ServiceNow iOS App

# Location

The code for the app is located on the inergex-dev GitHub page: <https://github.com/inergex-dev/ServiceNowApp>.

# Overview

The purpose of this app is to offer a condensed, mobile alternative to the service-now website. The app provides support for creating, editing, and viewing tickets.

This app supports iPhone and iPad.

The project is built using xCode 4.6.2, using storyboards, with the project primarily using ARC (Automatic Reference Counting). Some classes don’t use ARC, and are indicated by going to the project’s target > Build Phases > Compile Sources and adding the *-fno-objc-arc* compiler flag.

# Forms

There are 3 forms: Login, Create Ticket, and Update Ticket. Each of these have certain commonalities:

* When a text-box is clicked out of, the on-screen keyboard should disappear.
* When next/go is clicked, the next text field is clicked or a submit action is performed.
* In cases where a drop-down menu would normally be used, the *PickerController* class is used.
* The tag is used as a way to differentiate between different UI elements

In the cases of Create Ticket and Update ticket, these are dynamically created tables, and thus are created in code. This is to offer as much flexibly as possible for adding fields, as well as potentially having custom fields for specific businesses.

# Classes

The naming convention for classes that are used by UI elements on the storyboard is “NameType” (example: LoginView**Controller**). TVC is an abbreviation for “TableViewController”, so as to shorten class names.

## Support Classes

### Utility

*Utility* is a static class, which holds variables / methods that are used across multiple classes, to avoid repetition. This includes functions for

* converting a severity/impact/state integer value to the corresponding string
* getting the username/password of the current user for sending requests
* displaying/dismissing a loading alert window

### PickerController

This class is responsible for handling how the user would choose an option from a range of set values. When the user indicates they wish to choose a value, a segue should be performed with this object, passing along an array of choices, as well as a pointer to a “*SelectedRow*” object. Once the user chooses an option and returns, the class will update the value of the *SelectedRow*, so that the form can access the new value.

### Ticket

The ticket class holds all ticket/incident data, for easy transportation between ViewControllers.

### SOAPRequest

Sends SOAP requests which take a method name and a list of comma separated *SOAPRequestVariable*s, and sends the SOAP request to service-now. The result is returned through the use of delegate methods, after being converted into a *TBXMLElement* (or an *NSError*). The delegate method should ideally be the caller of the *SOAPRequest*.

### PullToRefresh

An open-source set of files that enable functionality for pulling down a UITableView and refreshing the content of the table, as well as indicating the table is refreshing.

### TBXML

An open-source set of files, that enables a tree style XML parser; much more manageable than the default objective-c XML parser available at the time.

## Core Classes

### LoginViewController

This is where the program starts. Once the user successfully logs in (Using *SOAPRequest*), their username and password are stored in the *Utility* class, so that they can be retrieved for SOAP calls. If the user wishes to have the device remember their credential, they are also stored in the **standardUserDefaults**.

### MainMenuTVC

This is a static table view that acts as the top level of navigation, with 4 options:

* **Create Ticket** - Performs a segue to *CreateTicketTVC*
* **View Open Tickets** - Performs a segue to *OpenTicketsTVC*
* **View Closed Tickets** - Performs a segue to *ClosedTicketsTVC*
* **Sign Out** - Clears stored passwords, and returns to *LoginViewController*

### CreateTicketTVC

A dynamic table view form, which creates all fields in code; it uses *SOAPRequest* as well as *PickerController*. Once all fields are filled in and valid, a ticket request is sent, and if it returns positive, the user is sent back to *MainMenuTVC*, with an alert window informing them the ticket was successfully sent.

### OpenTicketsTVC

This dynamic table view shows a table of all the user’s open incidents. The user may pull down on the view to refresh the table (Using *PullToRefresh*). The view refreshes every time it appears. Once a row is selected, a segue brings the user to *ViewOpenTicketTVC*, passing along the selected ticket information.

### ViewOpenTicketTVC

This dynamic table view displays all of the information pertaining to the open status of a ticket. It is also possible to edit/update a ticket, but hitting the soft “Update” key on the view’s navigation bar, which will segue the view to *EditTicketTVC*, passing along the current ticket.

### EditTicketTVC

A dynamic table view form, which creates all fields in code; it uses *SOAPRequest* as well as *PickerController*. Once a change has been made (so as to avoid sending a pointless update) and all fields are filled in and valid, an edit request is sent, and if it returns positive, the user is sent back to viewing the ticket (*ViewOpenTicketTVC*), with an alert window informing them the ticket was successfully sent.

### ClosedTicketsTVC

This dynamic table view shows a table of all the user’s closed incidents. The user may pull down on the view to refresh the table (Using *PullToRefresh*). The view refreshes every time it appears. Once a row is selected, a segue brings the user to *ViewClosedTicketTVC*, passing along the selected ticket information.

### ViewClosedTicketTVC

This dynamic table view displays all of the information pertaining to the closed status of a ticket.

### AppDelegate

This class is a default iOS class, and handles different actions on launch. This app uses it to register for push notifications, as well as handle notifications when the app is active.

# Pointing app from the test web service to the actual web service

This can be done by opening the project in Xcode, opening “SOAPRequest.m”, and then changing **WEB\_SERVICE\_URL** constant to point to the new location of the web service.

# Push Notifications

An in-depth article about push notifications and how to create them for an app can be found on this article: <http://www.raywenderlich.com/32960/apple-push-notification-services-in-ios-6-tutorial-part-1>

As well as this one: <https://parse.com/tutorials/ios-push-notifications>

First, you need a certSigningRequest file, which can be gotten from the keychain access. After you have the certificate request, you need to make an app ID on the [the apple developer member center](https://developer.apple.com/) (make sure the ID has push notifications enabled). After the app ID is created, a developer certificate must be created, using the certificate request made earlier (not that for publishing the app, a production certificate needs to be made the same way). Once the certificate is generated, download it and install it into the keychain access. After it’s installed, it must be exported as a .p12 file. Next a provisioning profile must be created from the apple developer member center and then installed.

Now everything necessary should be installed. Now the app must be modified by setting the provisioning profile and app ID under build settings. Code for registering push notifications is in the file AppDelegate.m \*(see below).

Once a user enables push notifications from their app, a token is created, which is specific to that device. This token must be stored in a database, as it is used to identify the user. When you send a push notification (automatically when a ticket is updated [not done yet, as no middleware is available]), you must send the request to an apple service along with the token, and from there apple will send the push notification to the user’s phone.

Notifications are taken care of by the iOS Notification Center. If the app is in use the app must handle it (currently does nothing, since the app is small).

\* Currently, nothing is done after an app registers for tokens, as there is nowhere to send the token. The token is passed to the method [application:didRegisterForRemoteNotificationsWithDeviceToken:] in AppDelegate.m, and is currently just printed to the command line. For testing purposed, it can be copied and used to send a push notification manually.

# Releasing the App (Generating the .ipa file)

Note: If push notifications are enabled, a certificate should already be created, allowing most of this to be skipped.

A good discussion / Q&A can be found here (along with many helpful links): <http://stackoverflow.com/questions/5265292/xcode-4-create-ipa-file-instead-of-xcarchive>

Some simple instructions: <http://appfurnace.com/2012/11/publishing-your-apps-how-to-create-a-p12-file/>

First, you need a private key pair with a certificate in the Mac’s Keychain access (some common problem outlined [here](http://webbuilders.wordpress.com/2009/12/25/code-sign-errors-profile-doesnt-match-any-valid-certificateprivate-key-pair-in-the-default-keychain/)). You first create a certSigningRequest from your keychain access, and you get a certificate from [the apple developer member center](https://developer.apple.com/).

Once a certificate/private key pair are in your keychain access, generating an ipa file should be as easy as going to the top left of xcode and choosing the debug as an “iOS Device” (default is iPhone/iPad simulator), then going to “Product > Archive” on the menu bar.

With the ipa file generated, you can test it on your device by installing it through itunes.

# What’s left?

Do to the current state of the webservice, the following have not been finished:

* Comments don't display
  + This shouldn’t be hard to add when the webservice implements it. The method that would parse this input is **returnedSOAPResult** which is located in both the *OpenedTicketsTVC* and *ClosedTicketsTVC* classes. There is some commented-out code near the bottom of the method that should work without any trouble (in theory).
* The values for severity/impact/state are still hard-coded in arrays on the app, rather than read in from service-now. These arrays are located in *Utility*.