Desarrollo iOS

FORMADOR

Pablo Formoso Estrada
pabloformoso.com
softwhisper.es

@pabloformoso
http://www.linkedin.com/in/pabloformoso
Skype: pablo.formoso

PROGRAMA

1. Conceptos clave de Objetive-C
2. Introducción al IDE XCode e iOS SDK
3. Interfaces en iOS y Storyboards
4. Componentes de UIKit
5. Personalización de tablas y componentes de la UI
6. Eventos y notificaciones
7. Personalización de tablas y componentes de la UI
8. Conectividad y servicios API
9. Tareas en background
10. Debbuging

Objective-C

Introducción al lenguaje, protocolos, categorías y bloques de código.

Historia

- Creado por Brad Cox y la corporación StepStone
- 1980
- 1988 NextStep lo adopta como lenguaje de trabajo
- 1992 se libera bajo licencia GPL para GCC
- Además de Apple se usa en GNU (capa step)

C, C++, Obj-C

Extensión de C
Paradigma de POO
En su mayoría el desarrollo no es estructurado
Los bloques de código en C no tienen acceso a los beneficios de Objective-C

```
#import <stdio.h>
int main( int argc, const char *argv[] ) {
    printf( "Hola Mundo\n" );
    return 0;
}
```

```
int main( int argc, const char *argv[] ) {
    NSLog( @"Hola Mundo\n" );
    return 0;
}
```

Hello Log!

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

int main(int argc, const char * argv[]) {

    @autoreleasepool {
        NSLog(@"Curso de Desarrollo iOS!!!");
    }

    return 0;
}
```

NSLog es uno de nuestro mejores amigos desde ya!

Estructuras de código

Variables

```
BOOL foo = NO;
```

Objetos

```
Estructuras *unObjeto = [[Estructuras alloc] init];
```

Funciones

```
- (BOOL)condicionales;
- (BOOL)condicionales dadoElArray:(NSArray *)array;
```

Mensajes

```
[unObjeto condicionales];
[unObjeto looping];
```

Estructuras de código

Definición de una clase

Interfaz (*.h)

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

@interface Estructuras : NSObject

- (BOOL)condicionales;
- (void)looping;

@end
```

Implementación (*.m)

```
#import "Estructuras.h"

@implementation Estructuras

- (BOOL)condicionales {
    BOOL foo = NO;

    if (foo)
        NSLog(@"FOO es true");
    else
        NSLog(@"FOO es false");


    foo = YES;

    foo ? NSLog(@"FOO es true") : NSLog(@"FOO es false");

    return foo;
}

- (void)looping {
    NSArray *array = [NSArray arrayWithObjects:@"1", nil];

    for (NSString *cadena in array) {
        NSLog(cadena);
    }
}

@end
```

Estructuras de código

Elementos básicos de la interfaz de un objeto

```
- (id)initForVitae; // Constructor
```

```
@property (nonatomic, strong) NSString *name; // Atributo público
```

En la implementación usaremos `@synthesize` para generar los getter y setters

```
#pragma mark - Métodos públicos
- (BOOL)condicionales;
- (void)looping;
```

Métodos privados (en el *.m)

```
@interface Estructuras (PrivateMethods)
- (void)metodoPrivado;
@end
```

Protocolos y delegados

Representan la parte más importante del lenguaje

A modo de símil los protocolos actúan como interfaces abstractas de Java

Los delegados son los encargados de implementar estos protocolos en nuestro código

Protocolos y delegados

Declaración de un protocolo (iOSTeamDelegate.h)

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

@protocol AulaDelegate <NSObject>
- (void)agregarAlumnos:(NSString *)nombre;
- (void)listarAlumnos;

@optional
- (void)esteNoLoHacemos;

@end
```

`@optional` describe los métodos cuya implementación es opcional, el resto son obligatorios

Categorías

Definen un comportamiento extendido para un objeto

NSString+PSA.h

```
@interface NSString (PSA)

- (void)imprimirConPSAEstaCadena;

@end
```

NSString+PSA.m

```
#import "NSString+PSA.h"

@implementation NSString (PSA)

- (void)imprimirConPSAEstaCadena {
    NSLog(@"%@ (PSA)", self);
}

@end
```

main.m

```
#import "NSString+PSA.h"
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSString *cadena = @"Cadena simple";
        NSLog(@"Sin Categoría %@", cadena);
        [cadena imprimirConPSAEstaCadena];
    }
    return 0;
}
```

Bloques de código

Agrupan código bajo una declaración de bloque.

`@autoreleasepool` existe en todas las apps y se encarga de gestionar el release de los objetos.

```
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSString *cadena = @"Cadena simple";
        NSLog(@"Sin Categoría %@", cadena);
        [cadena imprimirConPSAEstaCadena];
    }
    return 0;
}
```

`@syncronize` es un de los más usados para hacer transacciones.

Bloques de código

Asigando a variables

```
void (^simpleBlock)(void) = ^{
 NSLog(@"This is a block");
};
simpleBlock();
```

Bloques de código

Parámetros y retunrs

```
float (^multiFloat)(float) = ^(float n) {
 return n * 2.0f;
};

NSLog(@"Multiplica %.2f", multiFloat(10.0f));
```

Bloques de código

Funciones

```
- (void)beginTaskWithCallbackBlock:(void (^)(void))callbackBlock {

 callbackBlock();

}

[beginTaskWithCallbackBlock:^{
 NSLog(@"Dentro de un bloque");
}];
```

El bloque siempre tiene que ser el último parámetro.

Intro a iOS SDK

Estructura de la API, componentes de una App, XCode4
y su entorno, limitaciones y sistema de ficheros.

Estructura de la API

Nuestro código... MCV

o... MCCV

Cuando hacemos aplicaciones híbridas para iPhone y iPad en muchas ocasiones nos encontraremos con la encrucijada de tener que hacer un controlador diferente para cada vista según el dispositivo.

Otra variante más complicada para los casos híbridos es el uso de MCPV (Model - Controller - Presnter - View) pero requiere más complejidad y los beneficios son estructurales.

Componentes de una App

1. Elementos gráficos obligatorios
2. AppDelegate (es el que manda y lanza el MainThread)
3. *.plist contiene la configuración de la app
4. *.pch contine las declaraciones globales
5. UserDefaults un punto de apoyo para trabajar con datos
6. Targets y schemas
7. *.strings para internacionalización
8. UIWindows único
9. Fuentes, xibs, recursos etc...

XCode4

1. Editor
2. Biblioteca de recursos
3. Snippets
4. Interface Builder integrado
5. Organizer
6. Analyzer: leaks, timming, performance
7. Diferentes vistas de código

Sistema de ficheros de una app

Carpeta **Documents**

Almacena los recursos **propiedad del usuario** para el funcionamiento de la aplicación. Se guarda en iCloud con la copia de seguridad y se sincroniza con iTunes.

Carpeta **tmp**

Recursos temporales que podamos necesitar para manipular datos. Volátil en cada arranque de la aplicación.

Carpeta **Library/Caches**

Carpeta donde podemos almacenar elementos de forma prolongada. Al actualizar la aplicación se eliminan y tampoco se sincronizan con iCloud o iTunes

Interfaces en iOS

Principales componentes, plantillas de XCode, jerarquía de vistas, tablas en detalle, StoryBoards

Jerarquía de vistas de una App

Toda app tiene al menos un **UIWindow** principal
UIWindow es el contenedor para todos nuestros Views

UIView es el contenedor más usado para agregar los componentes de la interfaz. La gran mayoría extienden su comportamiento para hacer un UIScrollView, UILabel, UIButton, etc...

Los componentes que generan acciones extienden también UIResponder, parte que permite capturar eventos.

Jeraquía de vistas de una App

Los UIView normalmente dependen de un UIViewController para gestionar eventos, IBOutlets o IBActions que generemos.

Los IBOutlets son las conexiones que generamos entre el IB y una instancia de nuestro código.

Las IBActions son las acciones que desencadenan los componentes en nuestro controlador.

Plantillas de proyectos

Plantillas de XCode

La más común es Tabbed Application

En la última versión de XCode4 sacaron la de navegación y toca agregarlo a mano.
Con iOS5 como target podemos usar StoryBoards.

PageBased

Basan su modelo en la premisa de una vista por componente de datos. Es útil para publicaciones, PDFs o hacer slides sencillos sobre un mismo modelo de datos.

SingleView

Nos crea un UIViewController con su respectivo *.xib con la interfaz para que trabajamos sobre ella.

Es un buen punto de partida para una aplicación sencilla como el contador que veremos a continuación

Utility Application

La más usada por las “chorri apps” tipo iRecio y iMourinho. Nos permite tener una zona de configuración o de información fuera de la vista principal, todo ello por defecto.

No es muy recomendable ya que Apple a día de hoy ya no pasa este tipo de aplicaciones

Tabbed App

Sin duda la más usada de todas las plantillas.

Anteriormente su uso era más programático pero se ha simplificado como veremos en el código a continuación.

Ejemplo de ficheros

NSBundle

Permite acceder a los fichero dentro la app compilada.

Objeto de tipo Singleton.

[NSBundle mainBundle] para sacar la instancia.

Solo necesitamos nombre y extension, localiza el fichero de forma automática.

NSFileManager

Objeto en forma Singleton

Permite manipular ficheros de las carpetas a las que tenemos acceso.

Operaciones de lectura y escritura.

[NSFileManager defaultManager]

Storyboards

A partir de iOS

Simplifican la creación de Interfaces usando un solo fichero.

Nos permite establecer relaciones y acciones entre diferentes vistas y sus controladores.

La relaciones son dependientes del componente.
Las acciones se usan para navegar entre vistas. Estas reciben el nombre de Segues.

Segues

Usados para navegar ente vistras con los StoryBoards

Comportamiento estándar que podemos sobre escribir en el método prepareForSegue:

La navegación la realizamos invocando al metodo performSegueWithIdentifier:sender:

Todas las clases que hereden de UIViewController tiene acceso a estos métodos.

Un proyecto de 0

How Long Does it Take to Build an iOS or Android App?

Based on a survey of 100 iOS, Android and HTML5 developers, it takes nearly 18 weeks to build v1 of a native app.

SERVER-SIDE LOGIC

How a developer truly customizes the user's experience.

E R

USER MANAGEMENT

Creating user accounts, managing authentication, security and access control.

C

Start Here

DATA STORAGE

The building block of any native app's backend.

C

CACHING

Storing data locally to speed load time.

SYNCHRONIZATION

Enable off-line usage and resolve data conflicts.

E

WIREFRAMING

Blueprint for user interface and experience.

C

E U

DATA INTEGRATION

Allowing users to access information from and publish data to third party sources, including social networks.

E U

PUSH

Maintaining engagement with users continuously.

C

VERSIONING

Make version 2 live ... without breaking version 1.

E U

UI POLISH

The differentiating "last mile" of the user interface, where an app truly stands out from the crowd.

C U

UI DEVELOPMENT

Translating mock-ups into functioning user interface code.

C U

UI DESIGN

Pixel perfect mock-ups of user interface.

- 1 DAY OF TIME SPENT
- C - CORE REQIUREMENT
- U - USER FACING
- E - ENGAGEMENT DRIVER
- R - REVENUE OPPORTUNITY

THIS MESSAGE WAS BROUGHT TO YOU BY kinvey

Credits: Joe Chernov, Dave Wasmer of Kinvey. Survey data: AYTM

SERVER-SIDE LOGIC
How a developer truly customizes the user's experience.
E
R
USER MANAGEMENT
Creating user accounts, managing authentication, security and access control.
C
Start Here
DATA STORAGE
The building block of any native app's backend.
C
13
13
12
backend
10
weeks
12
13
6
E
U
E
U
C
DATA INTEGRATION
Allowing users to access information from and publish data to third party sources, including social networks.
PUSH
Maintaining engagement with users continuously.
VERSIONING
Make version 2 live ... without breaking version 1.

CACHING

Storing data locally to speed load time.

SYNCHRONIZATION

Enable off-line usage and resolve data conflicts.

WIREFRAMING

Blueprint for user interface and experience.

UI POLISH

The differentiating "last mile" of the user interface, where an app truly stands out from the crowd.

UI DEVELOPMENT

Translating mock-ups into functioning user interface code.

UI DESIGN

Pixel perfect mock-ups of user interface.

Nuestra App

- Aplicación para la consulta de clases, alumnos y recursos
- Desarrollada en Rails con servicios REST
- Soporte para xml y json
- Se compone de:
  - Listado de alumnos y detalle
  - Listado de clases y su detalle
  - Listado de recursos con vista previa o detalle
  - Mapa para ubicar cursos y como llegar

Repaso

Introducción a ObjC
Estructura de una app en iOS y su SDK
Introducción a XCode
Plantillas básicas
Sistema de ficheros
Interface Builder: IBOutlets y IBActions
Principales componentes
Introducción a los Storyboards

Nuestra App

Creamos un app con Storyboards
“Diseñamos” el flujo de navegación con los Segues
Integramos la primera tabla
Creamos el primer detalle de vista

UITableViews

UITableViewDelegate

Describe los métodos para el manejo de la tabla.
Taps, edición, etc...

UITableViewDataSource

Describe los métodos para nutrir la tabla con datos.
La obtención de datos puede venir de varias fuentes.
Define parámetros como el número de secciones y elementos de cada sección.

Persistencia de datos

Formas de persistencia

NSKeyedArchiver

SQLite

CoreData

NSKeyedArchiver

Serialización de objetos para almacenado rápido.
Volúmenes bajos de datos.
Se guardan en el NSUserDefault para su consulta.
No se puede hacer modificación directa.

Métodos del “protocolo”
- (id)initWithCoder:(NSCoder *)coder
- (void)encodeWithCoder:(NSCoder *)coder

SQLite

Trabajo a nivel de bases de datos.
Soporta un volumen de datos suficiente para una app.
Almacenamiento en el sistema de ficheros.
CRUD y operaciones normales de SQLite

La mejor estrategia es la categorización.
Construimos nuestras sentencias a partir de SQLite.h

Conectividad y parsing

NSURLConnection

Junto con NSURLRequest es la principal clase para le trabajo de red.

Puede funcionar de forma síncrona o asíncrona.

NSURLConnectionDelegate incorpora los métodos para trabajar con la conexión.

NSURLConnection + XML

NSXMLParser y NSXMLParserDelegate son los objectos con los que podemos gestionar el parsing de XML.

Encapsulamos el delegado del parser en en BaseXmlParser.

En la respuesta al NSURLConnection realizamos el parsing.

Desde la llamada a el render

6 Llamada por protocolo INFORMAL

Librería AFNetworking

Encapsula todo el comportamiento de NSURLConnection.

Facilita la conexión eliminado código.

Funcionamiento por bloques y soporte para JSon o XML

AFNetworking + JSONKit

AFJSONRequestOperation nos permite realizar un request a un recurso JSON.

El objeto JSON nos da acceso al diccionario con la información del objeto.

Extiende NSOperation por lo que podemos agregarla a una nueva NSOperationQueue o a la principal.

AFNetworking + POST

Para enviar datos por POST usaremos el AFHTTPClient.

A través del AFHTTPClient obtenemos una request multipart y calcula por nosotros ciertos aspectos de un POST como la frontera.

Los parámetros o datos los agregamos usando NSDictionary.

Más componetes

Edicción de tablas

Eliminado rápido con swipe.

Modo edición:

- Eliminar elementos
- Reordenar elementos

Formularios

UITextField, UIText, UIButtons

Con el UITextFieldDelegate gestionamos los eventos con sobre los diferentes textfields.

Para ocultar el teclado podemos invocar al resignFirstResponder.

Podemos gestionar acciones a través de las notificaciones del teclado.

UIWebView

Cargamos el detalle del recurso creado.

El UIWebViewDelegate

- webViewDidStartLoad:
- webViewDidFinishLoad:

Uno por cada request necesaria en la web.

UIScrollView

Uno de los layouts más potentes que tenemos.

UIScrollViewDelegate es el protocolo con todas las acciones sobre las que podemos trabajar sobre el scroll.

Las más interesantes:

- viewForZoomingInScrollView:
- scrollViewDidScroll:

Diferenciar el Frame del ContentSize.

MapKit - MKMapView

MKMapView es el componente principal para represtar mapas.

El MKMapViewDelegate nos permite realizar acciones de tracking, detectar cambios entre zonas, responder a eventos.

MapKit - MKAnnotation

MKAnnotation interfaz para definir los Pins del mapa.

Se posicionan pasando coordenadas CLLocationCoordinate2D.

Por defecto soportan título y subtítulo.

MapKit - MKPolyline

MKPolyline es el objeto que nos permite agregar rutas o líneas encima de un mapa.

Recibe un conjunto de coordenadas que establecen los puntos de la ruta.

Con [MKPolyline polylineWithCoordinates:puntos count:numero_de_puntos] generamos el objeto.

Los agregamos al mapa a través de su Overlay.

CoreLocation

Nos permite acceder al GPS y realizar tracking.

Su principal delegado es CLLocationManagerDelegate.
A través del CLLocationManager gestionamos los datos.

Principales métodos:
- didUpdateToLocation: fromLocation:
- didFailWithError:

CoreLocation

Se activa en el momento con autorización del usuario.

Encapsulamos en SWLocationManager el comportamiento básico.

Lanzamos y paramos el tracking con:

- startUpdatingLocation
- stopUpdatingLocation

NotifcationCenter

NSNotificationCenter

Es la central de notificaciones/eventos en iOS

Se instanción por medio de “observer” y se envía como mensaje por push

Implementado en forma de Singlenton accedemos a través del [NSNotificaiónCenter defaultCenter]

NSNotificationCenter

Es la central de notificaciones/eventos en iOS

Se instanción por medio de “observer” y se envía como mensaje por push

Implementado en forma de Singlenton accedemos a través del [NSNotificaiónCenter defaultCenter]

NSNotificationCenter

El observer es el encargado de atender a un “nombre” de notificación para lanzar un determinado método.

La forma más común es:

– addObserver:selector:name:object:

Recibe quién atenderá al mensaje, el método, el nombre de la notificación y posibles objetos.

NSNotificationCenter

La publicación es más sencilla. Se realiza a través del método:

- postNotificationName:object:

Debemos acordarnos de eliminar siempre los observers:

- removeObserver:

NSNotificationCenter

Ejemplo:

- Notificaciones del teclado.
  “UIKeyboardWillShowNotification”
  “UIKeyboardWillHideNotification”

- Ejemplo propio al eliminar una clase.

Tap, Tap Tap, Pinch...
Gestos

UIGestureRecognaizer

Objeto principal encargado de la gestión de gestos.

Podemos agregar varios predefinidos o inclusos patrones pregrabados.

Se ejecutan a través de trigers o de la implementación del UIGestureRecognaizerDelegate

UIGestureRecognaizerDelegate

Contienen todos los métodos para que podamos gestionar los gestos agregados a los componentes.

El más relavante es:
– gestureRecognizerShouldBegin:

Para gesitionar múltiples gestos simultaneos debemos usar:
– gestureRecognizer: shouldRecognizeSimultaneouslyWithGestureRecognizer:

UIGestureRecognaizer

A ráiz de este nos encontramos con sus propios derivados:

- UITapGesture....
- UIPinchGesture....
- UISwipeGesture....
- Pan, Rotation, LongPress...

y sus delegados.

Con los Storyboards podemos hacer un drop de estos sobre los componentes y configurarlos.

Background
Estados de la app

Estados de la app

Arranque de la app

Salto a segundo plano

Procesos en background

iOS nos permite gestionar un número limitado de tareas en segundo plano:

- Reproducir contenido multimedia
- Tracking de corelocation
- Soporte para Voip
- Procesos de descarga y actualización de contenidos del newstand
- i/o de dispositivos conectados

Procesos en background

Nuestra app puede registrar uno más de estos "permisos" en su .plist bajo el array *UIBackgroundModes*

Cada modo tiene sus particularidades.

Background: Audio

Se permite usar para:
- AirPlay
- Reproducciones con AVSession o Media Player
- Grabación

Puede recibir eventos remotos de la interfaz general de iOS y configurar el aspecto externo.

Background:VoIP

Se permite usar para:
- Mantener un socket abierto y realizar consultas periódicas con el fin de establecer una comunicación.

En algunos casos durante el proceso de validación Apple puede exigir una demo para comprobar la veracidad del servicio :(

- setKeepAliveTimeout:handler: nos permite desde aquí abrir los sockets y definir el handler para la conexión.

Background: Newsstand

Se permite usar para:
- consulta de la una rss de productos y descarga automática de los mismo si estamos suscritos.

La RSS se da de alta en el iTunesConnect y se ataca desde el propio Newsstand.framework

Background: CoreLocation

Se permite usar para:
- mantener al usuario informado de “eventos” cerca.

Apple es realmente duro con el uso de este modo y tira mucha apps para evitar tracking de usuarios.

Recomienda el uso de CoreLocation con el significant-change location service (una forma de parametrizar el location manager),

Nuestra app: Música

- Accederemos a la librería de audio
- Montaremos una sesión de audio con AVSession
- Crearemos el player
- Gestión de eventos remotos
- Reproducción en background con más apps corriendo otras AVSession

Animaciones CoreAnimation & CoreGraphics

Animar con UIView

- Por contexto:
  - + beginAnimations:context:
  - + commitAnimations
  - + setAnimation[Property]
- Por bloques:
  - + animateWithDuration:animations:completion:
  - + animateWithDuration:delay:options:animations: completion:

Animar con CoreAnimation

- Un nivel por debajo del UIKit y por encima de CoreGraphics
- Necesitamos menos configuración y ganamos en flexibilidad.

- Soporte para diferente tipos de capas
  - CALayer
  - CAScrollLayer
  - CATiledLayer

Animar con CoreAnimation

- Nos provee de un conjunto de métodos para facilitarlos las animaciones y la manipulación de componentes.
- Su sistema de coordenadas se mantiene pero usa el anchorPoint como ancla para las animaciones.
  - (0,0) - superior izq
  - (1,1) - inferior der
  - (0.5,0.5) - centro

Animar con CoreAnimation

- Preparación:
  - + setDisableActions:
  - + setAnimationDuration:
- Ejecutamos las modificaciones que queramos a continuación sobre el CALayer.

CA Avanzado

- Uso de Paths para animar por rutas.
- Loops continuos
- Efectos pulse
- SWinggle
- Reflexión de imágenes

CoreGraphics

- Nos permite trabajar con diferentes contenidos a través de contextos.
- Podemos desde renderizar PDFs, fusionar imágenes y aplicar filtros hasta escalar una imagen o recortarla.

CoreGraphics

- El contexto depende de la situación los más comunos son:
  - CGContextRef para proposito general
  - UIGraphicsBeginImageContext para trabajo con imágenes a un nivel un poco más alto.

CoreGraphics: Ejemplo de fusión

- UIGraphicsBeginImageContext
- Cargamos las dos imágenes
- Abrimos el contexto
- Las “renderizamos” en su interior
- Recuperamos la imagen resultado