User interface design

for iOS, using Objective-C

# Lab 2 – Asynchronously connecting to data

## Functional Goals

Take a functional iOS app that contains the necessary controls to display bandwidth usage data from Rose-Hulman’s network usage tool and add asynchronous data connections.

## Learning Goals

* Understand iOS paradigms, including the keychain and network calls.

## Prerequisites

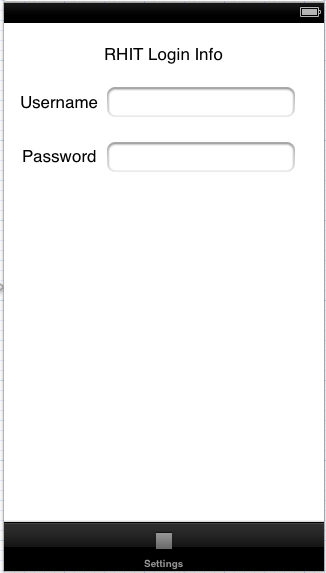
* You’ll need to install Xcode (Version 4.5.2 was used to create this lab) from the App Store
* A basic understanding of Xcode and Objective C development
* Code from Lab1 (a complete version may be available from your instructor)

## Submission Instructions

Submit answers to the **3** (or **4**, with extra credit)questions in this lab as a .pdf to the appropriate Moodle submission form.

## Get started: Make a Settings page

1. This is going to be a slight test on what you learned in the previous lab. Use the interface builder and your Second View look like this;



1. Go into SecondViewController.h and create IBOutlets for your text fields. Make sure you synthesize them in the .m file.
2. When a user sets these, we want to make sure that they are stored securely and retrieved next time the user visits the page.
3. Copy the KeychainItemWrapper.h and .m files into your project (make sure you add them to the target and copy them into the destination folder). These are slightly modified versions of the example files provided by Apple. For more information on completing this step yourself, you can see the developer article located at <http://developer.apple.com/library/ios/#documentation/Security/Conceptual/keychainServConcepts/02concepts/concepts.html>
4. Go back into your SecondViewController and make a property for the KeychainItemWrapper as follows: (note you’ll have to import KeychainItemWrapper.h)

@property (strong, nonatomic) KeychainItemWrapper \*passwordItem;

1. Make sure to synthesize it in the .m file.
2. Change your SecondViewController.m to look like the following code

- (void) viewDidAppear:(BOOL)animated

{

[super viewDidAppear:animated];

[[UIApplication sharedApplication] setNetworkActivityIndicatorVisible:NO];

}

-(void)findAndResignFirstResponder{

for (UIView \*aView in [self.view subviews]){

if ([aView isFirstResponder] ) {

[aView resignFirstResponder];

}

}

}

-(void)touchesBegan:(NSSet \*)touches withEvent:(UIEvent \*)event{

for (UITouch \*touch in touches){

if (touch.view == self.view){

[self findAndResignFirstResponder];

}

}

}

- (IBAction)textFieldDidEndEditing:(UITextField \*)textField

{

[textField resignFirstResponder];

}

#pragma mark - View lifecycle

- (void)viewWillAppear:(BOOL)animated

{

[usernameTextField setText:[self getUserName]];

[passwordTextField setText:[self getPassword]];

}

- (void)viewWillDisappear:(BOOL)animated

{

// Store username to keychain

if ([usernameTextField text])

{

[self setUserName:[usernameTextField text]];

}

// Store password to keychain

if ([passwordTextField text])

{

[self setPassword:[passwordTextField text]];

}

}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation

{

// Return YES for supported orientations

return (interfaceOrientation == UIInterfaceOrientationPortrait);

}

- (NSString \*) getUserName

{

[self loadPasswordItem];

return [passwordItem objectForKey:(\_\_bridge id)kSecAttrAccount];

}

- (NSString \*) getPassword

{

[self loadPasswordItem];

return [passwordItem objectForKey:(\_\_bridge id)kSecValueData];

}

- (void) setUserName:(NSString \*)username

{

[self loadPasswordItem];

[passwordItem setObject:username forKey:(\_\_bridge id)kSecAttrAccount];

}

- (void) setPassword:(NSString \*)password

{

[self loadPasswordItem];

[passwordItem setObject:password forKey:(\_\_bridge id)kSecValueData];

}

- (void) loadPasswordItem

{

if(passwordItem == nil)

{

passwordItem = [[KeychainItemWrapper alloc] initWithIdentifier:@"Password" accessGroup:@"\*.edu.rose-hulman.ScheduleLookup"];

}

}

Question 1: Why do we make the passwordItem property strong? (3 points)

Question 2: This will currently only work to store passwords in the keychain on the simulator, not a physical device. What changes would you have to make to have it work on both? The link provided above about the keychain will be invaluable. (9 points)

## Storing the Bandwidth Information

We want a single object that we can store all the bandwidth information in and pass around from the eventual scraper to the view. Make a new file called BandwidthUsageRecord.h/.m. In this file we want to simply have getters and setters for several properties, so all we really need to do is declare the properties and synthesize them in the .m file. Make your two files look like the following:

/////////////BandwidthUsageRecord.h

@interface BandwidthUsageRecord : NSObject {

}

@property (nonatomic, strong) NSString \* kerberosName;

@property (nonatomic, strong) NSDate \* timestamp;

@property (nonatomic, strong) NSNumber \* policyReceived;

@property (nonatomic, strong) NSNumber \* policySent;

@property (nonatomic, strong) NSNumber \* actualReceived;

@property (nonatomic, strong) NSNumber \* actualSent;

@property (nonatomic, strong) NSString \* bandwidthClass;

@end

/////////////BandwidthUsageRecord.m

#import "BandwidthUsageRecord.h"

NSInteger dateIdentifier(NSInteger year, NSInteger month, NSInteger day) {

return 1000000 \* year + 1000 \* month + day;

}

@implementation BandwidthUsageRecord

@synthesize kerberosName;

@synthesize timestamp;

@synthesize policyReceived;

@synthesize policySent;

@synthesize actualReceived;

@synthesize actualSent;

@synthesize bandwidthClass;

@end

This should be fairly self-explanatory to you.

Question 3: Expalin the dateIdentfier method. Can it ever give a non-unique date? Why would you ever use this instead of a simple NSDate? (3 points)

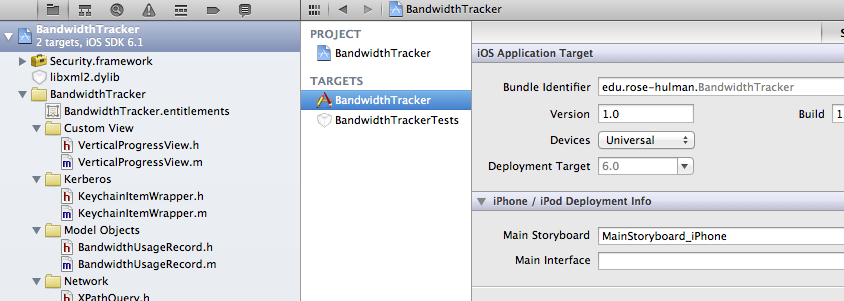
We need one of these usage records in our FirstViewController. In the .h file write:

@property (nonatomic, strong) BandwidthUsageRecord \*record;

And synthesize it in the .m file.

## Scraping the Network

1. You have been provided with 5 files – XpathQuery.m/.h, BandwidthScraper.h/.m, and BandwidthScraperDelegate.h. Copy all 5 into your project now. These are the files that scrape the IAIT network page using Xpath, an XML query language.
2. You will need to import the libxml2.dylib to make the code work. Click on the BandwidthTracker item to pull up the project settings. Select the BandwidthTracker target.



1. Under Linked Frameworks and Libraries, select the “plus” icon to add a new library. Simply type in libxml2 and select the dylib that appears. Just below that, under “Entitlements”, select the check box to Use Entitlements.
2. Take a look at BandwidthScraperDelegate.h. You should know all about delegates through your previous dealings with Objective-c. Essentially an object implements all of the methods in this protocol (think like a java interface) then you set that object of the delegate of a bandwidth scraper. When the scraper wants to send information back, it simply calls the agreed-upon methods defined in the delegate.
3. We want our FirstViewController to be the delegate, so we change the .h file like this:
4. Now we have to go into our FirstViewController and implement some or all of the methods in our protocol. The main one we care about is when we find bandwidth usage amounts, so add the method:

@interface FirstViewController : UIViewController <BandwidthScraperDelegate>

1. Note we are refreshing the view and setting our record to the record provided by the scraper. So we should probably now update our dummy methods from Lab 1 to reflect what is stored in the usage record!

- (CGFloat) getCurrentPolicyDownloadValue {

if (record != nil) {

return [[record policyReceived] floatValue];

}

NSLog(@"record was nil...");

return 0;

}

- (CGFloat) getCurrentActualDownloadValue {

if (record != nil) {

return [[record actualReceived] floatValue];

}

return 0;

}

- (CGFloat) getCurrentPolicyUploadValue {

if (record != nil) {

return [[record policySent] floatValue];

}

return 0;

}

- (void)scraper:(BandwidthScraper \*)scraper foundBandwidthUsageAmounts:(BandwidthUsageRecord \*)usage {

record = usage;

[self updateViews:nil];

[UIApplication sharedApplication].networkActivityIndicatorVisible = NO;

}

Note: You are left to change the getCurrentActualUploadValue method yourself. It shouldn’t be too hard.

## But wait – we aren’t scraping yet!

Oh yes, we need to actually fire off the call to scrape. In the refreshData method, change it to the following. You will probably need to handle some imports to make them work.

- (IBAction) refreshData:(id)sender {

NSLog(@"refreshing data");

[UIApplication sharedApplication].networkActivityIndicatorVisible = YES;

[[[BandwidthScraper alloc] initWithDelegate:self] beginScraping];

}

## Congratulations

You’re done; run your app and check it out! Don’t forget: submit answers to the **3** (or **4**, with extra credit)questions in this lab as a .pdf to the appropriate Moodle submission form.

## Above and beyond

From here on is extra credit. It is possible to earn full credit on the lab without doing this question.

For extra credit – go ahead and implement the scraper delegate method that is fired off when an error occurs. Pop up an NSAlert that says something has gone wrong. Worth 6 points.