

Adapter-based authentication in native iOS applications

This is a continuation of the Adapter-based authentication (../) tutorial.

Creating the client-side authentication components

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

Storyboard

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



ChallengeHandler

Create a MyChallengeHandler class as a subclass of ChallengeHandler

We will implement some of the ChallengeHandler methods to respond to the form-based 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 WLCClient's registerChallengeHandler.

```
[[WLCClient 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 {
    NSLog(@"Inside isCustomResponse");
    if(response &&& [response getResponseJson]){
        if ([[response getResponseJson] objectForKey:@"authRequired"]) {
            NSLog(@"Detected adapter auth - return true");
            NSString* authRequired = (NSString*) [[response getResponseJson] objectForKey:@"authRequired"];
        };
        return [authRequired boolValue]; //return if auth is required
    }
}
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];
}
-(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 submitAdapterAuthentication to send the credentials to the submitAuthentication procedure you wrote previously.

@implementation LoginViewController

/**

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

Sample application

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/NativeAdapterBasedAuthProject.zip>)

the Studio project.

Click to download

(<http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/iOSNativeAdapterBasedAuthProject.zip>)

the Native project.

