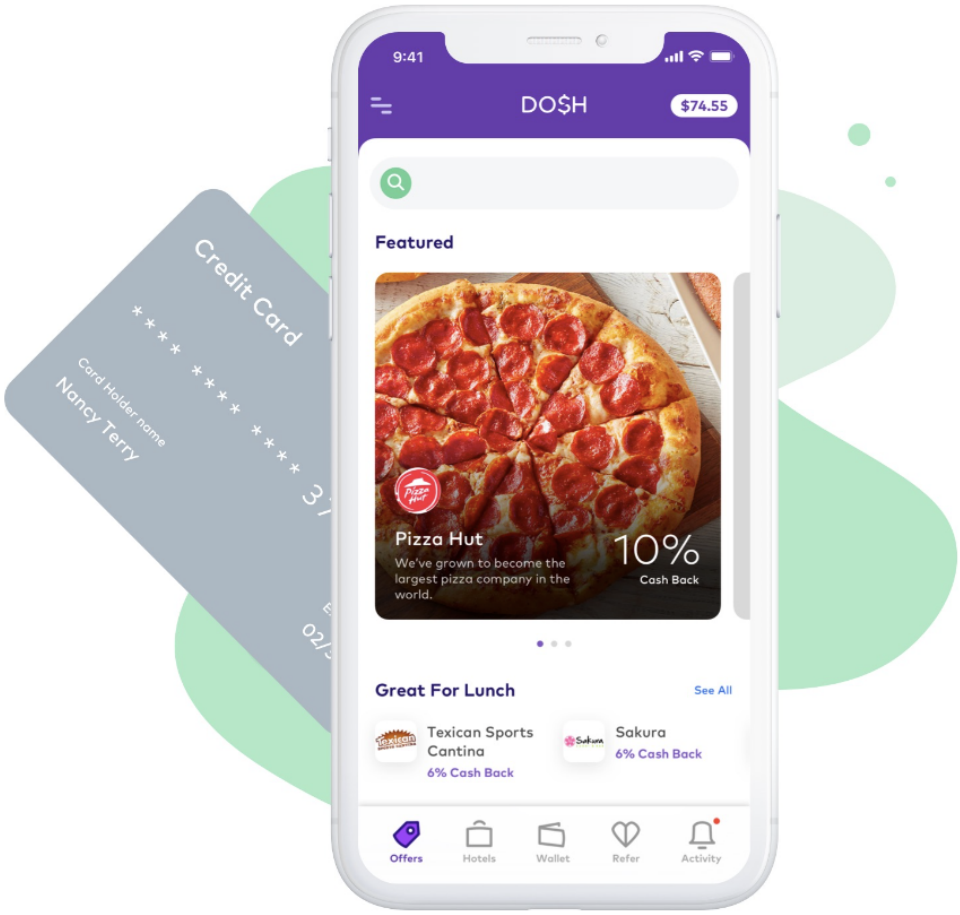
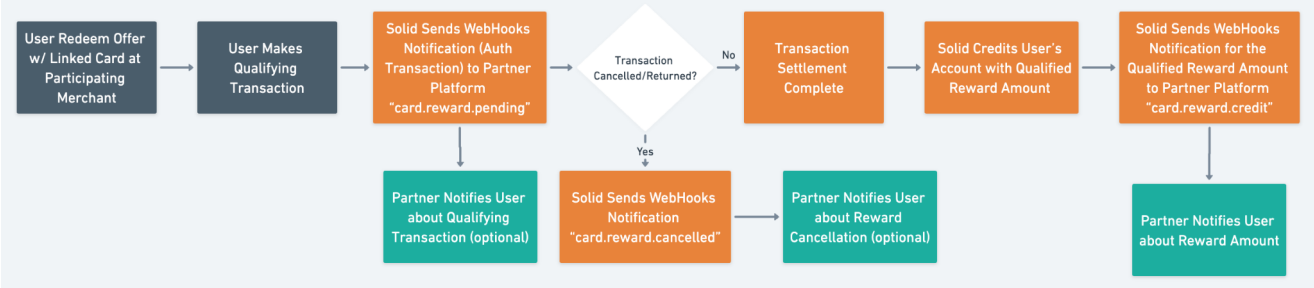
Overview

The rewards solution makes it simple for platforms to increase card value by leveraging the Solid platform. The primary integration path is through an embeddable mobile rewards experience that showcases a rich variation of offer content through a series of provided modules.

The offer's module supports a wide variety of offer content including: In-store offers, online offers, featured offers / just for you, offers by category, local / nearby offers, curated offer collections, limited time offers, and more.



How it Works



Terms & Conditions

https://help.solidfi.com/hc/en-us/articles/9196440416539

Mobile SDK Integration

The Mobile SDK makes it quick and easy to build an excellent rewards experience into your existing mobile application.

The Mobile SDK is a native iOS and Android library for showcasing cardholder rewards. It's an embeddable rewards experience that enables you to showcase a rich variation of offer content using provided modules.

Requirements

iOS

The SDK requires Xcode 13.0 or higher. It is compiled using the iOS 14 SDK and is compatible with apps targeting iOS 13 or above.

Android

The SDK requires a minSdkVersion of 21 and is built with a targetSdkVersion of 29.

Getting Started

Integrating the Mobile SDK consists of adding the SDK to your mobile project and initializing it using your provided API key.

1. Add the SDK Dependency

iOS

Framework Installation

1. Direct Downloads

Direct download is available from the following URL. Replace <major.minor.patch> in the URL with the SDK version you would like to download.

https://poweredby-sdk-release.dosh.com/ios/2.8.0/PoweredByDosh.zip

1. Drag PoweredByDosh.xcframework into your project navigator to add it to your project file.
2. Select your target, and in the General configuration tab add PoweredByDosh.xcframework to the Frameworks, Libraries, and Embedded Content section. Set the framework to Embed & Sign
3. (Recommended) Support crash symbolication
   Crash symbolication is enabled by including the PoweredByDosh dSYMs and BCSymbolMaps into your app's built archive.
   - **Xcode 12** - Crash symbolication is automatically supported when compiling with Xcode 12 or later.
   - **Xcode 11** - To enable crash symbolication when compiling with Xcode 11, add a Run Script Build Phase after the Embed Frameworks build phase for your target, and execute the following script:

```
# Execute script to manually copy debug symbols into your built app.
# NOTE: Replace "${PROJECT_DIR}/Frameworks" with the correct path for your project.
Bash "${PROJECT_DIR}/Frameworks/PoweredByDosh.xcframework/copy-framework-symbols.sh"
```

2. **CocoaPods**

The SDK is provided as a precompiled xcframework and may be included in your project through:

Include the following declaration in your Podfile:

```
# Podfile
pod 'PoweredByDosh'
```

Crash symbolication is automatically supported when including the dependency with CocoaPods.

Note: CocoaPods version 1.10.0 or later is required for binary xcframework dependencies.

3. **Swift Package Manager**

Swift Package Manager is available for SDK versions 2.3.0 and later when building with Xcode 12 by using the following URL as the package source:

```
https://github.com/dosh-com/powered-by-dosh-ios-releases.git
```

When integrating the package, you will have the option to choose which framework targets to include.

Crash symbolication is automatically supported when including the dependency with Swift Package Manager.

Android

The SDK can be accessed via our maven repository. Please add the following to the buildscript block in your root build.gradle script:

```
allprojects {
    ...
    repositories {
        ...
```

```
        //The maven repository for Dosh.
        maven {
            url = "https://dosh.jfrog.io/dosh/libs"
            //Note: There is a bug with Artifactory and though this
is a public repository
            //      you will need to submit blank credentials in
order for access to the repository.
            credentials {
                username = ""
                password = ""

            }
        }
    }
}
```

Then in your dependency block in your app build.gradle add the following:

```
implementation('com.dosh:poweredby:2.7.0')
```

The SDK includes the ability to display Offers on a map. In order to support that functionality you will need to provide a Google Maps key via your application manifest. To generate that key please follow these instructions.

The SDK uses the RxJava library version 1.2.0. If your project is also using RxJava version 1.x.y and want to avoid collisions or our version overriding yours add the following:

```
implementation('com.dosh:poweredby:2.7.0'){
    exclude group: 'io.reactivex', module: 'rxjava'
}
```

2. Initialize the SDKs

The first call that you make to the SDK should be to initialize. Below is the dosh-application-id which needs to be used within the app.

dosh-application-id = <To be provided via secure email>

Following are the three steps to be followed for SDK initialization.

a) User Location

The user's current location can be passed into the SDK using the userLocation property. This location will be used to provide more relevant offers to the user. Updating the location does not automatically reload the feed, but will be used in any subsequent network queries. Because of this, it is recommended to pass in the user's location before presenting the SDK.

Note that the userLocation field is an optional parameter. **If location is not directly provided, the experience will degrade to leveraging location context associated with the IP Address of the incoming request.** This context will be used to approximate the user's location and return local offers relative to that approximation. Local offers require location context to be displayed and leveraging IP-based geo ensures that content is always available to the end user.

iOS

```
Dosh.instance?.userLocation = CLLocation(latitude: 30.275039, longitude:
-97.740320)

Example:
   //pass the user's location before Dosh.initialize..
   let isLocationEnabled = LocationHelper.shared.isLocationEnable()
   if isLocationEnabled {
     let latitude = LocationHelper.shared.latitude
     let longitude = LocationHelper.shared.longitude
        Dosh.instance?.userLocation = CLLocation(latitude: latitude, longitude:
longitude)
    }
```

Android

```
val location = android.location.Location(LocationManager.GPS_PROVIDER).apply {
    latitude = 30.275039
    longitude = -97.740320
}
PoweredByDosh.instance?.userLocation = location

Example:

val location = android.location.Location(LocationManager.GPS_PROVIDER).apply {
            latitude = currentLat
            longitude = currentLong
```

```
        }
        PoweredByDosh.instance?.userLocation = location
```

b) Initialize

```
Call Solid platform API to get DOSH token:

PROD-TEST: https://test-api.solidfi.com/v1/card/doshtoken
PROD-LIVE: https://api.solidfi.com/v1/card/doshtoken
```

iOS

```
// This should be done before any other calls to the PoweredByDosh SDK.
let dosh = Dosh.initialize(applicationId: "dosh-application-id")

/// Enabling this will print integration related logs to the console
dosh.debugLoggingEnabled = true
```

Android

```
// This should be done before any other calls to the PoweredByDosh SDK.
var dosh = PoweredByDosh.initialize("dosh-application-id", applicationContext)

Example:

dosh = PoweredByDosh.initialize(dosh-application-id, this)
```

c) Implement User Authorization Callback

For the SDK to provide user-specific content, the user authorization callback must be implemented. User authorization between your app and Dosh is coordinated by providing the SDK with an authorization token that is signed with a secret key that we both share.

From the app's perspective, the SDK provides a closure that we will call at any point in time when a new authorization token is needed. You can expect a token to be requested when:

1. The experience is launched the first time.
2. User information is cleared.
3. The existing token is expiring.

iOS

```
// Call Solid API /v1/card/doshtoken to get JWT token and pass to completion

dosh.userAuthorization = { completion in
  // Implement call to your server to generate a signed token.
  // When complete, call the completion block, passing in the
  // signed token. If your request for the token fails, then
  // pass nil into the completion block.


  //Exmaple:
    <doshTokenAPIcallMethod> { (response, errorMessage) in
            guard errorMessage == nil else {
              completion(nil)
              return
            }
            completion(response?.accessToken)
   }
}


//Sandbox Testing:

dosh.userAuthorization = { completion in
completion("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJEdW1teVVzZXJJZDEyMyIsImlzcy
I6InNsZGZpOjkwZDFjMmU0LTBlNGYtNDE0ZC1iZjg3LTQ3YWRjZmUzNDdmNSIsImV4cCI6MTY5OTQ0NTE2NSwia
nRpIjoiODlhMDQ4N2Y2ZTFhNGUxMGEyNDUwNDE0MjgzMzQ0YTgifQ.DD18arXzdwN4tVSvg6tSZ8mTF9TP6Jf-C
09fwJFOthM")

}
```

Any time the app's current user changes, such as when the user logs out, the user's information should be cleared from the SDK:

```
dosh.clearUser()
```

Android

```
// Call Solid API /v1/card/doshtoken to get JWT token and pass to completion

dosh.authorize { sessionTokenResponse ->
    // Implement call to your server to generate a signed token.
    // When complete, call the sessionTokenResponse lambda, passing
    // in the signed token.
    ...
    sessionTokenResponse("server-generated-jwt")
}

Example:
        dosh.authorize {
            it(doshSessionToken)
```

```
        }


//Sandbox Testing:

  dosh.authorize {
  it("eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJEdW1teVVzZXJJZDEyMyIsImlzcyI6InNsZ
  GZpOjkwZDFjMmU0LTBlNGYtNDE0ZC1iZjg3LTQ3YWRjZmUzNDdmNSIsImV4cCI6MTY5OTQ0NTE2NSwianRpIjo
  iODlhMDQ4N2Y2ZTFhNGUxMGEyNDUwNDE0MjgzMzQ0YTgifQ.DD18arXzdwN4tVSvg6tSZ8mTF9TP6Jf-C09fwJ
  FOthM")
}
```

Any time the app's current user changes, such as when the user logs out, the user's information should be cleared from the SDK:

```
dosh.clearUser()
```

4. Presenting the Rewards Experience

Presenting the rewards experience displays a full-screen view that is managed by the SDK. The section below on theming will cover how to style this view to match the look of your app.

iOS

To present the experience on iOS, call the presentRewards(from:) function in the SDK, passing in a view controller to be used as the presenting view controller for a modal presentation.

```
class MyViewController: UIViewController {
    ...
    @IBAction func openDoshRewards() {
        Dosh.instance?.presentRewards(from: self)
    }
}
```

Alternatively, there is a debugging experience that can be presented to assist with initial integration of the SDK. This experience is intended to only be used by the engineers integrating the SDK, and is not intended to be shown to consumers. The experience provides a visual representation of: the visual theme that was provided and validation of the authentication integration. To present this experience:

```
class MyViewController: UIViewController {
```

```
    ...
    @IBAction func openDoshIntegrationChecklist() {
        Dosh.instance?.presentIntegrationChecklist(from: self)
    }
}
```

Android

To present the experience on Android, call the showDoshRewards function in the SDK, passing in the context to be used to display the DoshRewardsActivity.

```
class MainActivity : AppCompatActivity() {

    private fun openDoshRewards() {
        PoweredByDosh.instance?.showDoshRewards(this)
    }
}
```

Alternatively, there is a debugging experience that can be presented to assist with initial integration of the SDK. This experience is intended to only be used by the engineers integrating the SDK, and is not intended to be shown to consumers. The experience provides a visual representation of: the visual theme that was provided and validation of the authentication integration. To present this experience:

```
class MainActivity : AppCompatActivity() {

    private fun openDoshIntegrationChecklist() {
        PoweredByDosh.instance?.presentIntegrationChecklist(this)
    }
}
```

Mobile SDK - Theming

The Mobile SDK makes it easy to customize the default theme of the SDK to match your custom branding. This document helps outline how the provided values map to the current experience and how to override them.

Fonts and Colors

The SDK uses four different font weights (light, regular, medium, and bold) and three main color variations (header, primary, and interactive). The following code samples demonstrate how to override those values. To better understand how those values map to various pieces of the UX, click here to see a visual representation.

iOS

To support this in your iOS app, create a concrete implementation of the PoweredByDoshTheme protocol. The protocol requires that the main colors be defined, but provides default values for the fonts, gray colors, and interactive elements so that they can be optionally overridden. The following example provides the main colors and custom fonts, but chooses to use the default values for everything else:

```
struct CustomTheme: PoweredByDoshTheme {
    var headerColor: UIColor { UIColor.red }
    var primaryColor: UIColor { UIColor.green }
    var interactiveColor: UIColor { UIColor.blue }
    var boldFontName: String? { "MarkOT-Bold" }
    var mediumFontName: String? { "MarkOT-Medium" }
    var regularFontName: String? { "MarkOT-Book" }
    var lightFontName: String? { "MarkOT-Light" }
}
```

Android

To support this in your Android app, you can also override the color and font values that are in the PoweredByDoshTheme by specifying what values you want to override in your colors.xml file

In your colors.xml resource file

```
<color name="dosh_core_header">Your color here</color>
<color name="dosh_core_primary">Your color here</color>
<color name="dosh_core_interactive">Your color here</color>
```

And to override the fonts, you can upload your own font type in your font folder.
Afterwards create a font-family resource file for each font you want to override from us.

Our current default font family names are

- `dosh_font_bold`
- `dosh_font_medium`
- `dosh_font_regular`
- `dosh_font_light`

To be able to use your own fonts, you'll have to create a font family with the same name as one of the above font family names. For example, if you wanted to override our dosh_bold_font you would create a resource in the font folder by the name dosh_bold_font.xml and specify your custom font within the attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<font-family xmlns:android="http://schemas.android.com/apk/res/android">
    <font
        android:font="@font/your_font_name"
        android:fontWeight="400"
        android:fontStyle="normal"/>
</font-family>
```

You'll have to create a font family for each font you want to override

Since Fonts in XML are a newer feature on Android, you can learn more about them here

Navigation Bar

iOS

The SDK also supports theming the navigation bar used throughout the SDK experience. If the SDK theming is not provided, then the navigation bar defaults to the primary theme color as the background with white UI elements.

```
struct CustomTheme: PoweredByDoshTheme {
    ...
    var navigationBarStyle: DoshNavigationBarStyle = DoshNavigationBarStyle(
        backgroundColor: UIColor.white,
        separatorColor: UIColor(hex: "d6dbe0"),
        backButtonImage: UIImage(named: "myBackButton")!,
        titleTextStyle: DoshTextStyle(weight: .medium, size: 16, color:
UIColor.black))
}
```

Android

In your colors.xml resource file

```
<color name="dosh_core_navigation_bar_title">Your color here</color>
<color name="dosh_core_navigation_bar_background">Your color here</color>
<color name="dosh_core_navigation_bar_back_button">Your color here</color>
<color name="dosh_core_navigation_bar_separator">Your color here</color>
```

For the navigation bar font, you can override our dosh_font_navigation_bar. To do so, create a resource in the font folder by the name dosh_font_navigation_bar.xml, and then specify your custom font within the attributes.

The back button icon can also be overridden by adding a Drawable resource with the name dosh_nav_back_button.xml. If you override this resource, please ensure the color of your Drawable is the same color as dosh_core_navigation_bar_back_button in your colors.xml.

Our Activity uses the theme, PoweredBy.Default.Main. If you would like to customize items, like the status bar or navigation bar color, you can do so by overriding our theme and then the items you would like customized.

```
<style name="PoweredBy.Default.Main">
   <item name="android:statusBarColor">@color/colorPrimaryDark</item>
   <item name="android:navigationBarColor">@color/colorPrimaryDark</item>
</style>
```

Supported Theme Overrides

Note: currently we do not support the ability to customize all of our PoweredBy.Default.Main theme. The following is a list of items that can be safely customized.

- `android:statusBarColor`
- `android:navigationBarColor`
- `android:windowLightNavigationBar`

Program Name

When you first launch the SDK you'll notice the navigation bar title defaults to Offers. Additionally, if you're leveraging the optional account summary module, you'll notice the title there defaults to Offers Summary. These titles are configurable to match the nomenclature of your specific program. For example, If your program refers to offers as Rewards, you could set that as the program name and it would update the feed title to Rewards and the account summary title to Rewards Summary. To update the program name, please refer to the following examples.

iOS

```
dosh.rewardsProgramName = "Rewards"
```

Android

```
dosh.rewardsProgramName = "Rewards"
```

Section Titles

The section titles in the offers feed can be customized. These fields are configured server-side, so please work with your Partner Manager if you prefer a value other than the default.

Logo and Header Styles

Another optional theming feature is the ability to customize logo image treatments as well as the header shape on the view showing details about a selected merchant. Logos can either have a circular or rounded rectangle treatment. The header shape be either be diagonal across the screen or rectangular. To better understand how those values visually effect the UX, click here to see a visual representation.. An example choosing circular logos and a rectangular header is shown below:

iOS

```
struct CustomTheme: PoweredByDoshTheme {
    ...
    let logoStyle: DoshLogoStyle = .circular
    let brandDetailsHeaderStyle: DoshBrandDetailsHeaderStyle = .rectangular
}
```

Then assign the theme to the PoweredByDosh instance after the SDK has been initialized:

```
Example:

let dosh = Dosh.initialize(applicationId:"dosh-application-id")
dosh.theme = CustomTheme()
```

Android

```
Example:

val uiOptions = PoweredByUiOptions("Rewards", DoshLogoStyle.CIRCLE,
DoshBrandDetailsHeaderStyle.RECTANGLE)
```

Web Experience - Integration

Web Experience provides a technology agnostic way to quickly and easily integrate our comprehensive rewards experience into your existing web application.

The integration point between your application and our technology is a dynamically built URL, which your application can use to initialize and present rewards Web Experience to your consumer.

Initialize and trigger rewards web experience

1. Make a post call on PROD-TEST or PROD-LIVE url setting 'sd-person-id' header
2. Fetch accessToken from response and generate JWT token
3. Once a valid JWT has been generated for the respective user, it should be added as a 'jwt' attribute within a stringified JSON object (format shown in javascript example below). That object should then be base64 encoded.
4. After base64 encoding the authentication payload containing the user's valid JWT, you are ready to construct the full entry point URL. The full URL should adhere to the following format where ${encodedPayload} would be replaced with the payload you generated in step 2a.

   `https://poweredby.dosh.com/partners/solidfi/setup/${encodedPayload}`

JavaScript

```
//<url>
//PROD-TEST: https://test-api.solidfi.com/v1/card/doshtoken
//PROD-LIVE: https://api.solidfi.com/v1/card/doshtoken

//add axios header
axios.defaults.headers.common['sd-person-id'] = <solid_person_id>;

axios.post(<url>, config)
  .then(data => {
      const base64encode = btoa;
      let payload = {
         jwt: data.accessToken,
      };
      //Enabling this will print integration related logs to the console
      payload.debugLoggingEnabled = true
```

```
    const payloadString = JSON.stringify(payload);
    const encodedPayload = base64encode(payloadString);
    let url = 'https://poweredby.dosh.com/partners/solidFi/setup/';
    url += encodedPayload;
    window.open(url, '_blank');
});
```

Webhook Notifications

The following Webhook notifications are sent to the partner's endpoint:

- `card.reward.pending` – Sent upon qualifying Authorization transaction
- `card.reward.credit` – Sent upon qualifying Settled transaction
- `card.reward.canceled` – Sent upon transaction cancellation/return prior to transaction settlement