Componentes para iPad y aplicaciones universales

Índice

En este módulo comentaremos distintos componentes de diseño de interfaz de usuario tanto para sistemas iOS como para Android. Comenzaremos explicando el funcionamiento de los controladores específicos de iPad para continuar detallando la manera de cómo podemos programar aplicaciones universales (compatibles tanto para iPhone como para iPad). En la segunda sesión nos centraremos en comentar las distintas guias de estilo que recomienda Apple para el diseño de aplicaciones iOS, algo que deberemos cumplir en la medida de lo posible en el proceso de diseño de la interfaz de usuario.

1. Componentes específicos para iPad

Debido a las diferencias evidentes de tamaño entre un iPhone y un iPad, *Apple* ha desarrollado una API específica para este último, de esta forma se pretende que aquellas aplicaciones que estén disponible en iPad utilicen esta api en la medida de lo posible ya que así se mejorará considerablemente la experiencia de usuario y la usabilidad a la hora de usarlas.

Con la salida del iPad al mercado aparecieron un par de nuevos controladores enfocados exclusivamente a este. El uso de ambos esta unido muy a menudo debido a que se complementan mutuamente, estos son los controladores:

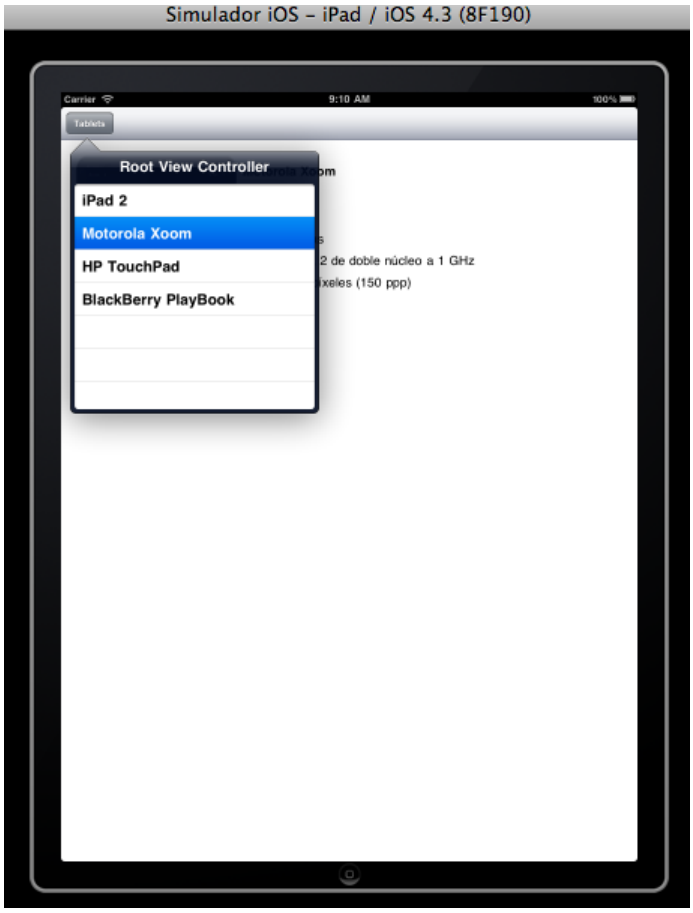
- **Split View**: Formado por dos vistas independientes, una situada en la parte izquierda y la otra en la parte derecha. Divide la pantalla principal en dos y se utiliza muy frecuentemente para mostrar en la vista de la izquierda el listado de opciones de un menu en forma de tabla y en la parte de la derecha el detalle de la vista seleccionada.
- **Popover**: Es una especie de ventana emergente que puede aparecer en cualquier parte de la pantalla, normalmente junto a un botón al pulsarlo. Muy útil para mostrar un listado de opciones o una tabla con información dentro de un *Split View*. *Apple* recomienda usar un popover para mostrar el listado de opciones de la tabla de la parte de la izquierda de un *Split View* cuando este está en posición vertical (*portrait*).

1.1. Split View

Un **Split View** o *Vista divida* es, como hemos comentado anteriormente, una combinación de dos vistas, la primera de las cuales es equivalente a una vista "tipo" de iPhone cuando este se encuentra en orientación vertical, usando incluso la misma anchura. Esta vista se utilizará principalmente para la navegación principal dentro de la aplicación. Por otro lado en la segunda vista, que corresponderá a la porción más grande de la pantalla, mostraremos la información que hayamos seleccionado desde la vista de la izquierda en detalle. Comentar que este es el uso que *Apple* propone para este tipo de controlador aunque puede tener muchos más.

Split View

En modo vertical (*Portrait Orientation*), el *Split View* cambia y la vista de la izquierda (el menú) pasa ahora a ser una vista de tipo *popover* que aparecerá, cuando se pulsa un botón a modo de cuadro de diálogo desde la parte superior izquierda de la pantalla. Esta transformación hace falta programarla, aunque es bastante sencillo. De esta forma quedaría la vista de la derecha ocupando toda la pantalla del iPad.



Split View vertical

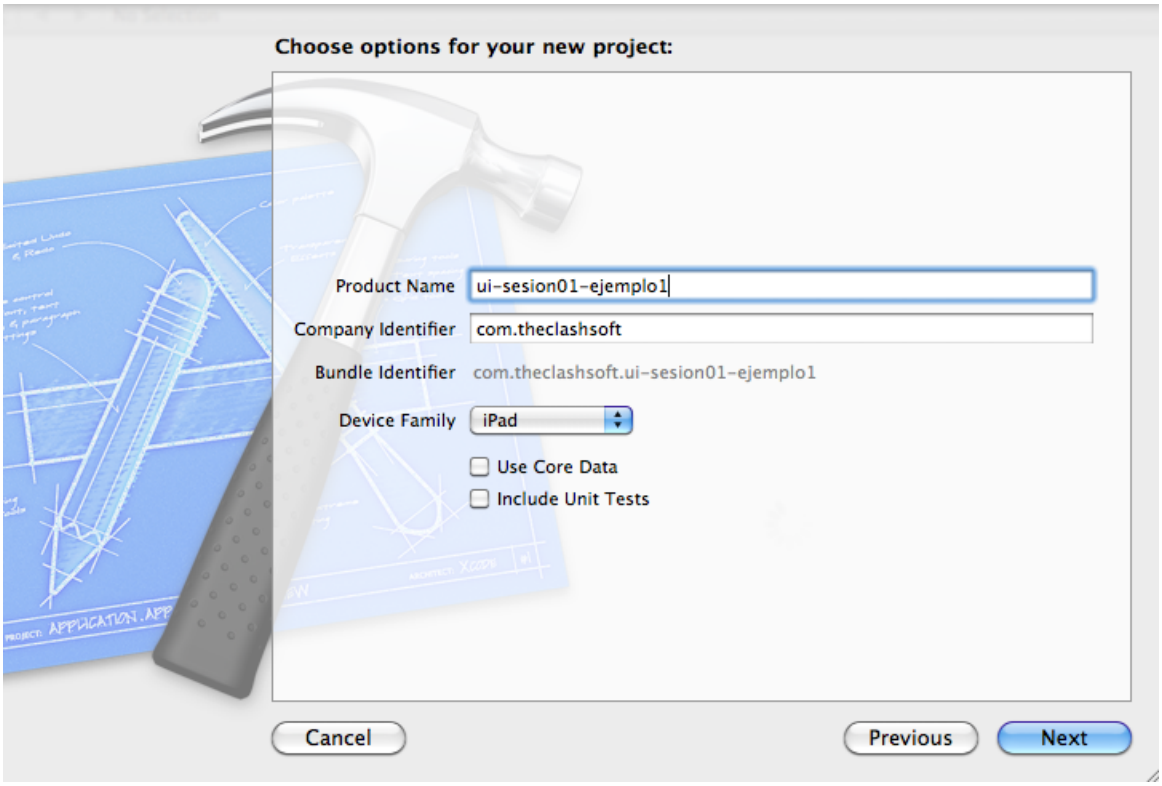
Incorporar un controlador *Split View* en nuestra aplicación es bastante sencillo aunque puede existir alguna confusión a la hora de implementarlo, es por ello que vamos a explicarlo con un sencillo ejemplo el cual lo usaremos más adelante como punto de partida para nuestras aplicaciones. En la aplicación de ejemplo realizaremos una comparativa simple de los distintos tablets que existen en el mercado: tendremos un listado en forma de tabla de todos los tablets en la parte izquierda del *Split View* y cuando seleccionemos uno nos aparecerán sus detalles en la parte derecha. Para finalizar el ejemplo, cuando tengamos todo funcionando, implementaremos un *popover* para que cuando rotemos el iPad a posición vertical, podamos navegar por las distintas opciones.

1.1.1. Creando el proyecto

A pesar de que XCode nos da la opción de crear un proyecto de tipo `UISplitView` en el que tendremos un controlador *Split View* desde el inicio, nosotros vamos a crearlo desde cero, ya que para futuras aplicaciones necesitaremos saber cómo incorporar un *Split View* en cualquier vista.

Plantilla Master Detail Application

A partir de la versión 4.2 de XCode y coincidiendo con la aparición de iOS 5, XCode renovó todas las plantillas que disponia para crear nuevos proyectos iOS, entre las que añadió se encuentra una que se llama `Master Detail Application`. Al seleccionar esta plantilla a la hora de crear un nuevo proyecto, XCode nos creará la estructura básica para una aplicación Universal (por defecto) usando la controladora `UISplitView Controller` como base.



Crea proyecto

Por lo tanto, empezamos creando un proyecto usando la plantilla `Single View Application`, seleccionamos en la lista de la familia de dispositivos *iPad*, deseleccionamos `Use Storyboard` y `Use Automatic Reference Counting` y hacemos click en *Next*. Guardamos el proyecto con el nombre *ui-sesion01-ejemplo1*. De esta forma tenemos ya la estructura básica de un proyecto diseñado para iPad, si lo ejecutamos nos aparecerá el simulador de iPad con la pantalla en blanco.

1.1.2. Creando las clases de las vistas

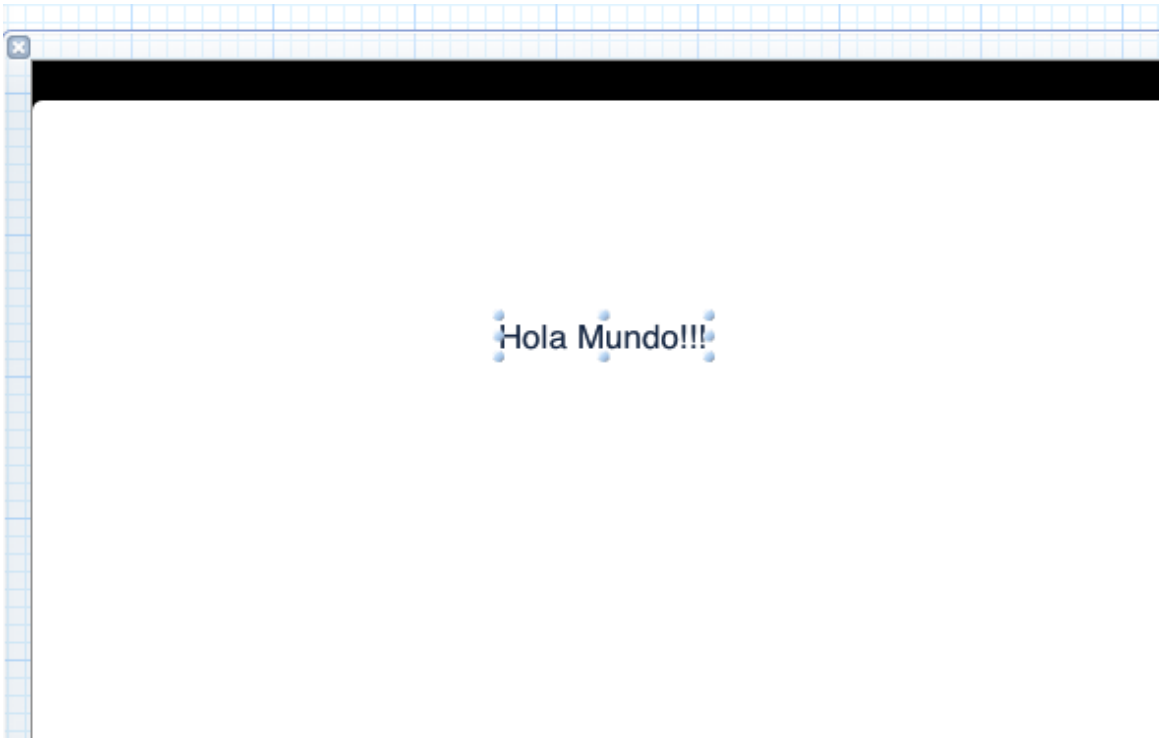
Ahora vamos a crear las dos vistas del *Split View* como nosotros queremos: por un lado crearemos una vista de tipo `Table View Controller` que será la de la parte izquierda y por otro lado una de tipo `View Controller` para la parte derecha del *Split View*.

Comenzamos creando el `Table View Controller`, para ello hacemos click en *File* > *New File* y seleccionamos *Cocoa Touch* > *UIViewController Subclass* y seleccionamos en el siguiente paso "Subclass off" *UITableViewController* y nos aseguramos de tener marcada la opción de *Targeted for iPad*. Lo guardamos con el nombre de *MenuViewController*.

Ahora abrimos el fichero *MenuViewController.m* y modificamos algún dato de la clase: en el método `numberOfSectionsInTableView` devolvemos `1` y en el método `numberOfRowsInSection` devolvemos `5`. De esta forma tendremos una tabla con cinco filas vacias, esto lo hacemos para hacer una prueba rápida, más adelante meteremos datos útiles.

Ahora pasamos a crear la vista vacia `UIViewController`, para ello hacemos click en *New* > *New File* y seleccionamos `UIViewController Subclass`. Después seleccionamos "Subclass of" *UIViewController* y nos aseguramos de marcar la opción de *Targeted for iPad*. Guardamos con el nombre de `DetalleViewController` y esta vista será la que corresponda a la parte derecha del *Split View*.

Para hacer una prueba rápida, vamos a poner algo dentro de la vista de detalle, para ello abrimos la vista `DetalleViewControler.xib`, arrastramos un *label* y escribimos por ejemplo `Hola Mundo!`. Con esto ya tenemos las dos clases creadas y listas para asignarlas a nuestro *Split View*.



Vista Split

1.1.3. Asignando las vistas

Una vez que tenemos creadas la vista *Master* y la vista de detalle vamos a crear un objeto de tipo `UISplitViewController` y a asignarle ambas vistas. Esto lo haremos dentro de la clase `didFinishLaunchingWithOptions` del fichero `ui_sesion01_ejemplo1AppDelegate.m`:

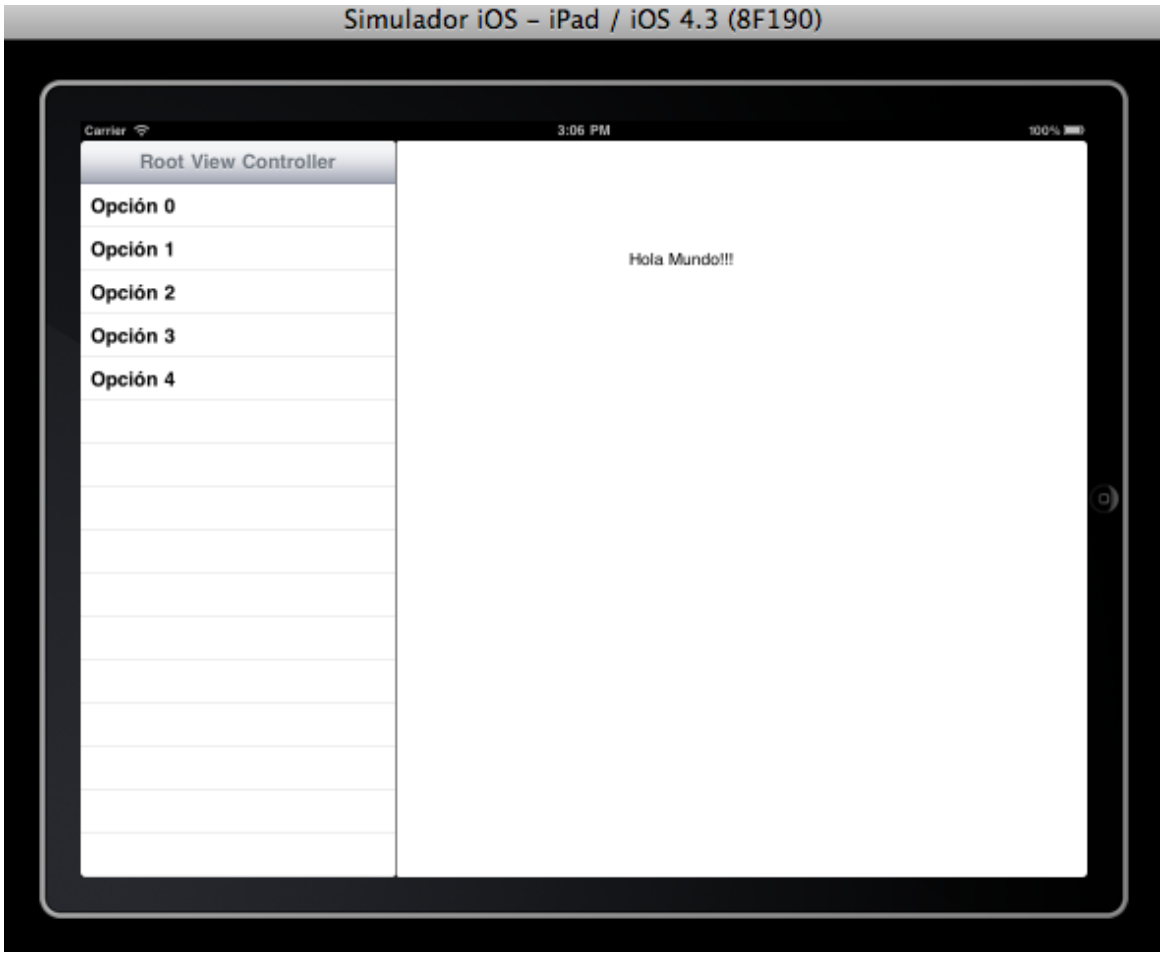
`ui_sesion01_ejemplo1AppDelegate.m`

```
        #import "MenuViewController.h"
#import "DetalleViewController.h"

        @synthesize window=_window;

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions
{

    self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen]
bounds]] autorelease];
    // Override point for customization after application launch.


    MenuViewController *listado = [[MenuViewController alloc]
initWithNibName:@"MenuViewController" bundle:nil];
    DetalleViewController *detalle = [[DetalleViewController alloc]
initWithNibName:@"DetalleViewController" bundle:nil];

    UISplitViewController *splitViewController = [[UISplitViewController
alloc] init];
    [splitViewController setViewControllers:[NSArray
arrayWithObjects:listado,detalle, nil]];


    self.window.rootViewController = splitViewController;

    [self.window makeKeyAndVisible];
    return YES;
}
```

Nota

La programación de un `Split View Controller` se puede realizar de muchas formas. Esta que hemos explicado es la más sencilla y más clara. Otra manera de crearlo sería mediante las vistas. Añadiendo a la vista principal de la aplicación un objeto de tipo `Split View Controller` desde el Interface Builder.

Ahora ya podemos compilar y ejecutar el proyecto y ver los resultados. Si giramos el simulador podremos observar como se adapta el *Split View* de forma automática.



Split View iPad

1.1.4. Programando el modelo de datos

El siguiente paso es programar nuestro modelo de datos. Esto se podría hacer utilizando algún método de persistencia como *ficheros, SQLite* o *Core Data*, pero por simplificar no utilizaremos ninguno y simplemente crearemos los objetos en memoria directamente.

Vamos a programar la clase que representará cada tablet que queremos mostrar, para ello hacemos click en *New* > *New File* y seleccionamos `Objective-C class`, subclass of `NSObject`. Guardamos el fichero con el nombre `Tablet`. Ahora abrimos cada uno de los archivos creados y escribimos el siguiente código:

`Tablet.h`

```
        #import <Foundation/Foundation.h>

@interface Tablet : NSObject {
```

```
    NSString *_nombre;
    NSString *_procesador;
    NSString *_grafica;
    NSString *_memoria;
    NSString *_pantalla;
    NSString *_resolucion;
    NSString *_imagen;
}

@property (nonatomic, copy) NSString *nombre;
@property (nonatomic, copy) NSString *procesador;
@property (nonatomic, copy) NSString *grafica;
@property (nonatomic, copy) NSString *memoria;
@property (nonatomic, copy) NSString *pantalla;
@property (nonatomic, copy) NSString *resolucion;
@property (nonatomic, copy) NSString *imagen;

- (Tablet *)initWithNombre:(NSString *)nombre
                procesador:(NSString *)procesador
                   grafica:(NSString *)grafica
                   memoria:(NSString *)memoria
                  pantalla:(NSString *)pantalla
                resolucion:(NSString *)resolucion
                    imagen:(NSString *)imagen;


@end
```

Tablet.m

```
        #import "Tablet.h"


@implementation Tablet

@synthesize nombre = _nombre;
@synthesize procesador = _procesador;
@synthesize grafica = _grafica;
@synthesize memoria = _memoria;
@synthesize pantalla = _pantalla;
@synthesize resolucion = _resolucion;
@synthesize imagen = _imagen;

- (Tablet *)initWithNombre:(NSString *)nombre
                procesador:(NSString *)procesador
                   grafica:(NSString *)grafica
                   memoria:(NSString *)memoria
                  pantalla:(NSString *)pantalla
                resolucion:(NSString *)resolucion
                    imagen:(NSString *)imagen
{
    if ((self = [super init])) {
        self.nombre = nombre;
        self.procesador = procesador;
        self.grafica = grafica;
        self.memoria = memoria;
        self.pantalla = pantalla;
        self.resolucion = resolucion;
        self.imagen = imagen;

    }
    return self;
```

```
}

-(void) dealloc {
    self.nombre = nil;
    self.procesador = nil;
    self.grafica = nil;
    self.memoria = nil;
    self.pantalla = nil;
    self.resolucion = nil;
    self.imagen = nil;

    [super dealloc];

}

@end
```

Con este código ya tenemos el modelo de datos definido. Ahora vamos a implementar la parte izquierda del *Split View*.

1.1.5. Programando la vista detalle

La parte izquierda del *Split View* es una tabla en donde aparecerán los nombres de los distintos modelos de tablets, para implementar esto añadiremos una nueva variable a la clase `MenuViewController` que será un array con los objetos `Tablet`. Abrimos `MenuViewController.h` y escribimos lo siguiente:

```
        #import <UIKit/UIKit.h>


@interface MenuViewController : UITableViewController {
    NSMutableArray *_tablets;
}

@property (nonatomic, retain) NSMutableArray *tablets;

@end
```

Ahora añadimos las siguientes líneas en el archivo *MenuViewController.m*:

```
        // Justo debajo del #import
#import "Tablet.h"

// Debajo de @implementation
@synthesize tablets = _tablets;

// En numberOfRowsInSection, cambia el 5 con lo siguiente:
return [_tablets count];

// En cellForRowAtIndexPath, después de "Configure the cell..."
Tablet *tablet = [_tablets objectAtIndex:indexPath.row];
cell.textLabel.text = tablet.name;

// En dealloc
self.tablets = nil;
```

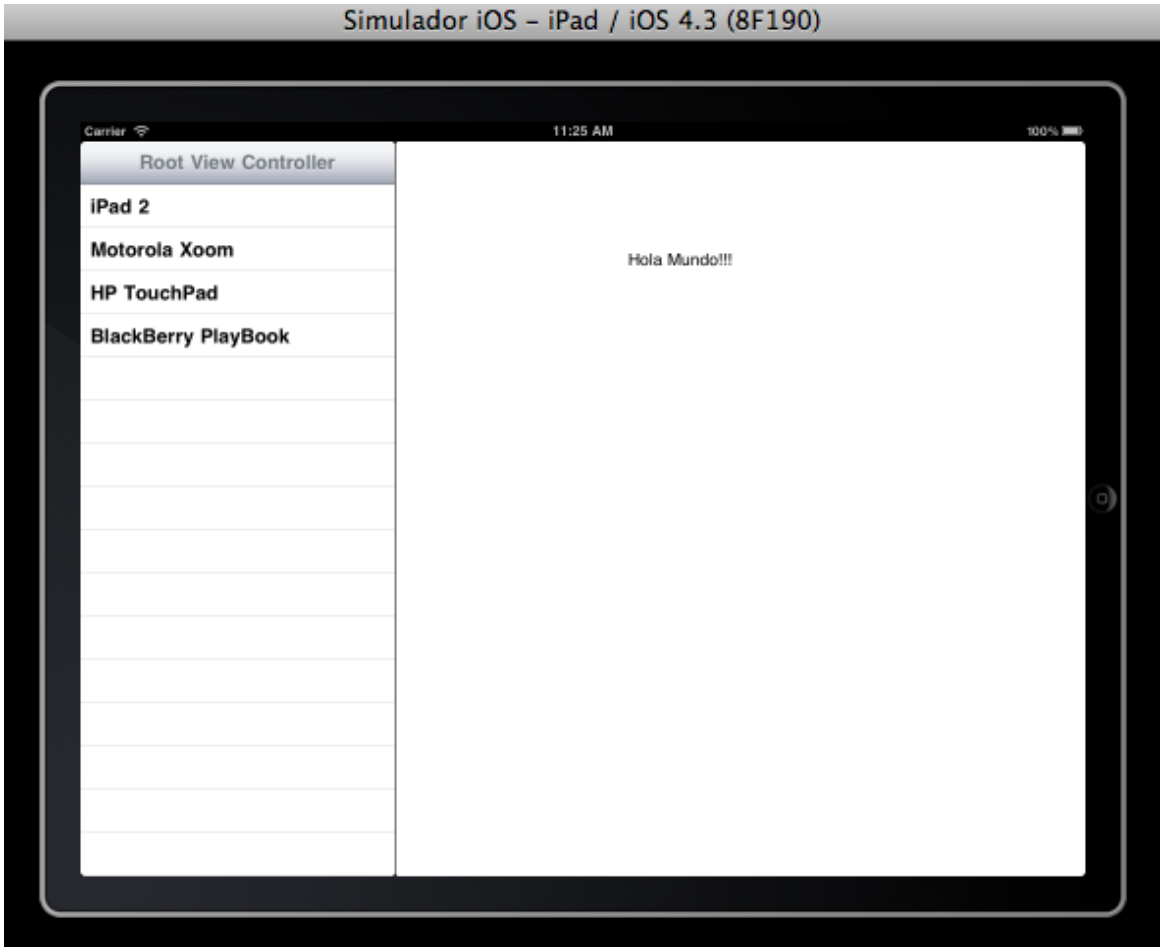
Con esto ya tenemos el *Table View* terminado. Ahora vamos a crear los objetos tablets en la clase delegada, para ello abrimos `uisesion01_ejemplo1AppDelegate.m` y escribimos el siguiente código dentro del método `didFinishLaunchingWithOptions` justo antes de `[window addSubview:_splitViewController.view];`

```
        Tablet *ipad2 = [[Tablet alloc] initWithNombre:@"iPad 2"
                                        procesador:@"Apple A5 de doble
núcleo a 1 GHz"
                                           grafica:@"PowerVR SGX543MP2"
                                           memoria:@"512 MB"
                                         pantalla:@"9,7 pulgadas"
                                        resolucion:@"1024 × 768 píxeles
(132 ppp)"
                                            imagen:@"ipad2.jpg"];

    Tablet *motorola = [[Tablet alloc] initWithNombre:@"Motorola Xoom"
                                        procesador:@"Nvidia Tegra 2 de
doble núcleo a
                                        1 GHz"
                                           grafica:@"GeForce GPU 333"
                                           memoria:@"1 GB"
                                         pantalla:@"10,1 pulgadas"
                                        resolucion:@"1280 × 800 píxeles
(150 ppp)"
                                            imagen:@"motorola.jpg"];

    Tablet *hp = [[Tablet alloc] initWithNombre:@"HP TouchPad"
                                        procesador:@"Qualcomm Snapdragon
de doble núcleo
                                        a 1,2 GHz"
                                           grafica:@"Adreno 220"
                                           memoria:@"1 GB"
                                         pantalla:@"9,7 pulgadas"
                                        resolucion:@"1024 × 768 píxeles
(132 ppp)"
                                            imagen:@"hp.jpg"];

    Tablet *blackBerry = [[Tablet alloc] initWithNombre:@"BlackBerry
PlayBook"
                                        procesador:@"Procesador de doble
núcleo a 1 GHz"
                                           grafica:@"Desconocido"
                                           memoria:@"1 GB"
                                         pantalla:@"7 pulgadas"
                                        resolucion:@"1024 × 600 píxeles"
                                            imagen:@"blackberry.jpg"];

    NSMutableArray *listaTablets = [NSMutableArray array];
    [listaTablets addObject:ipad2];
    [listaTablets addObject:motorola];
    [listaTablets addObject:hp];
    [listaTablets addObject:blackBerry];

    _menuViewController.tablets = listaTablets;
```

Imágenes

Las imágenes de las tablets las podemos descargar desde aquí. Una vez descargadas, las descomprimimos y tenemos arrastrarlas al directorio de `Supporting Files`.

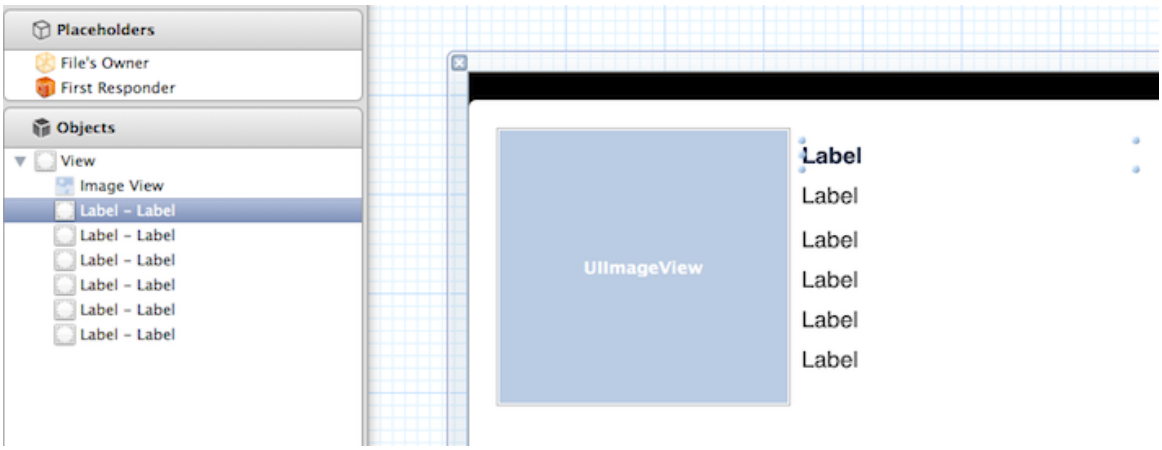
Compilamos ahora y si todo ha ido bien nos debe de aparecer lo siguiente:



Split View iPad 2

1.1.6. Programando la vista derecha o principal

Una vez programado el menu del *Split View*, vamos a diseñar y programar la parte derecha, o sea, la vista de detalle. Abrimos el fichero `DetalleViewController.xib`, borramos el `Label` creado anteriormente y diseñamos la vista de la manera que se muestra en la siguiente imagen:



Estructura vista detalle Split View

Como podemos ver en la imagen anterior, hemos añadido los siguientes componentes a la vista de detalle:

- UIImageView > `imgTablet`
- UILabel > `labelNombre`
- UILabel > `labelProcesador`
- UILabel > `labelGrafica`
- UILabel > `labelMemoria`
- UILabel > `labelPantalla`
- UILabel > `labelResolucion`

Ahora abrimos el fichero `DetalleViewController.h` y añadimos los "Outlets" para poder referenciarlos desde la vista que hemos creado:

```
        #import <UIKit/UIKit.h>
#import "Tablet.h"

@interface DetalleViewController : UIViewController {
    Tablet *_tablet;
    UIImageView *_imgTablet;
    UILabel *_labelNombre;
    UILabel *_labelProcesador;
    UILabel *_labelGrafica;
    UILabel *_labelMemoria;
    UILabel *_labelPantalla;
    UILabel *_labelResolucion;
}

@property (nonatomic, retain) Tablet *tablet;
@property (nonatomic, retain) IBOutlet UIImageView *imgTablet;
@property (nonatomic, retain) IBOutlet UILabel *labelNombre;
@property (nonatomic, retain) IBOutlet UILabel *labelProcesador;
@property (nonatomic, retain) IBOutlet UILabel *labelGrafica;
@property (nonatomic, retain) IBOutlet UILabel *labelMemoria;
@property (nonatomic, retain) IBOutlet UILabel *labelPantalla;
@property (nonatomic, retain) IBOutlet UILabel *labelResolucion;
```

```
@end
```

En `DetalleViewController.m` escribimos lo siguiente debajo de `@implementation DetalleViewController`

```
        @synthesize tablet = _tablet;
@synthesize imgTablet = _imgTablet;
@synthesize labelNombre = _labelNombre;
@synthesize labelGrafica = _labelGrafica;
@synthesize labelMemoria = _labelMemoria;
@synthesize labelPantalla = _labelPantalla;
@synthesize labelProcesador = _labelProcesador;
@synthesize labelResolucion = _labelResolucion;
```

Ahora abrimos la vista `DetalleViewController.xib` y enlazamos todos los "Outlets" a su objeto correspondiente. Con esto ya tenemos la vista de detalle creada, ahora sólo nos falta enlazarla con la vista de la izquierda, que es lo que vamos a hacer a continuación.

1.1.7. Enlazando las dos vistas

Existen muchas formas de programar la correspondencia entre la vista de la izquierda con la de la derecha, una de ellas es la que se propone en la plantilla de *Split View* de XCode, en el que se incluye la vista de detalle en la vista del menu, y desde esta se accede a las propiedades de la vista detalle. Este método no es muy recomendado usarlo ya que dificulta enormemente la reutilización de las vistas en otros proyectos o en este mismo.

Nosotros vamos a utilizar el método recomendado por *Apple* que se explica en este link, usando un objeto delegado que recibe una petición desde la clase principal cuando se selecciona una fila en la tabla del menu de la izquierda.

La idea principal es que vamos a definir un protocolo con un sólo método que llamaremos *selectedTabletChanged*. El lado derecho se encargará de implementar dicho método y el lado izquierdo será el que realize la llamada mediante una clase delegada.

Para implementar esto debemos de crear una nueva clase *New > New File* seleccionando *Subclass of NSObject*. Guardamos la clase con el nombre `TabletSelectionDelegate`. Borramos el fichero *.m* ya que no lo necesitaremos y escribimos lo siguiente en `TabletSelectionDelegate.m`:

```
        #import <Foundation/Foundation.h>

@class Tablet;

@protocol TabletSelectionDelegate

- (void)tabletSelectionChanged:(Tablet *)curSelection;

@end
```

Ahora modificamos el archivo `MenuViewController.h` para incluir `TabletSelectionDelegate`:

```
        #import <UIKit/UIKit.h>
#import "DetalleViewController.h"
#import "TabletSelectionDelegate.h"

@interface MenuViewController : UITableViewController {
    NSMutableArray *_tablets;
    DetalleViewController *_detalleViewController;
    id<TabletSelectionDelegate> *_delegate;
}

@property (nonatomic, retain) NSMutableArray *tablets;
@property (nonatomic, retain) DetalleViewController
*detalleViewController;
@property (nonatomic, assign) id<TabletSelectionDelegate> *delegate;

@end
```

Y ahora cambiamos también `MenuViewController.m`:

```
        // Debajo de @implementation
@synthesize delegate = _delegate;

// Dentro del metodo didSelectRowAtIndexPath:
if (_delegate != nil) {
        Tablet *tablet = (Tablet *) [_tablets
objectAtIndex:indexPath.row];
        [_delegate tabletSelectionChanged:tablet];
    }

// Dentro de dealloc
self.delegate = nil;
```

Ya queda menos, ahora modificaremos la clase del detalle `DetalleViewController.h` añadiendo el protocolo:

```
        #import "TabletSelectionDelegate.h"

@interface DetalleViewController : UIViewController
<TabletSelectionDelegate> {
```

Editamos el método `viewDidLoad` en el que asignaremos los textos de las etiquetas y la imágen y añadimos la implementación del método del protocolo dentro de `DetalleViewController.m`:

```
        - (void)viewDidLoad
{

    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.

    if (self.tablet != nil){
```

```
        self.labelNombre.text = self.tablet.nombre;
        self.labelGrafica.text = self.tablet.grafica;
        self.labelMemoria.text = self.tablet.memoria;
        self.labelPantalla.text = self.tablet.pantalla;
        self.labelProcesador.text = self.tablet.procesador;
        self.labelResolucion.text = self.tablet.resolucion;

        self.imgTablet.image = [UIImage imageNamed:self.tablet.imagen];
    }
}

        - (void)tabletSelectionChanged:(Tablet *)curSelection {
    self.tablet = curSelection;
    [self viewDidLoad];
}
```

Finalmente tenemos que asignar el `delegate` que acabamos de crear dentro del método `didFinishLaunchingWithOptions` la clase delegada `uisesion01_ejemplo1AppDelegate.m`:

```
            _menuViewController.delegate = _detalleViewController;
```

Si está todo correcto, al compilar y ejecutar el proyecto debe de funcionar perfectamente el *Split View*:

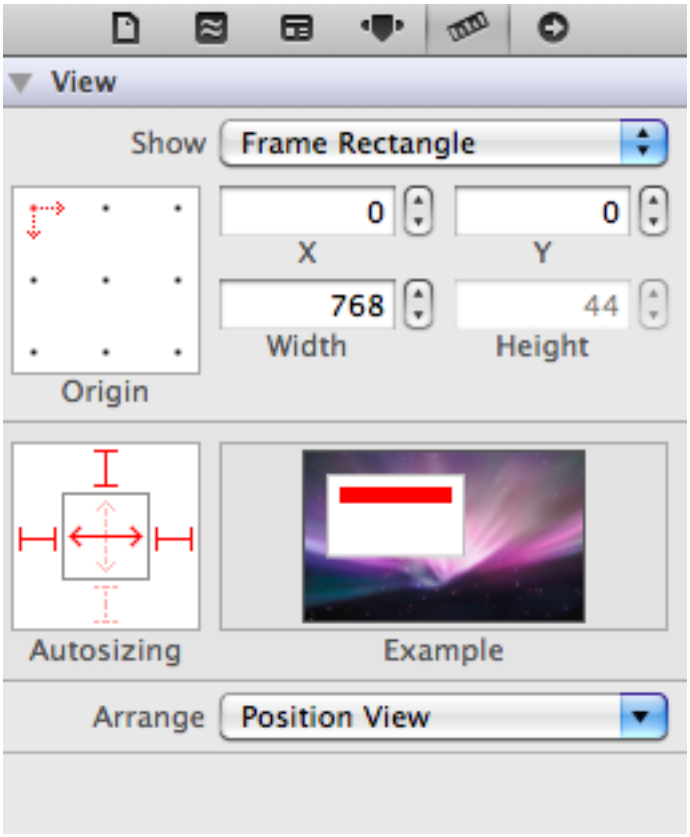
Split View terminado

Si giramos ahora el simulador (o el iPad) veremos que desaparece el menu de la izquierda y no podemos acceder a el, para evitar esto debemos de hacer uso de un elemento nuevo y exclusivo del iPad: *popover*. Aunque hablaremos con más detalle de el en el siguiente apartado, ahora debemos utilizarlo para mostrar una ventana emergente que contenga el menu de la izquierda. La programación de esta parte es muy sencilla y siempre es la misma, ¡vamos a implementarlo!

1.1.8. Añadiendo un Popover al Split View

La solución que propone *Apple* al problema comentando anteriormente es añadir un *Toolbar* con un *Button* en la parte derecha del *Split View* y al pulsar el botón que aparezca un *Popover* con el menu correspondiente a la parte izquierda. Para implementar esto debemos primero añadir el `Toolbar` dentro de la vista de la derecha, para ello abrimos `DetalleViewController.xib` y arrastramos el componente `UIToolbar` a la vista. Para que aparezca en la parte superior de la vista debemos de configurarlo de la siguiente

forma:



ToolBar size

Ahora hacemos los siguientes cambios a *DetalleViewController.h*:

```
       // Añadimos UISplitViewControllerDelegate a la lista de protocolos
de la interfaz
@interface DetalleViewController : UIViewController
<TabletSelectionDelegate,
UISplitViewControllerDelegate> {

// Dentro de la definición de la clase
UIPopoverController *_popover;
UIToolbar *_toolbar;

// Añadimos estas dos propiedades nuevas
@property (nonatomic, retain) UIPopoverController *popover;
@property (nonatomic, retain) IBOutlet UIToolbar *toolbar;
```

También debemos de añadir los siguientes métodos a `DetalleViewController.m`:

```
      - (void)splitViewController: (UISplitViewController*)svc
    willHideViewController:(UIViewController *)aViewController
         withBarButtonItem:(UIBarButtonItem*)barButtonItem
       forPopoverController: (UIPopoverController*)pc
```

```
{
    barButtonItem.title = @"Tablets";
    NSMutableArray *items = [[_toolbar items] mutableCopy];
    [items insertObject:barButtonItem atIndex:0];
    [_toolbar setItems:items animated:YES];
    [items release];
    self.popover = pc;
}

- (void)splitViewController: (UISplitViewController*)svc
     willShowViewController:(UIViewController *)aViewController
  invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem
{
    NSMutableArray *items = [[_toolbar items] mutableCopy];
    [items removeObjectAtIndex:0];
    [_toolbar setItems:items animated:YES];
    [items release];
    self.popover = nil;
}
```

```
        // Después de @implementation
@synthesize popover = _popover;
@synthesize toolbar = _toolbar;

// En deallloc
self.popover = nil;
self.toolbar = nil;

// En tabletSelectionChanged
if (_popover != nil) {
    [_popover dismissPopoverAnimated:YES];
}
```

Volvemos a abrir la vista `DetalleViewController.xib` y asignamos el "Outlet" al `ToolBar` que hemos añadido anteriormente. Por último nos falta asignar desde la vista principal el *Delegate* del *Split View Controller* al *Detalle View Controller*, para hacer esto abrimos la vista `MainWindow.xib`, seleccionamos desde la lista de objetos el *Split View Controller* y relacionamos el Outlet "delegate" con el objeto *Detalle View Controller*, debe de quedar de la siguiente manera:

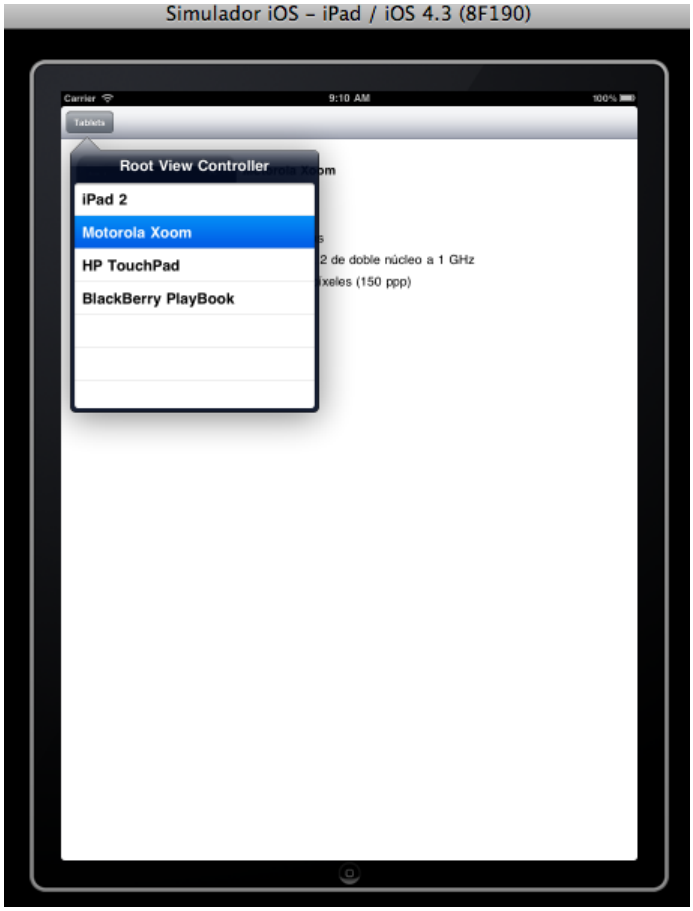
Vista general outlets

Si ejecutamos ahora la aplicación veremos que al girar el dispositivo y situarlo en

posición vertical aparecerá en la barra superior un botón que si lo pulsamos nos debe salir un *popover* con el listado de tablets. Como podemos ver, el tamaño del *popover* es muy alto y sobra casi todo, esto lo podemos evitar introduciendo el siguiente fragmento de código en el método `viewDidLoad` de la clase `MenuViewController`:

```
        - (void)viewDidLoad {
    [super viewDidLoad];
    self.clearsSelectionOnViewWillAppear = NO;
    self.contentSizeForViewInPopover = CGSizeMake(320.0, 300.0);
}
```

Volvemos a ejecutar la aplicación y comprobamos ahora que el tamaño del *popover* se ha reducido considerablemente.



Popover en iPad

1.2. Popovers

1.2.1. Introducción

Como hemos podido ver en el apartado anterior, un *popover* es una ventana secundaria (estilo *pupup*) que suele aparecer encima de la ventana principal de la interfaz. Se puede programar para que desaparezca automáticamente cuando el usuario toca cualquier parte de la pantalla o para que no desaparezca. El tamaño de la ventana *popover* puede cambiar también y por supuesto puede contener todo lo que se desee, por ejemplo: un listado de opciones de configuración para nuestra aplicación, un álbum de fotos de la cual el usuario deberá seleccionar una, un formulario simple para que el usuario introduzca un texto, etc. En un iPhone, un *popover* puede corresponderse perfectamente con cualquier vista que ocupe la pantalla completa.

Con el fin de aclarar un poco más la definición de *popover* y ver cómo podemos utilizarlo en nuestras aplicaciones vamos a realizar un pequeño y sencillo ejemplo de uso partiendo del ejemplo del apartado anterior. Este ejemplo consistirá en mostrar un *popover* al pulsar un botón dentro de la vista de detalle. Este *popover* tendrá un pequeño listado de opciones que al seleccionarlas se mostrarán dentro de la vista de detalle. ¡Comenzamos!

1.2.2. Creando la clase delegada

Primero tenemos que crear la clase controladora que heredará de `UITableView Controller`, esto lo hacemos pulsando en *New > New File*, seleccionamos `UIViewController Subclass`, `subclass of UITableViewController`. Importante marcar la opción de *Targeted for iPad* y desmarcar *With XIB for user interface*. Guardamos el fichero con el nombre *SeleccionaOpcionController*.

Ahora abrimos *SeleccionaOpcionController.h* y escribimos lo siguiente:

```
        #import <UIKit/UIKit.h>

@protocol SeleccionaOpcionDelegate

- (void)opcionSeleccionada:(NSString *)nombreOpcion;

@end



@interface SeleccionaOpcionController : UITableViewController {
    NSMutableArray *_opciones;
    id<SeleccionaOpcionDelegate> _delegate;
}

@property (nonatomic, retain) NSMutableArray *opciones;
@property (nonatomic, assign) id<SeleccionaOpcionDelegate> delegate;

@end
```

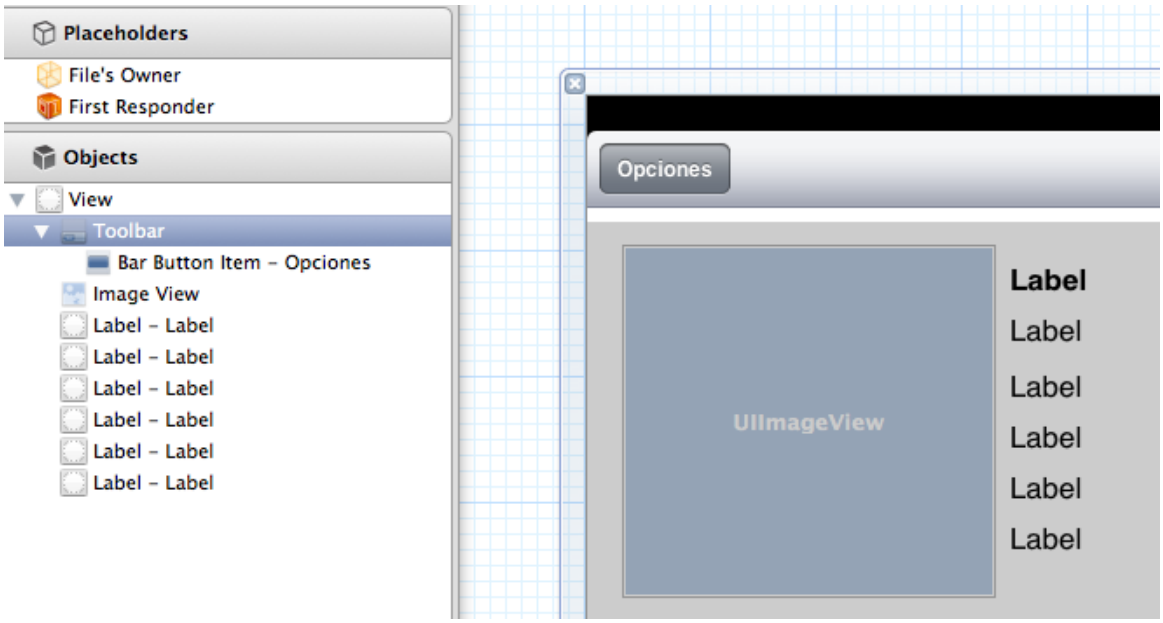
En el fragmento de código anterior creamos un sólo método delegado que se encargará de notificar a otra clase cuando un usuario seleccione una opción del listado. A continuación

abrimos `SeleccionaOpcionController.m` y hacemos los siguientes cambios:

```
        // Debajo de @implementation
@synthesize opciones = _opciones;
@synthesize delegate = _delegate;

// Método viewDidLoad:
- (void)viewDidLoad {
    [super viewDidLoad];
    self.clearsSelectionOnViewWillAppear = NO;
    self.contentSizeForViewInPopover = CGSizeMake(150.0, 140.0);
    self.opciones = [NSMutableArray array];
    [_opciones addObject:@"Opción 1"];
    [_opciones addObject:@"Opción 2"];
    [_opciones addObject:@"Opción 3"];
}

// en dealloc:
self.opciones = nil;
self.delegate = nil;

// en numberOfSectionsInTableView:
return 1;

// en numberOfRowsInSection:
return [_opciones count];

// en cellForRowAtIndexPath:
NSString *opcion = [_opciones objectAtIndex:indexPath.row];
cell.textLabel.text = opcion;

// en didSelectRowAtIndexPath:
if (_delegate != nil) {
    NSString *opcion = [_opciones objectAtIndex:indexPath.row];
    [_delegate opcionSeleccionada:opcion];
}
```

1.2.3. Mostrando el popover

Una vez que hemos programado la clase delegada encargada de gestionar la selección de la opción sólo nos queda mostrar el *popover* dentro de la vista detalle de nuestra aplicación. Para ello abrimos la vista `DetalleViewController.xib`, añadimos un botón (*Bar Button Item*) al *toolbar* y le ponemos como título del botón: "Opciones". También añadimos un nuevo *Label* que será el que muestre la opción que seleccionemos de la lista.



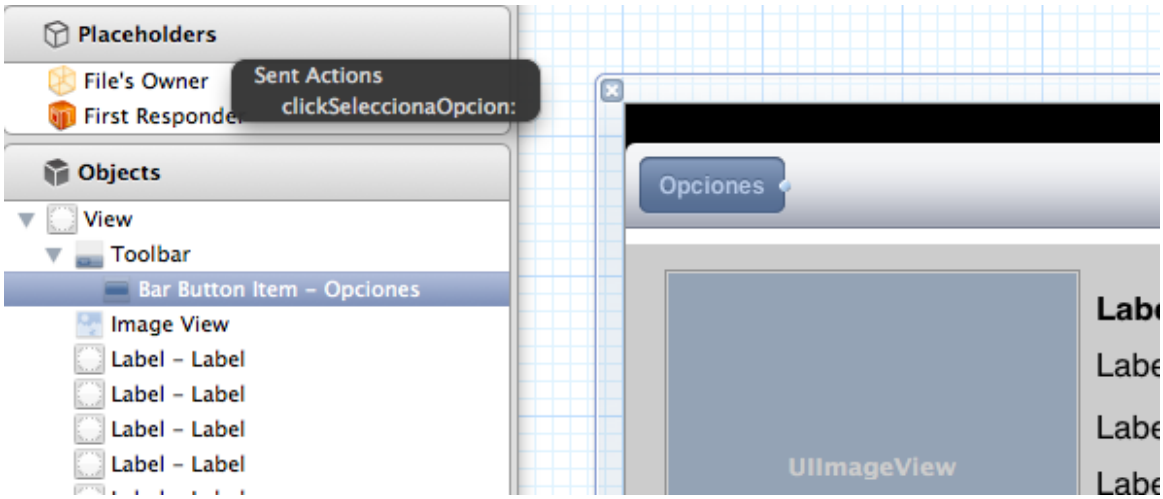
Vista detalle

Ahora tenemos que programar el método que se llamará cuando se pulse sobre el botón y un par de variables necesarias:

`DetalleViewController.h`

```
        // debajo de los #import
#import "SeleccionaOpcionController.h"

// Añadimos el Delegate de selección de opción a la definición
@interface RightViewController : UIViewController
<TabletSelectionDelegate,
    UISplitViewControllerDelegate, SeleccionaOpcionDelegate> {

// Dentro de la clase añadimos estas dos variables
SeleccionaOpcionController *_seleccionaOpcion;
    UIPopoverController *_seleccionaOpcionPopover;
    UILabel *_labelOpcionSeleccionada;

// Añadimos estas tres propiedades
@property (nonatomic, retain) SeleccionaOpcionController
*seleccionaOpcion;
@property (nonatomic, retain) UIPopoverController
*seleccionaOpcionPopover;
@property (nonatomic, retain) IBOutlet UILabel *labelOpcionSeleccionada;


 // Añadimos la definición del evento click del botón
-(IBAction) clickSeleccionaOpcion:(id) sender;
```

Ahora asignamos el Outlet del evento del botón dentro de la vista, *Ctrl+click* y arrastrar hasta File's Owner, ahi seleccionar "clickSeleccionaOpcion":



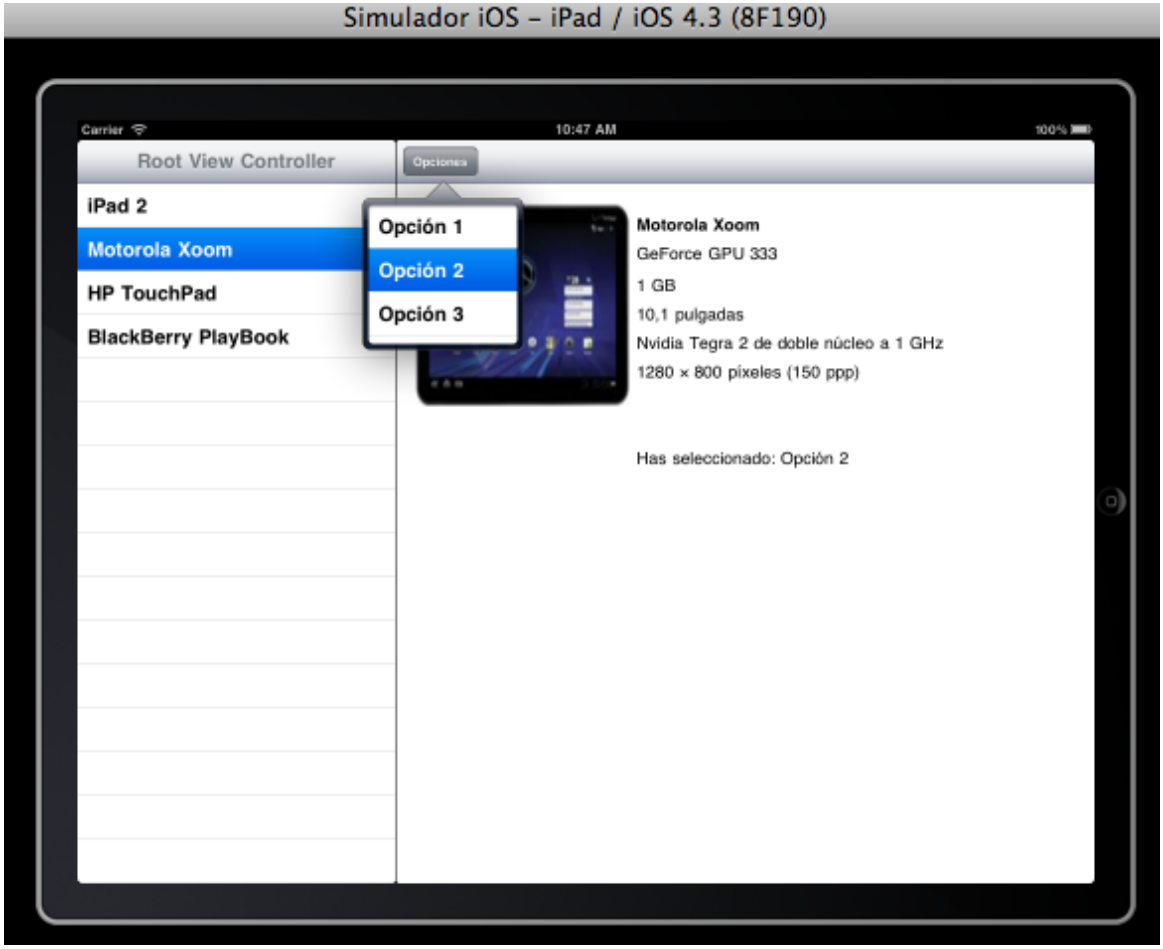
Outlets vista detalle

Nos queda completar el método de la clase delegada y el del evento click del botón. Editamos ahora `DetalleViewController.m`

```
        // Añadimos los @synthesize nuevos
@synthesize seleccionaOpcion = _seleccionaOpcion;
@synthesize seleccionaOpcionPopover = _seleccionaOpcionPopover;
@synthesize labelOpcionSeleccionada = _labelOpcionSeleccionada;


// en dealloc
self.seleccionaOpcion = nil;
self.seleccionaOpcionPopover = nil;

// Añadimos estos métodos
- (void)opcionSeleccionada:(NSString *)nombreOpcion {

    //escribimos la opción seleccionada en la label
        self.labelOpcionSeleccionada.text = [NSString stringWithFormat:
        @"Has seleccionado: %@", nombreOpcion];

        //ocultamos el popover
    [self.seleccionaOpcionPopover dismissPopoverAnimated:YES];
}

- (IBAction)clickSeleccionaOpcion:(id)sender {
    if (_seleccionaOpcion == nil) {
        self.seleccionaOpcion = [[[SeleccionaOpcionController alloc]
            initWithStyle:UITableViewStylePlain] autorelease];
        _seleccionaOpcion.delegate = self;
        self.seleccionaOpcionPopover = [[[UIPopoverController alloc]
            initWithContentViewController:_seleccionaOpcion] autorelease];
    }
    [self.seleccionaOpcionPopover presentPopoverFromBarButtonItem:sender
        permittedArrowDirections:UIPopoverArrowDirectionAny animated:YES];
}
```

Ahora volvemos a abrir `DetalleViewController.xib` y relacionamos el label que hemos creado anteriormente con el Outlet *labelOpcionSeleccionada*. Ejecutamos el

proyecto y comprobamos que todo funciona correctamente:



Popover en iPad

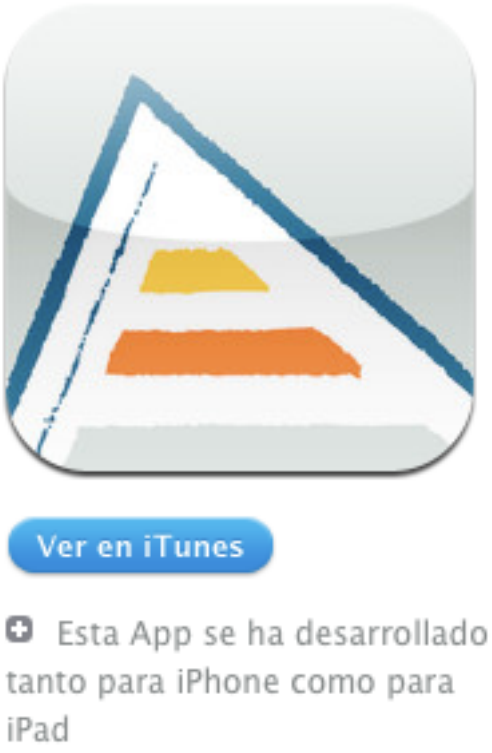
2. Aplicaciones universales

2.1. Introducción

En este apartado trataremos las **aplicaciones universales**, veremos qué ventajas e inconvenientes podemos encontrar a la hora de diseñarlas, cuál es el proceso de programación y las recomendaciones por parte de *Apple*.

Utilizamos el término **universal** para denominar a todas aquellas aplicaciones compatibles con todos los dispositivos iOS. Una aplicación universal podremos utilizarla tanto en iPhone como en iPad, sin necesidad de instalar dos distintas. Los usuarios agradecen enormemente este tipo de aplicaciones ya que mediante una sóla compra

pueden hacer uso en todos sus dispositivos. Por el lado de los desarrolladores no se tiene tan claro si este tipo de aplicaciones son más rentables que realizar dos distintas a la hora de posicionarse mejor en la *App Store* y conseguir más ventas. Lo que sí está claro es que el desarrollo de una aplicación universal ahorra un tiempo de programación muy a tener en cuenta ya que programamos una sóla aplicación y no dos. Otro punto a tener en cuenta a favor es a la hora de actualizar, ya que tendremos que preocuparnos de implementar los cambios sólo en una aplicación, no en dos. A la hora de implementar una aplicación universal tendremos que tener en cuenta si el usuario la está ejecutando sobre un iPhone / iPod Touch o sobre un iPad.



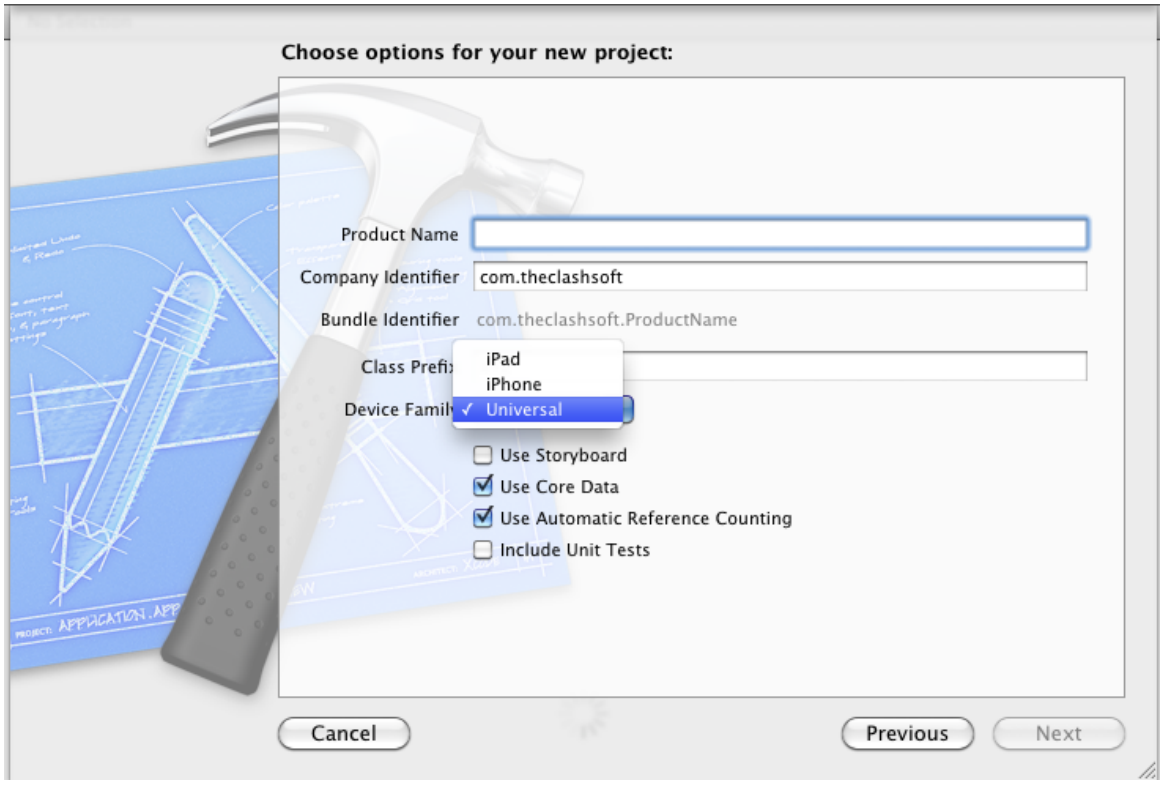
App universal UA

Con la llegada del iPad, el SDK de iPhone 3.2 empieza a soportar tres tipos de aplicaciones:

- **Aplicaciones iPhone**. Optimizadas para funcionar sobre un iPhone o iPod Touch. Estas aplicaciones se pueden ejecutar sobre un iPad en su resolución original o duplicando su resolución por 2 ajustándose de esta manera a la pantalla más grande.
- **Aplicaciones iPad**. Desde la versión de iOS 3.2, se pueden desarrollar aplicaciones optimizadas para iPad. Estas **sólo** funcionarán en iPad, no en otros dispositivos.
- **Aplicaciones Universales**. También desde la llegada de iOS 3.2 los desarrolladores pueden crear aplicaciones universales. Estas están optimizadas para funcionar en

todos los dispositivos iOS. Es, en su esencia, una aplicación iPhone y una aplicación iPad compiladas en un mismo binario.

A partir de la versión 4.2 de XCode podemos crear desde cero aplicaciones universales usando cualquier plantilla de las disponibles, utilizando esto ahorraremos mucho tiempo en cuanto a configuraciones y planteamientos iniciales. Entre las plantillas que encontramos en XCode destaca la ya mencionada anteriormente `Master Detail Application`, si la seleccionamos y marcamos la opción de *Universal* en `Device Family`, XCode nos creará la base para una aplicación universal empleando un controlador de tipo `Split View`, el cual hemos comentado en apartados anteriores.



Proyecto nuevo Universal

2.2. Diseñando la interfaz de una aplicación universal

El primer paso que debemos de realizar a la hora de hacer una aplicación universal desde cero es diseñar la interfaz para cada una de las vistas separando por un lado las vistas de iPhone/iPod Touch de las de iPad. A continuación detallaremos algunos de los puntos en los que deberemos pararnos a pensar a la hora de diseñar la interfaz:

- **Orientación.** Gracias al acelerómetro que viene en el hardware del dispositivo es posible detectar la orientación actual. Con esta información podremos adaptar la interfaz de la aplicación para acomodarla a la orientación que tenga el dispositivo en

un momento dado. En aplicaciones iPhone, la adaptación de la interfaz no es tan importante como en aplicaciones para iPad. Haciendo uso en la medida de lo posible de esta funcionalidad mejoraremos considerablemente la experiencia de usuario.
- **Estructura de vistas**. La pantalla del iPad, al ser más grande y de mayor resolución que la del iPhone permite al usuario acceder a más información en un mismo lugar.
- **Gestos.** En la pantalla del iPad podremos realizar muchos más gestos que en la del iPhone debido al tamaño de su pantalla, por ejemplo gestos con cuatro dedos al mismo tiempo.
- **Vistas partidas (Split Views).** Como hemos explicado anteriormente, una vista partida o *Split View* es una estructura de ventanas única en iPad. Mediante esta podremos dividir la pantalla del iPad en dos, mostrando distinta información en cada una de las vistas y mejorar de esta manera la usabilidad y la experiencia de uso.
- **Ventas emergentes (Popovers).** Es otra de las funcionalidades disponibles únicamente en iPad. Mediante este tipo de ventanas podremos mostrar listados o distintas opciones en un momento determinado dentro de la ejecución de nuestra aplicación.
- **Características Hardware.** Distintos dispositivos iOS poseen distintas funcionalidades hardware, unos tienen cámara de fotos, otros tienen brújula, etc. Como es lógico, hay que tener en cuenta la ausencia de este tipo de características a la hora de implementar una aplicación universal.

En la documentación que nos facilita *Apple* podremos encontrar toda la información detallada sobre las aplicaciones universales así como una extensa guía de compatibilidades entre dispositivos iOS. Podrás acceder desde esta url.

2.3. Programando una aplicación universal

A la hora de programar una aplicación universal en iOS tendremos que tener en cuenta la disponibilidad de las características de nuestra aplicación cuando se esté ejecutando en todos los dispositivos, esto lo programamos mediante *código condicional*. A continuación detallamos una serie de categorías de funcionalidades en las que tendremos que pensar a la hora de programar una aplicación universal:

- **Recursos.** En una aplicación universal necesitarás distinguir los ficheros de interfaz (nibs) a utilizar según la plataforma en donde se esté ejecutando la aplicación. Normalmente necesitarás dos ficheros distintos para cada vista o controladora, una para iPad y otra para iPhone. En cuanto a las imágenes o gráficos a utilizar dentro de la aplicación normalmente tendrás que distinguirlos también según el tipo de dispositivo, por ejemplo, si se está ejecutando sobre un iPad las imágenes deberán tener mayor resolución que si se está ejecutando sobre iPhone/iPod Touch. En el caso de iPhone 4/4S, con la pantalla *retina display*, se pueden utilizar las mismas imágenes en alta resolución que en el iPad.
- **Nuevas APIs.** Existen APIs destinadas únicamente para iPhone y otras para iPad. Por ejemplo, la librería `UIGraphics` para iPad soporta PDFs. El código debe de comprobar si estamos ejecutando la aplicación es un iPhone o en un iPad ya que los

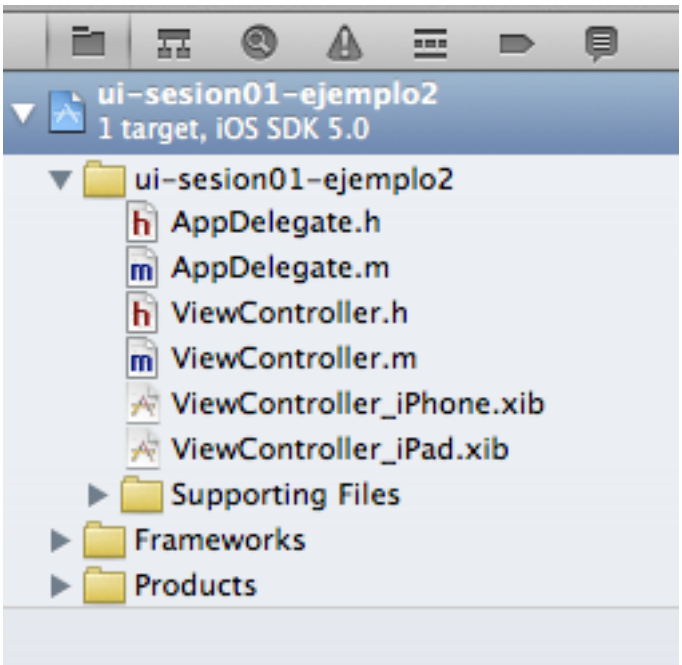
métodos correspondientes a la gestión de los PDFs sólo estarán disponibles para este último.

- **Capacidades Hardware.** A veces es necesario detectar si ciertas capacidades hardware están disponibles en el dispositivo en el que estamos ejecutando la aplicación, por ejemplo a la hora de usar la cámara ya que hay dispositivos que no la tienen.

Para entender mejor en qué consiste una aplicación universal y de cómo podemos comenzar a programar una, vamos a realizar un sencillo ejemplo paso a paso en el que mostraremos los detalles de una pelicula en una vista. Por el tamaño de la información a mostrar sobre el libro tendremos que crear **dos vistas distintas**: por un lado una adaptada al tamaño de pantalla de un iPhone/iPod Touch y por otro lado otra adaptada a un iPad.

Comenzamos abriendo *XCode* y creando un nuevo proyecto de tipo `Single View Application` al que llamaremos `ui-sesion01-ejemplo2`. Del resto de opciones seleccionamos el tipo de aplicación `Universal` (esto también se puede cambiar más adelante en el apartado "Summary" del proyecto) y desmarcamos `Use Storyboard`, pusamos sobre "Next" y guardamos el proyecto.

Como podemos ver, XCode automáticamente nos ha creado la estructura inicial básica para poder comenzar con el desarrollo de una aplicación universal. Si nos fijamos en el árbol de ficheros veremos que, aparte de la clase delegada (`AppDelegate`) tenemos un `View Controller` y dos vistas del *Interface Builder* que hacen uso este mismo `View Controller`: una vista con diseño para iPhone (`ViewController_iPhone.xib`) y otra para iPad (`ViewController_iPad.xib`).



Arbol proyecto universal

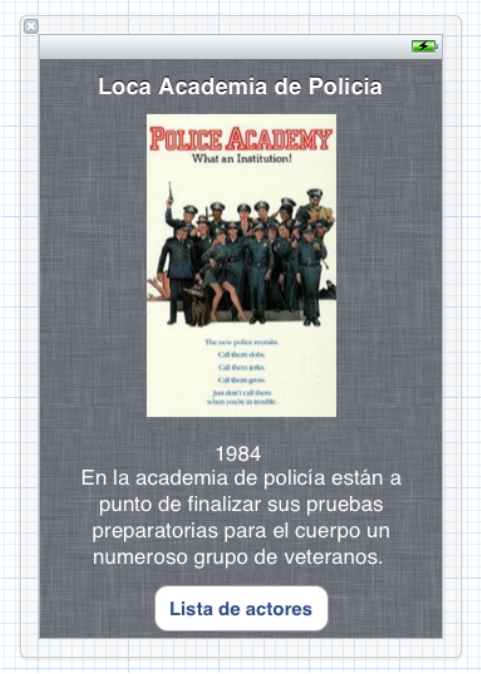
Vamos a analizar ahora en detalle la clase `AppDelegate`, en ella podemos ver como dentro del método `didFinishLaunchingWithOptions` cargamos una vista u otra dependiendo del tipo de dispositivo que esté ejecutando la aplicación. Esto se hace mediante el `userInterfaceIdiom` de la clase `UIDevice`:

```
        if ([[UIDevice currentDevice] userInterfaceIdiom] ==
UIUserInterfaceIdiomPhone) {
        self.viewController = [[[ViewController alloc]
        initWithNibName:@"ViewController_iPhone" bundle:nil] autorelease];
    } else {
        self.viewController = [[[ViewController alloc]
        initWithNibName:@"ViewController_iPad" bundle:nil] autorelease];
    }
```

Como podemos ver en el fragmento de código anterior, si el dispositivo es *iPhone* cargamos la controladora *View Controller* con la vista `ViewController_iPhone`; en caso contrario cargamos el mismo controlador pero con la vista para iPad `ViewController_iPad`. De este modo estamos ejecutando la misma controladora pero con distintas vistas dependiendo del dispositivo que se utilice. Entre muchas de las ventajas que encontramos usando este método es que sólo deberemos programar y mantener una sóla clase (en este caso `ViewController`), la cual estará "asociada" a dos vistas distintas.

Para completar el ejemplo vamos a "rediseñar" las vistas de modo que queden de la siguiente manera:

`ViewController_iPhone.xib`



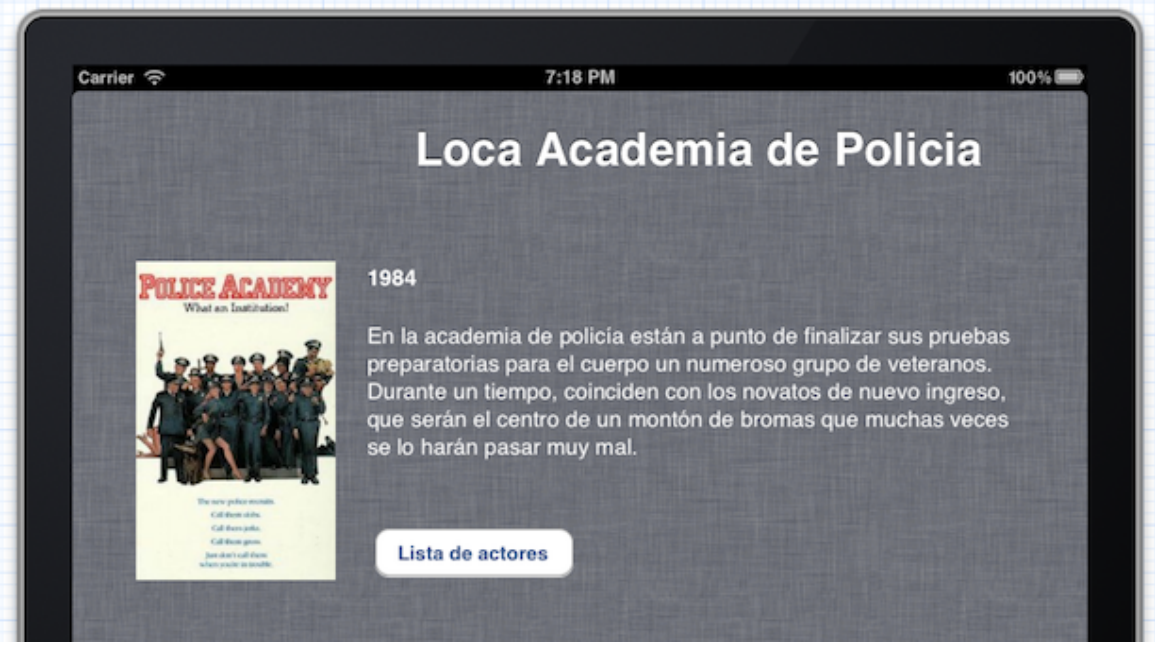
Vista iPhone

`ViewController_iPad.xib`

Vista iPad

Una vez diseñadas las vistas vamos a ejecutar la aplicación usando primero el simulador de iPhone y luego el de iPad. Veremos que funciona como esperamos, las vistas se cargan según el tipo de dispositivo que hayamos indicado:

Vista en iPad



Detalle iPad

Una vez estudiado la gestión básica de vistas en aplicaciones universales vamos a comprobar que la gestión de tablas (`UITableView`) es aún más simple. Para ello vamos a terminar este sencillo ejemplo creando la vista del listado de actores de nuestra pelicula. Para hacer esto vamos a añadir una nueva clase a nuestro proyecto de tipo `UIView Controller`. Pulsamos sobre *File > New > New File* seleccionando `iOS/Cocoa Touch`, `UIView Controller Subclass`. En el campo de *Class* escribimos **"ListaActoresTableView"** y en *Subclass of* seleccionamos `UITableViewController`. Dejamos desmarcada la opción de "Targeted for iPad" y marcamos "With XIB for user interface".

Abrimos el fichero `ListaActoresTableView.h` y definimos un objeto de tipo *NSMutableArray* que contendrá el listado de actores que queremos mostrar en la tabla:

```
        #import <UIKit/UIKit.h>
@interface ListaActoresTableView : UITableViewController
@property (strong, nonatomic) NSMutableArray *listaActores;
@end
```

Ahora en archivo `ListaActoresTableView.m` escribimos la variable con el prefijo `@synthetize` e inicializamos el array. Completamos también las funciones del protocolo de *UITableView Controller*. El código quedaría de la siguiente manera:

```
        @synthesize listaActores = _listaActores;
```

```
        - (void)viewDidLoad
{
    [super viewDidLoad];


    self.listaActores = [[NSMutableArray alloc] init];

    [self.listaActores addObject:@"Steve Guttenberg"];
    [self.listaActores addObject:@"Kim Cattrall"];
    [self.listaActores addObject:@"G.W. Bailey"];
    [self.listaActores addObject:@"Bubba Smith"];
    [self.listaActores addObject:@"Donovan Scott"];
    [self.listaActores addObject:@"George Gaynes"];
}


        - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Return the number of sections.
    return 1;
}

- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
    // Return the number of rows in the section.
    return [self.listaActores count];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault
        reuseIdentifier:CellIdentifier] autorelease];
    }

    // Configure the cell...
    cell.textLabel.text = [self.listaActores objectAtIndex:indexPath.row];

    return cell;
}
```

Por último nos falta enlazar el botón de la primera vista para que enlace con este controlador, para ello creamos una acción para el botón dentro de la clase `ViewController`. Añadimos la definición del método en el fichero .h y después desde la vista lo enlazamos para que "salte" con el evento "Touch Up Inside". Esto último deberemos hacerlo en las dos vistas (la del iPhone y la del iPad). Este el método que implementa la acción del botón y que abre en un *Modal View* la vista de la lista de actores que hemos creado anteriormente:

```
        -(IBAction)abreVistaActores:(id)sender {

    ListaActoresTableView *listaActoresTableView = [[ListaActoresTableView
```

```
alloc]
    initWithNibName:@"listaActoresTableView" bundle:nil];

    [self presentModalViewController:listaActoresTableView animated:YES];
}
```

Una vez programado esto último podemos ejecutar de nuevo la aplicación y probarla en iPhone y en iPad. Podemos ver que en el caso de las tablas, estas se adaptan al tamaño de la pantalla evitándonos de esta forma crear dos vistas distintas como hemos comentado al principio del ejercicio. En el caso que queramos profundizar más en el diseño posiblemente tendremos que crear dos vistas distintas, pero como base se puede emplear una única.

Vista en iPad

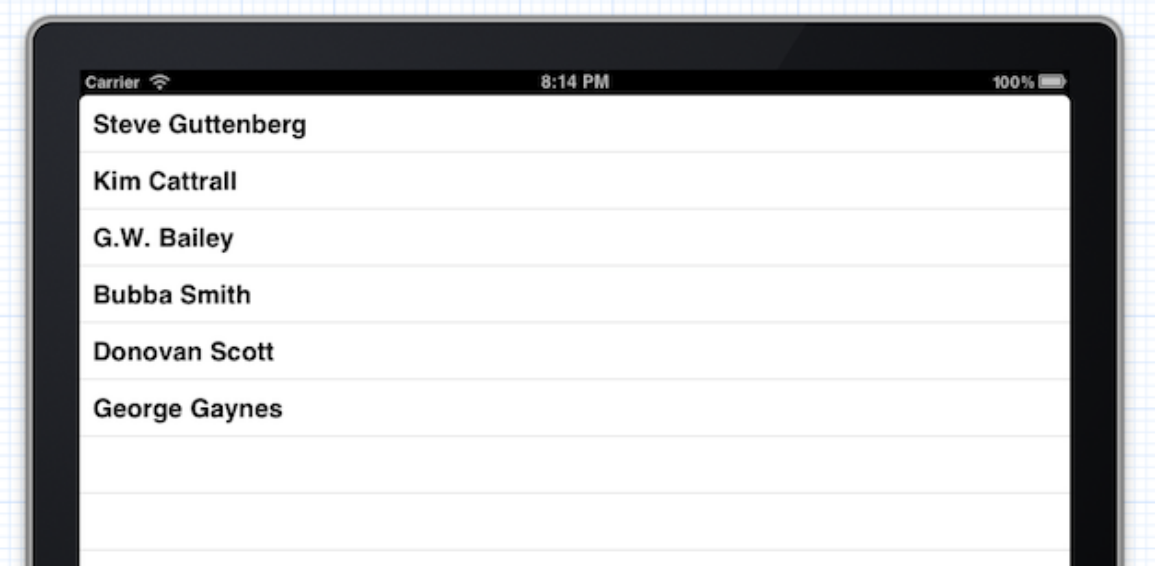


Table View iPad