Hey there Mr/Mrs. Customer,

Thanks for reaching out and for your interest in Branch! Per your request, I've gone ahead and implement a few of Branch's most popular features in an iOS app. This app serves to demonstrate how each feature works in a fun setting — we call it Branchsters.

A few things to consider before you'll be able to access the Branchsters demo I've created for you:

- You'll need an active internet connection in order to clone and access the Branchster's repo on GitHub
- Branchsters was created using iOS's version of the objective-c language, as such some familiarity with objective-c and Xcode (the IDE Branchsters is written with) would be useful
- You'll need to have already installed Xcode in order to access Branchsters once you've downloaded it, if you don't already have Xcode, you can install it here: https://itunes.apple.com/us/app/xcode/id497799835?mt=12]
- Some familiarity with git/GitHub will also be helpful in order to clone the repo where Branchsters is located as well as if you'd like to modify any sections of the app and save them for later. A simple GitHub demo can be found here: https://try.github.io/
- You'll need a paid iOS developer account ($99) in order to access some of the features implemented in the app, specifically Associated Domains which is how Branch's API connects to Branchsters. For a full list of what is included in a free vs. paid Apple developer account, checkout Apple's website here: https://developer.apple.com/support/compare-memberships/
- Before starting the implementation of the code below, you'll first need to follow the initial set-up guide steps for Branch found here: https://docs.branch.io/pages/apps/ios/. You'll only need to read up until the 'Initialize Branch' section. In this version of Branchsters, I've used my keys to test, for your use, you'll want to change them over to your own.

———

Whew, now that we've got a few considerations out of the way, let's get started with a step-by-step guide on what we've added to Branchsters. Below I've included a list of the features in Branchsters of note, complete with some snippets should you prefer to look at the code here rather than in Xcode.

First things first:

You'll want to clone Branchster's repo via GitHub here: https://github.com/aharshbe/BranchInterview

Implementing Branch in an iOS app

Implementing Branch is pretty simple, if you'd like a more detailed guide on how to do so vs. just some code snippets like below or if you are an Android developer, feel free to browse our step-by-step guide here: https://docs.branch.io/pages/apps/android/.

In the *AppDelegate.m* file, I added the following code at the top of the file:

```
#import "Branch/Branch.h"
```

Within the *didFinishLaunchingWithOptions* object, add the following:

```
    [[Branch getInstance] setDebug];
```

```
    // listener for Branch Deep Link data
    [[Branch getInstance] initSessionWithLaunchOptions:launchOptions
andRegisterDeepLinkHandler:^(NSDictionary * _Nonnull params, NSError * _Nullable error) {
        // do stuff with deep link data (nav to page, display content, etc)
        NSLog(@"%@", params);
    }];
```

It instantiates an instance of the Branch object and initializes a session in order to connect to Branch's API.

Just below *didFinishLaunchingWithOptions*, you also need the following code in order to open a connection to Branch's API and track user activity:

```
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:
(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options {
    [[Branch getInstance] application:app openURL:url options:options];
    return YES;
}

- (BOOL)application:(UIApplication *)application continueUserActivity:(NSUserActivity
*)userActivity restorationHandler:(void (^)(NSArray * _Nullable))restorationHandler {
    // handler for Universal Links
    [[Branch getInstance] continueUserActivity:userActivity];
    return YES;
}

- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary
*)userInfo {
    // handler for Push Notifications
    [[Branch getInstance] handlePushNotification:userInfo];
```

Setting up custom event tracking and content sharing

In order to share content via Branch deep links, you'll need to add a couple lines of code. In this particular implementation, I've added some code to the *MonsterCreatorViewController.m* file.

First, make sure Branch's header file is added at the top of the file:

```
#import "Branch/Branch.h"
```

To enable the ability to track specific user events within your app you'll want to add the following code like so:

```
    /* Create content reference */
    BranchUniversalObject *buo = [[BranchUniversalObject alloc]
initWithCanonicalIdentifier:@"content/12345"];
    buo.title = @"Monster Edit View";
    buo.contentDescription = @"User modifying monster";
    buo.publiclyIndex = YES;
    buo.locallyIndex = YES;
    buo.contentMetadata.customMetadata[@"key1"] = @"value1";

    /* Create link refernce */
    BranchLinkProperties *lp = [[BranchLinkProperties alloc] init];
    lp.channel = @"sharing";

    [lp addControlParam:@"$match_duration" withValue: @"2000"];
    [lp addControlParam:@"custom_data" withValue: @"yes"];
    [lp addControlParam:@"look_at" withValue: @"this"];

    /* Track users */
    [[Branch getInstance] setIdentity:@"monster_edit"];
    /* Track events  */
```

```
    [[BranchEvent customEventWithName:@"monster_edit" contentItem:buo] logEvent];
    // logout
    [[Branch getInstance] logout];
```

Above, we first created a reference to a branch object that is used to track user activity. Then we've assigned an identity to our Branch instance in order to track each event. Since we are simply checking that a user has viewed the given activity in our app, we can use a *customEvent*. In order to see that our user has been on the Branchster's monster edit page, we labeled it: "monster_edit". — You can label a custom event anything you'd like or use standard events, more detail on types of events can be found in the guide mentioned before.

****Tracking the user viewed a page with additional state information:**

In the *MonsterViewerViewController.m* file you'll see the following added code:

Your basic branch header again:

```
#import "Branch/Branch.h"
```

A modified version of the code we implemented a bit earlier in the monster edit section:

```
    /* Create content reference */
    BranchUniversalObject *buo = [[BranchUniversalObject alloc]
initWithCanonicalIdentifier:@"content/12345"];
    buo.title = @"Monster view page";
    buo.contentDescription = @"User on monster view page";
    buo.publiclyIndex = YES;
    buo.locallyIndex = YES;
    buo.contentMetadata.customMetadata[@"key2"] = @"value2";

    /* Create link refernce */
    BranchLinkProperties *lp = [[BranchLinkProperties alloc] init];
    lp.channel = @"sharing";

    [lp addControlParam:@"$match_duration" withValue: @"2000"];
    [lp addControlParam:@"custom_data" withValue: @"yes"];
    [lp addControlParam:@"look_at" withValue: @"this"];

    /* Track users */
    [[Branch getInstance] setIdentity:@"monster_view"];
    /* Track events  */
    [[BranchEvent customEventWithName:@"monster_view" contentItem:buo] logEvent];
    // logout
    [[Branch getInstance] logout];
```

**The code above is the exact same as the *MonsterCreatorViewController.m* with minor name changes — here I know there was a request to add state information, in order to implement this particular feature, I'd need a little more time to research/speak with some colleagues. I was able to find a Swift implementation, but the objective-c version was quite elusive. Not to worry, I'll be sure to circle back to you once I figure a solution. In the meantime, if you'd prefer Swift, feel free to checkout our repo with its implementation here: https://github.com/BranchMetrics/Branch-Example-Deep-Linking-Branchster-iOS

Obtaining a shortURL with parameters and displaying it:

The following code achieves this, though it is worth noting I had to make some simple modifications to the *urlTextView* so that the link text was viewable and not obstructed by the Branchster Monster's description:

```
    /* Create deep link */
    [buo getShortUrlWithLinkProperties:lp andCallback:^(NSString* url, NSError* error) {
```

```
        if (!error) {
            self.urlTextView.text = url;
            self.urlTextView.textContainerInset = UIEdgeInsetsMake(16, 20, 0, 20);;
        }
    }];
```

Sending an SMS message via button click and tracking delivery via Branch:

Below is an implementation of sending a deep link via Branch using sms:

Make sure the message headers are added at the top of the file:

```
#import <MessageUI/MessageUI.h>
#import <MessageUI/MFMailComposeViewController.h>
```

To track the click:

```
    BranchUniversalObject *buo = [[BranchUniversalObject alloc]
initWithCanonicalIdentifier:@"content/12345"];
    buo.title = @"Clicked Message";
    buo.contentDescription = @"User clicked share via sms";
    buo.publiclyIndex = YES;
    buo.locallyIndex = YES;
    buo.contentMetadata.customMetadata[@"key3"] = @"value3";

    /* Create link refernce */
    BranchLinkProperties *lp = [[BranchLinkProperties alloc] init];
    lp.channel = @"sharing";

    [lp addControlParam:@"$match_duration" withValue: @"2000"];
    [lp addControlParam:@"custom_data" withValue: @"yes"];
    [lp addControlParam:@"look_at" withValue: @"this"];

    /* Track users */
    [[Branch getInstance] setIdentity:@"clicked_share_sms"];
    /* Track events  */
    BranchEvent *event = [BranchEvent customEventWithName:@"clicked_share_sms" contentItem:buo];

    event.customData[@"monster_name"] = self.monsterName;
    event.customData[@"monster_description"] = self.monsterDescription;
    event.customData[@"monster_title"] = self.monsterMetadata[@"og_title"];
    event.customData[@"mosnter_image_url"] = self.monsterMetadata[@"og_image_url"];
    [event logEvent];

    // logout
    [[Branch getInstance] logout];
```

To send the link via sms with parameters:

```
 NSMutableDictionary *params = [NSMutableDictionary dictionary];
        params[@"referringUsername"] = @"RandomKey";
        params[@"referringUserId"] = @"RandomValue";

        [self.progressBar show];

        [[Branch getInstance] getShortURLWithParams:params andChannel:@"SMS"
andFeature:@"Referral" andCallback:^(NSString *url, NSError *error) {
            if (!error) {
                // Check to make sure we can send messages on this device
                if ([MFMessageComposeViewController canSendText]) {
                    MFMessageComposeViewController *messageComposer =
[[MFMessageComposeViewController alloc] init];

                    // Set the contents of the SMS/iMessage -- be sure to include the URL!
```

```
                    [messageComposer setBody:[NSString stringWithFormat:@"I created a monster
named %@ 💖!\n%@.\nCreate your own monster here 🔮: %@", self.monsterName,
self.monsterDescription, url]];


                    messageComposer.messageComposeDelegate = self;
                    [self presentViewController:messageComposer animated:YES completion:^{
                        [self.progressBar hide];
                    }];

                } else {
                    [self.progressBar hide];
                    [[[UIAlertView alloc] initWithTitle:@"Sorry" message:@"Your device does not
allow sending SMS or iMessages." delegate:nil cancelButtonTitle:@"Okay" otherButtonTitles:nil]
show];
                }
            }
        }];
```

To confirm the sms share was successful:

```
- (void)messageComposeViewController:(MFMessageComposeViewController *)controller
                 didFinishWithResult:(MessageComposeResult)result {
    if (MessageComposeResultSent == result) {
        NSLog(@"finished presenting");
        // track successful share event via sms
        [[Branch getInstance] userCompletedAction:@"share_sms_success"];
    }
    [self dismissViewControllerAnimated:YES completion:nil];
}
```

Upon clicking the sms message icon, the user's click event will be tracked, then a Branch link is created, the description is loaded into the *messageComposer*, and the message is displayed. Once the message is started, it's logged. Thanks to the *messageComposeViewController*, once the message is sent, it is tracked as being successful or errors out which is also tracked.

———

And that's how the cookie crumbled, code snippets complete!

**If you'd like to view the events mentioned above, please note you'll need to change the Project settings of Branchsters in Xcode to reflect your personal Brach details — for a guide as to how to do this, please follow the first couple steps here: https://docs.branch.io/pages/apps/ios/ and create a Branch account here: https://dashboard.branch.io/ in order to view the events in your own Branch Dashboard.

I hope the following examples were simple to understand and met the guidelines you outlined in your email to us, should you have any other questions/concerns regarding Branch deep links with iOS or any of our other services, please don't hesitate to reach out, we'd be glad to help.

Until then, best wishes for a wonderful rest of your day and an even better week going forward!

Cheers,

Austin Harshberger

Thanks for being a part of Branch!