Mecolab – David Galemiri
28 de septiembre del 2015

Tutorial Networking iOS

Paso a paso para crear una aplicación usando POST y GET request.

En este tutorial se enseña paso a paso como crear una aplicación iOS con networking. Se hace una reseña sobre lo que son los servidores y el protocolo REST. Posteriormente se detalla como se crea la interfaz gráfica y la programación del código para llevar a cabo este proyecto.

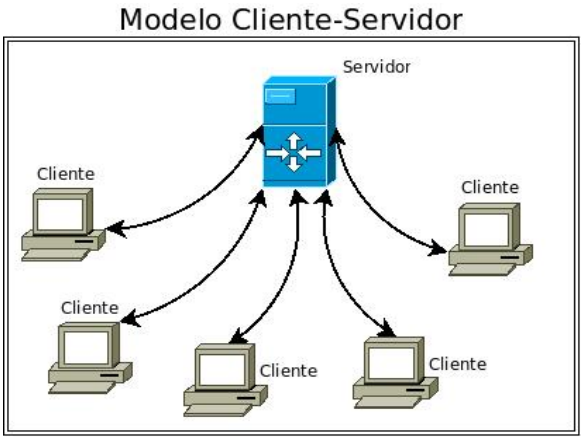
Indice:

Introducción

Las aplicaciones móviles se masifican cada día más. El creciente uso de sistemas operativos como Andriod y iOS demanda el desarrollo de nuevas librerías y tutoriales para los desarrolladores. En este instructivo se explicará la interacción de una aplicación iOS en lenguaje Swift con Networking. Se podrá descargar el código de la aplicación al final del tutorial.

Al interactuar con internet es muy frecuente escuchar la palabra servidor, pero ¿Qué es un servidor?. Se llama servidor a una entidad que se relaciona con un cliente, esta interacción es para lograr una comunicación. De este modo se puede enviar información de un servidor a un cliente, de un cliente a un servidor, entre otras cosas. En términos sencillos un servidor atiende peticiones y devuelve una respuesta.

El esquema de interacción es el siguiente:



Existe un agente llamado cliente el cual se conecta al servidor. En este caso, el cliente es la aplicación de iOS. El proceso por el cual el cliente interactúa con el servidor se llama Request. Al momento de interactuar se debe establecer un protocolo de comunicación en la red. En este tutorial se trabajará con el protocolo REST.

Interacción con el servidor/REST

La palabra REST viene de "REpresentational State Transfer", lo que en español significa “transferencia de representación de estado”. Un servicio REST tiene una característica es esencial: no tiene estado, ¿Qué quiere decir eso?. Esto significa que entre un Request y el otro se pierden todos los datos, por lo tanto al entregarle datos a un servicio REST no los recordará para la siguiente petición (por ejemplo si se entrega un usuario y un password, no se guardarán). El cliente debe pasar el estado en cada llamada (credenciales, entre otros), lo que implica que cada mensaje contiene toda la información necesaria para la petición. Estos Request devuelven información a través de algún formato (JSON, XML...), por lo tanto es

necesario extraer la información de ahí para el uso. Puede que el hecho de entregar cada vez el estado sea un poco problemático pero tiene una gran ventaja: su eficiencia y escalabilidad. Los servicios REST se usan mucho en aplicaciones móviles por ser simples y ligeros.

En contraste de otros servicios webs como SOAP (Simple Object Access Protocol) que invocan métodos sobre un servicio remoto (RPC, Remote Procedure Call – Llamada a método remoto), REST esta orientado a recursos (elementos de información); esto quiere decir que opera sobre recursos, no sobre servicios. En este tipo de arquitecturas se tienen recursos accesibles mediante URIs (identificadores)..
Cada recurso solo se puede direccionar por su URI. Sobre estos se pueden realizar acciones, las cuales están determinadas por los métodos del protocolo HTTP. Algunos de estos métodos son los siguientes

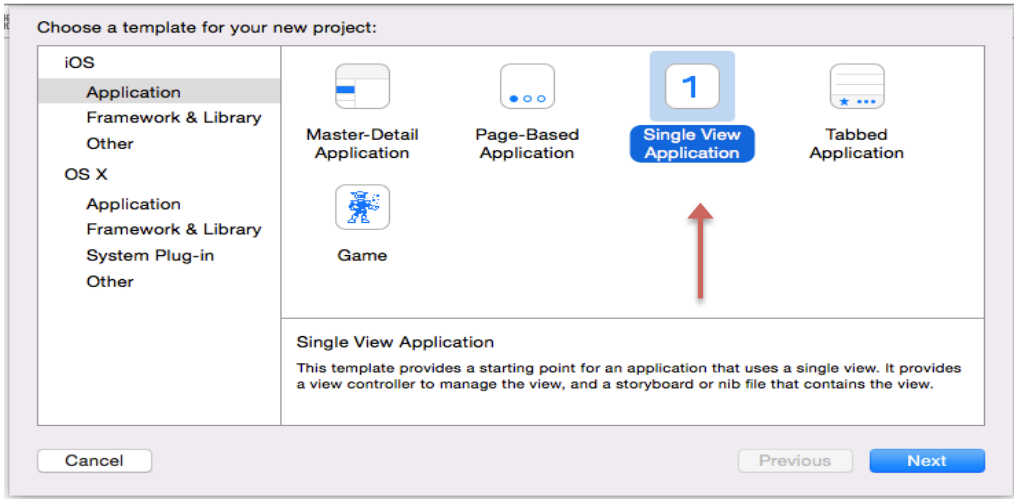
- GET: consulta y lee recursos
- POST: crea nuevos recursos y envía información para procesarla en el servidor
- PUT: editar recursos
- DELETE: eliminar recursos

Creación de la aplicación

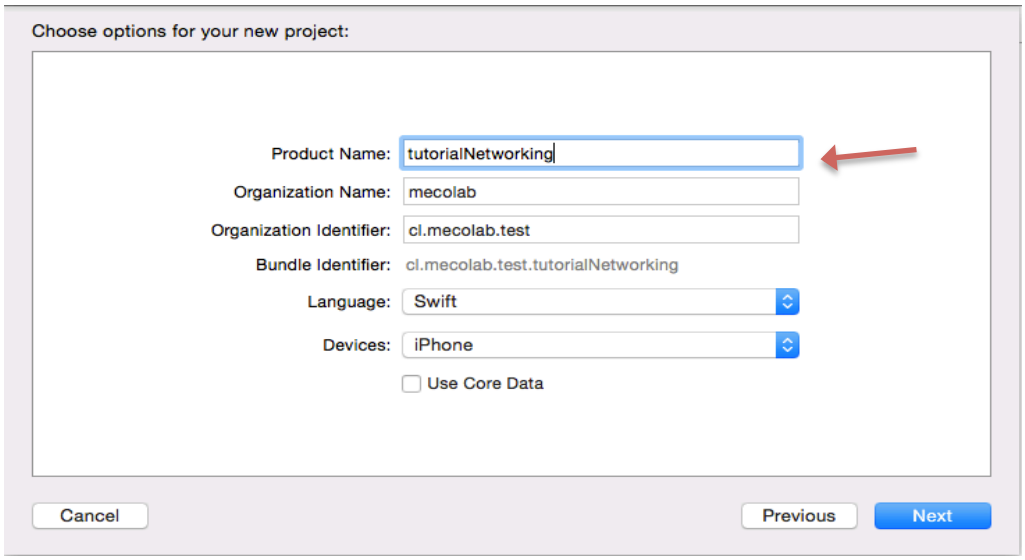
La aplicación que se presenta en este tutorial consiste en un GET y POST. Para interactuar con el servidor, es necesario una interfaz gráfica funcional que pueda realizar los Request. Para esto dividiremos la aplicación en dos: en la primera parte se realiza el GET y en la segunda el POST.

Interfaz gráfica GET Request

La primera parte de la aplicación corresponde a un visualizador de imágenes provenientes de Google. Se ingresa una palabra y se obtienen las primeras cuatro imágenes que aparecen en este sitio web. Para esto se usará una interfaz sencilla que se describe a continuación:
Primero que todo se creará un proyecto. Se escoge una “Single View Applicacion” de iOS.



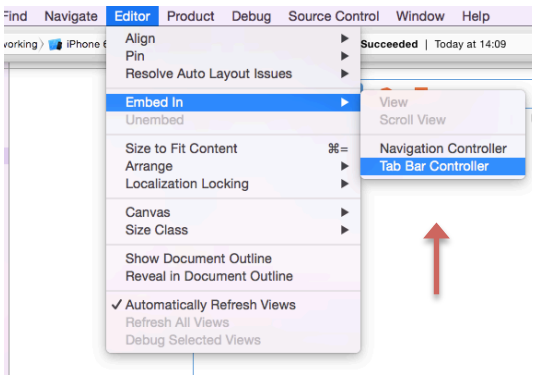
Se escribe el nombre de la aplicación, esta se llamará “tutorialNetworking”. Se escoge el lenguaje Swift.



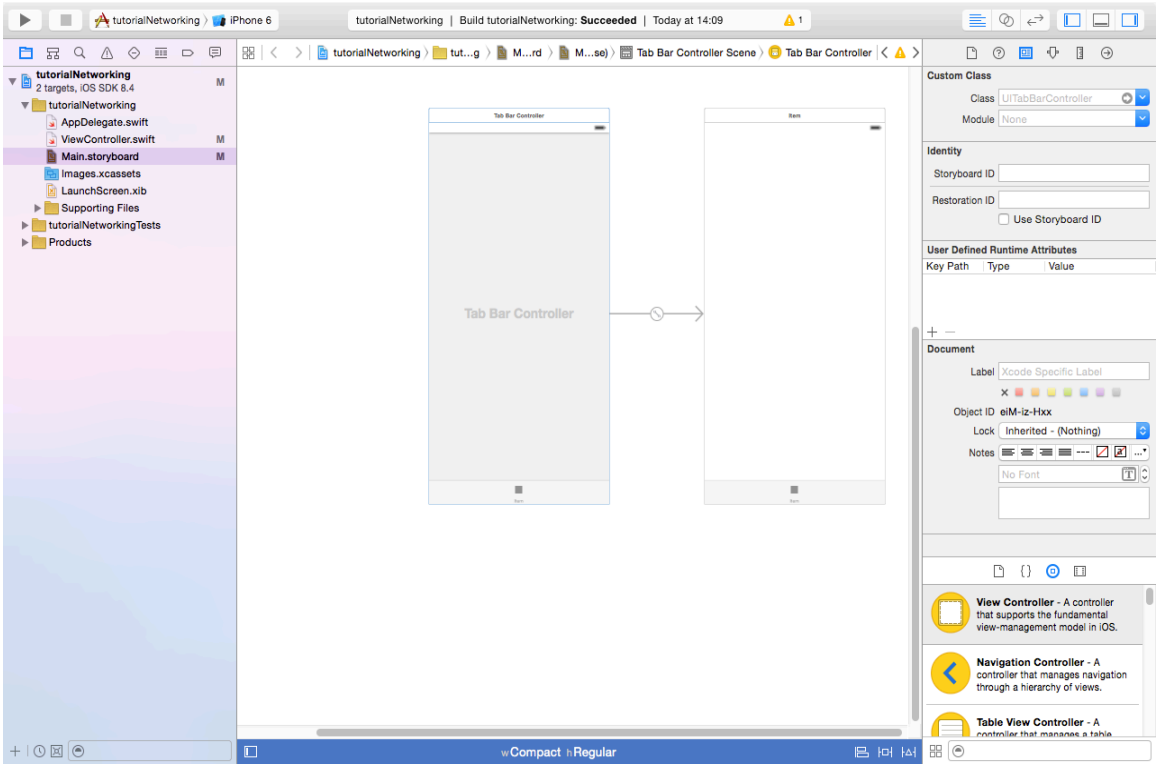
Una vez que se haya creado el proyecto, aparecerá un archivo llamado “Main.storyboard” el cual será responsable de la interfaz gráfica. Es evidente notar que existe un “View Controller“ por defecto, se usará este para realizar el GET Request. Para continuar con la aplicación se agregarán los siguientes elementos a este controlador:

- Text Field : para escribir la palabra a buscar en la web
- Label : etiqueta que dirá el número de imagen que es mostrado
- Button “Search” : realiza el request con la palabra del Text Field
- Tab Bar Controller : con esta herramienta cambiaremos de GET a POST
- UIImageView : mostrará la imagen
- Dos UIButton “Right” y “Left” : para ver imágenes.
- UIActivityIndicatorView : este elemento se muestra cuando se están descargando las imagenes (se debe seleccionar la propiedad “Hidden when stopped”).

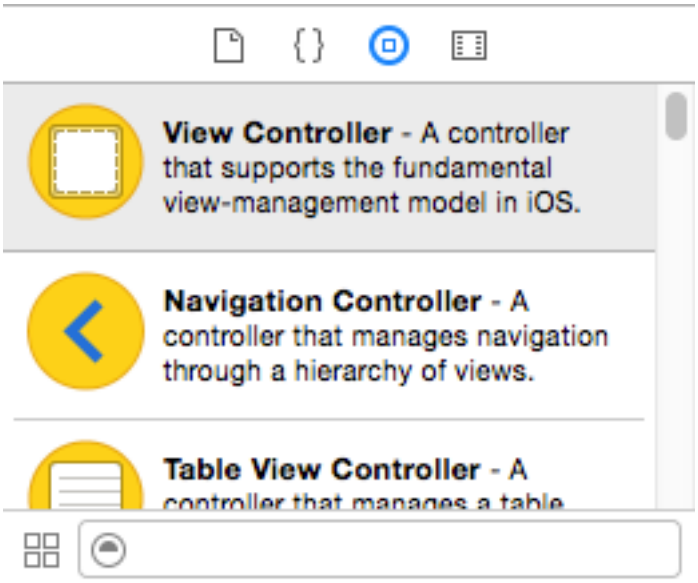
Lo primero que se hará es agregar un “Tab Bar Controler”. Seleccionamos el View Controller que viene por defecto y vamos a la barra de herramientas que está en la parte superior de la pantalla. Vamos a la pestaña Editor -> Embed In -> Tab Bar Controller.



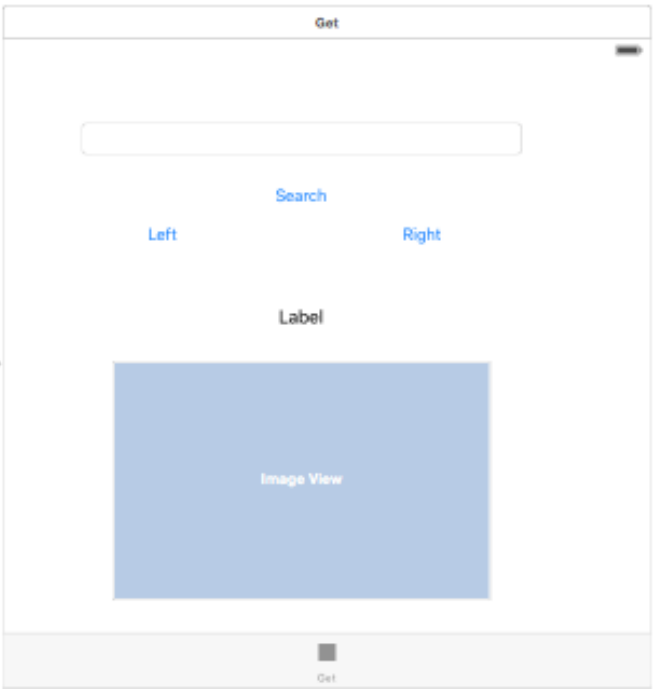
Al seleccionarlo, aparecerá el Tab Bar Controller en la pantalla.



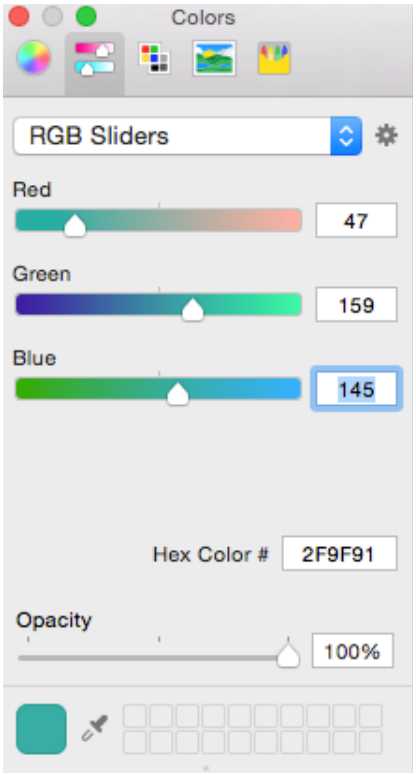
El siguiente paso es agregar los elementos que fueron mencionados antes (Label, Button, etc...). Esto se realiza con una barra de herramientas propia de Xcode la cual tiene todos los elementos para interactuar con el celular. Esta se encuentra por defecto en esquina inferior derecha.
Se busca en esta barra el elemento que se desea implementar a la aplicación y se agrega con "Drag and Drop".



Se debe agregar un UILabel, tres UIButton, un UITextField y un UIImageView. Se hace doble click sobre los elementos para cambiar el nombre.

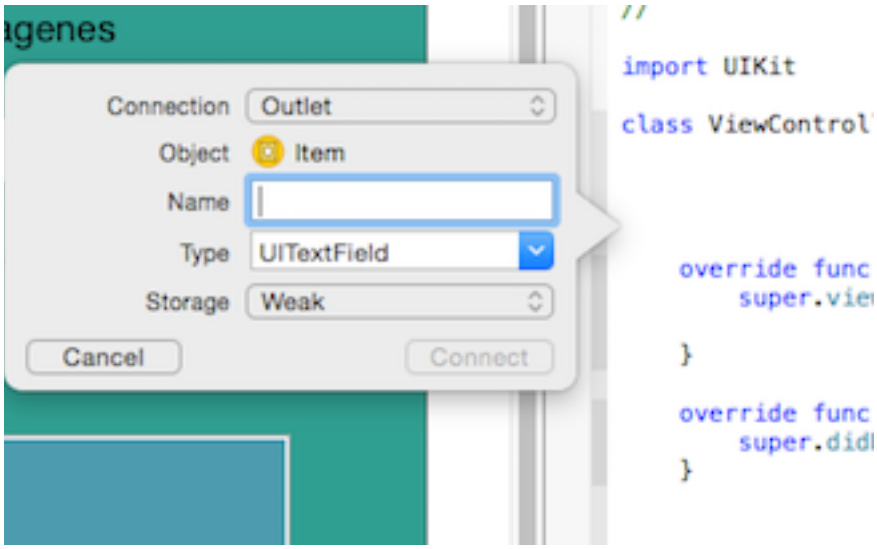


La aplicación se puede personalizar con distintos colores. Para cambiar el color del background se selecciona el View Controller y en la pestaña de atributos hay que dirigirse donde dice “Background” y cambiar la tonalidad.



Para poder controlar los elementos de la aplicación, se deben generar enlaces entre un controlador y una clase. Para hacer eso, se necesita que el View Controller este administrado desde la clase “View Controller”. Hay que verificar que este escrita la clase correcta, de no ser así, escribirla.

Luego se debe generar un Outlet, el cual es el vínculo entre un elemento y la clase. Se realiza un Outlet del Text Field y de UIImageView. Para crear este enlace se debe mantener presionada la tecla “ctrl” y seleccionar un elemento del View Controller, después se debe arrastrar el cursor hasta la clase donde esté suscrito el controlador y soltarlo. Aparecerá un cuadro como el siguiente, donde se escribe el nombre del elemento.



El UITextField se llamará “text”, el UIImageView “image”, el UIActivityIndicator “activityIndicator” y el UIButton “Search”. Continuando con la aplicación, se debe generar una lista llamada “images” la cual contendrá los cuatro URL de las imágenes que se mostrarán y otra lista “imagesSaved” la cual tendrá las imágenes en NSData (objeto que puede guardar data). También se debe tener una variable llamada “presentImage”, la cual tendrá el valor de la posición del número de las cuatro imágenes que hay (del 0 al 3) y se define otra variable llamada “emptyImage”, esta contiene una instancia vacía de NSData para representar que no hay imágenes.

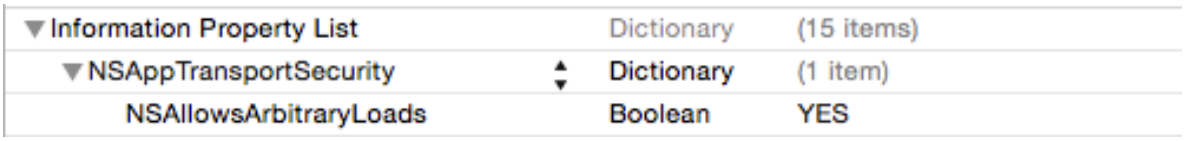
```
@IBOutlet weak var text: UITextField!
@IBOutlet weak var image : UIImageView!
@IBOutlet weak var presentImageLabel: UILabel!
@IBOutlet weak var activityIndicator: UIActivityIndicatorView!

var images: [String]!
var presentImage = 0
let emptyImage = NSData()
```

Se declara las variables “var” si son mutables y “let” si son inmutables. Ahora se procede a hacer el método que hará el GET request.

Código GET request

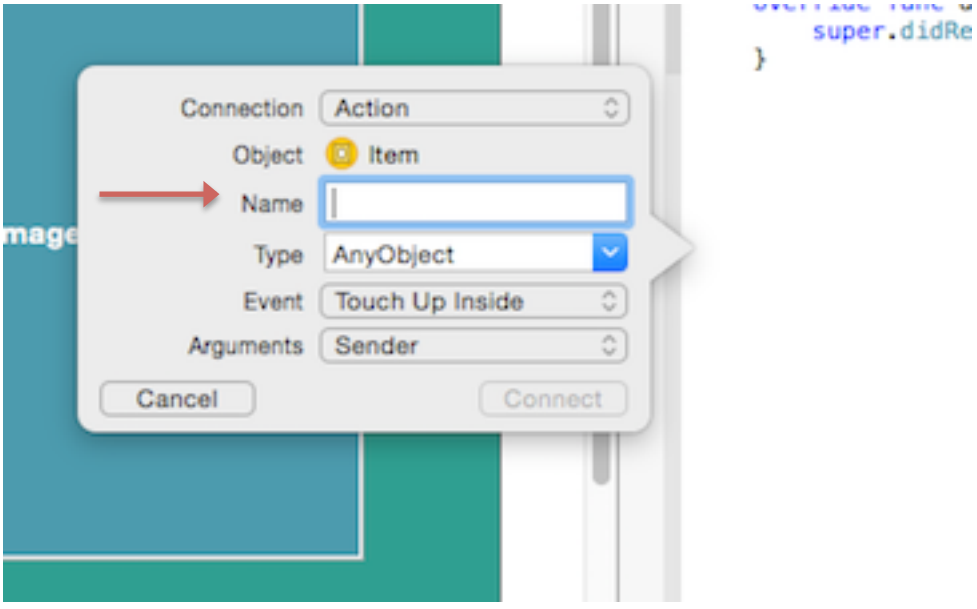
El sistema operativo iOS 9 tiene más restricciones de seguridad que los anteriores, es por eso que para acceder a contenido en internet o descargar archivos, se debe declarar una autorización. Es por eso que para continuar con el tutorial, es necesario que se agregue en Info.plist un campo llamado NSAppTransportSecurity el cual será un diccionario. A este se le debe agregar el elemento NSAllowsArbitraryLoads, el cual será de tipo Bolean y trendrá valor YES.



Cuando se abre el View Controller por primera vez se ejecuta el método viewDidLoad, por lo tanto se instancia la lista “images” e “imagesSaved”:

```
override func viewDidLoad() {
        super.viewDidLoad()
        self.images = []
         self.imagesSaved = [self.emptyImage, self.emptyImage, self.emptyImage,
emptyImage]
    }
```

El Request se hará por medio del UIButton “Search”, por lo cual se realiza una conexión entre el View Controller y la clase. Como ahora se necesita activar un evento al apretar el botón, no se realiza un “Outlet”, si no que un “Action”. Esto se crea con el mismo procedimiento que el outlet del Text Field, pero en la parte que dice “Connection”, ahora se selecciona “Action”. Se llamará al método “get”, este obtendrá los URL de las imágenes.



El activityIndicator empieza su animación.

```
self.activityIndicator.startAnimating()
```

Se resetea presentImage y las listas de URL y NSData.

```
self.presentImage = 0
self.images = []
self.imagesSaved = [self.emptyImage, self.emptyImage, self.emptyImage,
emptyImage]
```

Se utiliza la siguiente URI:

https://ajax.googleapis.com/ajax/services/search/images?v=1.0&q=search

Donde "search" es lo que se buscará. Esta URI retorna un JSON con toda la información del request.

Se instancia la URI con la clase NSURL:

```
let url = NSURL(string:
https://ajax.googleapis.com/ajax/services/search/images?v=1.0&q=search)
```

Ahora se debe crear el Request. Para esto se usa la clase NSMutableURLRequest, la cual toma como parámetro un URL (el cual ya fue instanciado). Se debe agregar "!" para especificar que la variable nunca será nula, en cambio si la variable puede ser nula se agrega "?". Las variable con "?" se llaman opcionales y aquellas con "!" se llaman desempaquetadas.

```
let request = NSMutableURLRequest(URL:url!)
```

El siguiente paso es explicitar que se desea realizar un GET Request. Esto se hace mediante el atributo "HTTPMethod" de la clase NSMutableURLRequest.

```
Request.HTTPMethod = "GET"
```

(Este paso no era necesario ya que el método GET viene por defecto)

La función que realiza la solicitud trabaja de manera asíncrona . Recibe como parámetro el request instanciado anteriormente y una cola. Al ejecutarse este método se lanza una función lambda que entrega los datos recibidos, una respuesta y un error en caso de que exista.

```
NSURLConnection.sendAsynchronousRequest(request, queue:
NSOperationQueue.mainQueue()) {(response, data, error) in }
```

En caso de que no se ejecute correctamente el Request, se lanzará el error y el parámetro "data" estará vacío. En cambio, si la petición se ejecuta de manera satisfactoria, se

dispondrá de la información. A pesar de que se obtiene un formato JSON, el parámetro data no lo entrega como tal. Esto requiere un paso intermedio para convertir esta variable en el archivo deseado.

```
let jsonSwift: AnyObject? = try? NSJSONSerialization.JSONObjectWithData(data!,
options: NSJSONReadingOptions.MutableContainers)
```

Este método transforma "data" en un JSON. Los parámetros que recibe son "data" (obtenido del Request), la forma en la cual se leerá el "data" (options) y una variable de error, la cual se ejecuta si el método no actúa de forma correcta.

Con los pasos descritos anteriormente se obtiene la variable jsonSwift con el JSON de request. El JSON tendrá la siguiente forma:


```
{
    "responseData": {
        "results": [
            {
                "GsearchResultClass": "GimageSearch",
                "width": "1600",
                "height": "1100",
                "imageId": "ANd9GcTKy_yX7H9eXxzUZ48croPRzxA_t6D2jHQYnC5uaUAA7vkME20N96Q5nZSm",
                "tbWidth": "150",
                "tbHeight": "103",
                "unescapedUrl": "http://gpi-blog.s3.amazonaws.com/wp-content/uploads/2014/03/casa.jpg",
                "url": "http://gpi-blog.s3.amazonaws.com/wp-content/uploads/2014/03/casa.jpg",
                "visibleUrl": "www.wattpad.com",
                "title": "I&#39;m not a normal girl™ - cap 1: &quot; <b>casa</b> nueva&quot; - Page 1 - Wattpad",
                "titleNoFormatting": "I&#39;m not a normal girl™ - cap 1: &quot; casa nueva&quot; - Page 1 - Wattpad",
                "originalContextUrl": "http://www.wattpad.com/49733408-i'm-not-a-normal-girl%E2%84%A2-cap-1-casa-nueva",
                "content": "1: &quot; <b>casa</b> nueva&quot; - Page 1",
                "contentNoFormatting": "1: &quot; casa nueva&quot; - Page 1",
                "tbUrl": "
http://t0.gstatic.com/images?q=tbn:ANd9GcTKy_yX7H9eXxzUZ48croPRzxA_t6D2jHQYnC5uaUAA7vkME20N96Q5nZSm"
            },
            {
                "GsearchResultClass": "GimageSearch",
                "width": "1250",
                "height": "1000",
                "imageId": "ANd9GcRCY2rATXtPAX0OM7iYdKqJUEkBxR21hpHM9orPHoFnyGyuSK7ZffRsXHYT",
                "tbWidth": "150",
                "tbHeight": "120",
                "unescapedUrl": "
http://goplaceit.s3.amazonaws.com/fotos_corredoras/chile/Inmobiliarias/PD_FOTO_1266_CASA_CIPRES_ok_7401C.jpg",
                "url": "
http://goplaceit.s3.amazonaws.com/fotos_corredoras/chile/Inmobiliarias/PD_FOTO_1266_CASA_CIPRES_ok_7401C.jpg",
                "visibleUrl": "www.goplaceit.com",
                "title": "<b>Casa</b> en Venta en Calle San Carlos Sur 1939, Puente Alto | Goplaceit",
                "titleNoFormatting": "Casa en Venta en Calle San Carlos Sur 1939, Puente Alto | Goplaceit",
                "originalContextUrl": "
http://www.goplaceit.com/cl/propiedad/venta/casa/puente-alto/1131301-en-venta-nueva",
                "content": "<b>Casa</b> en Venta en Calle San",
                "contentNoFormatting": "Casa en Venta en Calle San",
                "tbUrl": "
```


Para obtener las imágenes de la API, nos interesa la URL de estas. La forma de obtener este dato es a través del campo "responseData" el cual posee otro campo llamado "results". Aquí se encuentra un arreglo de largo cuatro, donde cada elemento es una imagen con toda su información. El campo "url" es el que posee la dirección que se necesita para visualizar la imagen.

Ahora para continuar con esta aplicación, debemos manipular el JSON y así obtener el URL. Se puede hacer una analogía entre un campo y su valor como un diccionario, donde la llave es el nombre del campo (String) y el valor es un NSObject, ya que no se puede saber a priori el tipo de esta variable.

Por lo tanto se hace un casting sobre jsonSwift para transformarlo a diccionario:

```
if let jsonDiccionario = jsonSwift as? Dictionary<String,NSObject>{}
```

En caso de que se pueda realizar el casting, se procede con el algoritmo. El siguiente paso es obtener el valor del campo “responseData”:

```
var responseData = jsonDiccionario["responseData"] as!
Dictionary<String,NSObject>
```

Cabe mencionar que a este campo también se le debe realizar un casting a diccionario, ya que tiene esa estructura. En este diccionario existe una llave (campo) llamado “results”, el valor de este campo es el que interesa para continuar con la aplicación.

```
var results = responseData["results"] as! NSArray
```

Esta variable tiene la información relativa de las cuatro imágenes del Request. Se hace un casting como un arreglo, ya que cada elemento del arreglo es una imagen.
Se trabaja con cada elemento del arreglo y se realiza un casting para formar un diccionario. La llave “url” entregará el URL de la imagen desde Google. Se guarda el link en la lista imágenes.

```
for element in results {
    if let imageJson = element as? Dictionary<String,NSObject> {
       let urlImage = ImageJson["url"] as! String
       self.images.append(urlImage)
    }
}
```

Terminado este proceso, se deben descargar las imágenes y resetear el contenido del UITextView.

```
self.download()
self.text.text = ""
```

Codigo get:

```
@IBAction func get(sender: AnyObject) {
    self.activityIndicator.startAnimating()
    self.presentImage = 0
    self.images = []
    self.imagesSaved = [self.emptyImage, self.emptyImage, self.emptyImage,
emptyImage]
    let search = self.text.text

    let url = NSURL(string :
"https://ajax.googleapis.com/ajax/services/search/images?v=1.0&q="+search!)

    let request = NSMutableURLRequest(URL : url!)
    request.HTTPMethod = "GET"

    NSURLConnection.sendAsynchronousRequest(request,
queue: NSOperationQueue.mainQueue()) {(response, data, error) in

       if error != nil || search == "" {

          let alert = UIAlertController(title: "UPS",
message: "An error occurred", preferredStyle: UIAlertControllerStyle.Alert)

          alert.addAction(UIAlertAction(title: "Continue",
```

```
            style: UIAlertActionStyle.Default, handler: nil))

            self.presentViewController(alert, animated: true, completion: nil)
            self.activityIndicator.stopAnimating()
        }
        else {

        let jsonSwift: AnyObject? = try?
NSJSONSerialization.JSONObjectWithData(data!,
options: NSJSONReadingOptions.MutableContainers)

        if let jsonDiccionario = jsonSwift as? Dictionary<String,NSObject> {

            let responseData = jsonDiccionario["responseData"] as!
Dictionary<String,NSObject>
            let results = responseData["results"] as! NSArray
            for element in results {
                if let imageJson = element as? Dictionary<String,NSObject> {
                    let urlImage = imageJson["url"] as! String
                    self.images.append(urlImage)
                }
        }
        self.download()
    }
   }
  }
 self.text.text = ""
}
```

El metodo download descarga de forma síncrona las imágenes. Se verifica que los String almacenados en la lista “images” pueden instanciarse como NSURL.

```
if let url = NSURL(string: self.images[index]) {}
```

Se verifica si se puede descargar la información en NSData.

```
if let data = NSData(contentsOfURL: url) {}
```

En caso de que lo anterior se pueda realizar, se guarda la NSData en “imagesSaved” y además, cuando se descarga la primera imagen se muestra en la aplicación y se debe cambiar el nombre al UILabel indicando el número de la imagen. Se debe detener el “activity indicator.

```
func download() {
     for index in 0...3 {
         print(index)
            if let url = NSURL(string: self.images[index]) {
               if let data = NSData(contentsOfURL: url) {
                  if index == 0 {
                     self.imagesSaved[index] = data
                     self.image.image = UIImage(data:self.imagesSaved[index])
                  }
                  else {
                       self.imagesSaved[index] = data
                  }
               }
            }
        }
```

```
     self.presentImageLabel.text = " 1 of 4"
     self.activityIndicator.stopAnimating()
    }
```

Antes de terminar se debe programar el código para adelantar y retroceder imágenes. Se crea un método llamado “right” y uno “left”, los cuales no recibe parámetros. Para el caso de “right”, se debe avanzar hacia la siguiente imagen aumentando en uno el contador de la variable “presentImage”.

```
if self.presentImage < 3 {
      self.presentImage += 1
}
else {
      self.presentImage = 0
}
```

Si se realiza descargaron las imágenes de forma adecuada, la lista “imagesSaved” tendrá los NSData de las imágenes. Se obtiene una imagen a partir de la información en “imagesSaved” y se muestra en pantalla. En caso contrario se mostrará un error.

```
@IBAction func right(sender: AnyObject) {
        if self.presentImage < 3 {
            self.presentImage += 1
        }
        else {
            self.presentImage = 0
        }

        if let image = UIImage(data:self.imagesSaved[self.presentImage]) {
                self.image.image = image
                self.changePresentImageLabel()
        }
        else {
            print("Error")

            let alert = UIAlertController(title: "UPS",
message: "The image couldn´t be loaded",
preferredStyle: UIAlertControllerStyle.Alert)

            alert.addAction(UIAlertAction(title: "Continue",
style: UIAlertActionStyle.Default, handler: nil))

            self.presentViewController(alert, animated: true, completion: nil)
        }
    }
```

Además se implementa de la misma forma el metodo “Left” para retroceder imágenes.

```
    @IBAction func left(sender: AnyObject) {
        if self.presentImage > 0 {
            self.presentImage -= 1
        }
        else {
            self.presentImage = 3
        }

        if let image = UIImage(data:self.imagesSaved[self.presentImage]) {
            self.image.image = image
            self.changePresentImageLabel()
```

```
        }
        else {
            print("Error")

            let alert = UIAlertController(title: "UPS", message: "The image
couldn´t be loaded", preferredStyle: UIAlertControllerStyle.Alert)

            alert.addAction(UIAlertAction(title: "Continue", style:
UIAlertActionStyle.Default, handler: nil))

            self.presentViewController(alert, animated: true, completion: nil)
        }
    }
```

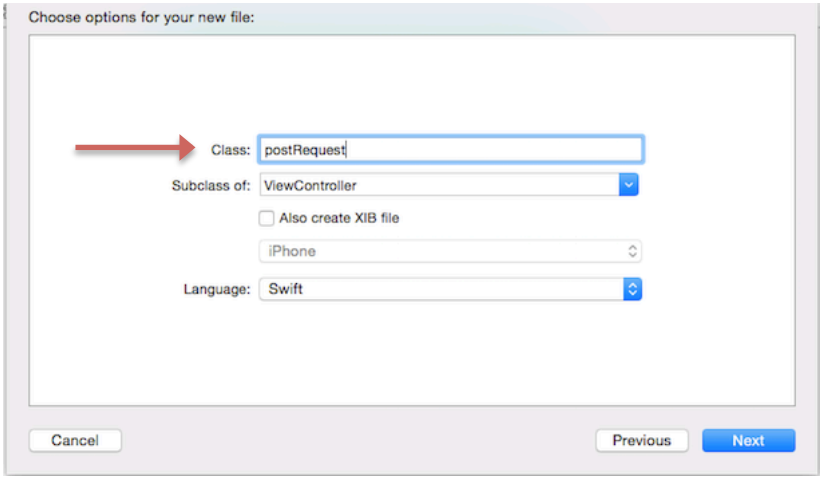
Cada vez que se cambia la imagen, se actualiza el UILabel con el numero de imagen, por lo que debe existir un método encargado de esto que dependa del valor de "presentImage".

```
  func changePresentImageLabel() {
        switch self.presentImage {
        case 0:
            self.presentImageLabel.text = " 1 of 4"
        case 1:
            self.presentImageLabel.text = " 2 of 4"
        case 2:
            self.presentImageLabel.text = " 3 of 4"
        case 3:
            self.presentImageLabel.text = " 4 of 4"
        default:
            break
        }
    }
```

Interfaz gráfica POST request

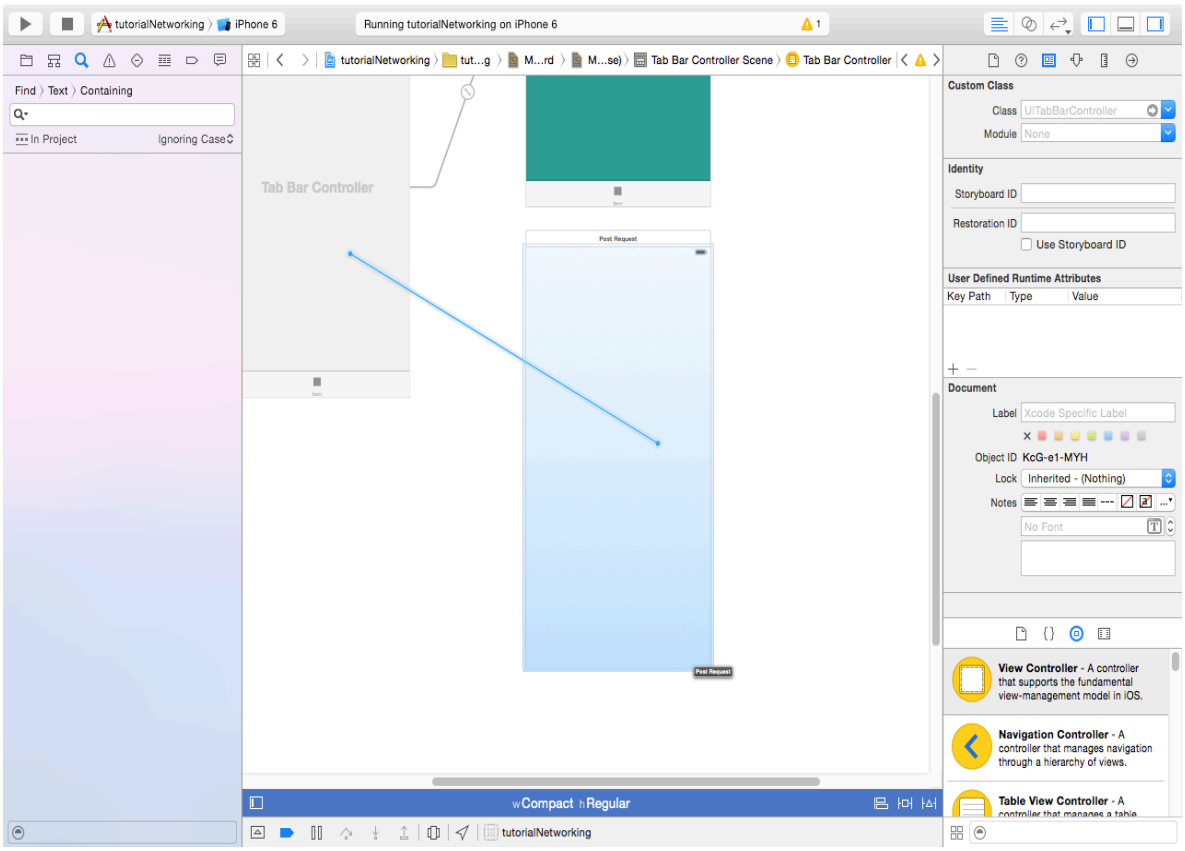
Ahora se realiza el segundo paso del tutorial. Para crear el POST request, se trabajará con la siguiente URI: https://www.codepunker.com/tools/http-requests .En esta API se pregunta "whoAreYou", el servidor procesa la información y responde a la pregunta. La forma de ejecutar el POST request tiene algunas similitudes y diferencias con el GET, las cuales se datallarán en el tutorial.

Primero que todo se creará una Cocoa Touch Class llamada "postRequest", que hereda de ViewController, la cual controlará el POST.

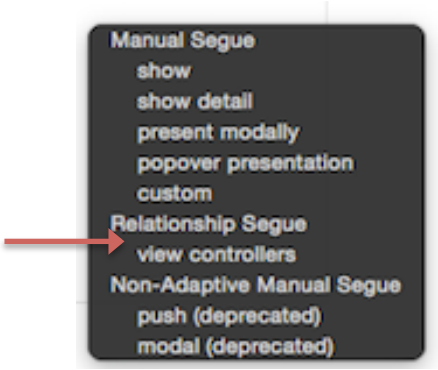


Ahora se crea el View Controller. Para esto arrastramos desde la librería de objetos un View Controller vacio. El paso siguiente será escribir el nombre de la clase “postRequest” en el apartado “Class” que aparece en “Show the identity inspector”.

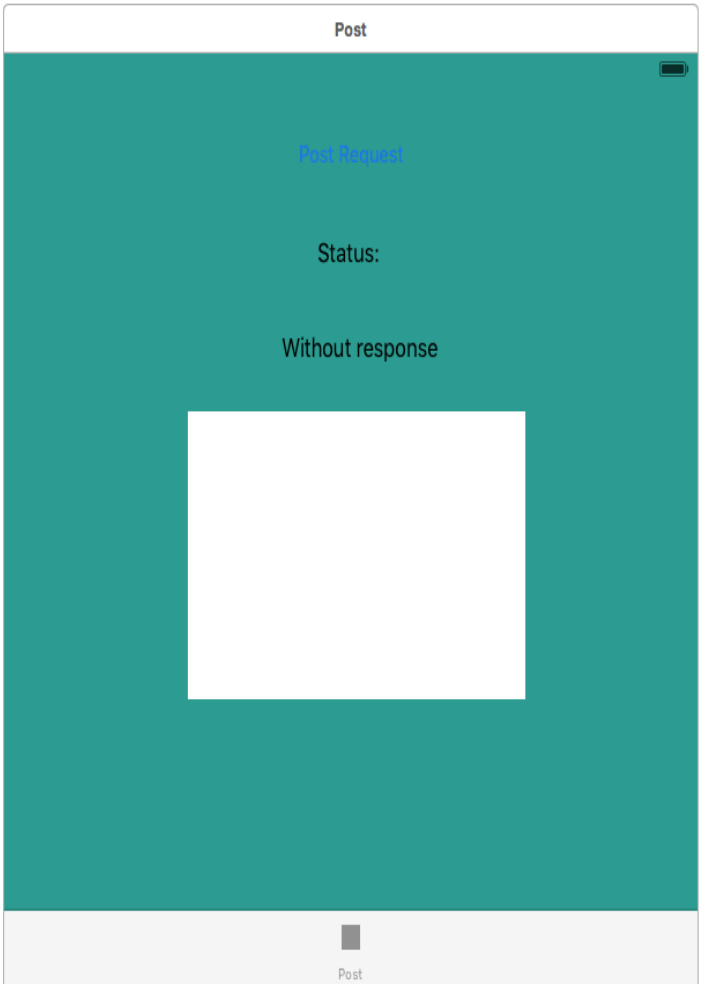
El siguiente paso es enlazar el View Controller nuevo con el Tab Bar Controller, esta conexión se conoce como “Segue”. Para esto se presiona “ctrl” sobre el Tab Bar Controller y después se mueve el cursor hasta el View Controller que se desea enlazar.



Al soltar el cursor aparecerá una cuadro, de este se debe seleccionar “view controllers”, el que se encuentra dentro de la opción “Relationship segue”



Realizado esto, se agrega un botón “Post Request”, un Label “Status”, un Label “Without response” y un UITextView. Se deben agregar estos elementos como se mostró en el tutorial del método GET. Se obtendrá una interfaz como la se observa:



Código POST request

Ahora se crean cuatro outlets: el outlet “status” corresponde a un UILabel, “textView” a UITextView.

```
@IBOutlet weak var status: UILabel!
@IBOutlet weak var textView: UITextView!
```

Además, se crearán dos variables que serán utiles para mostrar la información en la interfaz, las cuales guardarán la informacion que se mostrará en el Label “status” y en el UITextView “textView”.

```
var saveStatus : String!
var saveResponse : String!
```

Realizado esto, se procede con el código del Request. Lo primero que se debe hacer es instanciar el URL, lo cual se hace de la misma forma que en el método GET, pero con el nuevo URI

```
let request =
NSMutableURLRequest(URL:NSURL(string:"https://www.codepunker.com/tools/http-
requests")!)
```

El siguiente paso es explicitar que se desea realizar un POST Request (para diferenciarlo del GET). Por lo tanto en el atributo “HTTPMethod” se escribe “POST”

```
request.HTTPMethod = "POST"
```

Cuando se hace el request, se debe entregar la pregunta “whoAreYou”, lo cual debe ir en el Body del JSON. El Body es la sección que contiene la información que se procesará en el servidor y se entregará una respuesta, se compone de parámetros y valores. En este caso existen dos parámetros y valores. El primer parametro es “extra” y su valor es “whoAreYou”, el segundo parametro es “execute” y su valor es “demoRequests”.

Hay que tener cuidado al momento de escribir el Body porque la sintaxis es muy importante. La estructura es “parámetro=valor”. Para agregar el segundo parámetro se separa mediante el signo “&”, por lo tanto el Body sería “parámetro1=valor&parámetro2=valor2”. Por lo tanto el Body queda instanciado en forma de String.

```
let postString = "extra=whoAreYou&execute=demoRequests"
```

La variable postString que contiene el String del Body, se debe asignar al request. Para esto, usamos el atributo HTTPBody de la clase NSMutableURLRequest. La property HTTPBody acepta valores de tipo NSData, por lo cual se debe usar el método de la clase NSString “dataUsingEncoding”. Este método retorna un NSData a partir de un String en base a un encoding, en este caso se usará NSUTF8StringEncoding.

```
request.HTTPBody = postString.dataUsingEncoding(NSUTF8StringEncoding)
```

Para crear el request se usará el mismo método usado en el GET. Se le entrega un request y entrega “data”, “response” y “error”.

```
NSURLConnection.sendAsynchronousRequest(request,
queue: NSOperationQueue.mainQueue()) {(response, data, error) in }
```

Como el request es distinto, se recibe un "data" distinto al entregado en GET; en este caso el JSON entregado es mucho mas sencillo.

```
{
    "type": "success",
    "response": "Congratulations, you've just sent a POST request... The response is: My name is Daniel! Nice to meet you!"
}
```

La llave "type" entrega "success" si se realizó el POST de forma correcta y la llave "response" entrega la respuesta a la petición.

Al igual que en el método GET, el primer paso después de realizar el POST es obtener el JSON a partir de la variable "data", para esto se usa el método estático JSONObjectWithData de la clase NSJSONSerialization. Este método recibe los mismos parámetros que los usados anteriormente en el tutorial.

```
var json: AnyObject? = NSJSONSerialization.JSONObjectWithData(data,
options: .MutableLeaves, error: nil)
```

Si se ejecutó correctamente la petición, entonces se procede a trabajar sobre el JSON. Primero que todo se realiza un casting para transformar la variable "Json" a diccionario con llave de tipo "String" y valor "NSObject".

```
if let jsonDictionary =  json as? Dictionary<String,NSObject>
```

Si no se completo esta operación, se lanza una alerta avisando por eso.

```
let alert = UIAlertController(title: "UPS", message: "An error occurred",
preferredStyle: UIAlertControllerStyle.Alert)

alert.addAction(UIAlertAction(title: "Continue",
style: UIAlertActionStyle.Default, handler: nil))

self.presentViewController(alert, animated: true, completion: nil)
```

Si se completa satisfactoriamente esta acción, entonces se obtiene del JSON el valor contenido de la llave "type" y "response"

```
let jsonStatus = jsonDictionary["type"] as! String
let jsonResponse = jsonDictionary["response"] as! String
```

Habiendo obtenido el valor de las respectivas llaves, se asignan a las variables "saveStatus" y "SaveResponse" este valor.

```
self.saveStatus =  jsonStatus
self.saveResponse = jsonResponse
```

Ahora solo resta ejecutar el Request. Está solicitud se ejecutará cuando se oprima el botón "Post Request", por lo que se crea un "Action" a partir de este botón con el nombre "Post".

```
@IBAction func post(sender: AnyObject) {

    let request = NSMutableURLRequest(URL: NSURL(
    string : "https://www.codepunker.com/tools/http-requests")!)
```

```
    request.HTTPMethod = "POST"

    let postString = "extra=whoAreYou&execute=demoRequests"
    request.HTTPBody = postString.dataUsingEncoding(NSUTF8StringEncoding)

    NSURLConnection.sendAsynchronousRequest(request,
    queue: NSOperationQueue.mainQueue()) {(response, data, error) in

    if error != nil {
       print(error)

       let alert = UIAlertController(title: "UPS", message: "An error occurred",
preferredStyle: UIAlertControllerStyle.Alert)

       alert.addAction(UIAlertAction(title: "Continue", style:
UIAlertActionStyle.Default, handler: nil))

       self.presentViewController(alert, animated: true, completion: nil)
    }
    else {

       let json : AnyObject? = try? NSJSONSerialization.JSONObjectWithData(data!,
options: .MutableLeaves)

       if let jsonDictionary = json as? Dictionary<String,NSObject> {
          let jsonStatus = jsonDictionary["type"] as! String
          let jsonResponse = jsonDictionary["response"] as! String
          self.saveStatus =  jsonStatus
          self.saveResponse = jsonResponse
          self.textView.text = self.saveResponse
          self.status.text = self.saveStatus
       }
       else {

            let alert = UIAlertController(title: "UPS", message: "An error
occurred", preferredStyle: UIAlertControllerStyle.Alert)

            alert.addAction(UIAlertAction(title: "Continue", style:
UIAlertActionStyle.Default, handler: nil))

            self.presentViewController(alert, animated: true, completion: nil)
       }
    }
 }
```

Cuando se ejecute correctamente el request, se presentarán en pantalla los elementos obtenidos del JSON