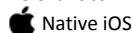


Adapter-based authentication in native iOS applications

Relevant to:



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

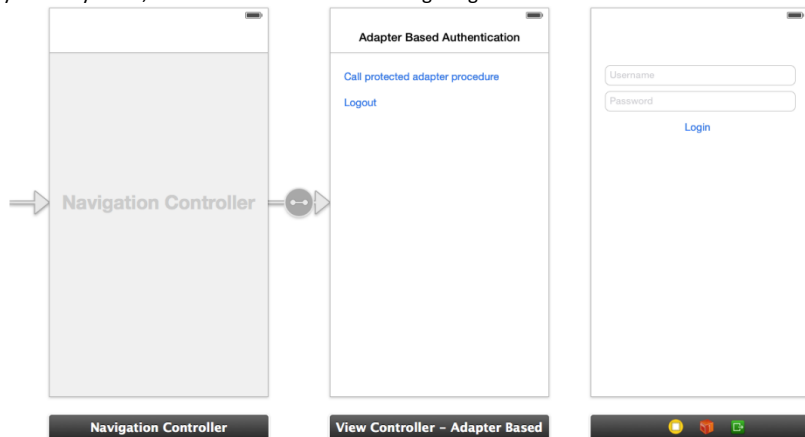
Prerequisite: Make sure that you read [Adapter-based authentication](#) first.

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](#).

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`.

```
@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 pushViewControllerAnimated:YES];
    [self submitSuccess:response];
}

-(void) onFailure:(WLFailureResponse *)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:@"AuthAdapter"
        procedureName:@"submitAuthentication"];
    myInvocationData.parameters = @[self.username.text, self.password.text];
    [self.challengeHandler submitAdapterAuthentication:myInvocationData options:nil];
}
```

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](#) the MobileFirst project.

[Click to download](#) the Objective-C project.

[Click to download](#) 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.

