

iCloud y notifications push

Índice

| | |
|---|----|
| 1 iCloud..... | 2 |
| 1.1 Definición y características de iCloud..... | 2 |
| 1.2 Ejemplos de casos de uso de iCloud..... | 5 |
| 1.3 Desarrollando con iCloud: Preparaciones iniciales..... | 5 |
| 1.4 Desarrollando con iCloud: Configuración del proyecto..... | 7 |
| 1.5 Desarrollando con iCloud: Programación de ficheros..... | 8 |
| 1.6 Desarrollando con iCloud: Pruebas de funcionamiento de ficheros..... | 15 |
| 1.7 Desarrollando con iCloud: Programación de clave-valor..... | 16 |
| 2 Notificaciones push..... | 17 |
| 2.1 Funcionamiento..... | 19 |
| 2.2 Registro en Urban Airship y configuración en iOS Provisioning Portal..... | 21 |
| 2.3 Programación de la aplicación de ejemplo..... | 26 |

Con la llegada de la nueva revisión de iOS al mercado, el iOS 5, los desarrolladores se han puesto manos a la obra para adaptar en la medida de lo posible todas sus aplicaciones a los nuevos requerimientos y especificaciones de *Apple*. A lo largo del curso hemos hablado de distintas novedades que nos trae iOS 5 como el nuevo sistema de ARC (Automatic Reference Counting), la integración de *Twitter* en el sistema operativo, los *Storyboards* para diseñar la estructura de las *apps*, etc. En esta sesión es el turno de hablar de una de los mayores novedades que nos proporciona esta nueva versión de iOS y es nada más y nada menos que **iCloud**.

Con iCloud podremos, en resumen, gestionar todos los documentos, configuraciones e incluso bases de datos completas en la *nube* para de este modo tener acceso de forma totalmente transparente e instantánea para el usuario final a la misma información en todos los dispositivos iOS, Macs y PCs que tengamos asociados a nuestra cuenta. En el primer apartado ampliaremos en profundidad este tema y veremos qué ventajas nos puede llegar a suponer el uso de este sistema en nuestras aplicaciones. También diseñaremos una aplicación desde cero que implemente *iCloud*.

Por otro lado hablaremos de las **notificaciones push**, las cuales podemos hacer uso en las aplicaciones que desarrollemos para determinados casos y que, sin duda, ofrecerán un punto a favor en cuanto a mejora en la experiencia del usuario y sobre todo, un gran avance en temas de marketing. De las *notificaciones push* hablaremos en el segundo apartado de esta sesión y veremos también cómo implementarlas en nuestras aplicaciones.

¡Comenzamos!

1. iCloud

Seguro que en algún momento has oído hablar de **iCloud**, ya sea por la televisión, por la prensa escrita o por Internet, el caso es que desde *Apple* nos "venden" esta nueva característica de iOS 5 como algo menos que "el mayor avance de todos los tiempos". Pero en realidad, ¿Qué supone para nosotros, los desarrolladores esta nueva funcionalidad? A continuación explicaremos en qué casos podremos implementarla, cuales son los requerimientos y un ejemplo concreto desarrollado desde cero con el que podremos comprobar cómo funciona iCloud en un entorno real. ¡Vamos para allá!

1.1. Definición y características de iCloud

iCloud es una nueva funcionalidad disponible a partir de iOS 5 en la que, mediante el uso de una API, podremos almacenar todo tipo de documentos y propiedades de nuestras aplicaciones en la nube. Todas las actualizaciones que se produzcan en iCloud se transmitirá de forma inalámbrica, automática y sincronizada a cualquier dispositivo que lo soporte (iDevice con iOS 5 ó PC/Mac debidamente configurados).



Cada usuario dispondrá de una cuenta que tendrá que configurar en su dispositivo y que será normalmente la misma que *iTunes*. La configuración de iCloud en los dispositivos con iOS se realiza desde la aplicación de *Ajustes*, dentro del apartado *iCloud*. Ahí deberemos especificar nuestro nombre de usuario y contraseña en el caso que dispongamos de ello, sino deberemos de crear un nuevo usuario.

El almacenamiento de iCloud es un conjunto de interfaces diseñadas para compartir datos a través de distintas instancias de la aplicación en distintos dispositivos. Los cambios que se realicen en estos datos se propagarán automáticamente por los distintos dispositivos que ejecuten la misma aplicación. Entre los grandes beneficios que esto ofrece a los desarrolladores están:

- Ya no hará falta en la mayoría de casos contratar un servicio externo que haga las funciones de servidor ya que usaremos el propio de *Apple* ahorrando, por tanto muchísimos costes adicionales.
- No hará falta escribir una API propia para el acceso a datos externos ya que la propia de *iCloud* nos ofrece todo lo que necesitamos tanto para la recepción de documentos desde iCloud como para el envío. También existen varios métodos referentes a la sincronización de los datos para evitar inconsistencias, algo que trataremos en los

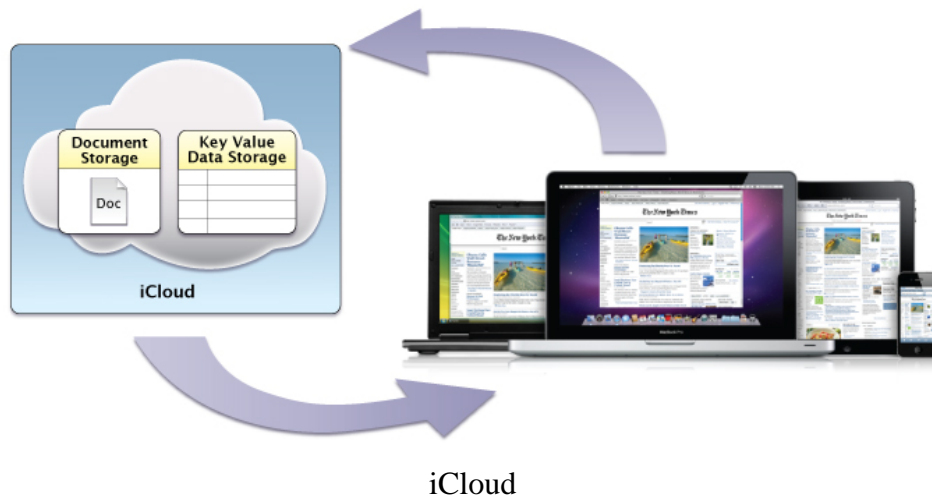
siguientes puntos.

Existen también una serie de condiciones que deberemos de tener en cuenta a la hora de incorporar este servicio en nuestras aplicaciones:

- **Límite de capacidad:** Cada usuario tiene como máximo en principio **5GB** de almacenamiento en su cuenta iCloud gratuita a repartir entre todas las aplicaciones que utilicen este servicio. En el caso que contrate una cuenta premium (de pago) este límite se aumentará hasta llegar a los 50GB. Los desarrolladores tendremos que tener en cuenta este límite y optimizar en la medida de lo posible los datos y/o documentos que nuestra aplicación comparta en la nube.
- **Sincronizaciones:** Las posibles incoherencias o conflictos que puedan aparecer en nuestra aplicación las debemos de gestionar nosotros de forma manual. La API de *iCloud* nos proporciona una serie de métodos que nos ayudan a esta gestión pero debemos ser nosotros mismos los que la implementemos en nuestras aplicaciones.
- **Autorizaciones (entitlements) necesarios:** Para que iCloud funcione deberemos de configurar previamente dentro del portal de desarrollador las autorizaciones necesarias y activar el servicio de iCloud para la aplicación que desarrollemos. Esto deberemos hacerlo también desde la configuración del proyecto en XCode, aunque lo detallaremos más adelante en el ejemplo que realicemos desde cero.
- **Funcionamiento sin iCloud:** Nuestra aplicación deberá funcionar correctamente en el caso de que iCloud no esté configurado en el dispositivo en donde se esté ejecutando. iCloud se debe tratar como una funcionalidad "extra" y nunca como "necesaria" para el funcionamiento correcto de nuestra aplicación, es por ello que deberemos de comprobar en nuestro código si el usuario dispone de una cuenta de iCloud configurada en el dispositivo y en el caso de que no sea así no utilizaremos iCloud pero sí el resto de funcionalidades.

Apple ha establecido dos formas de incorporar iCloud a nuestras aplicaciones, las cuales deberemos de tener en cuenta a la hora de implementarlo, son estas:

- **Almacenamiento de documentos:** Esta característica la utilizaremos para almacenar documentos (ficheros) de distintos tipos en la cuenta iCloud. No existe límite de almacenamiento (el máximo disponible en la cuenta del usuario).
- **Almacenamiento de datos tipo clave-valor:** La utilizaremos para almacenar pequeñas cantidades de datos como configuraciones del usuario, datos de acceso a la aplicación, etc. El límite máximo de almacenamiento está en 64KB.



1.2. Ejemplos de casos de uso de iCloud

iCloud es una funcionalidad que se puede incorporar en gran parte de las aplicaciones iOS que vemos en la actualidad, sin embargo hay muchas otras en la que su implementación no sería un acierto o simplemente no aportaría nada o casi nada a la mejora de la experiencia de usuario. A continuación comentaremos distintos tipos de aplicaciones en donde iCloud puede ser de utilidad:

- **Aplicaciones de gestión documental:** Para compartir documentos entre varios dispositivos.
- **Aplicaciones de fotografía:** Para compartir fotografías que hagamos con el iPhone / iPad entre varios dispositivos.
- **Aplicaciones que hagan uso de un calendario:** Compartir eventos del calendario.
- **Aplicaciones de gestión de tareas (ToDos):** Compartir tareas entre varios dispositivos.
- **Aplicaciones con apartados de configuración:** Compartir datos de la configuración de una aplicación entre dispositivos.
- **Juegos de distintos tipos:** Compartir datos de niveles alcanzados, puntuaciones, etc.

Como podemos ver existen muchos tipos de aplicaciones en la que usar el servicio de iCloud puede llegar a ser un acierto. A continuación veremos cómo implementar dicho servicio en nuestras aplicaciones iOS mediante un sencillo ejemplo.

1.3. Desarrollando con iCloud: Preparaciones iniciales

Una vez vistas las características principales del uso de iCloud así como distintos ejemplos de uso vamos a desarrollar desde cero nuestra primera aplicación con iCloud. En el ejemplo que vamos a comentar implementaremos una sencilla aplicación en la que el usuario creará un documento de texto simple y este se gestionará a través de iCloud.

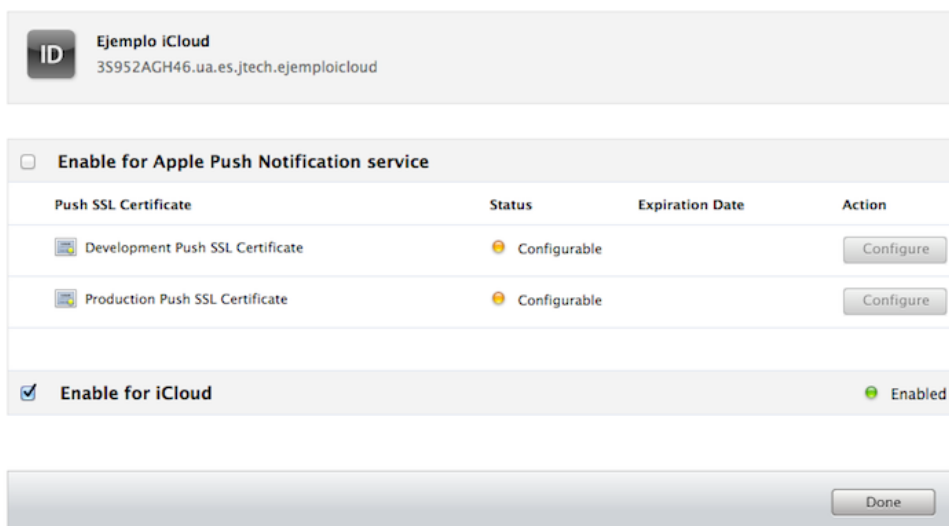
También almacenaremos los datos de acceso a la aplicación en la nube. Como se puede deducir, haremos uso de los dos métodos de almacenamiento comentados en el primer punto: **almacenamiento de documentos** y **almacenamiento de datos tipo clave-valor**.

Para empezar a desarrollar en iCloud deberemos de configurar previamente una serie de datos y opciones en nuestra cuenta de desarrollador (<http://developer.apple.com>). Comenzaremos configurando una nueva aplicación, para ello deberemos acceder a portal de desarrolladores (<http://developer.apple.com>) utilizando nuestro nombre de usuario y contraseña. Una vez dentro accederemos al *Provisioning Portal* y crearemos un nuevo App ID que llamaremos, por ejemplo, ua.es.jtech.ejemploicloud.

Atención

Tenemos que tener en cuenta que el App ID que creemos debe de coincidir plenamente con el Bundle identifier que tendrá la aplicación que creemos posteriormente en XCode.

Una vez creado el App ID nos aparecerá en la lista del *Provisioning Portal* tal y como aparece en la imagen siguiente:



Provisioning Portal: iCloud

Como podemos ver, iCloud no está configurado aún, para configurarlo deberemos pulsar sobre *configure*. En la ventana que nos aparece marcamos para activar iCloud tal y como aparece en la imagen de a continuación:

| Description | Apple Push Notification service | In App Purchase | iCloud | Action |
|---|---|-----------------|--------------|---------------------------|
| 3S952AGH46.* Xcode: Wildcard AppID | Unavailable | Unavailable | Enabled | Configure |
| 3S952AGH46.ua.es.jtech.ejemploicloud... Ejemplo iCloud | Configurable for Development Configurable for Production | Enabled | Configurable | Configure |

Configuración app con iCloud

Ahora abrimos la pestaña de Provisioning y creamos un nuevo provisioning profile para la aplicación que contenga el App ID creado anteriormente, el certificado que queramos usar y los dispositivos con los que queramos probar la aplicación. Ahora nos descargamos el *provisioning profile* que acabamos de crear y haremos doble click sobre el para adjuntarlo a nuestra lista en XCode.

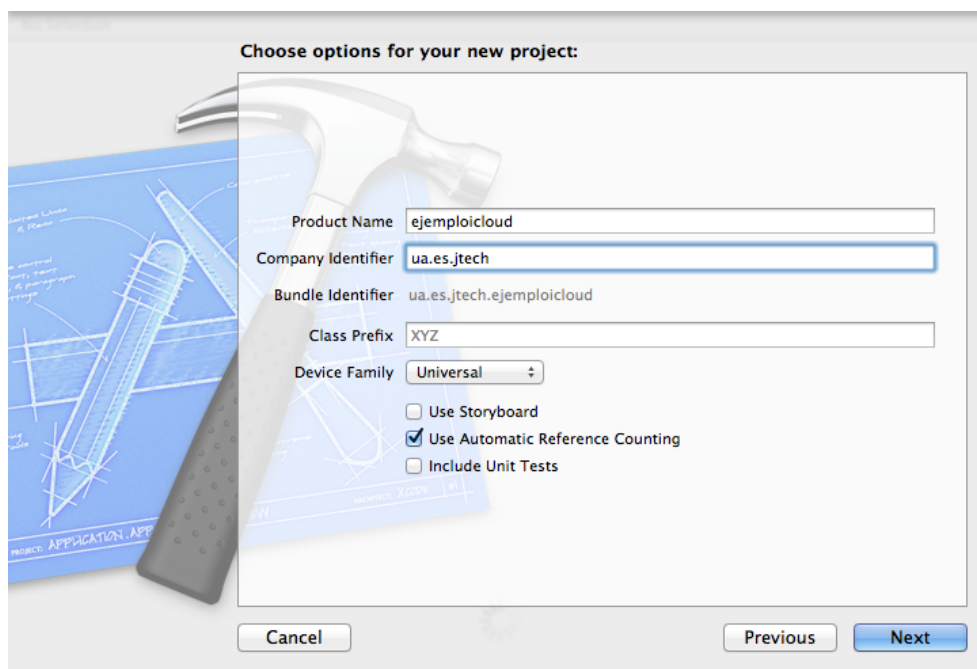
Pruebas de iCloud

Para poder probar iCloud deberemos de disponer de un dispositivo (iPhone , iPad o iPod Touch) debidamente configurado desde el *provisioning portal* y con iOS 5 instalado. iCloud no funcionará en el simulador de XCode.

Una vez que tenemos todos los preparativos previos para usar *iCloud* ya podemos empezar a programar nuestra aplicación.

1.4. Desarrollando con iCloud: Configuración del proyecto

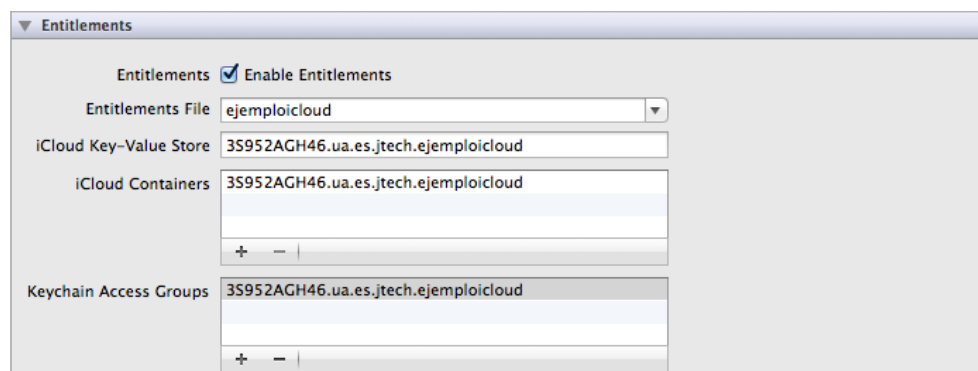
En este apartado comprobaremos lo sencillo que es hacer una aplicación compatible con *iCloud*. Comenzaremos abriendo XCode y creando un nuevo proyecto de tipo Single View Application que será Universal, que llamaremos ejemploicloud y tendrá como *Company Identifier*: ua.es.jtech. Pulsamos sobre Next y ya tenemos el proyecto de XCode creado.



Configuración proyecto XCode

Abrimos las propiedades del proyecto y nos dirigimos al apartado de `Entitlements` que se encuentra dentro de la pestaña de `Summary`, en el último apartado. Ahí marcamos la casilla de `Enable Entitlements`. Automáticamente nos aparecerá un nuevo archivo en el raíz del proyecto que se llamará `ejemploicloud.entitlements`. Ahora deberemos modificar los campos de `iCloud Key-Value Store` y `iCloud Containers` con el `Team-ID` de los *provisioning profile* que hemos creado anteriormente. Para ver el *Team-ID* debemos acceder de nuevo a la sección de *Apps IDs* del *iOS Provisioning Portal* y copiar la cadena de texto que aparece justo antes del nombre del *profile*, en nuestro caso es: `3S952AGH46`.

Ahora editamos los tres campos del apartado *Entitlements* del resumen del proyecto quedando de la siguiente manera:



Entitlements

Con esto último ya tenemos el proyecto debidamente configurado para usar iCloud. Ahora deberemos de implementarlo en nuestro código, ¡vamos a ello!

1.5. Desarrollando con iCloud: Programación de ficheros

Comenzaremos programando la gestión de documentos en iCloud, para ello deberemos de crear una clase que herede de `UIDocument` la cual personalizaremos e implementaremos toda la gestión dentro del `AppDelegate`. En la vista crearemos un campo de tipo `TextView` que será el que contenga el texto que almacenaremos o cargaremos (según convenga) en el fichero que creemos dentro del almacenamiento iCloud.

Nota

Realizar toda la implementación básica de gestión de iCloud del modelo dentro de la clase `AppDelegate` no es lo recomendado ya que no se cumpliría de forma estricta la estructura MVC propuesta por Apple. Nosotros lo hemos realizado de esa manera para simplificar el ejemplo.

Comenzamos creando la clase que gestionará el documento de iCloud, para ello hacemos click sobre *New > New File*, seleccionamos `Objective-C class` y lo llamamos

MiDocumento. Dentro del fichero MiDocumento.h escribimos lo siguiente:

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

@interface MiDocumento : UIDocument {
    NSString *documentText;
    id delegate;
}

@property (nonatomic, retain) NSString *documentText;
@property (nonatomic, assign) id delegate;

@end
```

Y en el fichero MiDocumento.m lo siguiente:

```
#import "MiDocumento.h"

@implementation MiDocumento

@synthesize documentText = _text;
@synthesize delegate = _delegate;

// ** LECTURA **

- (BOOL)loadFromContents:(id)contents ofType:(NSString *)typeName
error:(NSError **)outError
{
    NSLog(@"UIDocument: loadFromContents: state = %d, typeName=%@",
        self.documentState, typeName);

    if ([contents length] > 0) {
        self.documentText = [[NSString alloc] initWithBytes:[contents
bytes]
        length:[contents length] encoding:NSUTF8StringEncoding];
    }
    else {
        self.documentText = @"";
    }

    NSLog(@"UIDocument: Se ha cargado el siguiente texto desde iCloud:
%@",
        self.documentText);

    return YES;
}

// ** ESCRITURA **

-(id)contentsForType:(NSString *)typeName
error:(NSError **)outError
{
    if ([self.documentText length] == 0) {
        self.documentText = @"Nueva nota para el curso especialista de
moviles";
    }
}
```

```

NSLog(@"UIDocument: Guardare el siguiente texto en iCloud: %@",
      self.documentText);

return [NSData dataWithBytes:[self.documentText UTF8String]
        length:[self.documentText length]];
}
@end

```

Como podemos ver, la clase `MiDocumento` hereda de `UIDocument`, una nueva clase disponible en iOS 5, este tipo se encargará de encapsular todos los datos referidos a un documento de texto convencional. En iCloud deberemos usar esta clase para gestionar los documentos que creamos. Dentro de nuestra clase `MiDocumento` hemos sobrescrito la propiedad de `text` y la `delegate`. Ahora la propiedad `documentText` será la que contenga el texto.

Hemos sobrescrito dos métodos de la clase `UIDocument`:

- `loadFromContents`: Realiza la lectura del documento. Convierte los datos recibidos en bytes a una cadena de texto `NSString` codificada en UTF8, la cual almacenaremos dentro de la propiedad `documentText`.
- `contentsForType`: Realiza la escritura en el documento. Devuelve en forma de datos `NSData` la cadena de texto convertida a UTF8.

Ahora abrimos el fichero `AppDelegate.h` y creamos las siguientes propiedades y métodos al final de este. Estos métodos se explicarán más adelante:

```

@property (strong, nonatomic) UIWindow *window;
@property (strong, nonatomic) NSMetadataQuery *query;
@property (strong, nonatomic) MiDocumento *documento;

- (void)cargaDocumento;
- (void) escribeEnDocumento:(NSString *)texto;

```

Abrimos el `AppDelegate` y añadimos el siguiente código dentro del método `didFinishLaunchingWithOptions`:

```

// (1) Iniciamos iCloud
NSURL *ubiq = [[NSFileManager defaultManager]
URLForUbiquityContainerIdentifier:@"3S952AGH46.ua.es.jtech.ejemploicloud"];
if (ubiq) {
    NSLog(@"AppDelegate: Entra en iCloud!");
    [self cargaDocumento];
} else {
    NSLog(@"AppDelegate: No hay acceso a iCloud (puede que estés en el
simulador o que no esté configurado correctamente el
dispositivo");
}

```

El el fragmento de código anterior generamos la dirección del contenedor de iCloud correspondiente al identificador de nuestra aplicación mediante el método

URLForUbiquityContainerIdentifier del singleton NSFileManager. En el caso que encuentre esa dirección de iCloud pasamos a cargar el documento, en caso contrario significará o que no existe ese contenedor de iCloud o que no podamos acceder a el por cualquier otro motivo (no disponemos de conexión a internet, estamos probando el proyecto sobre el simulador o que el dispositivo no esté bien configurado para aceptar iCloud).

Ahora creamos el método `cargaDocumento` que será el encargado de hacer la llamada a iCloud para comprobar que existe el fichero que queremos cargar, en nuestro caso será `text.txt`:

```
- (void)cargaDocumento {  
    // (2) iCloud query: Busca a ver si hay un fichero llamado "text.txt"  
    en iCloud.  
  
    NSMetadataQuery *query = [[NSMetadataQuery alloc] init];  
    _query = query;  
  
    //Asignamos el scope  
    [query setSearchScopes:[NSArray arrayWithObject:  
    NSMetadataQueryUbiquitousDocumentsScope]];  
  
    //Creamos un predicado y lo asignamos a la query  
    NSPredicate *pred = [NSPredicate predicateWithFormat: @"%K == %@",  
    NSMetadataItemFSNameKey, @"text.txt"];  
    [query setPredicate:pred];  
  
    //Creamos una notificacion que se llame cuando la query haya terminado  
    // (esta se ejecutará en segundo plano de forma asincrona)  
    [[NSNotificationCenter defaultCenter] addObserver:self  
    selector:@selector(queryDidFinishGathering:)  
    name:NSMetadataQueryDidFinishGatheringNotification object:query];  
    [query startQuery];  
}  
  
- (void)queryDidFinishGathering:(NSNotification *)notification {  
    // (3) Si la query ha terminado cargaremos los resultados de esta en  
    el  
    // siguiente metodo: cargaQuery  
  
    NSMetadataQuery *query = [notification object];  
    [query disableUpdates];  
    [query stopQuery];  
  
    [self cargaQuery:query];  
  
    [[NSNotificationCenter defaultCenter] removeObserver:self  
    name:NSMetadataQueryDidFinishGatheringNotification object:query];  
    _query = nil; // ya no lo necesitaremos mas  
}
```

Ahora nos falta implementar el método encargado de gestionar la consistencia del documento. Comprobaremos que si existe dicho documento (la query devuelve un resultado) lo cargaremos en el campo de texto de la vista (esta la crearemos en el

siguiente paso). En el caso de que la query no devuelva ningún resultado crearemos un nuevo documento y lo almacenaremos en iCloud. Para hacer esto crearemos el siguiente método:

```

- (void)cargaQuery:(NSMetadataQuery *)query {
    // (4) cargaQuery: Si el fichero existe lo abrimos y se lo asignamos
    // al documento
    // de la clase. En caso contrario lo creamos, lo asignamos al documento
    // de la clase.

    if ([query resultCount] == 1) {
        // Encontrado el archivo en iCloud

        NSMetadataItem *item = [query resultAtIndex:0];
        NSURL *url = [item valueForKey:NSMetadataItemURLKey];

        MiDocumento *doc = [[MiDocumento alloc] initWithFileURL:url];
        _documento = doc;

        [doc openWithCompletionHandler:^(BOOL success) {
            if (success) {
                NSLog(@"AppDelegate: Abriendo documento existente de
iCloud");
                NSLog(@"El documento contiene el texto: %@",
doc.documentText);
                [[NSNotificationCenter defaultCenter]
postNotificationName:@"cargaTextoiCloud"
object:nil];
            } else {
                NSLog(@"AppDelegate: El documento existente en iCloud ha
fallado al abrirse");
            }
        }];
    } else {
        // No existe el documento en iCloud: lo creamos

        NSLog(@"AppDelegate: Documento no encontrado en iCloud");

        NSURL *ubiq = [[NSFileManager defaultManager]
URLForUbiquityContainerIdentifier:@"639M4BK859.com.theclashsoft.icloudtest"];
        NSURL *ubiquitousPackage = [[ubiq
URLByAppendingPathComponent:@"Documents"]
URLByAppendingPathComponent:@"text.txt"];

        MiDocumento *doc = [[MiDocumento alloc]
initWithFileURL:ubiquitousPackage];
        doc.documentText = @"Texto inicial del documento en iCloud...";
        _documento = doc;

        [doc saveToURL:[doc fileURL]
forSaveOperation:UIDocumentSaveForCreating
completionHandler:^(BOOL success) {
            NSLog(@"AppDelegate: new document save to iCloud");
            [doc openWithCompletionHandler:^(BOOL success) {
                NSLog(@"AppDelegate: Nuevo documento creado en iCloud");
                [[NSNotificationCenter defaultCenter]
postNotificationName:@"cargaTextoiCloud" object:nil];
            }];
        }];
    }
}

```

```
        }];  
    }  
}
```

Como podemos ver en el método anterior enviamos un mensaje a una notificación `cargaTextoiCloud` que aún no hemos definido, esta vamos a implementarla dentro de la vista `ViewController`:

```
- (void)cargaTextoiCloud:(NSNotification *)notification {  
    AppDelegate *delegate = [[UIApplication sharedApplication] delegate];  
    self.textView.text = delegate.documento.documentText;  
}
```

Ahora, dentro del fichero `ViewController.m`, en el método `viewDidLoad` definimos la notificación que se usará para indicar que el fichero se ha cargado de iCloud y se puede mostrar su texto en el componente `UITextView`:

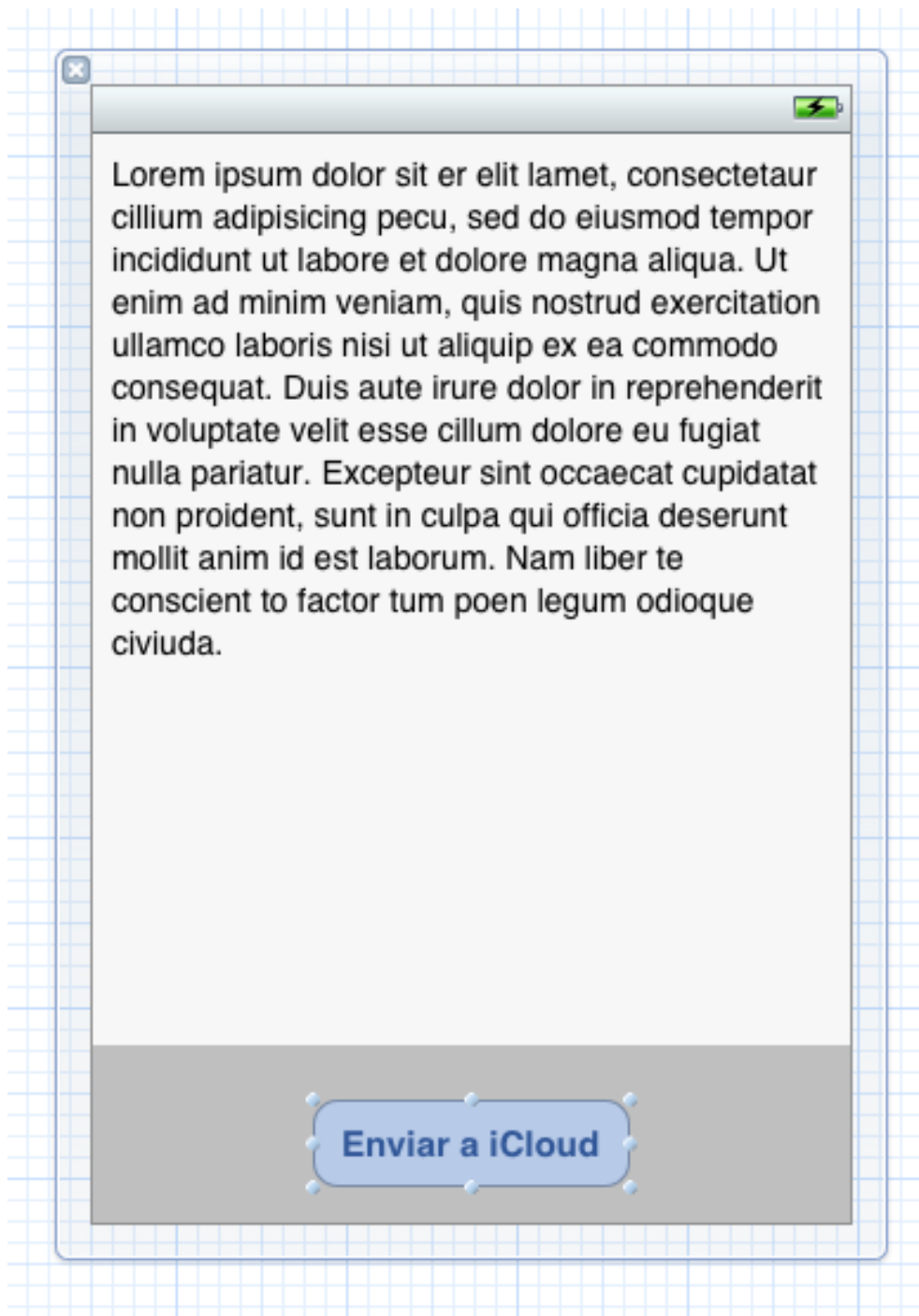
```
- (void)viewDidLoad  
{  
    [super viewDidLoad];  
    // Do any additional setup after loading the view, typically from  
    a nib.  
  
    [[NSNotificationCenter defaultCenter] addObserver:self  
        selector:@selector(cargaTextoiCloud:) name:@"cargaTextoiCloud"  
        object:nil];  
}
```

En el fichero `ViewController.h` deberemos de crear un *Outlet* para el *Text View* y un *Action* para la acción del botón que guardará el nuevo fichero en iCloud.

```
// Añadimos lo siguiente debajo de las definiciones  
@property (strong, nonatomic) IBOutlet UITextView *textView;  
  
-(IBAction)clickBoton:(id)sender;  
  
//Añadimos el @syntentize en ViewController.m  
@synthesize textView = _textView;  
  
//Creamos el siguiente método en ViewController.m que implementará la  
acción del botón  
-(IBAction)clickBoton:(id)sender {  
    AppDelegate *delegate = [[UIApplication sharedApplication] delegate];  
    [delegate escribeEnDocumento:self.textView.text];  
}
```

Por último nos queda diseñar la vista, para ello abrimos el fichero `ViewController_iPhone.xib` y `ViewController_iPad.xib` y añadimos en ambos un

componente `Text View` y un `Button`. Nos queda enlazarlos con los *Outlet* de la clase y quedaría de la siguiente manera para iPhone:



ViewController_iPhone.xib

Una vez hecho esto ya podemos ejecutar nuestra aplicación sin antes acordarnos de establecer los *provisioning profiles* adecuados en la pestaña de *summary* del proyecto para poder arrancarla en nuestros dispositivos.

1.6. Desarrollando con iCloud: Pruebas de funcionamiento de ficheros

Las pruebas se deberían de hacer en varios dispositivos ejecutando la aplicación en estos para comprobar que iCloud funciona correctamente. Deberemos de comprobar que el documento recién creado se actualiza de forma automática en nuestros dispositivos. También podemos comprobar que dentro de los ajustes de iCloud del dispositivo aparece el fichero creado `text.txt`.

La primera vez que arrancamos la aplicación nos debe de aparecer lo siguiente en la consola de XCode:

```
2011-11-02 13:22:01.644 icloudtest[3733:707] AppDelegate: Entra en
iCloud!
2011-11-02 13:22:02.255 icloudtest[3733:707] AppDelegate: Documento no
encontrado en
iCloud
2011-11-02 13:22:02.419 icloudtest[3733:707] UIDocument: Guardare el
siguiente texto en
iCloud: Texto inicial del documento en iCloud...
[Switching to process 8195 thread 0x2003]
2011-11-02 13:22:02.799 icloudtest[3733:707] AppDelegate: Nuevo documento
creado en
iCloud
2011-11-02 13:22:03.329 icloudtest[3733:707] UIDocument: loadFromContents:
state = 0,
typeName=public.plain-text
2011-11-02 13:22:03.330 icloudtest[3733:707] UIDocument: Se ha cargado el
siguiente
texto desde
iCloud: Nueva nota para el curso especialista de moviles
2011-11-02 13:22:03.352 icloudtest[3733:707] AppDelegate: Abriendo
documento existente
de iCloud
```

Como podemos ver, al arrancar la aplicación por primera vez no se encuentra ningún documento en iCloud por lo que se crea uno y se muestra el contenido en el `TextView`.

Si ahora borramos la aplicación del dispositivo y la volvemos a ejecutar deberá de encontrar el fichero de iCloud y cargar su contenido en el `Text View`. Aparecerá algo similar a lo siguiente en la consola de XCode:

```
2011-11-02 13:30:54.149 icloudtest[3766:707] AppDelegate: Entra en
iCloud!
2011-11-02 13:31:11.424 icloudtest[3766:707] UIDocument: loadFromContents:
state = 1,
typeName=public.plain-text
2011-11-02 13:31:11.425 icloudtest[3766:707] UIDocument: Se ha cargado el
siguiente
texto desde iCloud: Nueva nota para el curso especialista de moviles
```

```
[Switching to process 8451 thread 0x2103]
2011-11-02 13:31:11.755 icloudtest[3766:707] AppDelegate: Abriendo
documento existente
de iCloud
2011-11-02 13:31:11.756 icloudtest[3766:707] El documento contiene el
texto: Nueva nota
para el curso especialista de moviles
```

Como podemos ver se ha cargado el documento generado desde iCloud de manera correcta. Para modificar el texto del documento simplemente deberemos escribirlo dentro del Text View y pulsar sobre el botón *Enviar a iCloud*, si hacemos la prueba escribiendo el siguiente texto: Cambiando el texto del documento text.txt en iCloud... veremos que aparece lo siguiente en la consola:

```
2011-11-02 13:34:11.999 icloudtest[3766:707] UIDocument: Guardare el siguiente texto
en iCloud: Cambiando el texto del documento text.txt en iCloud... 2011-11-02
13:34:12.079 icloudtest[3766:707] AppDelegate: Nuevo documento creado en iCloud
2011-11-02 13:34:12.606 icloudtest[3766:707] UIDocument: loadFromContents: state =
0, typeName=public.plain-text 2011-11-02 13:34:12.607 icloudtest[3766:707]
UIDocument: Se ha cargado el siguiente texto desde iCloud: Cambiando el texto del
documento text.txt en iCloud... 2011-11-02 13:34:12.629 icloudtest[3766:707]
AppDelegate: Abriendo documento existente de iCloud
```

Ahora podemos seguir realizando pruebas en distintos dispositivos seguir observando el proceso que realiza iCloud.

1.7. Desarrollando con iCloud: Programación de clave-valor

La implementación de iCloud para su uso en el modo clave-valor es mucho más sencillo que para ficheros. Lo único que debemos hacer es modificar el código dentro de la inicialización de iCloud del método `didFinishLaunchingWithOptions` de la clase AppDelegate:

```
// Iniciamos iCloud
NSURL *ubiq = [[NSFileManager defaultManager]
URLForUbiquityContainerIdentifier:
@"639M4BK859.com.theclashsoft.icloudtest"];
if (ubiq) {
    NSLog(@"AppDelegate: iCloud access!");

    // inicio clave-valor
    NSUbiquitousKeyValueStore *cloudStore = [NSUbiquitousKeyValueStore
defaultStore];
    [cloudStore setString:[ubiq absoluteString] forKey:@"iCloudURL"];
    [cloudStore synchronize]; // Sincroniza los datos locales con los
de iCloud

    NSLog(@"Valor encontrado en iCloud: %@",[cloudStore
stringForKey:@"iCloudURL"]);
    // fin de clave-valor

    [self cargaDocumento];
```



```
    } else {  
        NSLog(@"AppDelegate: No iCloud access (either you are using  
simulator or,  
        if you are on your phone, you should check settings");  
    }
```

Como podemos ver, el método clave-valor de iCloud es muy similar usar `NSUserDefaults` para persistencia de datos dentro de un mismo dispositivo. En iCloud se almacena un diccionario con todos los datos clave-valor que deseemos. En este caso únicamente almacenamos una clave llamada `iCloudURL` que contendrá la dirección URL de almacenamiento de iCloud.

Este método es muy útil para almacenar propiedades, configuraciones, datos de usuario, etc en la nube y tener acceso de este modo en todos nuestros dispositivos de manera instantánea. Comentar que además de poder almacenar valores de tipo `String` se pueden almacenar de otros tipos como numéricos e incluso objetos completos.

2. Notificaciones push

Las aplicaciones iOS no pueden realizar (o no deben al menos) muchos de sus procesos en segundo plano por cuestiones de batería. Sabiendo esto, *¿cómo podemos hacer para avisar al usuario de que algo interesante está sucediendo?* Por ejemplo, imaginemonos que un usuario se descarga una aplicación de una biblioteca en un momento dado y un mes después sale un libro a la venta que le puede interesar... *¿cómo podemos avisar a ese usuario que el libro x le puede interesar?* Ahí es donde entran en acción las famosas **notificaciones push**.



Notificación Push

Para evitar tener uno o varios procesos corriendo en nuestro dispositivo que estén comprobando cada cierto tiempo si algún evento aparece Apple ha creado las notificaciones push. Estas notificaciones necesitarán de una configuración en un servidor que correrá por nuestra cuenta y por muy poca programación en la aplicación.

Las notificaciones push pueden enviar los siguientes tipos de mensaje o una combinación de estos:

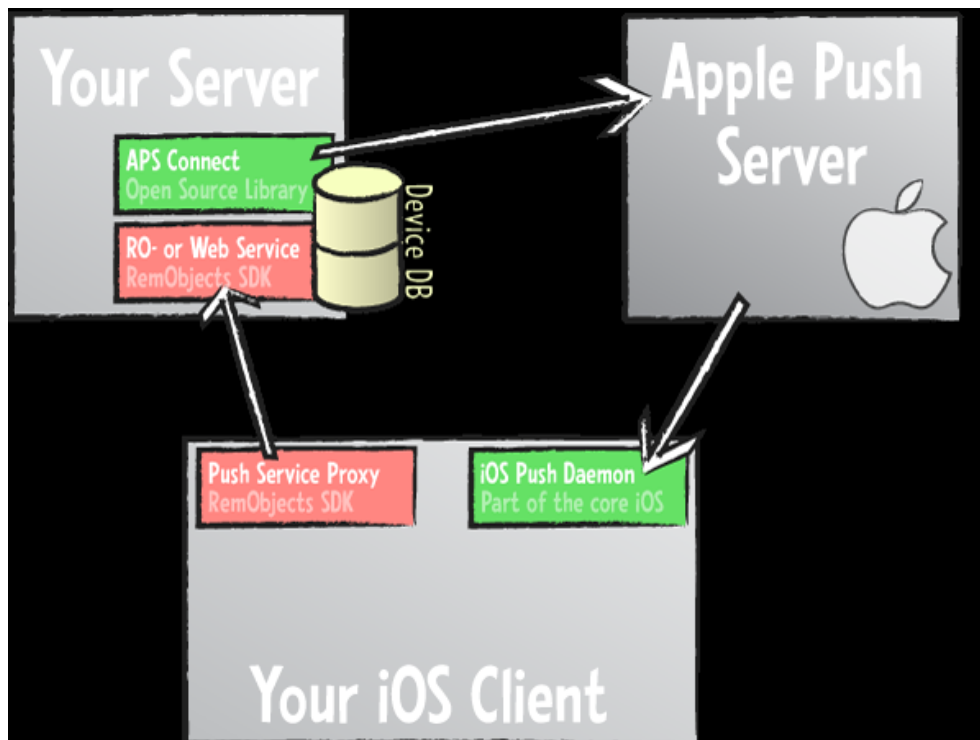
- Mensajes de texto cortos
- Reproducir un sonido
- Mostrar un número en el icono de la aplicación (*badge*)

Podremos enviar por ejemplo un mensaje de texto junto con un sonido para conseguir una mayor atención del usuario.

Las notificaciones push son un muy buen método para realizar campañas de marketing ya que podremos avisar a todos nuestros usuarios de algún evento o producto en cualquier momento y sin resultar ser demasiado intrusivos. Además todas las notificaciones que nuestros usuarios reciban tendrán la opción de ejecutar la aplicación en el caso de que esta se encuentre en segundo plano o cerrada. Conviene tener en cuenta que el uso desmesurado de notificaciones push por parte del desarrollador puede provocar el efecto contrario, que el usuario se borre la aplicación del dispositivo para dejar de recibir este tipo de avisos.

2.1. Funcionamiento

Como hemos comentado anteriormente, el uso de notificaciones push en nuestras aplicaciones implica que tengamos que configurar un servidor que sea el que envíe estas notificaciones mediante mensajes con formato *JSON* al servidor de notificaciones de Apple (*APNS Server*). Este servidor de Apple será el que envíe la notificación al dispositivo del usuario. En el siguiente gráfico podemos ver cómo funciona el servicio de notificaciones push de una manera más clara:



Esquema funcionamiento push

- 1. Registramos el ID del dispositivo en nuestro servidor. *Al usuario le aparecerá una ventana de confirmación preguntando si desea que nuestra aplicación le pueda enviar notificaciones.*
- 2. Nuestro servidor almacenará el ID del dispositivo en una base de datos para posteriormente usarlo para enviar notificaciones.
- 3. Enviamos una notificación desde nuestro servidor al APNS (Servidor de Apple). Necesitaremos haber configurado previamente un certificado SSL junto con una clave privada. La notificación será un mensaje con estructura JSON.
- 4. El APNS enviará la notificación al usuario del dispositivo.

Cuando el usuario recibe la notificación se mostrará una alerta, desde la cual podrá abrir la aplicación (importante tener en cuenta para temas de marketing). Las notificaciones push se mostrarán al usuario aunque la aplicación esté apagada o incluso el dispositivo en reposo.

¿Qué necesitaremos para poder implementar este sistema en mis aplicaciones? Necesitaremos lo siguiente:

- Un dispositivo (iPhone o iPad). Las notificaciones push en el simulador de XCode **no** funcionan.
- Una cuenta activa de desarrollador Apple. Necesitaremos crear un App ID en el portal de *Apple Developer* y configurar las notificaciones push para este App ID. También tendremos que generar un certificado SSL especial.

- Un servidor conectado a Internet. Será en encargado de enviar las notificaciones push al servidor de Apple (APNS). **Problema:** el servidor tendrá que ser capaz de ejecutar procesos en segundo plano, instalar certificados SSL y hacer conexiones TLS a ciertos puertos, por lo que necesitaremos contratar un servidor privado (VPS) o buscar soluciones alternativas de pago.

El tercer requerimiento es algo complicado de solucionar ya que necesitaríamos contratar un servidor privado virtual o, en cualquier caso, implementarlo nosotros configurando una serie de procesos, instalar un certificado, etc. Para simplificar este punto vamos a usar un servicio online complementario el cual nos proporcionará todo lo necesario para incorporar notificaciones push a nuestras aplicaciones sin preocuparnos por el servidor: **Urban Airship** (<http://urbanairship.com/>).

Urban Airship es un sitio online para desarrolladores de plataformas móviles (iOS, Android, Blackberry, etc) que ofrece distintos servicios para estas plataformas entre los que se encuentran el de notificaciones push e in-apps entre otros. El servicio de notificaciones push es bastante bueno y funciona muy bien. Empresas como Fox, Accenture, Macworld, Tapulous, Warner Bros o Yahoo lo utilizan en sus aplicaciones.



Urban Airship

Urban Airship ofrece distintas tarifas según nuestras necesidades. Por suerte disponen de una tarifa gratuita con limitaciones, nosotros escogeremos esa tarifa para realizar una aplicación de ejemplo.

2.2. Registro en Urban Airship y configuración en iOS Provisioning Portal

Antes de empezar a implementar nuestra aplicación de ejemplo vamos a registrarnos en el

site de Urban Airship, para ello abrimos un navegador y entramos en <https://go.urbanairship.com/accounts/register/>. Una vez dentro deberemos rellenar todos los campos del formulario para abrir una cuenta básica (*gratuita*) la cual nos permitirá enviar hasta un millón de mensajes push al mes (después veremos algunas limitaciones).

Una vez que nos hayamos registrado y confirmado la cuenta podremos acceder al panel de control desde donde podremos revisar todos los datos de nuestra cuenta, el tipo de cuenta (básica), el número de mensajes push enviados en total, etc. Ahora deberemos crear una nueva *app*, pero antes tendremos que generar el certificado SSL de Apple que permitirá a Urban Airship enviar las notificaciones push a los dispositivos.

The screenshot shows the Urban Airship account dashboard. On the left is a sidebar with navigation links: Account, Overview, Change Username, Change Email, Change Password, Billing, Upgrade Account, and Delete Account. The main content area is titled 'Account' and displays the following information:

| Account | |
|-----------------------------|--|
| Account Plan | Basic Change your account plan |
| Username | theclashsoft |
| Email | admin@theclashsoft.com |
| Account Created | 6 minutes ago |
| Number of Applications | 0 |
| Pushes sent (month) | 0 |
| Active Users (month) | 0 |
| In-App downloads (month) | 0 |
| Pushes sent (all time) | 0 |
| In-App downloads (all time) | 0 |
| How much we love you | Lots |

All numbers are for production only and do not include messages or downloads during development.

Datos app en Urban Airship

Para crear el certificado SSL de Apple accederemos al portal de desarrolladores usando nuestro nombre de usuario y contraseña. (<https://developer.apple.com/ios/manage/overview/index.action>). Una vez dentro abrimos la pestaña de *Apps IDs* y crearemos una nueva *App ID* para la aplicación de ejemplo que vamos a desarrollar. Completaremos el formulario con los datos que se muestran en la siguiente captura:

The screenshot shows the 'Create App ID' form in the iOS Provisioning Portal. The header includes 'iOS Provisioning Portal' and a welcome message for 'Javier Aznar de los Rios'. The left sidebar has links for Home, Certificates, Devices, App IDs (selected), and Provisioning. The main content area has tabs for 'Manage' and 'How To'. The form fields are: 'Description' with a text input containing 'Ejemplo notificaciones push'; 'Bundle Seed ID (App ID Prefix)' with a note that the Team ID (3S952AGH46) will be used; and 'Bundle Identifier (App ID Suffix)' with a text input containing 'es.ua.jtech.ejemplopush' and an example 'com.domainname.appname'. At the bottom right are 'Cancel' and 'Submit' buttons.

Create App ID

Una vez creado el App ID lo configuramos para activar las notificaciones push, para ello pulsamos sobre el link de "configure" y marcamos la casilla Enable for Apple Push Notification service tal y como se muestra a continuación:

The screenshot shows the configuration page for the App ID 'Ejemplo notificaciones push' (ID: 3S952AGH46.es.ua.jtech.ejemplopush). It features a table with columns for 'Push SSL Certificate', 'Status', 'Expiration Date', and 'Action'. The 'Enable for Apple Push Notification service' checkbox is checked. The table lists two certificates: 'Development Push SSL Certificate' and 'Production Push SSL Certificate', both with a status of 'Configurable' and a 'Configure' button. Below the table, the 'Enable for iCloud' checkbox is unchecked, also with a 'Configurable' status. A 'Done' button is at the bottom right.

| Push SSL Certificate | Status | Expiration Date | Action |
|----------------------------------|--------------|-----------------|-----------|
| Development Push SSL Certificate | Configurable | | Configure |
| Production Push SSL Certificate | Configurable | | Configure |

Configuración para push

Ahora pulsamos sobre el botón que pone Configure en el certificado SSL de desarrollo (Development Push SSL Certificate). Al pulsar sobre el botón nos aparecerá un

asistente que tenemos que seguir para generar el certificado, el cual después utilizaremos para el servidor, o en nuestro caso, Urban Airship. Por tanto simplemente seguimos los pasos que nos indica el asistente...



Certificado SSL

Una vez que nos hayamos descargado el certificado SSL generado en el asistente (`aps_development.cer`) lo agregamos a nuestro llavero simplemente pulsando sobre el. Una vez hecho esto ya podemos "subirlo" a la cuenta de Urban Airship, pero primero tenemos que crear una app, para ello debemos de pulsar sobre *Select an application* > *Create an app*. Una vez dentro rellenamos todo el formulario adjuntando la exportación del certificado de Apple Developer iOS Push Services: `es.ua.jtech.ejemplopush` desde la aplicación de llaveros del Mac (este fichero debe de tener extensión `p12`).



Apple Development IOS Push Services: es.ua.jtech.ejemplopush

Emitido por: Apple Worldwide Developer Relations Certification Authority

Caduca: martes 5 de marzo de 2013 10:17:40 Hora estándar de Europa Central

✓ Este certificado es válido

| Nombre | Clase | Caducidad |
|--|---------------|---------------------|
| Apple Development IOS Push Services: es.ua.jtech.ejemplopush | certificado | 05/03/2013 10:17:40 |
| Javier Aznar | clave privada | -- |


Certificado SSL en el llavero del Mac

Completamos el formulario de nueva aplicación:

| | |
|--|--|
| Application name | <input type="text" value="Ejemplo Notificaciones Push"/> |
| Application Icon | <input type="button" value="Seleccionar archivo"/> No se ha ...n archivo |
| Application Mode | <input type="button" value="Development - connecting to testing servers."/> |
| Rich Push Enabled | <input type="checkbox"/> |
| Push Notifications Support | <input checked="" type="checkbox"/> |
| Allow push from the device | <input type="checkbox"/> What's this? |
| Apple push certificate | <input type="button" value="Seleccionar archivo"/> CertificadoPush.p12 |
| Certificate password | <input type="password" value="...."/> |
| Push debug mode | <input checked="" type="checkbox"/> What's this? |
| BlackBerry Username | <input type="text"/> |
| | <small>RIM sometimes calls this the Application ID or serviceid.</small> |
| BlackBerry Password | <input type="password"/> |
| | <small>Per-application password, found in the email from RIM and on their push system dashboard</small> |
| BlackBerry Push URL | <input type="text" value="https://pushapi.eval.blackberry.com/"/> |
| | <small>The base Push URL given by RIM for your application, e.g. "https://cp290.pushapi.na.blackberry.com/",</small> |
| Android Package | <input type="text"/> |
| C2DM Authorization Token | <input type="text"/> |
| In-App Purchase and Subscriptions | <input type="checkbox"/> |
| | <input type="button" value="Create your app"/> |
| | <small>* means required.</small> |

Configuración push en Urban Airship

¡Ya tenemos la aplicación creada en los servidores de Urban Airship!. Sin cerrar el panel de control de Urban Airship podemos empezar a programar lo que es el código de las notificaciones push en nuestra aplicación de ejemplo.

| | |
|--|---|
|  Ejemplo Notificaciones Push | |
| Application Key | shEh_-5OQRC30m0dKxU5HQ |
| Application Secret | 0KJ9b540RJO6JSwuVod4EA (hide) |
| Application Master Secret | _TWS950OS6CjgqpomuCBBw (hide) |
| Application Mode | Development, connecting to test servers. |
| Push Notifications support | In development, connecting to sandbox push servers. |
| Push certificate status | Ready for use |
| Push certificate bundle ID | es.ua.jtech.ejemplopush |
| Push debug mode | On |
| Allow push from device | off |
| BlackBerry Username | Not set! |
| Android Package | Not set! |
| C2DM Auth Token | Not set! |
| Device Tokens | 0 |
| Active Device Tokens | 0 |
| Active Users (month) | 0 |
| Pushes Sent (month) | 0 |
| Pushes Sent (all time) | 0 |
| Subscriptions (month) | 0 |
| In-App Purchase support | Disabled. No In App Purchase integration. |
| Rich Push Enabled | No |
| Subscriptions Enabled | Yes |
| Statistics not in real time. You can also delete this app . | |

Notificaciones push en Urban Airship

2.3. Programación de la aplicación de ejemplo

En los puntos anteriores hemos realizado toda la preparación necesaria tanto en el portal de desarrolladores de Apple como en el sitio Urban Airship (nuestro servidor) para la implementación de las notificaciones push. Ahora vamos a programar una aplicación muy

sencilla que reciba las notificaciones que enviamos desde el servidor de Urban Airship.
¡Vamos a por ello!

Nota

Las notificaciones push **sólo** funcionarán en dispositivos reales (iPhone o iPad), no en el simulador de XCode.

Comenzamos creando una nueva aplicación en XCode usando la plantilla Single View Application. Los datos de la aplicación serán los siguientes:

- Product name: ejemplopush
- Company Identifier: es.ua.jtech
- Class prefix: UA
- Device family: iPhone
- Seleccionar sólo "Use Automatic Reference Counting", el resto de opciones las dejamos desmarcadas.

Atención

El valor del campo Product Name debe de ser exactamente igual que el "appname" del Bundle Identifier que hemos especificado en al crear el App ID en el iOS Provisioning Portal.

Modificamos la vista principal UAViewController.xib añadiéndole un fondo y un *Label*. Una vez hecho esto vamos a abrir el fichero UAppDelegate.m y añadiremos el siguiente código al principio del método didFinishLaunchingWithOptions:

```
[[UIApplication sharedApplication]
registerForRemoteNotificationTypes:(UIRemoteNotificationTypeBadge |
                                   UIRemoteNotificationTypeSound |
                                   UIRemoteNotificationTypeAlert)];
```

Mediante el código anterior estamos indicando al dispositivo que queremos registrarlo en los servidores de Apple para que permita recibir notificaciones push de los tres tipos (un número en el icono de la app, un sonido o un texto de alerta).

Si el usuario acepta podremos obtener un token de su dispositivo el cual utilizaremos para enviarle las notificaciones push, por tanto ese token deberemos de almacenarlo en algún sitio: base de datos de nuestro servidor o en nuestro caso, en Urban Airship. El código para obtener el token del dispositivo deberemos de escribirlo dentro del método didRegisterForRemoteNotificationsWithDeviceToken del protocolo de UIApplication:

```
- (void)application:(UIApplication *)application
didRegisterForRemoteNotificationsWithDeviceToken:(NSData
*)deviceToken {
    NSString *deviceTokenStr = [NSString stringWithFormat:@"%X",
                               [deviceToken description]
                               stringByReplacingOccurrencesOfString:@"<"
withString: @""]
```

```

        stringByReplacingOccurrencesOfString: @">" withString:
@""]
        stringByReplacingOccurrencesOfString: @" " withString:
@""]];

    NSLog(@"deviceToken: %@", deviceTokenStr);

    // Este token deberemos de enviarlo a nuestro servidor o a Urban
    Airship usando su
    // API...
}

```

Esto es lo único que deberemos de programar para que nuestra aplicación reciba notificaciones push. Ahora podemos probarlo para comprobar que funciona correctamente, para ello seguiremos los siguientes pasos:

Nota

Deberemos tener configurado un perfil adecuado asociado al certificado de push que hemos generado. Deberemos de compilar nuestra aplicación en el dispositivo usando este perfil, si no, no aparecerán las notificaciones push.

1. Arrancamos la aplicación y nos aparecerá una alerta preguntando si aceptamos recibir notificaciones. Aceptamos.



Confirmación recepción de notificaciones push

2. Una vez que hemos aceptado recibir notificaciones push nos aparecerá el token del dispositivo en la consola. Lo copiamos en el portapapeles porque lo necesitaremos para el siguiente paso.

```
[Switching to process 7171 thread 0x1c03]  
[Switching to process 7171 thread 0x1c03]  
2012-03-05 11:10:36.577 ejemplopush[2839:707] deviceToken: 6150a52eacd929266ae3fc8610fefa0c8e3941dfc8cf375e609e0f1f80c7b477
```

Token del dispositivo

3. Ahora vamos a crear una notificación push y a enviarla desde el panel de control de Urban Airship. Para ello accedemos al panel de control, entramos en nuestra aplicación de ejemplo y seleccionamos desde el menu de la izquierda *Push > Test Push Notifications*. Completamos el formulario con los siguientes datos y pulsamos en "Send it!":

| | iOS | Android | BlackBerry |
|---------------------|--|---------|------------|
| Device token | 6150a52eacd929266ae3fc8610fefa0c8e3941dfc | | |
| Alias | Ejemplo | | |
| Badge | 1 | | |
| Alert | Hola a todos!!! | | |
| Sound | chime | | |
| Payload | {"aps": {"badge": 1, "alert": "Hola a todos!!!", "sound": "chime"}, "device_tokens": ["6150a52eacd929266ae3fc8610fefa0c8e3941dfc8cf375e609e0f1f80c7b477"], "aliases": ["Ejemplo"]} | | |
| <div>Send it!</div> | | | |

Crear notificación en Urban Airship

¡Funciona! En aproximadamente un segundo nos aparecerá la notificación push en la pantalla del dispositivo.

Con esto ya tenemos el funcionamiento básico de las notificaciones push haciendo uso de un servidor de pago como Urban Airship. Este servidor además ofrece una API que nos permite acceder a todos sus servicios desde otra plataforma como podría ser un servidor propio en PHP, lo cual facilita mucho el trabajo. Es conveniente leerse la documentación de Urban Airship (<http://urbanairship.com/docs/>) la cual nos aclarará muchas dudas y nos proporcionará nuevas ideas y mejoras de funcionamiento en nuestra aplicación.

