Transiciones y storyboards

Índice

Como hemos comentado, normalmente tendremos un controlador por cada pantalla de la aplicación. Sin embargo, esto no quiere decir que en un momento dado sólo pueda haber un controlador activo, sino que podremos tener controladores dentro de otros controladores. Es muy frecuente encontrar un controlador que se encarga de la navegación (se encarga de mostrar una barra de navegación con el título de la pantalla actual y un botón para volver a la pantalla anterior), que contiene otro controlador que se encarga de gestionar la pantalla por la que estamos navegando actualmente. También tenemos otro tipo de controlador contenedor que se encarga de mostrar las diferentes pantallas de nuestra aplicación en forma de pestañas (*tabs*).

Otra forma de contención se da cuando mostramos una vista modal. Utilizaremos nuestro controlador para mostrar la vista modal, que a su vez estará gestionada por otro controlador.

1. Controladores modales

Los controladores modales son la forma más sencilla de contención. Cuando mostramos un controlador de forma modal, su contenido reemplazará al del contenedor actual. Para hacer que un controlador muestre de forma modal otro controlador, llamaremos a su método `presentModalViewController:animated:`

```
- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    UAAsignatura *asignatura =
            [asignaturas objectAtIndex: indexPath.row];

    DetallesViewController *controladorModal =
        [[DetallesViewController alloc]
             initWithAsignatura: asignatura];

    [self presentModalViewController: controladorModal
                            animated: YES];

    [controladorModal release];
}
```

En el ejemplo anterior tenemos un controlador de una tabla, en el que al seleccionar un elemento muestra un controlador de forma modal con los detalles del elemento seleccionado. Al presentar el controlador modal, éste se almacena en la propiedad `modalViewController` del controlador padre (la tabla en nuestro ejemplo). Por otro lado, en el controlador modal (detalles en el ejemplo) habrá una propiedad `parentViewController` que hará referencia al controlador padre (tabla en el ejemplo). El controlador modal es retenido automáticamente por el controlador padre cuando lo presentamos.

Nota

En el iPhone/iPod touch el controlador modal siempre ocupará toda la pantalla. Sin embargo, en el iPad podemos hacer que esto no sea así. Podemos cambiar esto con la propiedad `modalPresentationStyle` del controlador modal.

Hemos de remarcar que un controlador modal no tiene porque ser un controlador secundario. Se puede utilizar este mecanismo para hacer transiciones a pantallas que pueden ser tan importantes como la del controlador padre, y su contenido puede ser de gran complejidad. Cualquier controlador puede presentar otro controlador de forma modal, incluso podemos presentar controladores modales desde otros controladores modales, creando así una cadena.

Cuando queramos cerrar la vista modal, deberemos llamar al método `dismissModalViewControllerAnimated:` del padre. Sin embargo, si llamamos a este método desde el controlador modal también funcionará, ya que hará un *forward* del mensaje a su padre (propiedad `parentViewController`).

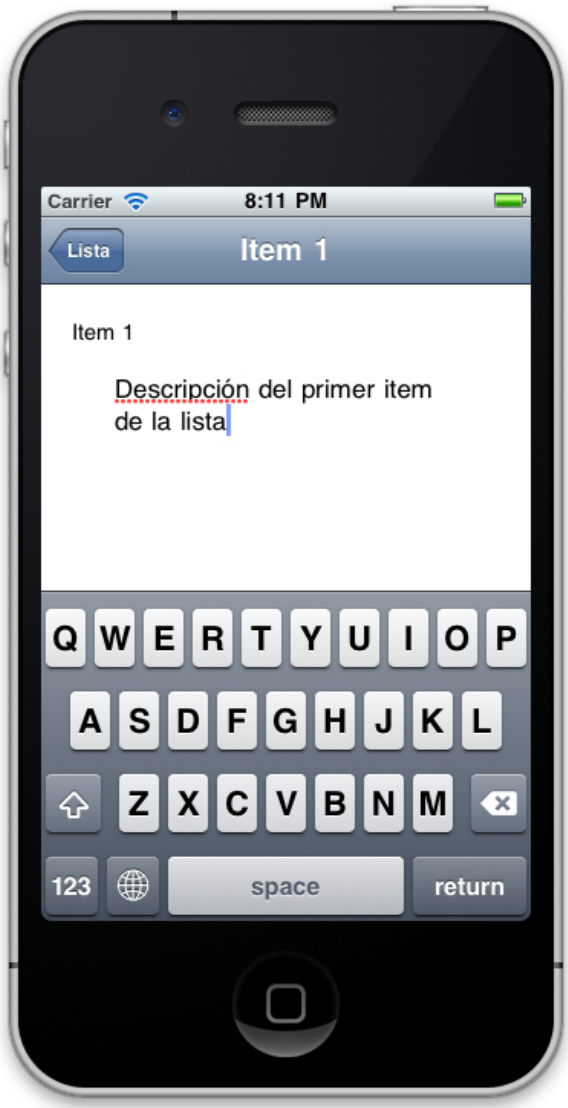
La llamada a `dismissModalViewControllerAnimated:` funcionará de la siguiente forma según sobre qué controlador la llamemos:

- Controlador con `modalViewController==nil` (el controlador modal que se está mostrando actualmente): Se redirije la llamada a su propiedad `parentViewController`.
- Controlador con `modalViewController!=nil` (controlador que ha presentado un controlador modal hijo): Cierra su controlador modal hijo, y todos los descendientes que este último tenga. Cuando se cierra con una única operación toda la cadena de controladores modales, veremos una única transición de la vista modal que se estuviese mostrando a la vista padre que ha recibido el mensaje para cerrar la vista modal.

2. Controlador de navegación

El tipo de aplicación más común que encontramos en iOS es el de las aplicaciones basadas en navegación. En ellas al pasar a una nueva pantalla, ésta se apila sobre la actual, creando así una pila de pantallas por las que hemos pasado. Tenenemos una barra de navegación en la parte superior, con el título de la pantalla actual y un botón que nos permite volver a la pantalla anterior.

Pantalla raíz de la navegación

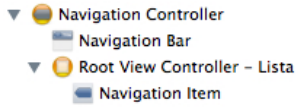


Pantalla de detalles en la pila de navegación

Para conseguir este funcionamiento, contamos con la clase `UINavigationController` que ya implementa este tipo de navegación y los elementos anteriores. Dentro de este controlador tenemos una vista central que es donde se mostrará el contenido de cada pantalla por la que estemos navegando. Cada una de estas pantallas se implementará mediante un controlador.

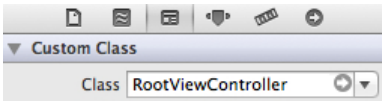
No será necesario crear ninguna subclase de `UINavigationController`, ya que en ella contamos ya con todos los elementos necesarios para implementar este esquema de navegación, pero si que tendremos que tener en cuenta que estamos utilizando este controlador contenedor en los controladores que implementemos para cada pantalla.

Cuando arrastramos un objeto de tipo `UINavigationController` a nuestro fichero NIB, veremos que crear bajo él una estructura con los siguientes elementos:



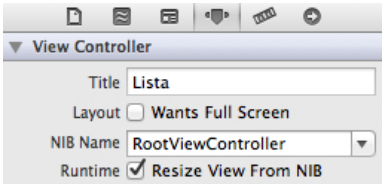
Elementos de Navigation Controller

- *Navigation Bar*: Define el fondo de la barra de navegación en la parte superior de la pantalla.
- *Navigation Item*: Define el contenido que se mostrará en la barra de navegación.
- *Root View Controller*: Controlador raíz que se utilizará para mostrar el contenido de la vista central. Este controlador podrá ser de un tipo propio en el que definamos el contenido de la pantalla raíz de la navegación. Para definir el tipo del controlador utilizaremos la propiedad *Class* del inspector de identidad.



Inspector de identidad del controlador raíz

Si queremos que cargue dicho controlador con un NIB distinto al NIB por defecto, podemos especificar el nombre del NIB en el inspector de atributos.



Inspector de atributos del controlador raíz

Atención

Si nuestro controlador (*Root View Controller* en el caso anterior) se crea dentro del NIB, nunca se llamará a su inicializador designado, ya que los objetos del NIB son objetos ya construidos que cargamos en la aplicación. Por lo tanto, si queremos realizar alguna inicialización, deberemos utilizar dentro de él el método `awakeFromNib`.

De forma alternativa, también podemos crear nuestro `UINavigationController` de forma programática. Para ello en primer lugar crearemos el controlador raíz, y después inicializaremos el controlador de navegación proporcionando en su constructor dicho controlador raíz:

```
RootViewController *rootViewController =
    [[RootViewController alloc] initWithNibName:@"RootViewController"
                                         bundle:nil];
UINavigationController *navController =
    [[UINavigationController alloc]
                      initWithRootViewController:rootViewController];
```

Vamos a centrarnos en ver cómo implementar estos controladores propios que mostraremos dentro de la navegación.

En primer lugar, `UIViewController` tiene una propiedad `title` que el controlador de navegación utilizará para mostrar el título de la pantalla actual en la barra de navegación. Cuando utilicemos este esquema de navegación deberemos siempre asignar un título a nuestros controladores. Cuando apilemos una nueva pantalla, el título de la pantalla anterior se mostrará como texto del botón de volver atrás.

Además, cuando un controlador se muestre dentro de un controlador de navegación, podremos acceder a dicho controlador contenedor de navegación mediante la propiedad `navigationController` de nuestro controlador.

Nota

Podemos personalizar con mayor detalle los elementos mostrados en la barra de navegación mediante la propiedad `navigationItem` de nuestro controlador. Esta propiedad hace referencia al objeto que veíamos como *Navigation Item* en Interface Builder. Los elementos que podremos añadir a dicha barra serán de tipo *Bar Button Item* (`UIBarButtonItem`). En Interface Builder podemos añadirlos como hijos de *Navigation Item* y conectarlos con *outlets* de dicho elemento.

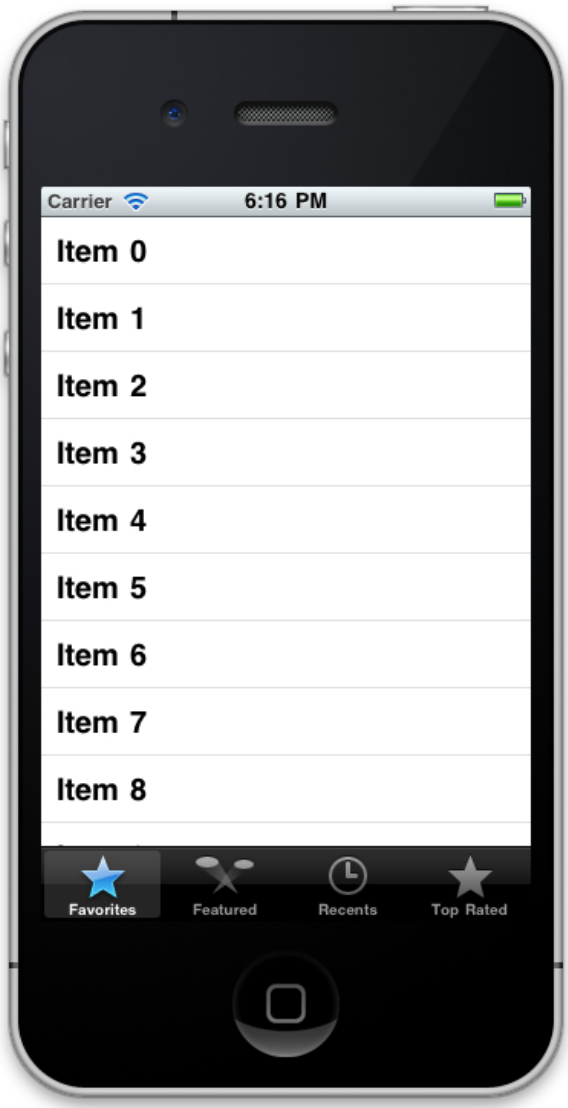
Cuando queramos pasar a la siguiente pantalla dentro de un controlador de navegación, utilizaremos el método `pushViewController:animated:` del controlador de navegación:

```
- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    DetailViewController *detailViewController =
        [[DetailViewController alloc]
            initWithNibName:@"DetailViewController"
                    bundle:nil];
    NSString *titulo = [NSString
        stringWithFormat: @"Item %d", indexPath.row];
    detailViewController.title = titulo;

    [self.navigationController
        pushViewController:detailViewController animated:YES];

    [detailViewController release];
}
```

Tras esta llamada se apila, se retiene, y se muestra la siguiente pantalla, que pasa a ser la cima de la pila. En la nueva pantalla veremos un botón para volver atrás en la barra de navegación. Al pulsarlo se desapilará la pantalla y se librerará la referencia. También podemos desapilar las pantallas de forma programática con `popViewControllerAnimated:`, que desapilará la pantalla de la cima de la pila. De forma alternativa, podemos utilizar `popToRootViewControllerAnimated:` o `popToViewController:animated:` para desapilar todas las pantallas hasta llegar a la raíz, o bien hasta llegar a la pantalla especificada, respectivamente.

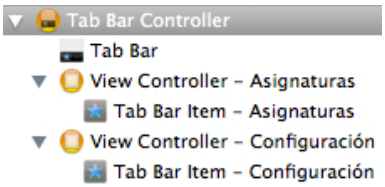
3. Controlador de barra de pestañas

Otro tipo de controlador contenedor predefinido que podemos encontrar es el controlador que nos permite organizar las pantallas de nuestra aplicación en forma de pestañas (*tabs*) en la parte inferior de la pantalla:



Aspecto de un Tab Bar Controller

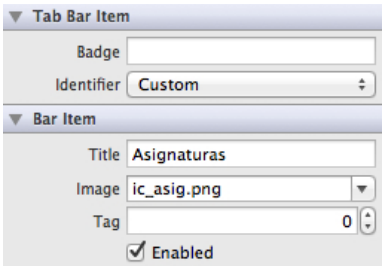
Dicho controlador se implementa en `UITabBarController`, y al igual que en el caso anterior, tampoco deberemos crear subclases de él. Cuando añadamos un elemento de tipo *Tab Bar Controller* a nuestro NIB se creará una estructura como la siguiente:



Elementos de Tab Bar Controller

- *Tab Bar*: Representa la barra de pestañas que aparecerá en la parte inferior de la pantalla, y que contendrá una serie de elementos.
- *Tab Bar Item*: Representa cada uno de los elementos de la barra de pestañas. Habrá un *item* por cada pestaña que tengamos en la barra. Aquí configuraremos el título y el icono que se mostrará en la pestaña.
- *View Controller*: Tendremos tantos controladores como pestañas en la barra. Podemos añadir nuevos controladores como hijos de *Tab Bar Controller* para añadir nuevas pestañas. Cada controlador tendrá asociado un *Tab Bar Item* con la configuración de su correspondiente pestaña.

Para configurar el aspecto de cada pestaña en la barra, seleccionaremos en el *dock* el correspondiente *Tab Bar Item* y accederemos a su inspector de atributos.



Inspector de atributos de Tab Bar Item

Como vemos, podemos o bien utilizar un aspecto predefinido (propiedad *Identifier*), o bien poner un título (*Title*) e imagen (*Image*) propios.

Iconos de las pestañas

Como iconos para las pestañas deberemos utilizar imágenes PNG de 30 x 30 px con transparencia. Los colores de la imagen (RGB) se ignorarán, y para dibujar el icono sólo se tendrá en cuenta la capa *alpha* (transparencia).

Podremos utilizar para cada pestaña cualquier tipo de controlador, que definirá el contenido a mostrar en dicha pestaña. Para especificar el tipo utilizaremos la propiedad *Class* de dicho controlador en el inspector de identidad (al igual que en el caso del controlador de navegación). Incluso podríamos poner como controlador para una pestaña un controlador de navegación, de forma que cada pestaña podría contener su propia pila de navegación.

Nota

Si abrimos un controlador modal desde una pestaña o desde una pantalla de navegación, perderemos las pestañas y la barra de navegación.

Podemos crear también este controlador de forma programática. En este caso, tras instanciar el controlador de pestañas, tendremos que asignar a su propiedad `viewController` un *array* de los controladores a utilizar como pestañas. Tendrá tantas pestañas como elementos tenga la lista proporcionada:

```
PrimerViewController *controller1 = [[PrimerViewController alloc]
                initWithNibName:@"PrimerViewController" bundle:nil];
SegundoViewController *controller2 = [[SegundoViewController alloc]
                initWithNibName:@"SegundoViewController" bundle:nil];
TercerViewController *controller3 = [[TercerViewController alloc]
                initWithNibName:@"TercerViewController" bundle:nil];

UITabBarController *tabBarController =
                        [[UITabBarController alloc] init];
tabBarController.viewControllers = [NSArray arrayWithObjects:
                        controller1, controller2, controller3, nil];
```

El título de cada pestaña y el icono de la misma se configurarán en cada uno de los controladores incluidos. Por ejemplo, en `PrimerViewController` podríamos tener:

```
- (id)initWithNibName:(NSString *)nibNameOrNil
                bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil
                           bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
        self.title = @"Asignaturas";
        self.tabBarItem.image = [UIImage imageNamed:@"icono_asig"];
    }
    return self;
}
```

4. Uso de storyboards

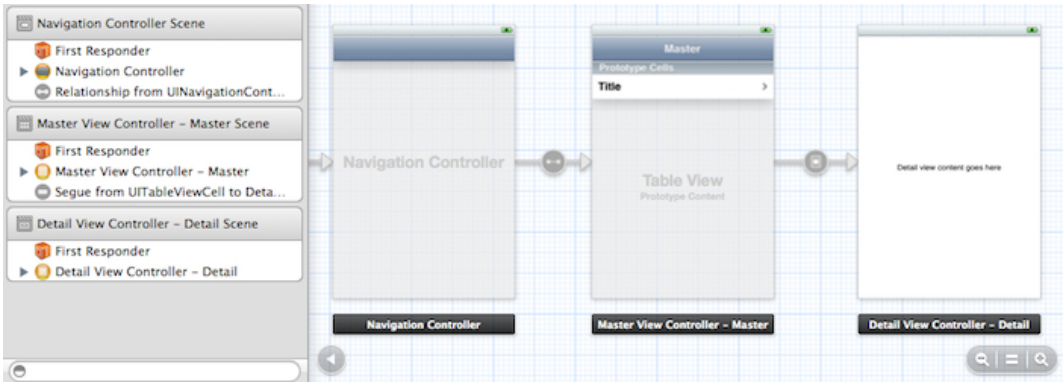
Los *storyboards* son una característica aparecida en iOS 5.0 y Xcode 4.2, que nos dará una forma alternativa para crear la interfaz de nuestras aplicaciones. Nos permitirán crear en un único fichero de forma visual las diferentes pantallas de la aplicación y las transiciones entre ellas. Están pensados principalmente para definir el flujo de trabajo de aplicaciones basadas en navegación o en pestañas.

Advertencia

Los *storyboards* no son compatibles con versiones de iOS previas a la 5.0.

Se definen en ficheros con extensión `.storyboard`, y podemos crearlos desde Xcode con la opción *New File ... > User Interface > Storyboard*. Por convenio, el *storyboard* principal de la aplicación se llamará `MainStoryboard.storyboard`.

Para que la aplicación arranque con el contenido del *storyboard*, deberemos especificar dicho fichero en la propiedad *Main Storyboard* del proyecto. Además, en `main.m` deberemos especificar como cuarto parámetro de `UIApplicationMain` la clase delegada de la aplicación, y en dicha clase delegada deberemos tener una propiedad de tipo `UIWindow` llamada `window`.



Editor del storyboard

Sobre el *storyboard* podremos arrastrar diferentes tipos de controladores. Cada uno de estos controladores aparecerán como una pantalla de la aplicación. Para cada controlador deberemos especificar su tipo en el atributo *Class* del inspector de identidad (podrán ser tipos definidos en Cocoa Touch, como `UINavigationController`, o tipos propios derivados de `UIViewController`). En las pantallas del *storyboard* no existe el concepto de *File's Owner*, sino que la relación de las pantallas con el código de nuestra aplicación se establecerá al especificar el tipo de cada una de ellas.

4.1. Segues

Las relaciones entre las distintas pantallas se definen mediante los denominados *segues*. Podemos ver los *segues* disponibles en el inspector de conexiones. Encontramos diferentes tipos de *segues*:

- *Relationship*: Define una relación entre dos controladores, no una transición. Nos sirve para relacionar un controlador contenedor con el controlador contenido en él.
- *Modal*: Establece una transición modal a otra pantalla (controlador).
- *Push*: Establece una transición de navegación a otra pantalla, que podrá utilizarse cuando la pantalla principal esté relacionada con un controlador de tipo `UINavigationController`.
- *Custom*: Permite definir una relación que podremos personalizar en el código.

Por ejemplo, si en una pantalla tenemos una lista con varias celdas, podemos conectar mediante un *segue* una de las celdas a un controlador correspondiente a otra pantalla. De esa forma, cuando el usuario pulse sobre dicha celda, se hará una transición a la pantalla con la que conecta el *segue*. Normalmente el origen de un *segue* será un botón o la celda

de una tabla, y el destino será un controlador (es decir, la pantalla a la que vamos a hacer la transición).

En el primer controlador que arrastremos sobre el *storyboard* veremos que aparece una flecha entrante. Este es un *segue* que no tienen ningún nodo de origen, y que indica que es la primera pantalla del *storyboard*.

Si al arrancar la aplicación queremos tener acceso a dicho controlador raíz, por ejemplo para proporcionale datos con los que inicializarse, podemos acceder a él a través de la propiedad `rootViewController` de la ventana principal:

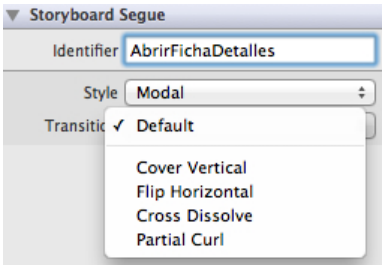
```
- (BOOL)application:(UIApplication *)application
     didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    UIViewController *controlador =
        (UIViewController *)self.window.rootViewController;
    ...
    return YES;
}
```

Cuando se ejecute un *segue*, se avisará al método `prepareForSegue:sender:` del controlador donde se originó. Sobrescribiendo este método podremos saber cuándo se produce un determinado *segue*, y en ese caso realizar las acciones oportunas, como por ejemplo configurar los datos de la pantalla destino.

```
-(void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
  NSIndexPath *indexPath = [self.tableView indexPathForSelectedRow];
  [segue.destinationViewController setDetailItem:
          [NSString stringWithFormat:@"Item %d", indexPath.row]];
}
```

También podemos lanzar de forma programática un *segue* con el método `performSegueWithIdentifier:sender:`. Cada *segue* tendrá asociado un identificador, que se le asignará en la propiedad *Identifier* de la sección *Storyboard Segue* de su inspector de atributos.

En el caso de tener un *segue* de tipo Modal, podremos especificar en su propiedad *Transition* el estilo de la transición a la siguiente pantalla.



Atributos de los segues

Si queremos crear una transición personalizada, podemos crear una subclase de `UIStoryboardSegue`, establecer el tipo de *segue* a *Custom* (propiedad *Style*), y

especificar la clase en la que hemos implementado nuestro propio tipo de *segue* en la propiedad *Segue Class*.

4.2. Tablas en el storyboard

Con el uso de los *storyboards* la gestión de las celdas de las tablas se simplifica notablemente. Cuando introduzcamos una vista de tipo tabla, en su inspector de atributos (sección *Table View*), veremos una propiedad *Content* que puede tomar dos posibles valores:

- *Dynamic Prototypes*: Se utiliza para tablas dinámicas, en las que tenemos un número variable de filas basadas en una serie de prototipos. Podemos crear de forma visual uno o más prototipos para las celdas de la tabla, de forma que no será necesario configurar el aspecto de las celdas de forma programática, ni cargarlas manualmente.
- *Static Cells*: Se utiliza para tablas estáticas, en las que siempre se muestra el mismo número de filas. Por ejemplo, nos puede servir para crear una ficha en la que se muestren los datos de una persona (nombre, apellidos, dni, telefono y email).

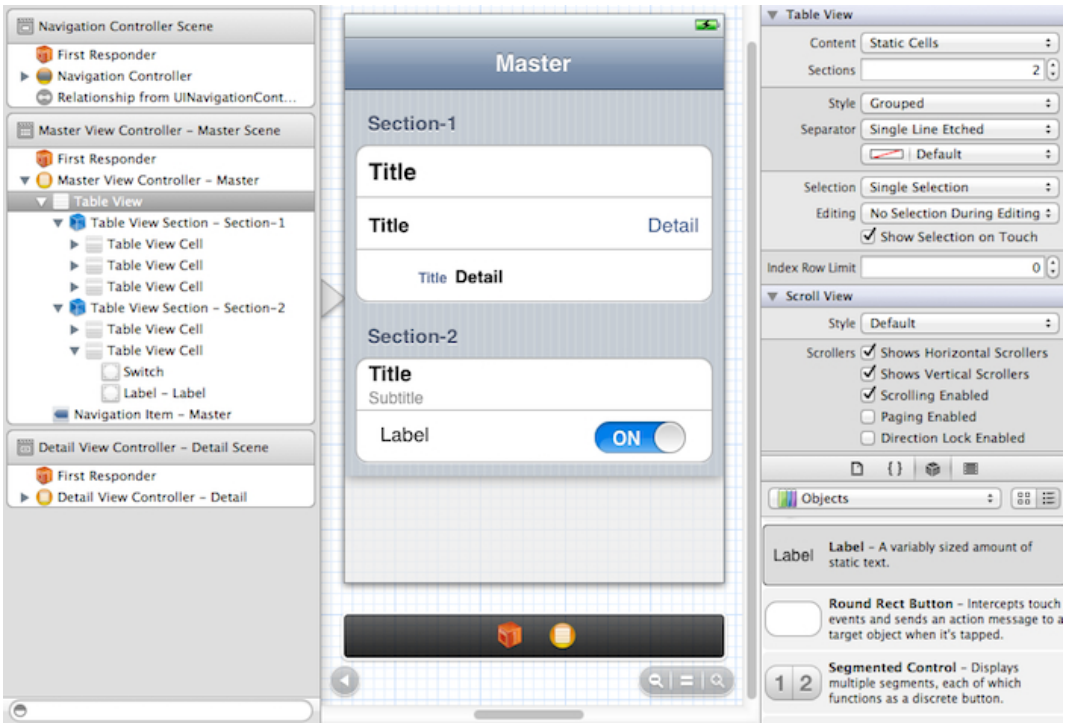


Tabla estática en el storyboard

En el caso de las tablas dinámicas, si definimos más de un prototipo, podemos dar a cada uno de ellos un identificador de reutilización diferente, para así en el código poder seleccionar el tipo deseado (esto lo especificaremos en el inspector de atributos de la celda prototipo, sección *Table View Cell*, propiedad *Identifier*).

Además, en el código se nos asegura que el método `dequeueReusableCellWithIdentifier:` siempre nos va a dar una instancia de una celda correspondiente al prototipo cuyo identificador se especifique como parámetro. Por lo tanto, ya no será necesario comprobar si nos ha devuelto `nil` y en tal caso instanciarla nosotros, como hacíamos anteriormente, sino que podremos confiar en que siempre nos devolverá una instancia válida.

```
UITableViewCell *cell = [tableView
       dequeueReusableCellWithIdentifier:CellIdentifier];

cell.textLabel.text = @"Item";
```

Las tablas estáticas podrán crearse íntegramente en el *storyboard*. Cuando creemos una tabla de este tipo podremos especificar el número de secciones (propiedad *Sections* en *Table View*), y en el *dock* aparecerá un nodo para cada sección, que nos permitirá cambiar sus propiedades. Dentro de cada sección podremos también configurar el número de filas que tiene (propiedad *Rows* en *Table View Section*), y veremos cada una de estas filas de forma visual en el editor, pudiendo editarlas directamente en este entorno.

Para cada celda podemos optar por utilizar un estilo predefinido, o bien personalizar su contenido (*Custom*). Esto lo podemos configurar en la propiedad *Style* de *Table View Cell*. En el caso de optar por celdas personalizadas, deberemos arrastrar sobre ellas las vistas que queramos que muestren.

También podremos conectar las celdas mediante *segues*, de forma que cuando se pulse sobre ellas se produzca una transición a otra pantalla.