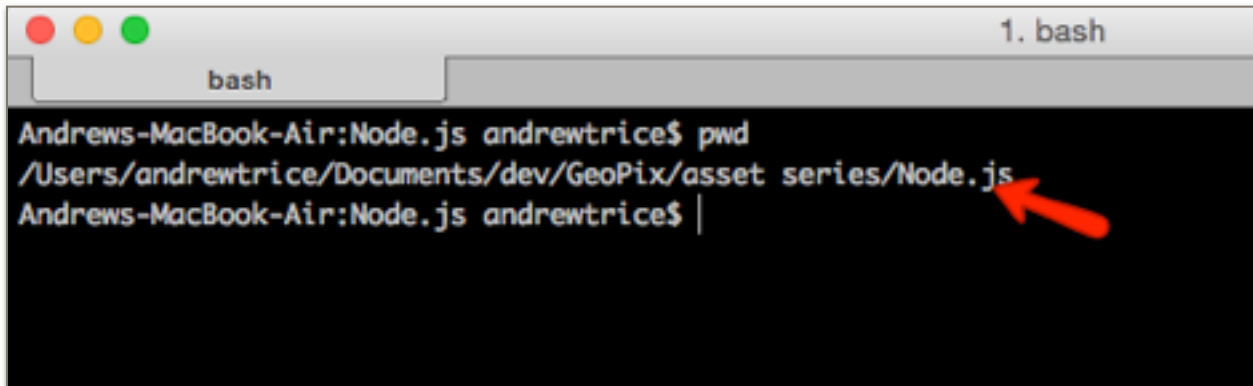
We can deploy this application to our Node.js instance very easily using the Cloud Foundry command line API.

1. If you don't already have Node.js installed, download and install it from nodejs.org

2. If you don't already have the Cloud Foundry command line API installed, download and install it from: https://www.ng.bluemix.net/docs/#!starters/install_cli.html

3. In a code editor, open the Node.js/mainifest.yml file and edit the services, host, and name to match the values for the Bluemix mobile application that you created earlier in this tutorial.

```
1   applications:
2   - services:
3       - GeoPix-Sample-cloudantNoSQLDB
4       - GeoPix-Sample-imfpush
5       - GeoPix-Sample-AdvancedMobileAccess
6     disk_quota: 1024M
7     host: geopix-sample
8     name: GeoPix-Sample
9     command: node app.js
10    path: .
11    domain: mybluemix.net
12    instances: 1
13    memory: 512M
14
```
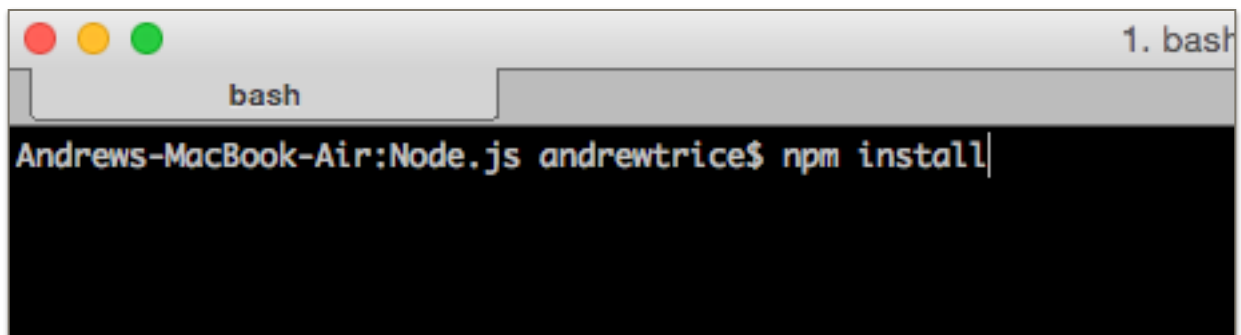
4. Next open a command-line terminal go to the Node.js folder.

```
                                                          1. bash
      bash
Andrews-MacBook-Air:Node.js andrewtrice$ pwd
/Users/andrewtrice/Documents/dev/GeoPix/asset series/Node.js
Andrews-MacBook-Air:Node.js andrewtrice$ |
```
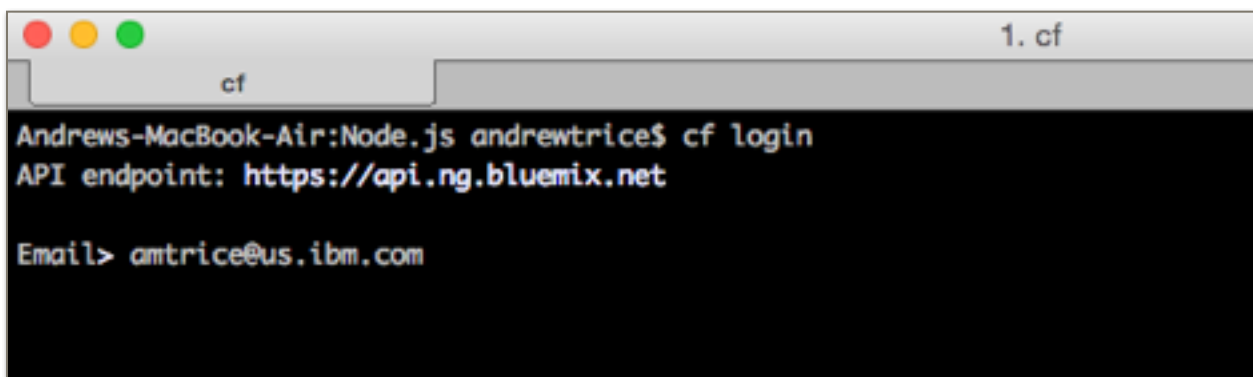
5. Now run the command "npm install" to install all dependencies for the Node.js project.

```
                                                          1. bash
         bash
Andrews-MacBook-Air:Node.js andrewtrice$ npm install|
```

6. We are now ready to push the entire codebase to the Bluemix Node.js instance once all of the dependencies have been downloaded locally.   Use the "cf login" method to log into Bluemix.

```
                                                          1. cf
       cf
Andrews-MacBook-Air:Node.js andrewtrice$ cf login
API endpoint: https://api.ng.bluemix.net

Email> amtrice@us.ibm.com
```

7. Next use the "cf push (app name)" command to push the local Node.js code to the local server. For the app name you should use the app name that was specified when creating the Bluemix app, which should match the manifest we just modified. In my case, I used "cf push GeoPix-Sample".

8. Once the application has successfully deployed to Bluemix you'll be able to see the server status and route/url that was deployed.

9. Now we can visit the Node.js app url in either a desktop or mobile browser.