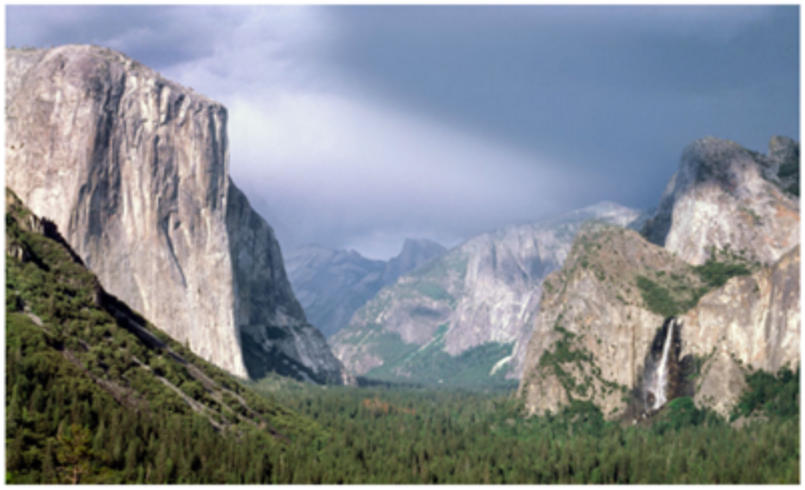
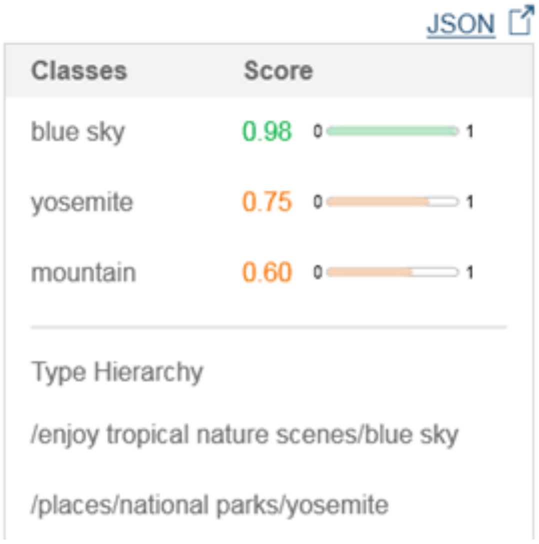
Lab: create a Watson image recognition app with Swift

Overview

One of the most impressive Watson cognitive analytic services is Visual Recognition. In this lab, you'll use that service to consume an image URL, identify the image, and return a confidence score for relevant classifiers representing things such as objects, events, and settings.

For example, the Visual Recognition service recognizes the blue sky and mountain in this image and suggests that this landscape is from Yosemite National Park in the United States.





Classes	Score
blue sky	0.98
yosemite	0.75
mountain	0.60

Type Hierarchy

/enjoy tropical nature scenes/blue sky

/places/national parks/yosemite

To create the application in this lab, follow these main steps:

1. Create a simple iOS application in Swift.
2. Instantiate the Watson SDK for iOS.
3. Create a Watson service in Bluemix and get the key token for it.
4. Add some lines of code in the Swift iOS application to call the Watson service.

You can see the code for the GUI for this lab in GitHub: https://github.com/blumareks/Swift-Watson-Guis.

Prerequisites

You need the following software:

- Mac OS X El Capitan
- Xcode 7.3 or later
- Swift 2.2.x
- Carthage (package manager similar to Ant): https://github.com/Carthage/Carthage#installing-carthage

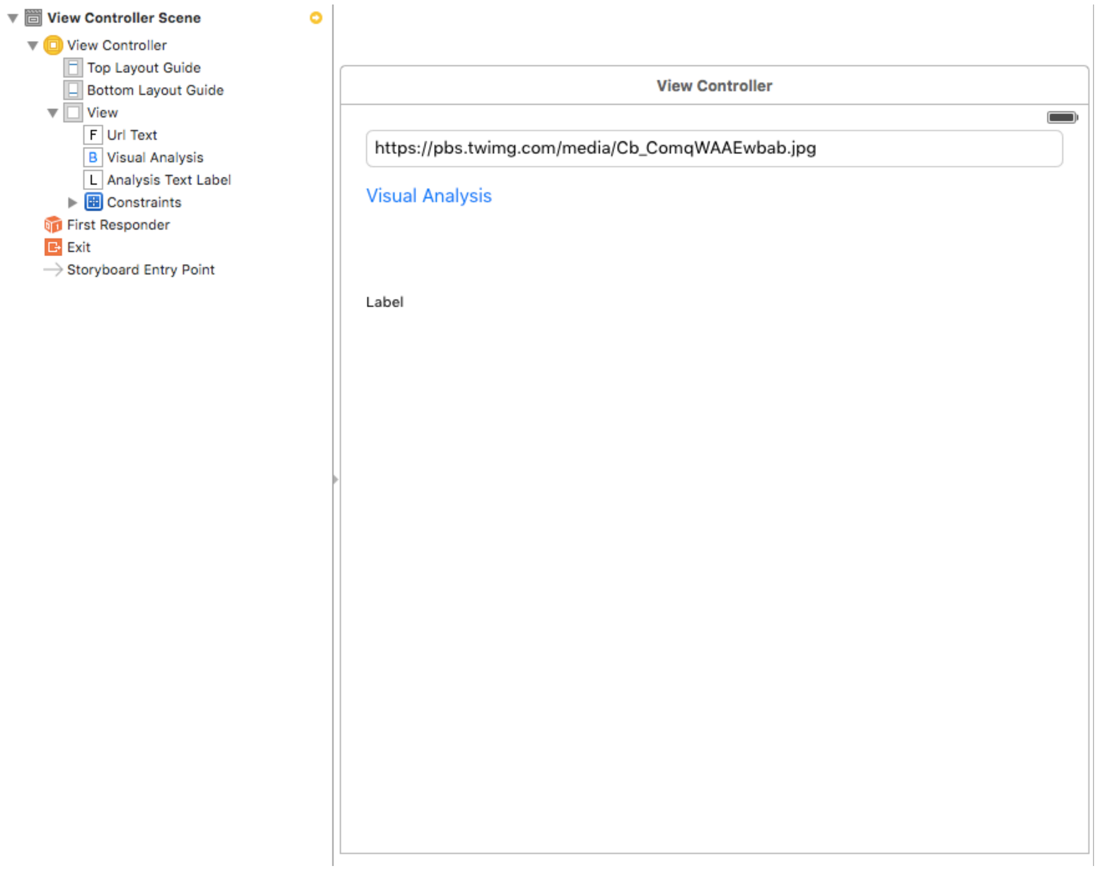
Complete the previous lab “Create a Watson sentiment analysis app with Swift.”

Step 1. Create a typical iOS application in Swift

You'll build a simple application with a **Submit** button, an editable text field, and an output label field.

Step 1a. Create the GUI

You need to create a simple single view application with a graphical user interface (GUI) that includes a text field, a label, and a button. When the user presses the button, the URL address in the text field is sent to Watson, which analyzes it and returns an opinion that is shown in the output field.



See the Xcode documentation for information about how to create basic GUIs in Xcode.

Step 1b. Connect the GUI in Main.storyboard to the code in the ViewController file

In Main.storyboard, connect the TextField:

1. Double-click **TextField** (by pressing the mouse pad with two fingers), which contains the URL. Enter this example text:

   `https://pbs.twimg.com/media/Cb_ComqWAAEwbab.jpg`
2. Select **New Reference Outlet** from the list.
3. Enter `urlText` in the dialog.
4. Insert the new reference between **class ViewController** and **override func viewDidLoad.**
5. Click **Connect**.

   The inserted text is `@IBOutlet weak var urlText: UITextField!`
6. Connect the button from the Event `Touch Up Inside` below the function `didReceiveMemoryWarning.`
7. Enter this text in the dialog: `analysisButtonPressed`

   The inserted text is:
   ```
   @IBAction func analysisButtonPressed(sender: AnyObject) {
   }
   ```
8. Connect the text label.

   The result is `@IBOutlet weak var analysisTextLabel: UILabel!`
9. Test the code by adding `NSLog("url: "+urlText.text!)` to the end of the `analysisButtonPressed` method.

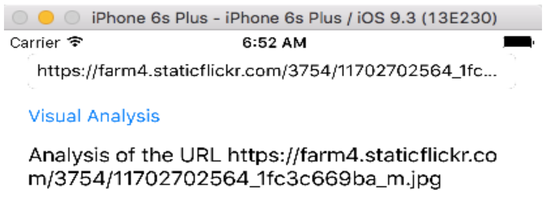
   NSLog allows you to log your actions. Finally add: `analysisTextLabel.text = urlText.text!`

   Setting `analysisTextLabel.text` with `urlText.text` allows you to show the output from the Text field in the Label field in the UI.

```
import UIKit


class ViewController: UIViewController {
    @IBOutlet weak var urlText: UITextField!
    @IBOutlet weak var analysisTextLabel: UILabel!


    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
typically from a nib.
```

```
        }

        override func didReceiveMemoryWarning() {
            super.didReceiveMemoryWarning()
            // Dispose of any resources that can be recreated.
        }

        @IBAction func analysisButtonPressed(sender: AnyObject) {
            NSLog("button pressed")
            NSLog("url: "+urlText.text!)
            analysisTextLabel.text = urlText.text!

            //call service

            //get output

        }
    }
```

10. Build and execute your application



iPhone 6s Plus - iPhone 6s Plus / iOS 9.3 (13E230)

Carrier 6:52 AM

https://farm4.staticflickr.com/3754/11702702564_1fc...

Visual Analysis

Analysis of the URL https://farm4.staticflickr.com/3754/11702702564_1fc3c669ba_m.jpg

Step 2. Install Carthage and add the Watson SDK to your project

Because you will use a separate project for this application, you must again install the Carthage dependency manager and the SDK, and then add that SDK to your Xcode project.

Step 2a. Install the Carthage dependency manager

Use the Carthage dependency manager to install libraries that are used by your application. You can install Carthage from GitHub or use Homebrew to update it only if you use Xcode 7.x.

Important: If you previously installed the binary version of Carthage, you must delete /Library/Frameworks/CarthageKit.framework.

- To install Carthage from GitHub, download and run the latest release of the Carthage.pkg file: https://github.com/Carthage/Carthage/releases. Then follow the installation prompts.
- To install Carthage by using Homebrew (only on Xcode 7.x), run the commands `brew update` and `brew install Carthage`.

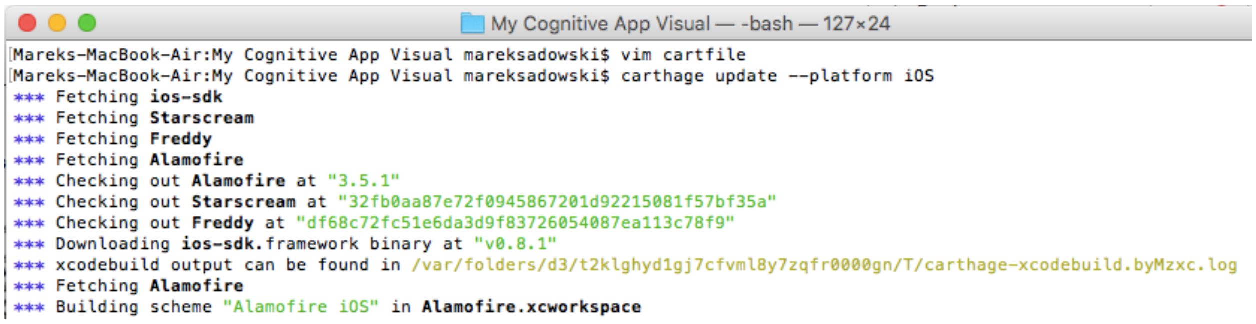
If want to run the latest development version of Carthage, which might be highly unstable or incompatible, clone the master branch of the repository and then run the command `make install`.

Step 2b. Install the Watson SDK

To install the SDK, open a Bash shell in the root directory of your project and follow these steps:

1. Enter `cat > cartfile`
2. Enter `github "watson-developer-cloud/ios-sdk"`
3. Enter a new line.
4. Press control + C to exit Edit mode.
5. From the command line at the root of the project, run the command `carthage update --platform iOS`. If you receive a compile failure for the framework AlamofireObjectMapper, run the command again.

   The following image shows output from the package manager as it fetches and builds iOS library components:



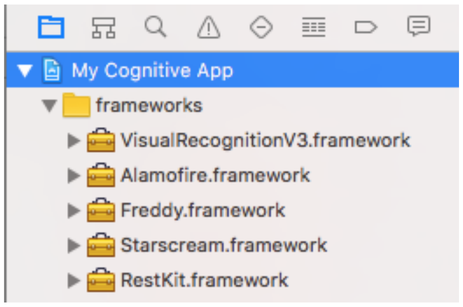
```
Mareks-MacBook-Air:My Cognitive App Visual mareksadowski$ vim cartfile
Mareks-MacBook-Air:My Cognitive App Visual mareksadowski$ carthage update --platform iOS
*** Fetching ios-sdk
*** Fetching Starscream
*** Fetching Freddy
*** Fetching Alamofire
*** Checking out Alamofire at "3.5.1"
*** Checking out Starscream at "32fb0aa87e72f0945867201d92215081f57bf35a"
*** Checking out Freddy at "df68c72fc51e6da3d9f83726054087ea113c78f9"
*** Downloading ios-sdk.framework binary at "v0.8.1"
*** xcodebuild output can be found in /var/folders/d3/t2klghyd1gj7cfvml8y7zqfr0000gn/T/carthage-xcodebuild.byMzxc.log
*** Fetching Alamofire
*** Building scheme "Alamofire iOS" in Alamofire.xcworkspace
```

Step 2c. Add the SDK to the Xcode project

1. Under the project name, create a new group in your Xcode project called `Framework`.
2. Navigate to the root folder of your project in Xcode and select all the framework files in the <your project>/Carthage/Build/iOS/ directory of your project (WatsonDeveloperCloud.framework, Alamofire, AlamofireObjectMapper, HTTPStatusCodes, ObjectMapper, Starscream). Drag and drop those files from Finder into the new `Framework` group in your project in Xcode.

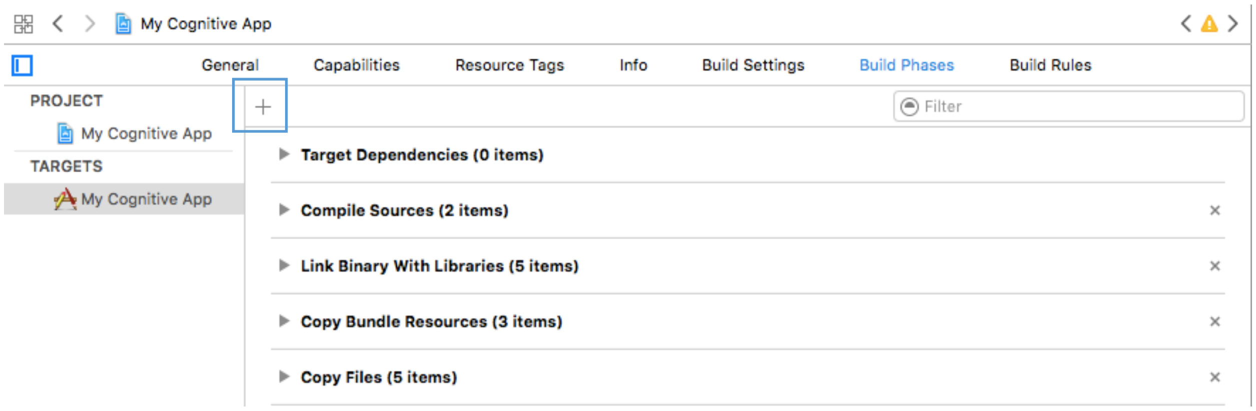
   Be sure you clear the option to copy items. By not copying the items, you create only a reference to those Framework files.

   This is the list of the framework libraries:



My Cognitive App
frameworks
VisualRecognitionV3.framework
Alamofire.framework
Freddy.framework
Starscream.framework
RestKit.framework

3. In the **Build Phases** tab, add a new **Copy Files** phase and set its destination to `Frameworks`.



My Cognitive App

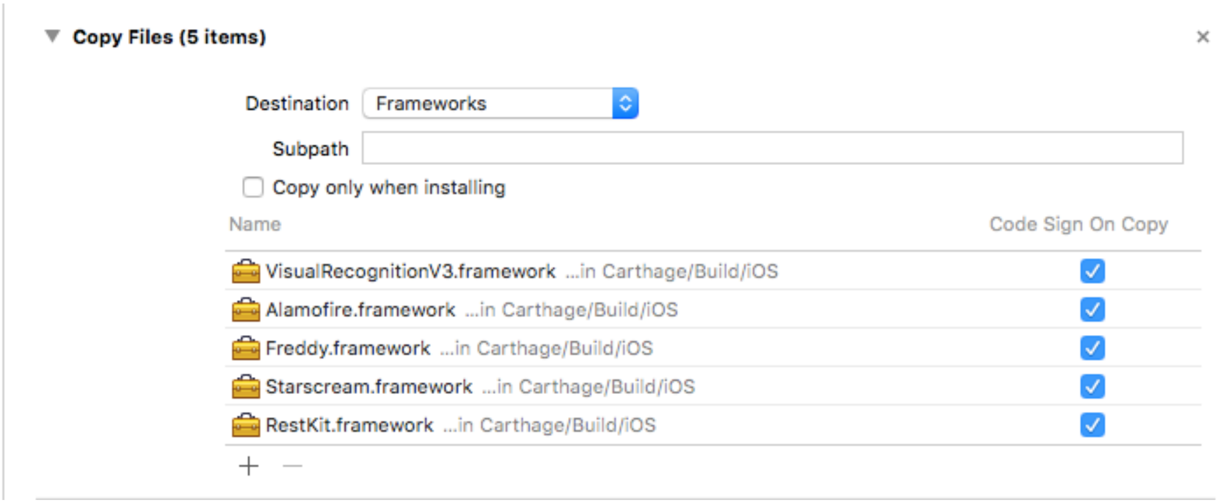
General | Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

PROJECT
My Cognitive App
TARGETS
My Cognitive App

Filter

Target Dependencies (0 items)
Compile Sources (2 items)
Link Binary With Libraries (5 items)
Copy Bundle Resources (3 items)
Copy Files (5 items)

4. For the frameworks that were added by Carthage, copy each framework individually. These are the files that you copy:


Copy Files (5 items)

Destination: Frameworks

Subpath:

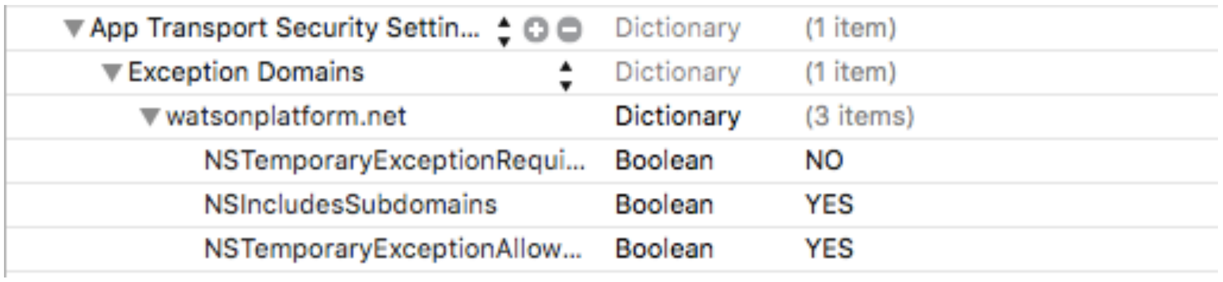
☐ Copy only when installing

Name	Code Sign On Copy
VisualRecognitionV3.framework ...in Carthage/Build/iOS	✓
Alamofire.framework ...in Carthage/Build/iOS	✓
Freddy.framework ...in Carthage/Build/iOS	✓
Starscream.framework ...in Carthage/Build/iOS	✓
RestKit.framework ...in Carthage/Build/iOS	✓

5. Whitelist your calls to Watson services and remove the security for watsonplatform.net. Whitelisting is the use of anti-spam filtering software to allow only specified addresses to get through.

   To do this, use either Xcode to add exceptions as shown in the following image or add the XML snippets below to your info.plist file:

   Whitelist the calls in Xcode:



▼ App Transport Security Settin...	Dictionary	(1 item)
▼ Exception Domains	Dictionary	(1 item)
▼ watsonplatform.net	Dictionary	(3 items)
NSTemporaryExceptionRequi...	Boolean	NO
NSIncludesSubdomains	Boolean	YES
NSTemporaryExceptionAllow...	Boolean	YES

Whitelist the calls by copying the following XML to the info.plist file:

```
<key>NSAppTransportSecurity</key>
  <dict>
        <key>NSExceptionDomains</key>
        <dict>
            <key>watsonplatform.net</key>
            <dict>
                <key>NSTemporaryExceptionRequiresForwardSecrecy</
key>
                <false/>
                <key>NSIncludesSubdomains</key>
                <true/>

<key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
                <true/>
            </dict>
        </dict>
    </dict>
```

Step 2d. Create a Watson service and get the key token for it

Now, you create the Watson AlchemyAPI service.

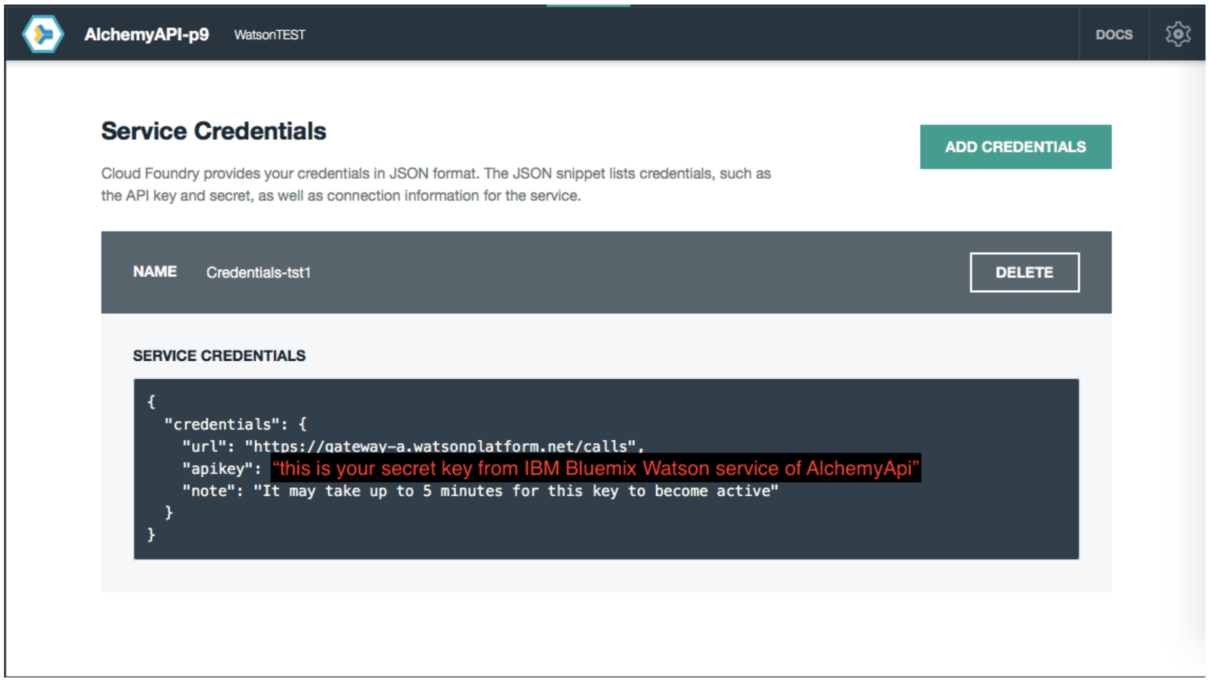
1. Use your existing AlchemyAPI credentials that you created in the previous lab or create a new one by clicking **Watson > AlchemyAPI** > **Create** from the Bluemix Catalog.

   You can create only one AlchemyAPI service (and credentials) in a 24-hour period.

   Be sure to use a static API key as shown in the following image.
2. Find the Service Credentials.

Bluemix provides your credentials in JSON format. The JSON snippet lists credentials, such as the API key and secret, and connection information for the service.

3. Add the following line of code to call the cognitive service. Insert the import field (`import VisualRecognitionV3`) under previously existing import fields.



Service Credentials

ADD CREDENTIALS

Cloud Foundry provides your credentials in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

NAME Credentials-tst1

DELETE

SERVICE CREDENTIALS

```
{
  "credentials": {
    "url": "https://gateway-a.watsonplatform.net/calls",
    "apikey": "this is your secret key from IBM Bluemix Watson service of AlchemyApi"
    "note": "It may take up to 5 minutes for this key to become active"
  }
}
```

In addition please add the global parameter (`let alchemyVisionInstance = AlchemyVision(apiKey: "<API key goes here>")`)

So far, your code should look like this:

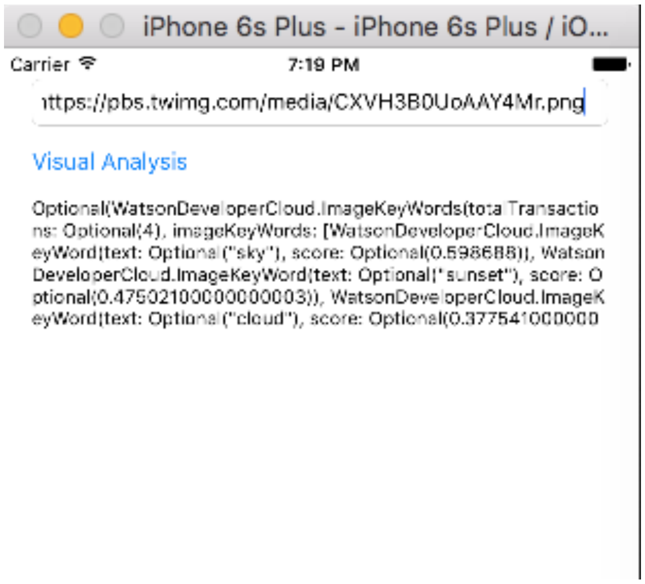
```
         //call service
            //Adding Watson Visual Recognition - based on the example
      from WDC sdk iOS
    let apiKey = "<enter the Watson APIKey here>"
        let version = "2016-05-22" // use today's date for the most
recent version
        let visualRecognition = VisualRecognition(apiKey: apiKey,
version: version)


        let url = urlText.text!
        let failure = { (error: NSError) in print(error) }
        visualRecognition.classify(url, failure: failure)
{ classifiedImages in
            //print(classifiedImages)
            status = "visual status ::::::::::::: " +
(classifiedImages.images.description)
        //detecting classification goes here
}
//setting feedback on sentiment
        self.analysisTextLabel.text = "1 : " + status
```

You might want additional analysis done through introduction of the classification and face recognition:

```
//detecting classification
            if (!classifiedImages.images.isEmpty && !
classifiedImages.images[0].classifiers.isEmpty &&
                !
classifiedImages.images[0].classifiers[0].classes.isEmpty) {
                status = status + "######## classification : " +
classifiedImages.images[0].classifiers[0].classes[0].classification


                //detecting faces on the pictures with people
                if (!
classifiedImages.images[0].classifiers[0].classes[0].classification.isE
mpty &&
                    "person" ==
classifiedImages.images[0].classifiers[0].classes[0].classification) {
                    self.analysisTextLabel.text = status + " ##########
person found ###########"
                    visualRecognition.detectFaces(url, failure:
failure) { imagesWithFaces in
```

```
                        //print(imagesWithFaces)
                        if (!imagesWithFaces.images[0].faces.isEmpty) {
                            status = status + " ###### the person's age
max : " + imagesWithFaces.images[0].faces[0].age.max.description
                            status = status + " age min : " +
imagesWithFaces.images[0].faces[0].age.min.description
                            status = status + " person's gender : " +
imagesWithFaces.images[0].faces[0].gender.gender
                            NSLog("faces ::::::::::::: ")
                            self.analysisTextLabel.text = "3 : " +
status


                        }
                    }
                }
            }
            //setting feedback on sentiment
            self.analysisTextLabel.text = "2 : " + status
```

When you run the application, you'll see the following results after the application checks the image at the given URL:



```
ıttps://pbs.twimg.com/media/CXVH3B0UoAAY4Mr.png
```

```
Optional(WatsonDeveloperCloud.ImageKeyWords(totalTransactio
ns: Optional(4), imageKeyWords: [WatsonDeveloperCloud.ImageK
eyWord(text: Optional("sky"), score: Optional(0.598688)), Watson
DeveloperCloud.ImageKeyWord(text: Optional("sunset"), score: O
ptional(0.47502100000000003)), WatsonDeveloperCloud.ImageK
eyWord(text: Optional("cloud"), score: Optional(0.37754100000
```

This is the complete code for the application:

```
import UIKit

import VisualRecognitionV3 //Watson library for Visual
Recognition


class ViewController: UIViewController {



    @IBOutlet weak var urlText: UITextField!

    @IBOutlet weak var analysisTextLabel: UILabel!



    override func viewDidLoad() {

        super.viewDidLoad()

        // Do any additional setup after loading the view,
typically from a nib.

    }


    override func didReceiveMemoryWarning() {

        super.didReceiveMemoryWarning()

        // Dispose of any resources that can be recreated.

    }


    @IBAction func analysisButtonPressed(sender: AnyObject) {

        NSLog("Button pressed")

        NSLog("URL: "+urlText.text!)

        var status = "waiting for Watson processing the URL : " +
urlText.text!


        //Adding Watson Visual Recognition - based on the example
from WDC sdk iOS

        let apiKey = "<enter the Watson APIKey here>"

        let version = "2016-05-22" // use today's date for the
most recent version

        let visualRecognition = VisualRecognition(apiKey: apiKey,
version: version)
```

```
        let url = urlText.text!
        let failure = { (error: NSError) in print(error) }
        visualRecognition.classify(url, failure: failure)
{ classifiedImages in
            //print(classifiedImages)
            status = "visual status ::::::::::::: " +
(classifiedImages.images.description)


            //detecting classification
            if (!classifiedImages.images.isEmpty && !
classifiedImages.images[0].classifiers.isEmpty &&
                !
classifiedImages.images[0].classifiers[0].classes.isEmpty) {
                status = status + "######## classification : " +
classifiedImages.images[0].classifiers[0].classes[0].classificati
on


                //detecting faces on the pictures with people
                if (!
classifiedImages.images[0].classifiers[0].classes[0].classificati
on.isEmpty &&
                    "person" ==
classifiedImages.images[0].classifiers[0].classes[0].classificati
on) {
                    self.analysisTextLabel.text = status + "
##########  person found ###########"
                    visualRecognition.detectFaces(url, failure:
failure) { imagesWithFaces in
                        //print(imagesWithFaces)
                        if (!
imagesWithFaces.images[0].faces.isEmpty) {
                            status = status + " ###### the
person's age max : " +
imagesWithFaces.images[0].faces[0].age.max.description
                            status = status + " age min : " +
imagesWithFaces.images[0].faces[0].age.min.description
                            status = status + " person's gender :
" + imagesWithFaces.images[0].faces[0].gender.gender
                            NSLog("faces ::::::::::::: ")
                            self.analysisTextLabel.text = "3 : "
+ status
```

```
                        }
                    }
                }
            }
            //setting feedback on sentiment
            self.analysisTextLabel.text = "2 : " + status
        }
        //setting feedback on sentiment
        self.analysisTextLabel.text = "1 : " + status
    }

}
```

Step 3. Run the Watson cognitive application

You can now run the Watson application by entering other image URLs in the text field and clicking the button.

The image specified by URL is sent to Watson through the Watson SDK, and Watson returns a JSON object that describes a recognized classification with some level of confidence about that image in the status field.

Check these image links by entering them into the text field:

- person with age: http://g1.computerworld.pl/news/thumbnails/2/6/265397_resize_620x460.jpg
- person = https://static.sched.org/a6/1606352/avatar.jpg.320x320px.jpg?3c4
- beach: http://1.bp.blogspot.com/_dWmIOlGB7_0/S9nvqlTPz9I/AAAAAAAADCw/PRmRH_UPseI/s1600/beach+palm+trees+%283%29.jpg
- hiking: http://www.backpaco.com/wp-content/uploads/2015/04/yosemite-park.jpg
- tatry: http://s3.flog.pl/media/foto/5654654_tatry-dolina-pieciu-stawow-polskich-widok-z-gory.jpg
- diving: http://www.nautica.pl/images/phocagallery/safari_warsztat_foto/thumbs/phoca_thumb_l_warsztaty_fotograficzne_egipt2.jpg
- diving: http://www.nautica.pl/images/phocagallery/safari_warsztat_foto/thumbs/phoca_thumb_l_warsztaty_fotograficzne_egipt3.jpg
- camping: http://cospuente.org/images/351.jpg
- skiing: https://murowanica.files.wordpress.com/2013/12/rersdtfghj.jpg
-