

Form-based authentication in native iOS applications

Overview

This tutorial illustrates the native Android client-side authentication components for form-based authentication. Make sure you read [Form-based authentication \(../\)](#) first.

Creating the client-side authentication components

Create a native iOS application and add the MobileFirst native APIs following the documentation. In your storyboard, add a ViewController containing a login form.



Create a `MyChallengeHandler` class as a subclass of `ChallengeHandler`

We will implement some of the `ChallengeHandler` methods to respond to the form-based challenge.

```

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

```

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`.

The default login form that is returned from the MobileFirst server contains the `j_security_check` string. If the challenge handler detects it in the response, return `true`.

```

1  @implementation MyChallengeHandler
2  //...
3  -(BOOL) isCustomResponse:(WLResponse *)response {
4      if(response &&& response.responseText){
5          if ([response.responseText rangeOfString:@"_security_check" options:NSCaseInsensitiveSearch].location != NSNotFound)
6              NSLog(@"Detected login form - return true");
7          return true;
8      }
9  }
10 return false;
11 }
12 @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.

```

1  @implementation MyChallengeHandler
2  //...
3  -(void) handleChallenge:(WLResponse *)response {
4      NSLog(@"Inside handleChallenge - need to show form on the screen");
5      LoginViewController* loginController = [self.vc.storyboard instantiateViewControllerWithIdentifier:@"LoginViewController"];
6      loginController.challengeHandler = self;
7      [self.vc.navigationController pushViewController:loginController animated:YES];
8  }
9  @end

```

`onSuccess` and `>onFailure` get triggered 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.

```

1  @implementation MyChallengeHandler
2  //...
3  -(void) onSuccess:(WLResponse *)response {
4      NSLog(@"inside challenge success");
5      [self.vc.navigationController popViewControllerAnimated:YES];
6      [self submitSuccess:response];
7  }
8  -(void) onFailure:(WLFailResponse *)response {
9      NSLog(@"inside challenge failure");
10     [self submitFailure:response];
11 }
12 @end

```

In your `LoginViewController`, when the user clicks to submit his credentials, you need to call `submitLoginForm` to send 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

```

Sample application

Click to download

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

Click to download

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

