UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

Guía de Laboratorio N°02 b

(para desarrollarse en casa, se recomienda para los grupos que harán la segunda versión del proyecto de aplicación en IOS)

"Introducción a la Interfaz de usuario de IOS"

Objetivos:

Que el estudiante:

- Conozca la declaración e implementación de los controles básicos en la programación en IOS haciendo uso de **ViewControllers** para realizar el diseño de la interfaz y el uso de **archivos de declaración (.h) e implementación (.m)** para definir el manejo de estos.
- Se le facilite el Incorporar nuevos controles, siguiendo la lógica de controles de similar comportamiento.

Descripción:

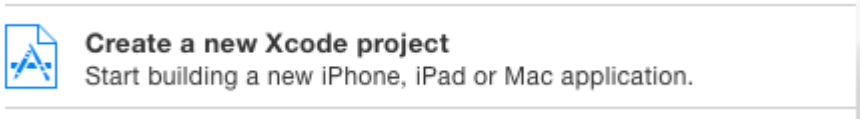
Esta práctica consistirá en crear un programa que contendrá una ViewControllers que mostrarán el uso de cada uno de los controles básicos de IOS (Button, Label, TextField, Switch, etc) con su respectiva interfaz de usuario (ViewController). Cada ViewController será llamado por medio de otro ViewController que llamaremos Main.storyboard, este lo usaremos para mostrar un "Menú" por medio de Botones (Buttons).

UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERIA Y ARQUITECTURA
ESCUELA DE INGENIERIA DE SISTEMAS INFORMATICOS
PROGRAMACION PARA DISPOSITIVOS MOVILES
PDM115

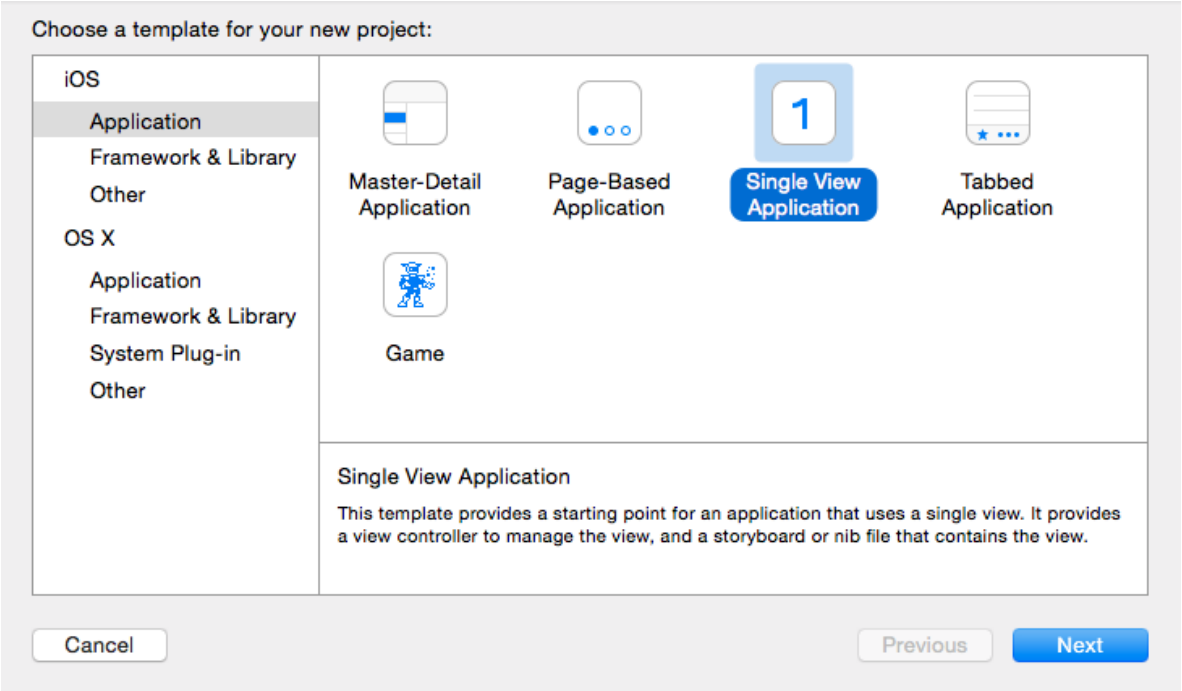
Índice

Creación de proyecto

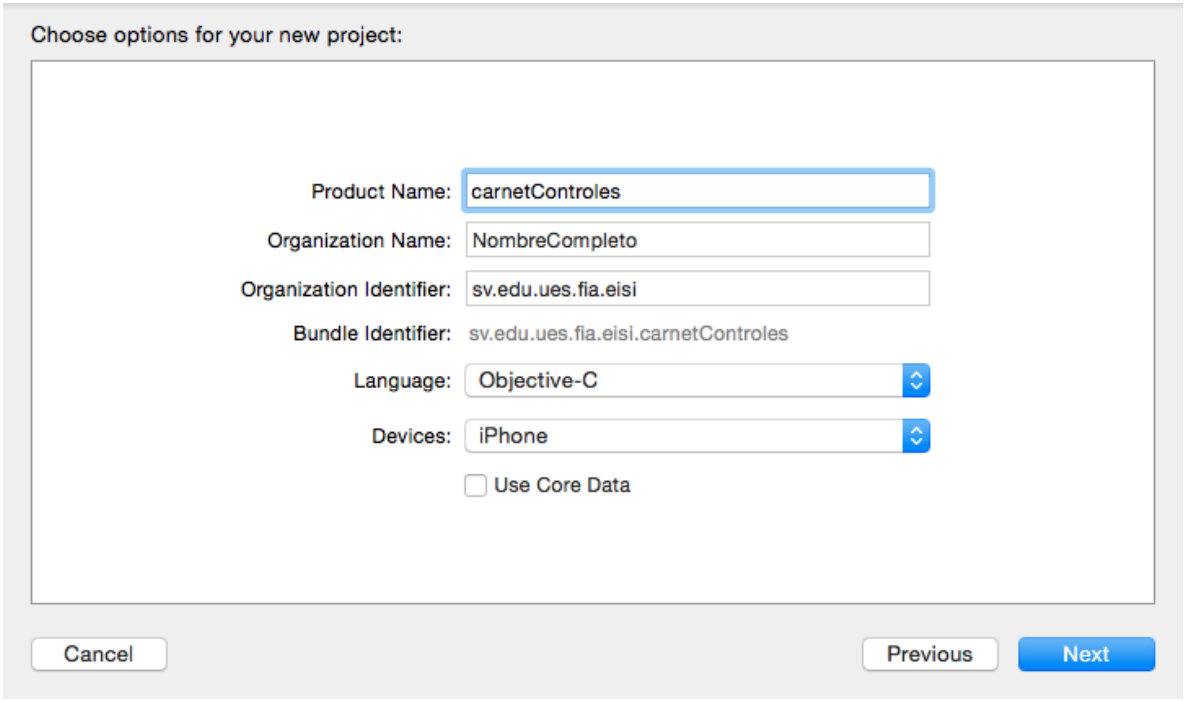
Al iniciar el programa Xcode, seleccione la opción para crear un nuevo proyecto.



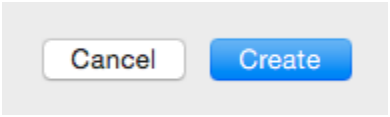
Al presentarse las opciones de proyectos a crear seleccionar la opción de "Single View Application", y presionar el botón "Next", como en la imagen que se muestra a continuación.



En la creación del proyecto poner como nombre del proyecto el número de carnet correspondiente seguido de la palabra Controles, y rellenar los campos siguientes como se muestra en la imagen a continuación.

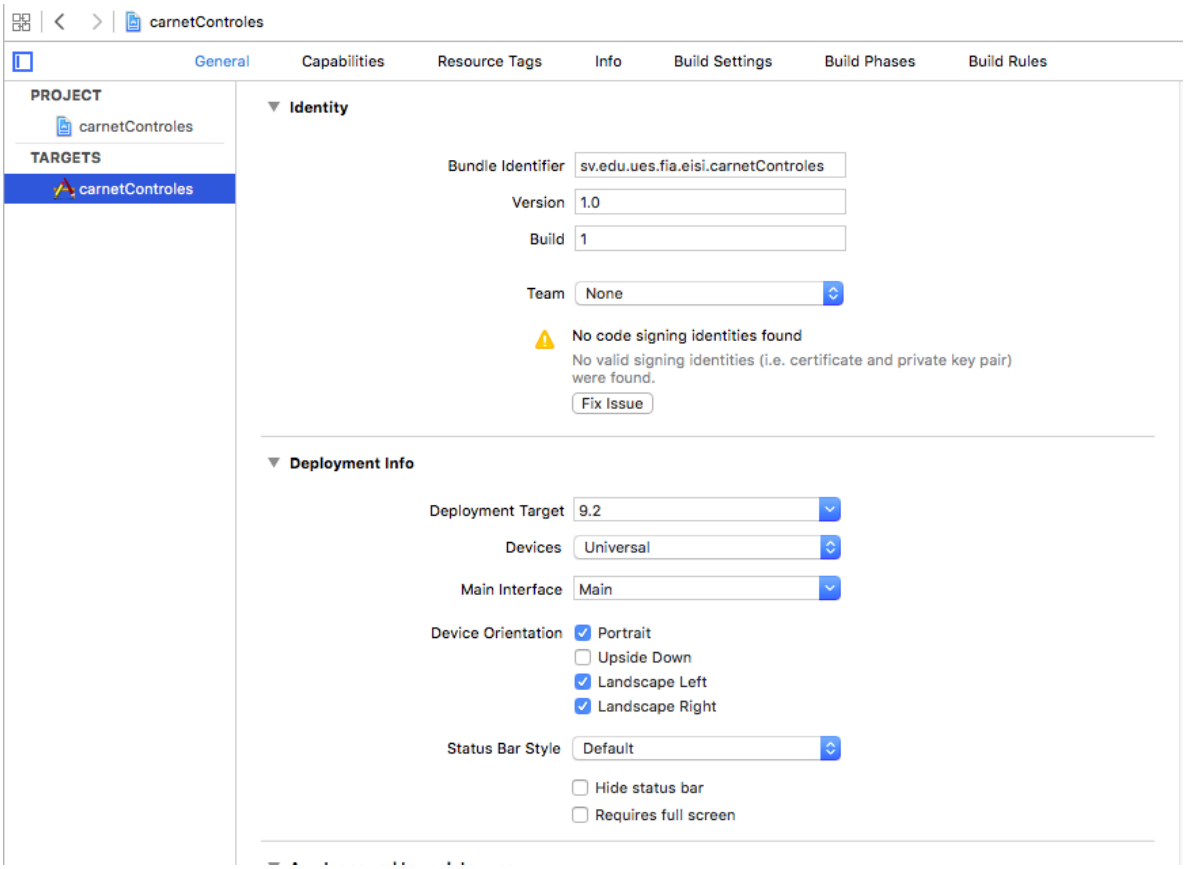


Al presionar “Next” se presentara otra pantalla en la cual se debe seleccionar la opción de “Create”.



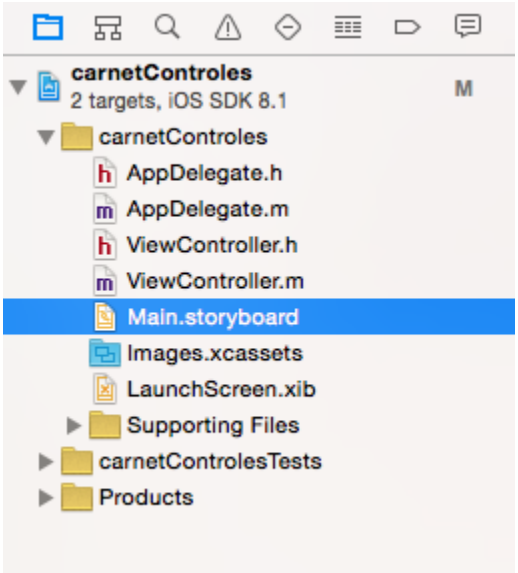
En la ventana “general” del proyecto, seleccione el Deployment Target a la versión 9.2

Y en la parte superior izquierda seleccione el dispositivo iphone , ios 9.2



Al iniciar el proyecto, se contienen la cantidad de archivos como los que se presentan a continuación, los cuales se irán trabajando mediante se va desarrollando la guía.

Primero acceder al archivo Main.storyboard, como se presenta a continuación.



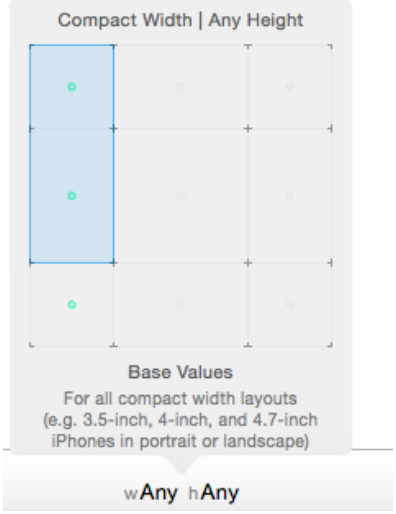
Al entrar al archivo Main.storyboard estará accediendo al ambiente grafico de la aplicación, si se trabajara una aplicación con uso Universal (lo cual implica que la misma aplicación funciona tanto para ipad o iphone) se presentarían 2 archivos de tipo storyboard, cada uno correspondiente a cada dispositivo.

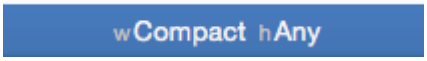
Main.storyboard (Menu)

Primeramente se presenta una sola pantalla en el Main.storyboard, la cual tiene como una flecha a la izquierda indicando que la aplicación inicia desde esa pantalla, como se muestra a continuación.

En este caso se probará con el simulador del Iphone 6 y hay que ajustar el tamaño del view Controller para ello, buscar en la parte inferior de la pantalla la siguiente opción:

Al dar clic aparecerá una pantalla para modificar el tamaño que debe ajustarse de la siguiente manera:

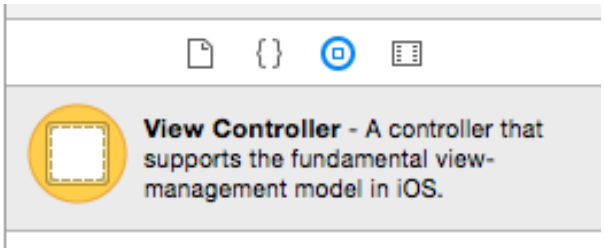


Ajustar de manera que quede con las opciones: 

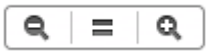
Al finalizar podrá apreciarse el View Controller de otro tamaño:

Creación de las Vistas para las opciones de menu

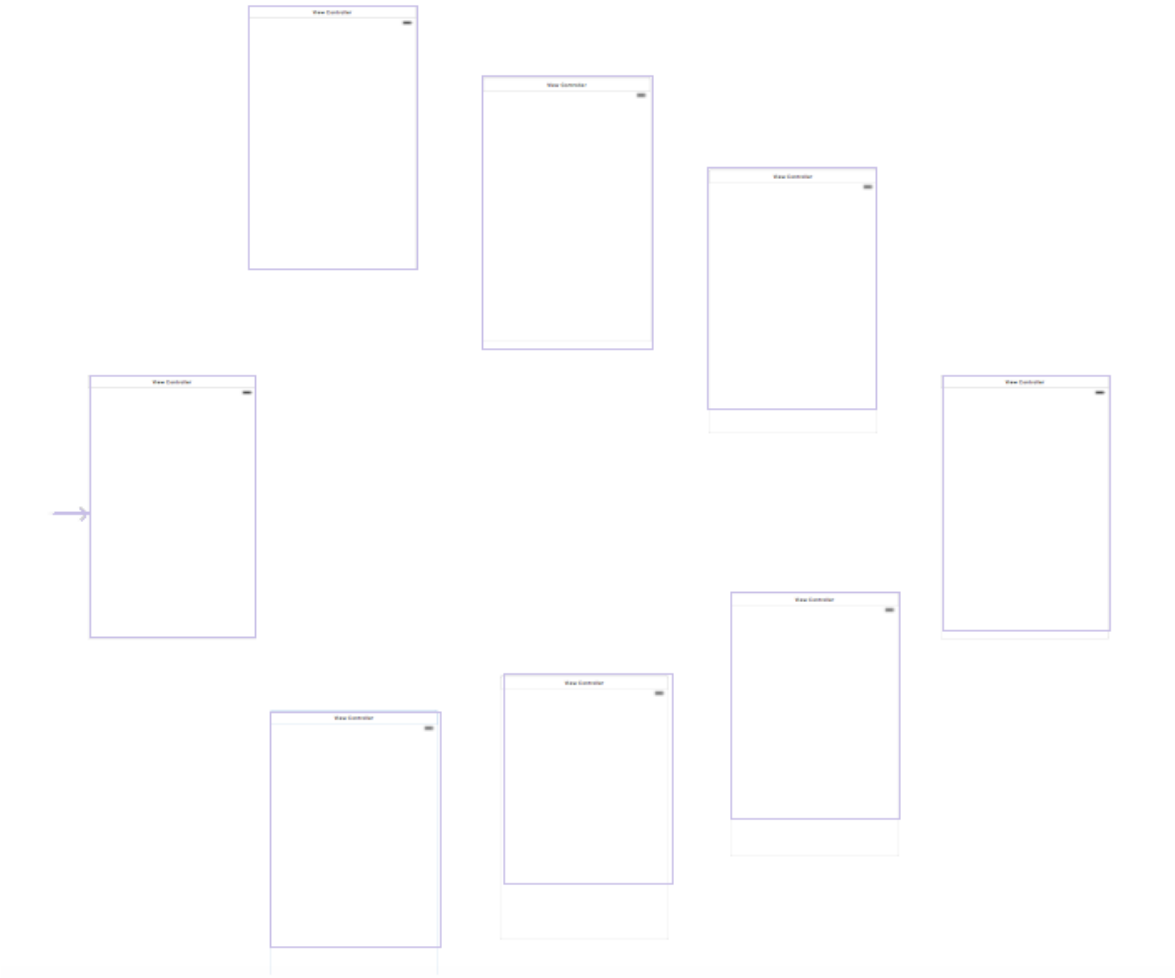
Cada pantalla presentada en el Main.storyboard es un objeto de tipo ViewController, los cuales se pueden agregar desde la barra de utilidades ubicada a la derecha del área de trabajo, arrastrando el objeto que se muestra a continuación hasta el área de trabajo del storyboard, generando así varias pantallas en el mismo archivo Main.storyboard.



Nota: en la parte inferior izquierda de la ventana de edición del Main.storyboard, esta una barra de configuración de tamaños para su ventana, presione 2 veces el icono de zoom out para ver los objetos de manera reducida..



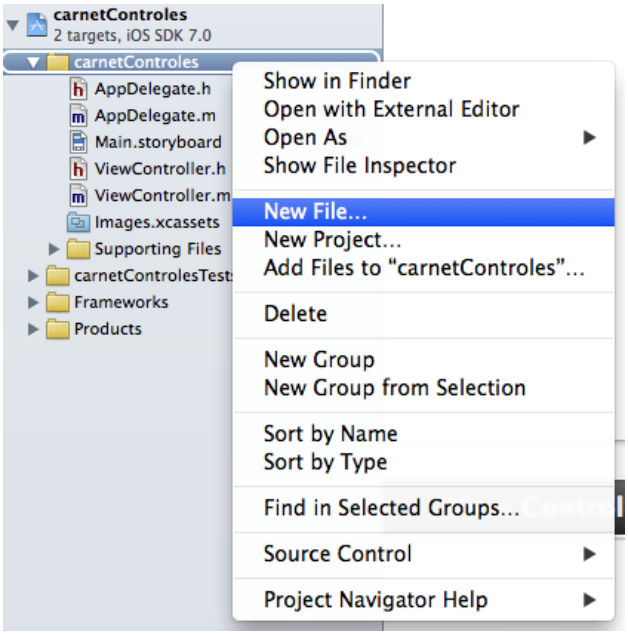
Agregue las pantallas necesarias (7 View Controller más) para que el Main.storyboard se observe como el que se muestra a continuación. Se observa que las pantallas agregadas no poseen la flecha al lado izquierdo como la principal.



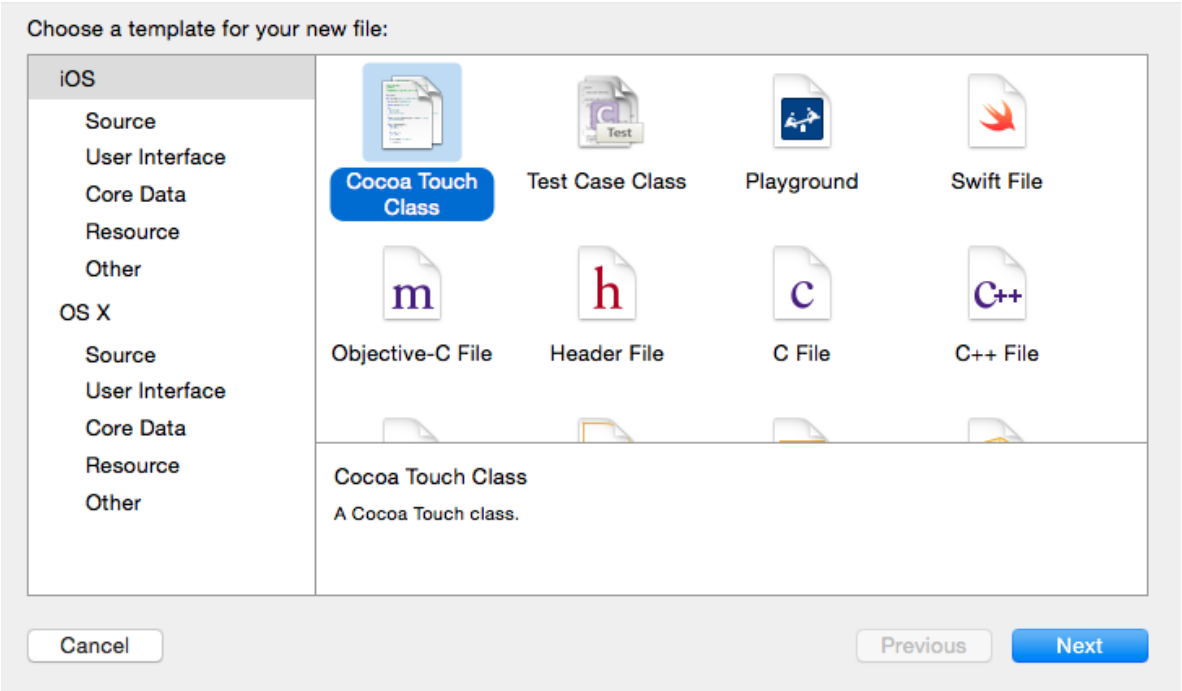
Para poder trabajar cada pantalla (vista), estas deben de tener archivos .m y .h correspondientes a cada una de ellas, para eso se debe de crear archivos de este tipo.

Creación de Aplicaciones

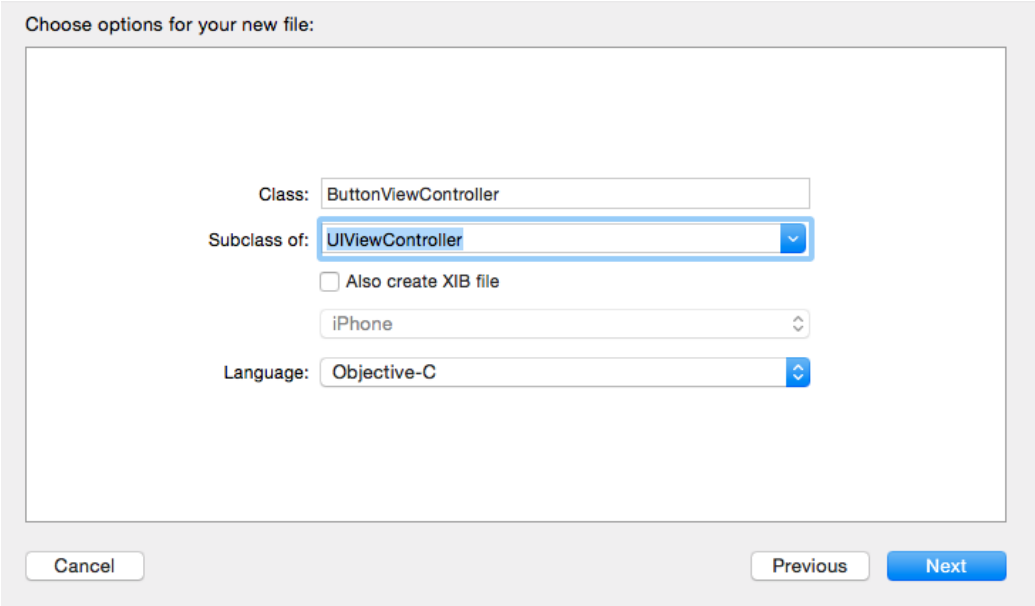
Para esto seleccionar la carpeta del proyecto (amarilla), hacer clic derecho sobre ella y seleccionar la opción de “New File…”, como se presenta a continuación.



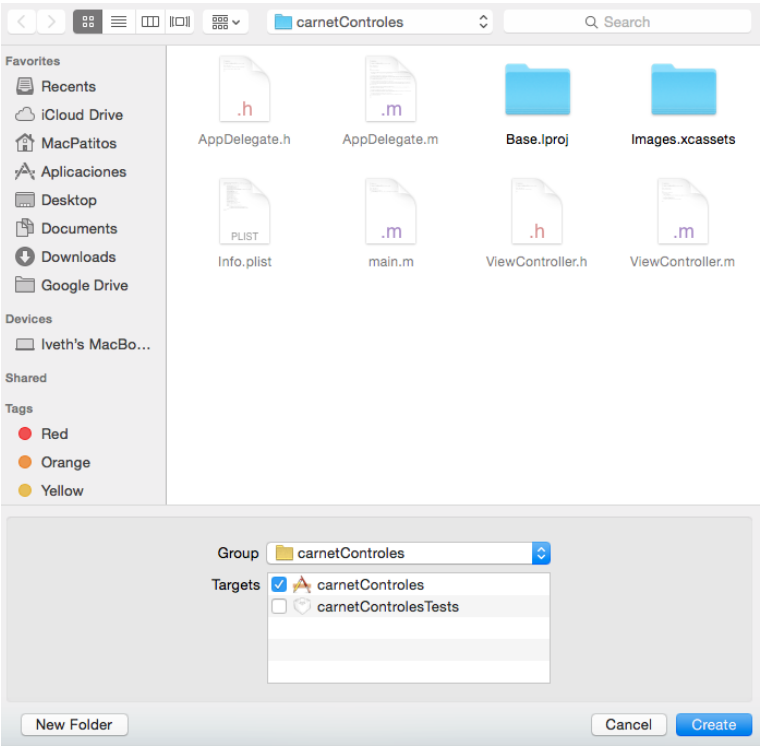
El tipo de archivo que se desea crear es “Cocoa Touch Class”, seleccionar esa opción cómo se presenta en la imagen.



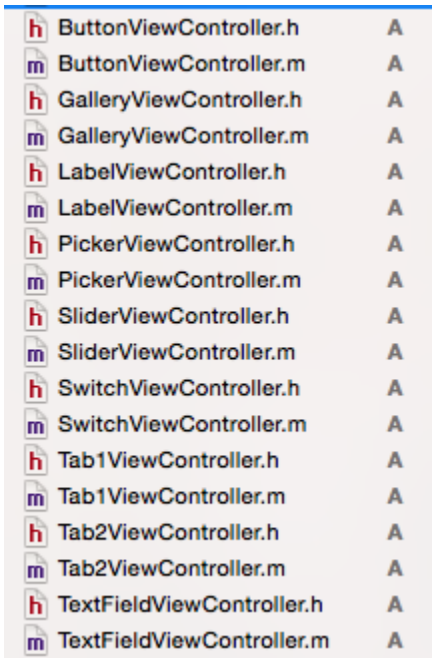
Al presionar “Next” se presentará la ventana en la cual se ingresara el tipo la subclase de la clase a crear y el nombre que se definirá para la clase, seleccionar la subclase “UIViewController” y colocar el nombre de la clase como “ButtonViewController”.



Presionamos next y luego Create.

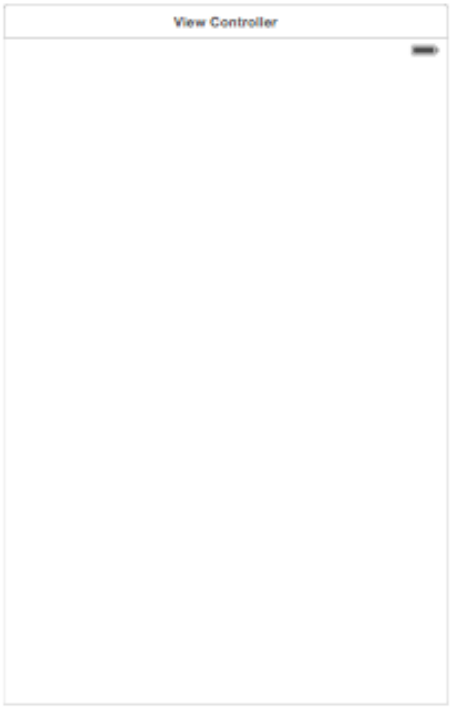


Repetimos lo anterior para las otras seis vistas y adicionalmente crearemos dos para controlar el tabView hasta tener la siguiente lista de archivos:

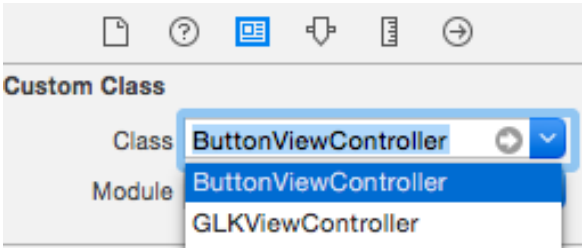


Vincular Vista con controlador (clases)

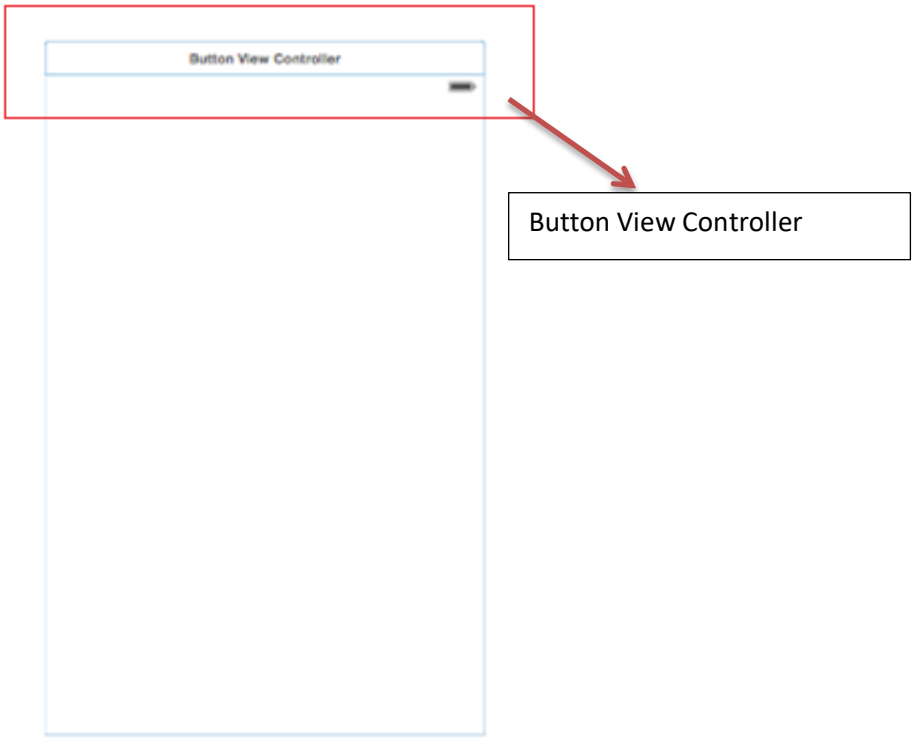
Para conectar cada “ViewController” con sus respectivos archivos .h y .m de los archivos de clase creados anteriormente, seleccionar en el proyecto el main.storyboard y dentro de este, presionar clic en el primer ViewController de los que se agregaron, que se vinculará con la clase “ButtonViewController” antes creada.



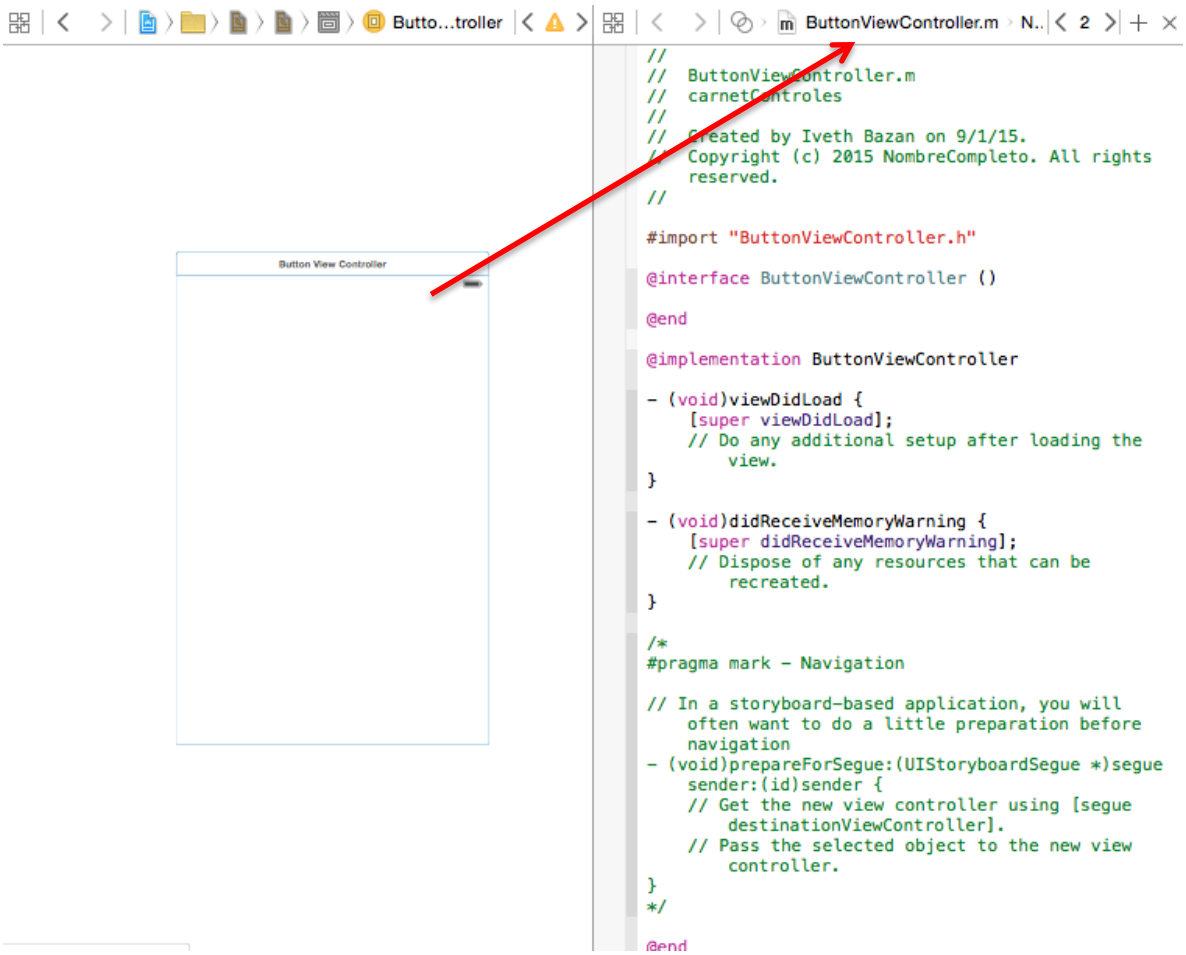
Acceder a la barra de utilidades en la parte superior derecha  y ubicarse en la pestaña de "Identity inspector" (inspector de identidad) que tiene el siguiente ícono  . Dentro de esta pestaña ubicarse en el apartado de "Custom Class", y en el espacio de clase, escribir(o seleccionar en el combo) la clase que se acaba de crear "ButtonViewController", como se observa a continuación.



Ahora al acceder al storyboard se observa que la pantalla seleccionada posee el nombre de la clase creada para ella. (Si no lo logra ver presione zoom in una o dos veces)



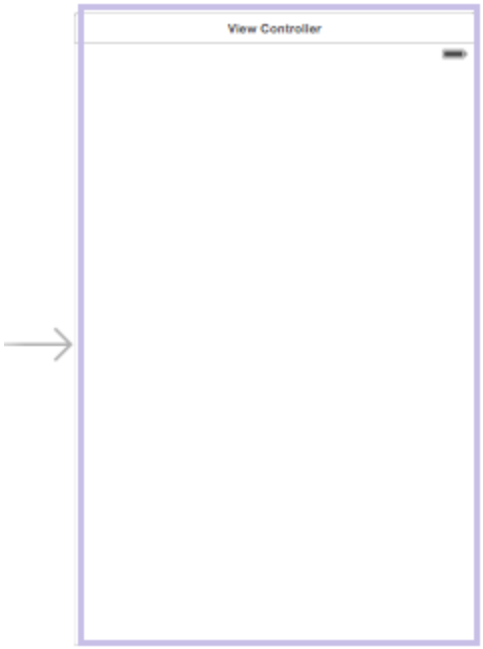
Para ver este enlace de manera dinámica, habilitar la vista de asistente, presionando el icono de "Assistent editor" con este icono:  este habilitara dos áreas de trabajo. Al seleccionar la pantalla que ya tiene archivos .m y .h vinculados, se muestra alguno de estos 2 archivos en la otra área de trabajo.



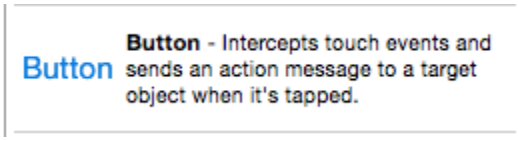
Repita lo anterior hasta vincular todas las vistas con sus respectivas clases(Label…. Hasta Gallery).

Configuración del menú

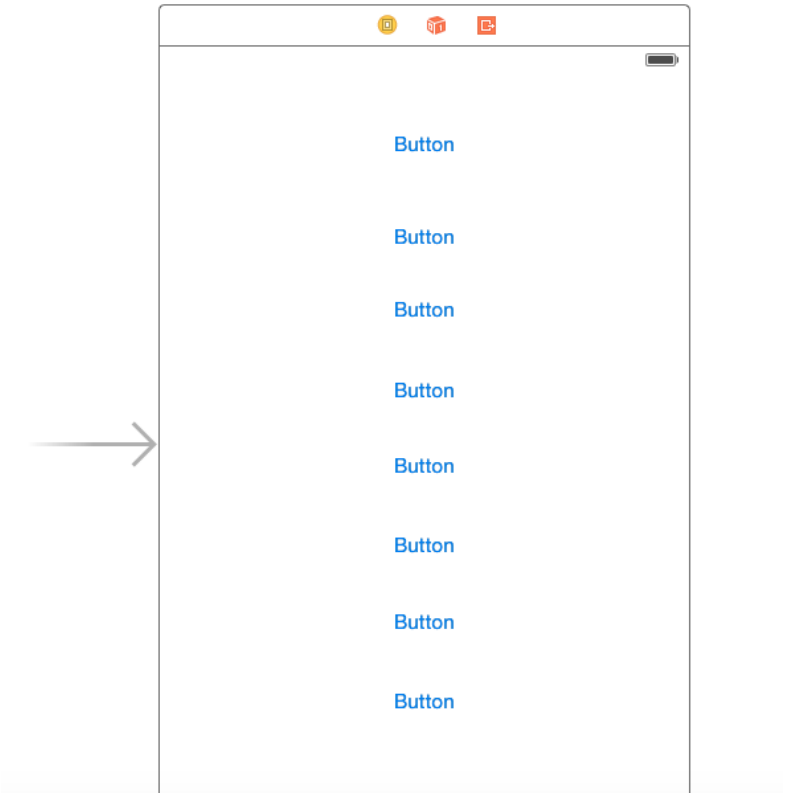
Siguiendo con la navegación en el storyboard, ubicarse en el ViewController de inicio, el cual posee la flecha a la izquierda.



A este ViewConroller, agregarle 8 botones, para esto acceder a la barra de utilidades y el apartado de objetos, arrastrar el objeto “Button” hasta el ViewController al cual se desea agregar el botón.

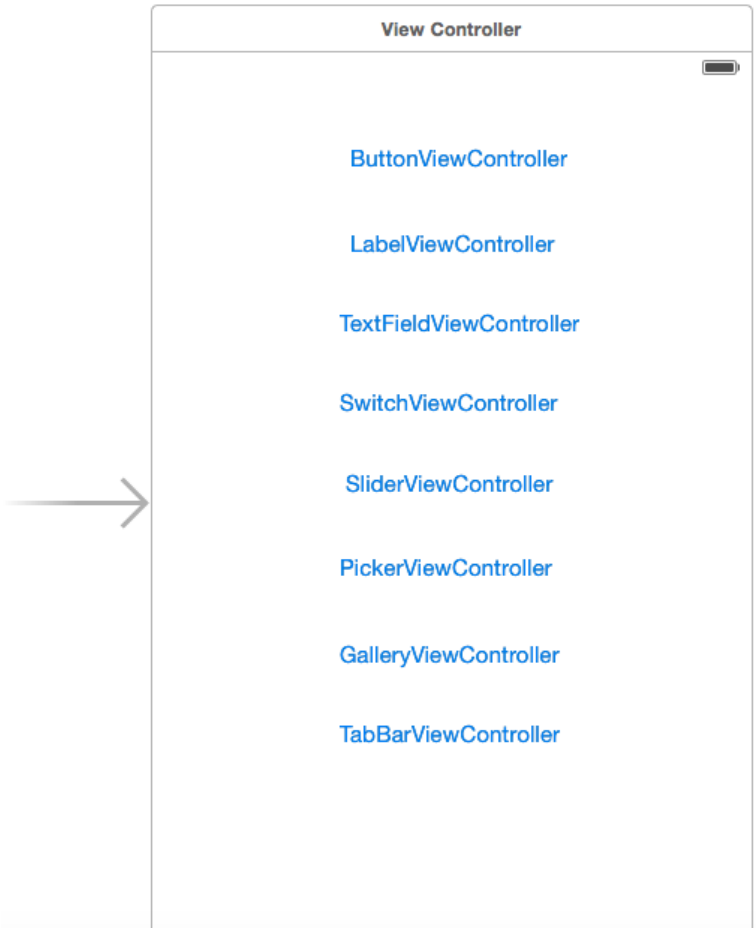


Cuando ya se ha agregado un botón, este mismo se puede copiar y pegar en el ViewController en lugar de estarlo arrastrando hasta el desde la barra de utilidades, al agregar los 8 botones, colocarlos como se observan en la pantalla siguiente.



A cada botón se le cambiara el texto desde el ambiente gráfico, seleccionándolo y dándole clic al texto, el cual se habilitara para edición.

Modificar el texto para que cada botón represente una pantalla a la cual se accederá desde este, como se presenta a continuación.



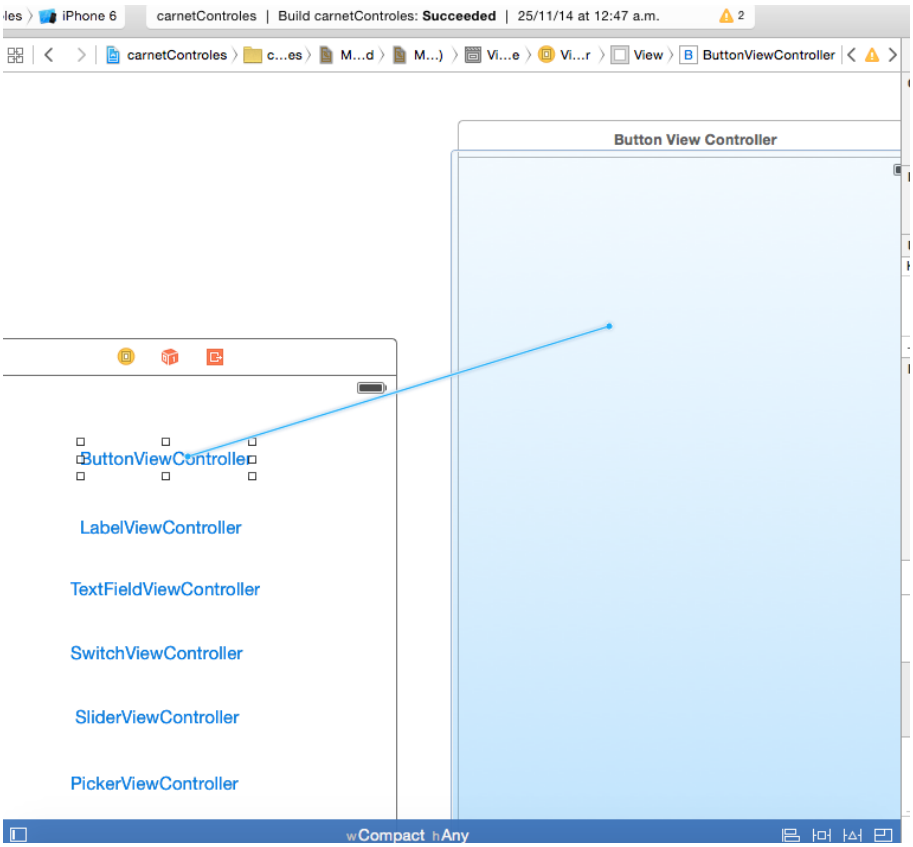
Vinculacion de Botones del Main.storyboard hacia Vistas

Para vincular la navegacion de los botones creados, proceder a seleccionar el boton "ButtonViewController"(un clic en el), desde este tiene dos formas de vincularlo:

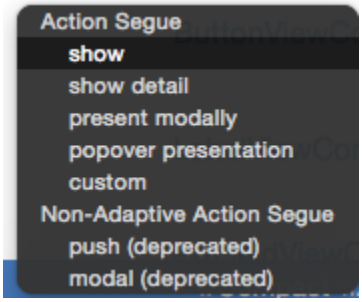
1)presione Control+arrastre del clic desde el boton hasta el ViewController respectivo o

2)Arrastrando con clic derecho desde el boton hasta el ViewController respectivo.

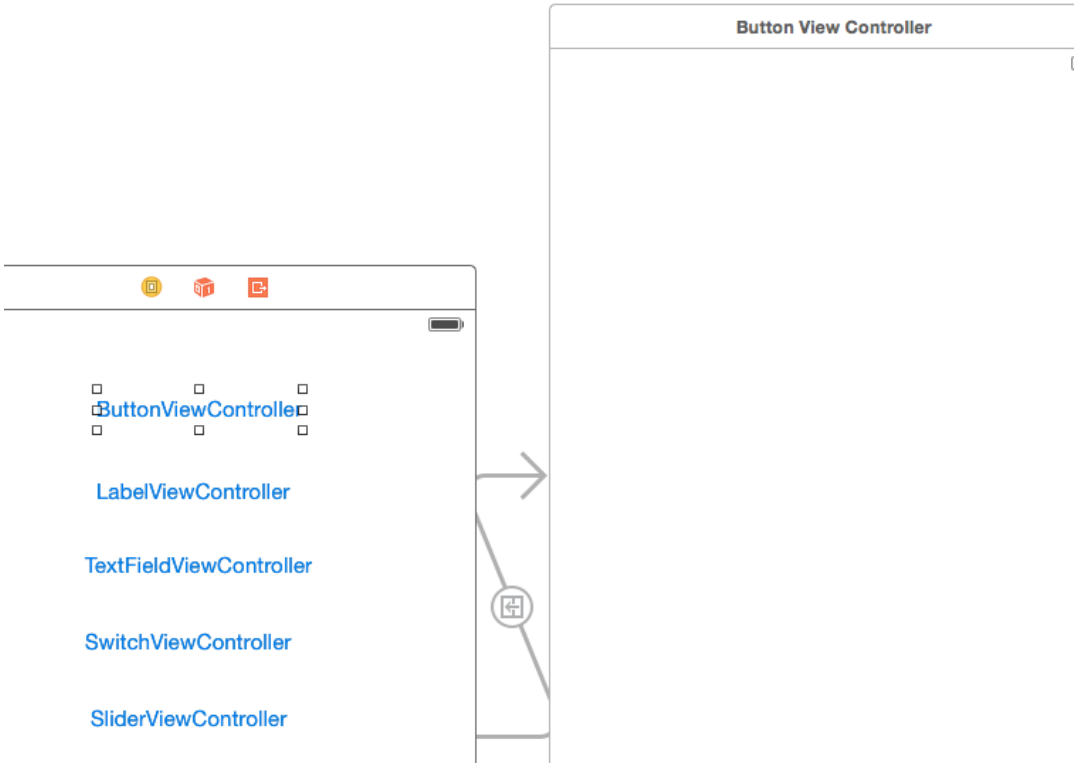
De cualquiera de las dos formas que se realice, se vera una imagen como esta:



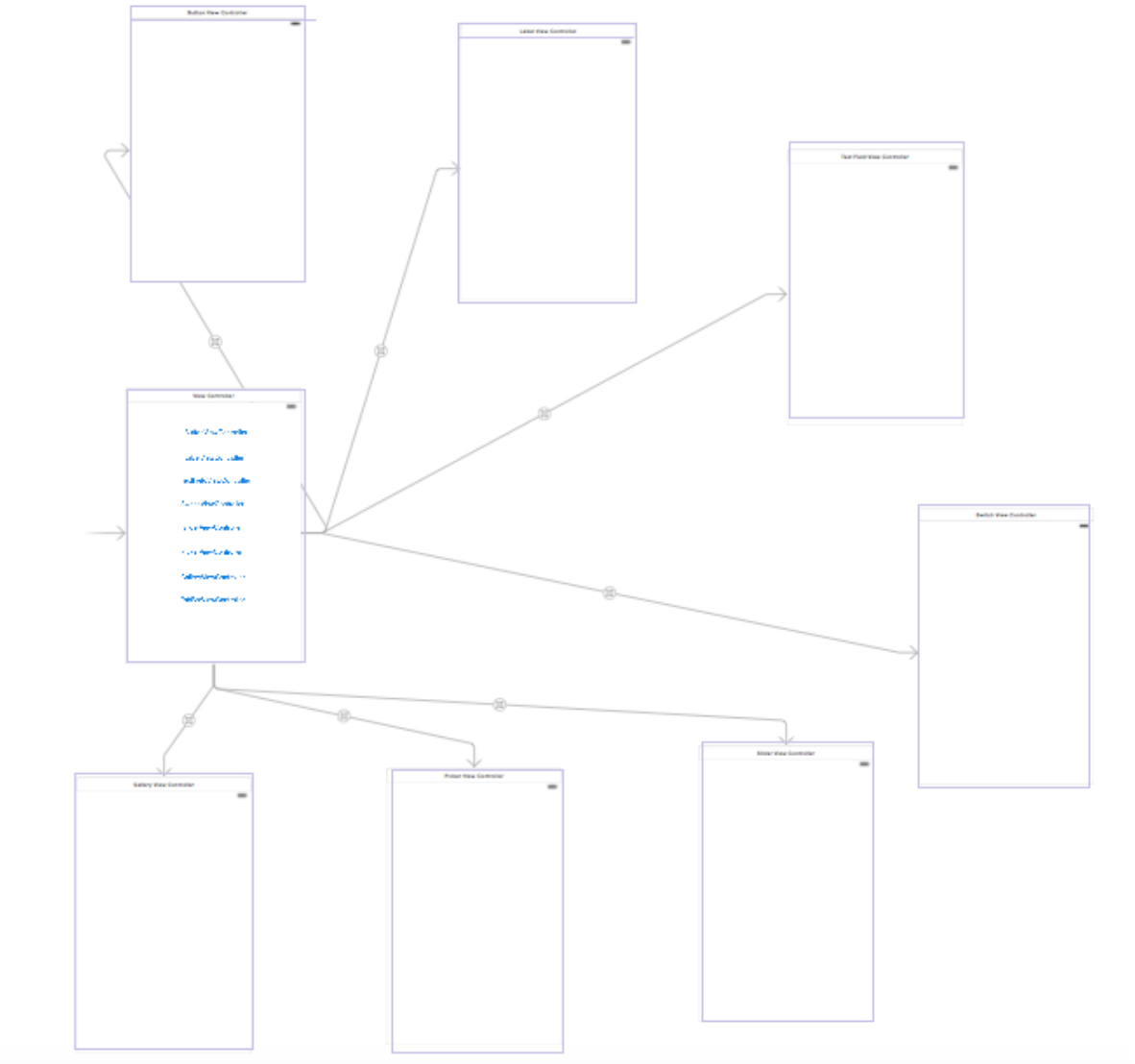
Al dejar de presionar el clic se presenta una ventana emergente en la cual debe seleccionar la opción “show”.



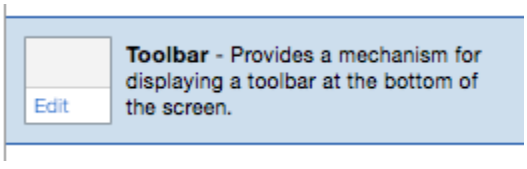
Esto genera una conexión visible desde el ViewController de inicio hacia el "ButtonViewController", como se observa a continuación.



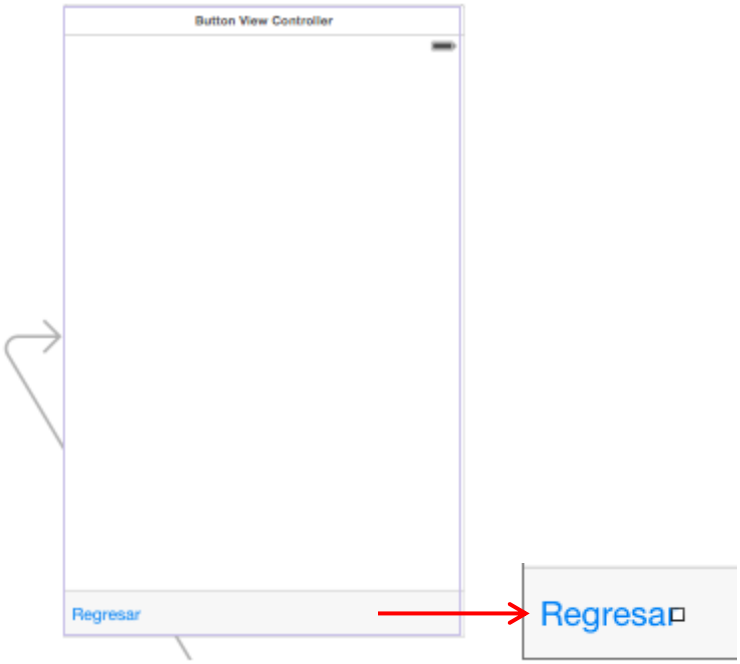
Repetir el proceso anterior para conectar los demás botones con sus pantallas correspondientes (ViewControllers). El stroryboard se mostrara como la imagen a continuación. Al hacer lo anterior, quedara un botón sin vínculo (TabBarViewController), más adelante se trabajara con él en la guía.



Para completar la navegación son necesarios botones o controles necesarios para el regreso a la pantalla de inicio desde cualquier otra pantalla, para esto utilizaremos el control llamado "Toolbar" que se presenta en la barra de utilidades en el apartado de objetos.



Arrastrar dicho control al “ButtonViewController” y cambiar el texto del botón en la barra agregada por el texto “Regresar”, la pantalla quedara como es muestra a continuación.

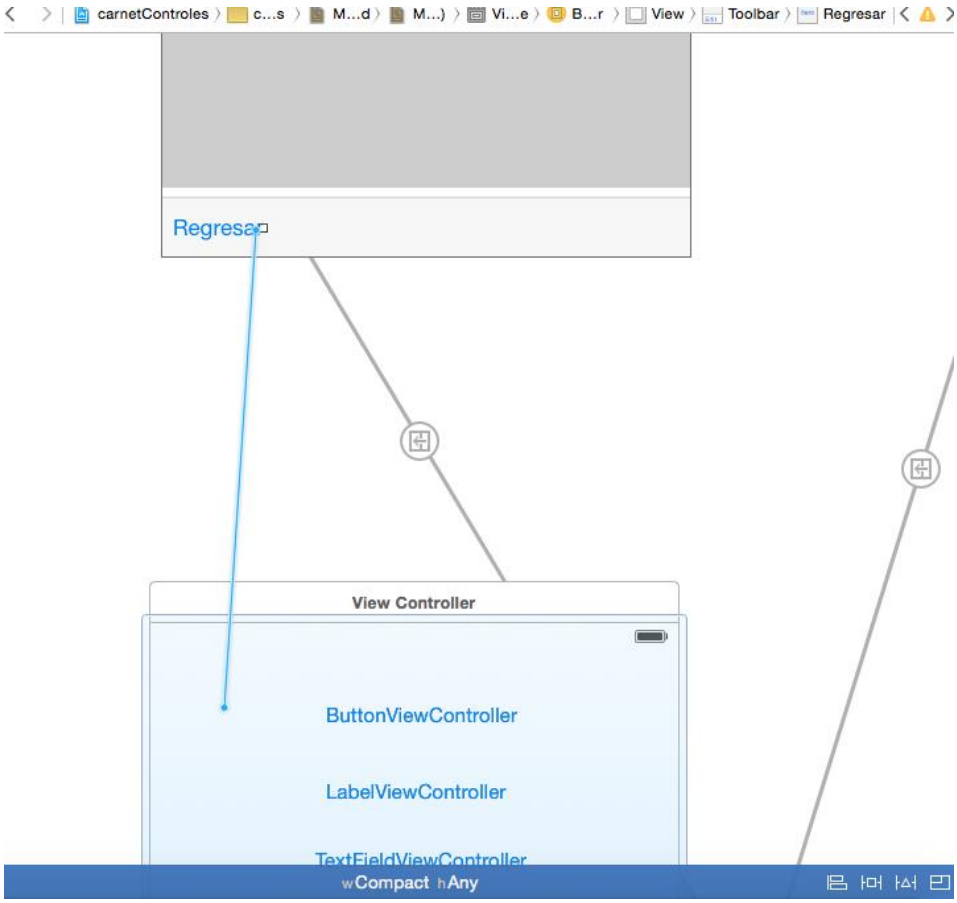


Para no estar repitiendo este procedimiento puede copiar esta barra de herramientas agregarla y modificar a los demás ViewController y solo ubicarla en el lugar correspondiente.

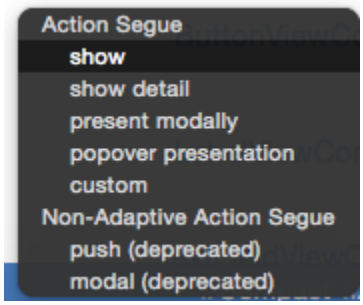
Enlace de Toolbars con Menú principal

Para enlazar esta barra con la pantalla de inicio, seleccionar el botón dentro de la barra (asegurarse que sea el botón seleccionado no la barra en general).

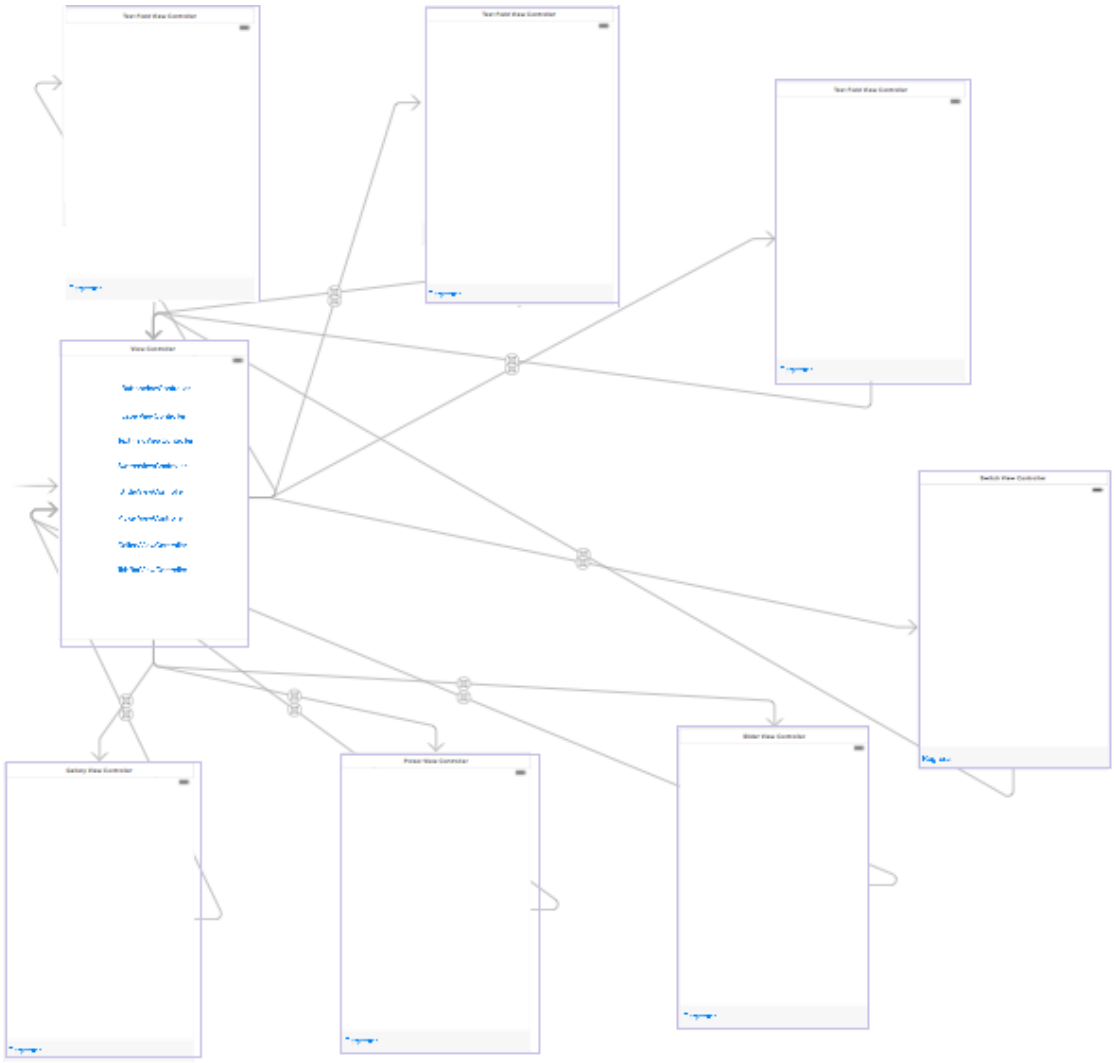
Luego de igual manera que con los botones presionando la tecla Ctrl junto al clic o solamente haciendo clic derecho, arrastrar desde el botón con el texto '"Regresar" hasta la pantalla de inicio.



Y al soltar aparecerá la une menú donde debe elegir show



Hacer lo mismo para todos los demás ViewControllers, al final se observaran las siguientes conexiones entre los ViewControllers ubicados en el storyboard.



Para probar la navegación ejecutar el programa en el simulador que desee de iphone, si no observa en pantalla completa, acceder al menú “Window” y en la opción Scale, puede modificar la visualización.

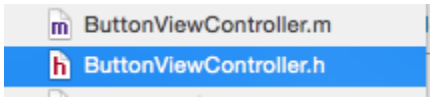
Nota: Recuerde que la programación relativa al botón para el TabBar se hará posteriormente.

Controles

ButtonViewController

Acceder a la vista de asistente (show the assistant editor)  para poder trabajar tanto el ambiente grafico como el código, ya que interactúan entre sí. Al acceder a esta vista si no se logra visualizar de manera cómoda, puede ocultar la barra de herramientas de la izquierda.

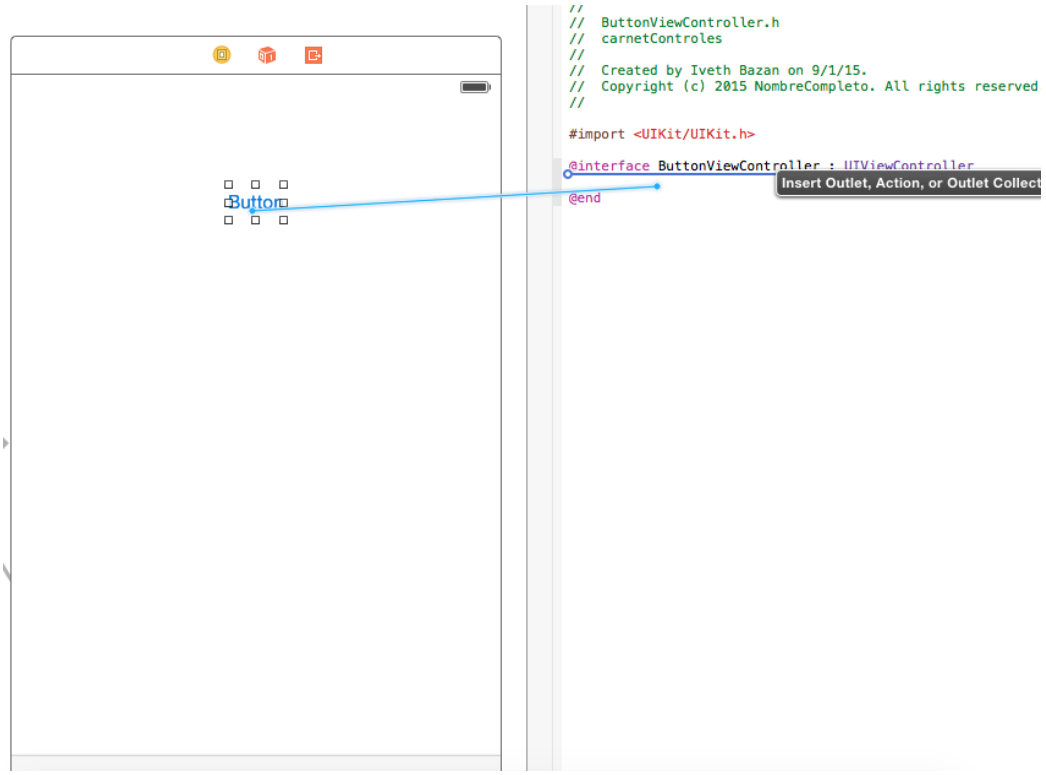
Seleccionar el "ButtonViewController", se observara que en el área de trabajo de la izquierda se presenta ya se el archivo .h o el archivo .m correspondiente a este ViewController, se puede alternar entre estos 2 archivos desde el la barra de direcciones como se muestra a continuación.



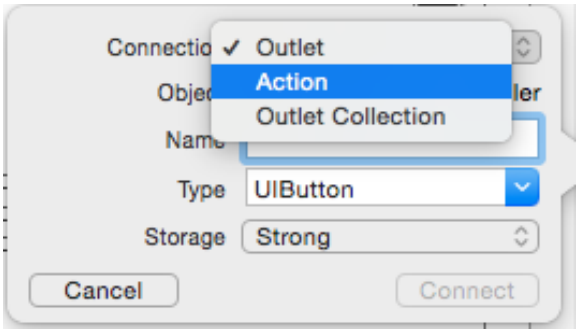
Ubicar en el área de trabajo la parte derecha será donde se podrá observar el archivo .h y .m. Mientras que en la izquierda se observará el main storyboard.

Agregar un botón en el "ButtonViewController".

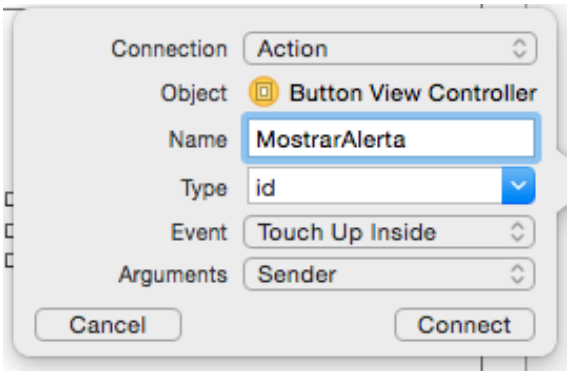
Seleccionar el botón y presionando Ctrl y clic izquierdo arrastrar hasta el archivo .h adentro del apartado @interface, como se observa a continuación.



Al realizar la conexión se presenta una ventana emergente adonde se especificara el tipo de enlace que desea crear, en esta ocasión se creara una conexión de tipo “Action” (acción), esto se cambia en el apartado de “Connection” en la ventana emergente.



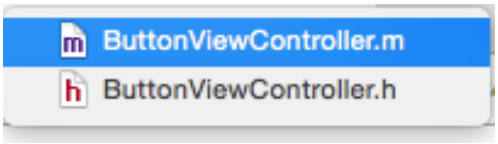
Modificar la conexión para que se establezca como se muestra en la siguiente imagen, luego presionar el botón “Connect”.



Se observa que en el archivo .h se ha modificado el siguiente código, generando así la declaración de la acción llamada MostrarAlerta.

```
@interface ButtonViewController : UIViewController
- (IBAction)MostrarAlerta:(id)sender;

@end
```

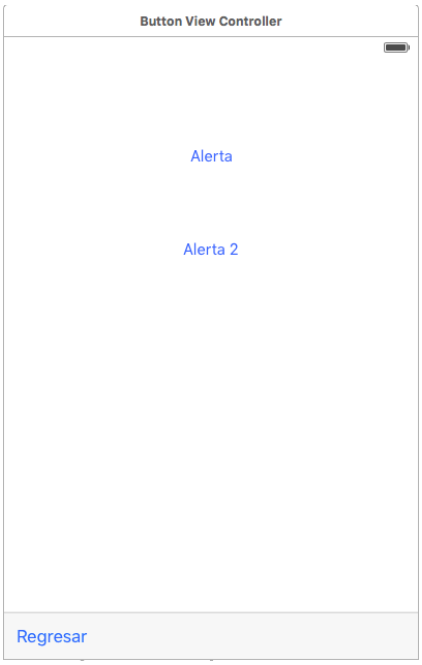
Para desarrollar el método MostrarAlerta acceder al archivo .m desde la barra de direcciones.



En el archivo .m se observara que el método ya se encuentra declarado, se modificara para que realice las siguientes acciones, este código agregado creara un nuevo objeto de tipo UIAlertView, el cual es una alerta con el texto especificado, el titulo especificado y el texto en el botón de la alerta también especificado.

```
- (IBAction)MostrarAlerta:(id)sender {
    UIAlertView *alerta = [[UIAlertView alloc] initWithTitle:@"Mensaje de alerta"
        message:@"Usted presiono el boton" delegate:nil cancelButtonTitle:@"Aceptar"
        otherButtonTitles:nil];
    [alerta show];

}
```

De la misma manera se agregará otro botón, con la acción llamada “MostrarAlerta2”, por lo cual, el viewController quedara de la siguiente manera.



El código a agregar en el segundo botón es el siguiente:

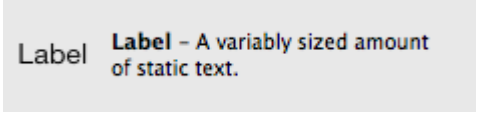
```
- (IBAction)MostrarAlerta2:(id)sender {
    UIAlertController *controlador = [UIAlertController alertControllerWithTitle:@"Error" message:@"Error de prueba"
        preferredStyle:UIAlertControllerStyleAlert];

    UIAlertAction *alertAction = [UIAlertAction actionWithTitle:@"Cancelar" style:UIAlertActionStyleDestructive handler:
        ^(UIAlertAction *action){
        NSLog(@"Se presiono cancelar!");
    }];

    [controlador addAction:alertAction];
    [self presentViewController:controlador animated:YES completion:nil];
}
```

Corralo y pruebe el “ButtonViewController” ya finalizado.

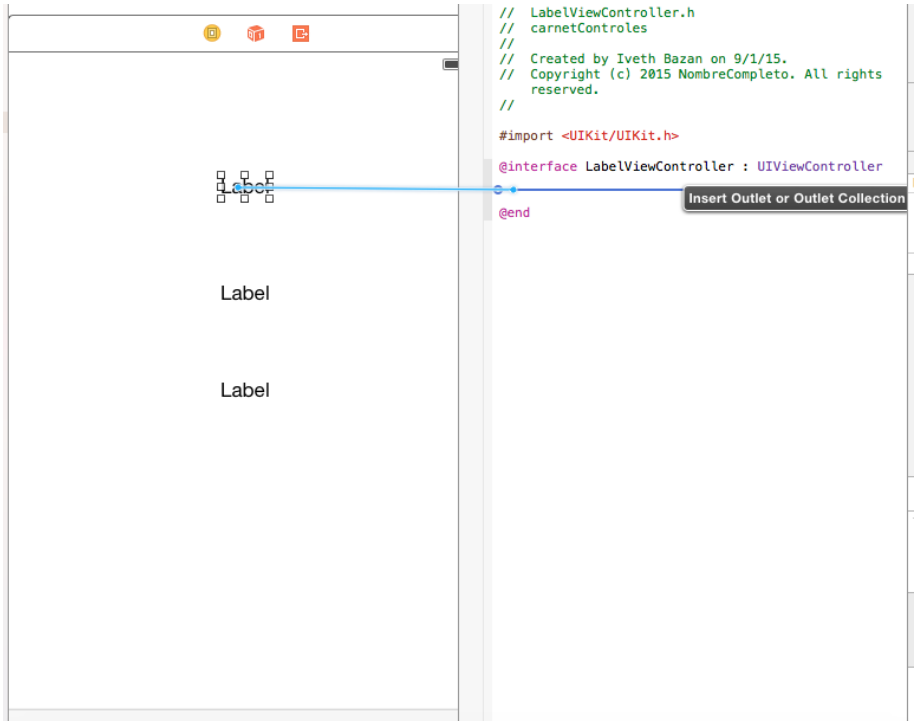
LabelViewController

Ubicarse en el storyboard y seleccionar el “LabelViewController”, esto debe generar un cambio en el área de trabajo, cargando así los archivos .h y .m correspondientes al ViewController.

Agregar al “LabelViewController“ 3 labels, arrastrándolos desde la barra de herramientas en el apartado de objetos.

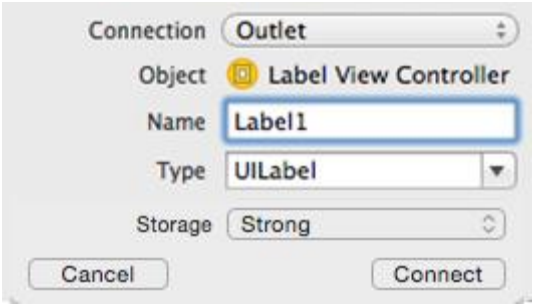


De igual manera como se realizó el botón, se realiza la conexión presionado Ctrl + clic izquierdo y arrastrando hasta el archivo .h.



Esto genera una ventana emergente de conexión, a diferencia del botón anteriormente trabajado, este tipo de conexión será de tipo “Outlet” (elemento), modificar la configuración de la conexión como se presenta a continuación.

Hacer este procedimiento para los 3 labels correspondientes.



Al agregar las 3 conexiones correspondientes a cada Label, se observara el archivo .h de la siguiente manera.

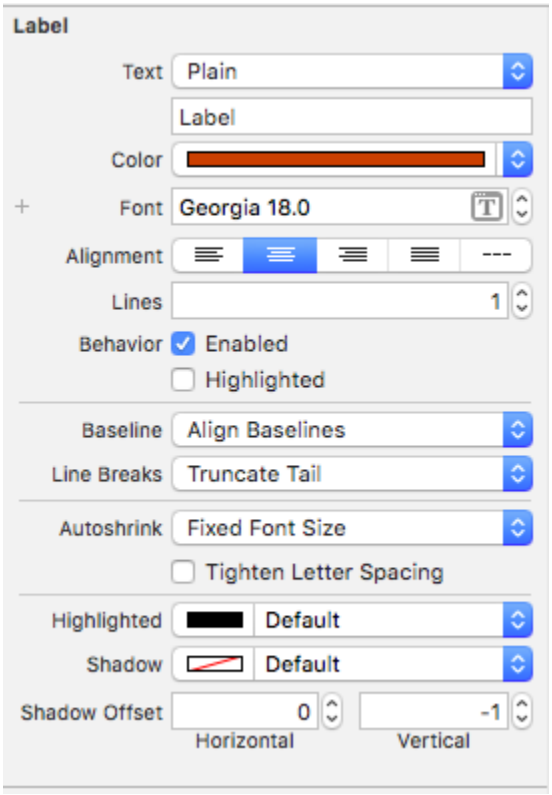
```
@interface LabelViewController : UIViewController

@property (strong, nonatomic) IBOutlet UILabel *Label1;
@property (strong, nonatomic) IBOutlet UILabel *Label2;
@property (strong, nonatomic) IBOutlet UILabel *Label3;

@end
```

A continuación acceder al archivo .m de “LabelViewController”, proceder a modificar el método “ViewDidLoad”, que es el método que se ejecuta cuando el ViewController se cargó con éxito, de la siguiente manera.

Se observa que se modifican diferentes atributos de cada label.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    UIColor *verde = [UIColor colorWithRed:0 green:255 blue:0 alpha:1];

    _Label1.text=@"Bienvenido a iOS";
    _Label2.text=@"Bienvenido a iOS";
    _Label3.text=@"Bienvenido a iOS";

    [_Label1 setAdjustsFontSizeToFitWidth:true];
    [_Label1 setTextColor:verde];
    [_Label1 setBackgroundColor:[UIColor cyanColor]];
    [_Label2 setTextColor:[UIColor colorWithRed:0 green:0 blue:255 alpha:1] ];
    [_Label2 setBackgroundColor:[UIColor magentaColor]];
    [_Label2 setFont:[UIFont fontWithName:@"Arial" size:12]];

}
```

Para modificar de manera visual el label, seleccionar el label3 y acceder en la barra de herramientas al apartado “Attribute inspector”, con el icono que es el inspector de atributos correspondientes al elemento seleccionado, en este caso proceder a modificar las características correspondientes al label.

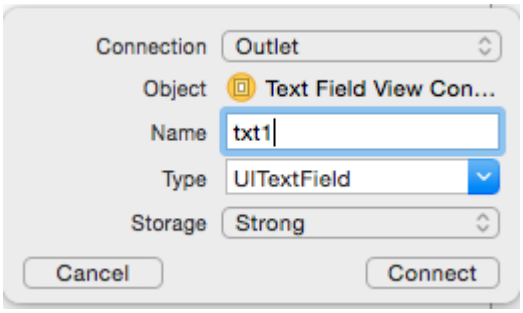


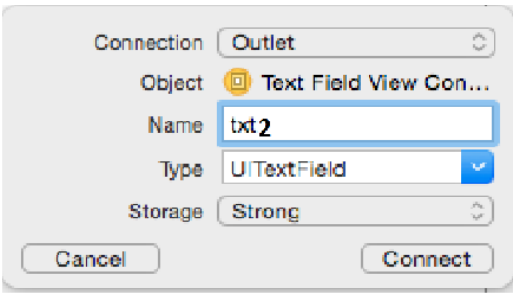
Probar la aplicación para observar si se realizan las modificaciones tanto codificadas como modificadas de manera gráfica.

TextFieldViewController

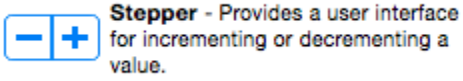
Para trabajar el objeto "TextField" se agregaran 2 de estos al "TextFieldViewController".

Acceder al archivo .h correspondiente en el área de trabajo de la derecha y crear conexiones para cada TextField, estas conexiones serán de tipo "Outlet" y dándole nombre a la conexión de "txt1" y "txt2" respectivamente, como se observa a continuación.

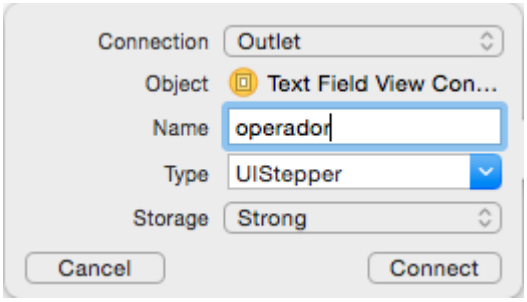


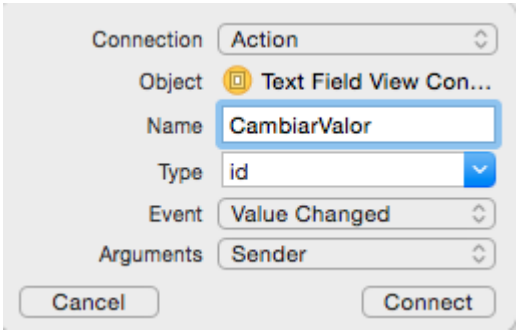


Se trabajara también con el elemento llamado "Stepper", arrastrar un elemento de este tipo desde la barra de herramientas hasta el "TextFieldViewController"

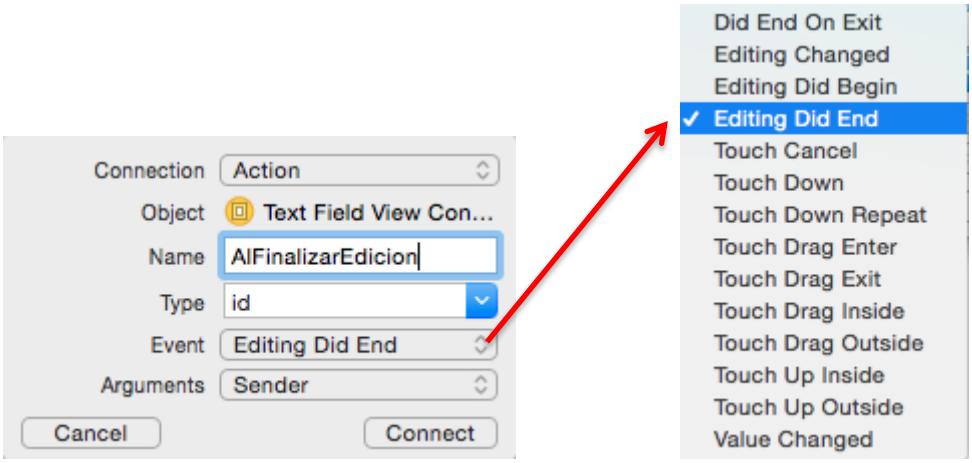


Se crearan 2 conexiones para el "Stepper" una de tipo "Outlet" y otra de tipo "Action", siempre se realizaran en el archivo .h correspondiente a "TextFieldViewController", con las configuraciones como se presentan a continuación.

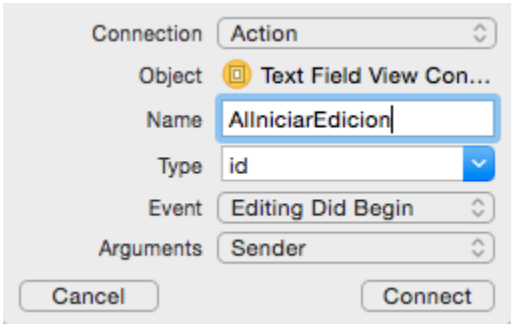




También se crearan conexiones de tipo “Action” para cada TextField. Para el txt1 crear una conexión de tipo “Action” y cambiar el evento a “Editing did end” que corresponde al evento cuando la edición del TextFIeld finaliza, colocar de nombre a esta conexión “Al finalizar edición.”



Para el txt2 crear una conexión de tipo “Action” con evento “Editing did begin” que correponde al evento cuando se inicia la edición, esta conexión tendrá el nombre de “AlIniciarEdicion”



Al agregar las 6 conexiones, 2 para cada elemento, se observara el archivo .h de la siguiente manera.

```
@interface TextFieldViewController : UIViewController
@property (strong, nonatomic) IBOutlet UITextField *txt1;
@property (strong, nonatomic) IBOutlet UITextField *txt2;
@property (strong, nonatomic) IBOutlet UIStepper *operador;
- (IBAction)AlFinalizarEdicion:(id)sender;
- (IBAction)CambiarValor:(id)sender;
- (IBAction)AlIniciarEdicion:(id)sender;

@end
```

Para los métodos creados modificar el código en el archivo .m para que se observe como la imagen siguiente.

```
- (IBAction)CambiarValor:(id)sender {
    _txt2.text=[NSString stringWithFormat:@"%.2f",_operador.value];
}

- (IBAction)AlFinalizarEdicion:(id)sender {
    [_txt1 setBackgroundColor:[UIColor redColor]];
}

- (IBAction)AlIniciarEdicion:(id)sender {
    _txt2.enabled=false;
}
```

Se observa que el método enlazado con el Stepper llamado “CambiarValor” genera un cambio en el texto del TextField declarado como txt2, dándole el nuevo valor del contador que está efectuando el elemento declarado “operador” que es de tipo Stepper.

El método “AlFinalizarEdicion” realiza un cambio en el fondo de color del elemento txt1, cambiándolo a rojo.

El método “AlIniciarEdicion” deshabilita el txt2, evitando así que se pueda modificar directamente este valor.

En algún momento se puede presentar inconvenientes con el teclado a la hora de modificar algún TextField, el teclado no se oculta automáticamente si no se presiona algún otro elemento, para evitar este inconveniente generaremos otra conexión de tipo “Action” al txt1, la configuración quedara de la siguiente manera.

Modificar el método en el archivo .m para que se vea como el código siguiente, esto ocultara el teclado al presionar Intro o Enter en el teclado.

```
- (IBAction)SalirDelTeclado:(id)sender {
    [_txt1 resignFirstResponder];
    [_txt2 resignFirstResponder];
}
```

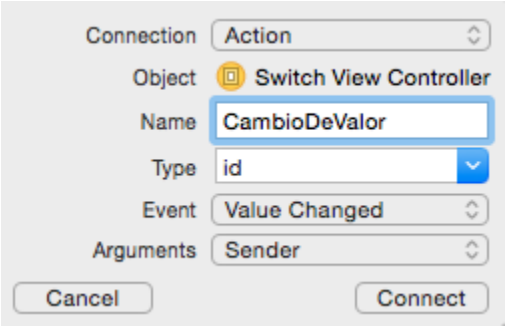
Correr la aplicación y probar el TextFieldViewController

SwitchViewController

En el "SwitchViewController" agregar un elemento de tipo Switch.

A este elemento se le crearan 2 tipos de conexiones. La conexión de tipo "Outlet", se llamara "switch1", realizando la conexión siempre al archivo .h correspondiente a "SwitchViewController".

La conexión de tipo "Action" se configurar de igual manera que la imagen mostrada a continuación.



También se agregar un elemento de tipo Label, al cual se le creara una conexión de tipo "Outlet" con el nombre de label1.

Al finalizar la configuración quedará el archivo .h de la siguiente manera:

```
@interface SwitchViewController : UIViewController
@property (strong, nonatomic) IBOutlet UISwitch *switch1;
- (IBAction)CambioDeValor:(id)sender;
@property (strong, nonatomic) IBOutlet UILabel *Label1;

@end
```

El método creado con el nombre de “CambioDeValor” se modificara en el archivo .m con el siguiente código.

```
- (IBAction)CambioDeValor:(id)sender {
    UIView *v = [self view]; //Accede al view del view
        controller
    if (![_switch1 isOn]) {
        [v setBackgroundColor:[UIColor blackColor]];
        [_Label1 setTextColor:[UIColor whiteColor]];
        _Label1.text=@"Apagado";
    }

    else
    {
        [v setBackgroundColor:[UIColor whiteColor]];
        [_Label1 setTextColor:[UIColor blackColor]];
        _Label1.text=@"Encendido";
    }

}
```

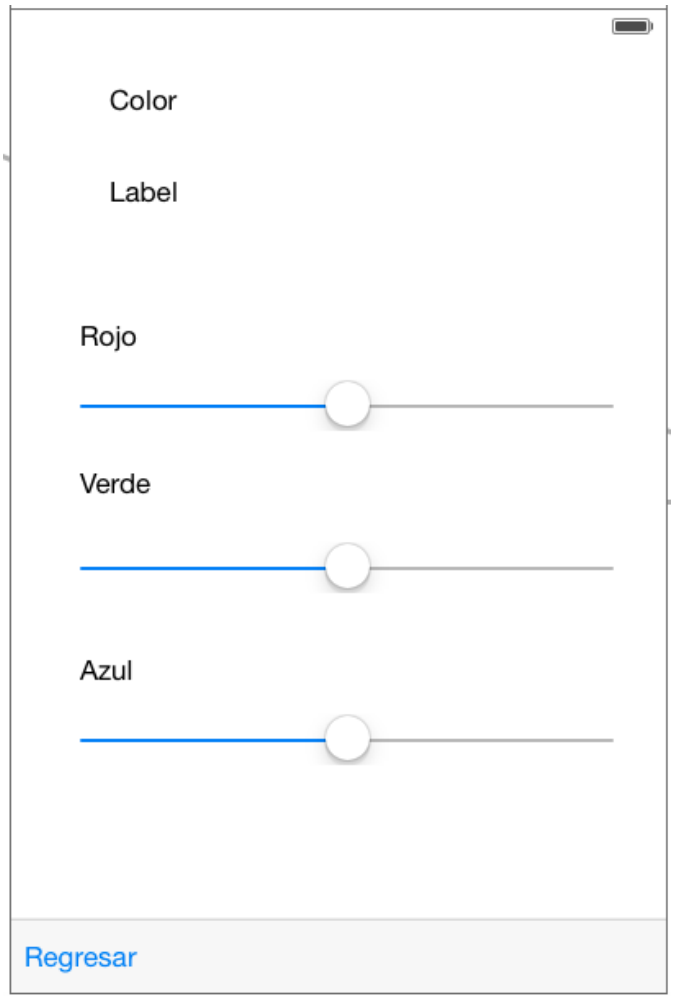
Como se observa el código presenta un cambio en ambos estados del switch, cada vez que se realice un cambio de valor el método será invocado y realizando así cambios de los atributos del View correspondiente al “SwitchViewController” como también atributos correspondientes al label.

Probar la aplicación para ver si se observa el cambio de los atributos correspondientes al cambiar el valor del switch.

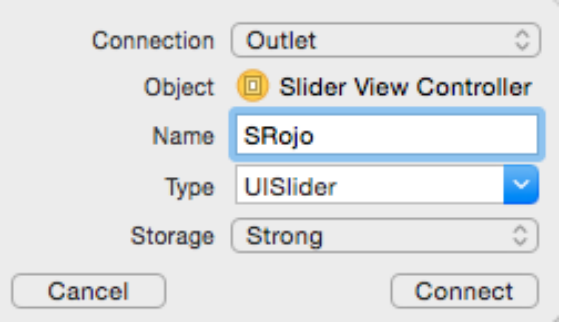
SliderViewController

Para el "SliderViewController" agregaremos 3 elementos de tipo Slider.

Se agregaran adicionalmente 5 labels y los textos de 4 de ellos serán modificados para que se observe como la pantalla siguiente:



Se realizaran conexiones de tipo “Outlet” para cada uno de los Sliders uno de los cuales llevará el nombre de “SRojo”.



Realizar el mismo procedimiento para los otros 2 sliders con nombres: “SVerde” y “SAzul” respectivamente. Como también se creara una conexión de tipo “Outlet” al label el cual el texto no ha sido modificado, este tendrá el nombre de “Color”. A continuacion se creara una conexión de tipo “Action” desde cualquiera de los Sliders, dándole a esta el nombre de “CambiarColor” y el evento “ValueChanged”.

Estas conexiones se realizaran siempre al archivo .h correspondiente a “SliderViewController”.

Al haber creado todas las conexiones el archivo “SliderViewController.h” se observara de la siguiente manera.

```
@interface SliderViewController : UIViewController
@property (strong, nonatomic) IBOutlet UISlider *SRojo;
@property (strong, nonatomic) IBOutlet UISlider *SVerde;
@property (strong, nonatomic) IBOutlet UISlider *SAzul;
@property (strong, nonatomic) IBOutlet UILabel *Color;
- (IBAction)CambiarColor:(id)sender;

@end
```

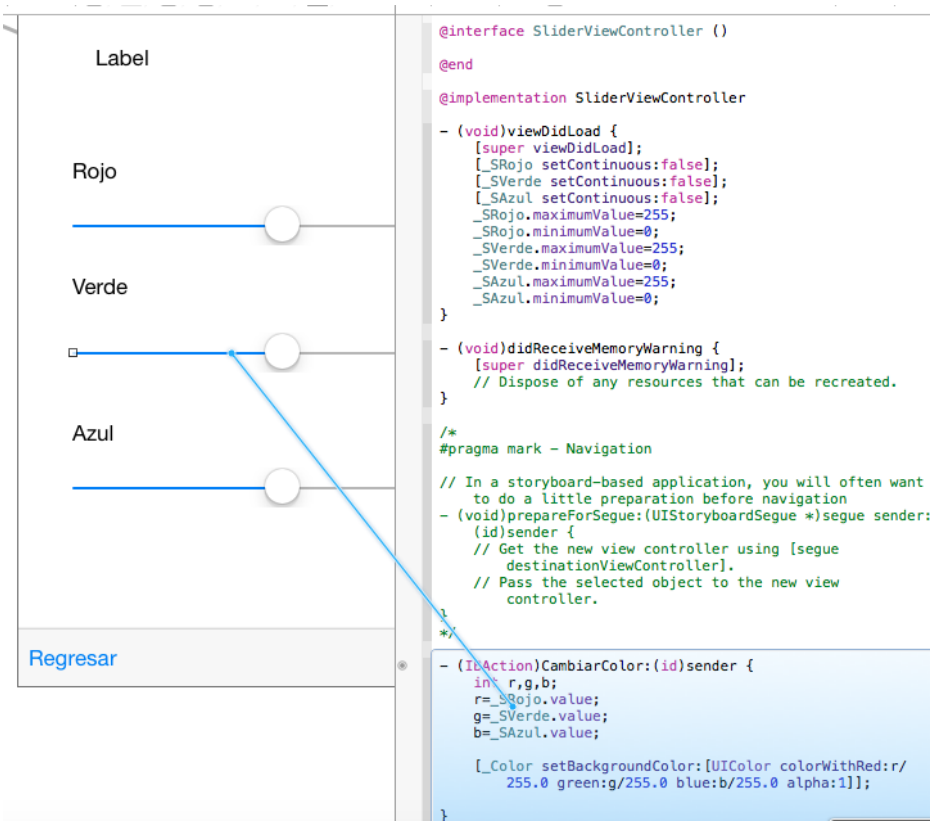
Acceder al archivo .m correspondiente y modificar el método viewDidLoad, como se muestra a continuación. Esta modificación asigna valores predeterminados de máximo y mínimo en cada slider.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    [_SRojo setContinuous:false];
    [_SVerde setContinuous:false];
    [_SAzul setContinuous:false];
    _SRojo.maximumValue=255;
    _SRojo.minimumValue=0;
    _SVerde.maximumValue=255;
    _SVerde.minimumValue=0;
    _SAzul.maximumValue=255;
    _SAzul.minimumValue=0;
}
```

Modificar el método CambiarColor con el código que presenta a continuación, este código obtiene los valores de los sliders y los trunca a enteros, siguiente de este truncamiento asigna un fondo de color al label con nombre “Color” que está determinado por el valor de los sliders, dividido entre 255, ya que el método de UIColor utilizado trabaja con porcentajes.

```
- (IBAction)CambiarColor:(id)sender {
    int r,g,b;
    r=_SRojo.value;
    g=_SVerde.value;
    b=_SAzul.value;

    [_Color setBackgroundColor:[UIColor colorWithRed:r/255.0 green:g/255.0 blue:b/255.0 alpha:1]];

}
```

Ya que el método CambiarColor solo corresponde a un slider, presionar Control + arrastrar el clic izquierdo desde algún slider que no esté enlazado con este método y finalizar la conexión con el método CambiarColor ya modificado, se observa que todo el método cambia a un color azul, al dejar de presionar se realiza la conexión.



Probar aplicación, observar como el label cambia de color según la combinación de los sliders.

PickerViewController

Para el funcionamiento del elemento "PickerView" agregaremos un elemento de este tipo, como también un elemento label.

La implementación del PickerView no posee métodos específicos para la modificación directa de este elemento, por lo cual se procederá a modificar métodos abstractos provenientes de una interfaz llamada PickerViewDelegate y de PickerViewDataSource, para esto tenemos que designar que nuestro "PickerViewController" hereda dichos métodos abstractos, para esto en el archivo .h, modificaciones la sentencia de @interface con la siguiente que se muestra a continuación.

```
@interface PickerViewController : UIViewController<UIPickerViewDelegate,UIPickerViewDataSource>
```

Después realizaremos una conexión de tipo "Outlet" para el PickerView, dándole de nombre "Selector" y una igual al label dándole de nombre "Capital". El archivo .h quedaría de la siguiente manera.

```
@interface PickerViewController : UIViewController<UIPickerViewDelegate,UIPickerViewDataSource>
@property (strong, nonatomic) IBOutlet UIPickerView *Selector;
@property (strong, nonatomic) IBOutlet UILabel *Capital;

@end
```

Para que la aplicación aplique toda modificación de los métodos abrastractos que se va a realizar al elemento Selector se agrega el siguiente código en el método viewDidLoad.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    _Selector.delegate=self;
    _Selector.showsSelectionIndicator=YES;
    [self.view addSubview:_Selector];
    [_Capital setAdjustsFontSizeToFitWidth:true];
}
```

Ahora proceder a modificar los métodos abstractos como se presenta a continuación.

```
//En este método se maneja el evento de cambio de selección
-(void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row inComponent:(NSInteger)component{

    switch (row) {
        case 0:
            _Capital.text=@"Brasilia";
            break;
        case 1:
            _Capital.text=@"Buenos Aires";
            break;
        case 2:
            _Capital.text=@"San Salvador";
            break;
        case 3:
            _Capital.text=@"Munich";
            break;
        case 4:
            _Capital.text=@"Paris";
            break;

        default:
            break;
    }

}
```

En el método “pickerViewdidSelectRowinComponent” se maneja el cambio de selección del elemento selector, se ejecuta este método cuando el evento de cambio de valor del elemento Selector se presenta.

```
// En este metodo se especifica la cantidad de filas que contendra el picker
- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:(NSInteger)component
    {
    NSUInteger numFilas = 5;

    return numFilas;
}
```

En el método “pickerViewnumberOfRowsInComponent” se especifica cuantas filas posee el elemento Selector.

```
// En este metodo se especifica cuantos componentes tendra cada fila del picker
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView {
    return 1;
}
```

En el método “numberOfComponentsInPickerView” especifica cuantos componentes tendrá cada fila del elemento Selector, en esta ocasión se trabajara solo con 1, de manera de ejercicio intentar modificar este valor y trabajar con varios componentes.

```
//Escpecifica ele titulo para cada fila del coponente expecifico

-(NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)
    component{
    NSString *titulo;
    switch (row) {
        case 0:
            titulo=[@"" stringByAppendingString:@"Brasil"];
            break;
        case 1:
            titulo=[@"" stringByAppendingString:@"Argentina"];
            break;
        case 2:
            titulo=[@"" stringByAppendingString:@"El Salvador"];
            break;
        case 3:
            titulo=[@"" stringByAppendingString:@"Alemania"];
            break;
        case 4:
            titulo=[@"" stringByAppendingString:@"Francia"];
            break;

        default:
            break;
    }
    return titulo;
}
```

En el método “pickerViewtitleForRowforComponent” se especifica el título que se muestre en cada fila para cada componente.

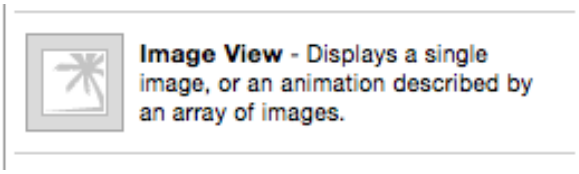
```
//Especifica el ancho del picker para cada elemento
-(CGFloat)pickerView:(UIPickerView *)pickerView widthForComponent:(NSInteger)component{
    int sectionWidth=300;
    return sectionWidth;
}
```

En el método “pickerViewwidthForComponent” especifica el ancho de cada componente en las filas.

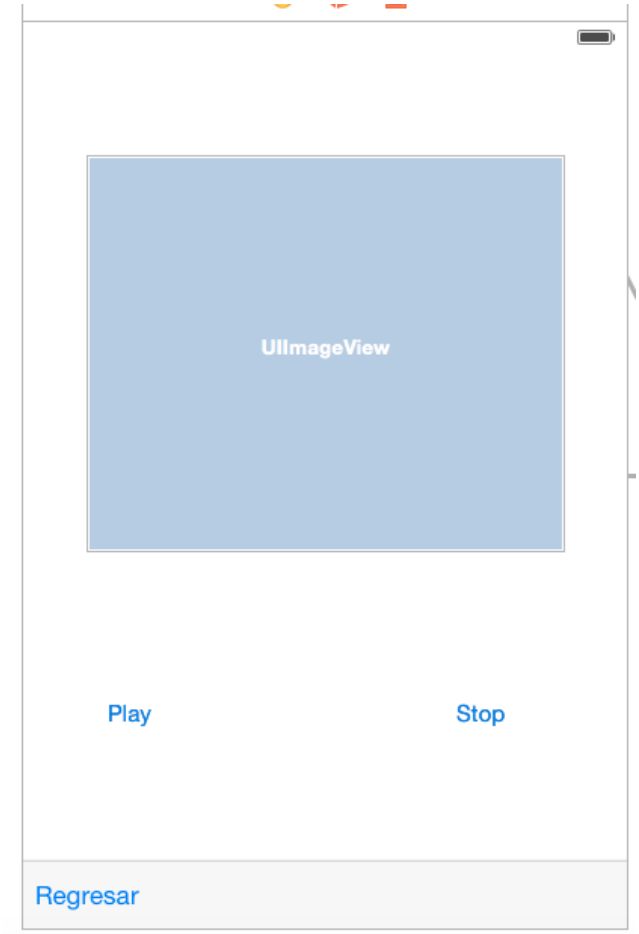
Probar la aplicación para ver si se observa el evento de cambio de selección y los titulos asignados a cada fila del selector.

GalleryViewController

Para mostrar una galería de imágenes de manera de presentación se trabajar con el elemento "ImageView", agregar uno al "GalleryViewController".



Para este elemento crear una conexión de tipo "Outlet" con el nombre de "Animacion", también se agregaran dos elementos de tipo "Button" a los cuales se les creara una conexión de tipo "Action" con el evento TouchUpInside, y con nombres "Play" y "Stop" respectivamente.



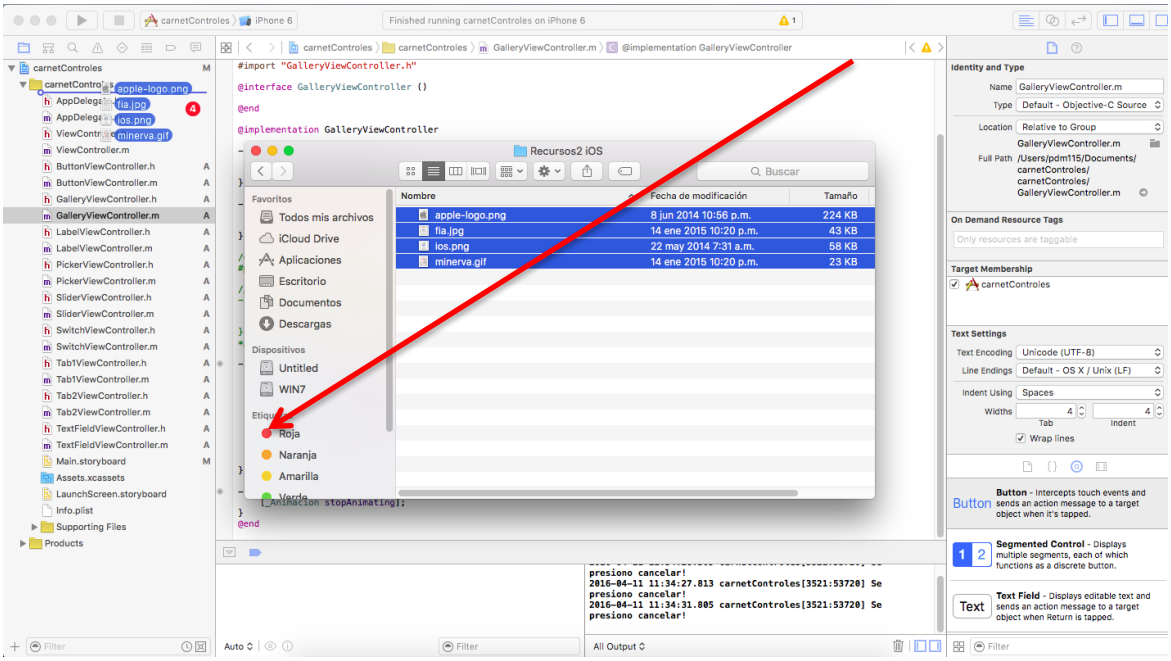
Al crear las conexiones se observara el archivo “GalleryViewController.h” de la siguiente manera.

```
@interface GalleryViewController : UIViewController
@property (strong, nonatomic) IBOutlet UIImageView *Animacion;
- (IBAction)Play:(id)sender;
- (IBAction)Stop:(id)sender;


@end
```

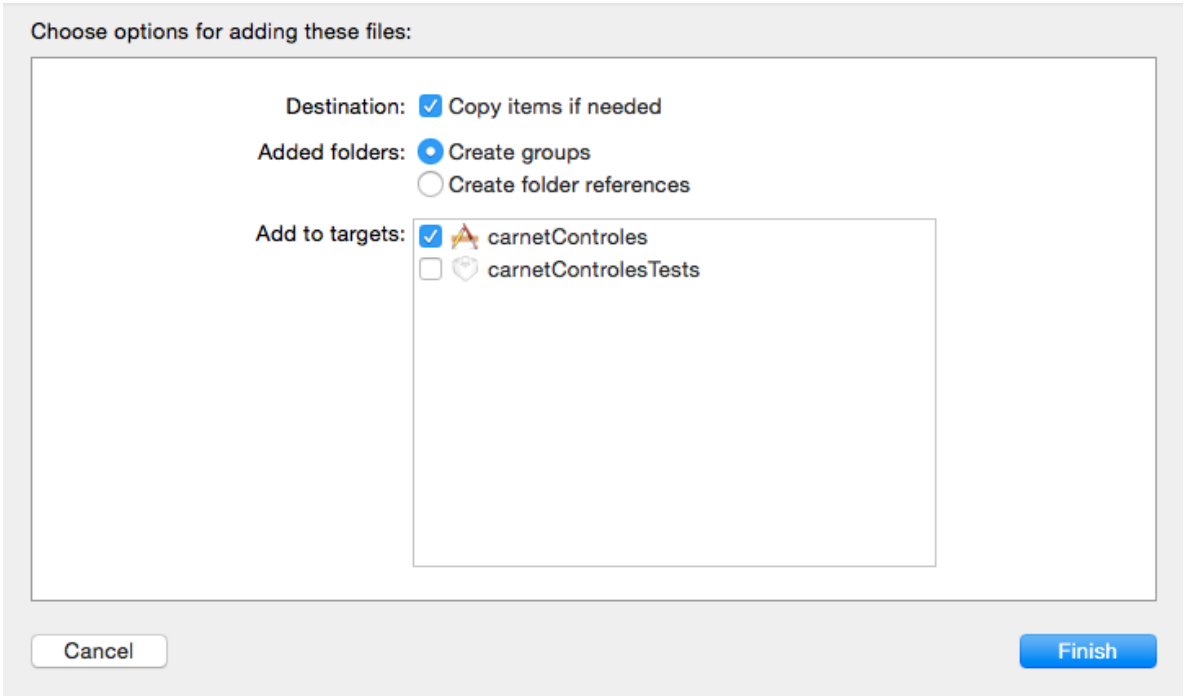
Para realizar el ejercicio de la galería de imágenes, se descargarán un conjunto de recursos los cuales se encuentran en el AulaVirtual.

Recursos_Guia2IOS

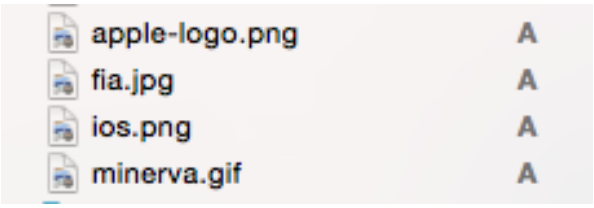
Para trabajar con imágenes se deben de seleccionar desde la carpeta adonde se posean y arrastrarse hasta la barra de la izquierda del área de trabajo, dentro del proyecto en el cual se está trabajando (en carpeta de supporting files).



Lo anterior generará una ventana emergente como la que se presenta a continuación, seleccionar las opciones como se presenta en la siguiente imagen y presionar “Finish”



Ahora se observara que los elementos arrastrados se verán incorporados en el proyecto, los cuales ya pueden ser accedidos desde el código directamente.



Modificar el los métodos correspondientes como se muestra a continuación

```
- (IBAction)Play:(id)sender {
    _Animacion.animationImages=[NSArray arrayWithObjects:[UIImage imageNamed:@"apple-logo.png"],
                                                         [UIImage imageNamed:@"fia.jpg"],
                                                         [UIImage imageNamed:@"ios.png"],
                                                         [UIImage imageNamed:@"minerva.gif"],nil];

    _Animacion.animationDuration=2.0f;
    _Animacion.animationRepeatCount=0;
    [_Animacion startAnimating];
    [self.view addSubview:_Animacion];

}
```

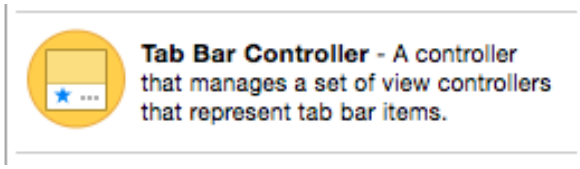
El método Play asigna al elemento Animación una lista de elementos de tipo UIImage y a partir de ellos genera una animación de ciertos intervalos con repetición.

```
- (IBAction)Stop:(id)sender {
    [_Animacion stopAnimating];
}
```

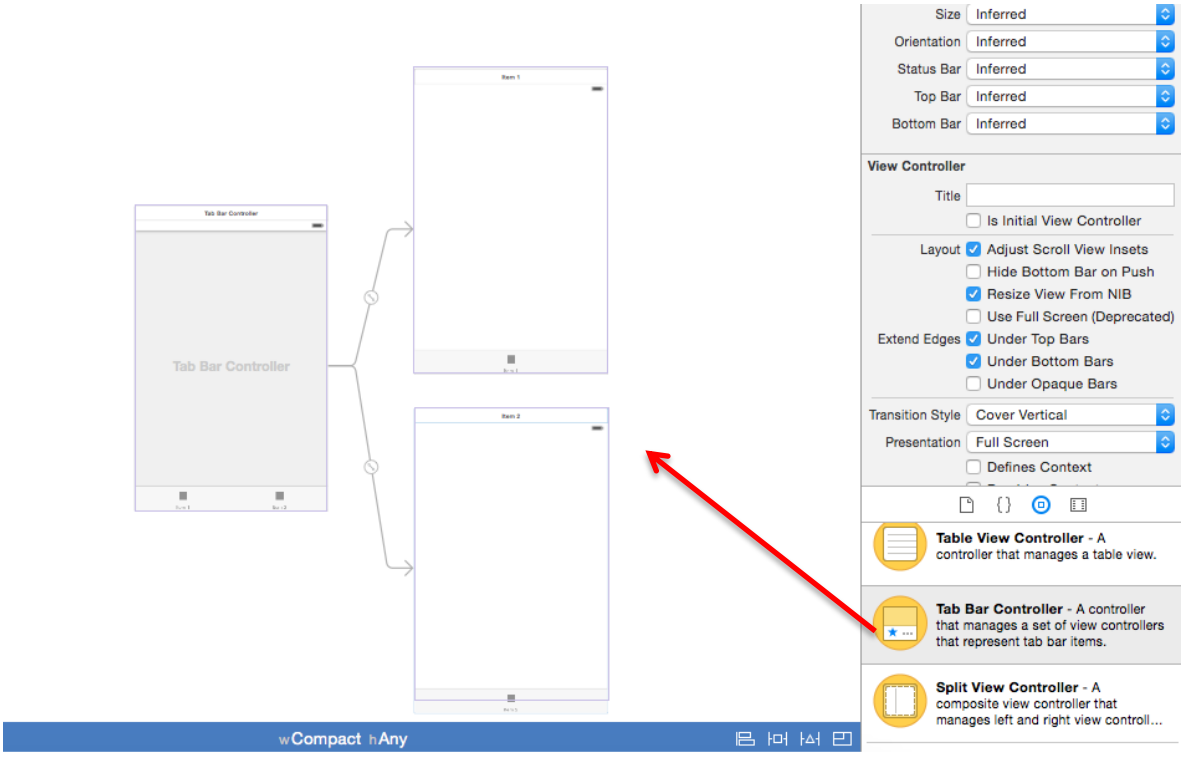
El método Stop, detiene la animación creada por el método Play.

TabViewController

Para manejar el funcionamiento del elemento Tab, se procederá a agregar un ViewController de diferente tipo, el cual es llamado “Tab Bar Controller”

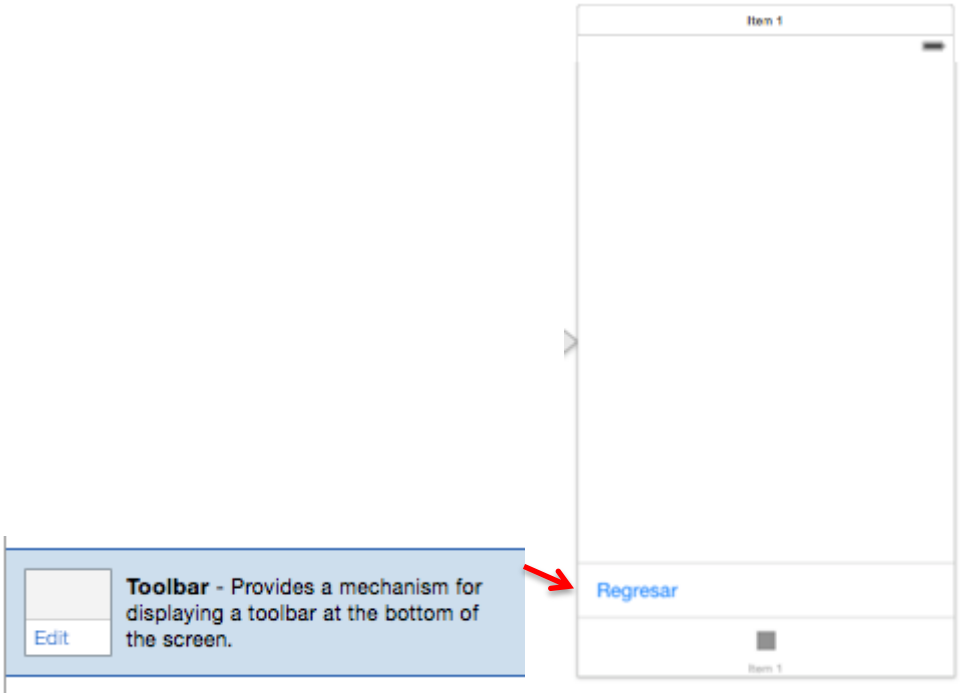


Arrastrar este elemento al archivo Main.storyboard, sin especificar algún ViewControler, lo cual generara una nueva pantalla que se observara de la siguiente manera.



De esta una pantalla de la cual se derivan otras dos pantallas, estas correspondientes cada una a una pestaña en el TabBarController.

Agregar un ToolBar en las pantallas que se derivan del TabBarController (item1 e item2).
Cambiarle el nombre a "Regresar" y hacer la misma funcionalidad como los anteriores ViewController.



Ahora realizamos el vínculo de los archivos .h y .m de "Tab1ViewController" y "Tab2ViewController" correspondiente a las pantallas agregadas.

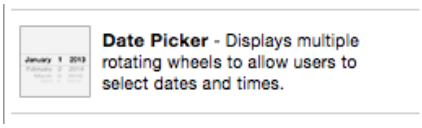
Omitiendo la pantalla de la cual derivan las otras 2(la de la izquierda), esto se realiza de igual manera con los ViewControllers trabajados al principio de la guía.[1]

Vincular el botón sobrante del ViewController de inicio con la pantalla llamada "TabBarController".

[1] Vincular Vista con controlador (clases) pag.10 de esta guía.

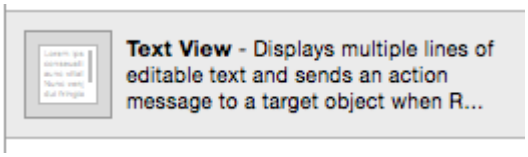
A cada una de las vistas se puede modificar el nombre de estas, seleccionando la imagen inferior de cada ViewController y seleccionando el texto el cual se habilitara para edición, cambiar los nombres de estas pestañas a: Vista1 y Vista2 respectivamente.

En el Tab1ViewController agregar un label y agregaremos un elemento llamado “DatePicker”, este elemento es un PickerView de varios componentes el cual genera alguna fecha determinada con su hora determinada, presentando así información de fecha y hora.

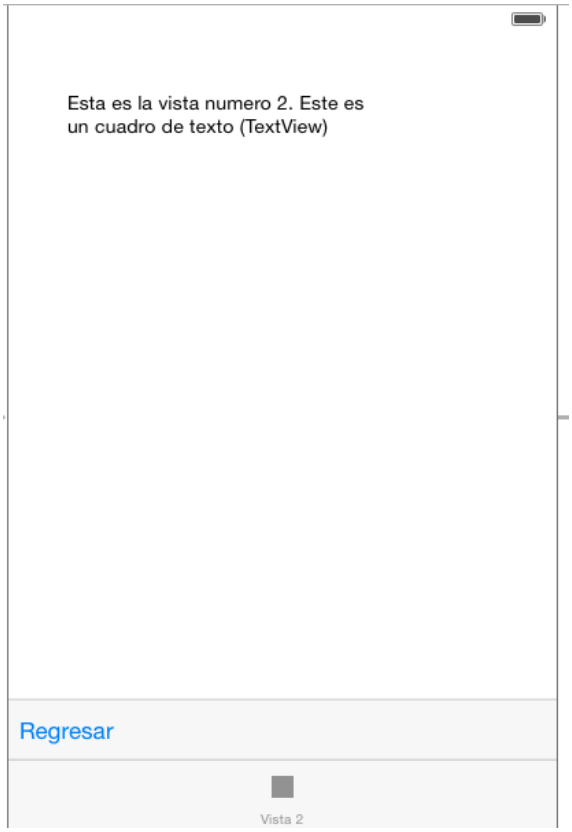


Modificar el label y el DatePicker para que se observe de la siguiente manera.

Para el "Tab2ViewController" se agregara un elemento de tipo TextView, el cual trabaja como un cuadro de texto, el cual puede ser modificado tanto por medio de código como por medio gráfico como se trabajó con los labels previamente en la guía.

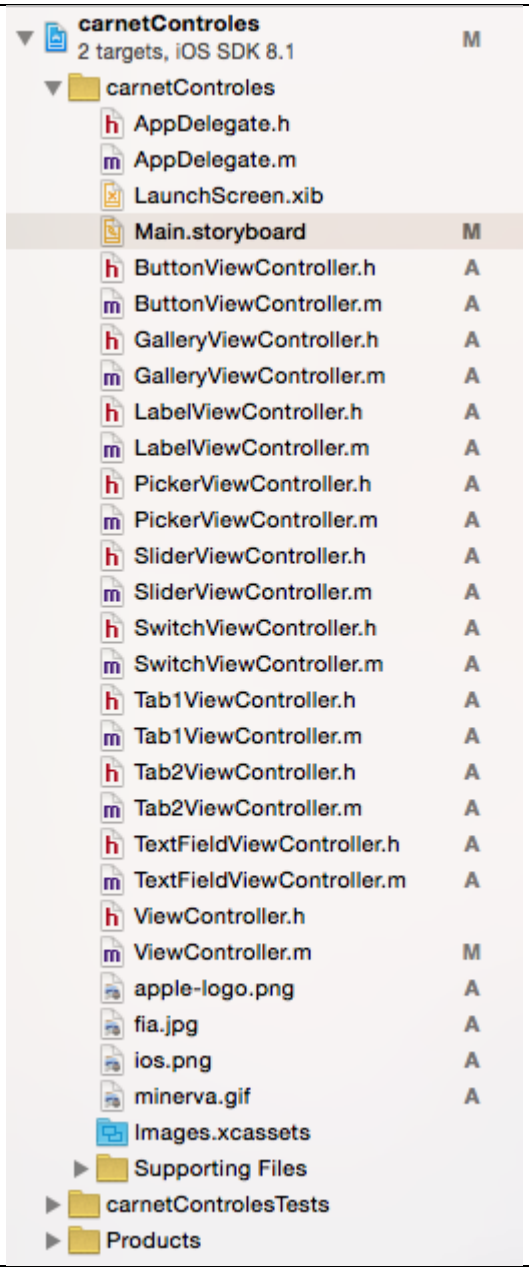


Al agregar el TextView, modificar el texto para que se observe como a continuación.



Teniendo todas las vistas modificadas el archivo Main.storyboard y la barra de navegación se observaran como lo siguiente. Probar toda la funcionalidad de la aplicación.

Listado de Objetos finales en el Proyecto



StoryBoard Final