Lab 1: create a Watson sentiment analysis app with Swift

Overview

This lab shows you how to build a mobile application that will analyze tone, or sentiment, in text. You'll build the application in Swift and use the IBM Watson Natural Language Understanding service to analyze sentiment conveyed in text.

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

1. Create a typical iOS application in Swift.
2. Install the Watson SDK for iOS.
3. Create the Bluemix Watson service and get the credentials to it.
4. Add some code to invoke the cognitive service.

You can see the code the GUI for this lab is 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

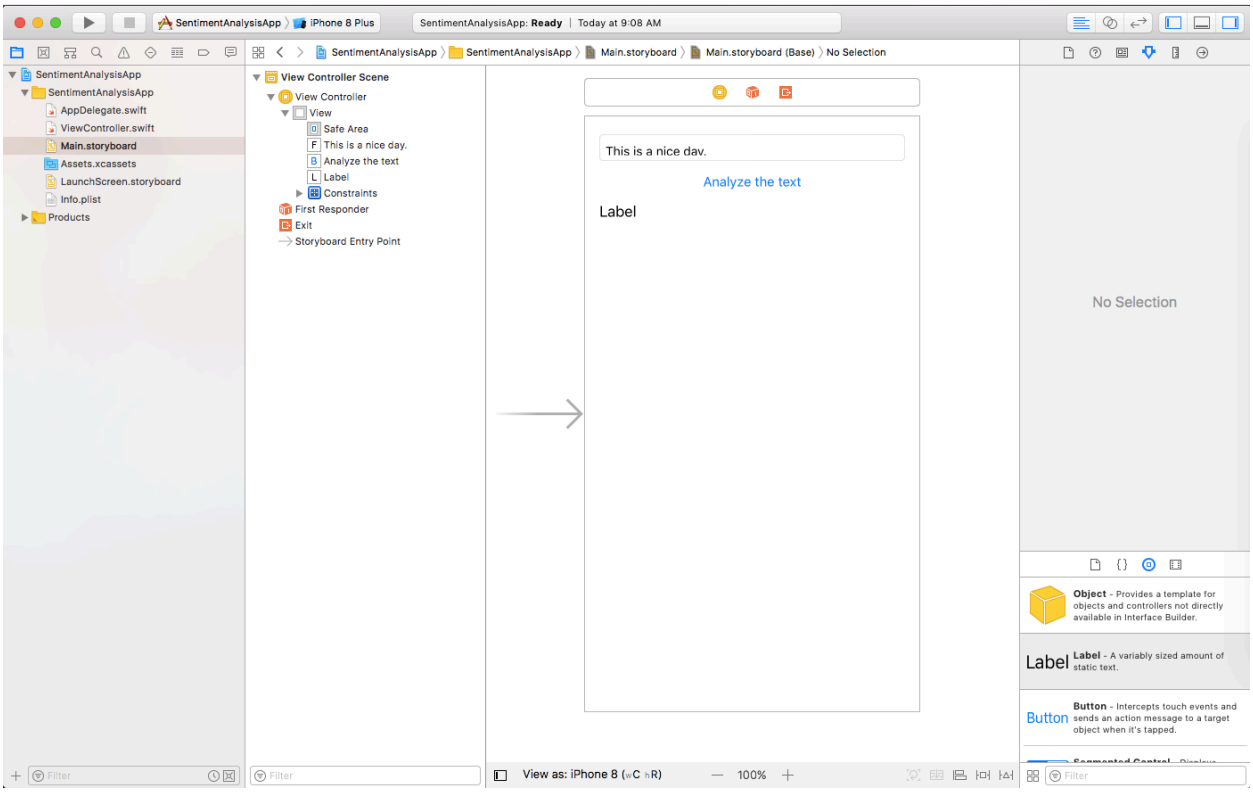
Step 1. Create a typical application in Swift

You'll build a simple application with a submit button, an editable text field, and an output 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 a user presses the button, the URL in the text field is sent to Watson, which analyzes it and returns an opinion that is shown in the output field.

   Take the text in the text field and echo it in the label field when the button is pressed.

   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:

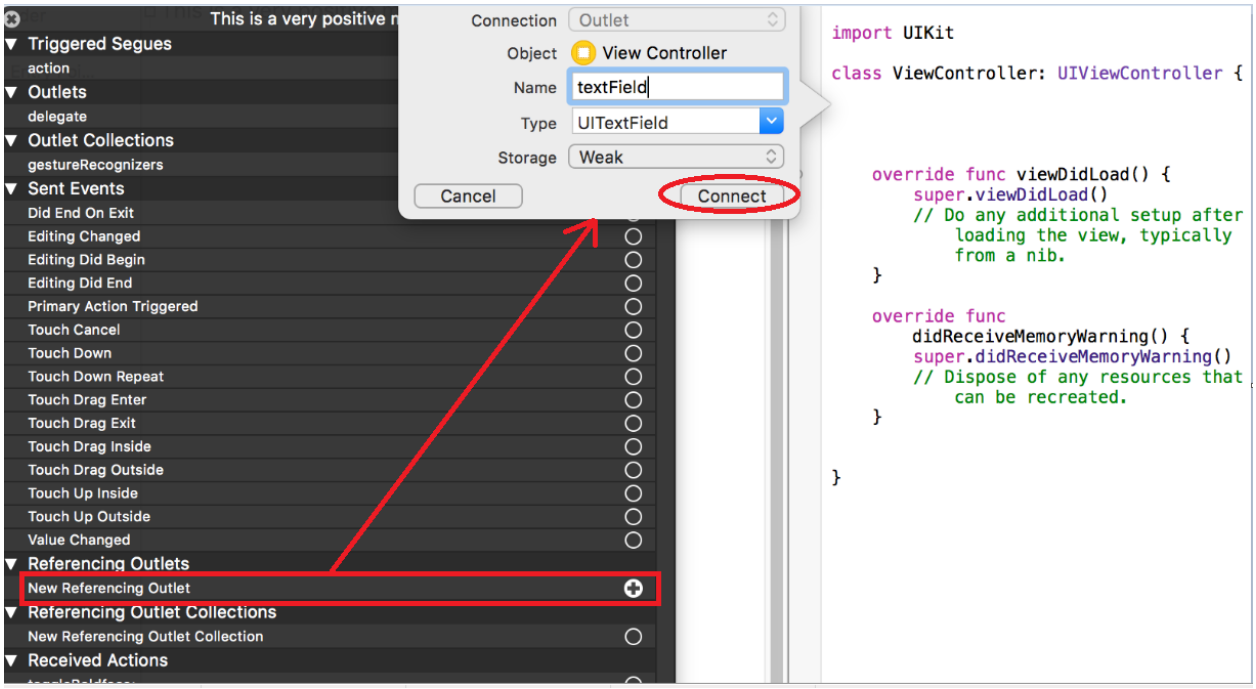
   a. Double-click **TextField** (by pressing the mouse pad with two fingers), which contains the URL to the document https://www.wired.com/2017/09/review-apple-iphone-8-and-8-plus/.

   b. Select **New Referencing Outlet** from the list.

   c. Enter `textField` in the dialog.

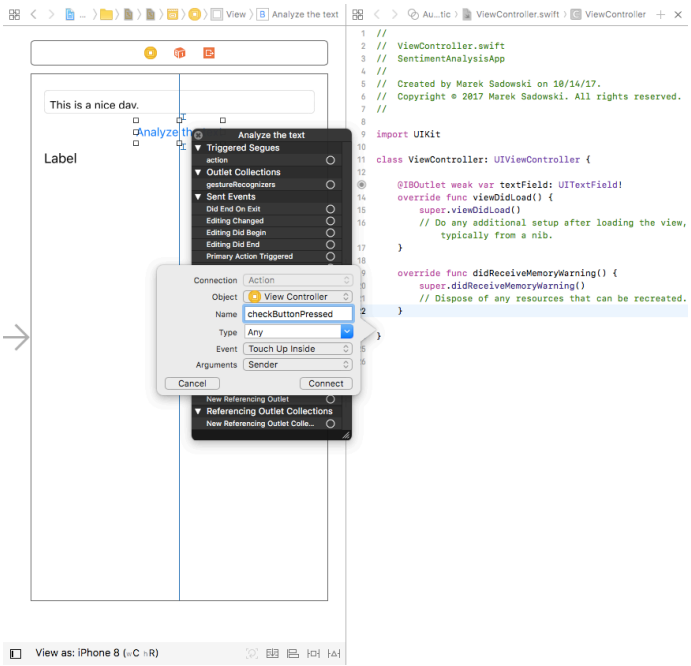
   d. Insert the new reference between **class ViewController** and **override func viewDidLoad.**

e. Click **Connect**.



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

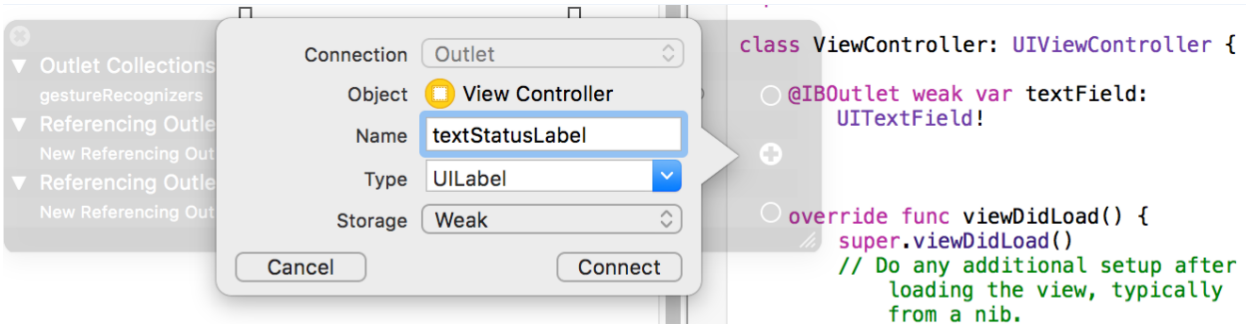
f. Connect the button. Insert the sent event named **Touch up Inside** before the last curly bracket.



The inserted text is:

```
@IBAction func checkButtonPressed(_ sender: Any) {
        }
```

g. Connect the text label. Insert the parameter **UILabel** between `@IBOutlet weak var textField: UITextField!` and `override func…`



The inserted text is `@IBOutlet weak var textStatusLabel: UILabel!`

h. Test the code by adding `NSLog(textField.text!)` to the end of the checkButtonPressed method.

NSLog allows you to log your actions. Setting textStatusLabel.text with textField.text allows you to show the entered text of the Text field in the Label field.

```
import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var textStatusLabel: 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 checkButtonPressed(_ sender: Any) {
```

```
            NSLog(textField.text!)

            //checking sentiment

            //setting feedback on sentiment
            textStatusLabel.text = textField.text
        }
    }
```

i. Build and execute your application.

   If you see the text "`http://www.huffingtonpost.com/2010/06/22/iphone-4-review-the-worst_n_620714.html`" the GUI is working.

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

Install the Carthage dependency manager, 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.

   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`.

   If you 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`.

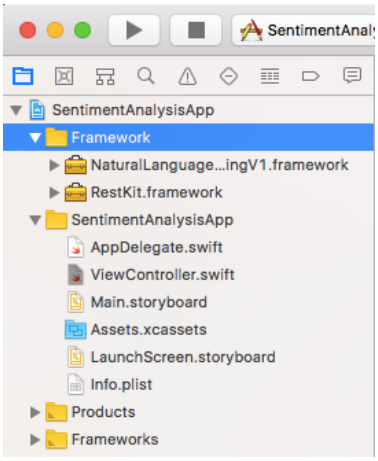
   Important: If you have two-factor authentication, Carthage requires an OTP header.

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:

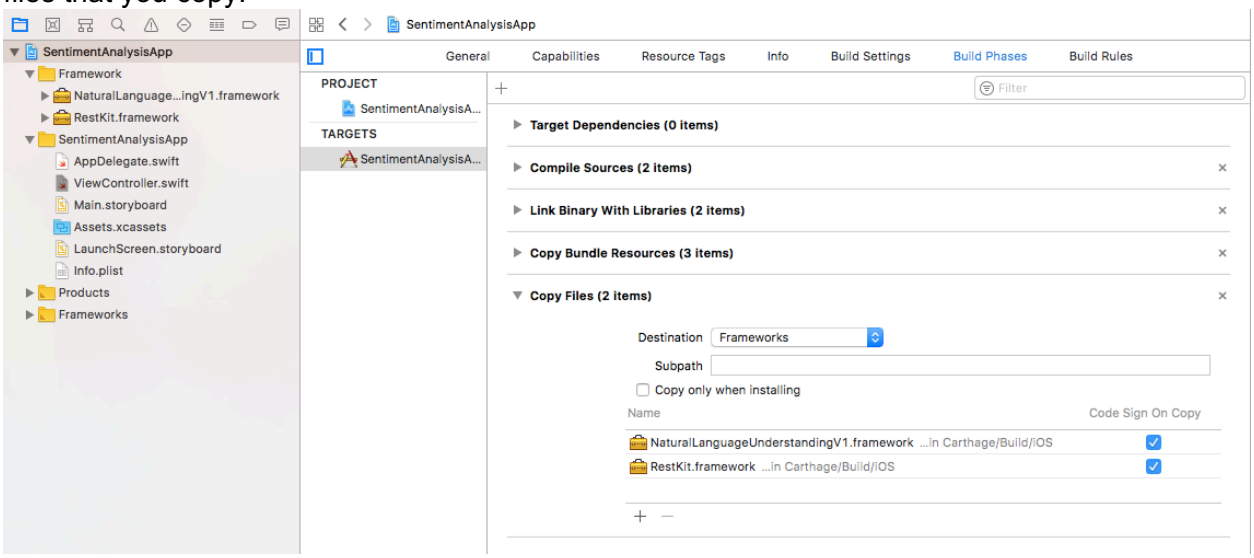
3. Add the SDK to the Xcode project:
   a. Under the project name, create a new group in your Xcode project called `Framework`.

      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 (NaturalLanguageUnderstandingV1.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.



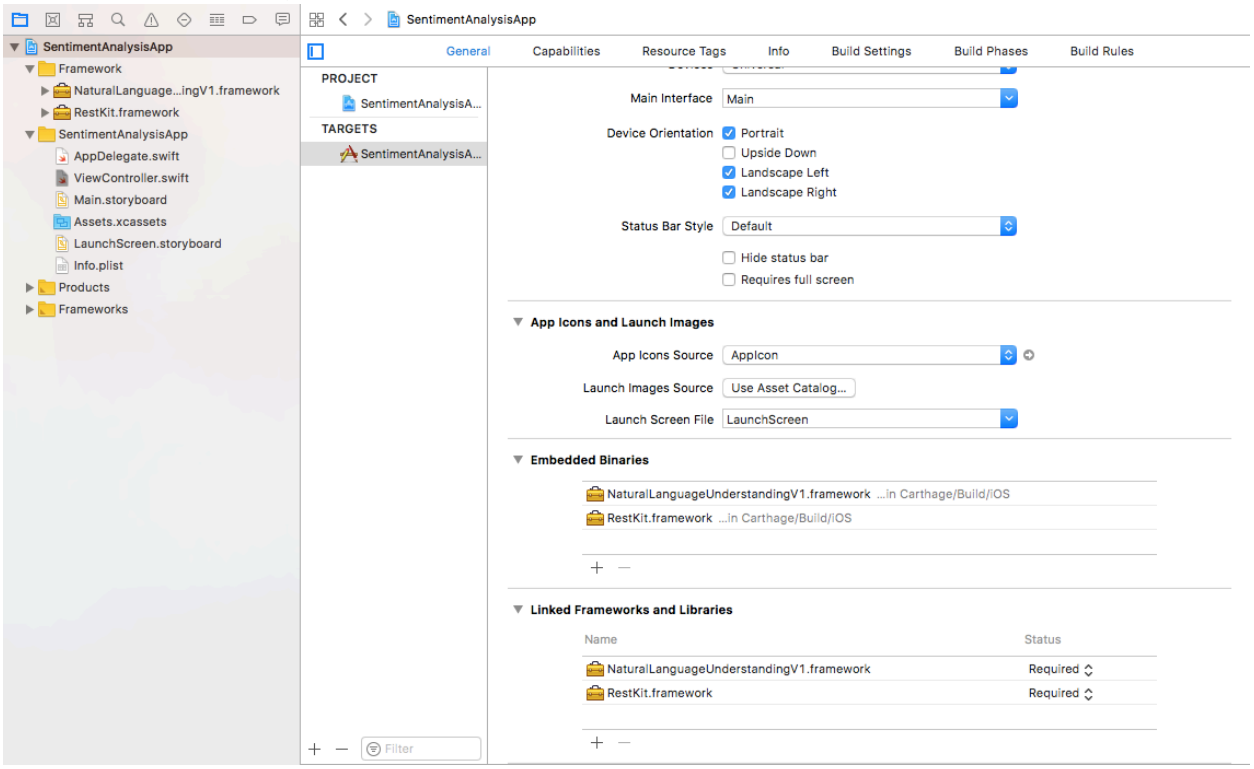
b. In the **Build Phases** tab, add a new **Copy Files** phase and set its destination to `Frameworks`.

c. For the frameworks that were added by Carthage, copy all frameworks. These are the files that you copy:



You should see also linked frameworks and libraries in order to be able to add the

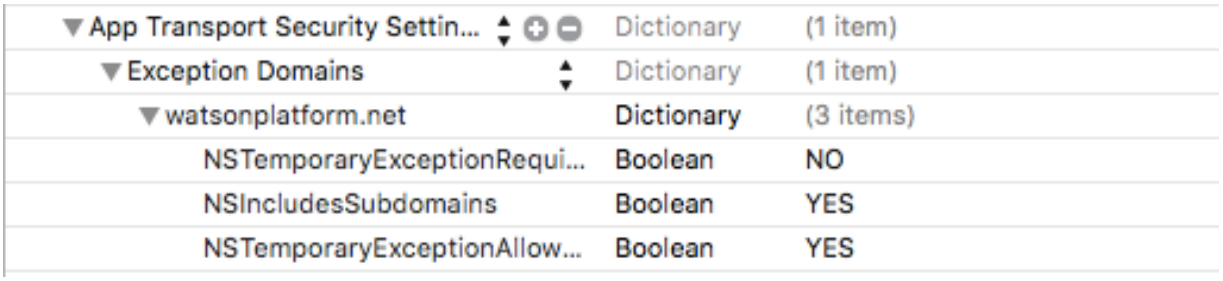
libraries to the code by "`import NaturalLanguageUnderstandingV1`"



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:



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

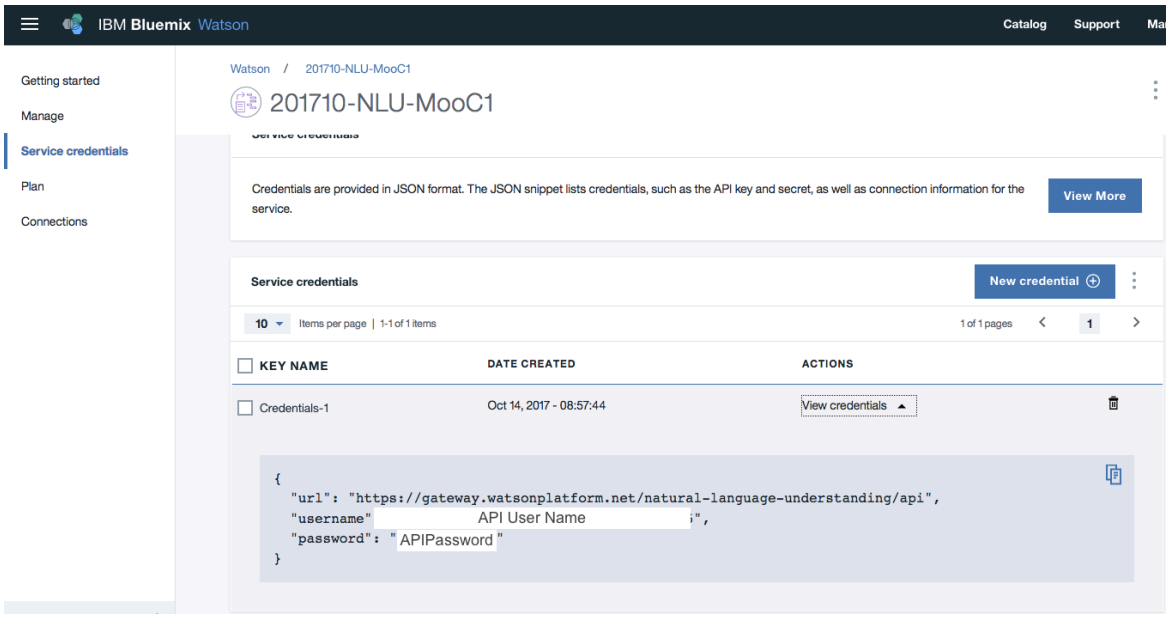
```
<key>NSAppTransportSecurity</key>
  <dict>
        <key>NSExceptionDomains</key>
        <dict>
            <key>watsonplatform.net</key>
```

```
            <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 AlchemyAPI service.

1. From the IBM Bluemix catalog, click **Watson > Natural Language Understanding** > **Create**. Be sure to use a static API key as shown in the following image.
2. Find and copy 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 lines of code to call the cognitive service. Insert the import field under previously existing import fields.

   Import field:

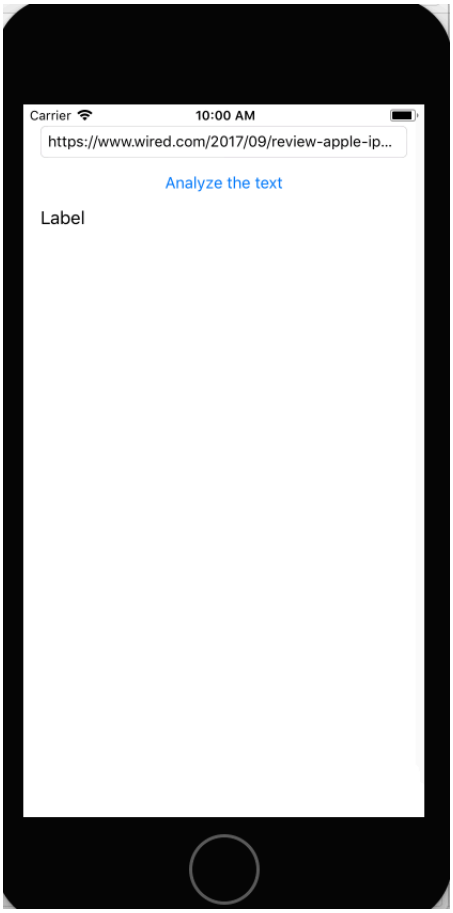
```
import NaturalLanguageUnderstandingV1
```

   So far, your code should look like this:

```
//checking sentiment
let username = "your-username-here"
let password = "your-password-here"
let version = "2017-02-27" // use today's date for the most recent version

        let naturalLanguageUnderstanding = NaturalLanguageUnderstanding(username: username, password: password, version: version)

        let urlToAnalyze = textField.text //"www.wsj.com/news/markets"
```

```
        let features = Features(sentiment: SentimentOptions(document:
true))
        let parameters = Parameters(features: features, url:
urlToAnalyze, returnAnalyzedText: true ) //check url
        //let parameters = Parameters(features: features, text:
urlToAnalyze, returnAnalyzedText: false) //check text
        let failure = { (error: Error) in print(error) }
        naturalLanguageUnderstanding.analyzeContent(withParameters:
parameters, failure: failure) {
            results in

            let score = results.sentiment?.document?.score
            var sentimentValue = "positive"
            if (score! < 0.0) {
                sentimentValue = "negative"
            } else if (score! == 0.0) {
                sentimentValue = "neutral"
            }
            NSLog("!!!!!!!!!!!!!! result: " +
results.sentiment.debugDescription)
            DispatchQueue.main.async {
                // Update UI in the main thread
                self.textStatusLabel.text = "analyzed text score " +
sentimentValue
            }
        }
```

4. Run the application. You should see the following results after the application checks the sentiment of some text:



This is the complete code for the application:

```
import UIKit
import NaturalLanguageUnderstandingV1

class ViewController: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var textStatusLabel: 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 checkButtonPressed(_ sender: Any) {
        NSLog(textField.text!)
        //checking sentiment
        let username = "your-username-here"
         let password = "your-password-here"
        let version = "2017-02-27" // use today's date for the most
recent version

        let naturalLanguageUnderstanding =
NaturalLanguageUnderstanding(username: username, password: password,
version: version)

        let urlToAnalyze = textField.text //"www.wsj.com/news/markets"
        let features = Features(sentiment: SentimentOptions(document:
true))
        let parameters = Parameters(features: features, url:
urlToAnalyze, returnAnalyzedText: true ) //check url
        //let parameters = Parameters(features: features, text:
urlToAnalyze, returnAnalyzedText: false) //check text
        let failure = { (error: Error) in print(error) }
        naturalLanguageUnderstanding.analyzeContent(withParameters:
parameters, failure: failure) {
            results in

            let score = results.sentiment?.document?.score
            var sentimentValue = "positive"
            if (score! < 0.0) {
                sentimentValue = "negative"
            } else if (score! == 0.0) {
                sentimentValue = "neutral"
            }
            NSLog("!!!!!!!!!!!!!! result: " +
results.sentiment.debugDescription)
            DispatchQueue.main.async {
                // Update UI in the main thread
                self.textStatusLabel.text = "analyzed text score " +
sentimentValue
            }
        }
    }
}
```


Summary

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

The URL is sent to Watson through the Watson SDK, and Watson returns a sentiment type that is displayed in the status field.