**Beginning Programing iOS**

# 1. Bắt đầu với iOS

Với việc các SmartPhone ngày càng phổ biến rộng rãi, mỗi người đều dễ dàng để sở hữu cho mình 1 SmartPhone sành điệu, thì việc phát triển các dụng trên các nền tảng di động sẽ còn phát triển mạnh mẽ hơn nữa.

Khái niệm SmartPhone gắn liền với nhiều hãng sản xuất trên thế giới, nhưng đi tiên phong trong lĩnh vực này và đã rất thành công thì không thể không nhắc tới Apple. Và không thể không nói tới iphone, ipod, ipad cùng hệ điều hành iOS.

Với thị phần lớn của mình, cùng những hứa hẹn trong phát triển ứng dụng trên nền tảng iOS đã thu hút rất nhiều Developer tham gia phát triển. Và nếu bạn lỡ yêu thầm Quả táo khuyết hay trở thành tín đồ của nó thì việc dấn than vào Lập trình iOS sẽ còn làm bạn thấy tuyệt vời hơn nữa.

Đó là những chia sẻ gắn gọn nhất của bản thân tôi về iOS, và sau đây, với tài liệu này, tôi xin được trình bày cho các bạn những kiến thức cơ bản nhất về lập trình iOS.

## 1.1 Công cụ lập trình iOS

Để lập trình iOS, bạn cần có các công cụ như sau:

+ **Hệ điều hành MAC OS X**. Sẽ thật là tốt nếu bạn có cho mình những iMac, Macbook pro, Macbook Air. Nhưng nếu không có bạn có thể sử dụng máy tính cảu mình cài đặt máy ảo chạy MAC OS X

+ **Xcode**: Editor lập trình iOS, nó là công cụ cực kì mạnh. Hãy làm quen và để thấy được sức mạnh của nó.

+ **Ngôn ngữ lập trình Objective-C** (có hiểu biết về Objective-C sẽ là vô cùng tuyệt vời khi bạn lập trình iOS, nhưng nếu không có cũng không sao vì bạn sẽ được học Objective-C qua các ví dụ tôi sẽ giới thiệu, chỉ cần bạn có tình yêu, kiên trì với iOS).

## 1.2 Công việc đầu tiên

Để bắt đầu với những ứng dụng đầu tiên thì bạn cần làm những công việc sau:

+ Cài đặt MAC OS X

+ Cài đặt Xcode

+ Đọc tài liệu hướng dẫn của tôi và hãy cố gắng làm theo hướng dẫn.

# 2. Ứng dụng iOS đầu tiên

Chúng ta sẽ cùng nhau làm ứng dụng đầu tiên trên iOS, ứng dụng khi học ngôn ngữ bất kì nào bạn đều đã trải qua. Ứng dụng **Hello World.**

**Lưu ý: Trong tài liệu này tôi sẽ hướng dẫn các bạn lập trình trên iOS >=5, sử dụng Xcode 4.6.1. Các bạn sẽ sử dụng những cải tiến của iOS 5 và sau nữa đó là ARC và StoryBoard.**

**Bước 1: Mở ứng dụng Xcode**

Giao diện của Xcode sẽ xuất hiện như sau:



+ Bên tay phải là danh sách các Project bạn đã mở gần đây. Bạn có thể Click vào bất kì Project nào bạn muốn chỉnh sửa.

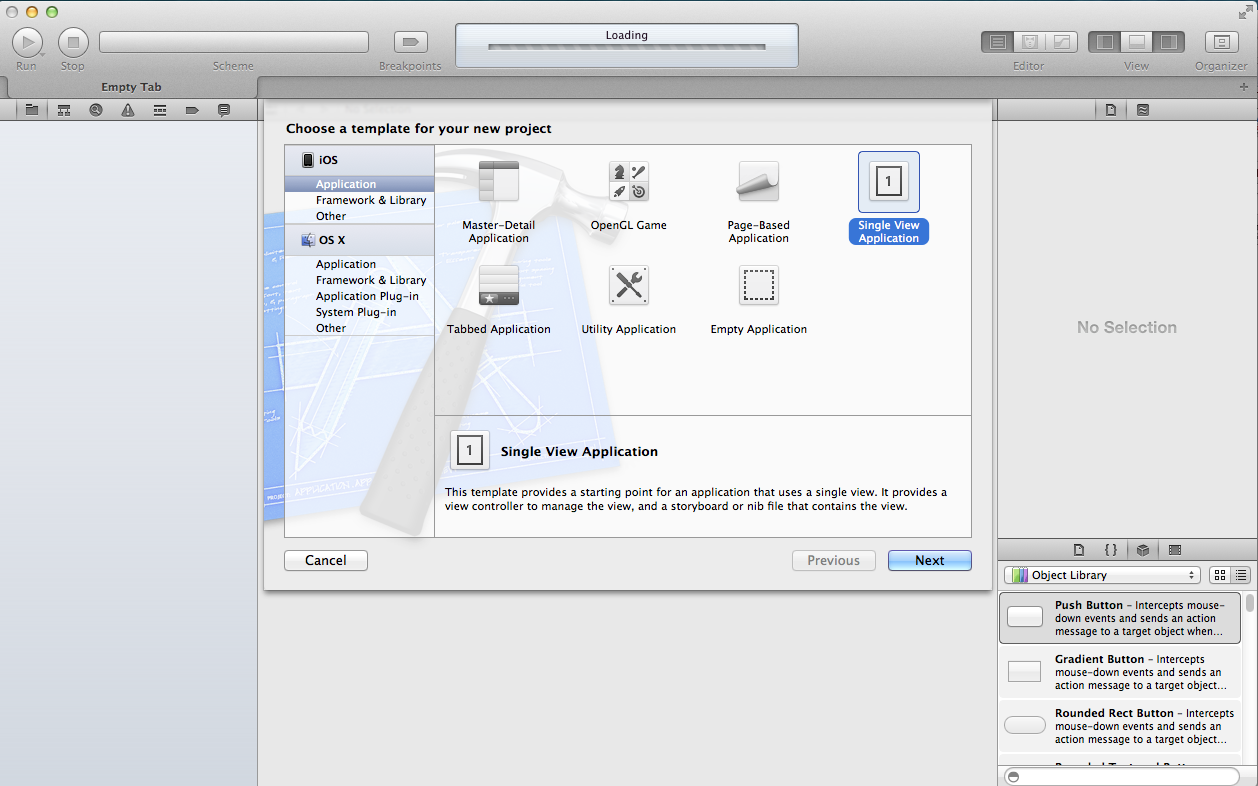
+ Bạn có thể Click vào Button **Open**  để dẫn đến Project mà bạn muốn mở.

+ Bạn có thể Click **Create a new Xcode project** để tạo mới một Project.

Ở ví dụ này ta sẽ tạo mới hoàn toàn một ứng dụng.

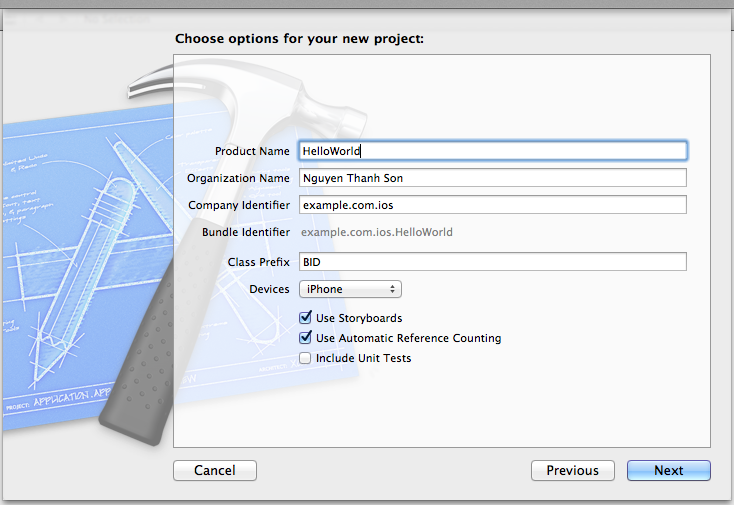
**Bước 2: Tạo một ứng dụng mới trên Xcode**

Sau khi Click vào tạo mới một ứng dụng, giao diện của Xcode sẽ như sau:



+ Cột bên trái thể hiện nền tảng bạn muốn lập trình là iOS hoặc MAC OS. Ở đây, ta sẽ chọn **Application** của mục **IOS.** Tiếp đến chọn **Single View Application.** Click **Next.**

Màn hình tiếp theo sẽ hiển thị như sau:



+ **Product Name:** Tên của Project

**+ Organization Name:** Tên máy tính của bạn (Mặc định). Nó sẽ là Copyright cho ứng dụng của bạn.

+ **Company Identifier:** Định danh cho project của bạn

+ **Class Prefix:** Các lớp trong project của bạn đều được thêm các tiền tố bạn nhập vào. Mặc định là BID (Beginning iPhone Development), bạn có thể thay đổi tuỳ ý giá trị này. Bạn nên đặt giá trị cho mục này, việc này sẽ cực kì hữu ích khi bạn muốn phân biệt các đội trong một dự án, phân biệt các công ty và phân biệt các class bạn tự viết với các class bạn thêm vào.

+ **Device:** Chỉ ra iDevice bạn muốn ứng dụng của mình chạy trên đó (iphone, ipad).

+ **Use StoryBoard:** Là một cải tiến từ iOS 5 trở lên:

* **Bỏ đánh dấu**: File giao diện của ứng dụng sẽ có dạng .xib. Hỗ trợ tất cả các thế hệ iOS. Nhưng đã có những dấu hiệu cảu sự cổ lỗ xĩ.
* **Đánh dấu**: File giao diện của ứng dụng sẽ có dạng .storyboard. Sử dụng StoryBoard không chỉ có thể giúp bạn dễ hình dung ra cái gì bạn đang viết mà còn cho bạn thấy cái nhìn khái quát cả hệ thống của bạn đang sử dụng. Storyboard nếu bạn sử dụng tốt sẽ giúp bạn rút ngắn thời gian lập trình khoảng 30%. Dùng StoryBoard có nhiều đoạn không cần phải code mà vẫn chạy tốt. Nhưng nó có yếu điểm là chỉ hỗ trợ các phiên bản iOS >=5.

Ở tài liệu này, tôi xin hướng dẫn các bạn sử dụng Storyboard để thiết kế giao diện cảu ứng dụng. Nếu ai vẫn dung .xib cũng đừng quá lo lắng vì tại liệu này vẫn có thể giúp ích được cho bạn. Vì cơ bản là không có nhiều sự khác biệt về hình thức của nó.

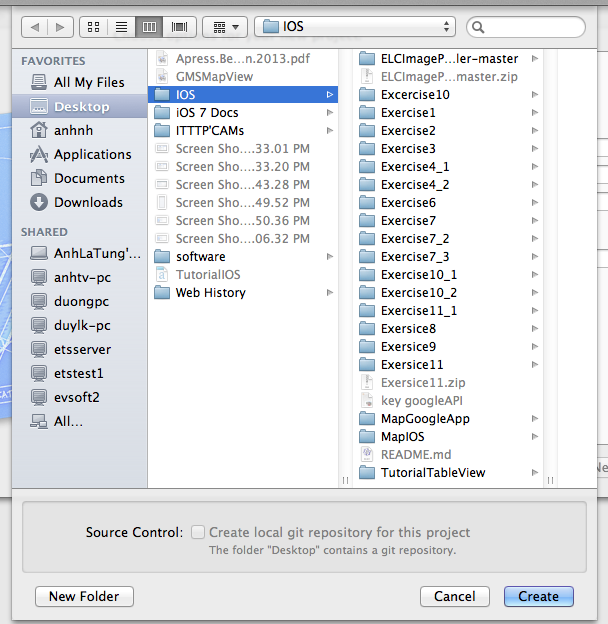
+ **Use Automatic Reference Counting (ARC):** Là một cải tiến của Xcode 4.2 trở lên:

* **Bỏ đánh dấu**: Không sử dụng ARC cho ứng dụng của bạn, khi đó bạn phải tự quản lý bộ nhớ trong ứng dụng cảu mình. Việc khai báo sẽ trở nên phức tạp hơn. Bạn luôn phải đảm bảo đối tượng sẽ phải được thu hồi lại bộ nhớ sau khi nó đã hoàn thành xong công việc cảu mình.
* **Đánh dấu**: Sử dụng ARC. ARC là một tính năng cực hay của Xcode 4.2 trở lên. Nó không chỉ giúp lập trình viên làm việc dễ dàng hơn (không cần biết đến việc quản lý bộ nhớ) mà nó còn giúp cho bạn tránh được những lỗi không đáng có. ARC giúp bạn tập trung vào những tính năng của chương trình hơn là cứ phải chăm chăm vào những thứ không đáng. Hãy sử dụng những công nghệ mới đừng chăm chăm vào những thứ cũ kĩ mà phung phí thời gian làm những cái mà lẽ ra máy làm rồi.

Ở tài liệ này tôi cũng sẽ sử dụng ARC.

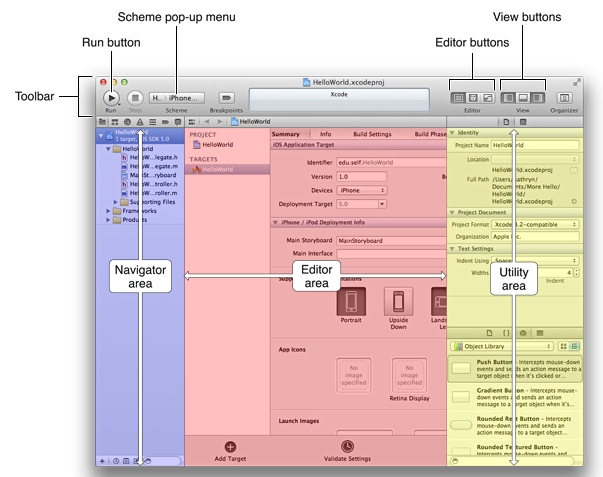
+ **Include Unit Test:** cho phép bạn xác định khi có một sự thay đổi thực hiện cho code của bạn phá vỡ một cái gì đó đã được làm việc trước đó. Mặc dù nó có thể là một công cụ có giá trị, nhưng tôi sẽ không sử dụng chúng trong tài liệu này.

Sau khi điền đầy đủ thông tin cho Project của mình, bạn Click **Next**. Màn hình tiếp theo sẽ hiển thị:



Ở đây, bạn sẽ lựa chọn nơi lưu trữ project của mình và chọn **Create.** Như vậy là bạn đã tạo mới xong một project mới trong Xcode.

Không gian làm việc của Xcode sẽ như sau:



+ **Toolbar:** chứa các công cụ trong lập trình iOS:

* **Run button:** chạy ứng dụng, một máy ảo sẽ chạy và mô phỏng ứng dụng của bạn
* **Stop button:** Tạm dừng chạy ứng dụng
* **Scheme pop-up menu:** Lựa chọn thiết bị mô phỏng ứng dụng (iphone 4, ipad,…)
* **Editor button:** kiểu xem màn hình Editor (có thể là 1 hoặc 2 màn hình editor).
* **View button:** kiểu xem giao diện cảu Xcode.

+ **Navigator area:** cung cấp bảy cấu hình cho bạn cái nhìn khác nhau vào project của bạn. Nhấp vào một trong các biểu tượng ở phía trên cùng của Navigator View để chuyển đổi giữa các Navigator, lần lượt từ trái sang phải:



* **Project navigator:** có chứa một danh sách các file được sử dụng bởi project của bạn.
* **Symbol navigator**: tập trung vào những biểu tượng được định nghĩa trong không gian làm việc. Biểu tượng chứa mục mà trình biên dịch nhận ra, chẳng hạn như các class Objective-C, cấu trúc, và các biến toàn cầu.
* **Search navigator**: thực hiện tìm kiếm trên tất cả các tập tin trong không gian làm việc của bạn
* **Issues navigator:** Khi bạn xây dựng dự án của bạn, bất kỳ lỗi hoặc cảnh báo sẽ xuất hiện trong navigator này, và thông báo chi tiết số lượng các lỗi sẽ xuất hiện trong giao diện hoạt động ở phía trên của cửa sổ. Khi bạn nhấp vào một lỗi, bạn sẽ nhảy vào dòng code tương ứng xảy ra lỗi.
* **Debug navigator:** Chi tiết các luồng, trình tự biên dịch khi debug ứng dụng
* **Breakpoint navigator**: cho phép bạn xem tất cả các điểm dừng mà bạn đã thiết lập.
* **Log navigator**: giữ một lịch sử của kết quả build gần đây và các bản ghi chạy.

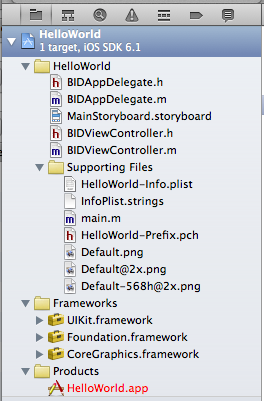
**+ Editor area:** Hiển thị các file, các class bạn đã lựa chọn bên navigator view. Tại đây ta có thể Edit các class nếu muốn.

**+ Utility area:** hiển thị thuộc tính của các đối tượng (UILabel, UIButton,…) cho phép ta có thể Edit và Librarian Object.

Chúng ta đã khám phá không gian làm việc của Xcode, Bây giờ, chúng ta hãy nhìn vào các file tạo nên project **Hello World**. Chuyển sang **project navigator** bằng cách nhấn vào tận cùng bên trái trong bảy biểu tượng ở phía bên trái của **navigator view.** Ta có thể thấy các file trong project bao gồm:

+ Mục đầu tiên trong navigator mang tên tương tự như tên project của bạn, trong trường hợp này, HelloWorld. Nó đại diện cho toàn bộ project của bạn, và nó cũng là nơi các thiết lập có thể được thực hiện. Nếu bạn nhấp chuột duy nhất, bạn sẽ có thể chỉnh sửa một số thiết lập trong trình soạn thảo project của Xcode. Bạn không cần phải lo lắng về các thiết lập cụ thể bây giờ, bởi vì, tại thời điểm này, giá trị mặc định sẽ làm việc tốt.

+ Khi Click vào mục tam giác của mục đầu tiên ấy sẽ xuất hiện các thư mục con, bao gồm:



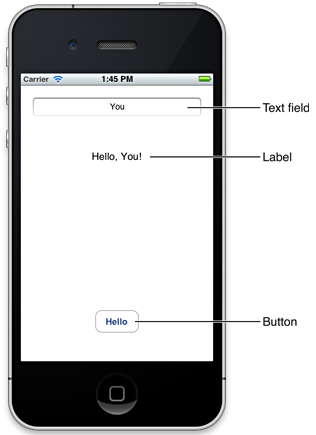
* **HelloWorld**: Các thư mục đầu tiên, mà luôn luôn được đặt theo tên project của bạn, là nơi mà bạn sẽ dành phần lớn thời gian của bạn. Đây là nơi mà hầu hết các đoạn code mà bạn viết nói chung sẽ chạy, như các file sẽ tạo nên giao diện người dùng của ứng dụng. Bạn được tự do tạo ra các thư mục con trong thư mục Hello World để giúp tổ chức đoạn code của bạn, và bạn thậm chí còn cho phép sử dụng các nhóm khác nếu bạn thích một cách tổ chức khác nhau. Ở ví dụ này, ta sẽ giữ nguyên như mặc định mà không có bất cứ thay đổi gì.
  + **MainStoryboard.storyboard**: là file giao diện cảu ứng dụng
  + Một file \*.h định nghĩa trước các biến thành phần và tên các phương thức, file \*.m định nghĩa phần thực thi cho các phương thức trong file \*.h
  + **ViewController**: đóng vai trò trung tâm trong hầu hết các ứng dụng. Chúng chịu trách nhiệm quản lý một khung nhìn, trợ giúp điều hướng và quản lý bộ nhớ.
  + **AppDelegate**: là một đối tượng tùy chỉnh được tạo ra ở thời gian khởi động ứng dụng, thường là do chức năng UIApplicationMain. Công việc chủ yếu của đối tượng này là để xử lý chuyển trạng thái trong ứng dụng.
* **Supporting Files:** Thư mục này chứa các tập tin mã nguồn và tài nguyên mà không phải là các class Objective-C nhưng đó là cần thiết cho project của bạn. Thông thường, bạn sẽ không dành nhiều thời gian trong thư mục Other Sources. Khi bạn tạo một ứng dụng iOS, thư mục này chứa bốn tập tin quan trọng:
  + **HelloWorld-Info**.plist: là một danh sách có chứa thông tin về các ứng dụng.
  + **InfoPlist.strings**: là một tập tin văn bản có chứa string có thể đọc được mà có thể được tham chiếu trong danh sách thông tin. Không giống như các danh sách thông tin riêng của nó, tập tin này có thể được nội địa hóa, cho phép bao gồm nhiều bản dịch ngôn ngữ trong ứng dụng của bạn.
  + **main.m** chứa phương thức main() của ứng dụng của bạn. Thông thường, bạn sẽ không cần phải chỉnh sửa hoặc thay đổi tập tin này. Trong thực tế, nếu bạn không biết những gì bạn đang làm, nó thực sự là một ý tưởng tốt không để chạm vào nó.
  + **Hello\_World\_Prefix.pch** là một danh sách các tập tin tiêu đề từ khung bên ngoài được sử dụng bởi project của bạn.

Nhìn chung các mục trong thư mục này bạn chỉ cần biết về nó, và không cần quá cố gắng hiểu kĩ về nó. Bởi như đã đề cập ở trên thì đây là phần mà ta ít tác động tới.

* **Frameworks:** Thư mục này chứa các thư viện có thể chứa mã cũng như các nguồn tài nguyên, chẳng hạn như hình ảnh và các tập tin âm thanh. Bất kỳ framework hoặc thư viện nào mà bạn thêm vào thư mục này sẽ được kết nối vào ứng dụng của bạn, và code của bạn sẽ có thể sử dụng bất kỳ đối tượng, chức năng, và tài nguyên chứa trong framework hoặc thư viện đó. Các khuôn khổ phổ biến nhất cần thiết được kết nối vào project của bạn theo mặc định, vì vậy hầu hết thời gian, bạn sẽ không cần thêm bất cứ điều gì vào thư mục này. Nếu bạn cần các thư viện và các khuôn khổ ít được sử dụng, thật dễ dàng để thêm chúng.
* **Products**: Thư mục này chứa ứng dụng được sản xuất ra khi nó được xây dựng.

**Bước 3: Hoàn thành ứng dụng theo yêu cầu**

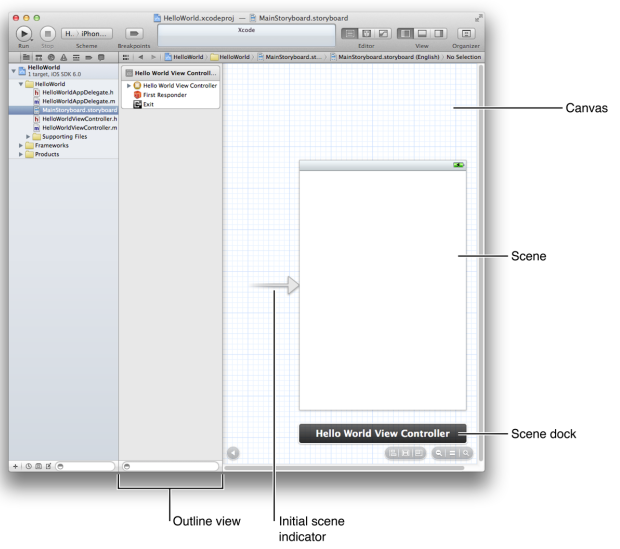
1. Yêu cầu về ứng dụng HelloWorld của chúng ta như sau:



1. **Phân tích:** Với ứng dụng đơn giản này thì chúng ta chỉ cần làm việc với file giao diện của ứng dụng là **MainStoryboard.storyboard** là đủ. Với những ứng dụng cần có thêm những đoạn code trên ViewControler sẽ được giới thiệu ở những ứng dụng tiếp sau. Các thành phần trong giao diện bao gồm:

* 1 UITextField
* 1 UILabel
* 1 Button

1. Khi bạn click vào File giao diện của ứng dụng là **MainStoryboard.storyboard** nằm trong thư mục HelloWorld thuộc project navigator. Thì tại Editor area sẽ xuất hiện file giao diện mà ta có thể tiến hành chỉnh sửa.



Một storyboard chứa các cảnh (sences) và các chuyển dịch mượt (segues) của view. Một cảnh biểu diễn một view controller và một chuyển dịch mượt biểu diễn một sự di chuyển giữa các cảnh.

Bởi vì project thuộc loại Single View cung cấp cho ta chỉ mộtview controller nên storyboard trong ứng dụng của bạn chỉ chứa một cảnh và một di chuyển. Mũi tên chỉ về phía bên trái của khung cảnh là initial scene indicator, nó chỉ ra cảnh được load ra lần dầu tiên khi ứng dụng bắt đầu.

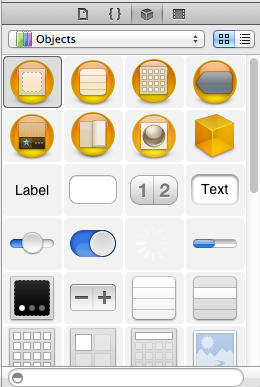
Cảnh mà bạn thấy trên canvas được đặt tên là “Hello World View Controller” bởi vì nó được quản lý bởiHelloWorldViewController.

Nếu giữ nguyên hiện trạng của file giao diện và click vào Button **Run** để tiến hành chạy ứng dụng thì ta sẽ nhận được màn hình mô phỏng như sau:



1. **Thực hiện:** Bây giờ ta sẽ tiến hành Add các Object như yêu cầu của ứng dụng

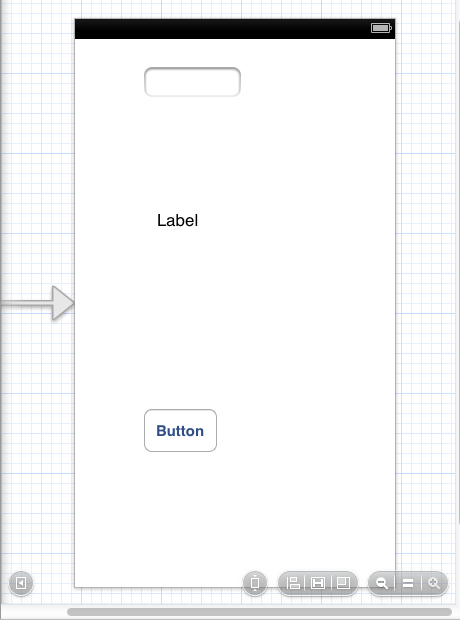
Các đối tượng sẽ nằm ở trong Utility Area:



Bạn có thể click chuột trái vào các Object để biết tên của các Object đó là gì.

Với yêu cầu của ứng dụng ở trên ta sẽ tìm đến UITextField, UILabel, UIButton để thêm chúng vào giao diện. Việc thêm chúng rất đơn giản, khi ta chỉ việc nhấn và giữ chuột trái vào đối tượng muốn Add vào giao diện rồi kéo thả chúng vào file giao diện.

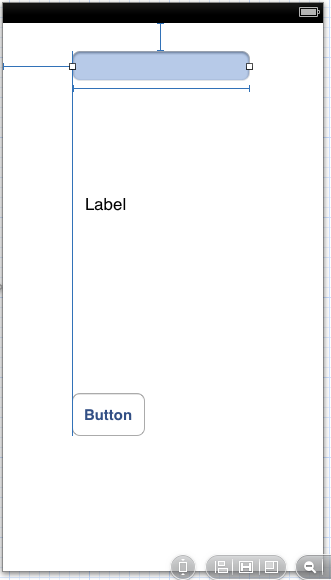
Giả sử bạn đã kéo thả được các đối tượng bạn mong muốn như hình sau:



Công việc tiếp theo mà bạn cần làm đó là chỉnh vị trí cũng như chỉnh Text hiển thị trên các đối tượng theo như yêu cầu đặt ra của ứng dụng.

Để chỉnh sửa vị trí, kích thước của các đối tượng bạn có thể sử dụng 2 cách:

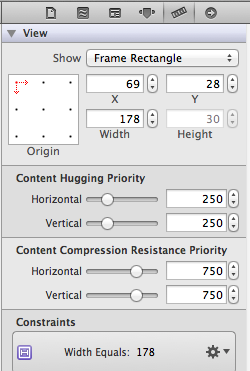
* **Sử dụng chuột:**
* Click và giữ chuột trái vào đối tượng muốn di chuyển, di chuyển đến vị trí bạn mong muốn.
* Click chuột trái vào một đối tượng bạn muốn chỉnh sửa kích thước. Bạn sẽ thấy xuất hiện các thanh cho thay đổi kích thước của đối tượng và công việc cảu bạn là kéo dãn đối tượng theo kích thước bạn mong muốn.



* **Sử dụng thanh công cụ trong Utility Area**



* + Lần lượt từ trái qua phải là các đối tượng:
    - **File Inspector**: Hiển thị các thông tin về Identifier và Interface Build Document.
    - **Quick help inspector**: Miêu tả các đối tượng
    - **Identity inspector:** hiển thị về class đối tượng thuộc về, identity và các thông số về Accessibility.
    - **Attributes inspector**: Các thuộc tính của đối tượng: Text, Color, Background,...
    - **Size inspector:** Các thông số về kích thước, vị trí của đối tượng
    - **Connection inspector**: Các kết nối giữa các đối tượng
  + Sử dụng thanh công cụ trong Utility area: Click chuột trái vào đối tượng muốn thay đổi vị trí và kích thước
    - Click chuột trái vào Size inspector
    - Điền các thông số về kích thước và vị trí của đối tượng theo ý muốn của bạn



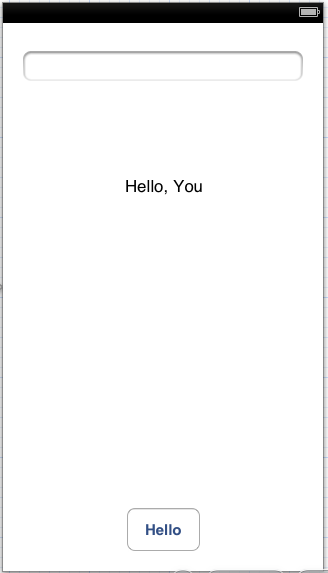
Sau khi căn chỉnh kích thước và vị trí của các đối tượng theo yêu cầu đặt ra ta sẽ có file giao diện như sau:



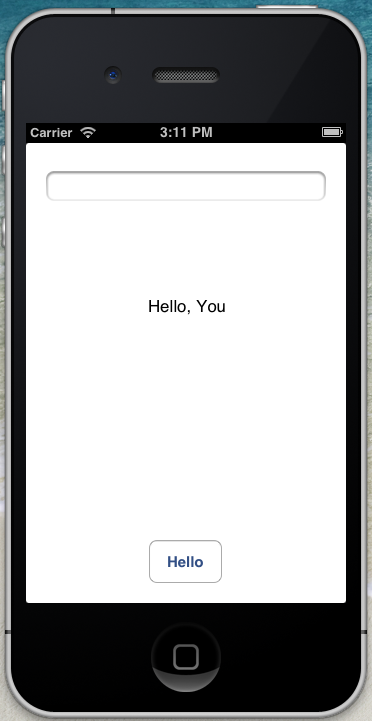
Đó là về chỉnh sửa vị trí và kích thước của đối tượng, để chỉnh sửa Text, Color, ... các thuộc tính hiển thị của đối tượng, ta tiến hành như sau:

* Click chuột trái vào đối tượng muốn chỉnh sửa
* Click chuột trái, chọn Attributes inspector
* Chỉnh sửa các thuộc tính cảu đối tượng đã chọn theo yêu cầu

Hoàn thành các yêu cầu về hiển thị của các đối tượng, ta thu được màn hình giao diện:



Như vậy, ta đã hoàn thành những yêu cầu đặt ra của ứng dụng ban đầu, tiến hành click button **Run** để xem thành quả bạn đã làm.



Với ví dụ đầu tiên, cõ lẽ bạn đã biết cách làm thế nào để tạo mới và chạy một ứng dụng đơn giản. Với các tài liệu tiếp theo tôi sẽ chia sẻ kĩ hơn với các bạn về các basse trên lập trinh iOS

# 3. UITableView

## 3.1 Giới thiệu về UITableView

Có lẽ một UIView được sử dụng nhiều nhất trong các ứng dụng của iOS chính là UITableView. Và ở phần này tôi xin được trình bày một cách kĩ càng nhất những hiểu biết của tôi về UITableView.

Đầu tiên chúng ta hãy cùng xem sét một số giao diện của UITableView:



UITableView hiển thị danh sách các dữ liệu. Mỗi mục trong danh sách của bảng là một hàng. Bảng iOS có thể có một số lượng không giới hạn của các hàng, chỉ hạn chế bởi số lượng bộ nhớ có sẵn. TableView chỉ cso một cột duy nhất.

Một Table View là đối tượng View có hiển thị dữ liệu của bảng và là một thể hiện của class UITableView. Mỗi hàng có thể nhìn thấy bảng được thực hiện bởi class UITableViewCell. Vì vậy, một cái nhìn bảng là đối tượng hiển thị nhìn thấy một phần của một bảng, và một cell chỉ hiển thị một hàng của bảng.

Tableview không chịu trách nhiệm lưu trữ dữ liệu bảng. Nó chỉ lưu trữ đủ dữ liệu để vẽ các hàng hiện đang nhìn thấy được. Dự liệu TableView được cấu hình từ một đối tượng phù hợp với giao thức UITableViewDelegate và dữ liệu hàng của nó từ một đối tượng phù hợp với giao thức UITableViewDataSource.

Như đã đề cập, tất cả các bảng được thực hiện như một cột duy nhất. Nhưng mỗi hàng trong bảng được đại diện bởi một UITableViewCell duy nhất. Mỗi đối tượng UITableViewCell có thể được cấu hình với một hình ảnh, một số văn bản, và một biểu tượng tùy chọn, đó là một biểu tượng nhỏ ở bên phải.

Các loại UITableView bao gồm:

* Grouped Table: Nhóm một số dữ liệu có điểm chung thành một nhóm
* Plain Table: Mặc định của UITableView, hiển thị các dữ liệu một cách lần lượt, không tách biệt thành các nhóm dữ liệu.



Từ trái qua phải: Group TableView, Plain TableView không có chỉ số, Plain TableView có chỉ số.

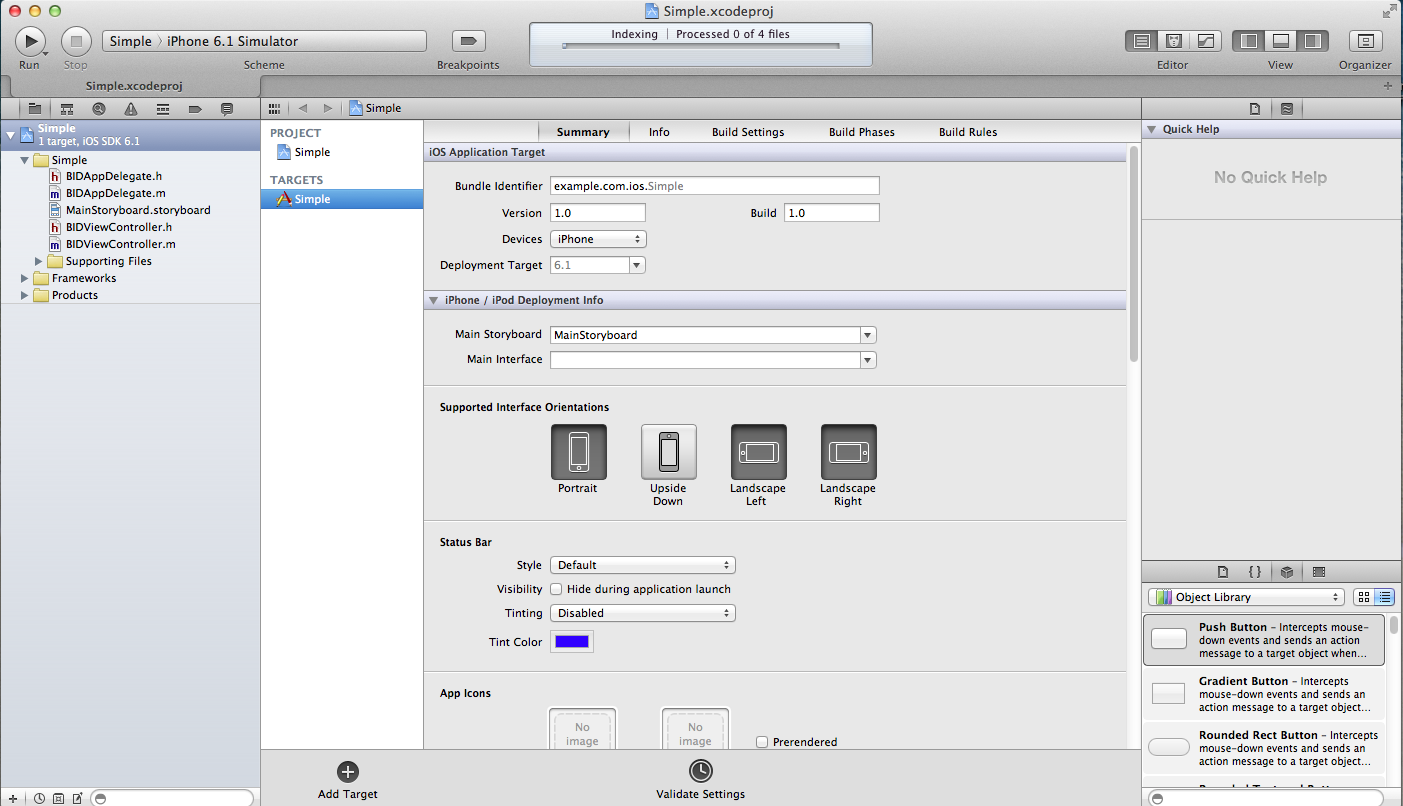
## 3.2 Hiển thị UITableView đơn giản

Việc tạo một project mới là hoàn toàn giống với ví dụ đầu tiên HelloWorld.

Để hiển thị một UITableView, ta sẽ tiến hành làm theo các bước sau:

* Add một UITableView
* Khai báo một mảng (NSArray) chưa dữ liệu của UITableView
* Thực hiện hiển thị UITableView thông qua các hàm của UITableViewDelegate, UITableViewDataSource.

Giả sử ta đã tạo thành công một project mới với tên là “**Simple Table**”

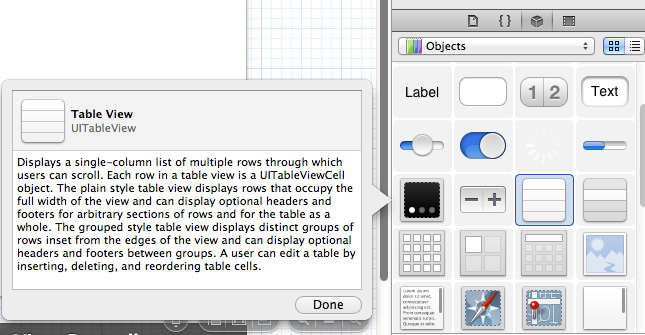


Yêu cầu của bài toán: dùng UITableView hiển thị một mảng danh sách là {“One”, “Two”, “Three”, “Four”, “Five”, “Six”, “Seven”, “Eight”, “Night”, “Ten”, “Eleven”, “Twelve”, “Thirdteen”, “Fourteen”}.

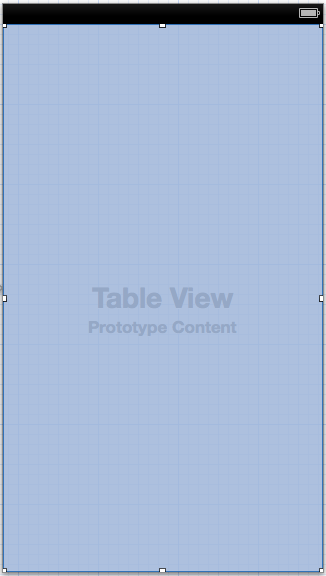
Chúng ta sẽ tiến hành giải quyết bài toán này theo lần lượt từng bước đã nêu ở trên.

### 3.2.1 Add UITableview

Click vào file giao diện **MainStoryboard.storyboard,** để tiến hành Add một UITableView vào giao diện của ứng dụng. Cũng tương tự như ví dụ một, ta tiến hành chọn đối tượng ta muốn thêm vào giao diện:

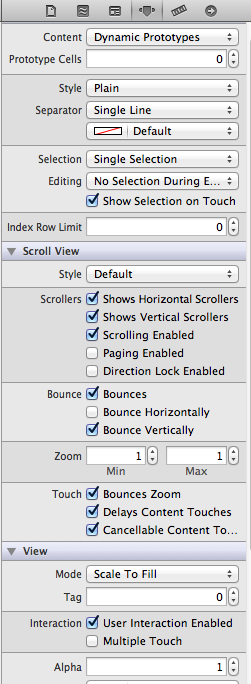


Tiếp theo, ta tiến hành kéo thả UITableView vào file giao diện



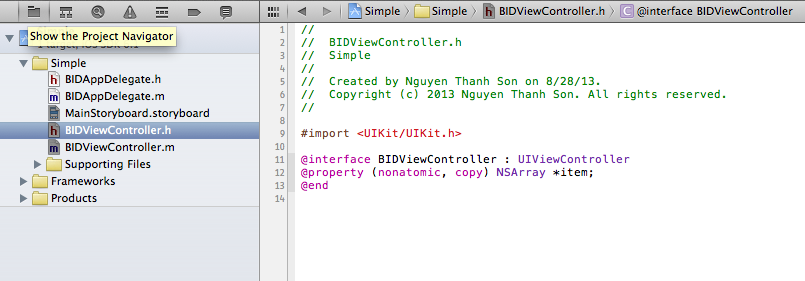
Khi kéo thả UITableView vào file giao diện cần đảm bảo UITableView đã Constrain vào cả 4 cạnh của file giao diện (để đảm bảo rằng UITableView là vừa khít với giao diện không bị lệch).

Click vào **Attributes inspector** để tiến hành những khai báo về thuộc tính cho TableView vừa được thêm vào giao diện. Ở ví dụ này, tôi sẽ chọn loại TableView là Plain TableView, với các thông số khác, tôi sẽ để mặc định.



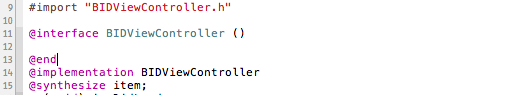
### 3.2.2 Mảng dữ liệu của UITableView

+ Khai báo thuộc tính **NSArray \*item** (chứa dữ liệu của UITableView) trong file **BIDViewController.h** (định nghĩa trước các biến thành phần và tên các phương thức để các đối tượng của **BIDViewController** có thể truy cập).



+ Trong file **BIDViewController.m** nếu muốn truy xuất đến **item,** có 2 cách như sau:

* Truy xuất thông qua **self.item**
* Khai báo **@synthesize** trong **BIDViewController.m** và sử dụng trực tiếp từ khoá **item** để truy xuất biến **item**



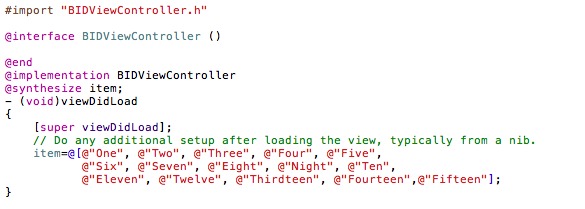
+ Khai báo nội dung của **item.** Ở đây, tôi sẽ add trực tiếp nội dung vào mảng **item.** Việc khai báo này sẽ được thực hiện trong phương thức **- (void)viewDidLoad**

item=@[@"One", @"Two", @"Three", @"Four", @"Five",

@"Six", @"Seven", @"Eight", @"Night", @"Ten",

@"Eleven", @"Twelve", @"Thirdteen", @"Fourteen",@"Fifteen"];

Và đây là code trong chương trình:



### 3.2.3 Hiển thị UITableView

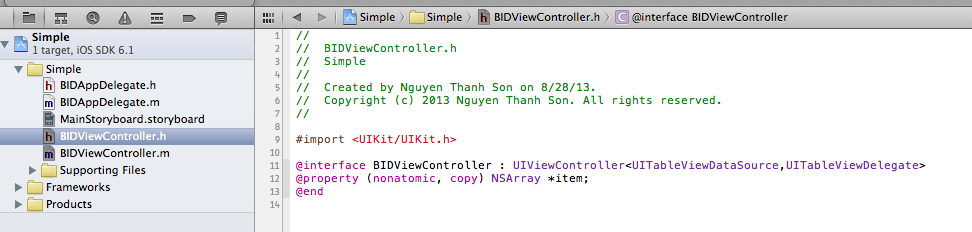
#### 3.2.3.1 Khai báo dataSource, delegate

+ Sau khi thực hiện khai báo một UITableView và một mảng chứa dữ liệu của bảng, công việc tiếp theo ta cần làm là khai báo **UITableViewDelegate, UITableViewDataSource.**

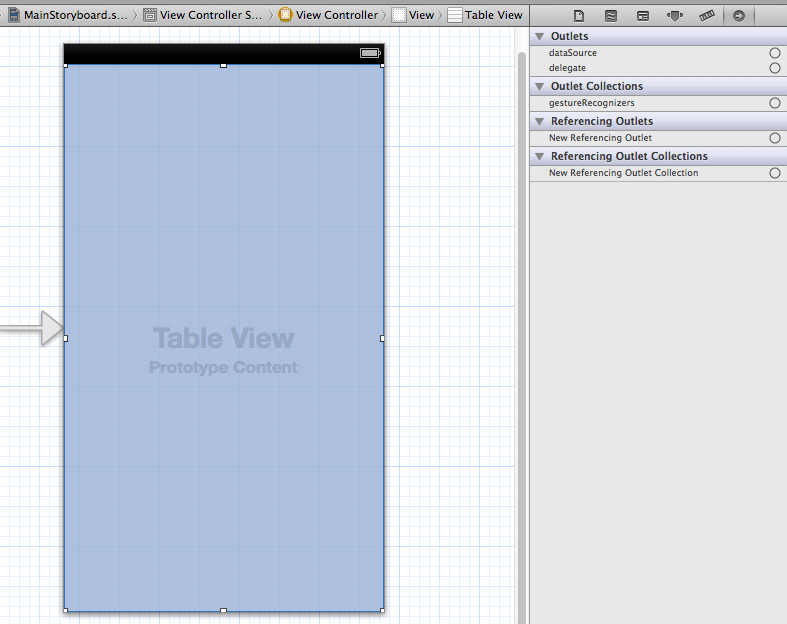
* Với **dataSource** thì việc khai báo này sẽ giúp dữ liệu của bạn có thể liên kết dược với nhau
* Với **Delegate** thì bạn có thể hiểu như sau: Bạn có 2 class A và B và có 2 object a từ class A và b từ class B. Khi bạn khai báo b.delegate = A, khi đó nếu bạn đang trong class B, bạn có thể gọi hàm từ class A để sử dụng vì bạn đã truyền delegate của A vào B. Ở đây, khi bạn khai báo **UITableViewDelegate** (một **delegate** mặc định trong **UITableView**) trong **BIDViewController.h** thì bạn có thể thực hiện các phương thức có sẵn trong **BIDViewController.m** để hỗ trợ hiển thị đắc lực trong hiển thị **UITableView**.

Khai báo **dataSource** và **delegate** cho **TableView** như sau:

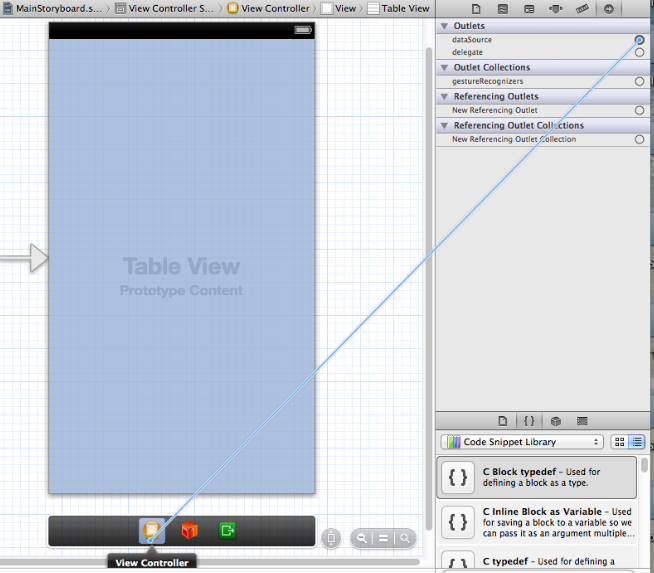
* Khai báo trong **BIDViewController.h**



* Kéo thả trong file giao diện:
  + Click chuột trái vào file giao diện **MainStoryboard.storyboard** trong **Project Navigator**
  + Chọn UITableView rồi click vào **connection inspector,** ta sẽ thấy giao diện làm việc như sau:



* + Kéo thả **dataSource** và **delegate** vào UITableView



* + Sau khi kéo thả thành công ta sẽ thu được kết quả như sau:



#### 3.2.3.2 Thực thi các phương thức

Công việc tiếp theo của hiển thị **UITableView** đó là ta sẽ tiến hành thực thi các phương thức hỗ trợ trong hiển thị **UITableView.** Cấu trúc một UITableview thì bao gồm:

* **Section:** chứa một nhóm các hàng trong UITableView
* **Cell (Row):** Là một hang dữ liệu trong UITableView

Các phương thức thực thi trong hiển thị UITableView cũng xoáy quanh vào việc hiển thị các **Section, Cell:**

**+ Số lượng Section trong một UITableView**

**-(NSInteger )numberOfSectionsInTableView:(UITableView \*)tableView{**

**return 1;**

**}**

Ở đây, ta chọn UITableView có một **Section,** tuỳ vào yêu cầu mà con số trong **return** sẽ được thay đổi.

**+ Số lượng cell trong một Section một UITableView**

**-(NSInteger)tableView:(UITableView \*)tableView numberOfRowsInSection:(NSInteger)section{**

**return [item count];**

**}**

Với yêu cầu bái toán dặt ra như ở trên ta sẽ tạo một UITableView gồm 1 **Section** và số **cell** là số lượng. **[item count]**: đếm số lượng phần tử trong mảng **item.**

**+ Khai báo và hiển thị cell**

**-(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath{**

**static NSString \*SimpleTableIdentifier = @"SimpleTableIdentifier";**

**UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:**

**SimpleTableIdentifier];**

**if (cell == nil) {**

**cell = [[UITableViewCell alloc]**

**initWithStyle:UITableViewCellStyleDefault**

**reuseIdentifier:SimpleTableIdentifier];**

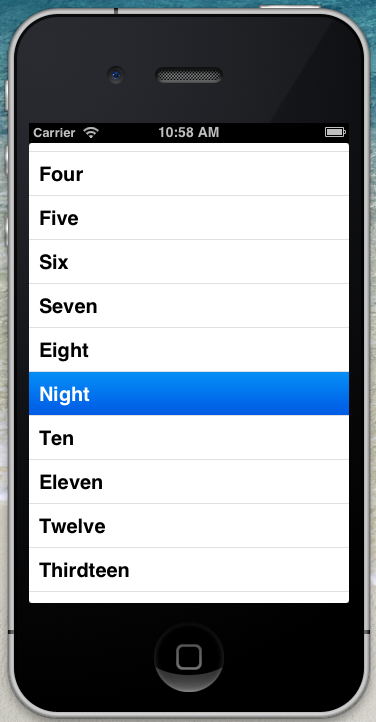
**}**

**cell.textLabel.text =item[indexPath.row];**

**return cell;**

**}**

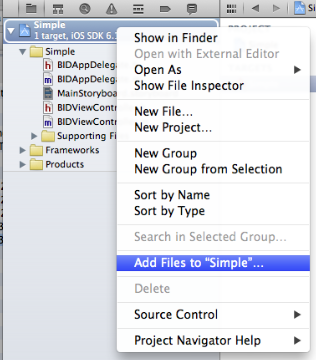
Với việc thực thi 3 phương thức trên, bạn đã có thể có cho mình một TableView như yêu cầu đề bài, Click **Run** để hiển thị kết quả.

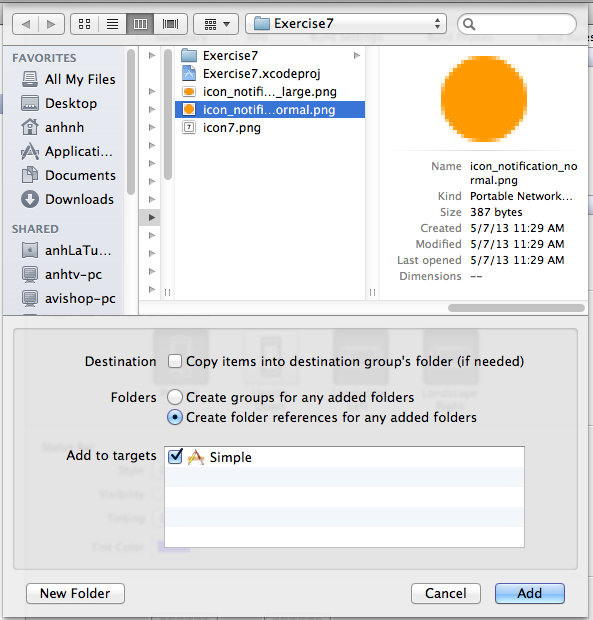


Đó là 3 phương thức nhất thiết cần thực thi để có thể hiển thị đựợc UITableView. Sau đây, tôi sẽ giới thiệu cho bạn biết thêm về các phương thức thực thi trong UITableView.

**+ Thêm ảnh vào cell**

* Thêm ảnh vào project

****

****

Sau đó, thực hiện:

-(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath{

static NSString \*SimpleTableIdentifier = @"SimpleTableIdentifier";

UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:

SimpleTableIdentifier];

if (cell == nil) {

cell = [[UITableViewCell alloc]

initWithStyle:UITableViewCellStyleDefault

reuseIdentifier:SimpleTableIdentifier];

}

**UIImage \*image = [UIImage imageNamed:@"icon\_notification\_normal.png"];**

**cell.imageView.image = image;**

**cell.textLabel.text =item[indexPath.row];**

return cell;

}

Click **Run** để xem kết quả hiển thị.



**Chú thích:** Để giúp bạn có cái nhìn tốt hơn về UITableview, tôi sẽ trình bày cho bạn hiểu về trình tự chạy của Project bạn vừa tạo. Ban đầu khi click **Run** thì ứng dụng sẽ bắt đầu load file giao diện **MainStoryboard.storyboard.** Sau đó, nó sẽ nhảy đến hàm **viewDidLoad()** và thực hiện các khai báo bạn đã thực hiện. Thực hiện xong, nó sẽ thực hiện -(NSInteger )numberOfSectionsInTableView:(UITableView \*)tableView để xác định số **Section** hiển thị trong UITableView. Kế tiếp đó sẽ là -(NSInteger)tableView:(UITableView \*)tableView numberOfRowsInSection:(NSInteger)section xác định số **cell** trong **Section.** Và tiếp nữa nó sẽ thực hiện -(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPat khai báo và duyệt qua từng **cell,** thực hiện các tác vụ lien quan đến mỗi **cell** như đã viết. Khi thực hiện xong các phương thức, UITableView được hiển thị như những gì bạn code.

**+ Hiển thị chú thích cho mỗi cell trong UITableview**

-(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath{

static NSString \*SimpleTableIdentifier = @"SimpleTableIdentifier";

UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:

SimpleTableIdentifier];

if (cell == nil) {

cell = [[UITableViewCell alloc]

initWithStyle:UITableViewCellStyleSubtitle

reuseIdentifier:SimpleTableIdentifier];

}

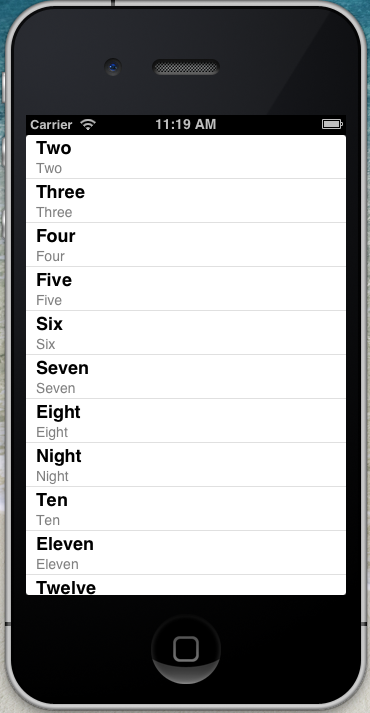
cell.textLabel.text =item[indexPath.row];

cell.detailTextLabel.text =item[indexPath.row];

return cell;

}

Kết quả hiển thị sẽ như sau:



**+ Thay đổi Font chữ cho Text trong cell**

Thực hiện thêm dòng code sau trong phương thức khai báo và hiển thị **cell**

**cell.textLabel.font = [UIFont boldSystemFontOfSize:40];**

****

**+ Thay đổi chiều cao của cell**

-(CGFloat)tableView:(UITableView \*)tableView heightForRowAtIndexPath:(NSIndexPath \*)indexPath{

return 40;

}

**+ Chiều cao của Header Section**

-(CGFloat)tableView:(UITableView \*)tableView heightForHeaderInSection:(NSInteger)section{

return 5;

}

**+ Chiều cao của Footer Section**

-(CGFloat)tableView:(UITableView \*)tableView heightForFooterInSection:(NSInteger)section{

return 5;

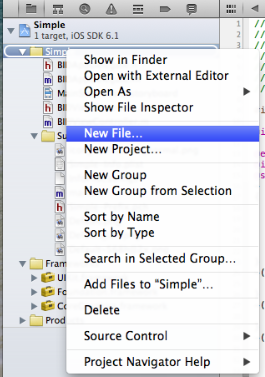
}

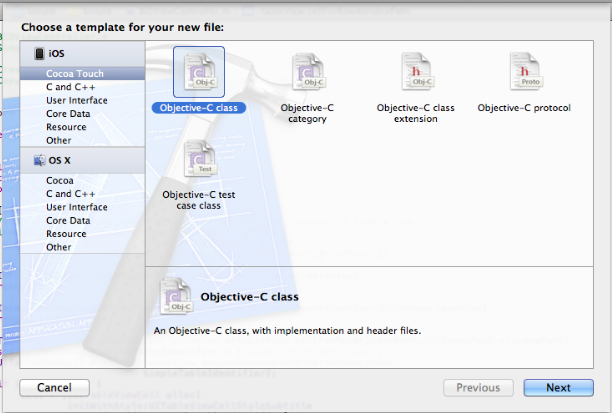
Vừa rồi là những phương thức cơ bản nhất trong hiển thị UITableView, sẽ còn nhiều phương thức khác, chúng sẽ được làm rõ trong các ví dụ tương tác với UITableView sau này.

### 3.2.4 Tạo một cell subclass

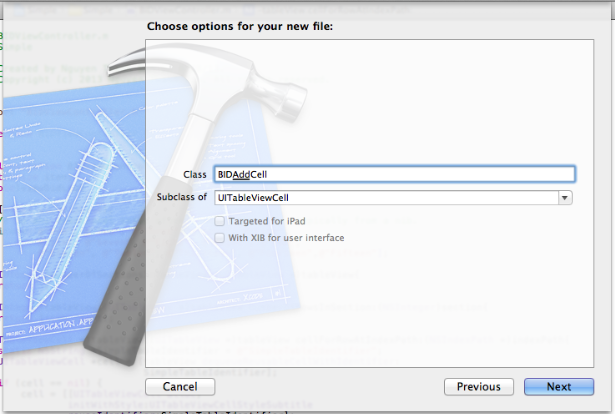
Việc ta khai báo và hiển thị các **cell** ở ví dụ trên chỉ giúp bạn hiển thị một UITableivew duy nhất. Nếu bạn muốn có thể sử dụng lại các **cell** (chỉ cần khai báo một lần, sử dụng nhiều lần) thì bạn cần tạo một lớp riêng khai báo **cell.** Sau đây, tôi sẽ làm lại ví dụ trên bằng cách sử dụng khai báo một **cell subclass.**

**+** Đầu tiên bạn cần tạo một lớp **cell** mới



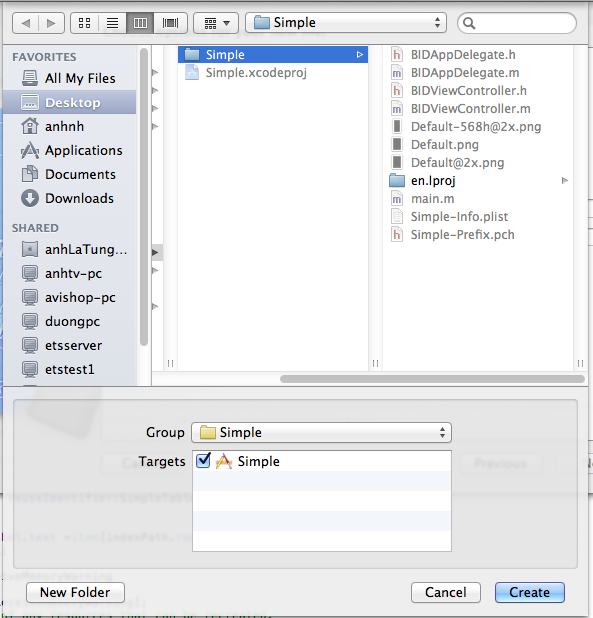


Chọn Objectice-C class, Click **Next.**

****

* **class:** Tên lớp bạn muốn đặt cho lớp mới này
* **Subclass of:** Lớp mới khởi tạo là lớp con của lớp này. Ở đây, ta tạo một lớp khai báo các **cell** trong UITableView nên lớp mới này là lớp con của class UITableViewCell.

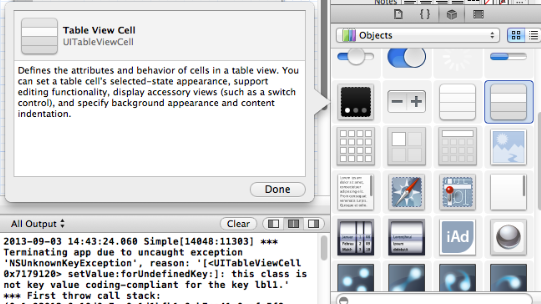
Tiếp tục Click **Next.**

****

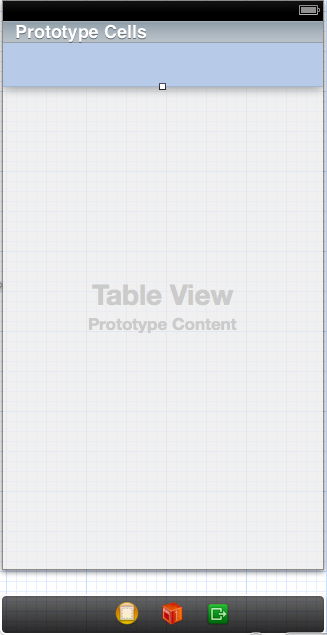
Click **Create** để hoàn tất tạo một lớp mới.

+ Tiếp theo, bạn cần khai báo một **cell** mẫu

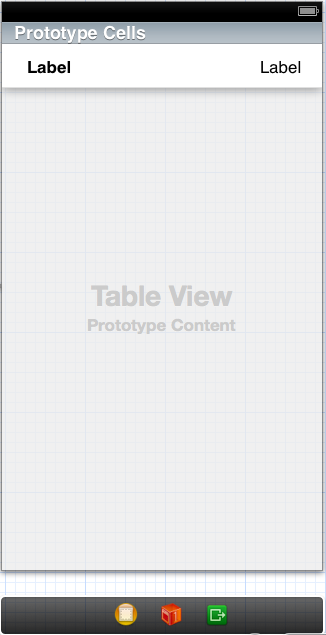
* Chọn file giao diện **MainStoryboard.storyboard.** Trong **Object Library** tìm đền, chọn object UITableViewCell.



* Kéo thả UITableViewCell vào UITableView đã kéo vào file giao diện trước đó.

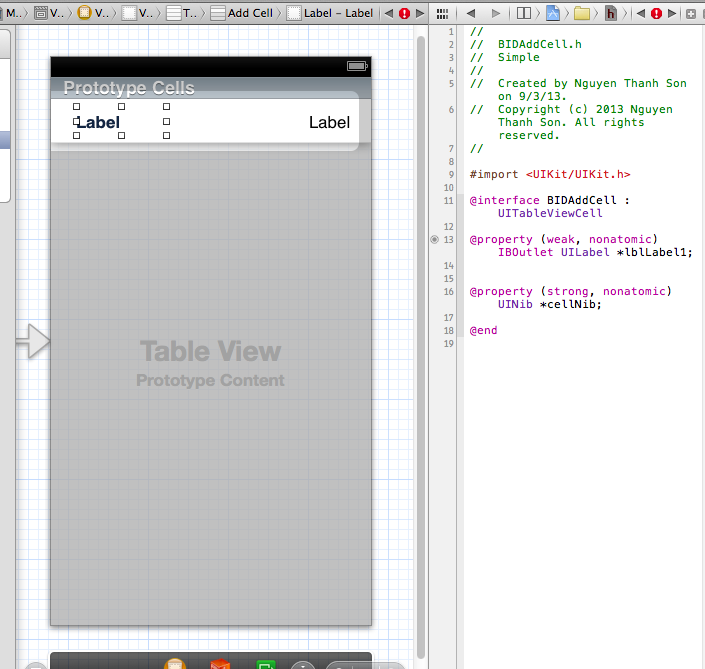
****

* Giả sử rằng, mỗi cell của chúng ta bao gồm 2 UILabel (Label bên trái là giá trị phần tử trong mảng, Label bên phải hiển thị số tương ứng với số dòng). Như vậy, bước tiếp theo, ta cần kéo thả 2 UILabel vào UITableViewCell ta mới kéo thả vào. Kéo tảh 2 UILabel vào và chỉnh các **Attribute** như mong muốn.

****

* Trên thanh công cụ Editor, chọn mục như hình vẽ để có thể xem 2 màn hình editor cùng 1 lúc:

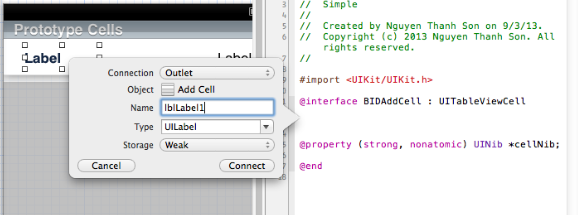




* + Nếu chúng ta chỉ kéo thả 2 UILabel như vậy thì giá trị Text của nó sẽ không thể thay đổi khi ta xử lý chúng. Vì vậy, ta cần tạo cho chúng thành 1 thuộc tính **@property** bằng cách:
  + Chọn màn hình bên trái là file giao diện **MainStoryboard.storyboard,** màn hình bên phải là file .h của lớp ta muốn nó chưa **cell** mẫu. Ở đây là **BIDAddCell.h.**
* Giữ **Ctrl+Click** vào 1 UILabel, kéo UILabel đó vào **BIDAddCell.h**

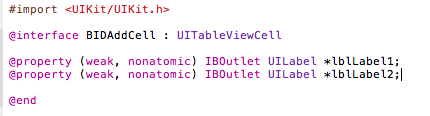


Màn hình đặt tên cho thuộc tính hiện ra:

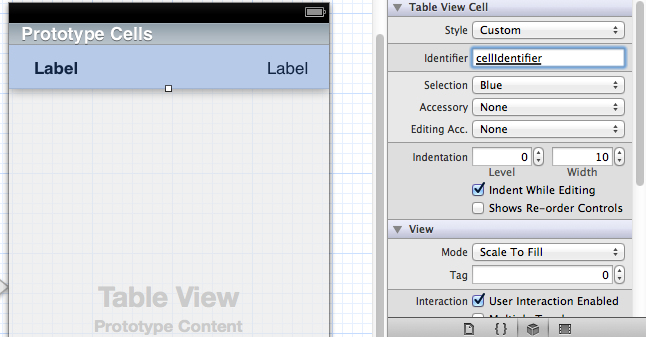


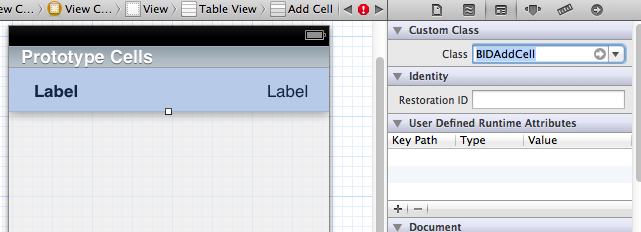
* Đặt tên thuộc tính sao cho dễ nhớ, và phân biệt được dễ dàng. Click **Connect**

Sau khi đặt thuộc tính cho cả 2 UILabel ta thu được kết quả sau:

****

* Click vào UITableViewCell, chọn **Attribute Inspector.** Trong phần **Identifier** gõ vào định danh cho **cell** mẫu mà ta muốn tạo, ở đây là “**cellIdentifier”**. Chọn **Identify,** trong phần class chọn **BIDAddCell**.

****

****

**+** Cuối cùng, code trên **BIDViewcontroller.m** để hiển thị UITableView sử dụng **cell** mẫu.

* Import class chứa **cell** mẫu
* Thực thi 3 phương thức bắt buộc để hiển thị UITableview

-(NSInteger )numberOfSectionsInTableView:(UITableView \*)tableView{

return 1;

}

-(NSInteger)tableView:(UITableView \*)tableView numberOfRowsInSection:(NSInteger)section{

return [item count];

}

-(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath{

BIDAddCell \*cell=[tableView dequeueReusableCellWithIdentifier:@"cellIdentifier"];

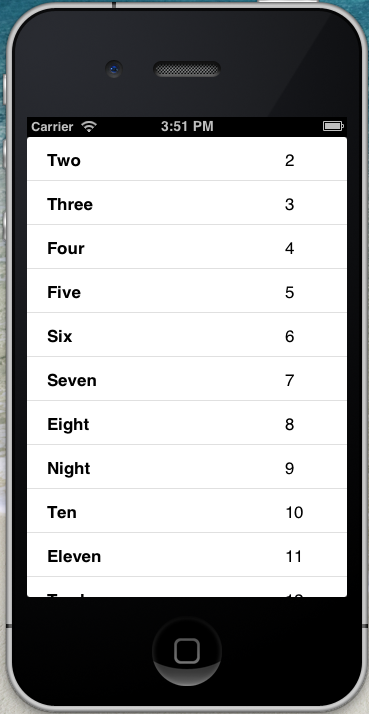
cell.lblLabel1.text =item[indexPath.row];

cell.lblLabel2.text=[NSString stringWithFormat:@"%d",indexPath.row+1];

return cell;

}

+ Kết quả thu được là như sau:



**Lưu ý:** Trong **BIDAddCell.m** có chứa phương thức khởi tạo cho **cell**

- (id)initWithStyle:(UITableViewCellStyle)style reuseIdentifier:(NSString \*)reuseIdentifier

{

self = [super initWithStyle:style reuseIdentifier:reuseIdentifier];

if (self) {

// Initialization code

}

return self;

}

Tại phương thức này, mỗi khi ta khởi tạo BIDAddCell tại ViewController, nó sẽ nhảy đến để khởi tạo 1 **cell** mẫu.

Với việc tạo một **cell** mẫu như vậy sẽ có rất nhiều lợi ích nếu mỗi **cell** chứa nhiều thành phần: Label, Image, TextField, Button,... Và các **cell** giống nhau về các thành phần mà nó chứa. Lợi ích khi rút ngắn đựơc kha khá số lượng dòng code, đơn giản chương trình, có thể sử dụng lại bất cứ lúc nào mà ta muốn.

### 3.2.5 Tương tác với các cell trong UITableView

Như ở trên, ta mới chỉ thực hiện những công iệc cực kì đơn giản trên UITableView. Vì sao tôi nói rằng UITableView là công cụ rất mạnh mẽ và đựcơ sử dụng rất nhiều trong các ứng dụng iOS, thi có lẽ các bạn vẫn chưa thấy rõ, vì trong những ví dụ đầu tiên ta chỉ thấy sự hiển thị cơ bản của UITableView mà chưa thấy được những ưu điểm trong tuỳ biến hiển thị và tương tác với các **cell.** Trong mục này, tôi sẽ trình bày sự tương tác với các **cell** một cách cơ bản nhất để bạn có cái nhìn rõ hơn về điểm mạnh của UITableview.

#### 3.2.5.1 Hiển thị khi Click cell

Ta có các tuỳ chọn khi Click vào một **cell** trong UITableView:

**+ Hiển thị màu của cell khi Click vào cell**

-(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath{

BIDAddCell \*cell=[tableView dequeueReusableCellWithIdentifier:@"cellIdentifier"];

**cell.selectionStyle=UITableViewCellSelectionStyleNone;**

cell.lblLabel1.text =item[indexPath.row];

cell.lblLabel2.text=[NSString stringWithFormat:@"%d",indexPath.row+1];

return cell;

}

* **UITableViewCellSelectionStyleNone:** Không hiện màu sắc khi Click vào **cell**
* **UITableViewCellSelectionStyleGray:** Hiển thị màu xám khi Click vào **cell**
* **UITableViewCellSelectionStyleBlue:** Hiển thị màu xanh da trời khi Click vào **cell** (Đây là mặc định trong UITableView)

****

****

****

**+ ContentInset, ContentOffset**

Như ví dụ bạn đã chạy ở trên, bạn đang thắc mắc tại sao UITableView của mình không thể kéo để xem toàn bộ bảng, việc xem cell đầu tiên và cuối cùng đang không như bạn mong đợi. Thì với công cụ **SetContentInset, ContentOffset** bạn sẽ giải quyết được những thắc mắc ấy.

* **ContentInset:** Thiết lập khoảng không gian thừa để UITableView của bạn có thể sử dụng

-(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath{

**tableView.contentInset=UIEdgeInsetsMake(100, 0, 100, 0);**

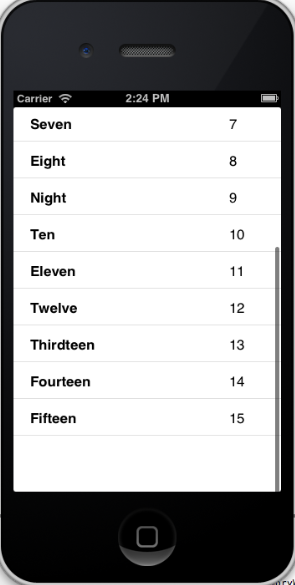
BIDAddCell \*cell=[tableView dequeueReusableCellWithIdentifier:@"cellIdentifier"];

cell.lblLabel1.text =item[indexPath.row];

cell.lblLabel2.text=[NSString stringWithFormat:@"%d",indexPath.row+1];

return cell;

}



Ở đây, tạo một ContentInset với cú pháp **UIEdgeInsetsMake(<#CGFloat top#>, <#CGFloat left#>, <#CGFloat bottom#>, <#CGFloat right#>)**

**Top:** tạo khoảng trống phía trên

**Left:** tạo khoảng trống bên trái

**bottom:** tạo khoảng trống phía dưới

**right:** tạo khoảng trống bên phải

* **ContentOffset:** Vị trí tương đối của một phần tử trong một **View** nào đó. Ở đây, ta thường tạo **contentInset** cho một UITableView để tạo them không gian thừa để UITableview có thể kéo và xem hết các phàn tử, **contentOffset** được thiết lập để xác định xem **View Center** đang nằm ở đâu.

#### 3.2.5.2 Bắt sự kiên khi Click vào cell

Khi Click vào **cell** nếu ta không muốn xử lý them gì ở thao tác Click này thì ta chỉ việc không bắt sự kiện Click **cell.** Ngược lai, nếu muốn có những xử lý nhất định với việc Click **cell** ta có thể thực hiện 2 phương thức sau:

//Xử lý sự kiện Click vào **cell** trong UITableView

-(void)tableView:(UITableView \*)tableView didSelectRowAtIndexPath:(NSIndexPath \*)indexPath{

BIDAddCell \*cell=(BIDAddCell \*)[tableView cellForRowAtIndexPath:indexPath];

cell.selected=NO;

}

Ở đây, **BIDAddCell \*cell=(BIDAddCell \*)[tableView cellForRowAtIndexPath:indexPath]** sẽ giúp xác định **cell** mà ta đã Click vào.

//Xử lý sự kiện sau khi Click **cell** xong

-(void)tableView:(UITableView \*)tableView didDeselectRowAtIndexPath:(NSIndexPath \*)indexPath{

}

#### 3.2.5.3 Edit UITableView

Phần này tôi sẽ đưa ra ví dụ về trường hợp **thêm, sửa, xoá, di chuyển** các **cell** trong UITableView.

Giả sử, khi ta Click vào một **cell** sẽ bắt sự kiện **Editting** UITableView, và ở chế độ **Editting** ta sẽ tiến hành các thao tác **thêm, sửa, xoá** như mong muốn. Tất cả các dòng đều có thể **di chuyển.**

**+ Vào chế độ Editting**

-(void)tableView:(UITableView \*)tableView didSelectRowAtIndexPath:(NSIndexPath \*)indexPath{

BIDAddCell \*cell=(BIDAddCell \*)[tableView cellForRowAtIndexPath:indexPath];

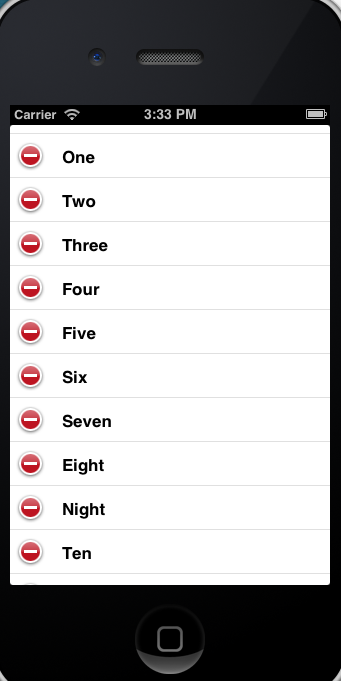
cell.selected=NO;

**[tableView setEditing:YES animated:YES];**

}

Ở đây,**[tableView setEditing:YES animated:YES]** sẽ giúp bạn đến với chế độ **Editting.**

Và giao diện ta nhận được sẽ như sau:



**+ Thiết lập trạng thái trong chế độ Editting**

Giả sử, ta muốn dòng cuối cùng sẽ ở trạng thái thêm **cell,** còn lại các dòng khác sẽ ở chế độ xoá **cell,** thì ta có thể them dòng code sau:

-(UITableViewCellEditingStyle)tableView:(UITableView \*)tableView editingStyleForRowAtIndexPath:(NSIndexPath \*)indexPath{

if(indexPath.row== [item count]-1){

return UITableViewCellEditingStyleInsert;

}

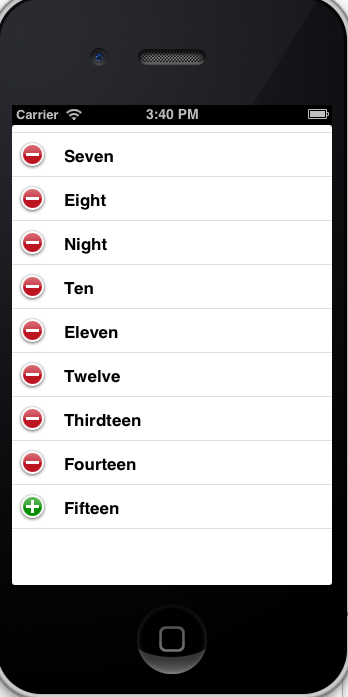
else{

return UITableViewCellEditingStyleDelete;

}

}

* **UITableViewCellEditingStyleInsert:** trả về **thêm cell**
* **UITableViewCellEditingStyleDelete**: trả về **xoá cell**

****

**+ Cho phép di chuyển cell**

Để di chuyển các item trong UITableview ta cần làm việc với mảng dữ liệu. Mảng dữ liệu ban đầu của ta là **NSArray.** Trong khi đó, việc di chuyển các **cell** trong UITableView chỉ làm việc với mảng dữ liệu có kiểu **NSMutableArray.** Vì vậy, công việc của chúng ta là cần tạo một mảng **NSMutableArray** có chứa các item như trong mảng **NSArray** như khai báo ban đầu và sau đó thực hiện phương thức **di chuyển** các item. Công việc cần làm như sau:

* Khai báo một mảng **itemMu** bên file **BIDViewController.h**

@interface BIDViewController : UIViewController<UITableViewDataSource,UITableViewDelegate>

@property (nonatomic, copy) NSArray \*item;

@property (nonatomic, copy) NSMutableArray \*itemMu;

@end

* Khai báo trong **BIDViewController.m** để có thể truy xuất trực tiếp

@synthesize item,itemMu;

* Tạo dữ liệu trong mảng mới như dữ liệu trong mảng khai báo ban đầu

- (void)viewDidLoad

{

[super viewDidLoad];

item=@[@"One", @"Two", @"Three", @"Four", @"Five",

@"Six", @"Seven", @"Eight", @"Night", @"Ten",

@"Eleven", @"Twelve", @"Thirdteen",@"Fourteen",@"Fifteen"];

itemMu=[[NSMutableArray alloc]initWithArray:item];

}

* Thực hiện hàm **di chuyển** item

-(void)tableView:(UITableView \*)tableView moveRowAtIndexPath:(NSIndexPath \*)sourceIndexPath toIndexPath:(NSIndexPath \*)destinationIndexPath{

id object=[itemMu objectAtIndex:sourceIndexPath.row];

[itemMu removeObjectAtIndex:sourceIndexPath.row];

[itemMu insertObject:object atIndex:destinationIndexPath.row];

}

Kết quả thu được như sau:

**+ Thực hiện thêm, xoá các cell**

Tương tự như việc **di chuyển,** công việc **them, xoá** item cũng chỉ làm việc với mảng dữ liệu kiểu **NSMutableArray.** Và việc **them, xoá** item sẽ ảnh hưởng đến số lượng **cell** nên việc khai báo số lượng **cell** sẽ cần được xem xét lại như sau:

-(NSInteger)tableView:(UITableView \*)tableView numberOfRowsInSection:(NSInteger)section{

return [itemMu count];

}

Tiếp đó, việc **thêm** và **xoá** các **cell** sẽ được thực hiện như sau:

-(void)tableView:(UITableView \*)tableView commitEditingStyle:(UITableViewCellEditingStyle)editingStyle forRowAtIndexPath:(NSIndexPath \*)indexPath{

//delete

if(editingStyle ==UITableViewCellEditingStyleDelete){

[itemMu removeObjectAtIndex:indexPath.row];

[tableView deleteRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:UITableViewRowAnimationAutomatic];

}

//insert

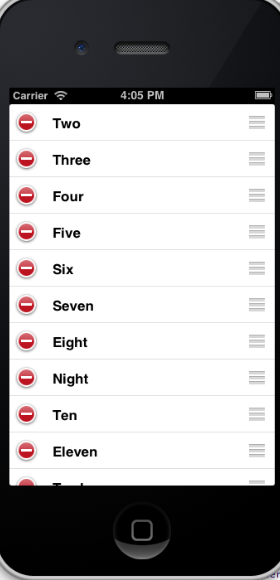
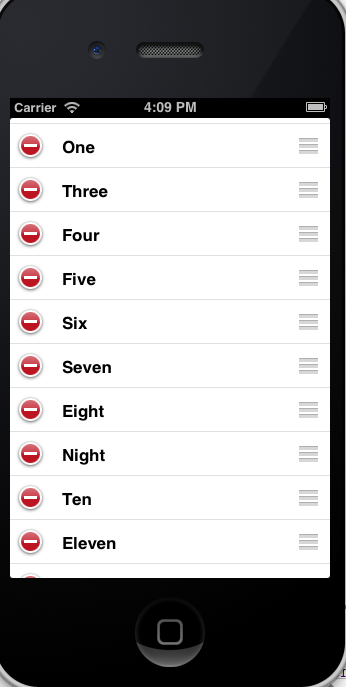
if(editingStyle==UITableViewCellEditingStyleInsert){

[itemMu insertObject:@"Sixteen" atIndex:indexPath.row];

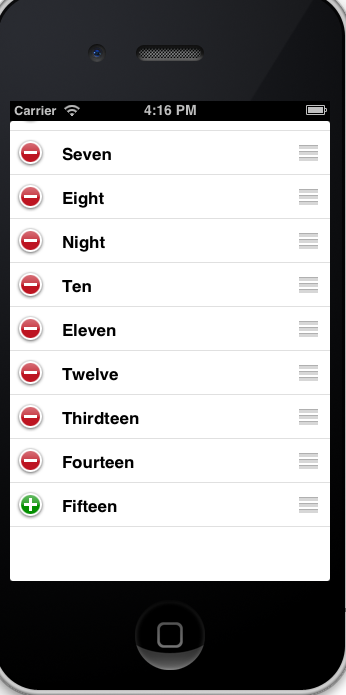
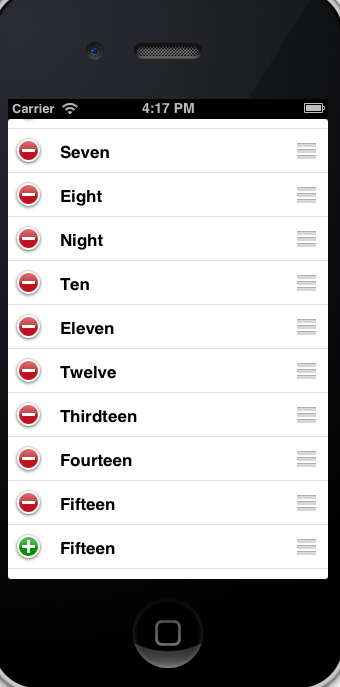
[tableView insertRowsAtIndexPaths:[NSArray arrayWithObject:indexPath] withRowAnimation:UITableViewRowAnimationBottom];

}}

**Xoá**

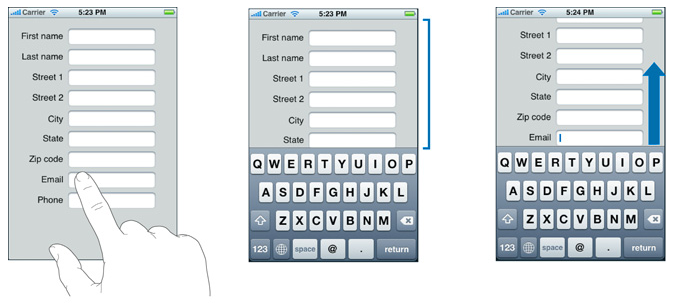
**thêm**

** **

Như vậy, đó là những kiến thức cơ bản nhất mà tôi muốn chia sẻ với các bạn về UITableView. Đối với từng bài toán cụ thể, bạn hãy vận dụng sự linh hoạt của UITableView để có thể đáp ứng được những yêu cầu mà ứng dụng đặt ra.

## 3.3 ContentInset, Content Offset và cell subclass

Trong phần trước, các bạn đã được giới thiệu qua về ContentInset ContentOffset, và đã được làm ví dụ về tạo một cellsubclass. Trong phần này, chúng ta sẽ cùng nhau làm một ví dụ về ba nội dung này.



Hình trên, ta thấy rằng nó là một UITableView có 9 cell, mỗi cell trong UITableView bao gồm: 1 UILabel, 1 UITextField.

Đó là về hình thức, đánh giá về hình thức của UITableView ta có thể thấy như sau:

* Các cell có cấu trúc như nhau nên ta có thể tạo cellsubclass. Điều này giúp việc tạo một list các cell một cách nhanh chóng, tiện lợi.
* Việc Click vào những UITextField cuối sẽ dẫn đến một điểm đáng lưu ý là bàn phím sẽ che lấp đi UITextField ta muốn điền thông tin. Và lúc này, ta sẽ cần ContentInset, Content Offset để có thể đẩy UITextField lên phía trên của bàn phím. Bạn cũng có thể sử dụng ContentInset, Content Offset rất linh hoạt tuỳ theo yêu cầu của ứng dụng.

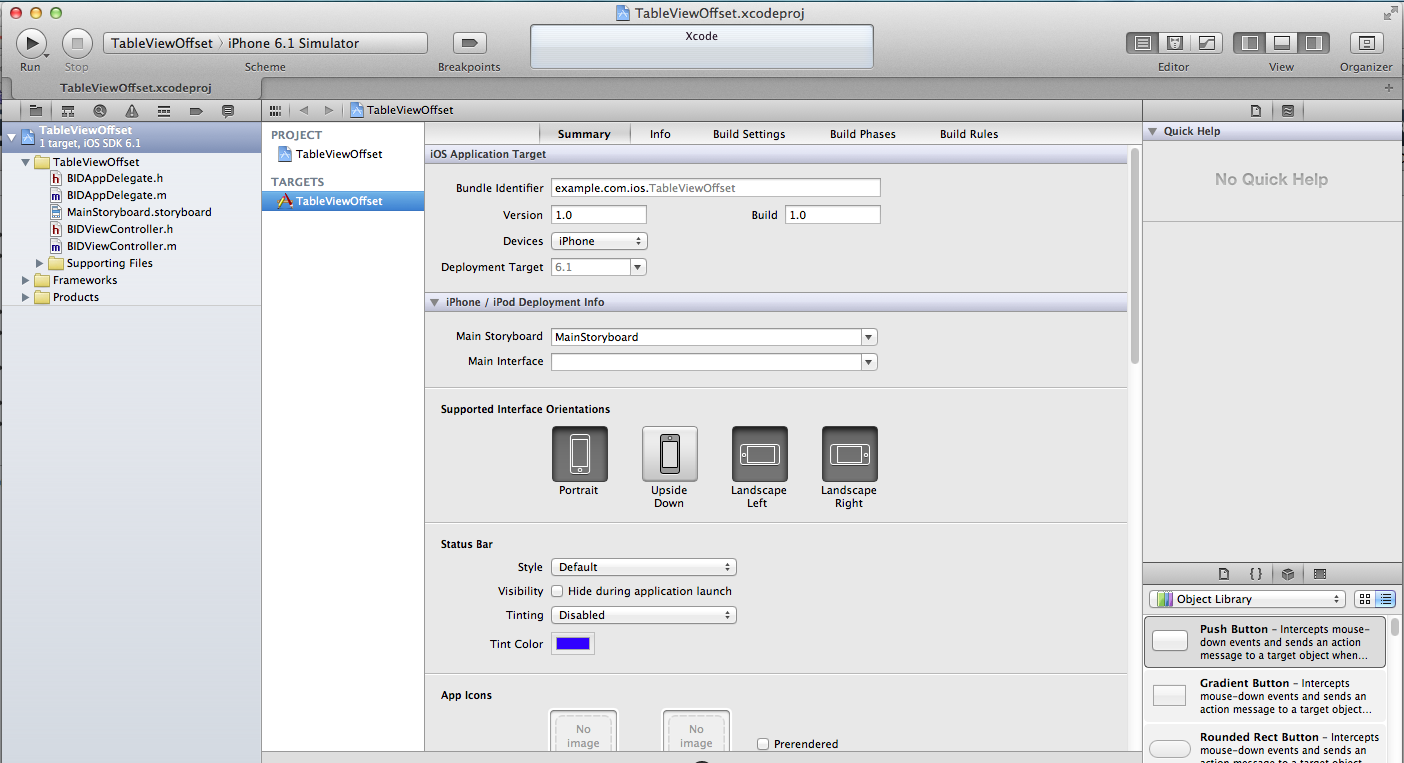
Công việc của chúng ta trong ví dụ này bao gồm các bước như sau:

### Bước 1: Tạo một Project mới

Tương tự như tạo project đã hướng dẫn ở trên. **Project** có tên là **TableViewOffset** (tất nhiên là tên Project, classs, … là tuỳ ý của bạn mà thôi).

Cần chú ý thêm là ở đây ta sẽ sử dụng Storyboard.

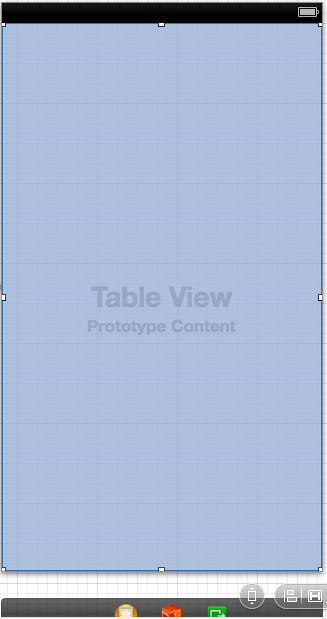
**Project** của chúng ta sẽ như sau:



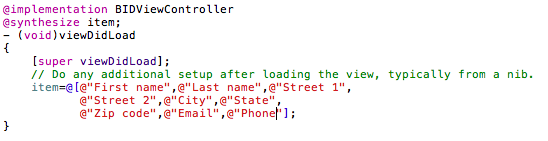
### Bước 2: Hiển thị UITableView, sử dụng subclass.

Bạn hãy xem lại mục [**3.2**](#_3.2_Hiển_thị)để hoàn thiện bước yêu cầu này.

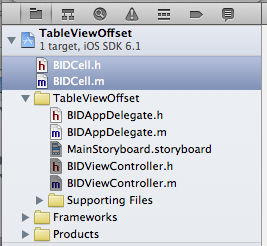
* Add UITableView

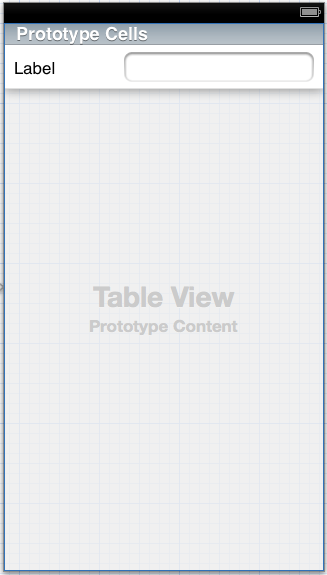


* Add mảng dữ liệu của UITableView



* Tạo một cellsubclass



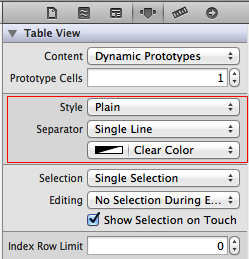


* Hiển thị UITableView
  + Khai báo dataSource, delegate
  + Thực thi các phương thức

Sau khi thực hiện đầy đủ các bước trên, ta sẽ thu được kết quả như sau:



Có thể thấy rằng, mặc định thì giữa các **cell** sẽ được phân cách với nhau bởi một đường sọc ngang, để remove sọc ngang này, bạn chọn File giao diện chính **MainStoryboard.storyboard.** Sau đó, Click chuột trái chọn UITableView, ta sẽ tiến hành chỉnh đường sọc trên trong **Attributes inspector.** Ta sẽ chỉnh thông số sau:



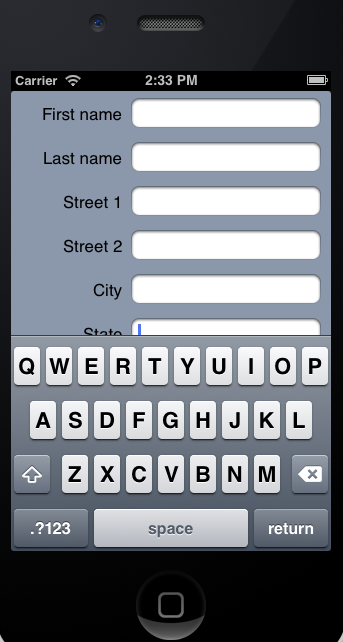
Và như vậy, ta sẽ thu được một UITableView như yêu cầu của ứng dụng:



Lưu ý, với những ứng dụng như thế này, bạn cần tắt chức năng chọn một **cell** trong UITableView.

### Bước 3: Add TapGestureRecognizer cho UITableView

Tại sao cần add **TapGestureRecognizer?** Khi bạn click vào những UITextField đầu tiên, nếu muốn Edit Text các UITextField thì chúng đã bị Bàn phím che mất không gian làm việc.



Vai trò của **TapGestureRecognizer,** ta có thể add các phương thức khi ta Click vào các khu vực khác ngoài UITextField. Ở đây, ta sẽ add phương thức kết thúc Edit Text khi **TapGestureRecognizer.** Khi đó, chỉ việc Click vào các khu vực khác ngoài UITextField thì bàn phím sẽ ẩn đi.

Thực hiện add **TapGestureRecognizer** sau:

* Trong **BIDViewController.h:** thực hiện khai báo phương thức ẩn bàn phím:

@interface BIDViewController : UIViewController<UITableViewDataSource,UITableViewDelegate>

@property (nonatomic, copy) NSArray \*item;

-(void)hideKeybroad;

@end

* Trong **BIDViewController.m:** thực hiện add **TapGestureRecognizer**
  + Thực hiện gán tag cho UITableView mà ta làm việc, bằng việc chỉnh thông số tag trong **Attributes inspector** của UITableView



* + Trong viewDidload(), thực hiện add **TapGestureRecognizer**

UITableView \*table=(id)[self.view viewWithTag:1];

UITapGestureRecognizer \*keybroad=[[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(hideKeybroad)];

[table addGestureRecognizer:(keybroad)];

* + Thực hiện phương thức hideKeybroad:

-(void)hideKeybroad{

[self.view endEditing:YES];

}

Click **Run** chương trình để có thể thấy được tác dụng của **TapGestureRecognizer**

### Bước 4: Add ContentInset cho UITalbeView

Dù đã add **TapGestureRecognizer** cho UITalbeView, tuy nhiên nếu để mặc định thì các cell cuối của UITableView luôn bị che lấp bởi bàn phím. Vì mặc định vị trí của UITableView luôn chỉ nằm gọn trong một màn hình.

Với **ContentInset** sẽ giúp bạn khắc phục được điểm yếu đó, với **ContentInset** bạn có thể tạo them không gian trái, phải, trên, dưới cho UITableView, để từ đó tạo lên 1 Scroll và có thể di chuyển toàn bộ UITableView lên trên màn hình ta có thể quan sát.

Tạo **ContentInset** có dạng như sau:

**UIEdgeInsetsMake(<#CGFloat top#>, <#CGFloat left#>, <#CGFloat bottom#>, <#CGFloat right#>)**

* **Top:** Kích thước phía trên UITableView được thêm
* **left:** Kích thước phía bên trái UITableView được thêm
* **Bottom:** Kích thước phía dưới UITableView được thêm
* **right:** Kích thước phía bên phải UITableView được thêm

Với ví dụ trên, ta có thêm đoạn code như sau:

-(UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath{

BIDCell \*cell=[tableView dequeueReusableCellWithIdentifier:@"Identifier"];

tableView.contentInset=UIEdgeInsetsMake(0, 0, 250, 0);

cell.selectionStyle=UITableViewCellSelectionStyleNone;

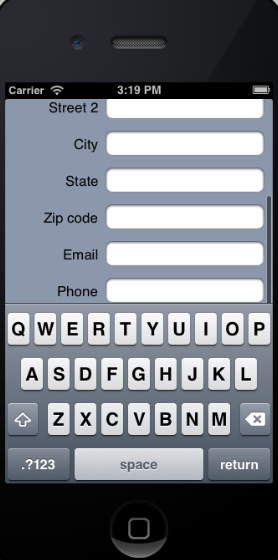
cell.lblLabel.text =item[indexPath.row];

[cell setSelected:NO];

return cell;

}

Ở đây, ta sẽ thêm vào phía dưới UITableView một kích thước là 250. Kết quả thu được:



Với **ContentInset** ta đã có thể xem xét cũng như tương tác với tất cả các cell, kể các các cell bị che lấp bởi bàn phím trước đó.

Tuy nhiên, việc muốn Edit Text các cell bị che lấp bơi bàn phím, ta cần Scroll UITableView thực tế là không tiện lợi, chính vì vậy, trong trường hợp này kết hợp **ContentInset và ContentOffset** sẽ là vô cùng tiện lợi

**Bước 5: Add ContentOffset vào UITableView**

Tác dụng của **ContentOffset:** là vị trí tương đối của một object trong một **ContentInset.** Trong ví dụ của chúng ta, ta sẽ thực hiện **setContentOffset** cho một cell khi cell bị che lấp bởi bàn phím khi ta click vào UITextField của cell đó. Kịch bản của chúng ta là: Khi click vào một UITextField, ta sẽ xem xét xem cell chứa UITextField đó có bị bàn phím che lấp hay không:

* Không bị che lấp: Giữ nguyên offset của UITableview đó
* Bị che lấp: Tính toán, rồi tiến hành setOffset phù hợp cho cell đó, để cell ta đang Edit nằm phía trên bàn phím một khoảng vừa đủ.

Sau đây, tôi sẽ cùng các bạn tạo một lớp chuyên xử lý **offset** của một **UITextField** trong một **UITableView**. Với việc tạo một lớp riêng này, bạn có thể sử dụng lại bất cứ lúc nào. Nó cũng tương tự như **cell subclass** ta tạo trước đó, hoàn toàn có thể sử dụng lại nó cho các **UITableView** khác.