Adapter-based authentication in native iOS applications

fork and edit tutorial (https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/#fork-destination-box) | report issue (https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/issues/new)

This tutorial explains how to implement the client-side of adapter-based authentication in native iOS.

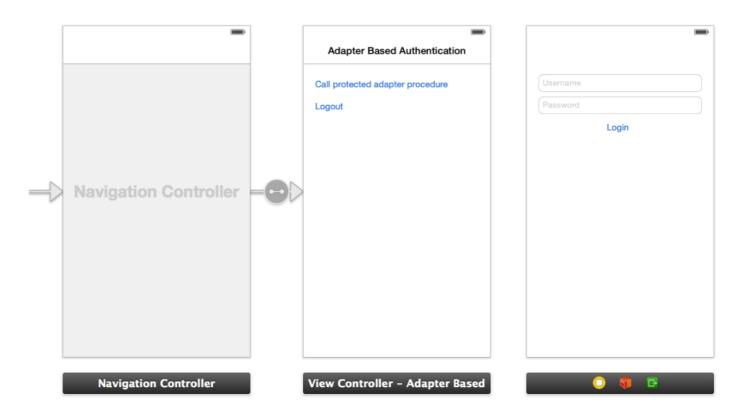
Prerequisite: Make sure that you read Adapter-based authentication (../) first.

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 view controller containing a login form.



Challenge Handler

• Create a MyChallengeHandler class as a subclass of ChallengeHandler.

```
@interface MyChallengeHandler : ChallengeHandler
```

• Call the initWithRealm method:

```
@implementation MyChallengeHandler
//...
-(id)init:{
    self = [self initWithRealm:@"NativeAdapterBasedAuthRealm"]
;
    return self;
}
```

• Add implementation of the following ChallengeHandler methods to handle the adapter-based 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.

```
@implementation MyChallengeHandler
//...
-(BOOL) isCustomResponse:(WLResponse *)response {
   if(response && [response getResponseJson]){
      if ([[response getResponseJson] objectForKey:@"authRequired"]) {
            NSString* authRequired = (NSString*) [[response getResponseJson]
      objectForKey:@"authRequired"];
        return [authRequired boolValue];
      }
    }
    return false;
}
@end
```

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.

```
@implementation MyChallengeHandler
//...
-(void) handleChallenge:(WLResponse *)response {
    NSLog(@"A login form should appear");
    LoginViewController* loginController = [self.vc.storyboard instantiateViewControllerWithIdentifier:@
"LoginViewController"];
    loginController.challengeHandler = self;
    [self.vc.navigationController pushViewController:loginController animated:YES];
}
@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.

```
@implementation MyChallengeHandler
//...

-(void) onSuccess:(WLResponse *)response {
    NSLog(@"Challenge succeeded");
    [self.vc.navigationController popViewControllerAnimated:YES]
;
    [self submitSuccess:response];
}
-(void) onFailure:(WLFailResponse *)response {
    NSLog(@"Challenge failed");
    [self submitFailure:response];
}
```

submitAdapterAuthentication

In your login View Controller, when the user taps to submit the credentials, call the submitAdapterAuthentication method to send the credentials to the your adapter procedure.

```
@implementation LoginViewController<
//...
- (IBAction)login:(id)sender {
   WLProcedureInvocationData *myInvocationData = [[WLProcedureInvocationData alloc
]
   initWithAdapterName:@"NativeAdapterBasedAdapter"
   procedureName:@"submitAuthentication"];
   myInvocationData.parameters = @[self.username.text, self.password.text];
   [self.challengeHandler submitAdapterAuthentication:myInvocationData options:nil];
}</pre>
```

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.

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

Sample application

Click to download

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

Click to download

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

Click to download

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

