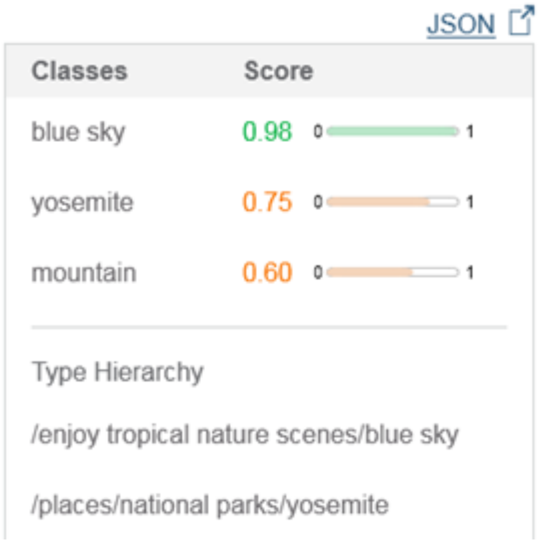
Lab: create a Watson image recognition app with Swift

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.


JSON

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 or later
- Xcode 9.0 or later
- Swift 4.0.x
- Carthage (package manager similar to Ant): https://github.com/Carthage/Carthage#installing-carthage

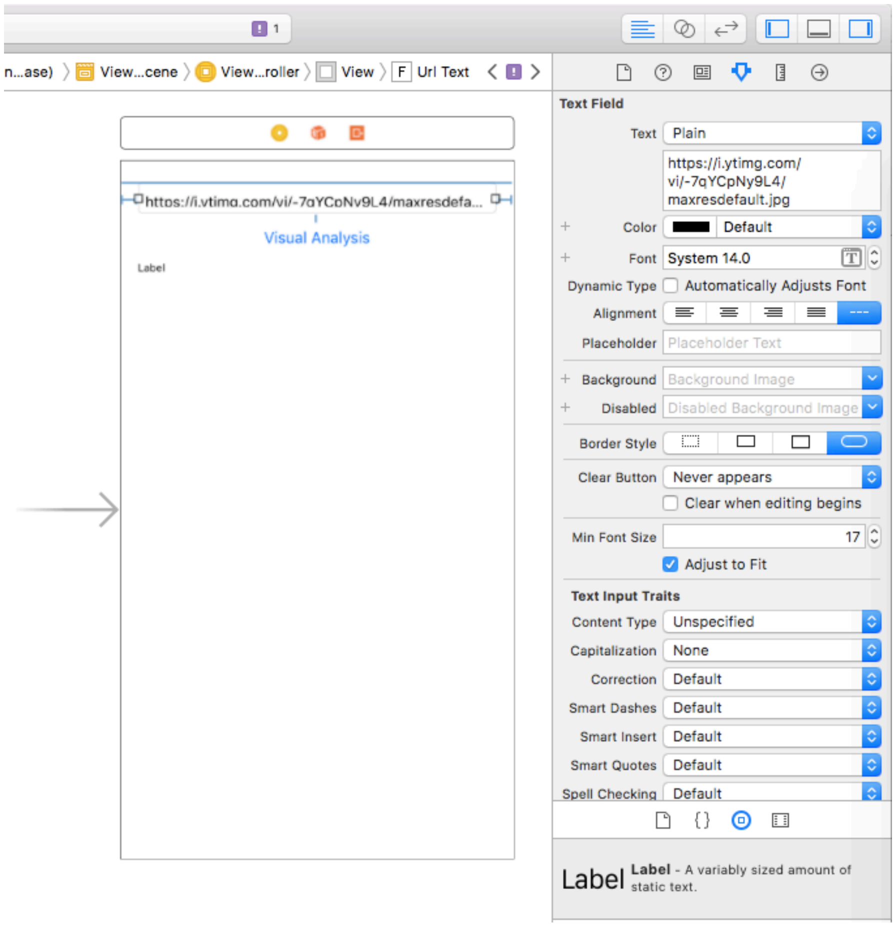
Complete the previous lab “Create a Watson sentiment analysis app with Swift.”

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.

1. 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 a classification that is shown in the label field.



Text Field

- Text: Plain
- https://i.ytimg.com/vi/-7qYCpNy9L4/maxresdefault.jpg
- Color: Default
- Font: System 14.0
- Dynamic Type: ☐ Automatically Adjusts Font
- Alignment
- Placeholder: Placeholder Text
- Background: Background Image
- Disabled: Disabled Background Image
- Border Style
- Clear Button: Never appears
- ☐ Clear when editing begins
- Min Font Size: 17
- ☑ Adjust to Fit

Text Input Traits

- Content Type: Unspecified
- Capitalization: None
- Correction: Default
- Smart Dashes: Default
- Smart Insert: Default
- Smart Quotes: Default
- Spell Checking: Default

Label **Label** - A variably sized amount of static text.

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

2. Connect the GUI in Main.storyboard to the code in the ViewController file:
   a. Double-click **TextField** (by pressing the mouse pad with two fingers), which contains the URL. Enter this example text:

      `https://i.ytimg.com/vi/-7qYCpNy9L4/maxresdefault.jpg`

   b. Select **New Reference Outlet** from the list. Then, enter `urlText` in the dialog.
   c. Insert the new reference between **class ViewController** and **override func viewDidLoad.**
   d. Click **Connect**.

      The inserted text is `@IBOutlet weak var urlText: UITextField!`

   e. Connect the button from the Event `Touch Up Inside` below the function `didReceiveMemoryWarning.`
   f. Enter this text in the dialog: `analysisButtonPressed`

      The inserted text is:
      ```
      @IBAction func analysisButtonPressed(_ sender: Any) {
      }
      ```

   g. Connect the text label.

      The result is `@IBOutlet weak var analysisTextLabel: UILabel!`

   h. Test the code by adding `NSLog("url: "+urlText.text!)` to the end of the `analysisButtonPressed` method.

      NSLog allows you to log your actions.

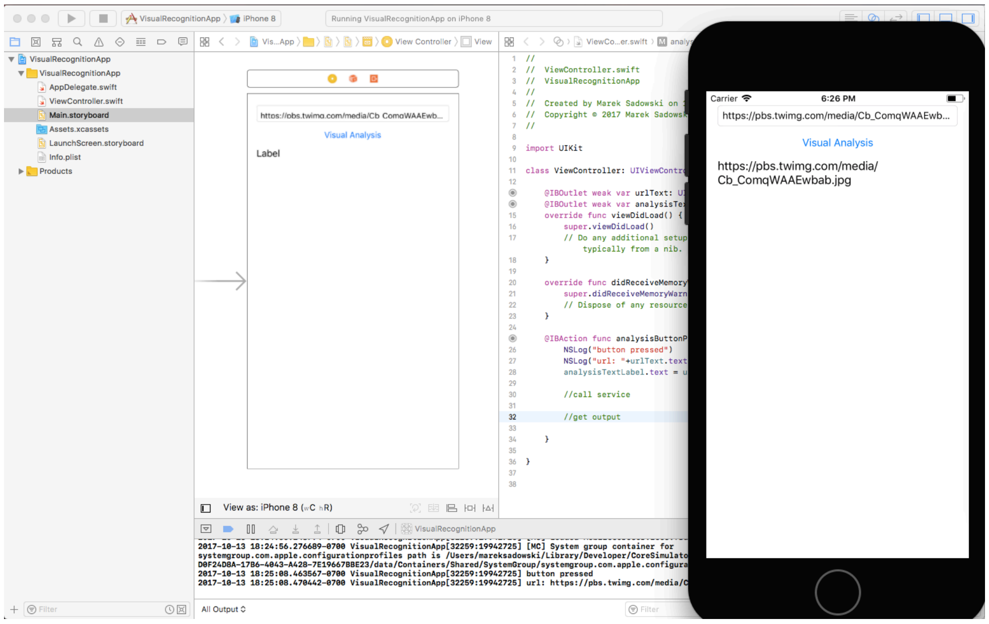
   i. Add `analysisTextLabel.text = urlText.text!`

      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: Any) {
        NSLog("button pressed")
        NSLog("url: "+urlText.text!)
        analysisTextLabel.text = urlText.text!

        //call service

        //get output

    }
}
```

j. Build and execute your application.



```
//
//  ViewController.swift
//  VisualRecognitionApp
//
//  Created by Marek Sadowski on 
//  Copyright © 2017 Marek Sadowsk
//

import UIKit

class ViewController: UIViewContro

    @IBOutlet weak var urlText: U
    @IBOutlet weak var analysisTe
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup
            typically from a nib.
    }

    override func didReceiveMemory
        super.didReceiveMemoryWarn
        // Dispose of any resource
    }

    @IBAction func analysisButtonP
        NSLog("button pressed")
        NSLog("url: "+urlText.text
        analysisTextLabel.text = u

        //call service

        //get output

    }

}
```

```
2017-10-13 18:24:56.276689-0700 VisualRecognitionApp[32259:19942725] [MC] System group container for
systemgroup.com.apple.configurationprofiles path is /Users/mareksadowski/Library/Developer/CoreSimulato
D0F24D8A-17B6-4043-A428-7E19667BBE23/data/Containers/Shared/SystemGroup/systemgroup.com.apple.configura
2017-10-13 18:25:08.463567-0700 VisualRecognitionApp[32259:19942725] button pressed
2017-10-13 18:25:08.470442-0700 VisualRecognitionApp[32259:19942725] url: https://pbs.twimg.com/media/C
```

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.

1. 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.

   - run the commands `brew update` and `brew install Carthage.`

2. Install the Watson SDK:

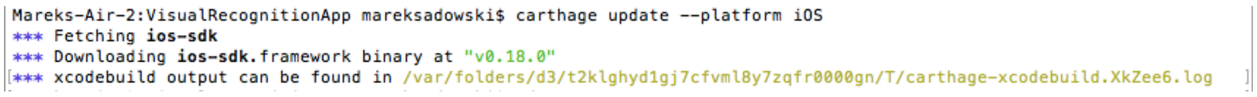
   a. Open a Bash shell in the root directory of your project.

   b. Enter `cat > cartfile`

   c. Enter `github "watson-developer-cloud/ios-sdk"`

   d. Enter a new line. Then, press control + C to exit Edit mode.

   e. 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:



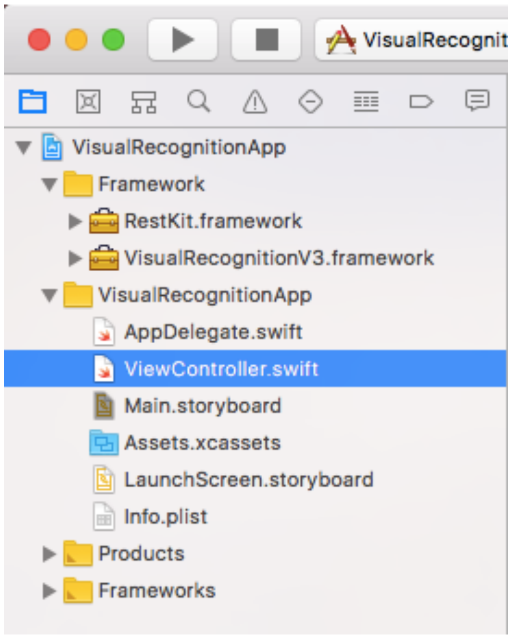
```
Mareks-Air-2:VisualRecognitionApp mareksadowski$ carthage update --platform iOS
*** Fetching ios-sdk
*** Downloading ios-sdk.framework binary at "v0.18.0"
*** xcodebuild output can be found in /var/folders/d3/t2klghyd1gj7cfvml8y7zqfr0000gn/T/carthage-xcodebuild.XkZee6.log
```

3. Add the SDK to the Xcode project:

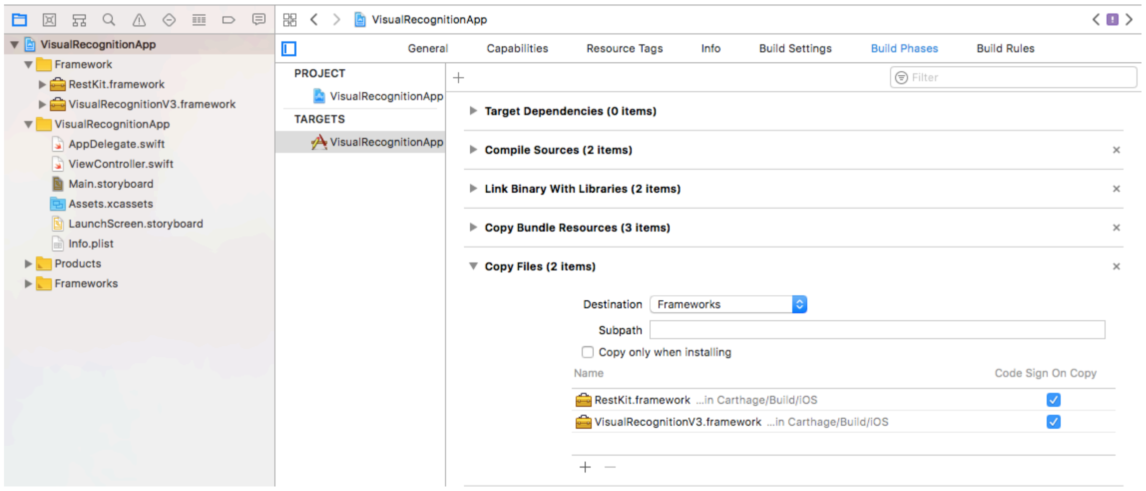
   a. Under the project name, create a new group in your Xcode project called `Framework`.

   b. 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 (VisualRecognitionV3.framework, RestKit.framework). 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:



c. In the **Build Phases** tab, add a new **Copy Files** phase and set its destination to `Frameworks`. For the frameworks that were added by Carthage, copy each framework. These are the files that you copy:



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

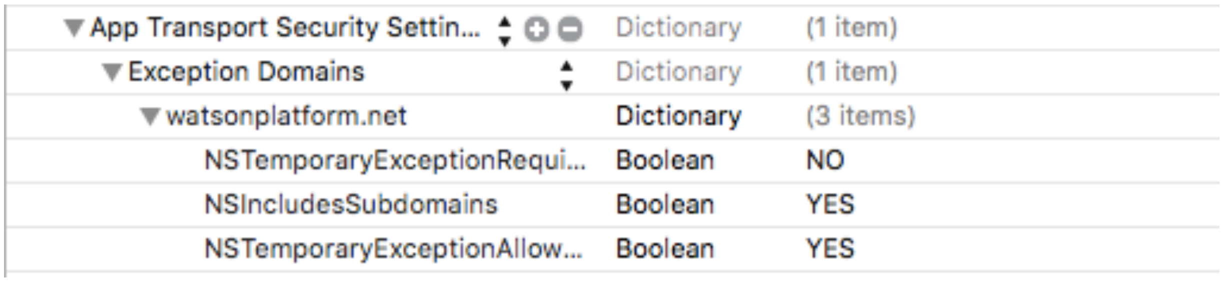
Destination: Frameworks
Subpath:
☐ Copy only when installing

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

d. Whitelist your calls to Watson services and remove the security for watsonplatform.net. Whitelisting allows only specified addresses to get through without the SSL protocol.

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

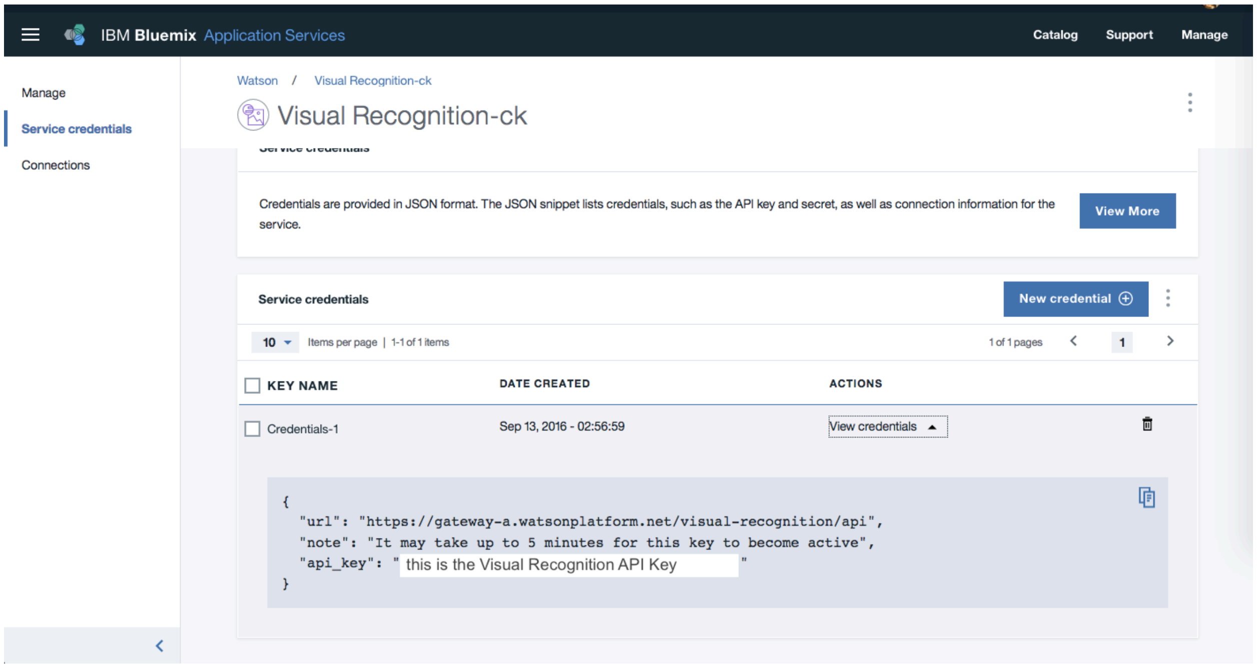
Whitelist the calls by copying the following XML to the info.plist file:

```
<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 3. Create the Watson service and get the key token for it

Create the Watson Visual Recognition service.

1. Use your existing AlchemyAPI credentials that you created in the previous lab or create a new one by clicking **Watson > Visual Recognition** > **Create** from the Bluemix Catalog.
2. Find and copy the Service Credentials.



Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

Service credentials

KEY NAME	DATE CREATED	ACTIONS
Credentials-1	Sep 13, 2016 - 02:56:59	View credentials

```
{
  "url": "https://gateway-a.watsonplatform.net/visual-recognition/api",
  "note": "It may take up to 5 minutes for this key to become active",
  "api_key": " this is the Visual Recognition API Key "
}
```

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

   Import field:

   ```
   import VisualRecognitionV3
   ```

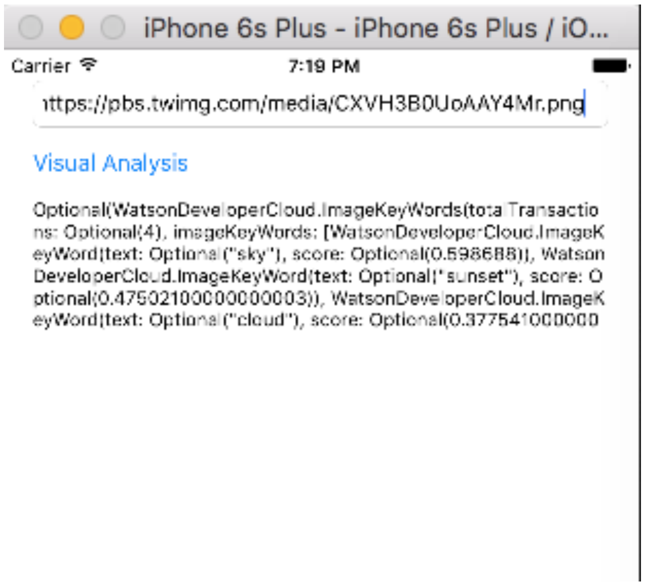
4. Add the global parameter (`let apiKey = "<enter the Watson APIKey here>")`.

   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 = "2016-11-04" // use today's date for the most recent
version
let visualRecognition = VisualRecognition(apiKey: apiKey, version:
version)

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

             //detecting classification



         }
         //setting feedback on analysis
         self.analysisTextLabel.text = "1 : " + 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(0.475021000000000003)), WatsonDeveloperCloud.ImageK
eyWord(text: Optional("cloud"), score: Optional(0.37754100000
```

This is the complete code for the application:

```
import UIKit
import VisualRecognitionV3
```

```
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: Any) {
        NSLog("button pressed")
        NSLog("url: "+urlText.text!)
        analysisTextLabel.text = urlText.text!
        var status = "waiting for Watson processing the URL : " +
urlText.text!

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

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

        let version = "2016-11-04" // use today's date for the most
recent version
        let visualRecognition = VisualRecognition(apiKey: apiKey,
version: version)

        let url = urlText.text!
        let failure = { (error: Error) in print(error) }
        visualRecognition.classify(image: 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.isE
mpty) {
                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.
isEmpty &&
                    classifiedImages.images.description.contains("pers
on")) {

                    DispatchQueue.main.async {
                        // Update UI in the main thread
                        self.analysisTextLabel.text = status + "
##########  person found ###########"
                    }

                    visualRecognition.detectFaces(inImage: 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 :::::::::::::: ")
                            DispatchQueue.main.async {
                                // Update UI in the main thread
                                self.analysisTextLabel.text = "3 : " +
status
                            }

                        }
                    }
                }
            }
            DispatchQueue.main.async {
                // Update UI in the main thread
                //setting feedback on analysis
                self.analysisTextLabel.text = "2 : " + status
            }
        }
        //setting feedback on analysis
        self.analysisTextLabel.text = "1 : " + status
    }
}
```

Step 4. Run the Watson cognitive application

1. 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.

2. Check these image links by entering the URLs into the text field:

   - A person (older): http://g1.computerworld.pl/news/thumbnails/2/6/265397_resize_620x460.jpg
   - A beach: http://1.bp.blogspot.com/_dWmIOlGB7_0/S9nvqlTPz9I/AAAAAAAADCw/PRmRH_UPseI/s1600/beach+palm+trees+%283%29.jpg
   - US national park: http://www.backpaco.com/wp-content/uploads/2015/04/yosemite-park.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
   - Snow skiing: https://murowanica.files.wordpress.com/2013/12/rersdtfghj.jpg

Summary

Now, you should understand how to create a Swift application with the Watson Visual Recognition service. The application can consume an image URL, identify the image, and return a confidence score for relevant classifiers representing things such as objects, events, and settings.