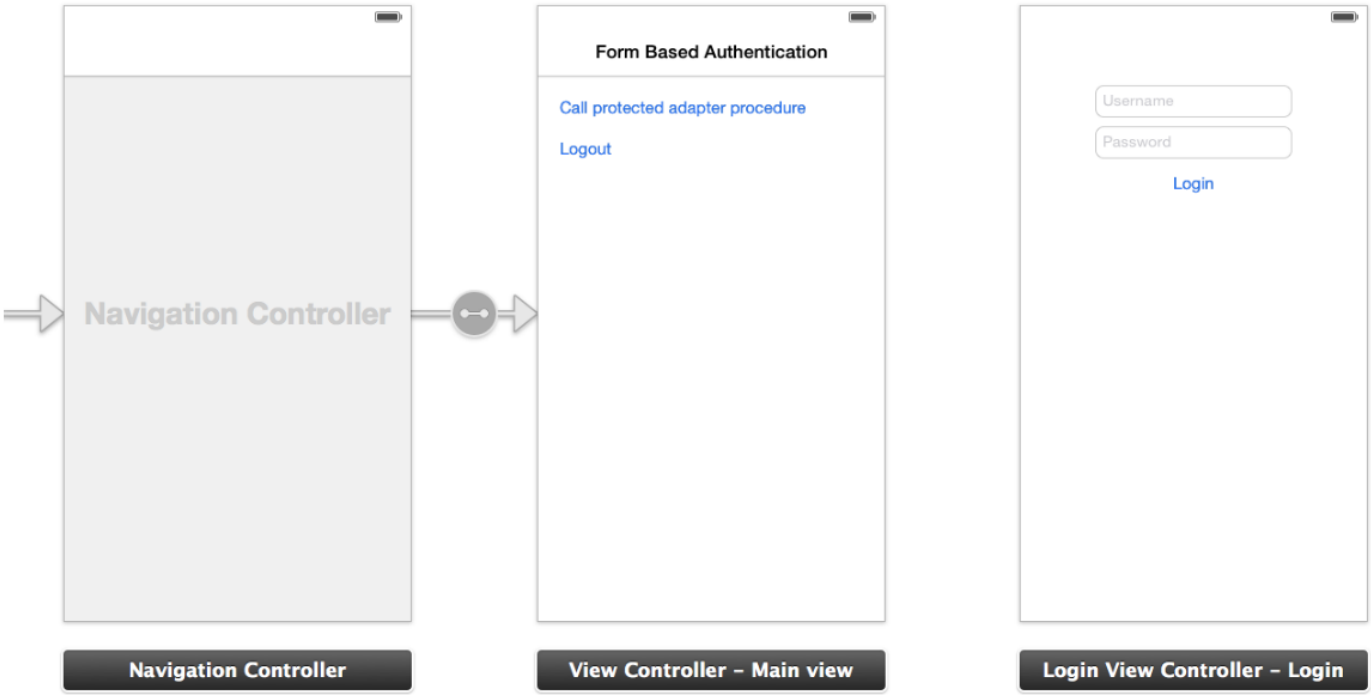
Form-based authentication in native iOS applications

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.

```
@implementation MyChallengeHandler
//...
-(BOOL) isCustomResponse:(WLResponse *)response {
    if(response &amp;amp;&amp;amp; response.responseText){
        if ([response.responseText rangeOfString:@"j_security_check" options:NSCaseInsensitiveSearch].location != NSN
            NSLog(@"Detected login form - return true");
            return true;
        }
    }
    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:@"LoginViewContro
    loginController.challengeHandler = self;
    [self.vc.navigationController pushViewController:loginController animated:YES];<br />
}
@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.

```
@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];
}
@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.

```
@implementation LoginViewController
//***
- (IBAction)login:(id)sender {
    [self.challengeHandler submitLoginForm:@"j_security_check"
        requestParameters:@{@"j_username": self.username.text, @"j_password": self.password.text}
        requestHeaders:nil
        requestTimeoutInMilliSeconds:0
        requestMethod:@"POST"];
}
@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.