

Custom Authenticator and Login Module in native iOS applications

This tutorial explains how to implement the client-side of custom authenticator and login module in native iOS.

Prerequisite: Make sure that you read Custom Authenticator and Login Module (../) first.

Implementing the client-side authentication

Create a native iOS application and add the MobileFirst native APIs as explained in Configuring a native iOS application with the MobileFirst Platform SDK (../.../hello-world/configuring-a-native-ios-with-the-mfp-sdk/).

Storyboard

In your storyboard, add a ViewController containing a login form.



Challenge Handler

- Create a `MyChallengeHandler` class as a subclass of `ChallengeHandler`.

```
1 | @interface MyChallengeHandler : ChallengeHandler
```

- Call the `initWithRealm` method:

```
1 | @implementation MyChallengeHandler
2 | //...<br />
3 | -(id)init:{
4 |     self = [self initWithRealm:@"CustomAuthenticatorRealm"];
5 |     return self;
6 | }
```

- Add implementation of the following `ChallengeHandler` methods to handle the custom authenticator and login module challenge:

1. `isCustomResponse` method:

The `isCustomResponse` method is invoked each time a response is received from the MobileFirst Server. It is used to detect whether the response contains data that is related to this challenge handler. It must return either `true` or `false`.

```

1  @implementation MyChallengeHandler
2  //...
3  -(BOOL) isCustomResponse:(WLResponse *)response {
4      if(response && [response getResponseJson]){
5          if ([[response getResponseJson] objectForKey:@"authStatus"]) {
6              NSString* authRequired = (NSString*) [[response getResponseJson] objectForKey:@"authStatus"];
7              return ([authRequired compare:@"required" == NSOrderedSame];
8          }
9      }
10     return false;
11 }
12 @end
13

```

2. `handleChallenge` method:

If `isCustomResponse` returns `true`, the framework calls the `handleChallenge` method. This function is used to perform required actions, such as hiding the application screen and showing the login screen.

```

1  @implementation MyChallengeHandler
2  //...
3  -(void) handleChallenge:(WLResponse *)response {
4      NSLog(@"A login form should appear");
5      LoginViewController* loginController = [self.vc.storyboard instantiateViewControllerWithIdentifier:@"LoginView"];
6      loginController.challengeHandler = self;
7      [self.vc.navigationController pushViewController:loginController animated:YES];
8  }
9  @end

```

3. `onSuccess` and `onFailure` methods:

At the end of the authentication flow, `onSuccess` or `onFailure` will be triggered

Call the `submitSuccess` method in order to inform the framework that the authentication process completed successfully and for the `onSuccess` handler of the invocation to be called.

Call the `submitFailure` method in order to inform the framework that the authentication process failed and for the `onFailure` handler of the invocation to be called.

```

1  @implementation MyChallengeHandler
2  //...
3  -(void) onSuccess:(WLResponse *)response {
4      NSLog(@"Challenge succeeded");
5      [self.vc.navigationController popViewControllerAnimated:YES];
6      [self submitSuccess:response];
7  }
8  -(void) onFailure:(WLFailResponse *)response {
9      NSLog(@"Challenge failed");
10     [self submitFailure:response];
11 }

```

submitLoginForm

In your login View Controller, when the user taps to submit the credentials, call the `submitLoginForm` method to send the credentials to the MobileFirst Server.

```
1  @implementation LoginViewController
2  //...
3  - (IBAction)login:(id)sender {
4      [self challengeHandler
5          submitLoginForm:@"my_custom_auth_request_url"
6          requestParameters:@{@"username": self.username.text, @"password": self.password.text}
7          requestHeaders:nil
8          requestTimeoutInMilliseconds:0
9          requestMethod:@"POST"];
10 }
```

Registering the challenge handler

Before calling the protected adapter, in order to listen to incoming challenges, make sure to register the challenge handler by using the `registerChallengeHandler` method of the `WLClient` class.

```
1  [[WLClient sharedInstance] registerChallengeHandler:[MyChallengeHandler alloc] initWithViewController:self];
```

Sample application

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/NativeCustomLoginModuleProject.zip>)
the Studio project.

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/iOSNativeCustomLoginModuleProject.zip>)
the Obj-C project.

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/SwiftNativeCustomLoginModuleProject.zip>)
the Swift project.

