Custom Authentication in native iOS applications

Overview

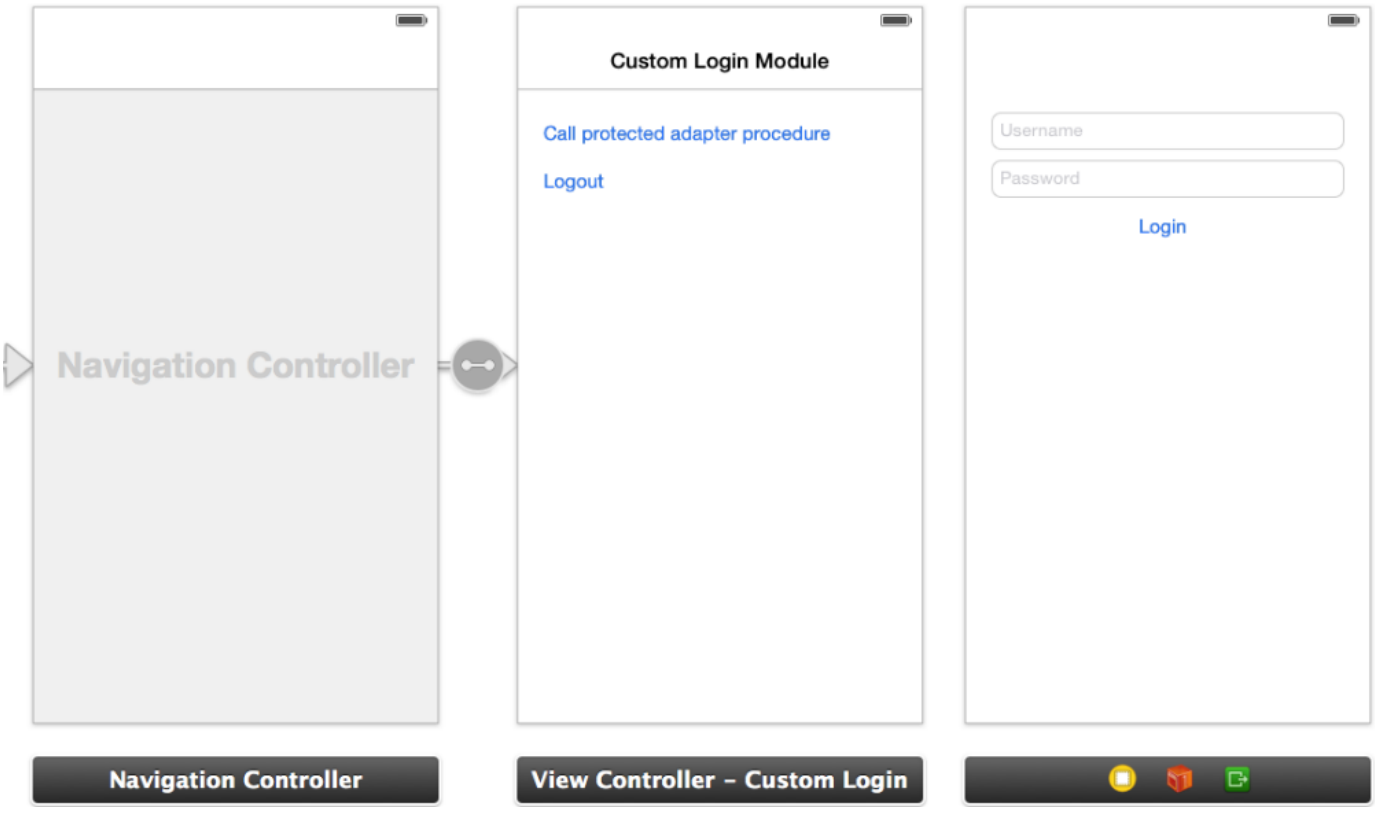
This tutorial explains how to implement the client side of a custom authentication in native iOS.
Prerequisite: Make sure that you read the Custom Authentication (../) tutorial first.

Implementing client-side authentication

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

Storyboard

In your storyboard, add a ViewController that contains a login form.



Challenge handler

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

   ```
   @interface MyChallengeHandler : ChallengeHandler
   ```

2. Call the `initWithRealm` method:

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

3. Add the 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`.

       ```
       -(BOOL) isCustomResponse:(WLResponse *)response {
           if(response &amp;amp;&amp;amp; [response getResponseJson]){
               if ([[response getResponseJson] objectForKey:@"authStatus"]) {
                   NSString* authRequired = (NSString*) [[response getResponseJson] objectFor
       Key:@"authStatus"];
                   return ([authRequired compare:@"required"] == NSOrderedSame);
               }
           }
           return false;
       }
       ```

    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.

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

    3. **`onSuccess` and `onFailure` methods:**
       At the end of the authentication flow, calls to the `onSuccess` or `onFailure` methods are triggered.

        - Call the `submitSuccess` method to inform the framework that the authentication process completed successfully, so that the `onSuccess` handler of the invocation is called.

- Call the `submitFailure` method to inform the framework that the authentication process failed, so that the `onFailure` handler of the invocation is called.

```
-(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 types 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"];
}
```

The Main ViewController

In the sample project, in order to trigger the challenge handler we use the `WLClient invokeProcedure` method.
The protected procedure invocation triggers MobileFirst Server to send the challenge.

- Create a `WLClient` instance and use the `connect` method to connect to the MobileFirst Server:

```
    MyConnectListener *connectListener = [[MyConnectListener alloc] init];
    [[WLClient sharedInstance] wlConnectWithDelegate:connectListener]
;
```

- In order to listen to incoming challenges, make sure to register the challenge handler by using the `registerChallengeHandler` method:

```
    [[WLClient sharedInstance] registerChallengeHandler:[[MyChallengeHandler alloc] initWit
hViewController:self] ];<br />
```

- Invoke the protected adapter procedure:

```
    NSURL* url = [NSURL URLWithString:@"/adapters/AuthAdapter/getSecretData"];
    WLResourceRequest* request = [WLResourceRequest requestWithURL:url method:WLHttp
MethodGet];
    [request sendWithCompletionHandler:^(WLResponse *response, NSError *error) {
    ...
    }];
```

Worklight Protocol

If your custom authenticator uses `WorklightProtocolAuthenticator`, some simplifications can be made:

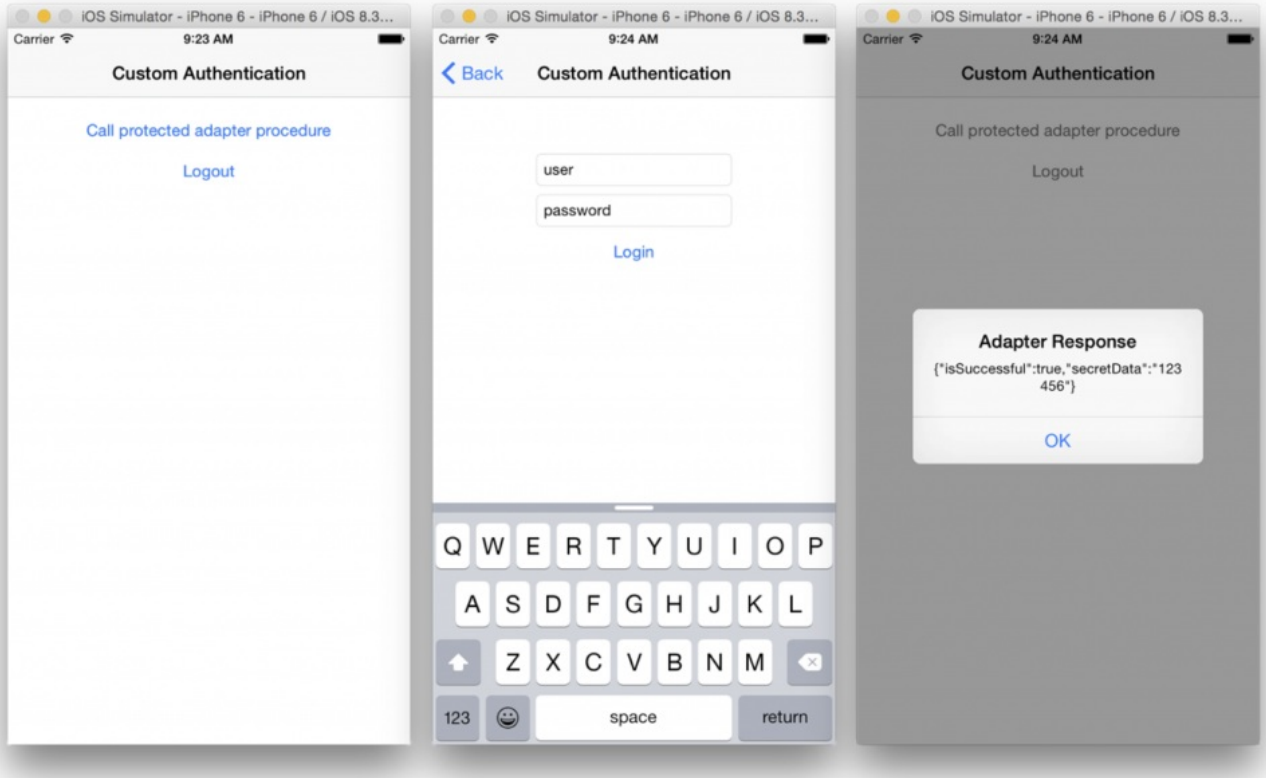
- You can subclass your challenge handler by using `WLChallengeHandler` instead of `ChallengeHandler`. Note the `WL`.
- You no longer need to implement `isCustomResponse` because the challenge handler automatically checks that the realm name matches.
- The `handleChallenge` method receives the challenge as a parameter, not the entire response object.
- Instead of `submitLoginForm`, use `submitChallengeAnswer` to send your challenge response as a JSON object.
- You do not need to call `submitSuccess` or `submitFailure` because the framework will do it for you.

> For an example that uses `WorklightProtocolAuthenticator`, see the Remember Me (../../../advanced-topics/remember-me/) tutorial or this video blog post (file:////home/travis/build/MFPSamples/DevCenter/_site/blog/2015/05/29/ibm-mobilefirst-platform-foundation-custom-authenticators-and-login-modules/).

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomAuth) the MobileFirst project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomAuthObjC) the Objective-C project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomAuthSwift) the Swift project.

- The `CustomAuth` project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The `CustomAuthObjC` and `CustomAuthSwift` projects contains a native iOS application that uses a MobileFirst native API library.
- Make sure to update the `worklight.plist` file in the native project with the relevant server settings.


iOS Simulator - iPhone 6 - iPhone 6 / iOS 8.3...
Carrier
9:23 AM
Custom Authentication
Call protected adapter procedure
Logout
iOS Simulator - iPhone 6 - iPhone 6 / iOS 8.3...
Carrier
9:24 AM
Back
Custom Authentication
user
password
Login
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return
iOS Simulator - iPhone 6 - iPhone 6 / iOS 8.3...
Carrier
9:24 AM
Custom Authentication
Call protected adapter procedure
Logout
Adapter Response
{"isSuccessful":true,"secretData":"123456"}
OK