# **Building a sample iOS application with OrderCloud**

[OrderCloudSwifty](http://github.com/Four51/ordercloudswifty) allows developers to integrate [OrderCloud](http://ordercloud.io) in their Swift applications.

This short tutorial provides a step-by-step guide to build a very basic iOS application. It is assumed that you have Xcode 7.2+ installed alongside with [Cocoapods](http://cocoapods.org).

## **Project setup**

In order to focus on the essential, we’ll start with a pre-created project and fill in the blanks related to OrderCloud. Open a terminal and clone the tutorial repository:

git clone -b tutorial https://github.com/ordercloud-api/ordercloud-ios-demo

Go into that directory, install CocoaPods dependencies and open OrderCloud-ios.xcworkspace in Xcode:

cd ordercloud-ios-demo  
pod install  
open OrderCloud-ios.xcworkspace

You will see that we used the excellent [SWRevealView](http://github.com/) to handle the side menu of our application.

## **Pre-coding necessities**

Before diving into the code, follow [this guide] to create the necessary data in the Api. Create an application in your dashboard and copy the client ID.

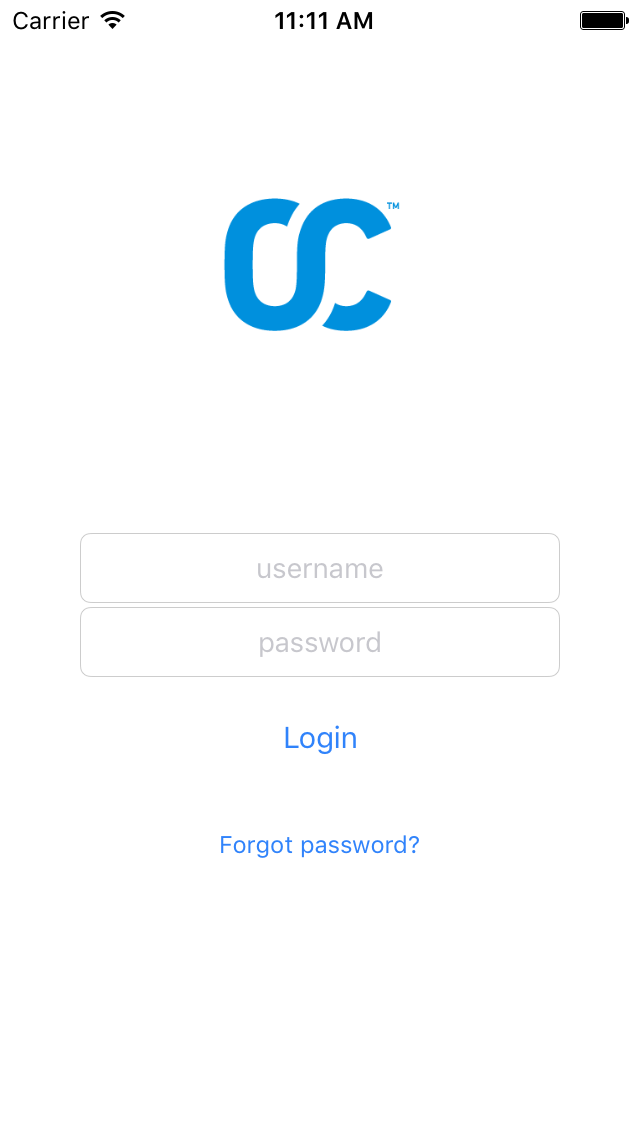
## **Setting up client ID**

In Xcode file explorer, open AppDelegate.swift and fill in your client ID:

OrderCloud.setupClientId("CLIENT\_ID")

## **Login**

Then we will continue with logging in as a user.



* Click on Main.storyboard on the project explorer on the left. Click on the login view. Now, on the top right of the Xcode window, click on the interlaced circles. This will create a slit view. The file opened on the right is now LoginViewController.swift.
* Now, create an outlet function to be called when the login button is called:

@IBAction func login(sender:AnyObject) {  
  
}

* Connect the Login button to the action:
  + Next, let’s write the code inside our function.

@IBAction func login(sender:AnyObject) {  
 // When the login button is clicked, we first   
 // start animating the activity indicator to  
 // indicate to the user that something is happening  
 activityIndicator.startAnimating()  
  
 // Next we call OrderCloud Authentication  
 // We pass a pre-built default handler and  
 // we merely have to indicate what functions  
 // to call. When the response is received, we  
 // notify the user by stopping the animation  
 // of the activity indicator.   
 OrderCloud.Auth.authenticate(self.usernameInput.text!, password:self.passwordInput.text!, completionHandler: OrderCloud.Auth.defaultHandler(  
 successfulLogin,  
 jsonErrors: apiError,  
 requestFailure: networkError,  
 responseReceived: {   
 self.activityIndicator.stopAnimating()   
 }))

* As you may have guessed, we now have to define three functions to handle our responses. First, we’ll handle the case of a success login. If the login is successful, we merely have to redirect the user to the home page since the Auth.defaultHandler takes care of getting the access\_token and de-serializing the JSON into the OrderCloud.Auth.currentUser object.

func successfulLogin() -> () {  
 let storyboard: UIStoryboard = UIStoryboard(name: "Main", bundle: nil)  
 let initViewController: UIViewController = storyboard.instantiateViewControllerWithIdentifier("Home") as UIViewController  
 self.presentViewController(initViewController, animated: true, completion: nil)  
}

* Then we’ll define the function to handle errors from the API. We merely iterate over the error messages and show an alert:

func apiError(err: [OrderCloud.ApiError]) {  
 var errorMsg = "Unknown error"  
 var errorTitle = "Login error"  
 if (err.count > 0) {  
 let firstErr: OrderCloud.ApiError = err.first!  
 errorMsg = firstErr.message()  
 errorTitle = firstErr.code()  
 }  
 let loginAlert = UIAlertController(title: errorTitle, message: errorMsg, preferredStyle: UIAlertControllerStyle.Alert)  
 loginAlert.addAction(UIAlertAction(title: "Ok", style: .Default, handler: { (action: UIAlertAction!) in  
 }))  
  
 self.presentViewController(loginAlert, animated: true, completion: { self.passwordInput.text = String() })  
}

* Finally, we take care of network errors:

func networkError(err: NSError) {  
 let failureAlert = UIAlertController(title: "Network error", message: String(err.localizedDescription), preferredStyle: UIAlertControllerStyle.Alert)  
 failureAlert.addAction(UIAlertAction(title: "Ok", style: .Default, handler: { (action: UIAlertAction!) in }))  
 self.presentViewController(failureAlert, animated: true, completion: { self.passwordInput.text = String() })  
}

* Build and run the application (CMD+R) and login with the credentials you defined in the API console. You should be able to login and see “Welcome to OrderCloud”



### **UX Improvement**

In order to improve the user experience, add the following function to the LoginViewController class:

func textFieldShouldReturn(textField: UITextField) -> Bool {  
 if(textField == self.passwordInput)  
 {  
 textField.resignFirstResponder()  
 login(textField)  
 return false  
 }  
 textField.nextField?.becomeFirstResponder()  
 return true  
}

Now you can merely hit the return button on the iOS keyboard to submit the login.

## **Listing products**

In our example, we will do a simple fetching of a list of products.

We created a UITableViewController class in the Product.storyboard and we implemented the ProductViewController class. The behavior we expect from this view is to load all products available when it loads. In our case, we merely have to add one line to our viewDidLoad function:

OC.Products.list(completionHandler: OrderCloud.ApiObject.completionHandler(loadProducts, jsonErrors: apiError, requestFailure: networkError))

As previously seen, we here call for apiError and networkError and we have a new function called loadProducts that will merely update the data and call our class to reload its data:

func loadProducts(json: OrderCloud.Json) {  
 \_products = ApiObjectList<OC.Product>(fromJson: json)  
 self.tableView.reloadData()  
}

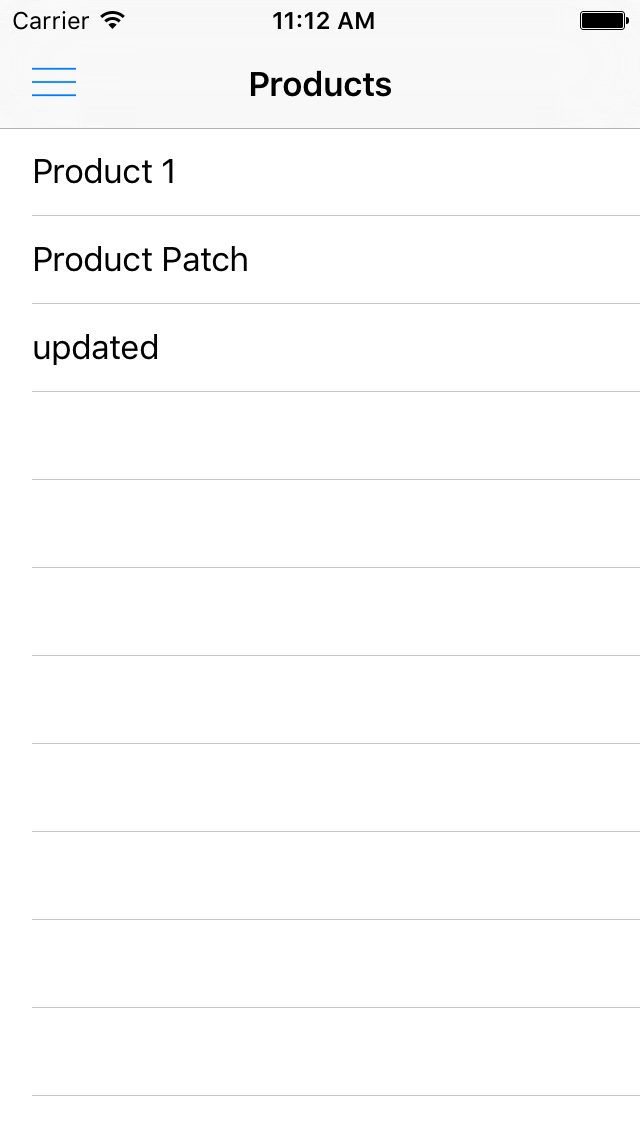
In order for the TableView to understand our data format, we override the tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell function. It’s as simple as that:

override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {  
  
 let cell = UITableViewCell()  
  
 if let prod: OC.Product = \_products.Items[indexPath.row] as OC.Product {  
 cell.textLabel!.text = prod.Name  
 } else {  
 cell.textLabel!.text = "Product?"  
 }  
  
 return cell  
}

We add one last detail to ensure the proper displaying of our products:

override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {  
 return \_products.Meta.TotalCount  
}

Now build and run again, you should be able to see your products in your view:



## **To go further**

If you want to expand the iOS application, here are suggestions:

* Complete the “Forgotten password” flow by filling in the changePassword function in AccountPasswordChangeViewController.swift

-