Adapter-based authentication in native iOS applications

Overview

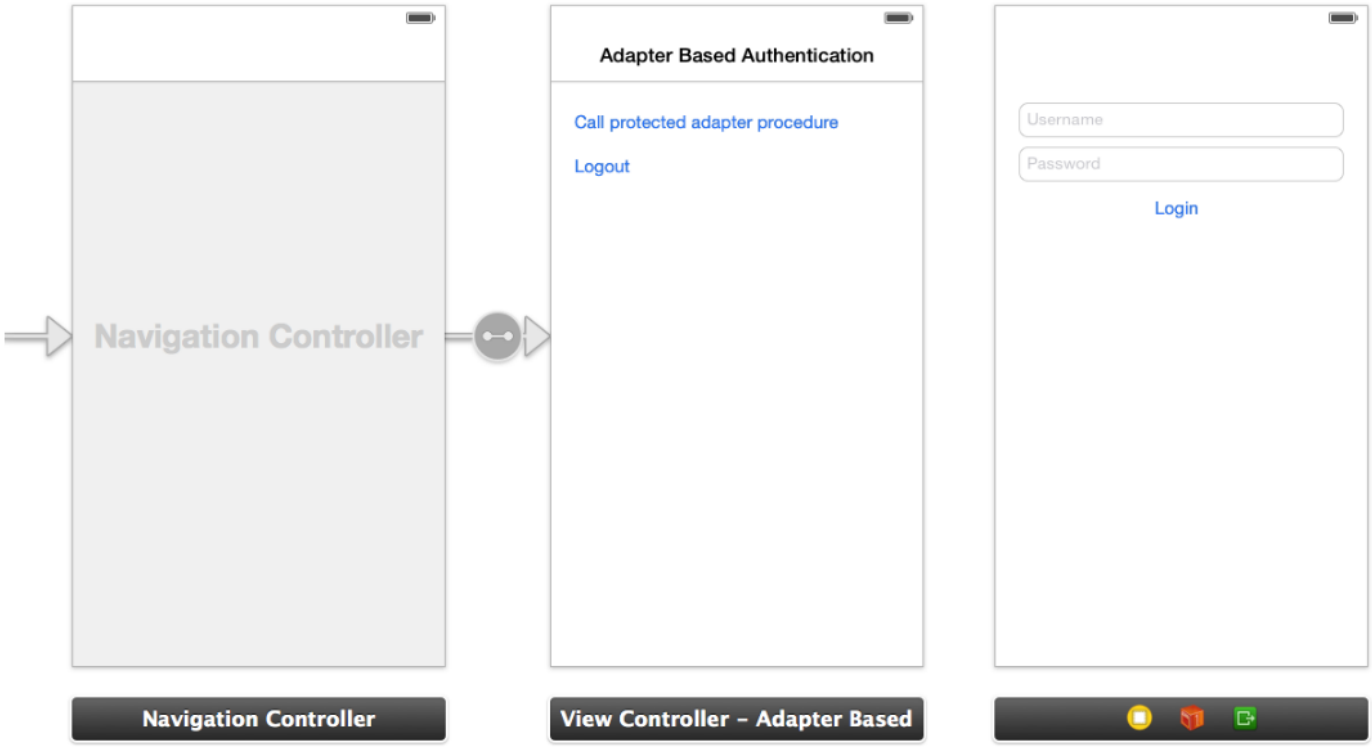
This tutorial explains how to implement the client-side of adapter-based authentication in native iOS.
Prerequisite: Make sure that you read the Adapter-based authentication (../) tutorial first.

Implementing the 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 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:@"AuthRealm"]
;
    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`.

       ```
       -(BOOL) isCustomResponse:(WLResponse *)response {
           if(response && response.responseJSON){
               if ([response.responseJSON objectForKey:@"authStatus"] != nil)
       {
                   return true;
               }
           }
           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 {
    NSString* authStatus = (NSString*) [response.responseJSON objectForKey:@"authSt
atus"];
    if([authStatus isEqual:@"complete"]){
        [self.vc.navigationController popViewControllerAnimated:YES];
        [self submitSuccess:response];
    }
    else{
        // Check if login form is already visible />
        if([self.vc.navigationController.visibleViewController isKindOfClass:
[LoginViewController class]]){>
            dispatch_async(dispatch_get_main_queue(), ^(void){
                LoginViewController*  loginController = (LoginViewController*) self.vc.navigation
Controller.visibleViewController;
                loginController.errorMsg.hidden = NO;
            });
        }
        else{
            [self.vc performSegueWithIdentifier:@"showLogin" sender:self.vc];
            dispatch_async(dispatch_get_main_queue(), ^(void){
                LoginViewController*  loginController = (LoginViewController*) self.vc.navigation
Controller.visibleViewController;
                loginController.challengeHandler = self;
                loginController.errorMsg.hidden = YES;
            });
        }
    }
}
```

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.

```
-(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 adapter procedure.

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

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] initWith
ViewController: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) {
    ...
    }];
```

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/AdapterBasedAuth) the MobileFirst project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/AdapterBasedAuthObjC) the

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

- The `AdapterBasedAuth` project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The `AdapterBasedAuthObjC` and `AdapterBasedAuthSwift` 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.