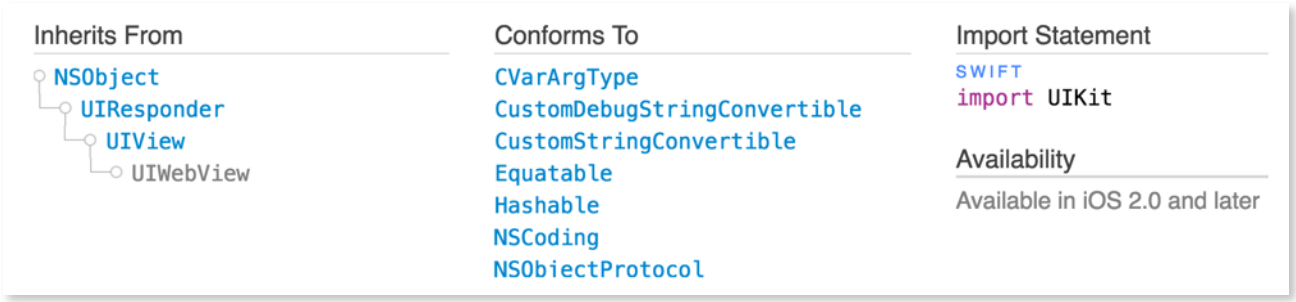
iOS 입문 강의

12강 <UI 기본 컨포넌트 (4) >

1. WebView

* 우리가 만들어온 UI들은 UIKit이라는 iOS 프레임워크에 포함되어있는 것을 객체로 만들어 사용하는 것이다.
  * 필요한 기능들에 대해 키워드를 구글에 영어로 검색해보면 많은 정보를 알 수 있다.



* WebView **실행 과정**

http://www.naver.com (String)

⇢ NSURL

⇢ NSURLRequest

⇢ WebView

(1) WebView : UIView

```
public var scrollView: UIScrollView { get }

 public func loadRequest(request: NSURLRequest)
 public func reload()
 public func stopLoading()
 public func goBack()
 public func goForward()
 public var canGoBack: Bool { get }
 public var canGoForward: Bool { get }
```

* WebView는 ScrollView를 기본으로 깔고 있다.
* loadRequest : Request를 받는다
* goBack / goForward : 페이지 단위로 앞 페이지 뒷 페이지로 가는 것
* canGoBack / canGoForward : 뒤 / 앞으로 갈 페이지가 있는지 체크
  * 앞 페이지, 뒷 페이지로 갈 버튼의 활성화 여부를 설정할 때 사용.

(2) UIWebViewDelegate

```
public protocol UIWebViewDelegate : NSObjectProtocol {
    @available(iOS 2.0, *)
    optional public func webView(webView: UIWebView,
shouldStartLoadWithRequest request: NSURLRequest, navigationType:
UIWebViewNavigationType) -> Bool
    @available(iOS 2.0, *)
    optional public func webViewDidStartLoad(webView: UIWebView)
    @available(iOS 2.0, *)
    optional public func webViewDidFinishLoad(webView: UIWebView)
    @available(iOS 2.0, *)
    optional public func webView(webView: UIWebView,
didFailLoadWithError error: NSError?)
}
```

* shouldStartLoadWithRequest : Request가 시작 되었을 때
* webViewDidStartLoad : 웹 뷰 로딩 시작된 후
* webViewDidFinishLoad : 웹뷰 로딩 끝난 후
* didFailLoadWithError : 웹뷰 로드 실패했을 때 처리 ex 안내메세지, 오류 화면 설정

(3) UIWebView - 예제

```
//텍스트 필드 객체 추가
let mainView:UIWebView! mainView:UIWebView = UIWebView(frame:
CGRect(x: 0, y:0, width: self.view.frame.size.width, height:
self.view.frame.size.height))

mainView.delegate = self
let url:NSURL? = NSURL(string:"http://naver.com")
mainView.loadRequest(NSURLRequest(URL:url))
self.view.addSubview(mainView)

//델리게이트 함수 작성
func webViewDidStartLoad(webView: UIWebView) {
     print("start")
}

func webViewDidFinishLoad(webView: UIWebView) {
     print("load complete")
}
```

* 객체 생성

* 델리게이트 설정

* 주소(String)을 NSURL로 변환

* NSURL을 Request로 변환

* addSubView

iOS 9 이 되며 바뀐 보안 이슈!

* 그냥 위의 코드를 보면 아래와 같은 오류가 뜰 것이다.

WebView[8488:1260848] App Transport Security has blocked a cleartext HTTP (http://) resource load since it is insecure. Temporary exceptions can be configured via your app's Info.plist file.

애플이 iOS9을 출시하면서 암호화 통신 없는 iOS 앱은 출시를 불가하게 하였다. 그래서 보안에 취약한 HTTP로 요청하면 실패하는 것이다.

해당 변경 사항에 대한 뉴스 기사

내년부터 웹과 연결되는 모든 iOS앱은 반드시 암호화 통신을 적용해야지만 애플 앱스토어에 올릴 수 있다.

15일(현지시간) 미국 지디넷, 테크크런치 등 외신에 따르면 세계개발자컨퍼런스(WWDC) 2016에서 애플은 '앱트랜스포트시큐리티(ATS)'를 연말까지 모든 앱에 의무적용한다는 계획이다. 앱과 서버가 정보를 주고받는 과정을 암호화해 프라이버시 보호를 강화한다는 것이다.

이에 따라 애플 앱스토어에 게재되는 모든 iOS앱들은 올해 말까지 ATS를 적용해야만 한다.

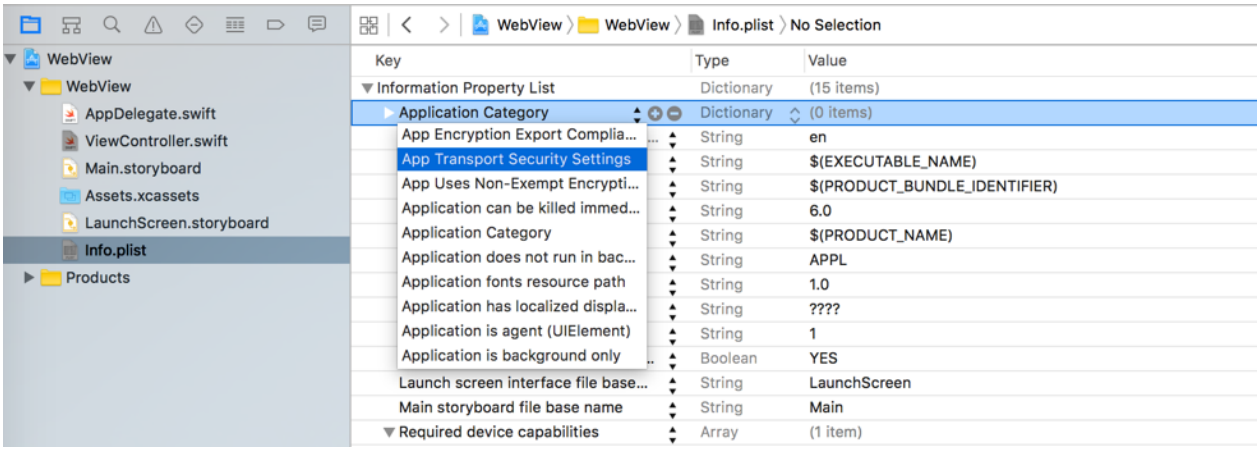
ATS는 지난해 iOS9을 발표하면서 처음 선보인 기능이다. 데이터를 평문 형태로 전송하는 대신 HTTPS(TLS)를 활용한 암호화 통신을 거쳐 전송되도록 하는 방법이다. 이를 통해 해커들이 iOS앱을 통해 송수신되는 데이터를 중간에서 가로채 악용하기 어렵게 만든다.

ATS를 적용한 iOS앱은 TLS 1.2 프로토콜로 암호화 통신을 하게된다. iOS9에서는 ATS가 기본설정으로 제공되고 있지만 개발자들은 ATS 기능을 해제하고, HTTP로 연결할 수 있었다.

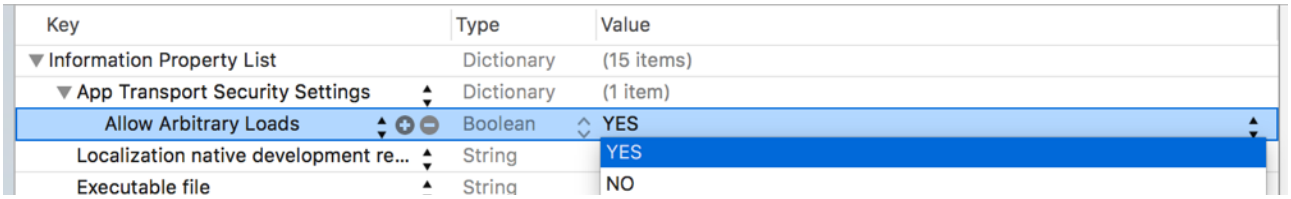
출처 - http://news.naver.com/main/read.nhn?mode=LSD&mid=sec&oid=092&aid=0002098293&sid1=001

< **해결 방법** >

* HTTP request **허용 설정**



info.plist **에서** Application Category**에서 + 버튼을 누르신 후**, App Transport Security Settiongs**을 선택 한다**.



이후, App Transport Security Settiongs**을 열고 (왼쪽 화살표가 아래로 향하게 하고)**

Value**를** YES**로 변경시켜준다**.

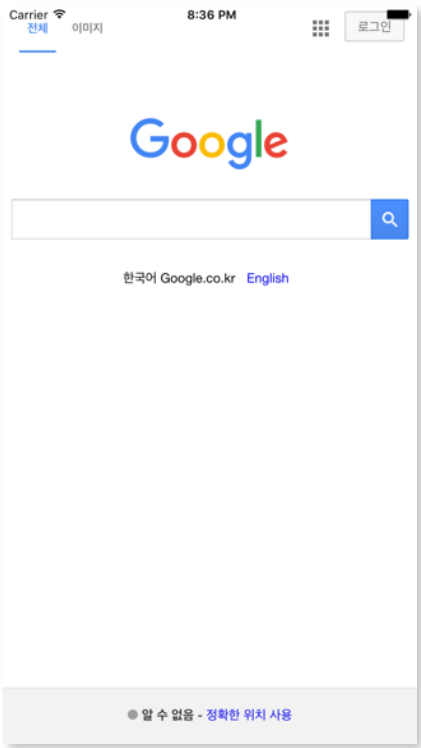
iOS 입문 화이팅 !

해당 설정을 해결하신 후에 실행해보시면 시뮬레이터에서 올바른 웹뷰 화면이 나옵니다.

- 예제 코드 -

```
import UIKit

class ViewController: UIViewController, UIWebViewDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()

        let webView:UIWebView = UIWebView(frame:self.view.
            bounds)
        webView.delegate = self
        self.view.addSubview(webView)

        webView.loadRequest(NSURLRequest(URL: NSURL(string:
            "http://www.google.com")!))
        // ! 붙이는 이유: string이 url이 아닐 경우도 있어서 옵셔널
    }

    func webViewDidStartLoad(webView: UIWebView) {
        print("start")
    }

    func webViewDidFinishLoad(webView: UIWebView) {
        print("load complete")
    }
```

url string -> NSURL -> NSURLRequest를 **한 줄로 합침**

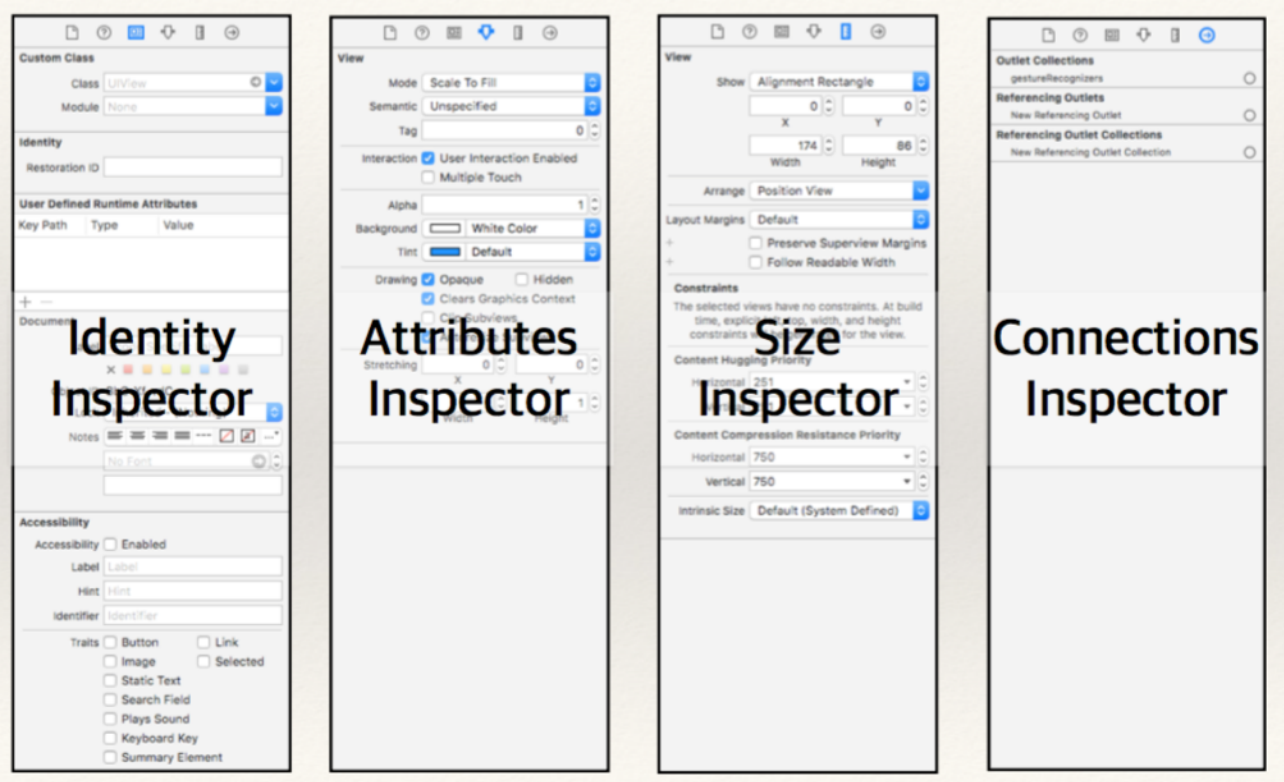


⇠ 시뮬레이터 화면

iOS 입문 화이팅 !

2. Storyboard

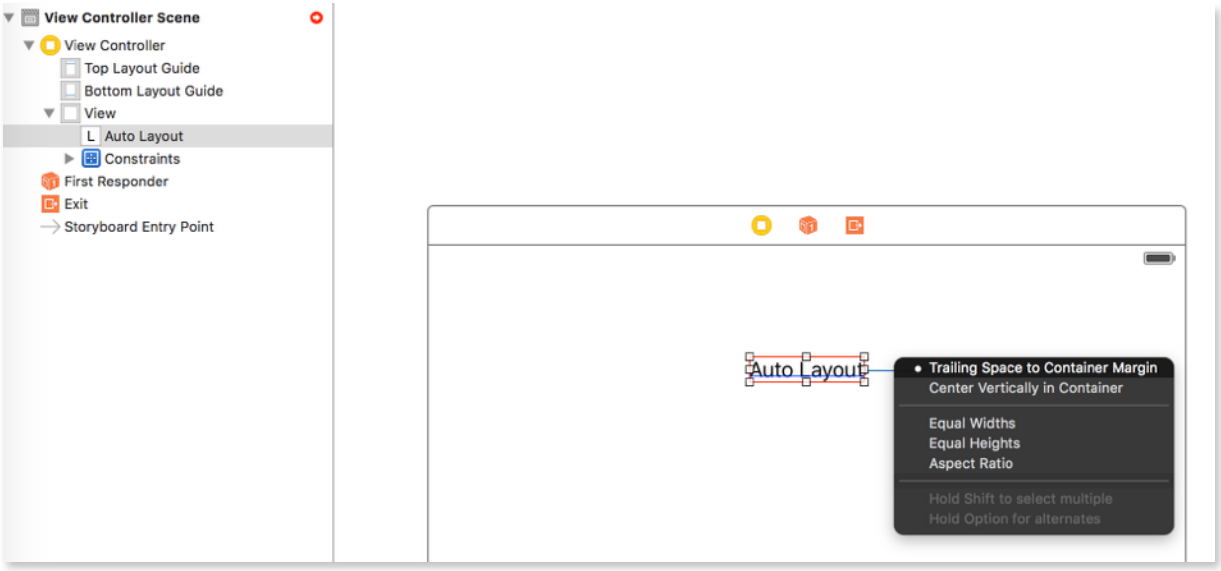
(1) Utilities - Inspector



* Identity Inspector
  * 객체 생성 부분
* Atrributes Inspector
  * UI 속성
  * 글씨 크기, 색깔 등
* Size Inspector
  * 절대적 크기
  * 오토레이아웃에 의한 관계형 크기
* Connetions Inspector
  * 객체의 관계도
  * 어떤 클래스에 어떤 이름으로 연결되어있는지

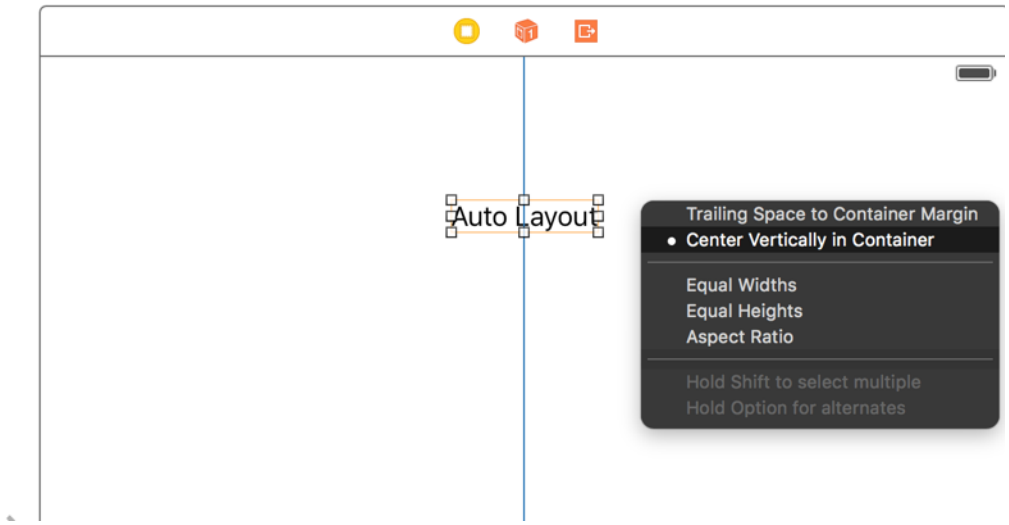
(2) Auto Layout

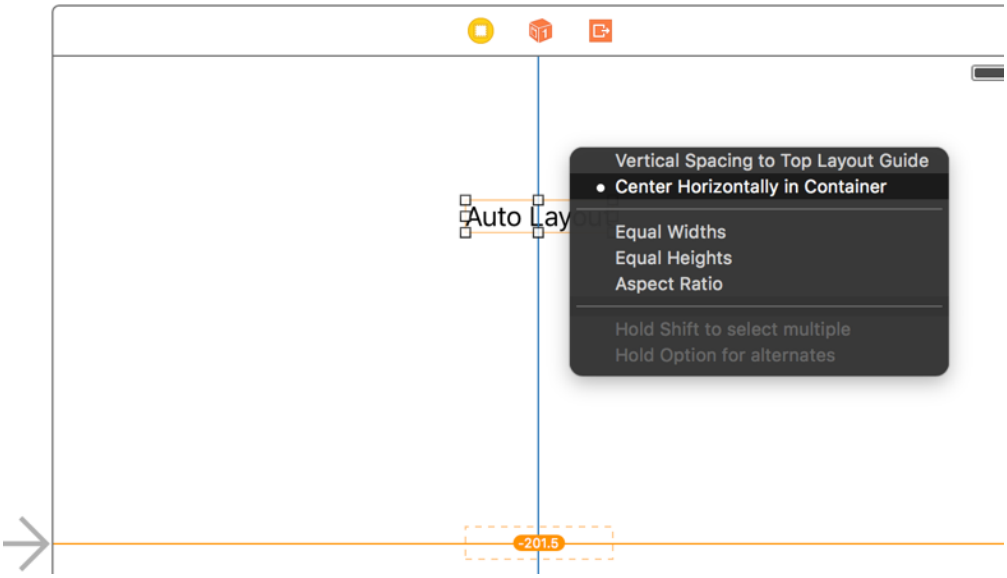
오토 레이아웃은 '제약사항 기반의 서술형 레이아웃 기술 언어' 이다.
반응형 웹과 비슷한 개념인데, 다양한 화면 크기와 언어 변화에 적응할 수 있는 UI를 만들기 위해 만들어진 기술이며, 절대적 크기가 아닌 부모 뷰와의 상대적 거리와의 간격을 기준으로 한다.



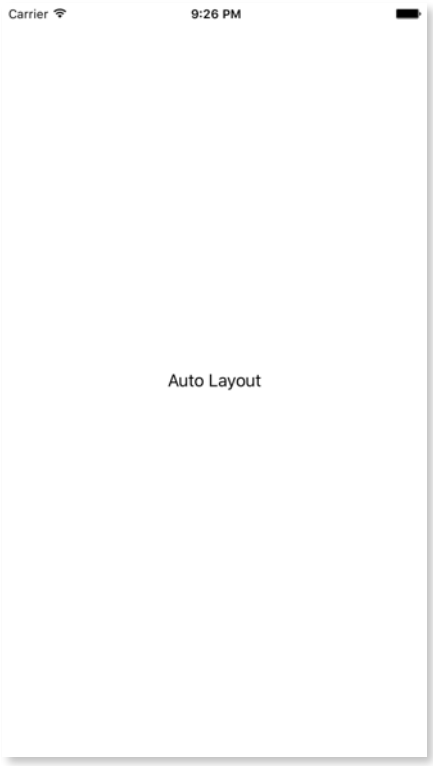
control 키를 누르고 끌면, margin 값을 정할 수 있다. 양쪽 가로, 세로의 간격을 정할 수 있다.

예를 들어, 정 중앙에 Label을 두고 싶으면



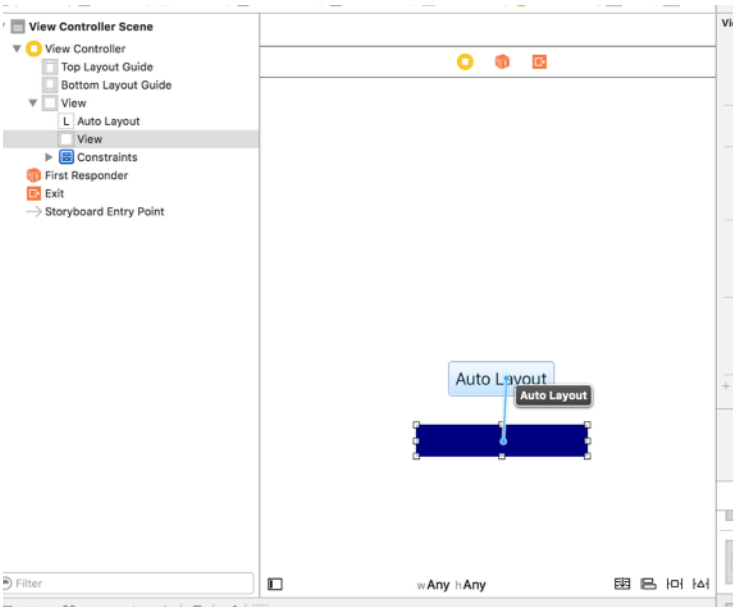


가로, 세로로 끌어 *Center Vertically in Container / Center horizontally in Container*로 **설정**

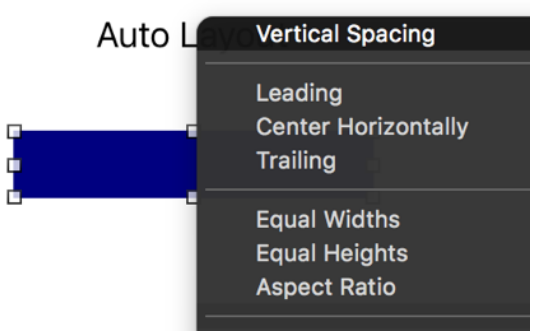


정 중앙에 맞춰집니다.

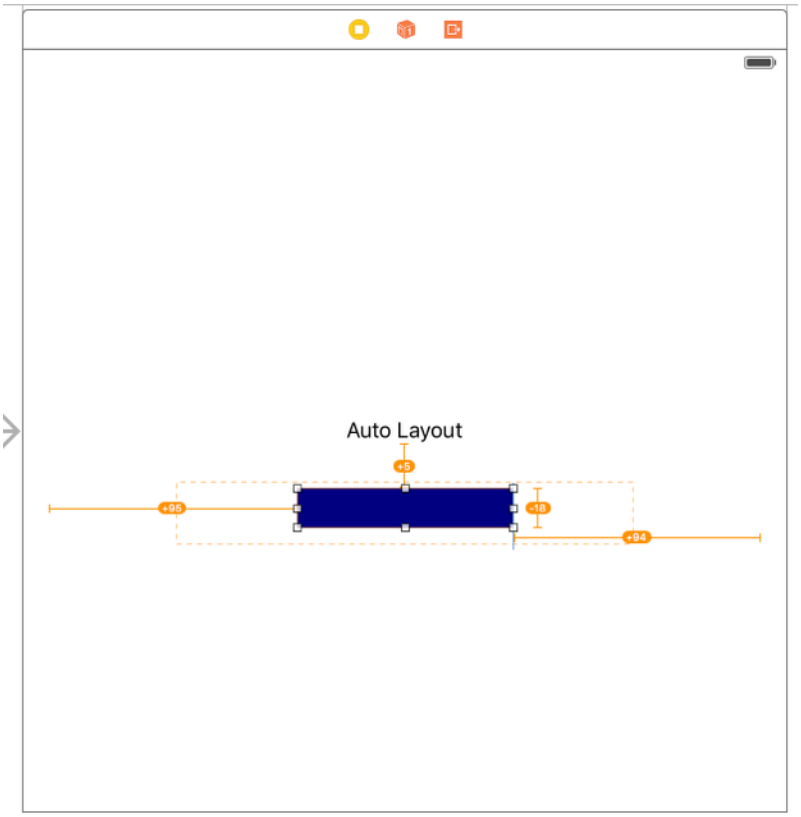
두 객체의 간격을 정하고 싶을 때에는



한 객체를 눌러, 간격을 정할 객체에 놓은 후, spacing을 정한다.



지금은 수직 관계이니 verical입니다.



같은 방법으로 원하는 대로 간격을 설정할 수 있습니다.

최종적으로 양쪽으로 간격을 100을 놓고, AutoLayout 라벨 거리를 10 정도 주었습니다.

시뮬레이터 화면

오토레이아웃은 많이 연습해보실수록 많은 것을 알게되요! 심오한 오토레이아웃의 세계.. 많이 연습해보세요!

iOS 입문 화이팅 !