

Custom Authenticator and Login Module in native iOS applications

fork and edit tutorial (<https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/authentication-security/custom-authenticator-login-module/custom-authenticator-login-module-native-ios-applications.html>) | report issue (<https://github.ibm.com/MFPSamples/DevCenter/issues/new>)

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`.

```
@interface MyChallengeHandler : ChallengeHandler
```

- Call the `initWithRealm` method:

```

@implementation MyChallengeHandler
//...<br />
-(id)init{
    self = [self initWithRealm:@"CustomAuthenticatorRealm"]
;
    return self;
}

```

- 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`.

```

@implementation MyChallengeHandler
//...
-(BOOL) isCustomResponse:(WLResponse *)response {
    if(response && [response getResponseJson]){
        if ([[response getResponseJson] objectForKey:@"authStatus"]) {
            NSString* authRequired = (NSString*) [[response getResponseJson] objectForKey:@"authStatus"];
            return ([authRequired compare:@"required"] == NSOrderedSame);
        }
    }
    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];
}

```

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.

```

@implementation LoginViewController
//...
-(IBAction)login:(id)sender {
    [self.challengeHandler
        submitLoginForm:@"my_custom_auth_request_url"
        requestParameters:@{@"username": self.username.text, @"password": self.password.text}
    ]

    requestHeaders:nil
    requestTimeoutInMilliseconds:0
    requestMethod:@"POST"];
}

```

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/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.

