

Custom Authenticator and Login Module in native iOS applications

This is a continuation of Custom Authenticator and Login Module (../).

Creating the client-side authentication components

Create a native iOS application and add the IBM MobileFirst Platform Foundation native APIs following the documentation.

In your storyboard, add a *ViewController* containing a login form.



Challenge Handler

Create a *MyChallengeHandler* class as a subclass of *ChallengeHandler*.

We will implement some of the *ChallengeHandler* methods to respond to the challenge.

```
@interface MyChallengeHandler : ChallengeHandler
@property ViewController* vc;
//A convenient way of updating the View
-(id)initWithViewController: (ViewController*) vc;
@end
```

Before calling your protected adapter, make sure to register your challenge handler using *WLClient*'s *registerChallengeHandler* method.

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

The *isCustomResponse* method of the challenge handler is invoked each time that a response is received from the 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 if auth is required
        return ([authRequired compare:@"required"] == NSOrderedSame);
    }
    return false;
}
@end
```

If *isCustomResponse* returns *true*, the framework calls the *handleChallenge* method. This function is used to perform required actions, such as hide application screen and show login screen.

@implementation MyChallengeHandler

```
//...
-(void) handleChallenge:(WLResponse *)response {
    NSLog(@"Inside handleChallenge - need to show form on the screen");
    LoginViewController* loginController = [self.vc.storyboard
    instantiateViewControllerWithIdentifier:@"LoginViewController"];
    loginController.challengeHandler = self;
    [self.vc.navigationController pushViewController:loginController animated:YES];
}
@end
```

onSuccess and *onFailure* get triggers when the authentication ends.

You need to call *submitSuccess* to inform the framework that the authentication process is over, and allow the invocation's success handler to be called.

@implementation MyChallengeHandler

```
//...
-(void) onSuccess:(WLResponse *)response {
    NSLog(@"inside challenge success");
    [self.vc.navigationController popViewControllerAnimated:YES]
;
    [self submitSuccess:response];
}
<p>-(void) onFailure:(WLFailResponse *)response {
    NSLog(@"inside challenge failure");
    [self submitFailure:response];
}
```

In your *LoginViewController*, when the user clicks to submit his credentials, you need to call *submitLoginForm* 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"];
}
```

Sample application

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/NativeCustomLoginModuleProject.zip>)
the Studio project.

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/iOSNativeCustomLoginModuleProject.zip>)
the Native project.

