

Form-based authentication in native iOS applications

Overview

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

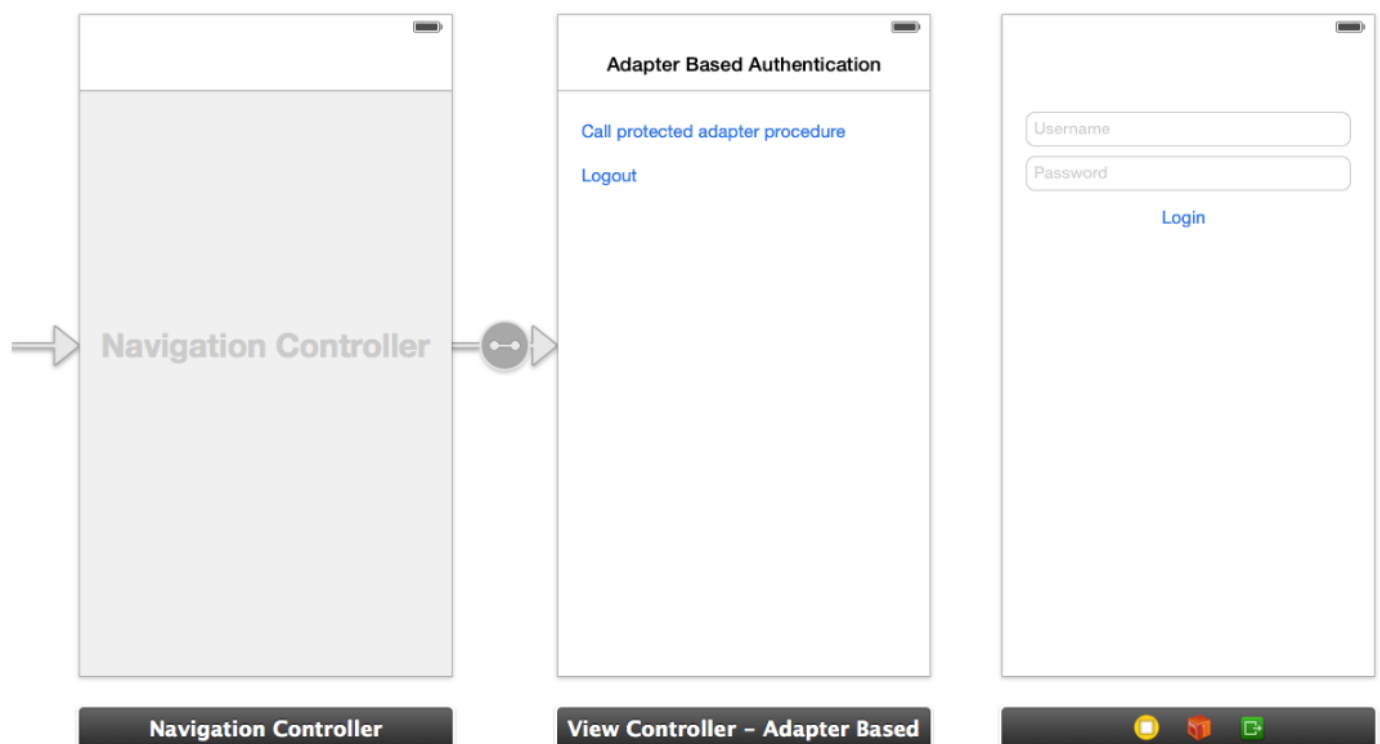
Prerequisite: Make sure that you read the Form-based authentication (../) tutorial first.

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`.

```
1 | @interface MyChallengeHandler : ChallengeHandler
```

- Call the `initWithRealm` method:

```

1  @implementation MyChallengeHandler
2  //...
3  -(id)init:{
4      self = [self initWithRealm:@"SampleAppRealm"];
5      return self;
6  }

```

- Add an implementation of the following ChallengeHandler methods to handle the form-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`.

The default login form that returns from the MobileFirst Server contains the `j_security_check` string. If the response contains the string, the challenge handler returns `true`.

```

1  -(BOOL) isCustomResponse:(WLResponse *)response {
2      if(response && response.responseText){
3          if ([response.responseText rangeOfString:@"j_security_check" options:NSCaseInsen:
4              NSLog(@"Detected j_security_check string - returns true");
5              return true;
6          }
7      }
8      return false;
9  }

```

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.

```

1  -(void) handleChallenge:(WLResponse *)response {
2      NSLog(@"A login form should appear");
3      LoginViewController* loginController = [self.vc.storyboard instantiateViewControllerWithI
4      loginController.challengeHandler = self;
5      [self.vc.navigationController pushViewController:loginController animated:YES];
6  }

```

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.

```
1  -(void) onSuccess:(WLResponse *)response {
2      NSLog(@"Challenge succeeded");
3      [self.vc.navigationController popViewControllerAnimated:YES];
4      [self submitSuccess:response];
5  }
6  -(void) onFailure:(WLFailResponse *)response {
7      NSLog(@"Challenge failed");
8      [self submitFailure:response];
9  }
```

submitLoginForm

In your login View Controller, when the user taps to submit the credentials, call the `submitLoginForm` method to send the `j_security_check` string and the credentials to the MobileFirst Server.

```
1  @implementation LoginViewController
2  //...
3  - (IBAction)login:(id)sender {
4      [self.challengeHandler submitLoginForm:@"j_security_check"
5          requestParameters:@{@"j_username": self.username.text, @"j_password": self.password.text}
6          requestHeaders:nil
7          requestTimeoutInMilliseconds:0
8          requestMethod:@"POST"];
9  }
10 @end
```

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:

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

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

```
1 | [[WLCClient sharedInstance] registerChallengeHandler:[MyChallengeHandler alloc] initWithVi
```

- Invoke the protected adapter procedure:

```
1 | NSURL* url = [NSURL URLWithString:@"~/adapters/AuthAdapter/getSecretData"];
2 | WLResourceRequest* request = [WLResourceRequest requestWithURL:url method:WLHttpMet
3 | [request sendWithCompletionHandler:^(WLResponse *response, NSError *error) {
4 | ...
5 | }];
```

Sample application

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/FormBasedAuth>) the MobileFirst project.

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

Click to download (<https://github.com/MobileFirst-Platform-Developer-Center/FormBasedAuthSwift>) the Swift project.

- The FormBasedAuth project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The FormBasedAuthObjC and FormBasedAuthSwift 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.

