

## 07 장.웹 뷰로 간단한 웹 브라우저 만들기

웹 뷰를 사용해 다양한 기능을 구현하는 웹 페이지를 만들어 보자. 텍스트 필드에 URL 주소를 입력하면 해당 주소로 이동할 뿐만 아니라 [Site1],[Site2]버튼을 누르면 다른 사이트로도 이동하도록 만들어 보자.

또한 웹 페이지 로딩 멈추기, 재로딩하기, 이전 페이지와 다음 페이지로 이동하기 등의 작업을 수행하는 웹 페이지도 만들어 보자.



## 07-1 웹 뷰란

웹 뷰(Web View)는 웹 콘텐츠를 뷰(View)형태로 보여 주는 앱이다.

다시 말해 익스플로러, 크롬 브라우저와 같이 HTML 로 작성된 홈페이지를 표시할 수 있다. 직접 인터넷에 연결된 주소를 입력하여 홈페이지에 접속할 수 있을 뿐만 아니라 미리 저장된 HTML 파일을 읽어들이 표시할 수도 있다.



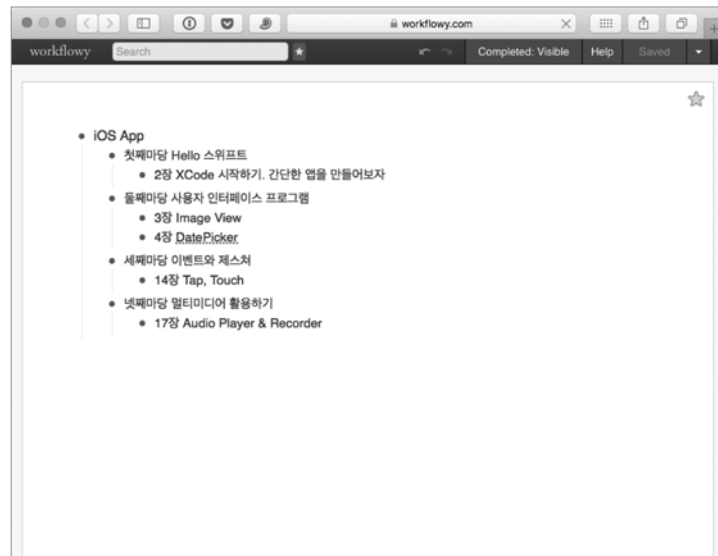
데이터베이스가 구축된 홈페이지라면 클라우드를 이용하지 않고도 웹 뷰를 이용하여 쉽게 데이터를 동기화할 수 있다.

예를 들면 웹 브라우저에서 workflow.com 에 접속하면 사용자의 에이터베이스에 저장된 내용이 나타나고, 워크플로워(workflow)앱은 같은 내용이 모바일에 맞추어 나타난다. workflow.com 의 경우 홈페이지가 반응형으로 작성되어 있기 때문에 앱에서는 자동으로 모바일 형식에 맞추어 표시된다.

이 처럼 웹을 통해 서버의 데이터베이스를 같이 사용하므로 웹과 앱에서 데이터가 동기화되는 효과를 얻을 수 있다.



workflowy 앱 화면



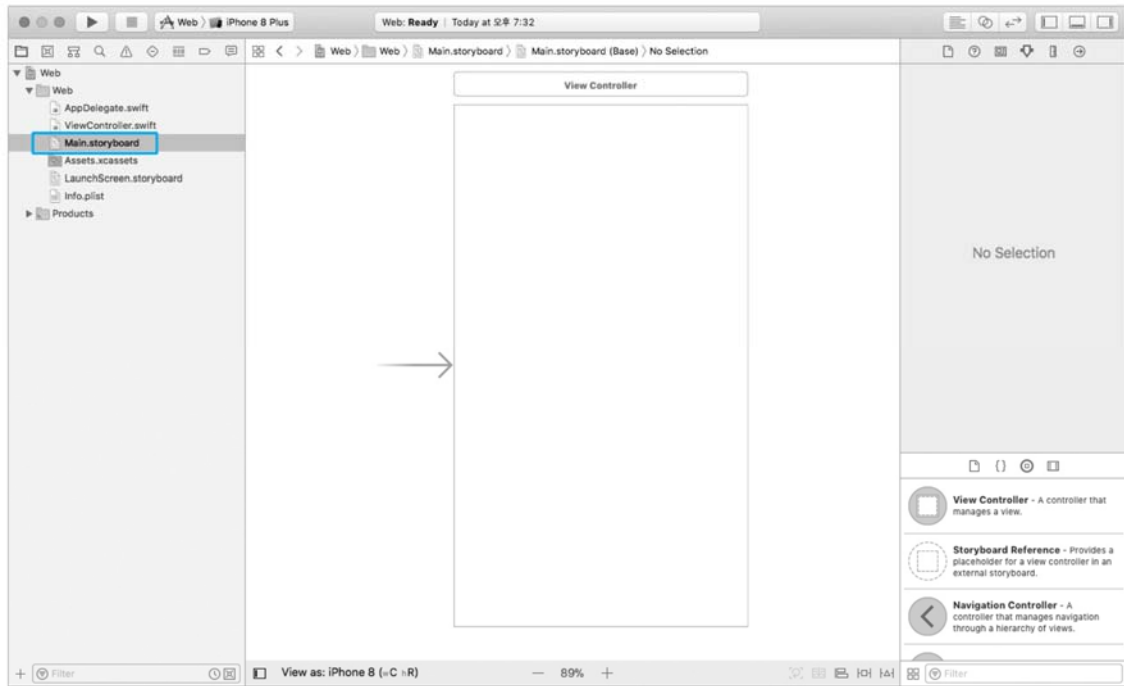
workflowy.com 웹 브라우저 화면

이외에도 웹 뷰를 이용하면 HTML 이나 파일을 이용하여 직접 웹 뷰에 디스플레이할 수 있다는 점도 기억하기 바란다. 만약 하이브리드 앱을 만들기 위한 HTML 파일을 가지고 있다면 이 파일들과 웹 뷰를 이용하여 하이브리드 앱이 아닌 iOS 전용 앱도 만들 수 있다.

## 07-2 웹뷰앱을위한기본환경구성하기

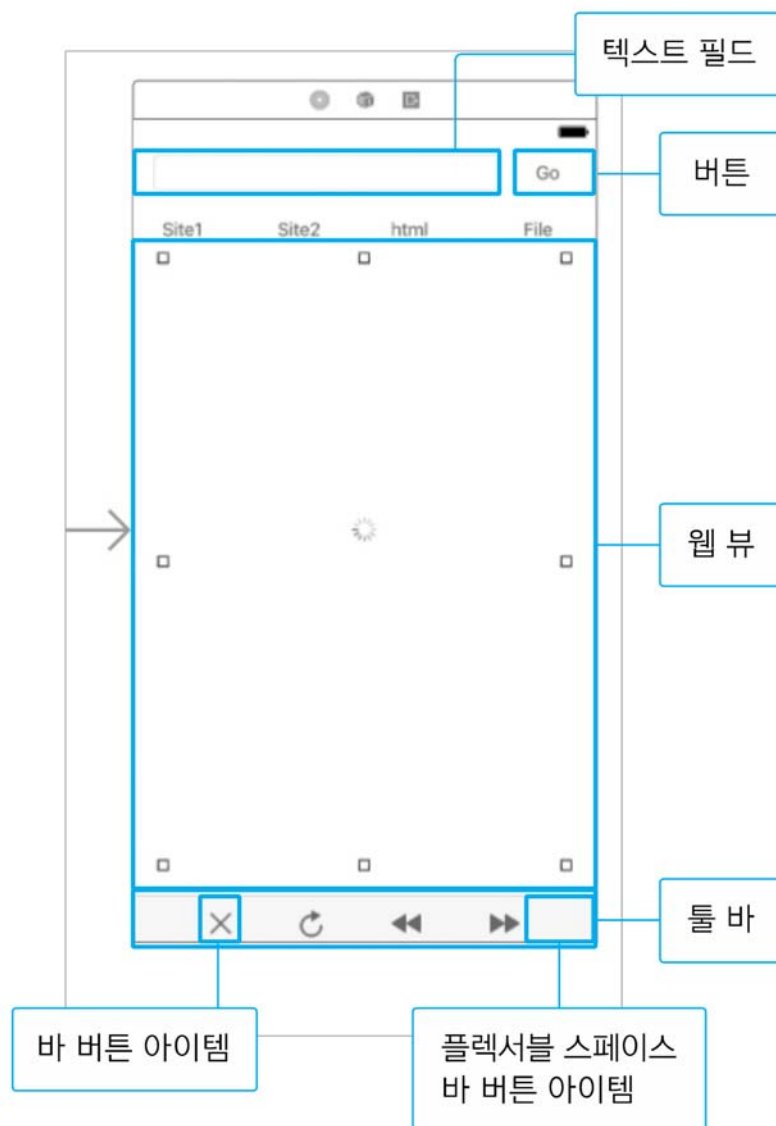
웹 뷰 앱을 구현하기 위해 프로젝트를 생성하고 스토리보드를 아이폰 형태로 수정한다.

1. Xcode 를 실행한 후 'Web'이라는 이름으로 프로젝트를 만든다.



### 07-3 스토리보드로 웹뷰 앱 화면 꾸미기

웹의 내용을 보여 주는 웹 뷰(Web View), 주소를 직접 입력할 수 있는 텍스트 필드(Text Field), 미리 정해둔 웹 주소로 이동하여 웹 뷰로 보여 줄 버튼(Button), 웹을 제어할 바 버튼 아이템을 품고 있는 툴바(Toolbar), 웹을 제어할 버튼인 바 버튼 아이템(Bar Button Item), 바 버튼 아이템을 균등하게 배치하기 위한 플렉서블 스페이스 바 버튼 아이템(Flexible Space Bar Button Item), 그리고 웹을 로딩하는 중임을 알리는 액티비티 인디케이터 뷰(Activity Indicator View) 객체를 사용한다.



#### 1. 홈페이지 URL 을 입력할 텍스트 필드 추가하기

홈페이지 URL 을 입력할 텍스트 필드를 만들어 보자. 오른쪽 아랫부분의 오브젝트 라이브러리의 검색란에 'te'를 입력하여 [텍스트 필드(Text Field)]를 찾아 스토리보드로 끌어와 왼쪽 윗부분에 배치한다.

2. 홈페이지 이동을 위한 버튼을 만들어 보자. 오른쪽 아랫부분의 오브젝트 라이브러리에서 [버튼(Button)]을 찾아 스토리보드로 끌어와 텍스트 필드의 오른쪽에 배치한다. 버튼을 마우스로 더블 클릭한 후 내용을 'Go'로 변경한다.



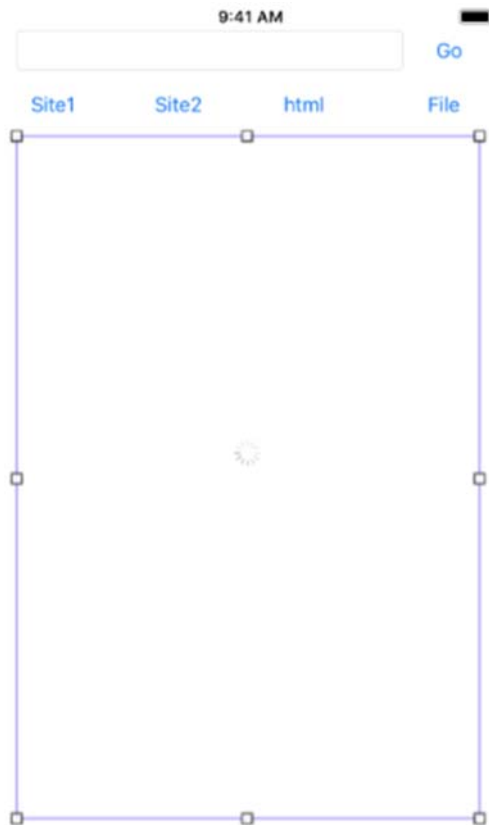
3. 같은 방법으로 다음과 같은 위치에 버튼을 네 개 더 추가한다. 그리고 나서 각 버튼을 마우스로 더블 클릭한 후 버튼의 이름을 'Site1', 'Site2', 'html', 'File'로 변경한다.

Site1 Site2 html File

#### 4. 웹 뷰 추가하기

홈페이지를 보여 줄 웹 뷰를 만들어 보겠다. 오른쪽 아랫부분의 오브젝트 라이브러리에서 [웹 뷰(Web View)]를 찾아 스토리보드로 끌어와 적당한 위치에 배치한다.

이 웹 뷰에서 홈페이지를 보여 줄 것이다.

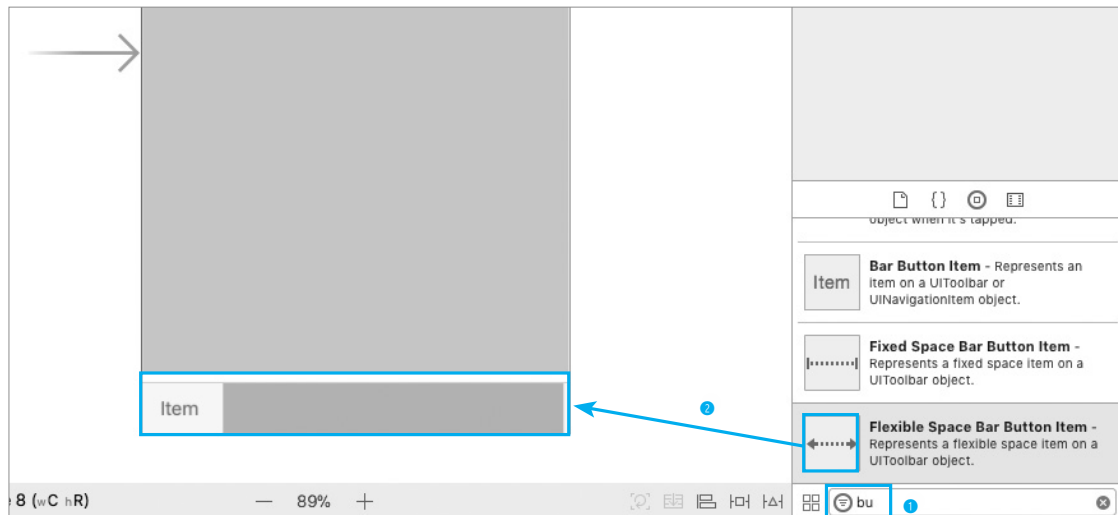


#### 5. 툴바 추가하기

홈페이지를 제어할 수 있는 툴바를 만들어 본다. 오른쪽 아랫부분의 오브젝트 라이브러리의 검색란에 'too'를 입력한 후 [툴바(Toolbar)]를 찾아 스토리보드로 끌어와 웹 뷰의 아래쪽에 배치한다.

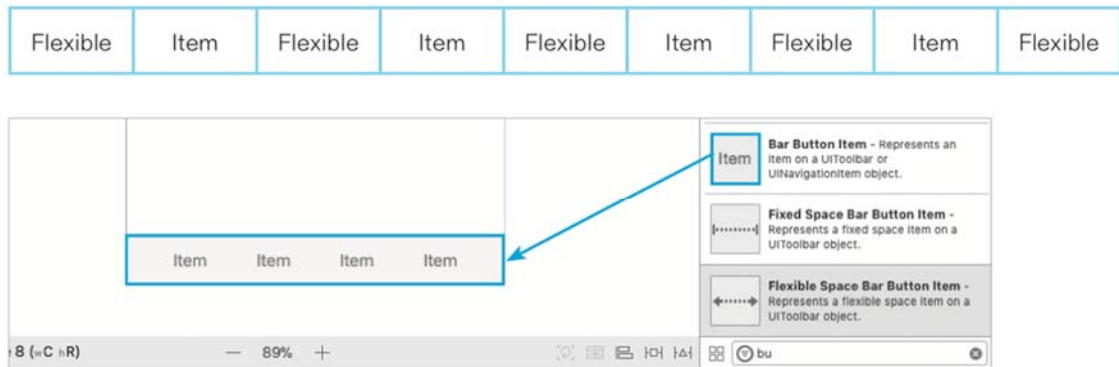
툴바는 앱 화면의 아래쪽에서 주로 볼 수 있고 투바에는 '툴바 아이템'이라고 하는, 한 개 이상의 버튼이 있다. 이 버튼은 보통 화면의 현재 내용과 관련된 도구를 제공한다. 예를 들면 지금 우리가 만들고 있는 앱의 투바는 웹 뷰를 통해 웹을 읽어 올 때 필요한 정지(Stop), 새로고침(ReFresh), 되감기(Rewind), 앞으로 감기(Fast Forward)등과 같은 도구(기능)를 제공한다. 이 도구는 투바나 내비게이션 바에 배치하기 위해 '바 버튼 아이템'이라고 하는 특화된 버튼으로 이루어져 있다.

6. 이 투바에 바 버튼 아이템을 추가해야 한다. 하지만 그 전에 버튼들이 투바에 균등하게 배치될 수 있도록 플렉서블 스페이스 바 버튼 아이템을 추가한다. 오른쪽 아랫부분의 오브젝트 라이브러리의 검색란에 'bu'를 입력하여 [플렉서블 스페이스 바 버튼 아이템(Flexible Space Bar Button Item)]을 찾은 후 스토리보드로 끌어와 투바 내부의 [Item]버튼 왼쪽에 배치 한다.




7. 플렉서블 스페이스 바 버튼 아이템(Flexible Space Bar Button Item)과 바 버튼 아이템(Bar Button Item)을 번갈아가면서 배치해 투바를 채워 보자.

바 버튼 아이템은 네개, 플렉서블 스페이스 바 버튼 아이템은 다섯 개가 되도록 추가한다. 먼저 바 버튼 아이템 네개를 연이어 배치하고, 사이 사이에 플렉스블 스페이스 바 버튼 아이템을 끼워 주면 쉽게 추가할 수 있다.



8. 툴바 내부의 바 버튼 아이템(Item)의 아이콘 모양을 수정해 보자. 첫번째에 위치한 [Item]바 버튼을 더블 클릭하고 오른쪽 윗부분의 [Show the Attributes inspector] 버튼을 클릭하면 System Item 이 [Custom]으로 되어 있는 것을 확인 할 수 있다.

이를 [정지(Stop)]로 변경하면 [Item]의 아이콘이  모양으로 변경된다.

Style 은 Plain, Bordered, Done 을 선택할 수 있으며 Plain 이 가장 큰 크기이고, Done 이 가장 작은 크기이다.

System Item 은 아이템의 디자인을 북마크, 카메라, 휴지통 등과 같은 모양으로 바꿀 수 있다.

Tint 는 해당 아이템의 색상을 바꿀 수 있다.

9. 같은 방법으로 두번째 바 버튼 아이템을 [새로고침(Reload)]으로 변경한다. 세번째 바 버튼 아이템은 [되감기(Rewind)], 네 번째 바 버튼 아이템은 [앞으로 감기(Forward)]로 변경한다.



10. 로딩을 표시하기 위한 액티비티 인디케이터 뷰 추가하기.

아이폰 사용자라면 로딩을 기다릴 때 화면 가운데에서 돌아가는 원 모양의 점선을 본 적이 있을 것이다.

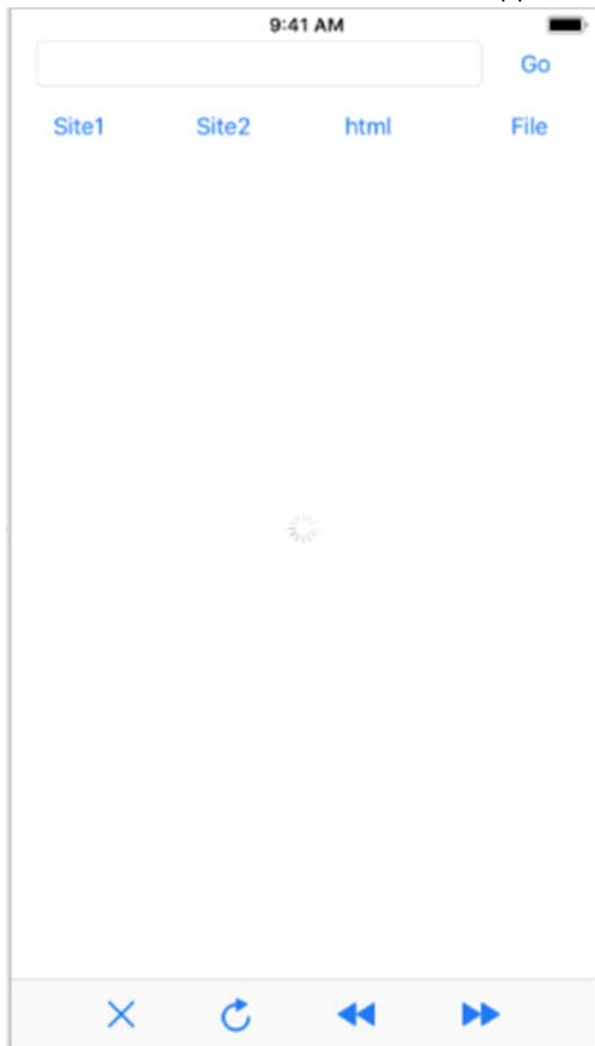
이것이 바로 '액티비티 인디케이터 뷰'이다. 이번에는 로딩을 표시하기 위한 액티비티 인디케이터 뷰를 만들어 보자. 오른쪽 아랫부분의 오브젝트 라이브러리의 검색란에 'in' 을 입력하여 [액티비티 인디케이터 뷰(Activity Indicator View)]를 찾는다.

그리고 이를 스토리보드로 끌어와 중앙에 배치한다,

배치한 액티비티 인디케이터 뷰를 다시 클릭한 후 오른쪽 윗부분의 '액티비티 인디케이터 뷰(Activity Indicator View)'에서 동작할 때만 인디케이터가 보이고 동작을 멈추면 보



이지 않게 하기 위해 [Hides When Stopped]항목에 체크한다.

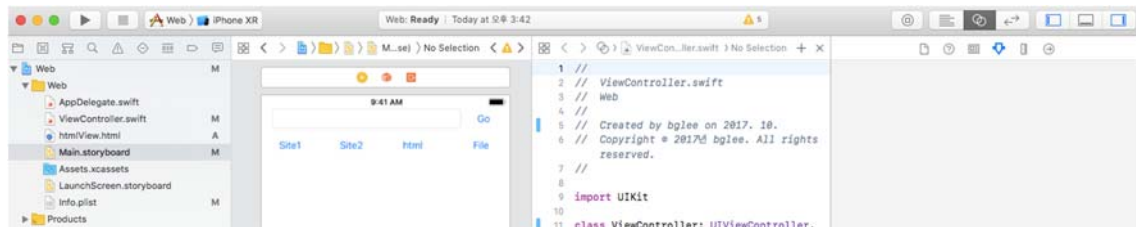


## 07-4 아웃렛 변수와 액션 함수 추가하기

스토리보드에서 추가한 객체들을 프로그램으로 제어하기 위해 소스코드에 변수와 함수 형태로 추가한다. 객체를 선택했을 때 어떤 동작을 수행해야 한다면 액션 함수로 추가하고, 객체의 값을 이용하거나 속성 등을 제어해야 한다면 아웃렛 변수 형태로 추가한다.

1. 보조 편집기와 액션 함수를 추가하려면 우선 오른쪽 윗부분의 [Show the Assistant editor] 버튼을 클릭하여 보조 편집기 영역을 열어야 한다.

가운데 화면의 스토리보드 부분이 둘로 나누어지면서 왼쪽에는 스토리보드, 오른쪽에는 보조 편집기 영역이 나타난다.



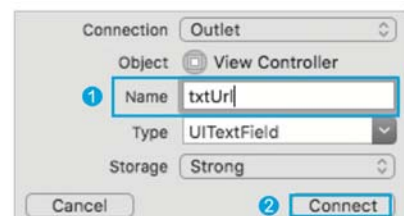
2. 텍스트 필드에 대한 아웃렛 변수 추가하기

첫번째로 텍스트 필드에 대한 아웃렛 변수를 추가한다.

텍스트 필드를 마우스 오른쪽 버튼으로 클릭한 후 보조 편집기 영역으로 드래그하면 아래 그림과 같이 연결선이 나타난다, 드래그한 연결선을 뷰 컨트롤러의 클래스 선언문 바로 아래에 끌어다 놓는다.

텍스트 필드를 마우스 오른쪽 버튼으로 클릭한 후 드래그하여 보조 편집기 영역에 드롭하면 다음과 같이 연결 설정 창이 나타난다. 이 설정 창에서 아웃렛 변수의 이름 (Name)을 'txtUrl'로 입력한 후 [Connect]버튼을 클릭하여 텍스트 필드와 아웃렛 변수를 연결한다.

위치	뷰 컨트롤러 클래스 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	txtUrl
유형(Type)	UITextField



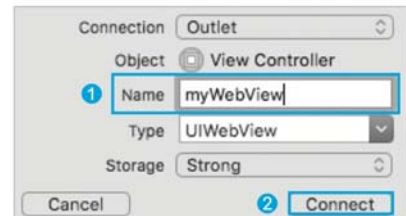
```
@IBOutlet var txtUrl: UITextField!
```

3. 다음으로 웹 뷰에 대한 아웃렛 변수를 추가한다. 웹 뷰를 마우스 오른쪽 버튼으로 클릭 한 후 오른쪽 보조 편집기 영역으로 드래그하면 연결선이 나타난다.

텍스트 필드 변수 아래에 끌어다 놓는다.

보조 편집기 영역에 드롭하면 다음과 같이 연결 창이 나타난다. 이 설정 창에서 아웃렛 변수의 이름(Name)을 'myWebView'로 입력한 후 [Connect] 버튼을 클릭하여 웹 뷰와 아웃렛 변수를 연결한다.

위치	txtUrl 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	myWebView
유형(Type)	UIWebView

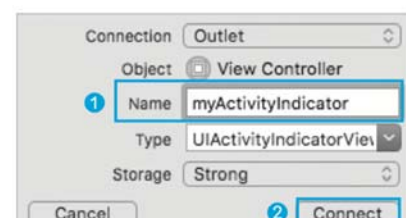


```
@IBOutlet var myWebView: UIWebView!
```

4. 액티비티 인디케이터 뷰에 대한 아웃렛 변수를 추가한다. 액티비티 인디케이터 뷰를 마우스 오른쪽 버튼으로 클릭한 후 오른쪽 보조 편집기 영역으로 드래그한다. 웹 뷰 변수 바로 아래에 끌어 놓는다.

액티비티 인디케이터 뷰를 마우스 오른쪽 버튼으로 클릭한 후 드래그하여 보조 편집기 영역에 드롭하면 연결 설정 창이 나타난다. 이 설정 창에서 아웃렛 변수의 이름(Name)을 'myActivityIndicator'로 입력한 후 [Connect] 버튼을 클릭하여 웹 뷰와 아웃렛 변수를 연결한다.

위치	myWebView 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	myActivityIndicator
유형(Type)	UIActivityIndicatorView



```
@IBOutlet var myActivityIndicator: UIActivityIndicatorView!
```

5. 필요한 모든 아웃렛 변수가 다음과 같이 추가 되었다.

```
11 class ViewController: UIViewController, UIWebViewDelegate {
12
13     @IBOutlet var txtUrl: UITextField!
14     @IBOutlet var myWebView: UIWebView!
15     @IBOutlet var myActivityIndicator: UIActivityIndicatorView!
16 }
```

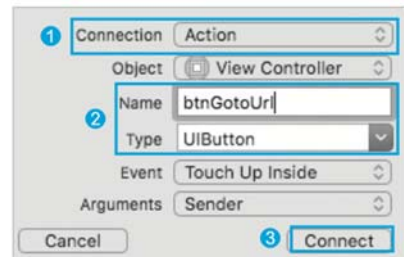
\*\*\* 웹 뷰의 액션 함수 추가하기

#### 1. [Go]버튼에 대한 액션 추가하기

[Go] 버튼에 대한 액션을 추가한다. 마우스 오른쪽 버튼으로 [Go]를 클릭한 후 드래그해서 오른쪽 편집기 영역의 뷰 컨트롤러 클래스 아래에 갖다 놓는다.

연결설정 창이 나타나면 아래와 같이 설정 한 후 [Connect]버튼을 클릭하여 추가한다.

위치	뷰 컨트롤러 클래스의 마지막 '}' 바로 위 (didReceiveMemoryWarning 함수 아래)
연결(Connection)	Action
이름(Name)	btnGotoUrl
유형(Type)	UIButton



```
@IBAction func btnGotoUrl(_ sender: UIButton) {
}
```

2. [Site1]버튼에 대한 액션을 추가한다. 마우스 오른쪽 버튼으로 [Site1]을 클릭한 후 드래그해서 오른쪽 보조 편집기 영역의 뷰 컨트롤러 클래스 아래쪽의 btnGotoUrl 함수 아래에 갖다 놓는다.

연결 설정 창이 나타나면 아래와 같이 설정한 후 [Connect]버튼을 클릭하여 액션을 추가 한다.

위치	뷰 컨트롤러 클래스의 마지막 '}' 바로 위 (btnGotoUrl 함수 아래)
연결(Connection)	Action
이름(Name)	btnGoSite1
유형(Type)	UIButton



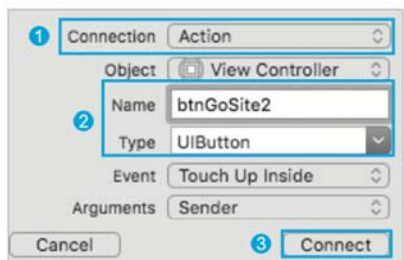
```
@IBAction func btnGoSite1(_ sender: UIButton) {
}
```

3. [Site2], [html]과 [File]버튼에 대한 액션을 추가한다. [Site1]버튼의 액션 함수 아래쪽에 소스를 연결하면 된다.

[Site2]버튼의 액션 함수를 추가한다. 마우스 오른쪽 버튼으로 [Site2]를 BtnGoSite1함수 아래 갖다 놓는다.

연결 설정 창이 나타나면 아래와 같이 설정한 후 [Connect]버튼을 클릭하여 액션을 추가한다.

위치	뷰 컨트롤러 클래스의 마지막 '}' 바로 위 (btnGoSite1 함수 아래)
연결(Connection)	Action
이름(Name)	btnGoSite2
유형(Type)	UIButton

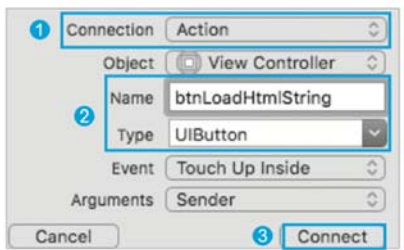


```
@IBAction func btnGoSite2(_ sender: UIButton) {
}
```

4. [html]버튼에 대한 액션 함수를 추가하자. 마우스 오른쪽 버튼으로 [html]을 클릭한 후 드래그해서 오른쪽 보조 편집기 영역의 뷰 컨트롤러 클래스 아래 쪽의 btnGotoSite2 함수 아래에 갖다 놓는다.

연결 설정 창이 나타나면 아래와 같이 설정 후 [Connect]버튼을 클릭하여 액션을 추가한다.

위치	뷰 컨트롤러 클래스의 마지막 '}' 바로 위 (btnGoSite2 함수 아래)
연결(Connection)	Action
이름(Name)	btnLoadHtmlString
유형(Type)	UIButton

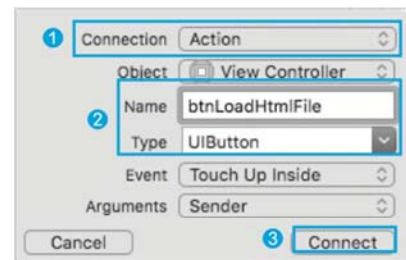


```
@IBAction func btnLoadHtmlString(_ sender: UIButton) {
}
```

5. 마지막으로 [File]버튼에 대한 액션을 취하자. 마우스 오른쪽 버튼으로 [File]을 클릭한 후 드래그해서 오른쪽 보조 편집기 영역의 뷰 컨트롤러 클래스 아래 쪽의 btnLoadHtmlString 함수 아래에 갖다 놓는다.

연결 설정 창이 나타나면 아래와 같이 설정한 후 [Connect]버튼을 클릭하여 액션을 추가한다.

위치	뷰 컨트롤러 클래스의 마지막 '}' 바로 위 (btnLoadHtmlString 함수 아래)
연결(Connection)	Action
이름(Name)	btnLoadHtmlFile
유형(Type)	UIButton

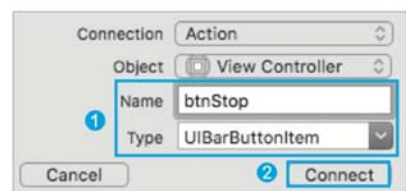


```
@IBAction func btnLoadHtmlFile(_ sender: UIButton) {
}
```

6. 툴바의 [Stop]아이템에 대한 액션을 추가한다. 마우스 왼쪽 버튼으로 [Stop]아이템을 더블 클릭하면 선택이 되고 선택된 상태에서 오른쪽 버튼으로 [Stop]아이템을 클릭 후 드래그해서 소스의 가장 아래쪽 btnLoadHtmlFile 함수 아래에 갖다 놓는다.

연결 설정 창에서 연결(Connection)을 [Action]으로 변경한다. 그리고 이름(Name)을 'btnStop'으로 입력하고, 유형(Type)은 UIBarButtonItem 의 액션을 추가하는 것이므로 [UIBarButtonItem]을 선택한다. 변경완료 후 [Connect]버튼을 클릭한다.

위치	뷰 컨트롤러 클래스의 마지막 '}' 바로 위 (btnLoadHtmlFile 함수 아래)
연결(Connection)	Action
이름(Name)	btnStop
유형(Type)	UIBarButtonItem

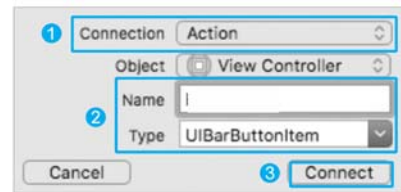


```
@IBAction func btnStop(_ sender: UIBarButtonItem) {
}
```

7. 'Refresh', 'Rewind', 'Fast Forward'버튼에 대한 액션을 추가한다. [Stop]버튼의 액션 소스 바로 아래에 이어서 추가한다.

툴바 위의 버튼 'Refresh', 'Rewind', 'Fast Forward'의 액션 함수를 추가할 때 연결(Connection)은 [Action]으로, 유형(Type)은 [UIBarButtonItem]으로 설정하고 이름(Name)만 다르게 설정한다,  
'Refresh'는 'btnRefresh'로, 'Rewind'는 'btnGoback'으로, 'Fast Forward'는 'btnGoForward'로 입력한다.

위치	뷰 컨트롤러 클래스의 마지막 '}' 바로 위 (btnStop 함수 아래)
연결(Connection)	Action
이름(Name)	[Refresh] 버튼 : btnReload [Rewind] 버튼 : btnGoBack [Fast Forward] 버튼 : btnGoForward
유형(Type)	UIBarButtonItem



```
@IBAction func btnReload(_ sender: UIBarButtonItem) {
}
@IBAction func btnGoBack(_ sender: UIBarButtonItem) {
}
@IBAction func btnGoForward(_ sender: UIBarButtonItem) {
}
```

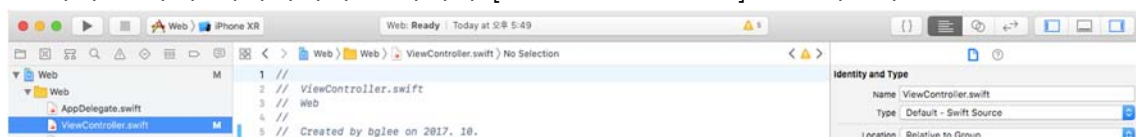
## 07-5 앱시작할때지정된페이지보여주기

웹 뷰에 웹 페이지를 보여 주기 위한 함수인 loadWebPage 함수를 구현해 보겠다.  
앱이 시작되면 loadWebPage 함수를 호출하여 지정된 웹 주소의 페이지를 보여 준다.

### 1. 스탠더드 에디터로 화면 모드 수정

코딩을 하기 위해 화면 모드를 수정하자. 오른쪽 윗부분의 [Show the Standard editor] 버튼을 클릭한다.

### 2. 이제 왼쪽의 내비게이터 영역에서 [ViewController.swift]를 선택한다.

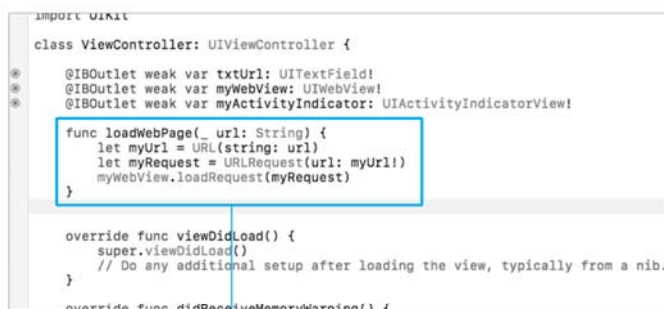


### 3. 지정 웹 페이지 보여주기

우선 웹을 처음 시작할 때 기본적으로 특정 웹 페이지를 보여 주기 위해 웹 페이지를 로드하는 함수를 만들어야 한다.

웹 페이지 주소를 url 의 인수를 통해 전달해서 웹페이지를 보여 주는 loadWebPage 함수를 myActivityIndicator 아웃렛 변수와 viewDidLoad 함수 사이에 추가한다.

### 4. 스트링형 url 을 이용하여 웹 페이지를 나타내는 순서는 다음 3 단계를 거친다. loadWebPage 함수 안에 추가한다.



```
func loadWebPage(_ url: String) {
    let myUrl = URL(string: url) —①
    let myRequest = URLRequest(url: myUrl!) —②
    myWebView.loadRequest(myRequest) —③
}
```

- 1) 상수 myUrl 은 url 값을 받아 URL 형으로 선언한다.
- 2) 상수 myRequest 는 상수 myUrl 을 받아 URL Request 형으로 선언한다.
- 3) UIWebView 형의 myWebView 클래스의 load Request 메서드를 호출한다.



5. 앱을 시작할 때 지정한 웹 페이지가 나타나도록 viewDidLoad 함수 안에 loadWebPage 함수를 추가해 보자.

loadWebPage 함수 아래에 있는 viewDidLoad 함수로 이동한다.

6. 앱이 처음 나타나면 접속할 웹 페이지 주소를 추가하여 viewDidLoad 함수 안에 loadWebPage 함수를 추가한다.

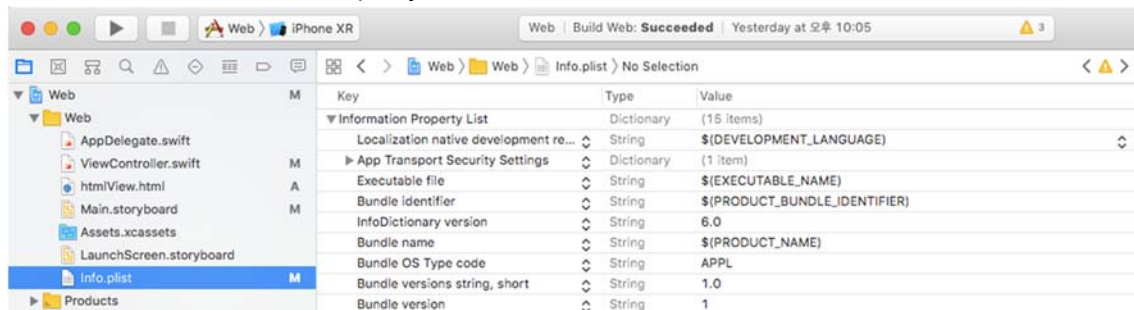
주소는 임의로 적어준다. (예, http://club.cyworld.com/smrit)

```
23 override func viewDidLoad() {  
24     super.viewDidLoad()  
25     // Do any additional setup after loading the view, typically from a nib.  
26     myWebView.delegate = self  
27     loadWebPage("http://club.cyworld.com/smrit")  
28 }
```

7. 시작 홈페이지 결과 보기

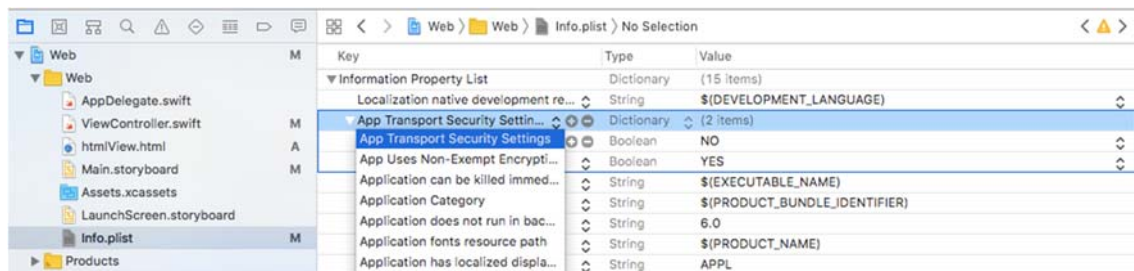
실행하기 전에 iOS 시뮬레이터의 기기를 원하는 기기로 변경한다. 그리고 [실행] 버튼을 클릭한다. [실행] 버튼을 클릭해서 시뮬레이터가 구동이 되어도 원하는 웹페이지가 나오지 않는다. 'Info.plist'를 수정하여 해결한다.

8. 'Info.plist'를 수정하기 위해 왼쪽의 내비게이터 영역에서 [Info.plist]를 선택한다. 그런 다음 [Information Property List]의 오른쪽에 있는 [+]눌러 항목을 추가한다.



드롭 다운 목록에서 키보드 방향 키로 이동해 [App Transport Security Settings]를 선택

return 을 한 번 더 눌러 세팅을 완료



App Transport Security Settings] 왼쪽의 화살표가 아래로 향하도록 한 다음 다시 한번 [+]를 누른다.

[Allow Arbitrary Loads]를 마우스로 선택한 return 을 눌러 완료 [Value]값을 [YES] 변경

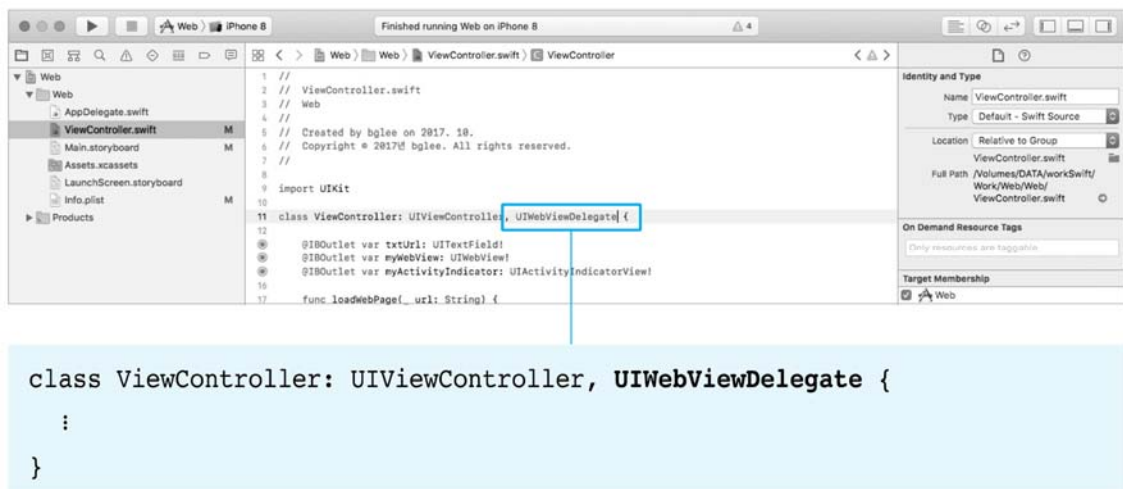


## 07-6 액티비티 인디케이터 구현하기

웹 페이지를 로딩할 때 네트워크의 속도가 느리거나 서버에 부하가 많이 걸려 로딩이 오라 걸렸던 경험은 누구나 있을 것이다. 이때 필요한 것이 바로 액티비티 인디케이터이다. 액티비티 인디케이터 원형으로 돌아가는 애니메이션 효과로, 앱이 동작 중임을 사용자에게 보여준다.

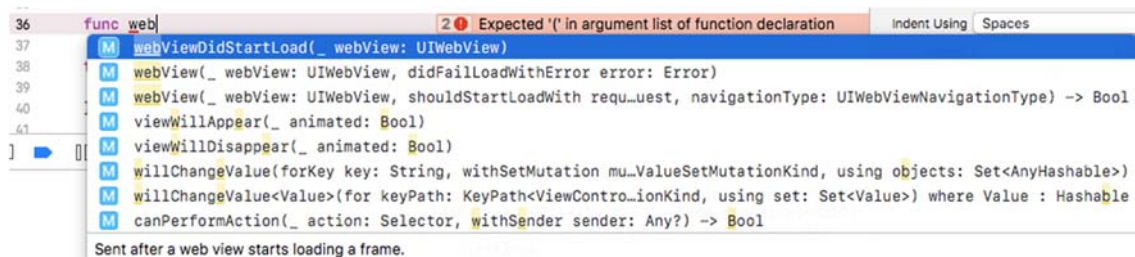
1. 다시 외쪽 네비게이터 영역의 [ViewController.swift]를 선택해서 코딩을 계속해 보자  
액티비티 인디케이터가 동작하는 시점이 웹 뷰가 로딩할 때이고, 종료하는 시점이 웹 뷰의 로딩이 끝났을 때이므로 `webViewDidStartLoad` 와 `webViewDidFinishLoad` 함수를 사용해야 한다.

이 함수를 사용하기 위해 `UIWebViewDelegate` 를 상속받는다.



2. 액티비티 인디케이터를 구현하기 위한 두 함수 중에서 먼저 `webViewDidStartLoad` 함수를 `didReceiveMemoryWarning` 함수 아래에 추가한다.

'Web'만 입력하면 자동완성이 알아서 찾아 주므로 전부 입력할 필요는 없다.



3. 액티비티 인디케이터의 동작을 시작하는 함수를 `webViewDidStartLoad` 함수 안에 추가한다.

```

23 override func viewDidLoad() {
24     super.viewDidLoad()
25     // Do any additional setup after loading the view, typically from a nib.
26     loadWebPage("http://2sam.net")
27 }
28
29 override func didReceiveMemoryWarning() {
30     super.didReceiveMemoryWarning()
31     // Dispose of any resources that can be recreated.
32 }
33
34 func webViewDidStartLoad(_ webView: UIWebView) {
35     myActivityIndicator.startAnimating()
36 }
37
38 @IBAction func btnGoToUrl1(_ sender: UIButton) {
39 }
40
41 @IBAction func btnGoToSite1(_ sender: UIButton) {
42 }
43
44 @IBAction func btnGoToSite2(_ sender: UIButton) {
45 }
46

```

```

func webViewDidStartLoad(_ webView: UIWebView) {
    myActivityIndicator.startAnimating()
}

```

4. 다음으로 두 번째 함수인 webViewDidFinishLoad 함수를 webViewDidStartLoad 함수 아래 추가한다,

5. 액티비티 인디케이터의 동작을 종료하는 함수를 그 안에 추가한다.

```

24 super.viewDidLoad()
25 // Do any additional setup after loading the view, typically from a nib.
26 loadWebPage("http://2sam.net")
27 }
28
29 override func didReceiveMemoryWarning() {
30     super.didReceiveMemoryWarning()
31     // Dispose of any resources that can be recreated.
32 }
33
34 func webViewDidStartLoad(_ webView: UIWebView) {
35     myActivityIndicator.startAnimating()
36 }
37
38 func webViewDidFinishLoad(_ webView: UIWebView) {
39     myActivityIndicator.stopAnimating()
40 }
41
42 @IBAction func btnGoToUrl1(_ sender: UIButton) {
43 }
44
45 @IBAction func btnGoToSite1(_ sender: UIButton) {
46 }
47

```

```

func webViewDidFinishLoad(_ webView: UIWebView) {
    myActivityIndicator.stopAnimating()
}

```

6. 마지막으로 viewDidLoad 함수에 myWebView 의 델리게이트(delegate)를 self 로 추가한다.

```

16 @IBOutlet var txtUrl: UITextField!
17 @IBOutlet var myWebView: UIWebView!
18 @IBOutlet var myActivityIndicator: UIActivityIndicatorView!
19
20 func loadWebPage(_ url: String) {
21     let myUrl = URL(string: url)
22     let myRequest = URLRequest(url: myUrl!)
23     myWebView.loadRequest(myRequest)
24 }
25
26 override func viewDidLoad() {
27     super.viewDidLoad()
28     // Do any additional setup after loading the view, typically from a nib.
29     myWebView.delegate = self
30     loadWebPage("http://2sam.net")
31 }
32
33 override func didReceiveMemoryWarning() {
34     super.didReceiveMemoryWarning()
35     // Dispose of any resources that can be recreated.
36 }
37
38 func webViewDidStartLoad(_ webView: UIWebView) {
39     myActivityIndicator.startAnimating()
40 }
41

```

```

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    myWebView.delegate = self
    loadWebPage("http://2sam.net")
}

```

## 07-7 [Site1], [Site2] 버튼 구현하기

1. [Site1]과 [Site2]버튼을 클릭해서 이동할 때 웹 사이트의 주소를 btnGoSite1, btnGoSite2 함수에 추가한다.

```
60 @IBAction func btnGoSite1(_ sender: UIButton) {  
61     loadWebPage("http://fallinmac.tistory.com")  
62 }  
64 @IBAction func btnGoSite2(_ sender: UIButton) {  
65     loadWebPage("http://club.cyworld.com/smrit")  
66 }
```

## 2. 결과



## 07-8 [정지], [재로딩], [이전 페이지], [다음 페이지] 버튼 구현하기

1. 웹브라우저에 있는 버튼을 이용해서 정지, 재로딩, 이전 페이지로 이동, 다음 페이지로 이동을 제어할 수 있는 코드를 추가하자.

```
@IBAction func btnStop(_ sender: UIBarButtonItem) {  
    myWebView.stopLoading() ❶ 웹 페이지의 로딩을 중지시키는 함수를 호출합니다.  
}  
  
@IBAction func btnReload(_ sender: UIBarButtonItem) {  
    myWebView.reload() ❷ 웹 페이지를 재로딩시키는 함수를 호출합니다.  
}  
  
@IBAction func btnGoBack(_ sender: UIBarButtonItem) {  
    myWebView.goBack() ❸ 이전 웹 페이지로 이동시키는 함수를 호출합니다.  
}  
  
@IBAction func btnGoForward(_ sender: UIBarButtonItem) {  
    myWebView.goForward() ❹ 다음 웹 페이지로 이동시키는 함수를 호출합니다.  
}
```

## 07-9 [html] 버튼 구현하기

HTML 코드를 변수에 저장하고 [html]버튼을 클릭하면 변수의 내용이 HTML 형식에 맞추어 웹 뷰로 나타나게 구현해 보자.

HTML로 변환할 수 있는 모든 것은 웹 뷰를 통해 표현한다.

1. 다음 소스를 btnLoadHtmlString 함수 안에 입력하자.

```
1. 다음 소스를 btnLoadHtmlString 함수 안에 입력하자.
@IBAction func btnLoadHtmlString(_ sender: UIButton) {
68     let htmlString = "<h1> HTML String </h1><p> String 변수를 이용한 웹페이지 </p>
        <p><a href=\"http://club.cyworld.com/smrirt\">smrirt</a>으로 이동</p>"
69     myWebView.loadHTMLString(htmlString, baseURL: nil)
70 }
```

68 행은 HTML 문을 변수에 저장한다.

69 행은 loadHTMLString 함수를 이용하여 변수에 저장된 HTML 문을 뷰에 나타낸다.

htmlString 문자열 상수에 HTML 코드를 대입할 때는 큰 따옴표(" ")안에 넣어야 한다. 그런데 이때 다음 코드처럼 줄을 바꿔 입력하면 에러가 발생한다. 따라서 아래의 HTML 코드를 줄바꿈 없이 사용해야 한다.

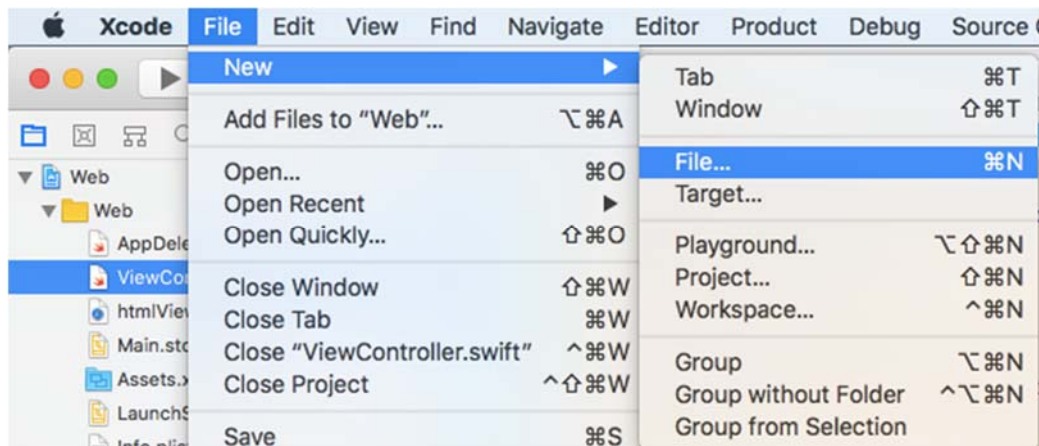




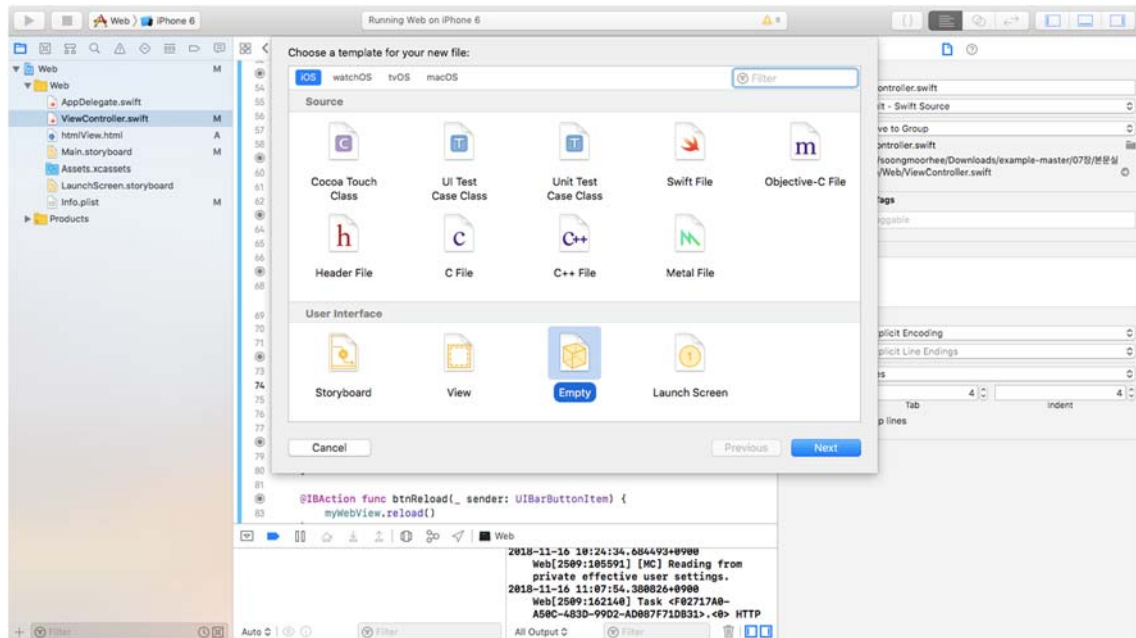
## 07-10 [File] 버튼 구현하기

간단하게 파일을 프로젝트에 추가하고 HTML 문법에 맞게 작성한 후 [File]버튼을 클릭하면 이 파일이 웹 뷰에 나타나도록 구현해 보자. 만약 모바일 형식에 맞게 작성된 HTML 파일이 있다면 웹 뷰를 이용하여 하이브리드 앱이 아닌 iOS 전용 앱으로 만들 수 있다.

1. [File]버튼을 클릭하면 HTML 파일을 읽어들이어 웹 뷰로 나타내는 코드를 추가해 보자. 메뉴에서 [File -> New -> File...]을 클릭한다.

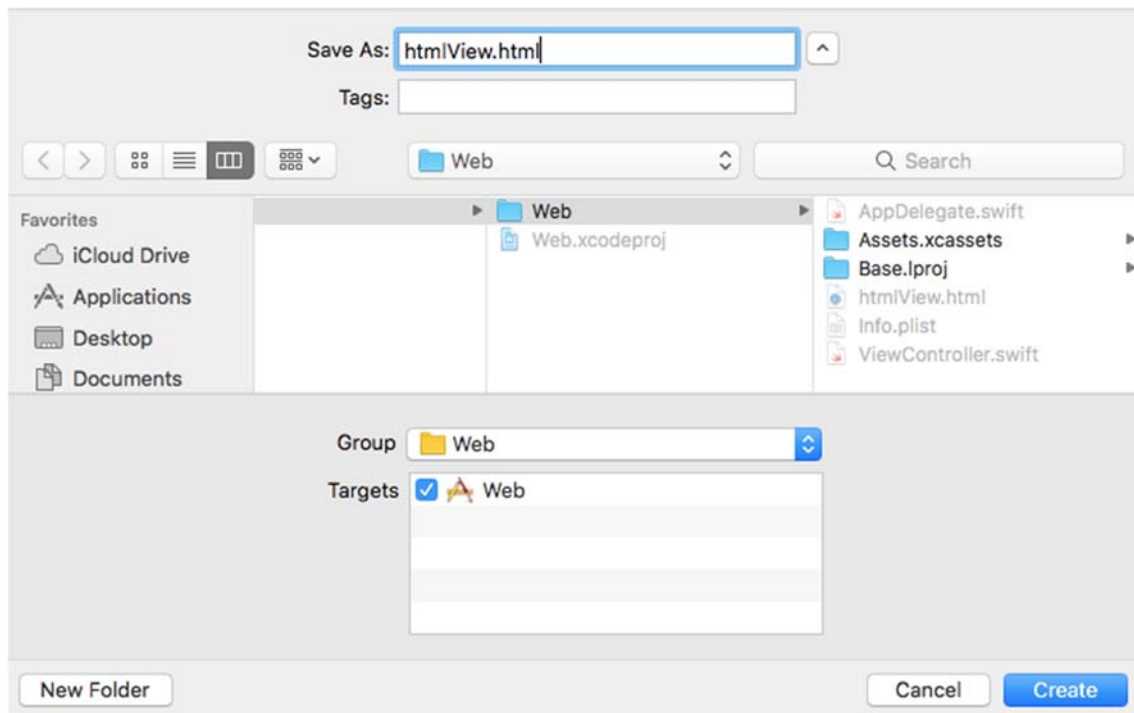


2. iOS 에서 마우스를 스크롤해 [Other]의 [Empty]를 선택한 후 [Next]버튼을 클릭한다.



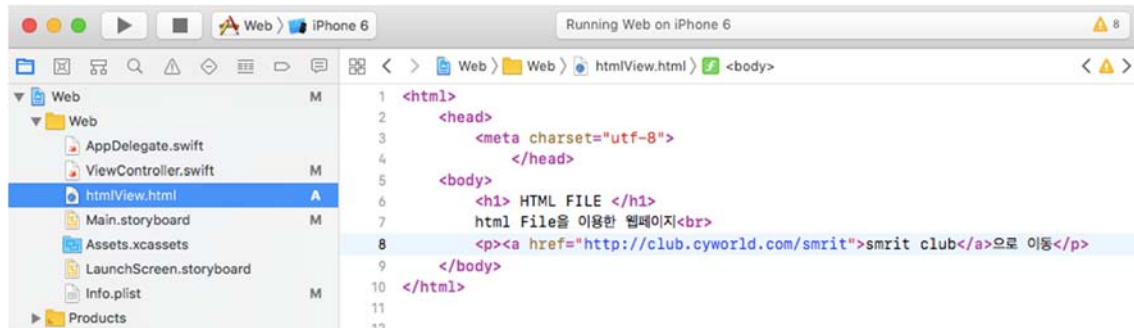
3. Save As 'htmlView.html'을 입력해 파일명을 정하고 [Create]버튼을 클릭하여 HTML 파일을 생성한다.



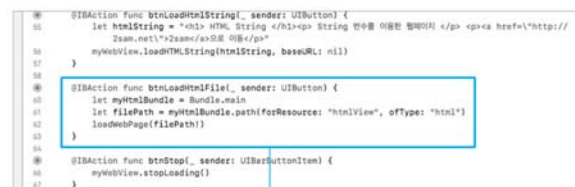


4. HTML 파일이 추가된 것을 확인할 수 있다. 추가된 HTML 파일을 [Web]폴더로 끌어다 놓는다.

HTML 문을 편집창에 추가한다.

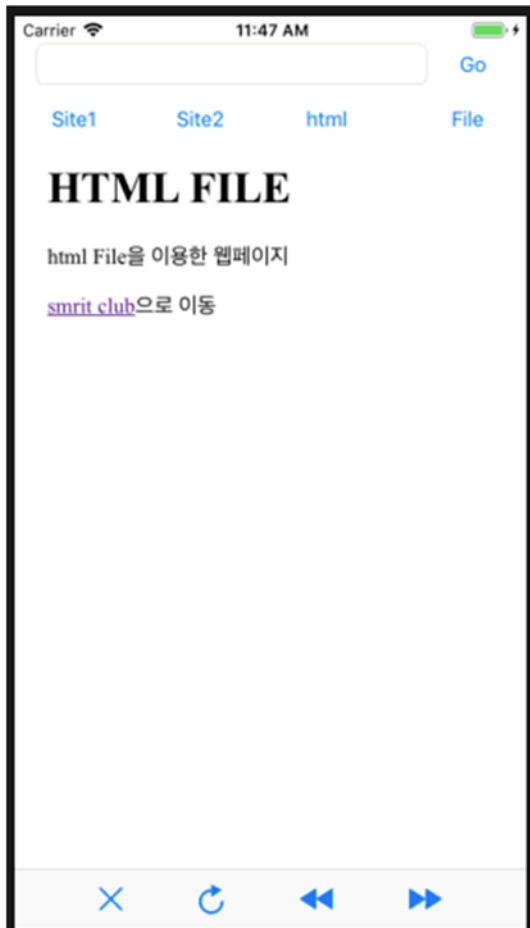


5. [ViewController.swift]를 선택하여 btnLoadHtmlFile 함수에 입력한다.



```
@IBAction func btnLoadHtmlFile(_ sender: UIButton) {
    let myHtmlBundle = Bundle.main
    let filePath = myHtmlBundle.path(forResource: "htmlView", ofType: "html")
    loadWebPage(filePath!)
}
```

- 1) Bundle 에서 main 으로 변수를 생성한다,
- 2) path 함수를 호출한다.
- 3) HTML 파일을 로딩한다.



## 07-11 'http://' 문자열 자동 삽입 기능 구현하기

웹 브라우저에서 홈페이지 주소를 입력할 때 보통은 '프로토콜(Http://)'을 직접 입력하지 않고 사용한다. 하지만 이는 웹 브라우저에서 기본적으로 설정을 해 놔기 때문이다. 실제로는 내부적으로 프로토콜이 있어야 한다.

1. 홈페이지 주소를 문자열(String)로 받고, 이를 처리한 후 다시 문자열로 가져오는 checkUrl 함수 코드를 btnGotoUrl 함수 위쪽에 추가하고 소스를 작성한다.

```
41 }
42
43 func checkUrl(_ url: String) -> String {
44     var strUrl = url
45     let flag = strUrl.hasPrefix("http://")
46     if !flag {
47         strUrl = "http://" + strUrl
48     }
49     return strUrl
50 }
51
52 @IBAction func btnGotoUrl(_ sender: UIButton) {
53 }
54
55 @IBAction func btnGoSite1(_ sender: UIButton) {
56     loadWebPage("http://fallinmac.tistory.com")
57 }
58
59 @IBAction func btnGoSite2(_ sender: UIButton) {
60     loadWebPage("http://blog.2sam.net")
61 }
```

```
func checkUrl(_ url: String) -> String {
    var strUrl = url
    let flag = strUrl.hasPrefix("http://")
    if !flag {
        strUrl = "http://" + strUrl
    }
    return strUrl
}
```

- 1) 입력 받은 url 스트링을 임시변수 strUrl 에 넣는다.
- 2) "http://"를 가지고 있는지 확인한 값을 flag 에 넣는다.
- 3) http:// 를 가지고 있지 않다면, 즉 "!flag"일 때 변수 strUrl 에 <http://>를 추가하고 이를 리턴한다,

2. [Go]버튼을 클릭할 때 텍스트 필드에 적힌 주소를 웹 뷰가 로딩되도록 btnGotoUrl 함수를 수정한다. 흔히 웹 브라우저의 주소 창에서 간단히 "club.cyworld.com/smrit"의 주소를 입력하지만 웹은 프로토콜로 http 를 사용하기 때문에 내부적으로는 자동으로 "http://club.cyworld.com/smrit"을 호출한다.

앞에서 만든 checkUrl 함수에 텍스트 필드에 적힌 주소를 호출하여 변수 myUrl 로 받고, 이를 loadWebPage 함수를 이용하여 웹 뷰에 로딩한다.

btnGotoUrl 함수에 소스를 추가한다.

```
42     strUrl = "http://" + strUrl
43 }
44 return strUrl
45 }
46
47 @IBAction func btnGotoUrl(_ sender: UIButton) {
48     let myUrl = checkUrl(txtUrl.text!)
49     txtUrl.text = ""
50     loadWebPage(myUrl)
51 }
52
53 @IBAction func btnGoSite1(_ sender: UIButton) {
54     loadWebPage("http://fallinmac.tistory.com")
55 }
56
57 @IBAction func btnGoSite2(_ sender: UIButton) {
58     loadWebPage("http://blog.2sam.net")
59 }
60 }
```

```
@IBAction func btnGotoUrl(_ sender: UIButton) {
    let myUrl = checkUrl(txtUrl.text!)
    txtUrl.text = ""
    loadWebPage(myUrl)
}
```

[스위프트 문법] '!' 연산자

!는 논리 연산자의 하나로 논리 NOT 연산자

'!' 연산자를 사용하면 true 는 false, false 는 true 가 됨

'!' 연산자 뒤에는 논리 값 (true 또는 false)을 가지는 Bool 형의 변수나 상수가 와야 함