# app iOS para buscar negocios locales

*Memoria Final*

alumno: Jorge Tercero Calderón
consultor: Jordi Ceballos Villach

# 1. Introducción

## 1.1. Plataforma tecnológica

iOS es el sistema operativo para dispositivos móviles de Apple. Está diseñado para funcionar en los dispositivos fabricados por la propia Apple, y restringido a estos.

Esa restricción se puede identificar como una debilidad y, al mismo tiempo, como una fortaleza.

Es una debilidad en cuanto a que se limita el alcance del sistema operativo, puesto que existen muchos dispositivos y fabricantes a los que no tienen acceso iOS ni las aplicaciones hechas para iOS.

En cambio es una fortaleza porque se reduce mucho la fragmentación del mercado: solo un número limitado de modelos pueden correr nuestra aplicación, lo que simplifica el desarrollo y las pruebas, pero ese limitado número de modelos no se traduce en pocos dispositivos. El número de iPhones, iPods touch e iPads es enorme y sigue creciendo.

# 2. Definición general del proyecto

La aplicación proporciona al usuario acceso a la información disponible online sobre negocios locales. Esa información se ofrece de un modo reactivo, en respuesta a las demandas del usuario; y de un modo proactivo: la aplicación intenta prever las necesidades y los deseos del usuario para sugerirle negocios que le sean útiles en un momento dado.

De cada negocio utilizaremos esta información:

- Información objetiva: datos para localizar al negocio, independientes de la opinión del usuario: nombre, dirección, teléfono, horarios, etc.
- Opiniones: reseñas escritas por clientes de ese negocio.
- Fotos: fotos tomadas por los clientes de ese negocio o por los propietarios del negocio.

Obtendremos el máximo partido de esa información gracias a tres características de los *smartphones*:

- Localización: la app sabe ubicar el teléfono, y dado que cuenta con las coordenadas espaciales de los negocios, puede filtrar u ordenar los negocios por distancia al usuario.
- Base de datos: El teléfono pone a disposición de la app una base de datos, que usaremos para guardar las decisiones del usuario y así poder ofrecerle resultados personalizados.
- Dispositivos personales: Podemos asumir que el teléfono es propiedad exclusiva de una persona, así que no necesitaremos sistemas de credenciales para identificar al usuario. Toda la actividad que se genere en ese teléfono la consideraremos asociada a la misma

persona, lo que simplifica mucho la experiencia de usuario, que no necesita contraseña, ni otros sistemas de autenticación.

## 2.1. Aplicaciones existentes

Existen otras apps que buscan resolver el mismo problema y, de hecho, nuestro proyecto se apoyará en ellas para obtener los datos. Destacamos las siguientes:

### 2.1.1. Yelp

Se puede considerar el líder mundial, su implantación en España es relativamente reciente, pero ya han conseguido una comunidad activa de usuarios que generan contenidos de calidad. Desde una perspectiva técnica es la app más sólida.

### 2.1.2. Foursquare

Comenzaron siendo una app para compartir tu posición con tus amigos (“estoy en el bar Pepe”) y fueron pioneros en el uso de técnicas de *gamificación*. Recientemente han virado su rumbo para convertirse en una empresa de acceso a información de negocios locales. Es la app que tiene un diseño más innovador.

### 2.1.3. 11870

Pionera en España de las opiniones de usuarios sobre negocios locales en internet. Cuenta con una sólida comunidad de usuarios muy activos y una base de datos grande y de calidad. La app para iOS se ha quedado algo anticuada. 11870 pone a disposición de terceras partes su base de datos a través de una API[1], que será nuestra fuente principal de información.

Si bien no pretendemos competir con estas apps, sí creemos que podemos ofrecer una experiencia mejor para algunos casos de uso. Su principal debilidad estriba en que pretenden condensar mucha funcionalidad en una sola app: buscar negocios, publicar opiniones, hacer amigos, hacer *check in*, etc.

Nosotros cubriremos solo parte de esa funcionalidad. Al concentrar todo el diseño y la tecnología en unas pocas funcionalidades, la experiencia final puede competir o mejorar a las apps existentes.

# 3. Objetivos del trabajo

El resultado del proyecto será una app para iPhone. La app podría funcionar correctamente en iPad también, pero creemos que es mejor excluirlo porque el caso de uso más habitual será el de buscar negocios alrededor mío con el teléfono.

[1] https://11870.com/api

## 3.1. Requerimientos funcionales

### 3.1.1. Localización

Al iniciar la aplicación el sistema localiza al usuario. La localización es visible de alguna forma para el usuario, que puede cambiarla con el objeto de buscar negocios en una ubicación distinta de la suya.

### 3.1.2. Resultados de búsqueda

La app proporciona una caja de búsqueda y usa la ubicación del usuario para ordenar y/o filtrar los resultados.

Una búsqueda da lugar a un conjunto de negocios que se muestran como un listado. De cada negocio deben aparecer unos pocos datos que permitan identificarlo (nombre, dirección) y una foto.

Los resultados de búsqueda se pueden ver también en un mapa.

El listado implementa paginación, cargando más resultados cuando el usuario hace *scroll* hasta el final de la pantalla.

### 3.1.3. Ficha de negocio

El usuario puede hacer *tap* en uno de los negocios de los resultados de búsqueda, eso le lleva a una pantalla con los detalles del negocio, esa pantalla tiene que incluir al menos los siguientes elementos:

- Nombre
- Dirección
- Mapa
- Teléfono
- Opiniones de usuarios
- Fotos

### 3.1.4. Sugerencias

El sistema intentará sugerir al usuario los negocios que más le interesen en cada momento. Para ello utilizará tres tipos de señales:

- Geográficas: negocios cerca de donde se encuentra el usuario.
- Temporales: negocios adecuados para ese día/hora, por ejemplo, un restaurante al mediodía.
- Personales: cada vez que el usuario hace *tap* en un negocio lo consideraremos un “voto”, creando así una lista de favoritos que nos ayudará a personalizar la experiencia.

## 3.2. Requerimientos no funcionales

### 3.2.1. Rendimiento

La app debe ser y parecer rápida. Intentaremos hacer la mayoría de las operaciones costosas en *background* para que la app siempre responda a las entradas del usuario. Asimismo intentaremos adelantarnos a las necesidades del usuario para que cuando pida datos los tengamos ya cargados, por ejemplo, si un usuario está viendo una foto, cargamos ya la siguiente del álbum aunque el usuario no lo ha pedido aún.

### 3.2.2. Estabilidad

La app debe ser robusta y estable, y funcionar bajo cualquier condición. En ningún caso se debe colgar ni cerrar espontáneamente. En dispositivos móviles tenemos que anticipar que la app puede usarse bajo circunstancias muy diversas.

### 3.2.3. Ortogonalidad

Haremos uso de la API de 11870, de Apple Maps y de los servicios de localización de Apple, pero tenemos que limitar las dependencias a ellos a lo más mínimo. Si uno de esos servicios no está disponible o no ofrece garantías de calidad la app debe dar una buena respuesta a esas circunstancias, no puede quedarse colgada ni cerrarse.

# 4. Planificación del proyecto

## 4.1. Recursos Humanos

El peso del diseño e implementación del proyecto recae en el alumno, con la supervisión del consultor.

## 4.2. Hardware

Contamos con un ordenador para hacer el desarrollo y las pruebas en simulador, y un dispositivo para probar la app en real.

| | hardware | sistema operativo |
|---|---|---|
| Estación de trabajo | MacBook Pro 13 inch mid 2010 | OS X Yosemite 10.10.2 |
| Dispositivo de pruebas | iPhone 5S | iOS 8.1 |

## 4.3. Software

La implementación la haremos con el software de desarrollo de Apple instalado en la estación de trabajo, usando el lenguaje de programación Swift. La memoria y la documentación los haremos con herramientas de Google Drive en la nube.

| | |
|---|---|
| Desarrollo | XCode 6.1 |
| Memoria | Google Docs |
| Repositorio de software | Git en Bitbucket |

## 4.4. Calendario

Basándonos en el calendarios de entrega de las PECs proponemos este calendario:

| tarea | inicio | fin | duración | descripción |
|---|---|---|---|---|
| PEC1 | 3/03/2015 | 11/03/2015 | 8 | comienzo de la memoria y planificación del proyecto |
| PEC2 | 12/03/2015 | 8/04/2015 | 27 | diseño |
| casos de uso | 12/03/2015 | 17/03/2015 | 5 | recopilación y descripción de casos de uso |
| diseño de pantallas | 18/03/2015 | 26/03/2015 | 8 | esbozos de las pantallas |
| diagrama de clases | 27/03/2015 | 29/03/2015 | 2 | definición de las clases Swift que compondrán la app |
| diseño de DB | 30/03/2015 | 2/04/2015 | 3 | diseño de tablas de base de datos |
| redacción | 3/04/2015 | 8/04/2015 | 5 | puesta en común de los productos creados en las anteriores fases |
| PEC3 | 9/04/2015 | 20/05/2015 | 41 | implementación |
| API 11870 | 9/04/2015 | 14/04/2015 | 5 | comunicación con la API de 11870 |

| | | | | |
|---|---|---|---|---|
| DB | 15/04/2015 | 20/04/2015 | 5 | implementación del modelo de datos |
| principal | 21/04/2015 | 26/04/2015 | 5 | pantalla de entrada a la app |
| buscador | 27/04/2015 | 3/05/2015 | 6 | buscador y resultados de búsqueda |
| mapa resultados | 4/05/2015 | 5/05/2015 | 1 | mapa con los resultados de búsqueda |
| sugerencias | 6/05/2015 | 11/05/2015 | 5 | sugerencias al usuario |
| ficha de sitio | 12/05/2015 | 16/05/2015 | 4 | detalle de negocio |
| mapa negocio | 17/05/2015 | 20/05/2015 | 3 | ubicación de un negocio en un mapa |
| Entrega final | 21/05/2015 | 21/06/2015 | 31 | preparación de la memoria y la presentación |
| memoria | 22/05/2015 | 13/06/2015 | 22 | redacción de la memoria |
| presentación | 14/06/2015 | 21/06/2015 | 7 | preparación de la vídeo presentación |

Revisión a 20 de Mayo. Marcamos en verde las tareas completadas, en rojo las no completadas, y en amarillo las parcialmente completadas.

| **tarea** | **inicio** | **fin** | **duración** | **descripción** |
|---|---|---|---|---|
| PEC1 | 3/03/2015 | 11/03/2015 | 8 | comienzo de la memoria y planificación del proyecto |
| PEC2 | 12/03/2015 | 8/04/2015 | 27 | diseño |
| casos de uso | 12/03/2015 | 17/03/2015 | 5 | recopilación y descripción de casos de uso |
| diseño de pantallas | 18/03/2015 | 26/03/2015 | 8 | esbozos de las pantallas |
| diagrama de clases | 27/03/2015 | 29/03/2015 | 2 | definición de las clases Swift que compondrán la app |

| | | | | |
|---|---|---|---|---|
| diseño de DB | 30/03/2015 | 2/04/2015 | 3 | diseño de tablas de base de datos |
| redacción | 3/04/2015 | 8/04/2015 | 5 | puesta en común de los productos creados en las anteriores fases |
| PEC3 | 9/04/2015 | 20/05/2015 | 41 | implementación |
| API 11870 | 9/04/2015 | 14/04/2015 | 5 | comunicación con la API de 11870 |
| DB | 15/04/2015 | 20/04/2015 | 5 | implementación del modelo de datos |
| principal | 21/04/2015 | 26/04/2015 | 5 | pantalla de entrada a la app |
| buscador | 27/04/2015 | 3/05/2015 | 6 | buscador y resultados de búsqueda |
| mapa resultados | 4/05/2015 | 5/05/2015 | 1 | mapa con los resultados de búsqueda |
| sugerencias | 6/05/2015 | 11/05/2015 | 5 | sugerencias al usuario |
| ficha de sitio | 12/05/2015 | 16/05/2015 | 4 | detalle de negocio |
| mapa negocio | 17/05/2015 | 20/05/2015 | 3 | ubicación de un negocio en un mapa |
| Entrega final | 21/05/2015 | 21/06/2015 | 31 | preparación de la memoria y la presentación |
| memoria | 22/05/2015 | 13/06/2015 | 22 | redacción de la memoria |
| presentación | 14/06/2015 | 21/06/2015 | 7 | preparación de la vídeo presentación |

La implementación está avanzada, pero nos hemos retrasado en algunos puntos, el que tiene mayor peso es el de guardado y gestión de favoritos, que implica acceso a la base de datos.

Las tareas pendientes se tendrán que acomodar en el tiempo restante del proyecto, conviviendo con las que ya teníamos previstas. Consideramos prioritario finalizar las tareas ya comenzadas, que darán lugar a un proyecto completo, y dejaremos para el final la funcionalidad de favoritos.

Así, el calendario de la última fase queda como sigue:

| tarea | inicio | fin | duración | descripción |
|---|---|---|---|---|
| sugerencias 1 | 21/05/2015 | 27/05/2015 | 6 | sugerencias al usuario |
| ficha de sitio | 28/05/2015 | 2/06/2015 | 5 | detalle de negocio |
| favoritos | 3/06/2015 | 8/06/2015 | 5 | preparación de la memoria y la presentación |
| memoria | 9/06/2015 | 15/06/2015 | 6 | redacción de la memoria |
| presentación | 16/06/2015 | 21/06/2015 | 5 | preparación de la vídeo presentación |

# 5. Sumario de productos obtenidos

El proyecto concluirá con tres entregables:

- Aplicación iOS: app instalable en un iPhone.
- Memoria: documento describiendo el proyecto y documentando la app.
- Presentación: vídeo explicando el proyecto.

# 6. Riesgos del proyecto

El proyecto se puede ver afectado por problemas propios del proyecto o ajenos al mismo, veamos una relación de riesgos y de cómo podemos gestionarlos.

| riesgo | descripción | prevención y gestión |
|---|---|---|
| exceso de optimismo | estimaciones demasiado optimistas | detección temprana de desviaciones y gestión de expectativas acorde |
| fallos de hardware | avería en la estación de trabajo | todo en la nube: todo el código en Git, toda la documentación en Google Drive o Dropbox |
| tiempo menguante | circunstancias ajenas al proyecto reducen el tiempo disponible | poner el foco en conseguir lo antes posible un *MVP*[2], que iremos mejorando a medida que las previsiones se cumplan. |

[2] Minimum Viable Product: producto más simple posible que contenga la funcionalidad necesaria. No debería ser el entregable final, pero concentrarnos en obtener el MVP aumenta las posibilidades de tener toda la funcionalidad cubierta.

# 7. Documentación y análisis

## 7.1. Requerimientos funcionales

### 7.1.1. Descripción básica del funcionamiento

El proyecto incluye funcionalidades que nos proporciona 11870 a través de su API y otras que añadimos nosotros.

Funcionalidades derivadas directamente de 11870:

- Sugerencias: el usuario nos ha autorizado a acceder a la localización del teléfono, y la usamos para buscar negocios populares en 11870 y con contenidos de calidad cercanos al usuario y se los sugiere.
- Buscador de negocios locales: el usuario introduce un texto de búsqueda, que lanzamos contra la API de 11870 para obtener negocios que coincidan. Las búsquedas pueden ser genéricas (“pizzerías”) o concretas (“bar manolo”).
- Negocios alrededor mío: usamos la localización para pedir a 11870 negocios alrededor de unas coordenadas concretas.
- Detalle de negocio: el usuario ha seleccionado un negocio del que quiere conocer más detalles, le mostramos los datos generales (nombre, dirección, teléfono, etc.) y las opiniones de los usuarios.
- Detalle de opinión: el usuario quiere ver más información de una opinión en concreto. Le mostramos el nombre y la foto del usuario que la ha creado, y el texto completo de la opinión.

A partir de los datos que conseguimos de 11870, ofrecemos otras funcionalidades:

- Cómo llegar: ubicamos el negocio en un mapa.
- Favoritos: guardamos en una base de datos los negocios favoritos del usuario.

### 7.1.2. Seguridad

La aplicación envía datos potencialmente sensibles por la red:

- búsquedas del usuario: la mayoría de las búsquedas serán neutras, no serán particularmente sensibles, pero algunas pueden ser más delicadas, y, en todo caso, el usuario tiene derecho a que su actividad permanezca confidencial.
- Localización del usuario: es la información más sensible que manejamos.

A diferencia de otras aplicaciones, nosotros no solicitamos al usuario que se identifique en ningún momento, de forma que la información anterior sería difícil asociarla a una persona concreta, no obstante, debemos proteger la privacidad de nuestro usuario, para ello utilizamos el protocolo HTTPS en todas nuestras comunicaciones con la API de 11870.

Gracias a HTTPS conseguimos protección en estos tres aspectos:

- Confidencialidad: la información viaja encriptada y es ininteligible para una tercera parte que la intercepte.
- Integridad: nos aseguramos de que nadie ha modificado la información que viaja por la red.
- Identidad: nos aseguramos de que el servidor con el que hablamos es quien dice ser.

## 7.2. Casos de uso

Los casos de uso identifican cada una de las funcionalidades, especificando, de forma aislada al resto de casos de uso, los pasos en los que se divide cada interacción entre el usuario y el sistema. La aplicación solo contempla un único actor que es el usuario de la misma.

| **Identificador** | **CU-001** |
|---|---|
| Nombre | Inicio |
| Prioridad | Alta |
| Descripción | El usuario inicia la aplicación haciendo tap en su icono. |
| Actores | Usuario |
| Precondiciones | La app está instalada en el dispositivo |
| Iniciado por | Usuario |
| Flujo | El sistema operativo inicia la app<br>La app solicita la ubicación del usuario al S.O.<br>La app obtiene la hora del sistema<br>En función de la hora decide qué negocios serán más interesantes para el usuario.<br>La app solicita negocios cercanos interesantes a la API de 11870<br>La app obtiene los favoritos de la DB<br>La app muestra la pantalla *home* con los negocios recomendados y un acceso a los favoritos, si los hay. |
| Postcondiciones | La app está en marcha<br>La app muestra la pantalla *home* |
| Notas | El acceso a la API de 11870 se hace a través de internet, si no hay acceso disponible, o si éste es muy lento, la app debe reconocerlo pronto y mostrar una home screen adaptada a estas circunstancias.<br>Del mismo modo, si no podemos obtener la localización del usuario, debemos gestionarlo con elegancia, dejando acceso a las funcionalidades no dependientes de la localización. |

| **Identificador** | **CU-002** |
|---|---|
| Nombre | Búsqueda con resultados |
| Prioridad | Alta |
| Descripción | El usuario hace una búsqueda de texto libre. |
| Actores | Usuario |
| Precondiciones | La app está en marcha<br>La app muestra la pantalla *home* |
| Iniciado por | Usuario |
| Flujo | La app lanza la cadena de búsqueda del usuario contra la API de 11870, limitando la respuesta a 10 negocios.<br>Si la app tiene las coordenadas del dispositivo, las incluye como parámetros de la búsqueda.<br>La app espera la respuesta de la API.<br>La API devuelve una lista de uno o más negocios.<br>Se muestra la pantalla *serp* con los resultados de la búsqueda y la caja de búsqueda rellena con el texto que introdujo el usuario. |
| Postcondiciones | La app está en marcha.<br>La app muestra la pantalla *serp*, rellena con los resultados de búsqueda. |
| Notas | En caso de no disponer de acceso a internet, la funcionalidad es imposible de completar, así que hay que mostrarle al usuario claramente que no se puede responder a su petición.<br>Si la búsqueda no devuelve ningún negocio, mostraremos la misma pantalla *serp*, pero con la lista de negocios vacía y un mensaje invitando al usuario a refinar la búsqueda.<br>Debe quedar perfectamente claro si la lista vacía es consecuencia de un problema de conectividad o de que no hemos encontrado negocios para esa búsqueda, de forma que el usuario pueda actuar en consecuencia. |

| **Identificador** | **CU-003** |
|---|---|
| Nombre | Cambio de página de búsqueda |
| Prioridad | Alta |
| Descripción | El usuario hace scroll hasta la última posición del listado. |
| Actores | Usuario |
| Precondiciones | La app está en marcha<br>La app muestra la pantalla *serp* rellena con 10 negocios<br>El usuario ha hecho scroll de forma que que queda visible el último negocio. |
| Iniciado por | Usuario |
| Flujo | La app repite la misma búsqueda que devolvió los resultados actuales, pero especificando que quiere los negocios en las posiciones 11 a 20, o las que correspondan en función de qué página estemos cargando.<br>La app espera la respuesta de la API.<br>La API devuelve una lista de uno o más negocios.<br>Se muestra la pantalla *serp* con los resultados de la nueva búsqueda añadidos al final de la lista de negocios que teníamos. La caja de búsqueda sigue rellena con el texto que introdujo el usuario. |
| Postcondiciones | La app está en marcha.<br>La app muestra la pantalla *serp*, rellena con los 20 primeros resultados de la búsqueda. |
| Notas | En caso de no disponer de acceso a internet, la funcionalidad es imposible de completar, así que hay que mostrarle al usuario claramente que no se puede responder a su petición.<br>Si la búsqueda no devuelve ningún negocio, mostraremos en la posición siguiente del listado un elemento de interfaz indicando que se ha alcanzado el último resultado. |

| **Identificador** | **CU-004** |
|---|---|
| Nombre | Favoritos |
| Prioridad | Alta |
| Descripción | El usuario hace tap en el botón de favoritos. |
| Actores | Usuario |
| Precondiciones | La app está en marcha.<br>La app muestra la pantalla *home.*<br>El usuario tiene al menos un favorito guardado. |
| Iniciado por | Usuario |
| Flujo | El usuario hace tap en el botón “favoritos”.<br>La app recoge los favoritos de la base de datos.<br>La app espera la respuesta de la base de datos.<br>La base de datos devuelve una lista de uno o más negocios.<br>Se muestra la pantalla *favs* con los favoritos del usuario. |
| Postcondiciones | La app está en marcha.<br>La app muestra la pantalla *favs*. |
| Notas | El botón “favoritos” solo se muestra en caso de que haya algún favorito en la base de datos, así que no necesitamos contemplar el caso de que no hayan favoritos. |

| **Identificador** | **CU-005** |
|---|---|
| Nombre | Pantalla de negocio |
| Prioridad | Alta |
| Descripción | El usuario hace tap en un negocio, la app le muestra los detalles del negocio. |
| Actores | Usuario |
| Precondiciones | La app está en marcha.<br>Una de estas tres condiciones:<br>● Pantalla *serp* con al menos un resultado de búsqueda.<br>● Pantalla *home* con al menos un negocio sugerido.<br>● Pantalla *favs* con al menos un negocio favorito. |
| Iniciado por | Usuario |
| Flujo | El usuario hace tap en un negocio.<br>La app recoge el identificador de ese negocio.<br>La app llama a la API De 11870 pidiendo los detalles del negocio con ese identificador.<br>La app espera la respuesta de la API.<br>La API devuelve los detalles de un negocio.<br>Se muestra la pantalla *business* con los detalles de ese negocio. |
| Postcondiciones | La app está en marcha.<br>La app muestra la pantalla *business*. |
| Notas | En caso de no haber conexión a internet la funcionalidad no se puede completar, así que hay que informar claramente al usuario.<br>Puede darse el caso de que el negocio ya no exista en la base de datos de 11870, por ejemplo porque ha cerrado, sobre todo en el caso de los favoritos. Es necesario estudiar la API de 11870 para ver cómo nos informa de esa situación para notificar al usuario y actualizar su lista de favoritos consecuentemente. |

| **Identificador** | **CU-006** |
|---|---|
| Nombre | Mapa |
| Prioridad | Alta |
| Descripción | El usuario hace tap en el mapa de un negocio. |
| Actores | Usuario |
| Precondiciones | La app está en marcha.<br>La app muestra la pantalla *business*.<br>La app tiene las coordenadas de ese negocio. |
| Iniciado por | Usuario |
| Flujo | El usuario hace tap en el mapa de un negocio.<br>La app recoge las coordenadas de ese negocio.<br>La app muestra la pantalla *map*.<br>La app muestra la vista de Apple Maps, solicitando que se muestre el punto correspondiente a las coordenadas del negocio. |
| Postcondiciones | La app está en marcha.<br>La app muestra la pantalla *map* con una “chincheta” sobre la posición del negocio. |
| Notas | En caso de no haber conexión a internet la funcionalidad no se puede completar, así que hay que informar claramente al usuario.<br>Si la API de 11870 no nos ha devuelto las coordenadas de ese negocio, no mostramos el mapa en la pantalla de detalle de negocio, así que el usuario no puede desencadenar esta funcionalidad. |

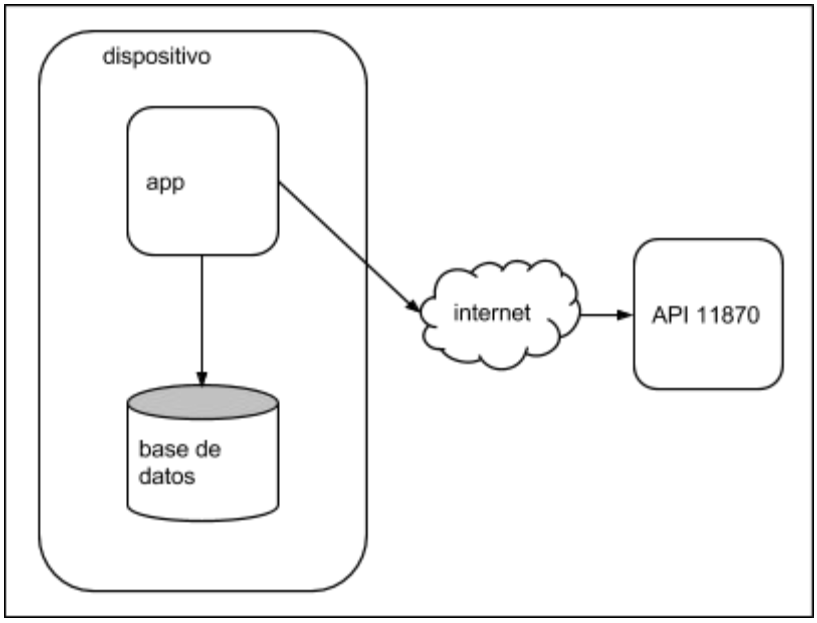
| **Identificador** | **CU-007** |
|---|---|
| Nombre | Galería |
| Prioridad | Alta |
| Descripción | El usuario hace tap en la foto de cabecera de un negocio. |
| Actores | Usuario |
| Precondiciones | La app está en marcha.<br>La app muestra la pantalla *business*.<br>La app tiene al menos una foto de ese negocio. |
| Iniciado por | Usuario |
| Flujo | El usuario hace tap en la foto de cabecera del negocio.<br>La app llama a la API de 11870 pidiendo todas las fotos del negocio.<br>La app espera respuesta de la API.<br>La app recibe una o más fotos del negocio.<br>La app muestra la pantalla *gallery* rellena con las fotos del negocio. |
| Postcondiciones | La app está en marcha.<br>La app muestra la pantalla *gallery* con las fotos. |
| Notas | En caso de no haber conexión a internet la funcionalidad no se puede completar, así que hay que informar claramente al usuario. |

## 7.3. Diseño técnico

A grandes rasgos la app está formada por tres componentes:

- Dispositivo: la app propiamente dicha, instalada en un dispositivo.
- API 11870: obtenemos los datos de negocios locales de la API de 11870[3].
- Base de datos: el usuario puede guardarse sus negocios favoritos. Usaremos la base de datos que proporciona iOS a las apps.

[3] https://sites.google.com/site/api11870/api

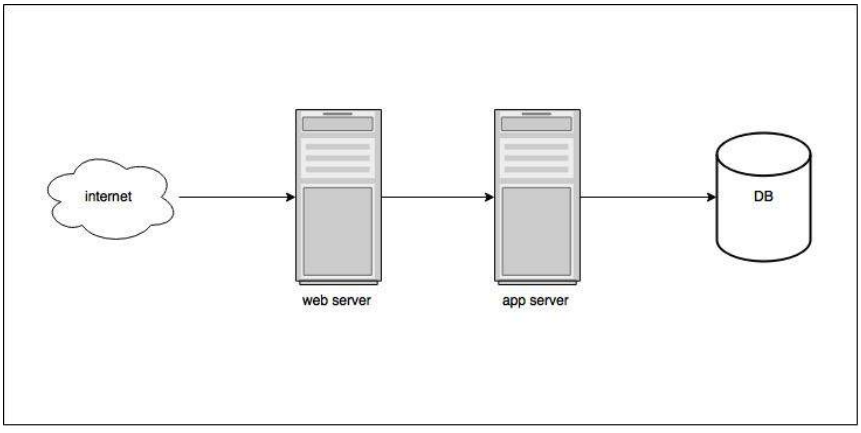



El proyecto sigue el modelo cliente-servidor, donde el servidor es la API de 11870, y el cliente es la aplicación nativa iOS que se conecta al servidor a través de internet.

### 7.3.1. Arquitectura del servidor

Desde el punto de vista de este proyecto el servidor es una caja negra que ofrece una determinada funcionalidad, y no necesitaríamos conocer los detalles de su arquitectura.

No obstante, sabemos que el servicio lo proporcionan servidores Linux ubicados en un centro de datos de Madrid, y las piezas principales son los servidores web, que reciben las peticiones de internet y las pasan a servidores de aplicaciones. Estos se conectan a varios servicios de backend, siendo los más relevantes bases de datos relacionales.



### 7.3.2. Arquitectura del cliente

El cliente es una aplicación iOS que corre sobre iPhone con el sistema operativo iOS 8.1 o superiores.

Se ha decidido excluir el iPad de los dispositivos objetivo por los requerimientos funcionales.

Por otro lado, la versión mínima del sistema operativo viene determinada por la elección del lenguaje Swift, que está disponible solo desde la versión 8 de iOS.

El sistema operativo proporciona numerosos servicios al programador de aplicaciones. Nosotros usaremos dos: Core Data para la persistencia (ver más detalle en el apartado 7.3.4) y los servicios de geolocalización, de lo que nos ocupamos en el apartado 8.

### 7.3.3. Arquitectura lógica

La app se implementa sobre iOS, usando el lenguaje Swift en su versión 1.2, la técnica de los *storyboards* y con el entorno de desarrollo Xcode 6.

El servidor proporciona una API a la que se puede acceder mediante XML o JSON, nosotros hemos escogido JSON por dos razones:

- Concisión: el tamaño de la información que viaja por la red es menor, de forma que el rendimiento de la app es mejor.
- Simplicidad del tratamiento: iOS incorpora librerías para el tratamiento de JSON.

La aplicación hace uso de varios patrones de diseño que veremos a continuación, pero el patrón MVC destaca por encima del resto porque guía el diseño general de la aplicación.

#### 7.3.3.1. Model View Controller

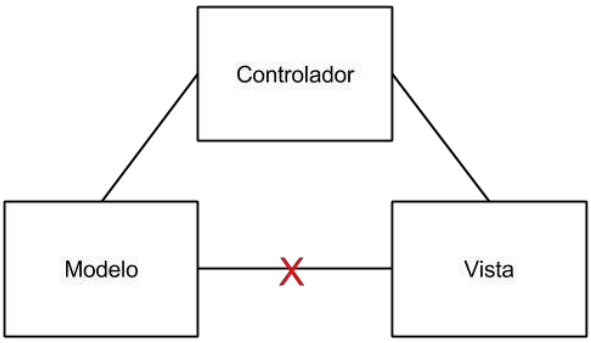
El patrón MVC se ha convertido prácticamente en un estándar en las aplicaciones con interfaz de usuario. Los *frameworks* más populares de aplicaciones web lo implementan, y lo mismo pasa en las aplicaciones nativas.

Apple recomienda que las aplicaciones que se desarrollen en sus plataformas sigan este patrón [4], y de hecho la mayoría de las piezas del sistema requieren que las aplicaciones que los usan toman uno de los roles del patrón.

El patrón MVC separa los componentes de la aplicación en tres roles:

[4] No solo aplicaciones iOS, también las aplicaciones para OSX https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

- Modelo: los objetos de esta capa representan los datos y las operaciones para manipular y procesar esos datos. En nuestro caso los datos provienen de dos fuentes: la base de datos local del dispositivo, y la API de 11870. Los objetos Modelo se comunican con los objetos Controlador, pero no con los objetos Vista.
- Vista: La interfaz gráfica, que implementaremos en un storyboard de Xcode. Los objetos Vista saben cómo pintarse en la pantalla y responden a las acciones de usuario, que son comunicadas a los objetos Controlador.
- Controlador: Actúan de intermediarios entre la interfaz gráfica y el modelo. Tendremos una clase controladora por cada pantalla. Los cambios en el Modelo se comunican a la Vista a través de un objeto Controlador, y viceversa.




### 7.3.3.2. Delegation

Las llamadas a la red o a la base de datos pueden ser lentas o incluso no llegar a completarse, por lo que es conveniente hacerlas asíncronamente. Las peticiones deben hacerse en un *thread* distinto del que gestiona la interfaz de usuario, para que la espera no afecte a la experiencia de uso.

Cuando el *thread* que está accediendo a los recursos de acceso lento finaliza, comunica el resultado al *thread* principal por medio del patrón *delegation*.

Apple usa extensivamente este patrón para comunicar los objetos Vista con el objeto Controlador correspondiente[5].

[5] https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html

### 7.3.4. Base de datos

La base de datos se encarga de guardar los negocios favoritos del usuario. La aplicación utiliza el *framework Core Data*[6] de Apple para definir e interactuar con la base de datos.

Core Data va más allá de ser una librería de acceso a base de datos. El *framework* se integra en el ciclo de vida de los objetos gestionados, lo que incluye la creación, el versionado, las modificaciones, la persistencia y la destrucción. Apple ha creado un modelo que se abstrae de la tecnología de persistencia, y podríamos persistir los datos en sistemas alternativos a las bases de datos relacionales, como sistemas clave-valor o ficheros planos, pero en nuestro caso utilizamos SQLite, que es el SGDB que forma parte del sistema operativo.


Nuestro modelo de datos está compuesto por una sola tabla, llamada Business, que guarda un registro por cada favorito del usuario. Está compuesta por las siguientes columnas:

| nombre | tipo | descripción |
|---|---|---|
| address | String | dirección del negocio |
| latitude | Double | latitud de las coordenadas geográficas del negocio |
| longitude | Double | longitud de las coordenadas geográficas del negocio |

6 https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CoreData/cdProgrammingGuide.html#//apple_ref/doc/uid/TP30001200-SW1

| | | |
|---|---|---|
| locality | String | localidad del negocio |
| name | String | nombre del negocio |
| photoURL | String | URL de la foto principal del negocio |
| slug | String | identificador único del negocio en la DB de 11870 |
| summary | String | texto corto describiendo el negocio |
| telephone | String | teléfono del negocio |

La funcionalidad de favoritos tiene la particularidad de que no necesita conexión a la red, ya que su base de datos reside en el propio dispositivo. Así, podemos acceder a la pantalla de favoritos sin necesidad de tener el dispositivo conectado a Internet.

Cuando diseñamos la base de datos tuvimos en cuenta esta característica, pero tomamos una decisión que, hasta cierto punto, la contradice: guardamos la URL de la foto del negocio, en lugar de guardar el fichero completo con la foto.

Si nos hubiéramos decidido por guardar el fichero completo, la pantalla de favoritos sería 100% funcional sin necesidad de tener conexión a Internet, sin embargo optamos por guardar solo la URL pensando que era un mejor uso de los recursos del usuario, porque la lista de favoritos puede extenderse indefinidamente, y guardar muchas fotos en disco puede ser una carga considerable para el dispositivo. Por otro lado, la pantalla sigue siendo funcional sin conectividad, porque se ve la lista de negocios, aunque no se vean las fotos.

### 7.3.5. Prototipo

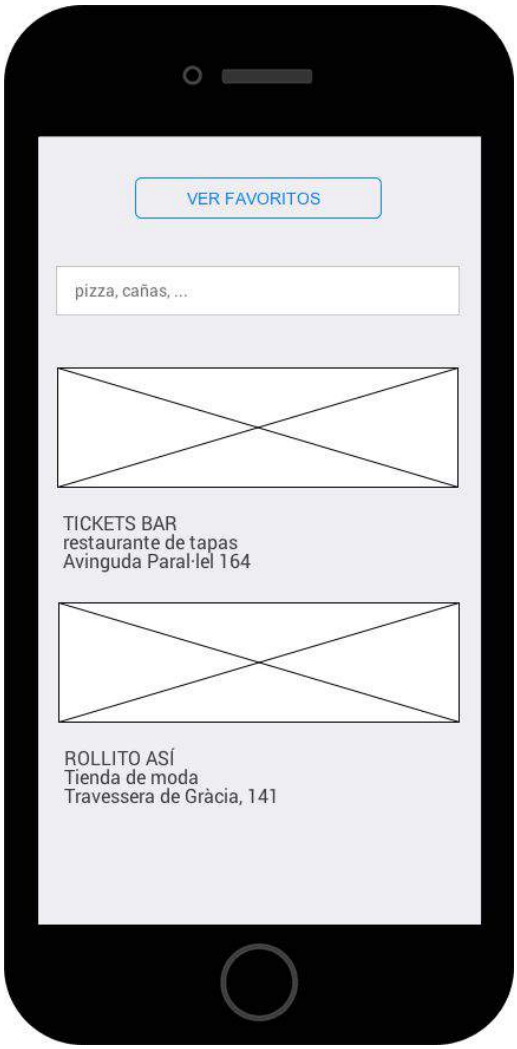
En una fase anterior a la implementación hemos diseñado un prototipo describiendo cada pantalla e interacción de la aplicación. Este prototipo nos ha permitido hacernos una idea global de la funcionalidad de la app.

Los prototipos son herramientas de trabajo muy útiles para discutir las funcionalidades de la aplicación entre las personas implicadas sin incurrir en el esfuerzo, mucho mayor, que supone implementarlas. En este proyecto las personas implicadas hemos sido el alumno y los consultores.

#### 7.3.5.1. Pantalla inicial

En la pantalla inicial encontramos el buscador en una posición central, dos sugerencias de sitios, en los que mostramos una foto grande, para que sean objetivos de contacto grandes y claros.

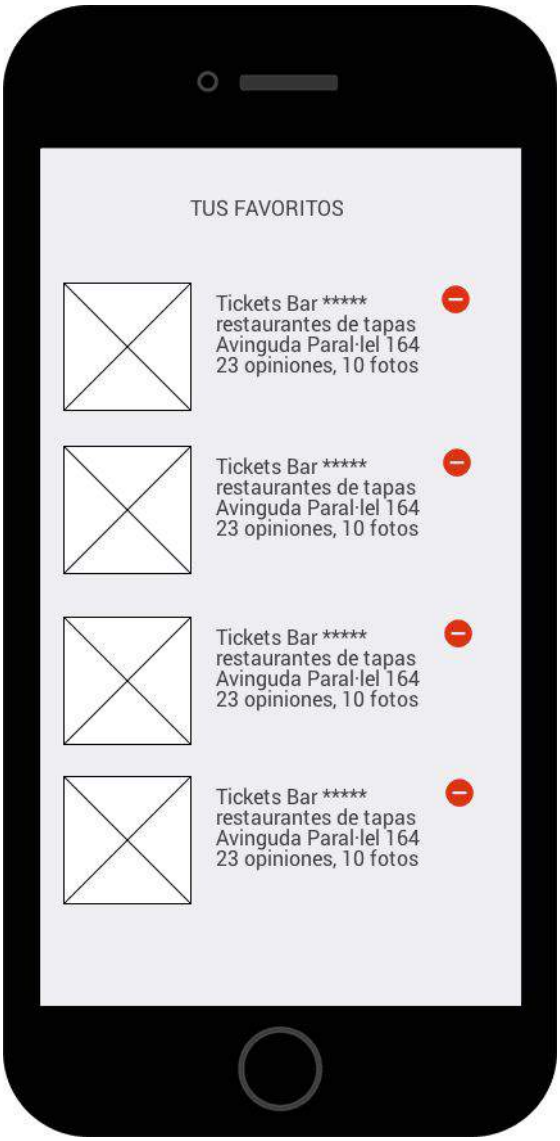
Por último, el botón de acceso a los favoritos del usuario, lo hemos ubicado en un lugar muy visible, pero de más difícil acceso, el usuario debe ver rápida y fácilmente que está accesible, pero pensamos que el acceso a favoritos no es el principal objetivo de esta página, así que no lo hemos ubicado en una posición de más sencillo acceso. Si en el futuro observamos que los usuarios tienden a acceder a favoritos al iniciar la app, podemos colocarlo más abajo, o incluso hacer de favoritos la *home screen*:



#### 7.3.5.2. Favoritos

Esta pantalla muestra los sitios guardados como favoritos, en orden cronológico inverso, primero los últimos añadidos. Al lado de cada negocio, ponemos un icono que desencadena la acción de eliminar el sitio de favoritos.

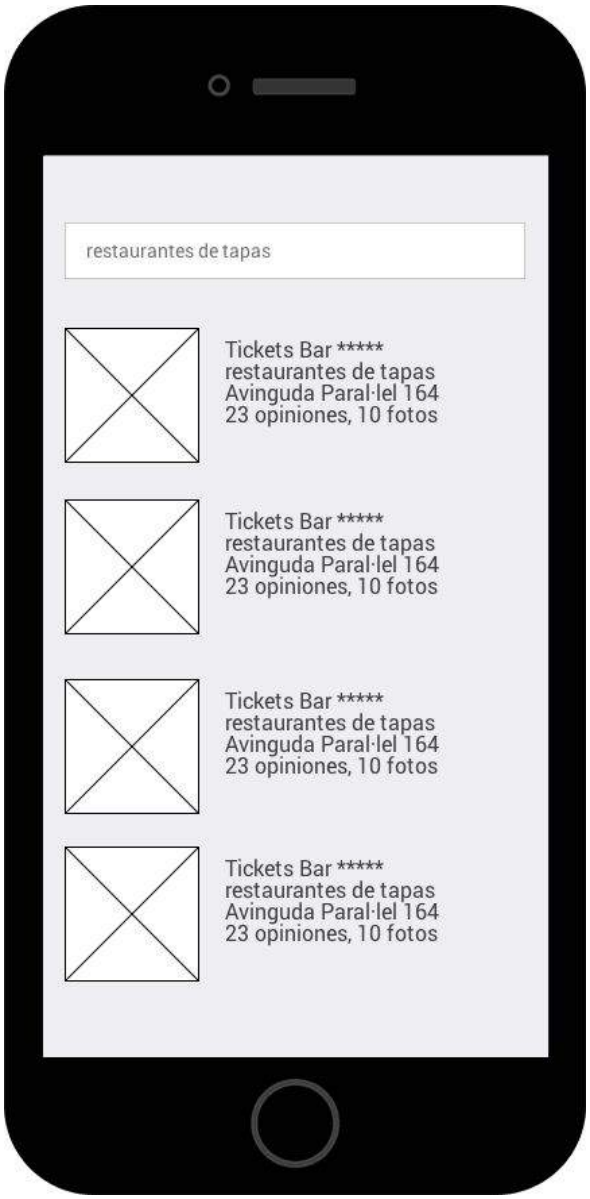
Es posible que los usuarios más activos tengan muchos favoritos, y la presentación como una lista sea inapropiada para encontrar uno concreto. En ese caso deberíamos implementar un buscador restringido a los favoritos, o alguna forma de categorización. En todo caso esperamos a tener ese problema y verlo con usuarios concretos antes de proponer una solución.



#### 7.3.5.3. Resultados de búsqueda

Los resultados de búsqueda se encabezan con la caja de búsqueda rellena con la entrada del usuario, editable, para que se puedan refinar las búsquedas.

A continuación mostramos los sitios, con una “paginación infinita”: cuando el usuario hace *scroll* hasta final de la lista cargamos una página más. El listado tiene el mismo formato que la lista de favoritos, excepto porque en este caso no mostramos el icono para eliminar.



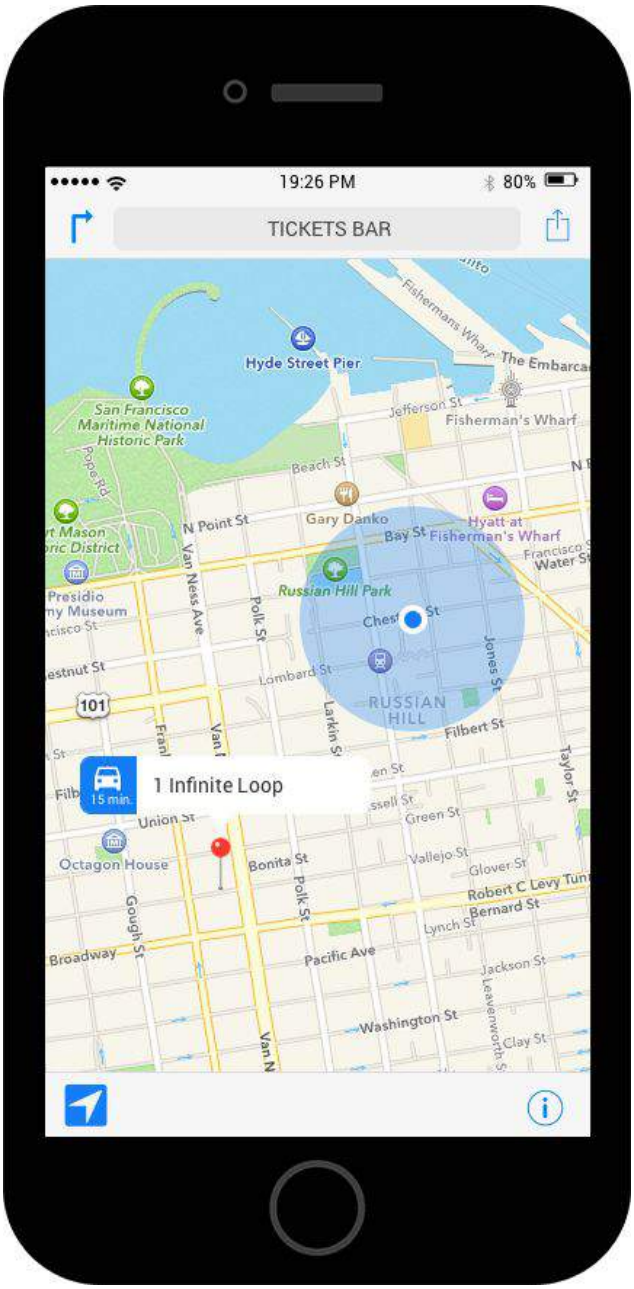
### 7.3.5.4. Negocio

Al tocar en un negocio, tanto desde la pantalla inicial a través de una sugerencia, o de las listas de favoritos y resultados de búsqueda, llegamos a la pantalla de negocio, en la que encontramos los datos básicos del negocio seguidos de una lista de opiniones, paginados de la misma forma que con los resultados de búsqueda: a medida que el usuario alcanza el final de la lista, cargaremos más elementos.

Desde esta pantalla podemos llamar al negocio, guardarlo como favorito, acceder al mapa y ver la galería de fotos.

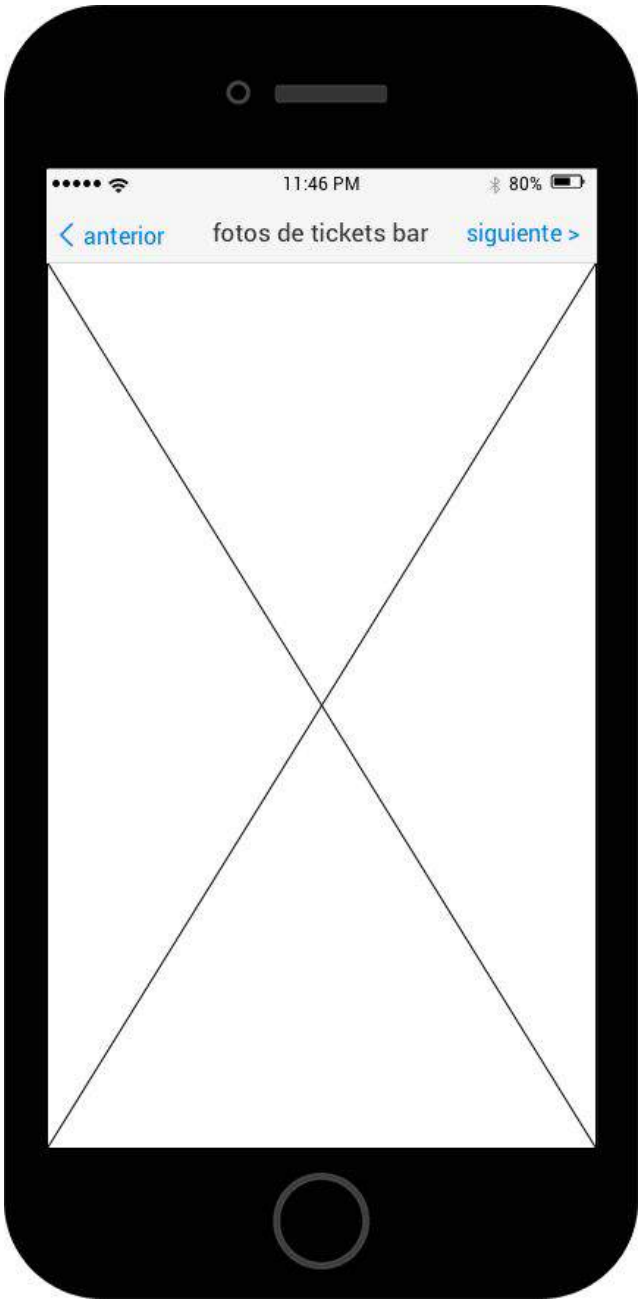
### 7.3.5.5. Mapa

Al tocar el mapa de la ficha de negocio, abrimos Apple Maps, con una “chincheta” en la ubicación del negocio y el indicador de posición del usuario si está cerca del sitio.



### 7.3.5.6. Galería

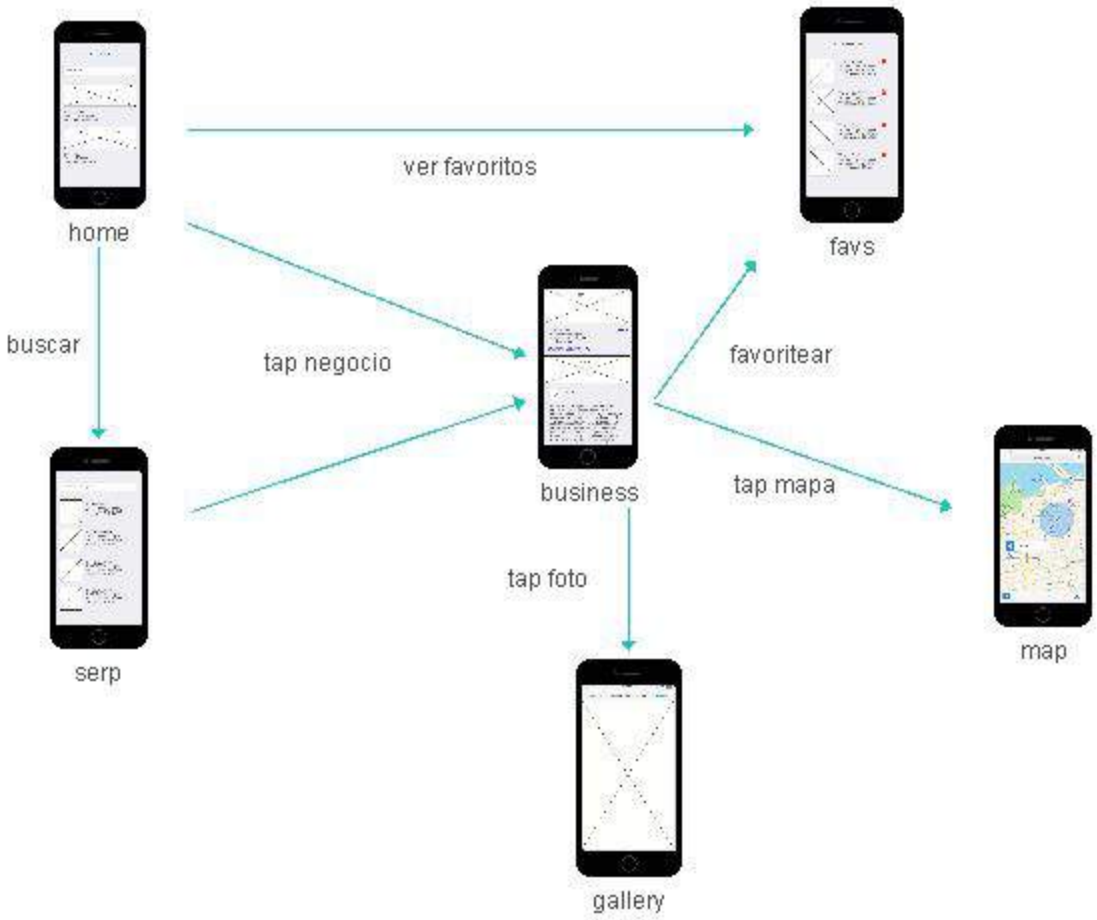
Se accede desde la pantalla del negocio, tocando en la foto de la cabecera. Permite navegar todas las fotos de este negocio.



### 7.3.5.7. Flujo de interacción

Estas pantallas se relacionan entre sí por medio del flujo de interacción que se describe a continuación.

La pantalla central es la del detalle de un negocio, a partir de la que se accede a pantallas accesorias, como el mapa o la galería de fotos.



#### 7.3.5.8. Mejoras posteriores al diseño

Durante la implementación y las pruebas hemos identificado funcionalidades que no estaban bien cubiertas por este diseño:

##### Detalle de opinión

Las opiniones de los usuarios pueden ser bastante largas y no encajaban bien con el modelo tabular propuesto en la pantalla de negocio, así que hemos creado una nueva pantalla a la que se accede al tocar una opinión, en la que se muestran el autor y el texto completo de la opinión.

##### Sitios cercanos

Nuestra idea original era restringir las búsquedas del usuario a sitios cercanos a su posición, pero de esa forma el usuario no podría acceder a sitios lejanos a su posición. Para resolverlo hemos hecho que el buscador funcione sin restricción geográfica. Para acceder a los sitios cercanos hemos habilitado un botón específico en la pantalla inicial. Ese botón se deshabilita en caso de no tener acceso a la localización del usuario.

##### Favoritos vacíos

Necesitábamos definir el comportamiento del botón de favoritos antes de que el usuario se guarde su primer favorito. Hemos decidido deshabilitarlo en caso de no haber ningún favorito en la base de datos.

# 8. Implementación

## 8.1. Consideraciones previas

### 8.1.1. Dispositivo y orientación

Durante el diseño hemos decidido concentrarnos en iPhone y en la orientación vertical.

Excluimos los iPad porque esta es una aplicación cuyos casos de uso están mucho más asociados a un teléfono que a una tablet.

La orientación natural de uso del teléfono es vertical, en nuestro caso la orientación horizontal no aporta nada que no podamos hacer en vertical así que decidimos excluirla.

Estas restricciones reducen complejidad en la implementación, y los casos de uso definidos se pueden implementar correctamente bajo estas condiciones.

### 8.1.2. Sistema operativo y lenguaje

Desde el inicio de la plataforma iOS, las aplicaciones se podían implementar solo en el lenguaje de programación Objective C, una restricción tanto técnica como legal, ya que Apple ha llegado hasta el extremo de prohibir las aplicaciones hechas en lenguajes interpretados que se compilan a Objective C.

A partir de la versión 8 del sistema operativo, Apple introdujo Swift, un nuevo lenguaje de programación. Es un lenguaje nuevo, creado por Apple, que en el momento de la redacción de esta memoria, tenía menos de un año de vida (fue presentado en septiembre de 2014).

Las apps a partir de iOS 8 se pueden implementar en Objective C, en Swift o combinando ambos lenguajes.

Hemos decidido implementar este proyecto en Swift, aprovechando para estudiar este nuevo lenguaje con características muy interesantes. Este hecho condiciona que el proyecto solo funciona con versiones del sistema operativo iguales o superiores a la 8. Apple afirma, no obstante, que sus usuarios adoptan las nuevas versiones del sistema operativo muy rápidamente, con lo que esta restricción dejará fuera a pocos usuarios[7].

### 8.1.3. Guía de estilo

Un mismo código puede implementarse de formas diferentes, semánticamente equivalentes. Así que es conveniente que los proyectos sigan una serie de convenciones respecto al formato del código y la sintaxis, que dará coherencia a los distintos archivos dentro de un mismo proyecto, y a los distintos proyectos de una determinada organización.

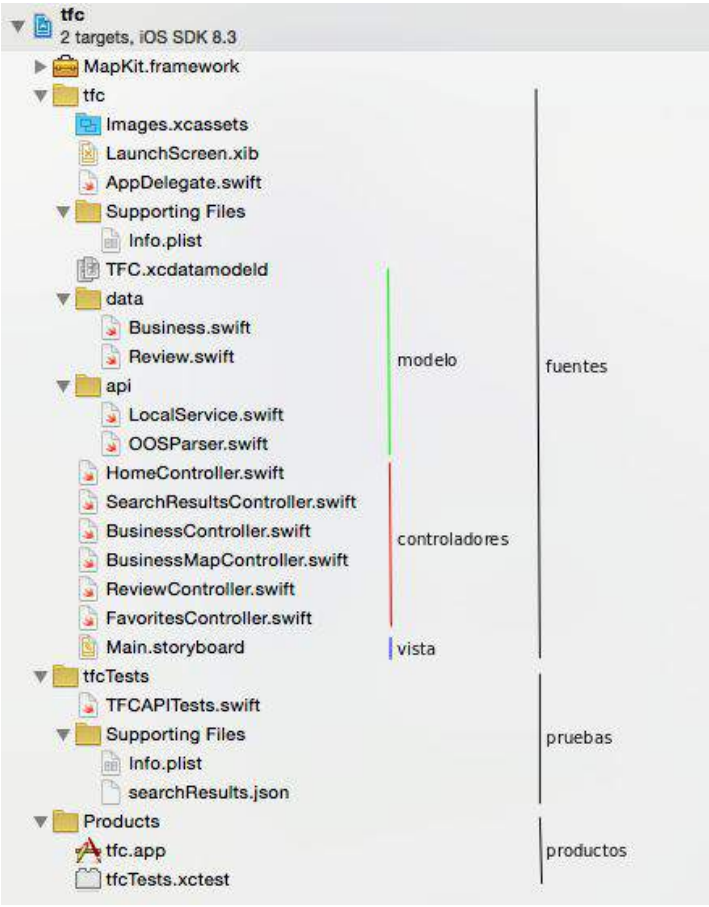
En este proyecto nos hemos basado en la guía de estilo para el lenguaje Swift que usan los programadores de GitHub[8].

### 8.1.4. Organización del código

Todos los proyectos iOS comparten una estructura que viene reforzada por la herramienta de desarrollo XCode. al crear el proyecto XCode genera tres directorios:

- Nombre del proyecto, en nuestro caso “tfc”, donde se ubican los fuentes.
- Test, en nuestro caso “tfcTests”, para las pruebas automatizadas.
- Products, donde XCode ubicará el proyecto compilado y empaquetado.

[7] En concreto, en Junio de 2015, Apple afirma que el 83% de los dispositivos activos usan iOS 8 https://developer.apple.com/support/app-store/
[8] https://github.com/github/swift-style-guide




Crearemos la mayoría de nuestro código en el directorio “tfc”. XCode crea unos ficheros necesarios para el funcionamiento de la app:

- Images.xcassets: contiene imágenes.
- LaunchScreen.xib: pantalla que se muestra al iniciar la app, mientras ésta se carga.
- AppDelegate.swift: Actúa de interfaz entre el sistema operativo y nuestra aplicación. Podemos sobrescribir métodos para responder adecuadamente a los cambios en el sistema.
- Info.plist: configuración del proyecto.

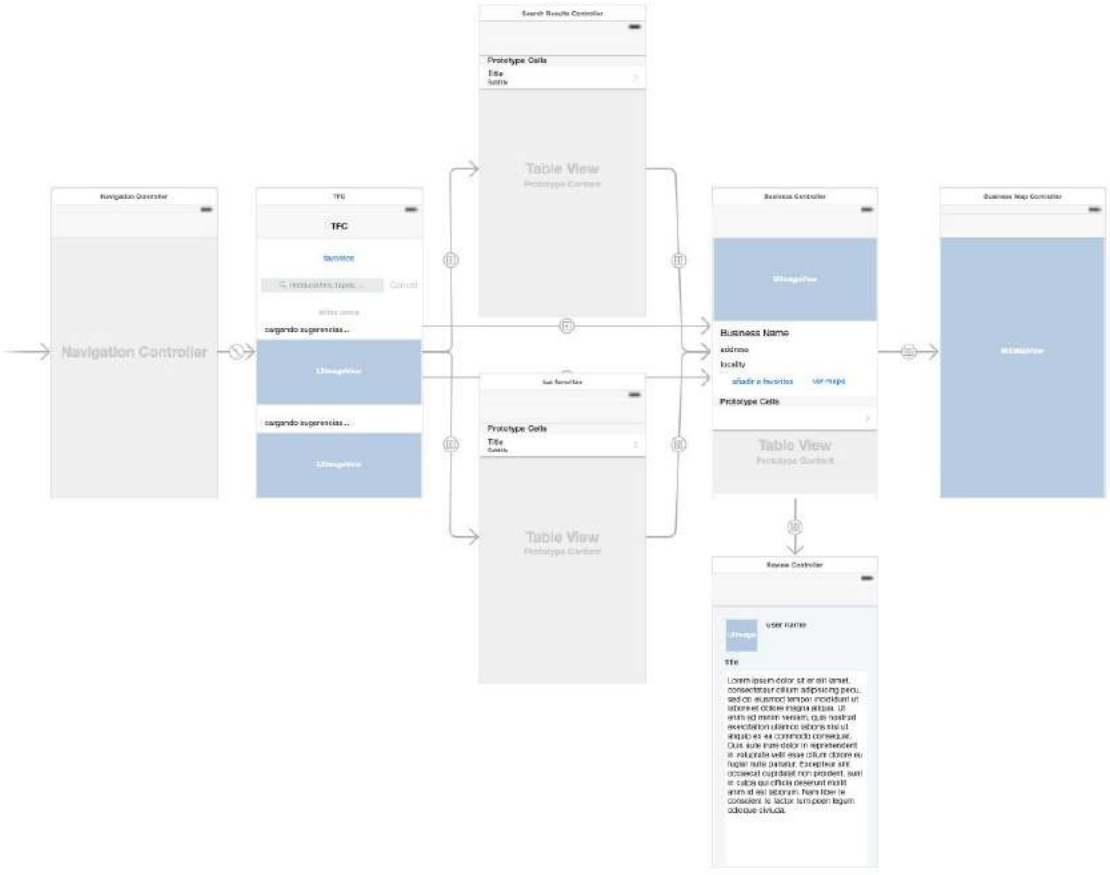
Nosotros hemos creado el resto de ficheros y directorios y los hemos estructurado de forma que sigan el patrón MVC.

## 8.2. MVC View

Apple recomienda definir la interfaz y las transiciones entre pantallas en un solo fichero llamado *storyboard*[9].

Un storyboard contiene una representación visual de la interacción de usuario y se compone de pantallas, que llama *scenes*, y transiciones entre pantallas, que llama *segues*.

El storyboard además se puede ver como un diagrama equivalente al flujo de interacción que se ha generado en la fase diseño. Podemos ver como el siguiente diagrama nos recuerda mucho al flujo presentado en el apartado 7.3.5.7.




9

https://developer.apple.com/library/ios/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html#//apple_ref/doc/uid/TP40009071-CH99-SW1

Cada pantalla está compuesta de una o varias vistas. Las vistas se pueden configurar desde el propio editor de storyboards. A su vez cada pantalla tiene un controlador asociado, y en este controlador podemos incluir referencias a las vistas para manipularlas posteriormente.

Por ejemplo, en la siguiente imagen vemos al controlador de la pantalla inicial (HomeController) en el que hemos incluido referencias a las vistas representando los botones cerca de mí (aroundMeButton) y favoritos (favoritesButton):

```
import UIKit
import CoreData

class HomeController: UIViewController, UISearchBarDelegate, SearchBusinessesDelegate,
    FeaturedBusinessDelegate {

    @IBOutlet weak var favoritesButton: UIButton!
    @IBOutlet weak var aroundMeButton: UIButton!
```

Estas referencias se llaman *IBOutlet* y se deben crear desde el editor de storyboards, arrastrando la vista hacia el controlador.

## 8.3. MVC Controller

Los controladores son los objetos que intermedian entre la vista y el modelo. Tal como hemos visto, los controladores tienen referencias a los objetos que representan los elementos de la interfaz de usuario, y, a través de estas referencias, reciben información de las acciones del usuario.

Asimismo los controladores mantienen referencias al modelo, le envían peticiones y gestionan sus respuestas. A menudo la interacción con el modelo se debe hacer asíncronamente, y el controlador se responsabiliza de recibir los datos del modelo en un *background thread* y presentarlos al usuario, lo que debe hacerse en el *thread* principal.

Veamos por ejemplo cómo el controlador `HomeController` recibe la cadena de búsqueda del usuario, la lanza contra el objeto del modelo que se comunica con la API de 11870, y recibe y presenta los resultados:

1. `HomeController` se ha registrado como delegado de `UISearchBar` implementando el protocolo `UISearchBarDelegate`, lo que provoca que cuando el usuario lanza la búsqueda se invoca al método `searchBarSearchButtonClicked`, que hemos sobreescrito nosotros.
2. La implementación de ese método accede a una referencia al objeto del modelo `LocalService`, obtiene la cadena de búsqueda, se registra como delegado para obtener los resultados, y lanza la búsqueda:

```
func searchBarSearchButtonClicked(searchBar: UISearchBar)
{
    // obtenemos una referencia a LocalService (model)
    let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
    let localService = appDelegate.localService!

    // nos registramos como delegate para recibir el resultado de búsqueda
    localService.searchDelegate = self

    // obtenemos la cadena de búsqueda de la serchBar, y la lanzamos al modelo
    localService.search(searchBar.text)
}
```

3. Una vez el objeto del modelo obtiene los datos de internet (omitimos ese código por claridad) el controlador recibe los resultados y los presenta en una nueva pantalla, a la que accede llamando una *segue*. Para tomar el control de la interfaz de usuario debe lanzar la *segue* desde el *thread* principal, y para ello utiliza la instrucción del sistema operativo *dispatch_sync*:

```
// MARK: SearchBusinessDelegate

func searchResults(results: [Business]) {
    self.searchResults = results
    // no podemos lanzar segues xq nos han llamado asíncronamente, trasladar al main thread
    dispatch_sync(dispatch_get_main_queue())
    {
        self.performSegueWithIdentifier("searchResults", sender: self)
    }
}
```

La aplicación tiene un controlador por cada pantalla. Los vemos en detalle, especificando a qué partes del modelo y la vista acceden.

| controlador | modelo | vista |
|---|---|---|
| `HomeController` | `LocalService`<br>`NSManagedObjectContext` | Pantalla inicial |
| `SearchResultsController` | | Resultados de búsqueda |
| `BusinessController` | `LocalService`<br>`NSManagedObjectContext` | Detalle de negocio |
| `BusinessMapController` | Geolocalización | Mapa de negocio |
| `ReviewController` | | Detalle de negocio |
| `FavoritesController` | `NSManagedObjectContext` | Pantalla de favoritos |

## 8.4. MVC Model

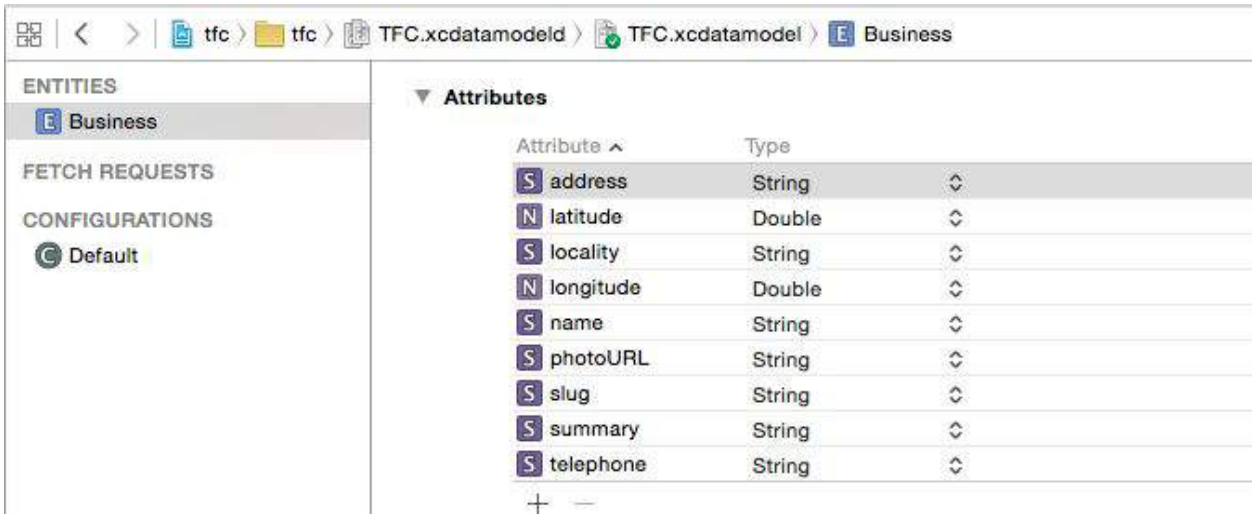
En el modelo reside la lógica de negocio y desde esta capa gestionaremos la interacción con las piezas del sistema que no pertenecen a la interfaz de usuario. Nuestra aplicación trabaja con la base de datos y los servicios de geolocalización que gestiona el dispositivo; y con la API de 11870, a la que accedemos por Internet usando los servicios de conectividad que pone a nuestra disposición el sistema operativo.

### 8.4.1. Base de datos

En la base de datos del dispositivo guardaremos los favoritos que el usuario selecciona.

Apple proporciona el *framework* Core Data para gestionar los objetos que persisten y el acceso a base de datos. XCode crea un fichero con el nombre del proyecto y extensión `.xcdatamodeld` en el que definiremos las entidades y atributos que forman parte de nuestro modelo, y las relaciones entre entidades.

En nuestro caso solo tenemos la entidad `Business`:




Apple proporciona código que tenemos que incorporar a nuestro `AppDelegate` para inicializar el framework. Hay dos partes relevantes de ese código a las que accederemos desde nuestra aplicación para interactuar con el framework:

- Objeto `ManagedObjectContext`. Es objeto que contiene referencias a nuestras entidades y gestiona su ciclo de vida.
- Función `saveContext`. Persiste los cambios que hayamos hecho en los objetos gestionados por el managed object context.

Vemos el código relevante en `AppDelegate`:

```
lazy var managedObjectContext: NSManagedObjectContext? = {
    let coordinator = self.persistentStoreCoordinator
    if coordinator == nil {
        return nil
    }
    var managedObjectContext = NSManagedObjectContext()
    managedObjectContext.persistentStoreCoordinator = coordinator
    return managedObjectContext
    }()

// MARK: - Core Data Saving support

func saveContext () {
    if let moc = self.managedObjectContext {
        var error: NSError? = nil
        if moc.hasChanges && !moc.save(&error) {
            NSLog("Unresolved error \(error), \(error!.userInfo)")
            abort()
        }
    }
}
```

Cuando necesitamos trabajar con la base de datos accedemos al Managed Object Context a través de `AppDelegate`. Por ejemplo, en `FavoritesController`, usamos este código para acceder al contexto, borrar un favorito y persistir los cambios:

```
// obtenemos el managed object context para interacturar con DB
let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
let context = appDelegate.managedObjectContext!

let biz = businesses[indexPath.row]
// borramos el favorito del context
context.deleteObject(biz)

// persistimos los cambios
var error: NSError?
if !context.save(&error) {
    println("Could not delete \(error), \(error?.userInfo)")
}
```

### 8.4.2. API 11870

El acceso a la API de 11870 lo hacemos a través de la clase `LocalService`. Cada interacción con la API se hace a través de una de sus funciones, y definimos delegados para recoger los resultados de cada función.

Tenemos una sola instancia de LocalService, referenciada desde la clase `AppDelegate`. El sistema operativo crea una instancia de esta clase y proporciona una referencia al arrancar la app. Solo hay una instancia de `AppDelegate`, y por ello solo hay una instancia de `LocalService`.

El sistema operativo nos da una forma de acceder a la instancia de `AppDelegate` y a través de ella, accederemos a la referencia de `LocalService`, por ejemplo, en `HomeController`:

```
// load featured businesses
let appDelegate = UIApplication.sharedApplication().delegate as! AppDelegate
let localService = appDelegate.localService!
```

### 8.4.2.1. Asincronismo y *delegates*

Las llamadas a `LocalService` implican peticiones a la API de 11870 a través de Internet. Esas peticiones pueden ser lentas o incluso no llegar a completarse, por lo que es conveniente hacerlas asíncronamente. Las peticiones se hacen en un *thread* distinto del que gestiona la interfaz de usuario, para que la espera no afecte a la experiencia de uso.

Vemos en el siguiente extracto el método `search`, que envía por Internet la consulta del usuario. La parte relevante es la llamada `dataTaskWithURL`, que crea y arranca una `task`. Una `task` es un objeto que contiene código que se llama en *background*.

```
//
// búsqueda texto libre
//
func search(query: NSString)
{
    let baseURL = BASE_URL + SEARCH
    let url = NSURLComponents(string: baseURL)!
    url.query = "appToken=\(APP_TOKEN)&q=\(query)&alt=json&count=30"

    // creamos la task que se llamará en el background thread
    let task = NSURLSession.sharedSession().dataTaskWithURL(url.URL!) {(data, response, error) in
        let results = self.parser.parseSearchResults(data)
        if let delegate = self.searchDelegate {
            delegate.searchResults(results)
        }
    }

    // lanzamos la task en background
    task.resume()
}
```

Para recibir la respuesta de una llamada asíncrona recurrimos al patrón de diseño recomendado en iOS: la delegación[10].

Cada llamada a LocalService tiene asociado un delegado que gestionará la respuesta. El controlador que accede a LocalService tiene la responsabilidad de registrar un delegado que atenderá la respuesta de la API. Por ejemplo, para la búsqueda, desde `HomeController` ejecutamos el siguiente código:

---

10 https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/Delegation.html

```
// nos registramos como delegate para recibir el resultado de búsqueda
localService.searchDelegate = self

// obtenemos la cadena de búsqueda de la serchBar, y la lanzamos al modelo
localService.search(searchBar.text)
```

### 8.4.2.2. Conversión JSON

El servicio que gestiona la interacción con la API de 11870 debe encargarse también de convertir los datos que enviamos en una URL, y, sobre todo, convertir en objetos los datos que recibimos en formato JSON. Es un código extenso y repetitivo, por lo que hemos decidido separarlo en una clase aparte de `LocalService`. La clase se llama OOSParser y tiene un par de métodos por cada llamada de la API que interpreta.

El primer método es de la forma `parseXXX(jsonSearchResults: NSData)` observamos que recibe una instancia de `NSData`, que es la respuesta del servidor como un array de bytes.

Este método se encarga de usar el intérprete JSON del sistema operativo para convertir la entrada en objetos genéricos del sistema (diccionarios, array, strings, etc.). Vemos el método que recibe los resultados de búsqueda y los convierte en un array de diccionarios:

```
func parseSearchResults(jsonSearchResults: NSData) -> [Business] {
    var results = [Business]()
    var parseError: NSError?
    let parsedObject: NSDictionary = NSJSONSerialization.JSONObjectWithData(
            jsonSearchResults,
            options: NSJSONReadingOptions.AllowFragments,
            error:&parseError) as! NSDictionary

    let entries = parsedObject["entries"] as! NSArray
    for entry in entries {
        let biz = parseBusiness(entry as! NSDictionary)
        results.append(biz)
    }
    return results
}
```

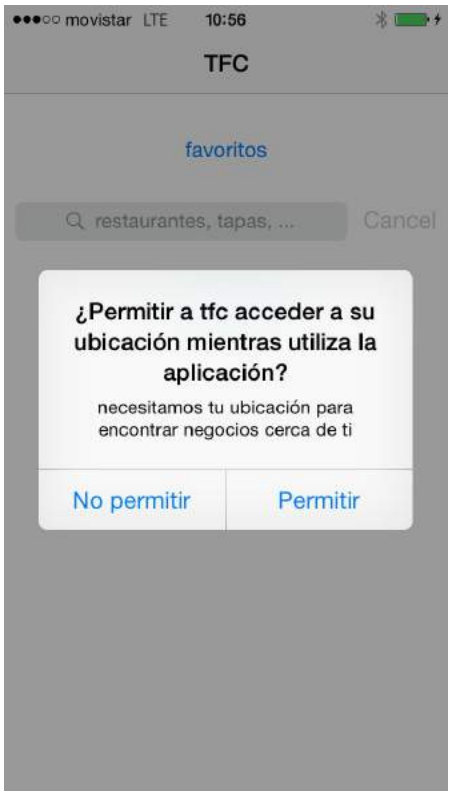
El siguiente paso es convertir cada uno de esos diccionarios en un objeto de nuestro modelo, en este caso en instancias de la clase `Business`:

```
func parseBusiness(entry: NSDictionary) -> Business {
    let name = entry["title"] as! String
    let summary = entry["summary"] as? String
    var slug: String = "slug"

    var address: String? = nil
    var locality: String? = nil
    var latitude: Double? = nil
    var longitude: Double? = nil

    var telephone: String? = nil
    var photoURL: String? = nil

    if let extensions = entry["extensions"] as? NSDictionary {
        slug = extensions["oos_slug"] as! String
        if let jsonaddress = extensions["oos_useraddress"] as? String {
            address = jsonaddress
        }
        if let jsonphone = extensions["oos_telephone"] as? String {
            telephone = jsonphone
        }
        if let jsonlocality = extensions["oos_locality"] as? NSDictionary {
            locality = (jsonlocality["name"] as! String)
        }
        if let georss = extensions["georss_where"] as? NSDictionary {
            if let gml_point = georss["gml_point"] as? NSDictionary {
                let position = (gml_point["gml_pos"] as! String)
                let pointArray = split(position) {$0 == " "}
                latitude = (pointArray[0] as NSString).doubleValue
                longitude = (pointArray[1] as NSString).doubleValue
            }
        }
    }
    if let links = entry["links"] as? NSArray {
        for link in links {
            if let rel = link["rel"] as? String {
                if rel == "media" {
                    photoURL = link["href"] as? String
                }
            }
        }
    }
```

### 8.4.3. Geolocalización

El sistema operativo proporciona acceso a la localización del dispositivo. Para ello el usuario tiene que haberlo autorizado, y el dispositivo debe tener disponible el *hardware* de localización, por ejemplo no lo tendría si estuviera en modo avión.

Gestionamos la localización del usuario desde la clase `LocalService`. Cuando ésta se inicializa solicitamos al usuario permiso para obtener la localización, si obtenemos la autorización nos registramos para obtener actualizaciones de las coordenadas del dispositivo. Solicitamos la localización solo mientras la app está activa:




`LocalService` se registra como delegado de `CLLocationManager`, y éste nos notifica cuando el usuario acepta (o rechaza) compartir su localización con la app, y cuando se producen cambios en la localización del dispositivo.

`LocalService` guarda la última localización del dispositivo como una propiedad, que actualiza cada vez que recibe una nueva localización.

```
// MARK: location manager

//
// el sistema nos notifica si el usuario autoriza compartir su ubicación
//
func locationManager(manager: CLLocationManager!, didChangeAuthorizationStatus status: CLAuthorizationStatus)
{
    if status == .AuthorizedWhenInUse {
        manager.startUpdatingLocation()
    }
}

//
// el sistema nos notifica la localización del usuario
//
func locationManager(manager: CLLocationManager!, didUpdateLocations locations: [AnyObject]!) {
    let firstLocation = self.location == nil
    self.location = locations.last as? CLLocation
    if firstLocation {
        if let coord = self.location?.coordinate {
            NSNotificationCenter.defaultCenter().postNotificationName("TFCLocationReady", object: self)
        }
    }
}
```

Tal como hemos visto en el apartado anterior, solo existe una instancia de `LocalService` en el sistema, a la que accedemos por medio de la clase `AppDelegate`. De esta forma tenemos siempre la última localización del usuario disponible, y evitamos problemas de duplicidad o concurrencia.

## 8.5. Pruebas

### 8.5.1. Pruebas unitarias

Cada respuesta de la API de 11870 se procesa con un método de `OOSParser` que traslada el JSON que nos ha devuelto la API a objetos Swift, y cada uno de esos métodos tiene un test unitario asociado. Para poder probar, guardamos en un fichero de texto una respuesta conocida de la API, que guardamos en el fichero `tfcTests/Supporting Files`

Por ejemplo, los resultados de búsqueda con la cadena “hola”, devuelven los datos que hemos guardado en `searchResults.json`.

La clase `TFCAPITests.swift` implementa las pruebas unitarias de `OOSParser`, y el método `testParseJSONSearch` prueba que `OOSParser` interpreta correctamente la respuesta de `searchResults.json`.


## 8.6. Funcionamiento de la aplicación

### 8.6.1. Pantalla de inicio

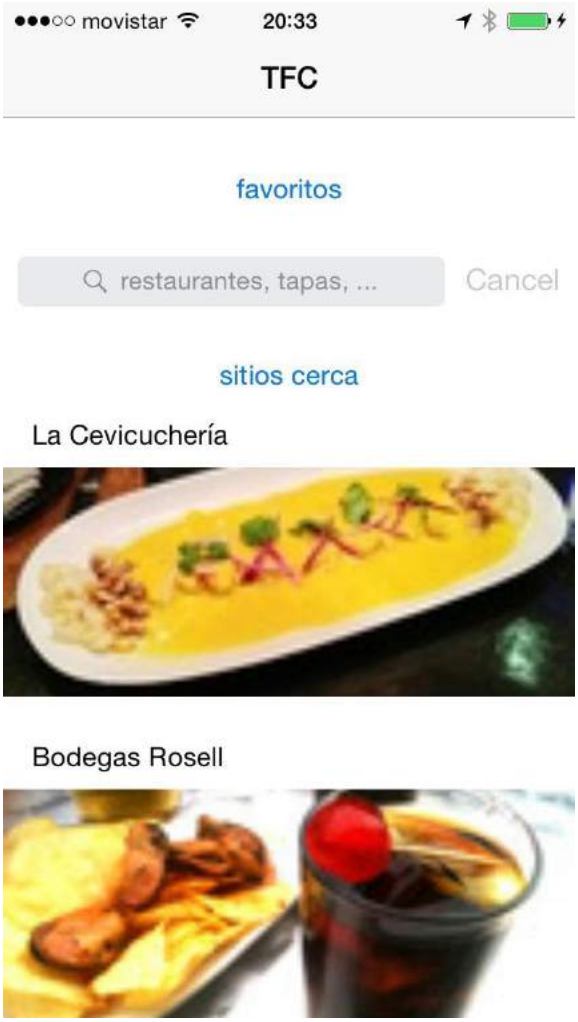
La pantalla inicial se divide en dos áreas:

Búsqueda

Es la parte superior de la pantalla, contiene un botón para acceder a los favoritos del usuario, la caja de búsqueda libre, y un botón para ver los sitios cercanos. Si la localización del dispositivo no está disponible, este botón aparece deshabilitado.

Sugerencias

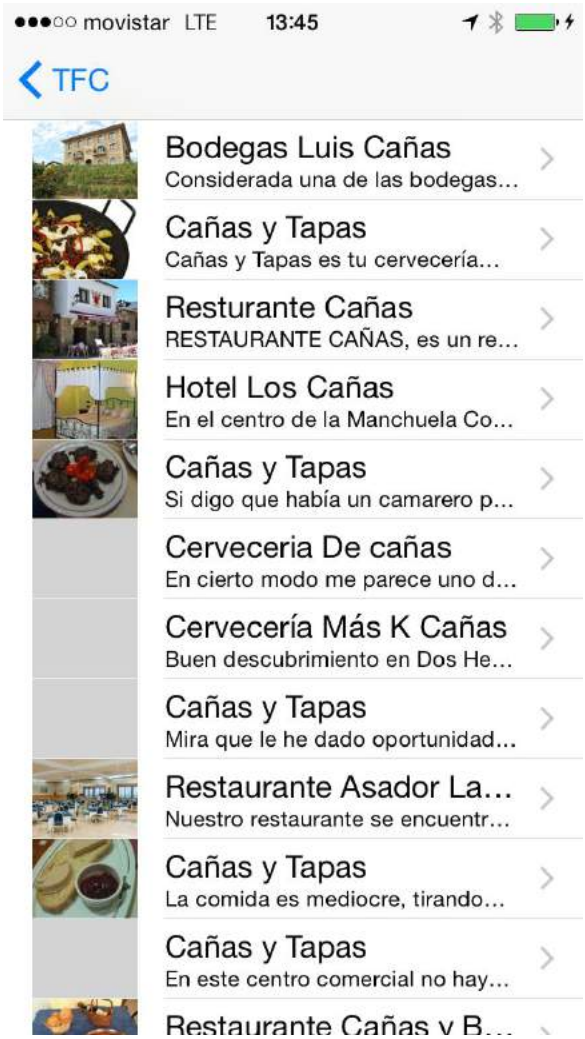
Muestra sugerencias de negocios basados en las preferencias del usuario, la ubicación y el momento del día. Solo se sugieren negocios que tengan un número elevado de opiniones y fotos, y que hayan recibido valoraciones altas de los usuarios. Si el usuario aún no tiene favoritos, el botón aparece deshabilitado.



### 8.6.2. Resultados de búsqueda (*serp*)

Tanto si el usuario hace una búsqueda de texto libre, como si toca el botón “sitios cerca”, los resultados se presentan como una tabla, en la que cada celda es un negocio, del que vemos una foto, el nombre y un sumario de una de las opiniones.

En caso de haber llegado buscando sitios cercanos, la ordenación es por distancia (primero los sitios más cercanos), si el usuario ha hecho una búsqueda libre, la ordenación es por relevancia respecto a la búsqueda y por contenido (cuantas más opiniones, mejor).



### 8.6.3. Ficha de negocio (*business*)

Cuando el usuario toca uno de los negocios le llevamos a una pantalla con los detalles de ese negocio. Si la API nos ha dado las coordenadas del negocio, mostramos un botón para ver el negocio en un mapa, en caso contrario el botón no es visible.

Permitimos que el negocio se añada a los favoritos del usuario con un botón específico. Si el sitio ya está en los favoritos, el texto del botón cambia a “quitar de favoritos” y su funcionalidad es la que sugiere el texto.

Mostramos una tabla con un extracto de cada opinión, y queda pendiente ofrecer al usuario una forma de acceder a la opinión completa, bien en esta misma pantalla o en una nueva pantalla específica.


●●●●○ movistar
20:38
Back
Cañas y Tapas
Calle de Atocha 42
Madrid
añadir a favoritos
ver mapa
Cañas y Tapas es tu cervecería de…
Cañas y Tapas es tu cervecería de siempre, la qu…
la cadena española de cervecerías…
La tentación en cuestión se llama Cañas y Tapas…
no vuelvo
No se que me impulsó un dia a entrar. Tampoco p…
El menu del dia es bararo si (9€) pero la comida n…

### 8.6.4. Mapa de negocio (*map*)

Si tenemos las coordenadas del negocio, mostramos el botón “ver mapa” que lleva al usuario a un mapa centrado en la ubicación del negocio, con una “chincheta” identificándolo.

### 8.6.5. Opinión (*review*)

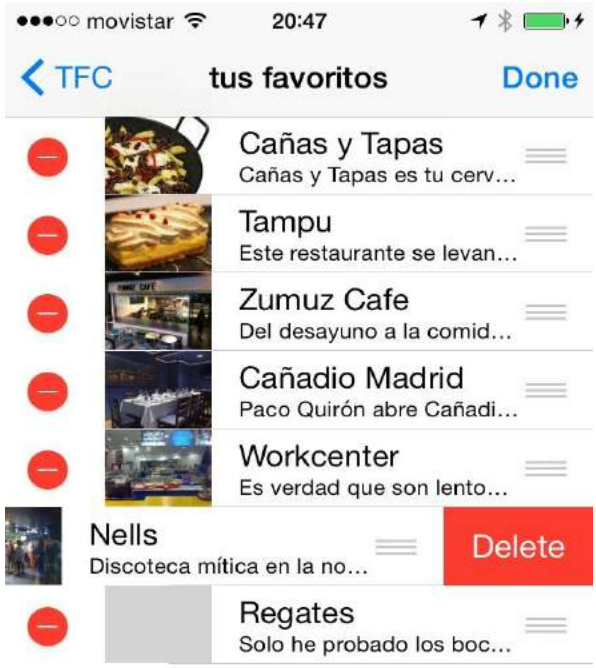
Desde la pantalla de negocio el usuario puede tocar una opinión para acceder al detalle. Esta pantalla muestra la foto del usuario, su nombre, y el título y el texto de la opinión.


### 8.6.6. Favoritos (*favs*)

Esta pantalla muestra los negocios que el usuario ha guardado como favoritos. Tocando en cada uno de ellos se puede acceder al detalle de ese negocio.

La pantalla permite el modo edición para eliminar favoritos, y cambiar el orden en el que se muestran. El borrado se puede desencadenar también desplazando uno de los negocios hacia la izquierda.


movistar
20:47
TFC
tus favoritos
Edit
Cañas y Tapas
Cañas y Tapas es tu cervecería...
Tampu
Este restaurante se levanta con...
Zumuz Cafe
Del desayuno a la comida, todo l...
Cañadio Madrid
Paco Quirón abre Cañadio en 19...
Workcenter
Es verdad que son lentos, much...
Nells
Discoteca mítica en la noche ma...
Regates
Solo he probado los bocadillos,...


movistar
20:47
TFC
tus favoritos
Done
Cañas y Tapas
Cañas y Tapas es tu cerv...
Tampu
Este restaurante se levan...
Zumuz Cafe
Del desayuno a la comid...
Cañadio Madrid
Paco Quirón abre Cañadi...
Workcenter
Es verdad que son lento...
Nells
Discoteca mítica en la no...
Delete
Regates
Solo he probado los boc...

# 9. Conclusión

## 9.1. Funcionalidad y diseño

Hemos conseguido finalizar una aplicación funcional cumpliendo los requerimientos marcados al inicio del proyecto, y siguiendo las pautas marcadas en el diseño. La aplicación es útil y podría usarse en el mercado ahora mismo.

Sin embargo para poder competir con las que existen actualmente en mercado deberíamos trabajar más la experiencia de usuario, sobre todo en el apartado gráfico.

Por otro lado añadir determinado funcionalidad haría de la app un producto más completo. Si esta aplicación tuviese continuidad en el futuro, sugeriríamos centrarnos en estos tres aspectos:

- Diseño: en particular el diseño gráfico es la principal carencia de la app.
- Creación de contenido: tal como ha sugerido el consultor Jordi Ceballos, la app mejoraría si añadiéramos funcionalidad para permitir al usuarios contribuir sus propias opiniones y fotos, y hacer seguimiento de sus visitas a negocios (hacer *check in*).
- Anticipación: la app debería “aprender” de la actividad del usuario: sus búsquedas, favoritos, etc. para sugerirle los negocios que pueden interesarle en cada momento, de esa forma anticipando sus acciones.

## 9.2. Tecnología

He querido aprovechar este trabajo para profundizar en la plataforma iOS, que solo conocía superficialmente, y en particular conocer el lenguaje Swift y las tecnologías de Storyboards y Autolayout.

Personalmente he conseguido con creces esos objetivos, si bien dedicar mucho tiempo al aprendizaje ha supuesto una ralentización importante del ritmo de trabajo.

### 9.2.1. Swift

El descubrimiento más interesante, hemos hecho este trabajo en Swift 1.2, pero durante la redacción de la memoria, Apple ha anunciado la versión 2, y la conversión del proyecto en *open source*. Así que podemos esperar ver el lenguaje en otras plataformas en un futuro cercano.

Swift es un lenguaje que facilita la creación de un código conciso, expresivo y robusto.

Recuerda un poco a Python y Ruby en concisión y expresividad, pero a diferencia de estos, Swift es un lenguaje diseñado para cazar muchos errores en tiempo de compilación, como podría ser por ejemplo Java.

La característica más llamativa de Swift son los tipos opcionales. El lenguaje diferencia entre tipos de datos que aceptan valores nulos o no, como por ejemplo hacen las bases de datos. Eso exige que el programador declare la nulabilidad en el momento de definir las referencias a los datos.

Swift obliga a declarar los datos que usamos como variables o constantes, lo que combinado con los tipos opcionales permite que el compilador sea capaz de encontrar muchos posibles errores relacionados con la asignación de valores nulos a tipos no *nulables*.

La contrapartida es que el código puede volverse algo farragoso y repetitivo, por ejemplo es lo que nos ocurre a nosotros en la clase OOSParser. Dado que no podemos afirmar si el JSON que recibimos tendrá todos los posibles atributos, necesitamos encadenar varias lecturas de datos condicionales, lo que se hace algo pesado:

```
if let jsoncontent = entry["content"] as? NSDictionary {
    content = (jsoncontent["value"] as! String)
}
if let jsonauthor = entry["author"] as? NSDictionary {
    authorName = (jsonauthor["name"] as! String)
    if let jsonext = jsonauthor["extensions"] as? NSDictionary {
        authorPhotoURL = (jsonext["oos_avatar"] as! String)
    }
}
```

### 9.2.2. Storyboards

Son una herramienta muy poderosa que permite definir gráficamente partes muy importantes de la interfaz de usuario y de la navegación. En proyectos pequeños será particularmente valioso por la capacidad de síntesis que tiene poder ver el flujo de interacción real de la app en una imagen. En proyectos grandes probablemente se pierda esta propiedad.

A cambio no me siento cómodo con el hecho de que un storyboard es un fichero cuyo formato está oculto a los ojos del programador, lo que quita la posibilidad de utilizar herramientas fuera del ámbito Apple. Por ejemplo, si queremos editar los ficheros `.swift`, podemos hacerlo con herramientas ajenas a XCode como Sublime Text, o sed y grep para hacer cambios masivos.

### 9.2.3. Autolayout

La técnica de Autolayout permite definir una misma escena de storyboards para dispositivos de distinto tamaño (iPad y iPhone) y para distintas orientaciones. Indicamos al sistema cómo tiene que comportarse ante distintas condiciones definiendo una *constraints* en lugar de medidas fijas. Es un sistema con mucho potencial, pero en algunos casos he encontrado que definiciones muy simples exigían tratamiento muy complejo. Hasta el punto de deshabilitarlo dado que el proyecto no exigía funcionar en iPad ni en orientación horizontal.

# 10. Bibliografía

iOS Developer Library. [En línea]. [https://developer.apple.com/library/prerelease/ios/navigation]

Developing iOS 8 Apps with Swift. Paul Hegarty, Stanford University. [Curso en iTunes U] [https://itunes.apple.com/us/course/developing-ios-8-apps-swift/id961180099]

The Swift Programing Language. [En línea]. [https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/]

Foros de Stack Overflow. [En línea]. [http://stackoverflow.com/]

Core Location in iOS 8. Mike Lazer-Walker. [En línea]. [http://nshipster.com/core-location-in-ios-8/]

NSURL/NSURLComponents. Mattt Thompson. [En línea]. [http://nshipster.com/nsurl/]