eman ta zabal zazu

Escuela Universitaria de Ingeniería Técnica Industrial de Bilbao

GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado
2013 / 2014

FOLLOW

MEMORIA TFG

DATOS DE LA ALUMNA O DEL ALUMNO	DATOS DEL DIRECTOR O DE LA DIRECTORA
NOMBRE: MARIO	NOMBRE: MIKEL
APELLIDOS: ALVARO SOLIS	APELLIDOS: VILLAMAÑE GIRONÉS
	DEPARTAMENTO: LENGUAJES Y SISTEMAS INFORMÁTICOS
FDO.:	FDO.:
FECHA:	FECHA:

Resumen

Este Trabajo Fin de Grado trata sobre el desarrollo de una aplicación para iPhone de guías de turismo de nombre “Follow”. La aplicación se caracteriza por la descarga de la información de manera segmentada y por la posibilidad de consultarla cuando se desee sin necesidad de tener acceso a Internet. Así mismo, el usuario que se registre en la aplicación, tendrá la opción de compartir sus propias guías con el resto de personas que se la descarguen. Las guías están conformadas por texto, fotos y vídeos, dividiéndose en varios apartados. De la misma manera, los usuarios podrán contactar entre ellos mediante el envío de mensajes o consultando sus perfiles.

En esta memoria se explicará detalladamente las diferentes fases del proceso en la creación de la aplicación con el fin de entender el desarrollo llevado a cabo hasta el resultado final.

ÍNDICE

Pág.

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1. Introducción al proyecto

A través de este documento, se pretende transmitir de una forma clara y formal, a través de las distintas secciones que lo componen, la causa y efecto del desarrollo de este trabajo fin de grado.

1.1. Introducción

Este proyecto surge como una propuesta por parte de los profesores, que después se ha ido ampliando con ideas propias y originadas de aplicaciones ya existentes.

Al utilizarse un lenguaje nuevo, cuyo aprendizaje no se ha desarrollado en el Grado, se trata del primer trabajo que voy a realizar en dicha área. Se utilizará el lenguaje Objective-C, que es el utilizado hasta ahora por los programadores para crear aplicaciones para dispositivos y ordenadores de Apple.

El objetivo del proyecto es crear una aplicación para la plataforma móvil de Apple, de manera que cada usuario pueda acceder a ésta y obtener guías de turismo con el fin de viajar de forma más cómoda. Dependiendo de en qué lugar se encuentre el usuario, el ancho de banda variará muchísimo en algunas ocasiones. No es lo mismo el acceso a internet de un bar, de tu casa o de una estación de autobuses. Por ello, se permitirá la descarga de la información segmentada, para que el usuario decida qué contenido, cantidad y formato desea descargarse para que haga uso únicamente de lo que crea conveniente.

Para cumplir este objetivo, se hará uso de algunos de los conceptos aprendidos durante el Grado. No obstante, y como Objective-C no se ha cursado en la carrera, se empleará más tiempo en el autoaprendizaje de dicho lenguaje mediante libros, foros de internet, tutoriales y vídeo-tutoriales.

Los teléfonos móviles son parte de la sociedad y de la forma de vida que llevamos en la actualidad. Los datos analizados por el portal *Xataka Móvil* lo confirma, indicando que "*por primera vez se han superado la venta de mil millones de dispositivos en el 2013*". En concreto, Apple ha vendido alrededor de 153 millones de iPhone, siendo la segunda compañía que más s*martphones* ha vendido, siendo el primer puesto de la lista para su principal competidor: Samsung.

Interpretando esta información, se deduce que si se viaja en compañía, al menos una persona tiene un *smartphone*. El propósito que se persigue con este proyecto es que si algún día la aplicación se pone a la venta, es dejar de usar multitud de mapas, folletos con horarios y precios, libros o revistas con información sobre los lugares que se van a visitar y que quizá contengan información que no es de interés.

1.2. Descripción

La idea de este proyecto orbita alrededor de una célebre frase utilizada en matemáticas, pero aplicable a muchos ámbitos de la vida: “Divide y vencerás”.

Por ello, se perseguirá crear una aplicación para iPhone de manera que cada usuario pueda acceder a ella y obtener guías de turismo para poder viajar cómodamente. Se podrá descargar todo lo que uno quiera y, por lo tanto, sin información irrelevante que no interese o sobre lugares que no se vaya a visitar.

La aplicación, en cuanto a información, se dividirá en cinco secciones y en cada una habrá diferente información:

1. Ciudad: Será la primera sección y desde aquí se accede a todo lo demás. Tendrá el nombre y el país al que pertenece y a su vez, podrá tener una imagen del mapa del metro.
2. Lugar: Se trata de todos los lugares que se puedan visitar dentro de una ciudad. Pueden ser monumentos, museos, calles, etc. La información que se presentará en cada uno de ellos será una descripción, los horarios de acceso si los tuviera y el precio de entrada por cada visita, si hubiera que pagar.
3. Rutas: Las rutas tienen como principio guiar a los usuarios por varios lugares. Se accede desde la ciudad que se quiere visitar y estarán divididas en los días que tarde la ruta en finalizarse. Para ello, se utilizará una descripción de texto en la que se mostrará una descripción detallada del camino a seguir.
4. Fotos: Este bloque se divide en dos partes. Las fotos serán imágenes sobre *Ciudades* y *Lugares* que los usuarios hayan subido desde el dispositivo o las que estén en la aplicación. Un lugar contendrá unas fotos y las ciudades otras diferentes, a menos que los usuarios decidan subir la misma foto a ambos apartados.
5. Vídeos: Con los vídeos pasará lo mismo que con las fotos. Cada *Ciudad* y cada *Lugar* tendrán diferentes vídeos informativos.

A su vez, al acceder a un lugar de una ciudad, se mostrará los kilómetros que separan al usuario de ese sitio utilizando la Geolocalización e incluso verlo marcado en un mapa, así como la ruta más rápida para llegar a ese lugar.

La información se podrá descargar de diferentes maneras:

-La guía completa de la ciudad: Se descargará toda la información de la ciudad a la que el usuario haya accedido. Este tipo de descarga es la que más tiempo llevará ya que descargará toda la información de los lugares de esa ciudad, las rutas, fotos y vídeos si los tuviera, y por último, el mapa de metro si la ciudad tuviera el transporte.

-Lugar: Con este tipo de descarga, se podrán diferenciar otras dos:

- Si se decide descargar toda la información de un lugar concreto, se almacenará las fotos, los vídeos, y la descripción, horario y precio que tenga ese lugar.

- Si se decide descargar sólo el texto, se almacenará la descripción, horarios y precios de ese lugar, de manera que la cantidad de descarga sea menor y por lo tanto más rápida.

En la descripción de los lugares se ha añadido una funcionalidad que permitirá al usuario oír todo el texto mediante los altavoces o auriculares. Así, a la hora de viajar, podrá escuchar la descripción sin tener que leerla.

-Fotos: Se descargan fotos tomadas por los usuarios que estén registrados en la aplicación o de las ya incluidas en la aplicación.
-Vídeos: Al igual que las fotos, los vídeos pueden ser subidos por los usuarios o incluidas en la aplicación. Se accede a ellos de la misma forma que a las fotos.
-Rutas: Con la descarga de las rutas, se descargará el texto con su descripción.
-Mapas de Metro: Se podrán descargar los planos de metro de las ciudades que dispongan de dicho transporte.

Por otro lado, se permitirá a los usuarios registrarse en el sistema. Debido a que las guías puedan no mostrar todos los lugares, monumentos, etc, que puedan ser "secundarios" a la hora de visitar, los usuarios registrados pueden incluir esta información adicional a modo de guía para que cualquiera pueda descargarla y aparezcan en los diferentes apartados descritos anteriormente. Aquellas personas que se registren, tendrán la oportunidad de valorar las guías y varias funcionalidades más.

El usuario podrá votar las guías que están en el sistema o las generadas por los usuarios. Las votaciones serán sobre los *Lugares*, *Fotos*, *Vídeos* y *Rutas*. La puntuación final de una Ciudad será la suma de todas ellas, mientras que las de los lugares será la suma de la puntuación de su descripción y de las fotos y vídeos que le pertenezcan.

También podrá incluir comentarios de los lugares para dejar su opinión o de la información que se ha dado con el fin de mejorarla o hacer una crítica constructiva.

Se podrá recibir y mandar mensajes a otros usuarios. A su vez, cada usuario, tendrá un perfil con dos apartados:

-No visible públicamente: Su información personal: Fecha de nacimiento, lugar de residencia, email, número de teléfono (si se incluye), intereses, viajes realizados y viajes deseados.
-Visible públicamente: Apartado con escueta información en la que aparecerá su nick, su edad y la foto de perfil.
Para diferenciar entre ambos, cada usuario podrá bloquear su perfil y así poder decidir quién le sigue y quién no y así controlar la información que quiere compartir.

Los usuarios registrados se podrán relacionar mutuamente añadiéndose como amigos y de esta forma, compartir con ellos los datos de su perfil que no están visibles públicamente.

En definitiva, la aplicación tendrá una interfaz intuitiva y lineal, de manera que el usuario al buscar una ciudad y entrar en ella, pueda ir navegando mediante vistas por todo su contenido fácilmente. Se ha dividido la información para descargar con el fin de que el usuario decida qué, cómo y cuándo consultar dicha información.

2. Conceptos

A continuación se explicarán varios conceptos que definen iOS, Objective-C y la forma de programar para hacer aplicaciones de iPhone, con el fin de que el lector entienda algunos temas que se tratan más adelante.

2.1. ¿Qué es iOS?

iOS es el sistema operativo diseñado por Apple que está presente en varios dispositivos como iPhone, iPad y iPod Touch. Tiene como núcleo XNU (X is Not Unix), que fue desarrollado por la propia compañía Apple, aunque originalmente fue desarrollado por NeXT, y forma parte del sistema operativo Darwin.

En cuanto a los desarrolladores, Apple proporciona una API de programación llamada Cocoa Touch. Se utiliza la herramienta Xcode que es un entorno de desarrollo desde el cual se pueden realizar aplicaciones para iOS y Mac OS X en C, C++, Java y Objective-C.

Para hacer aplicaciones en iOS el lenguaje que se ha de utilizar es Objective-C. Este lenguaje es un lenguaje de programación orientado a objetos, creado como un super-conjunto de C. Debido a esto, tiene la posibilidad de compilar cualquier programa escrito en C e incluir libremente código en C dentro de una clase de Objective-C.

2.2. Historia

El 9 de enero de 2007, en la Macworld Conference & Expo, Apple reveló el sistema operativo que llevaría el primer iPhone, iPhone OS, que tuvo su lanzamiento en junio del 2007.

En los meses siguientes, el interés en el SDK creció deprisa gracias al crecimiento de la plataforma iPhone y meses después, el iPod Touch ayudó a su éxito.

En 2010, Steve Jobs, director ejecutivo de Apple, anunció la *tablet* iPad que compartiría el mismo sistema operativo que los otros dos dispositivos y que en verano, en la presentación del iPhone 4, Steve Jobs anunció que iPhone OS pasaría oficialmente a ser llamado iOS.

Tras varios cambios en las versiones, actualmente se encuentra en la versión 7.1, y a lo largo de su historia la interfaz ha ido cambiando y, además, actualizándose gratuitamente.

2.3. Arquitectura

La arquitectura está basada en capas. Las más altas contienen los servicios y tecnologías más importantes para el desarrollo de aplicaciones. En cuanto a las más bajas, controlan los servicios básicos.

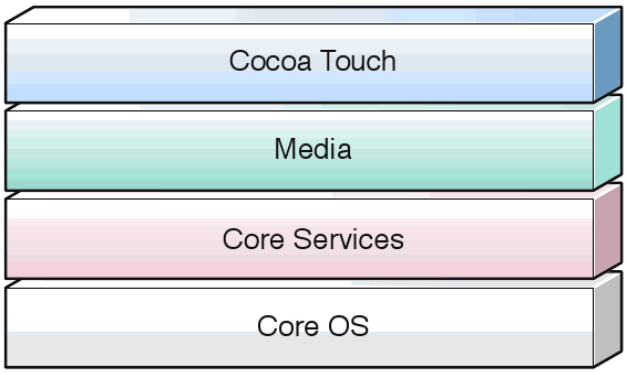


Ilustración 1.- Arquitectura iOS
Referencia:https://sites.google.com/site/tecnologiaiostm/desarrollo-de-aplicaciones/arquitectura-ios

A continuación, se detallarán las diferentes capas que se muestras en la ilustración 1:

- *Cocoa Touch*
  Es la más importante para el desarrollo de aplicaciones para iOS. Posee un conjunto de Frameworks, de los cuales, los dos fundamentales son: UIKit, que contiene todas las clases necesarias para el desarrollo de una interfaz y Foundation Framework, que define clases básicas, acceso, manejo de objetos y servicios del sistema operativo.

- *Media*
  Provee los servicios de gráficos y multimedia a la capa superior.

- *Core Services*
  Contiene los servicios fundamentales del sistema que usan todas las aplicaciones.

- *Core OS*
  Contiene las características de bajo nivel: ficheros del sistema, manejo de memoria, seguridad y drivers del dispositivo.

2.4. Tipos de datos

Con el fin de que el lector de este proyecto se familiarice con los diferentes tipos de datos que se han utilizado en la aplicación, y por consiguiente, con el fin de entender los apartados que siguen, a continuación se muestra un resumen de lo que se ha utilizado.

-id: Es un tipo que hace referencia al objeto o a una clase, sin importar el tipo del objeto en cuestión, pudiendo llamar a los métodos de ese objeto o clase,

bastando simplemente con saber que el objeto perteneciente a esa id que se está manipulando tiene dicho método.

-**UIRefreshControl**: A partir de iOS 6, se implementó este objeto con el fin de que una vez que el usuario arrastrara una tabla hacia abajo, ésta se volvería a cargar o actualizar en consecuencia.

-**UISearchBar**: Implementa un text field para hacer búsquedas basadas en texto. Se usa el delegado de este objeto, UISearchBarDelegate, con el fin de implementar las acciones cuando el texto ha sido introducido y los botones son pulsados.

-**MPMoviePlayerController**: Se trata de un reproductor de películas que gestiona las reproducciones de vídeo desde un archivo o en *streaming* desde la red.

-**UITableView**: Muestra una lista de *ítems* en una columna. Es una subclase de UISrollView, la cual permite subir y bajar por la tabla, aunque sólo verticalmente. El tamaño es controlado por las secciones y las filas que tengan cada sección y la apariencia puede ser diseñada por celdas.

-**UISegmentedControl**: Es un objeto hecho por múltiples segmentos o pestañas. Cada segmento funciona como un botón.

-**UIBarButtonItem**: Es un botón especial que se pone en un UIToolbar o en UINavigationBar. UIToolbar es una barra para colocar botones o elementos, mientras que UINavigationBar es la barra de arriba de las aplicaciones que permite al usuario moverse entre diferentes vistas.

-**NSArray**: Este clase es estática. Almacena objetos pero no puede ser modificado en la ejecución de la aplicación.

-**NSMutableArray**: Es un array de objetos que puede ser modificado durante la ejecución de la aplicación.

-**IBAction**: Es un calificador que es usado por Interface Builder para sincronizar acciones. Se usa para devolver una acción en la aplicación desde los métodos que se usen.

-**CGRect**: Se trata de una estructura que contiene la ubicación y las dimensiones de un rectángulo.

-**UICollectionView**: Esta clase gestiona una colección ordenada de elementos de datos y los presenta adaptándolos al diseño. Proporciona la misma función general de las UITableView, excepto que UICollectionView puede soportar más de una columna.

-**UIButton**: Se trata de una clase que implementa un botón en la pantalla. Intercepta eventos y envía acciones a los objetos que tienen como objetivos al pulsarlos.

-**UIScrollView**: Proporciona soporte para la visualización de contenido que es más grande que el tamaño de la vista de la aplicación. Permite a los usuarios desplazarse dentro de ese contenido, así como acercarse y alejarse.

-**UIPageControl**: Crea y administra controles de página. Muestra una serie horizontal de puntos, cada uno de los cuales corresponde a una página en el documento de la aplicación.

-**UITextView**: Esta clase implementa el comportamiento de una región de texto desplazable y de varias líneas. Muestra el texto con un estilo personalizado y además permite su modificación.

-**UITextField**: Es un objeto que muestra texto editable y envía una acción a un objeto cuando el usuario pulsa el botón de retorno. Normalmente, se utiliza para recoger pequeñas cantidades de texto del usuario y hacer una reacción inmediata.

-**UIImage**: Este objeto es la forma para mostrar una imagen.

-**CGPoint**: Es una estructura que contiene un punto concreto en un sistema de coordenadas bidimensionales.

-**UIImageView**: Es un objeto que proporciona un contenedor basado en la vista para mostrar, ya sea una sola imagen o para la animación de una serie de imágenes. Así mismo, proporciona controles para ajustar la duración y frecuencia de dicha animación.

-**MKMapItem**: Esta clase encapsula información sobre un punto específico en el mapa. Esta información incluye la localización en el mapa y otros datos que puedan ser relevantes.

-**MKLocalSearchRequest**: Este objeto es utilizado para especificar los parámetros de búsqueda basados en mapas. Se puede asignar una cadena de caracteres en lenguaje natural que contiene la dirección o punto de interés que se desea buscar.

-**UIPickerView**: Esta clase implementa objetos, llamados PickerView, que utilizan una “rueda” para mostrar uno o más conjuntos de valores. Los usuarios seleccionan los valores mediante la rotación de la rueda, de manera que la fila deseada se alinea con un indicador de selección.

-**CLLocation**: Este objeto incorpora las coordenadas geográficas y altitud de la ubicación del dispositivo junto con valores que indican la precisión de medición y cuándo se hicieron dichas mediciones.

-**UILabel**: Esta clase implementa una vista de texto de sólo lectura. Se utiliza para dibujar una o varias líneas de texto estático.

-**UINavigationItem**: Este objeto gestiona los botones y las vistas que se muestran en un UINavigationBar.

-**MKMapView**: Este objeto proporciona una interfaz de mapa, similar a la que se puede observar en la aplicación de Mapas de Apple. Se puede centrar el mapa en una coordenada determinada, especificar el tamaño del área que se quiere mostrar y anotar el mapa con información personalizada.

-**MKPlacemark**: Almacena la posición para una latitud y longitud determinada. Contiene información acerca del país, estado, ciudad y la dirección de la calle asociada a la coordenada especificada.

2.5. Cómo hacer una aplicación con Objective-C

Programar en Objective-C es algo diferente al resto de lenguajes como pueden ser Java o C++. El mejor IDE para hacerlo es el que proporciona Apple, llamado Xcode. Este entorno de desarrollo usa Interface Builder para construir la interfaz visual de la aplicación.

Para entender cómo hacer una aplicación en Objective-C y todas las partes implicadas, lo mejor será hacer una pequeña aplicación mostrando su funcionamiento. Esta app mostrará por pantalla: “Hola Mundo”.

Para ello, se creará un nuevo proyecto y se tendrá que elegir el tipo de aplicación que se vaya a implementar. En este caso, el tipo Single View Application es suficiente, el cual creará una vista en el *Storyboard* y unas cuantas clases que se explicarán a continuación.

El *Storyboard* es usado por Interface Builder para crear la interfaz de la aplicación. Así, tras dar un nombre a la aplicación, (por ejemplo HelloWorld), se creará un archivo en la aplicación llamado “Main.storyboard”. Al pulsar sobre él, aparecerá una vista vacía con todas las herramientas y opciones para empezar a personalizar la vista.

En el *storyboard* cada vista que se agregue se le asignará a una clase UIViewController.

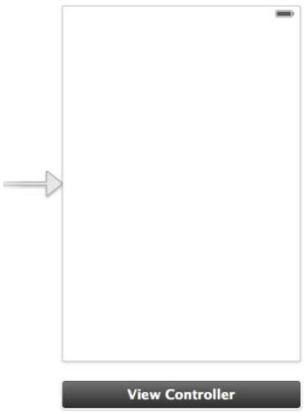


Ilustración 2.- Vista de un View Controller

Así mismo, el proyecto habrá creado cuatro archivos más. Dos que se llaman AppDelegate.h y otro AppDelegate.m. Todas las clases de Objective-C están divididas en dos ficheros con extensión .h y .m. El fichero .h se utilizará para la cabecera de la clase y el .m para la implementación. Así, los otros dos ficheros se llamarán ViewController.h y ViewController.m

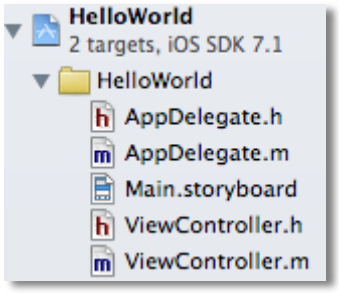


Ilustración 3.- Menú de clases

El AppDelegate es el "corazón de la aplicación". Un objeto delegado es un objeto que recibe una notificación cuando el objeto al que está conectado alcanza ciertos eventos y estados. En este caso, el AppDelegate es un objeto que recibe notificaciones cuando el objeto UIApplication, que es quien proporciona un punto de control y coordinación para las aplicaciones, alcanza ciertos estados.

El tipo UIViewController es el controlador de la vista de la interfaz. Es el lugar natural para coordinar acciones. Por ejemplo, cuando se pulsa un botón, se envía un mensaje al ViewController. Aunque la vista en sí misma puede ser ignorante de la tarea que realiza, se espera que el ViewController entienda lo que significa al pulsar un botón y la forma en que debe responder. El controlador puede actualizar la información de los objetos, animar o cambiar los valores de alguna propiedad almacenados en las vistas.

La ilustración 4 muestra un esquema para ver más claramente dónde está la función de cada elemento.

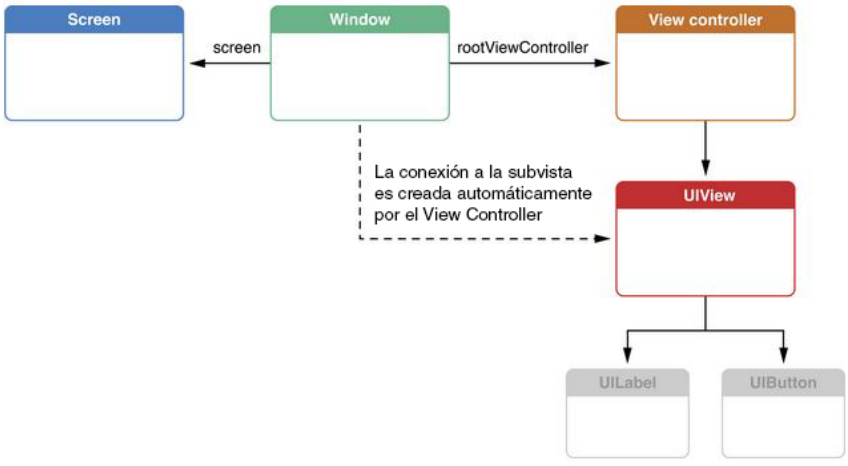


Ilustración 4.- Jerarquía de una vista
Referencia:https://developer.apple.com/library/ios/featuredarticles/viewcontrollerpgforiphoneos/AboutViewControllers/AboutViewControllers.html#//apple_ref/doc/uid/TP40007457-CH112-SW10

El objeto UIScreen identifica la pantalla física conectada al dispositivo. El objeto UIWindows proporciona soporte de dibujo para la pantalla. Un conjunto de objetos UIView realizan el dibujo. Estos objetos se adjuntan a la ventana y sacan sus contenidos cuando la ventana se lo pide.

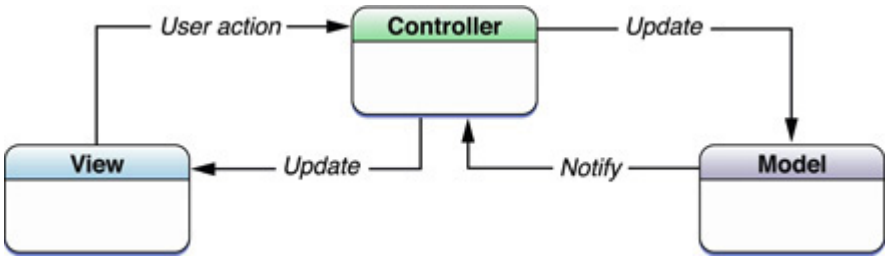


Ilustración 5.- Modelo-Vista-Controlador
Referencia:https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html

En este gráfico (véase, Ilustración 5) se muestra esta arquitectura (Modelo-Vista-Controlador) que separa los datos, la interfaz del usuario y la lógica de control en tres componentes. Para el caso de iOS, cada elemento se puede describir de la siguiente manera: Core Data(Modelo), UIView(Vista), UIViewController (Controlador).

Por lo tanto, en la pequeña app HelloWorld, al introducir un *label* en la vista UIView, se le conecta a la clase ViewController.h y se crea una propiedad del tipo UILabel llamada *label*. Entonces, en el ViewController.m se le da un valor a *label*, por ejemplo, *@"Hola Mundo"*. De esta manera, al arrancar la aplicación, el ViewController modificará el *label* y lo mostrará por pantalla con el nombre indicado.

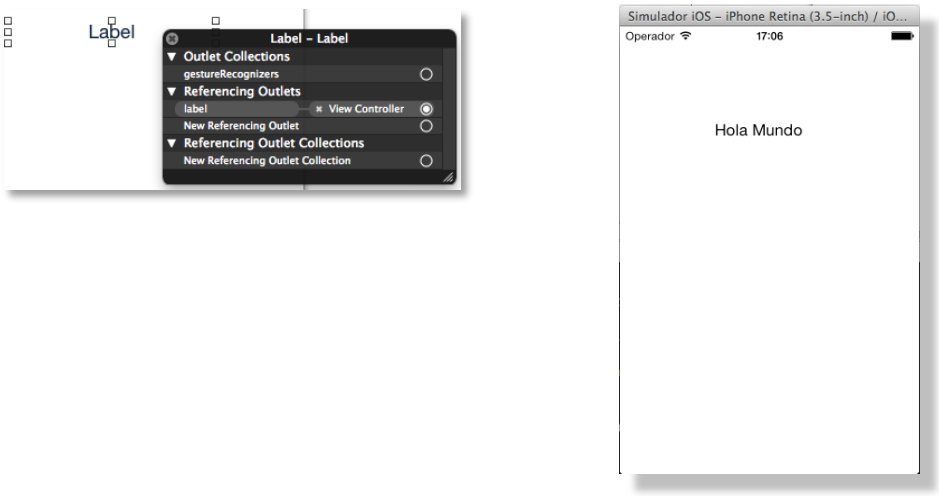


Ilustración 6.- Resultado final

3. Planteamiento inicial.

A continuación se detallará el planteamiento inicial de la aplicación. Se explicarán los objetivos que se persiguen, una planificación de las tareas a realizar y las herramientas que se utilizarán. Además se detallará una gestión de riesgos y una evaluación económica de la aplicación.

3.1. Objetivos

El objetivo que persigue esta aplicación, es la de ayudar a los viajeros a obtener todo lo que se necesita a la hora de viajar y hacer turismo, no teniendo que cargar encima con mapas, libros, hojas, etc. Solamente tener acceso a un dispositivo móvil y consultar todo desde ahí.

Si se da el caso de que no se disponga un contrato con tarifa de datos, la información siempre puede ser descargada previamente desde casa o descargarla cuando se acceda a una red con WIFI. Habiendo descargado la información de esa manera, el usuario tendría la información en local y podría ir consultándola sin que suponga un gasto adicional. Esto hace que la aplicación vaya a estar preparada tanto para la consulta *online* como *offline* de la información, controlando el usuario la información que quiera descargarse.

En el caso de que se disponga tarifa de datos, el usuario tendrá la posibilidad de ver un mapa lo que le separa del lugar al que quiere llegar y el punto de dónde se encuentra en esos momentos.

Junto con toda la utilidad de la aplicación, ésta se apoyará en ofrecer conocimientos y que los usuarios de la aplicación compartan ese conocimiento de los sitios y monumentos de las ciudades del mundo y enriquecer el turismo y hacerlo más divertido, accesible y fácil para las personas que utilicen la aplicación. Dará la posibilidad de conocer a gente, entablar amistades y viajar con ellas si se desea, compartir los viajes que al usuario le gustaría hacer y conocer a gente con sus mismos gustos y preferencias.

Y eso mismo es lo que se persigue, que el usuario comparta sus viajes, sus conocimientos, sus fotos, su viaje, de manera que todo el mundo salga ganando y pueda acceder a la información fácilmente, de manera sostenida y sin dañar el medioambiente.

Para que esto se cumpla, se hará la aplicación para el sistema operativo iOS de Apple, con lenguaje Objective-C. Y por consiguiente y como último objetivo, aprender nuevos conocimientos del mundo de las aplicaciones móviles que se basen en iOS, para en un futuro hacer nuevas aplicaciones.

3.2. Alcance

El ciclo de vida que tendrá este proyecto será el de prototipos incrementales. Este modelo entrega el software en partes pequeñas, pero utilizables. Cada parte se va construyendo sobre aquella que ya ha sido entregada y probada.

Además, en posteriores versiones permite incrementar la aplicación con nuevas funcionalidades.

A continuación, se detallarán las diferentes etapas del proyecto:

- Análisis: En esta etapa se analizará todo lo relacionado con la idea de la aplicación. Se estudiará y se usarán diferentes aplicaciones dedicadas al turismo o aplicaciones que te ofrezcan información sobre regiones o ciudades para recoger ideas tanto en funcionalidades como en asuntos de interfaz. Se diseñará, a grandes rasgos, las interfaces en papel, introduciendo ideas y bocetos de las funcionalidades y objetivos que vaya a satisfacer la aplicación.

  Por último, se utilizarán las herramientas necesarias junto con libros y tutoriales en la red para el aprendizaje del lenguaje Objective-C, el funcionamiento de entorno de desarrollo, etc.

- Planificación: Se determinará de cuántas tareas va a necesitar el proyecto y establecer así los tiempos necesarios, así como las funcionalidades que debe tener la aplicación. Se hará una captura de requisitos para documentar las funcionalidades con casos de uso y modelo de dominio.

  Por otra parte, se recogerá información de diferentes ciudades, monumentos, lugares, etc., que servirá como información para la aplicación.

- Diseño: Se diseñarán las interfaces con ayuda de las creadas en papel y se harán los cambios pertinentes sobre esa idea principal. Se diseñarán las bases de datos y las relaciones entre tablas. A su vez, se creará la estructura del proyecto con los diagramas de clase y de secuencia.

- Implementación: En esta etapa se irán desarrollando las funcionalidades establecidas en la planificación y diseño. Durante el proyecto, se han ido modificando en diferentes momentos detalles de la interfaz de la aplicación.

  - Prototipo 1: Creación de las bases de datos y añadir información.
  - Prototipo 2: Consultar “ciudades”, organización, búsquedas e información a mostrar, borrar y descarga y subida de la imagen del metro de la ciudad.
  - Prototipo 3: Crear, consultar, organizar, buscar y mostrar información relacionada sobre los “lugares”.
  - Prototipo 4: Mostrar en el mapa el “lugar” obtenido y la posición del usuario mediante geolocalización. A su vez, crear la ruta más corta y distancia que separa ambos lugares.
  - Prototipo 5: Escribir y mostrar comentarios de los usuarios de los “lugares”.
  - Prototipo 6: Descargar, subir, organizar, compartir y borrar fotos.

- Prototipo 7: Obtener vídeos del servidor, subirlos, reproducirlos y almacenarlos en el dispositivo.
- Prototipo 8: Creación y descarga de rutas, así como su consulta y organización.
- Prototipo 9: Creación de guías completas o que al menos contengan una ciudad y un lugar.
- Prototipo 10: Registro, inicio y cerrar sesión de un usuario, así como la modificación de los datos.
- Prototipo 11: Obtener creaciones del usuario y permitir borrar sus guías.
- Prototipo 12: Puntuar/Quitar puntuación y mostrar las puntuaciones del usuario.
- Prototipo 13: Obtener las amistades de los usuarios, así como la gestión de ellas. Búsqueda de usuarios y sugerencias de los más participativos en la aplicación.
- Prototipo 14: Obtener, borrar y mandar mensajes de los usuarios.

- Realización de pruebas: Se garantizará que el proyecto no contenga errores. Se harán pruebas de todas las funcionalidades del proyecto, así como de su integridad. Se comprobará que el resultado final de la aplicación satisface la planificación principal.

  - Realización de prueba 1: Pruebas de consulta, inserción, borrado y actualización en la base de datos del servidor y de la local.
  - Realización de pruebas 2: Pruebas relacionadas con la ciudad.
  - Realización de pruebas 3: Pruebas relacionadas con los lugares.
  - Realización de pruebas 4: Pruebas relacionadas con el mapa.
  - Realización de pruebas 5: Pruebas relacionadas con los comentarios
  - Realización de pruebas 6: Pruebas relacionadas con las fotos.
  - Realización de pruebas 7: Pruebas relacionadas con los vídeos.
  - Realización de pruebas 8: Pruebas relacionadas con las rutas.
  - Realización de pruebas 9: Pruebas relacionadas con la creación de una guía completa.
  - Realización de pruebas 10: Pruebas relacionadas con el registro, inicio de sesión y modificación de los datos de un usuario.
  - Realización de pruebas 11: Pruebas relacionadas la visualización y borrado de las creaciones del usuario.
  - Realización de pruebas 12: Pruebas relacionadas con las puntuaciones de un usuario.
  - Realización de pruebas 13: Pruebas relacionadas con las sugerencias y búsqueda de usuarios, así como su visualización y gestión de las amistades.
  - Realización de pruebas 14: Pruebas relacionadas con el envío y borrado de mensajes.

- Documentación final: Se concluirá la memoria, incluyendo los pasos anteriormente descritos. Y para finalizar, se realizará la presentación con la defensa del proyecto.

3.2.1. EDT

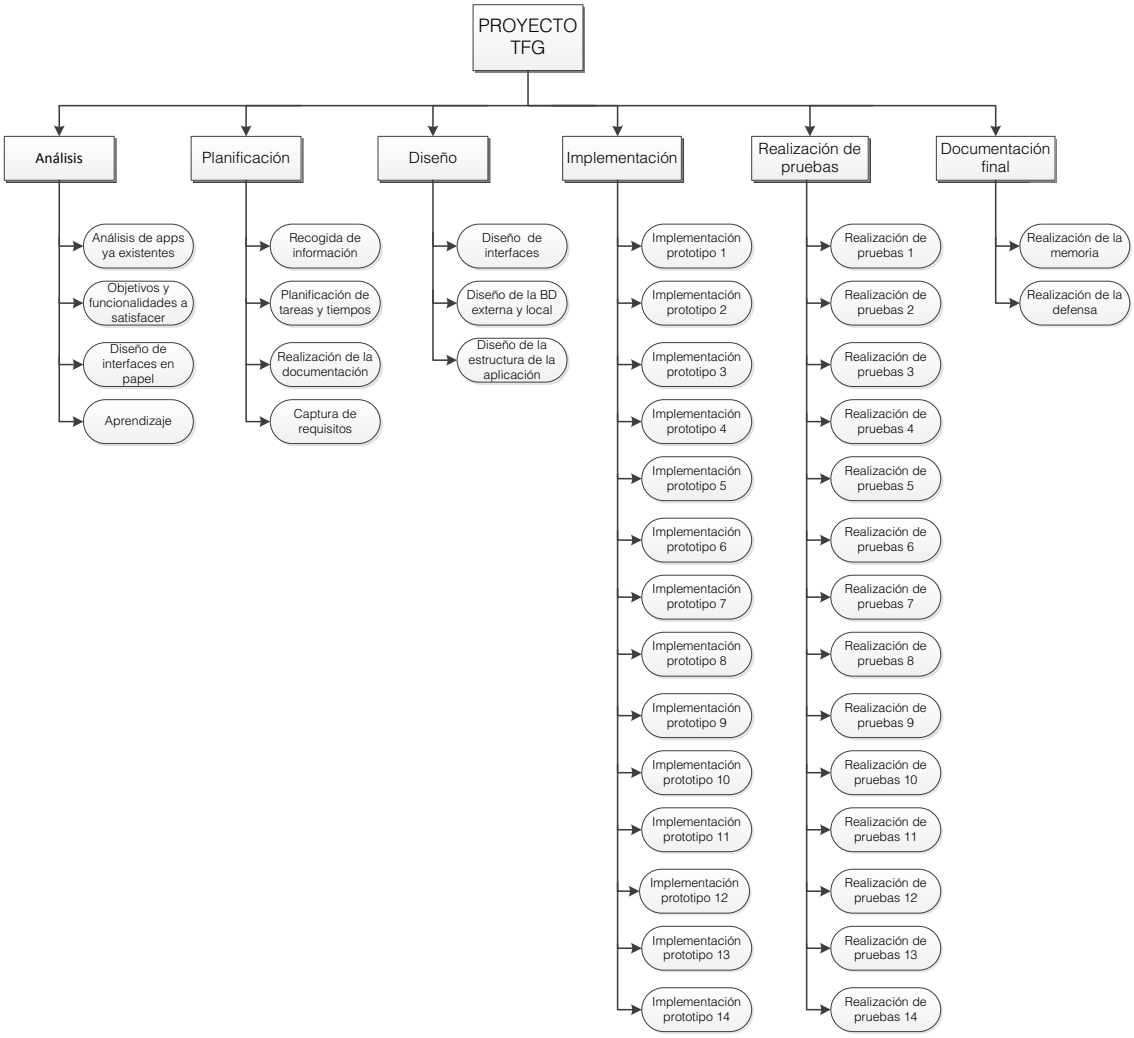


Ilustración 7.- Diagrama EDT

3.2.2. Tareas

-Análisis

Paquete de trabajo: Análisis de apps ya existentes.

Duración: 5 horas.

Descripción:

Realizar un análisis detallado de las aplicaciones sobre turismo y viajes que existen en la tienda de Apple.

Salidas:

Apuntes sobre ideas recabadas.

Recursos:

Tener un *smartphone* para probar las aplicaciones.

Paquete de trabajo: Objetivos y funcionalidades a satisfacer.

Duración: 5 horas

Descripción:

Redacción de los objetivos y funcionalidades de la aplicación, así como redactar la hoja de propuesta de Trabajo Fin de Grado.

Salidas:

Información sobre qué hará la aplicación, objetivos y descripción del proyecto.

Paquete de trabajo: Diseño de interfaces en papel.

Duración: 6 horas

Descripción:

Borrador para crear una idea general del proyecto y cómo se mostrará en la aplicación, indicando dónde se encontrarán las funcionalidades descritas anteriormente.

Entradas:

Esquema con los objetivos y funcionalidades.

Salidas:

Bocetos de las interfaces de la aplicación.

Precedencias:

Tener realizados los objetivos y funcionalidades que se van a realizar.

Paquete de trabajo: Aprendizaje.

Duración: 75 horas

Descripción:

Autoaprendizaje sobre las herramientas a utilizar y lenguaje para programar mediante libros y tutoriales en la red.

Recursos:

Libros de programación y tutoriales.

-Planificación

Paquete de trabajo: Recogida de información.

Duración: 9 horas

Descripción:

Recogida de información relevante para la aplicación. Se recolecta información de varios lugares para posteriormente poder visualizar en la aplicación, como pueden ser monumentos, museos, ciudades, etc.

Salidas:

Información para almacenar en la base de datos.

Paquete de trabajo: Planificación de tareas y tiempos.

Duración: 7 horas

Descripción:

Describir en qué consiste y estimar el tiempo para realizar cada tarea.

Precedencias:

Identificar las tareas.

Paquete de trabajo: Realización de la documentación.

Duración: 12 horas

Descripción:

Realización de la documentación donde aparecerá la introducción, descripción del proyecto, objetivos, herramientas a utilizar, las tareas y estaciones de tiempo, diagrama de Gantt, gestión de riesgos y viabilidad económica.

Salidas:

Documentación del proyecto.

Precedencias:

Tener realizada la planificación de tareas y tiempos, así como los objetivos del proyecto.

Paquete de trabajo: Captura de requisitos.

Duración: 15 horas.

Descripción:

Realización de los casos de uso y modelos de dominio, definiendo las funcionalidades y los datos que tendrá que permitir y aportar la aplicación.

Salidas:

La documentación formal de las funcionalidades de la aplicación.

Precedencias:

Tener claros los objetivos y las funcionalidades.

-Diseño

Paquete de trabajo: Diseño de las interfaces.

Duración: 10 horas.

Descripción:

Cambiar lo planificado en el boceto de las interfaces acorde a lo establecido en la documentación y captura de requisitos, para satisfacer los objetivos y que se cumplan las funcionalidades.

Precedencias:

Planificación de las funcionalidades y objetivos.

Paquete de trabajo: Diseño de la BD externa y local.

Duración: 10 horas

Descripción:

Diseñar las bases de datos que se van a utilizar en la aplicación y las relaciones entre tablas.

Salidas:

Boceto final de las bases de datos.

Paquete de trabajo: Diseño de la estructura de la aplicación.

Duración: 22 horas

Descripción:

Realización de diagramas de secuencia y la estructura que tendrá la aplicación, mostrando el comportamiento de las funcionalidades y los algoritmos que se realicen.

Entradas:

Documentación de funcionalidades.

Precedencias:

Análisis y planificación.

-Implementación

Paquete de trabajo: Implementación prototipo 1.

Duración: 10 horas

Descripción:

Implementación de la base de datos y la inserción de la información para hacer uso en la aplicación.

Precedencias:

Diseño de la estructura de la aplicación.

Paquete de trabajo: Implementación prototipo 2.

Duración: 40 horas

Descripción:

Implementación de todo lo relacionado con las Ciudades. La consulta, organización, descarga, borrado, búsqueda y la información a mostrar de las ciudades. Además, la descarga y subida de la imagen del metro de la ciudad.

Precedencias:

Pruebas del prototipo 1.

Paquete de trabajo: Implementación prototipo 3.

Duración: 45 horas

Descripción:

Implementación de todo lo relacionado con los lugares. La consulta, organización, creación, búsqueda, borrado y la información a mostrar de los lugares.

Precedencias:

Pruebas del prototipo 2.

Paquete de trabajo: Implementación prototipo 4.

Duración: 10 horas

Descripción:

Implementación del mapa. Marcar en el mapa el lugar elegido y crear una ruta entre el lugar marcado mostrando su distancia.

Precedencias:

Pruebas del prototipo 3.

Paquete de trabajo: Implementación prototipo 5.

Duración: 11 horas

Descripción:

Implementación para mostrar y escribir los comentarios de los lugares.

Precedencias:

Pruebas del prototipo 4.

Paquete de trabajo: Implementación prototipo 6.

Duración: 38 horas

Descripción:

Implementación de la descarga, subida, organización de las fotos. También compartirlas y borrarlas.

Precedencias:

Pruebas del prototipo 5.

Paquete de trabajo: Implementación prototipo 7.

Duración: 30 horas

Descripción:

Implementación de la descarga, subida, reproducción, organización y borrado de las vídeos.

Precedencias:

Pruebas del prototipo 6.

Paquete de trabajo: Implementación prototipo 8.

Duración: 25 horas

Descripción:

Implementación para crear, descargar, borrar y consultar rutas de las ciudades.

Precedencias:

Pruebas del prototipo 7.

Paquete de trabajo: Implementación prototipo 9.

Duración: 25 horas

Descripción:

Implementación para hacer guías completas o que al menos contengan una ciudad y un lugar.

Precedencias:

Pruebas del prototipo 8.

Paquete de trabajo: Implementación prototipo 10.

Duración: 28 horas

Descripción:

Implementación del registro de un usuario, así como el inicio y cierre de sesión, consultar la información, modificarla y borrar la cuenta.

Precedencias:

Pruebas del prototipo 9.

Paquete de trabajo: Implementación prototipo 11.

Duración: 20 horas

Descripción:

Implementación para obtener todas las creaciones del usuario y permitir borrarlas.

Precedencias:

Pruebas del prototipo 10.

Paquete de trabajo: Implementación prototipo 12.

Duración: 12 horas

Descripción:

Implementación para puntuar o quitar la puntuación de las guías y mostrar en el perfil del usuario.

Precedencias:

Pruebas del prototipo 11.

Paquete de trabajo: Implementación prototipo 13.

Duración: 32 horas

Descripción:

Implementación para obtener y mostrar las amistades de los usuarios, así como la gestión de ellas. Búsqueda de usuarios y sugerencias de los que más participan creando guías en la aplicación.

Precedencias:

Pruebas del prototipo 12.

Paquete de trabajo: Implementación prototipo 14.

Duración: 22 horas

Descripción:

Implementación para obtener, borrar y mandar mensajes entre los usuarios.

Precedencias:

Pruebas del prototipo 13.

-Realización de pruebas

Paquete de trabajo: Realizar pruebas 1.

Duración: 3 horas

Descripción:

Realizar pruebas de consulta, inserción, borrado y actualización en la base de datos del servidor y la base de datos local.

Salidas:

Prototipo 1 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 1.

Paquete de trabajo: Realizar pruebas 2.

Duración: 5 horas

Descripción:

Realizar pruebas para la correcta visualización de las ciudades y el correcto funcionamiento con ellas descargando, borrando, buscando, etc.

Salidas:

Prototipo 2 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 2.

Paquete de trabajo: Realizar pruebas 3.

Duración: 4 horas

Descripción:

Realizar pruebas para la correcta visualización de los lugares de una ciudad y el correcto funcionamiento de la descarga, borrado, búsqueda, etc.

Salidas:

Prototipo 3 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 3.

Paquete de trabajo: Realizar pruebas 4.

Duración: 2 horas

Descripción:

Realizar pruebas para mostrar los lugares en el mapa y que se dibuje la ruta entre el usuario y el lugar marcado en el mapa.

Salidas:

Prototipo 4 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 4.

Paquete de trabajo: Realizar pruebas 5.

Duración: 1 horas

Descripción:

Realizar pruebas para mostrar y escribir los comentarios.

Salidas:

Prototipo 5 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 5.

Paquete de trabajo: Realizar pruebas 6.

Duración: 5 horas

Descripción:

Realizar pruebas para la correcta visualización de las fotos y que funcione correctamente la descarga, compartir las fotos, borrado de fotos, etc.

Salidas:

Prototipo 6 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 6.

Paquete de trabajo: Realizar pruebas 7.

Duración: 4 horas

Descripción:

Realizar pruebas para la correcta visualización y reproducción de los vídeos y que funcione correctamente la descarga, borrado, etc.

Salidas:

Prototipo 7 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 7.

Paquete de trabajo: Realizar pruebas 8.

Duración: 2 horas

Descripción:

Realizar pruebas para la correcta visualización de las rutas de la ciudad y que funciona correctamente la descarga, la creación y borrado de las rutas.

Salidas:

Prototipo 8 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 8.

Paquete de trabajo: Realizar pruebas 9.

Duración: 3 horas

Descripción:

Realizar pruebas para crear guías completas, intercalando la creación de una ciudad y varios lugares y después subiendo fotos, vídeos, etc.

Salidas:

Prototipo 9 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 9.

Paquete de trabajo: Realizar pruebas 10.

Duración: 2 horas

Descripción:

Realizar pruebas registrando usuarios, iniciando y cerrando sesión y actualizando los datos del usuario que ha iniciado sesión.

Salidas:

Prototipo 10 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 10.

Paquete de trabajo: Realizar pruebas 11.

Duración: 2 horas

Descripción:

Realizar pruebas para la correcta visualización de las creaciones del usuario y la posibilidad de borrarlas.

Salidas:

Prototipo 11 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 11.

Paquete de trabajo: Realizar pruebas 12.

Duración: 1 hora.

Descripción:

Realizar pruebas puntuando y quitando las puntuaciones de lugares, rutas, fotos y vídeos. Además, comprobar la correcta visualización de las votaciones del usuario.

Salidas:

Prototipo 12 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 12.

Paquete de trabajo: Realizar pruebas 13.

Duración: 2 horas

Descripción:

Realizar pruebas de la correcta visualización de las amistades del usuario y la comprobación del cambio de estados al pulsar el botón. Además comprobar que se muestran las sugerencias de usuario y la búsqueda de un usuario concreto.

Salidas:

Prototipo 13 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 13.

Paquete de trabajo: Realizar pruebas 14.

Duración: 2 horas

Descripción:

Realizar pruebas mandando mensajes entre usuarios y su correcta visualización en los buzones de ambos usuarios y que se borren correctamente.

Salidas:

Prototipo 14 funcionando correctamente.

Recursos:

La aplicación.

Precedencias:

Implementación prototipo 14.

-Documentación final

Paquete de trabajo: Realización de la memoria.

Duración: 30 horas

Descripción:

Realizar la memoria del proyecto.

Entradas:

Documentos realizados en la planificación y diseño.

Precedencias:

Planificación, diseño, implementación y pruebas.

Paquete de trabajo: Realización de la defensa.

Duración: 15 horas

Descripción:

Preparar la presentación y defensa del proyecto.

Entradas:

La aplicación y memoria terminada.

Salidas:

Presentación del proyecto.

3.3. Planificación temporal

Código tarea	Tarea	Fecha Inicio	Fecha fin	Horas
1	**Análisis**	**30/09/2013**	**27/10/2013**	**91**
1.1	Análisis de apps existentes	30/09/2013	03/10/2013	5
1.2	Objetivos y funcionalidades a satisfacer	04/10/2013	05/10/2013	5
1.3	Diseño de interfaces en papel	06/10/2013	07/10/2013	6
1.4	Aprendizaje	08/10/2013	27/10/2013	75
2	**Planificación**	**28/10/2013**	**06/11/2013**	**43**
2.1	Recogida de información	28/10/2013	29/10/2013	9
2.2	Planificación de tareas y tiempos	30/10/2013	31/10/2013	7
2.3	Realización de la documentación	01/11/2013	03/11/2013	12
2.4	Captura de requisitos	04/11/2013	06/11/2013	15
3	**Diseño**	**07/11/2013**	**19/11/2013**	**42**
3.1	Diseño de interfaces	07/11/2013	08/11/2013	10
3.2	Diseño de la BD externa y local	11/11/2013	12/11/2013	10
3.3	Diseño de la estructura de la aplicación	13/11/2013	19/11/2013	22
4	**Implementación**	**20/11/2013**	**21/03/2014**	**348**
4.1	Implementación prototipo 1	20/11/2013	22/11/2013	10
4.2	Implementación prototipo 2	25/11/2013	04/12/2013	40
4.3	Implementación prototipo 3	06/12/2013	18/12/2013	45

Tabla 1.- Planificación temporal

Código tarea	Tarea	Fecha Inicio	Fecha fin	Horas
4.4	Implementación prototipo 4	20/12/2013	22/12/2013	10
4.5	Implementación prototipo 5	08/01/2014	10/01/2014	11
4.6	Implementación prototipo 6	13/01/2014	22/01/2014	38
4.7	Implementación prototipo 7	24/01/2014	31/01/2014	30
4.8	Implementación prototipo 8	03/02/2014	07/02/2014	25
4.9	Implementación prototipo 9	10/02/2014	14/02/2014	25
4.10	Implementación prototipo 10	17/02/2014	21/02/2014	28
4.11	Implementación prototipo 11	24/02/2014	27/02/2014	20
4.12	Implementación prototipo 12	03/03/2014	05/03/2014	12
4.13	Implementación prototipo 13	07/03/2014	14/03/2014	32
4.14	Implementación prototipo 14	17/03/2014	21/03/2014	22
5	**Realización de pruebas**	**23/11/2013**	**22/03/2014**	**38**
5.1	Prueba 1	23/11/2013	23/11/2013	3
5.2	Prueba 2	05/12/2013	05/12/2013	5
5.3	Prueba 3	19/12/2013	19/12/2013	4
5.4	Prueba 4	23/12/2013	23/12/2013	2
5.5	Prueba 5	11/01/2014	11/01/2014	1
5.6	Prueba 6	23/01/2014	23/01/2014	5
5.7	Prueba 7	01/02/2014	01/02/2014	4
5.8	Prueba 8	08/02/2014	08/02/2014	2
5.9	Prueba 9	15/02/2014	15/02/2014	3
5.10	Prueba 10	22/02/2014	22/02/2014	2
5.11	Prueba 11	28/02/2014	28/02/2014	2
5.12	Prueba 12	06/03/2014	06/03/2014	1
5.13	Prueba 13	15/03/2014	15/03/2014	2
5.14	Prueba 14	22/03/2014	22/03/2014	2
6	**Documentación final**	**24/03/2014**	**07/04/2014**	**45**
6.1	Realización de la memoria	24/03/2014	29/03/2014	30
6.2	Realización de la defensa	31/03/2014	04/04/2014	15
7	**Proyecto**	**30/09/2013**	**04/04/2014**	**607**

Tabla 2.- Planificación temporal 2

3.3.1. Diagrama Gantt planificado

El diagrama de Gantt es una herramienta que tiene como objetivo mostrar gráficamente las tareas a lo largo del tiempo que se ha determinado.

A continuación, se mostrarán divididas en varias imágenes el diagrama de Gantt previsto para entregar el Trabajo Fin de Grado en abril. Se ha segmentado el diagrama en varias imágenes, separadas por las tareas principales, con el fin de mejorar su legibilidad.

-Análisis

Id	Nombre de tarea	Comienzo	Fin
1	**Análisis**	**lun 30/09/13**	**dom 27/10/13**
2	Análisis de apps existentes	lun 30/09/13	jue 03/10/13
3	Objetivos y funcionalidades a satisfacer	vie 04/10/13	sáb 05/10/13
4	Diseño de interfaces en papel	dom 06/10/13	lun 07/10/13
5	Aprendizaje	mar 08/10/13	dom 27/10/13

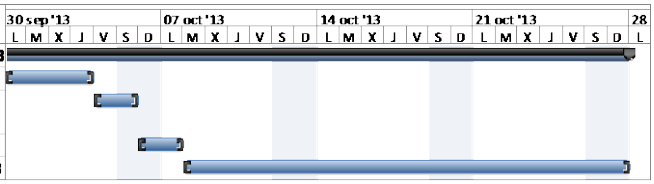


Ilustración 8.- Gantt. Análisis

-Planificación

Id	Nombre de tarea	Comienzo	Fin
6	**Planificación**	**lun 28/10/13**	**mié 06/11/13**
7	Recogida de información	lun 28/10/13	mar 29/10/13
8	Planificación de tareas y tiempos	mié 30/10/13	jue 31/10/13
9	Realización de la documentación	vie 01/11/13	dom 03/11/13
10	Captura de requisitos	lun 04/11/13	mié 06/11/13

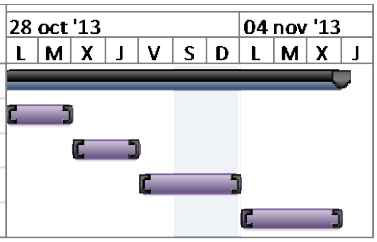


Ilustración 9.- Gantt. Planificación

-Diseño

Id	Nombre de tarea	Comienzo	Fin
11	**Diseño**	**jue 07/11/13**	**mar 19/11/13**
12	Diseño de interfaces	jue 07/11/13	vie 08/11/13
13	Diseño de la BD externa y local	lun 11/11/13	mar 12/11/13
14	Diseño de la estructura de la aplicación	mié 13/11/13	mar 19/11/13

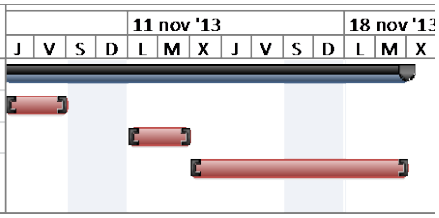
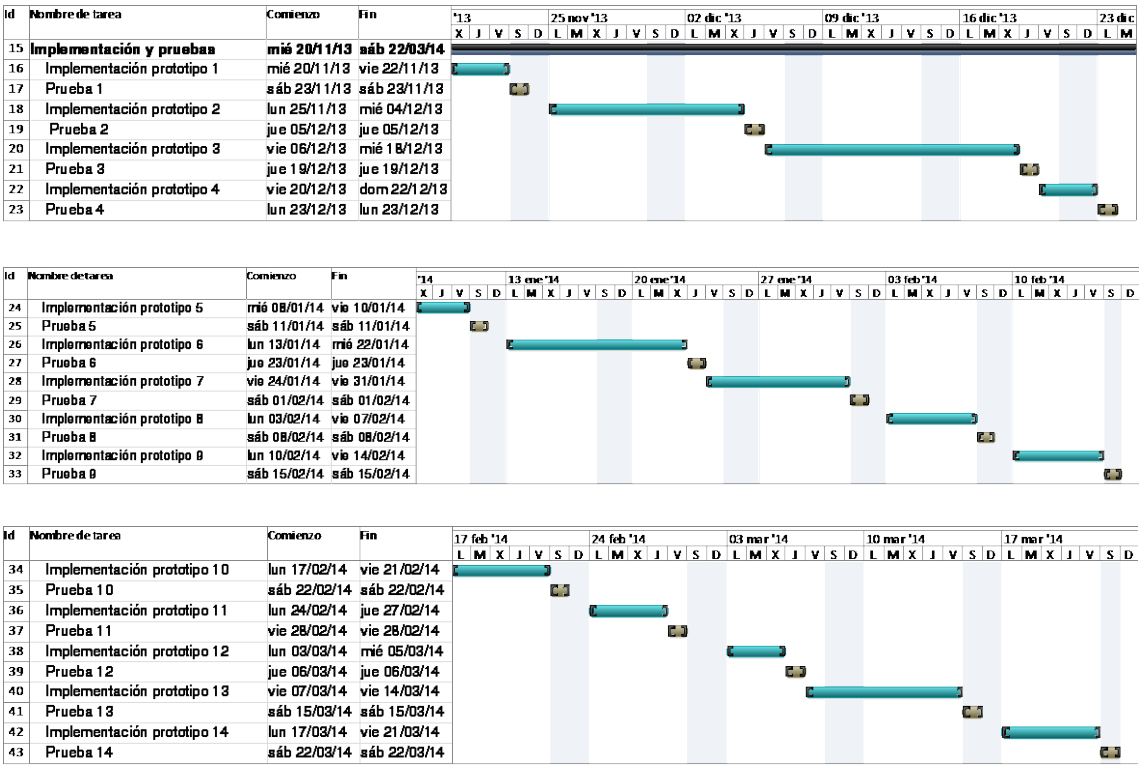


Ilustración 10.- Gantt. Diseño

-Implementación y pruebas

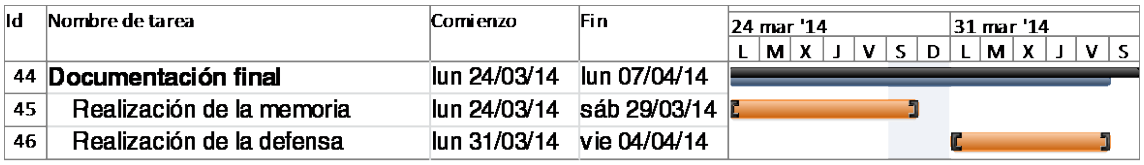
La parte de implementación y pruebas ha sido dividida en tres para verlo con más claridad. En medio de las dos primeras imágenes, están las vacaciones de navidad, en las cuales no se trabajará en el proyecto.



Id	Nombre de tarea	Comienzo	Fin
15	**Implementación y pruebas**	**mié 20/11/13**	**sáb 22/03/14**
16	Implementación prototipo 1	mié 20/11/13	vie 22/11/13
17	Prueba 1	sáb 23/11/13	sáb 23/11/13
18	Implementación prototipo 2	lun 25/11/13	mié 04/12/13
19	Prueba 2	jue 05/12/13	jue 05/12/13
20	Implementación prototipo 3	vie 06/12/13	mié 18/12/13
21	Prueba 3	jue 19/12/13	jue 19/12/13
22	Implementación prototipo 4	vie 20/12/13	dom 22/12/13
23	Prueba 4	lun 23/12/13	lun 23/12/13
24	Implementación prototipo 5	mié 08/01/14	vie 10/01/14
25	Prueba 5	sáb 11/01/14	sáb 11/01/14
26	Implementación prototipo 6	lun 13/01/14	mié 22/01/14
27	Prueba 6	jue 23/01/14	jue 23/01/14
28	Implementación prototipo 7	vie 24/01/14	vie 31/01/14
29	Prueba 7	sáb 01/02/14	sáb 01/02/14
30	Implementación prototipo 8	lun 03/02/14	vie 07/02/14
31	Prueba 8	sáb 08/02/14	sáb 08/02/14
32	Implementación prototipo 9	lun 10/02/14	vie 14/02/14
33	Prueba 9	sáb 15/02/14	sáb 15/02/14
34	Implementación prototipo 10	lun 17/02/14	vie 21/02/14
35	Prueba 10	sáb 22/02/14	sáb 22/02/14
36	Implementación prototipo 11	lun 24/02/14	jue 27/02/14
37	Prueba 11	vie 28/02/14	vie 28/02/14
38	Implementación prototipo 12	lun 03/03/14	mié 05/03/14
39	Prueba 12	jue 06/03/14	jue 06/03/14
40	Implementación prototipo 13	vie 07/03/14	vie 14/03/14
41	Prueba 13	sáb 15/03/14	sáb 15/03/14
42	Implementación prototipo 14	lun 17/03/14	vie 21/03/14
43	Prueba 14	sáb 22/03/14	sáb 22/03/14

Ilustración 11.- Gantt. Implementación y pruebas

-Documentación final



Id	Nombre de tarea	Comienzo	Fin
44	**Documentación final**	lun 24/03/14	lun 07/04/14
45	Realización de la memoria	lun 24/03/14	sáb 29/03/14
46	Realización de la defensa	lun 31/03/14	vie 04/04/14

Ilustración 12.- Gantt. Documentación final

Sin embargo, no siempre es posible seguir la planificación planteada, debido a desconocimiento del tema a tratar o que puedan surgir problemas o imprevistos con lo que no se esperaba. Todo ello supone un retraso en la planificación y en consecuencia, en la finalización del proyecto.

3.4. Herramientas

En este apartado se van a describir las herramientas utilizadas durante todo el proyecto.

- **Ordenador Mac**: Es necesario tener un ordenador de la gama de Apple ya que sólo se podrá programar en ellos de forma óptima. En este proyecto se ha utilizado un Mac Mini (2011).

-**iPhone 4**: El iPhone 4 es un dispositivo móvil de Apple que ha sido utilizado como simulador y como dispositivo de prueba a lo largo de la aplicación.

-**Sistema operativo**: Se ha utilizado la última versión del sistema operativo llamado OS X Mavericks, versión 10.9.1.

-**Xcode**: Utilizado como entorno de desarrollo e indispensable si se quiere crear aplicaciones para iOS. Las versiones que se han manejado durante todo el proyecto han sido las últimas: la 5.0 y 5.1.

-**Simulador iOS**: Se ha usado para simular las ejecuciones de lo programado en Xcode y así poder ver cómo queda la aplicación en sus diferentes etapas y probar su funcionamiento. Usado, a su vez, para entrenar y conocer más a fondo el lenguaje Objective-C.

-**Programa desarrollador de Apple**: No es un programa en sí, pero si una herramienta necesaria para hacer las pruebas de la aplicación en un dispositivo móvil o tableta real, en este caso, un iPhone.

-**MAMP**: Utilizada como aplicación que permite la ejecución de un servidor Apache y una base de datos MySQL, con el fin de crear una base de datos y hacer un banco de información para la aplicación. Se utilizó antes de tener un servidor externo.

-**Editores de ofimática**: Microsoft Office para Mac con el fin de poder realizar la documentación necesaria.

-**Navegadores Web**: Se ha utilizado Safari y Google Chrome para buscar tutoriales e información útil para aprender a programar y buscar información útil para el proyecto.

-**Photoshop**: Se ha utilizado este programa de manipulación de imágenes para crear los objetos en la interfaz gráfica de la aplicación, así como su icono y logo.

-**Sublime Text**: Entorno de desarrollo utilizado para crear archivos PHP necesarios para el proyecto.

-**Microsoft Visio**: Aplicación de Microsoft Office que se ha utilizado para hacer diagramas o esquemas relevantes para la documentación del proyecto. Por ejemplo, los diagramas de secuencia, casos de uso, etc.

-**Dropbox**: Programa o servicio en la nube que permite guardar archivos de todo tipo de manera externa a nuestro ordenador. Permite instalarlo en el equipo con el fin de acceder rápidamente a los archivos guardados en la nube.

-**iConify**: Es una aplicación rápida y sencilla de generar iconos para una aplicación de iOS. Permite crear iconos con diferentes tamaños.

-**CyberDuck**: Se ha usado este programa como cliente SFTP y así poder subir los archivos PHP, imágenes y vídeos al servidor. Permite navegar por la carpeta del servidor y modificar los archivos PHP.

-**Microsoft Project**: Es una herramienta para la programación y gestión de proyectos que aunque se instale por separado, pertenece a la familia de

Micosoft Office. Con ella, se ha creado la planificación temporal que tendrá el proyecto.

3.5. GESTIÓN DE RIESGOS

Con el fin de prevenir o minimizar un efecto negativo en el proyecto o en el transcurso de la creación de éste, se señalan a continuación una lista de prevención y gestión de los riesgos que puedan ocurrir a lo largo del proyecto.

Para ello, se anota como probabilidad baja, media o alta dependiendo de si es probable o no que ocurra el problema.

Descripción	Fallo de suministro de Internet por la compañía contratada.
Prevención	Ninguna.
Plan de contingencia	Posibilidad de ir a otro lugar seguro a trabajar.
Probabilidad	Media.
Impacto	Pérdida de tiempo en el caso de necesitar una búsqueda de información.

Tabla 3.- Gestión de riesgos 1

Descripción	Pérdida de información de la Base de Datos.
Prevención	Hacer copias de seguridad cada dos días.
Plan de contingencia	Recuperar la información desde el último back up realizado.
Probabilidad	Medio.
Impacto	Pérdida de toda la información guardada hasta el momento, con la consecuente pérdida de tiempo en recuperarla.

Tabla 4.- Gestión de riesgos 2

Descripción	Incapacidad de trabajo temporalmente por enfermedad o dolores de cabeza, consecuencia de estar tanto tiempo delante de la pantalla.
Prevención	Descansar durante breves periodos de tiempo.
Plan de contingencia	Despejarse y dejar la tarea aparcada durante un rato o hasta el día siguiente.
Probabilidad	Medio.
Impacto	Pérdida de tiempo y posponer lo planeado.

Tabla 5.- Gestión de riesgos 3

Descripción	Actualización de la versión en el entorno de programación y cambios en algunos de los "Frameworks". Estas actualizaciones, normalmente, se liberan al actualizarse el sistema operativo.
Prevención	Documentarse de los cambios que va a haber y si se va a producir una actualización antes de la finalización de la aplicación.
Plan de contingencia	Adecuar el código a la nueva actualización.
Probabilidad	Media.
Impacto	Código inservible para los dispositivos con la nueva actualización.

Tabla 6.- Gestión de riesgos 4

Descripción	Realizar una mala estimación.
Prevención	Procurar ser lo más realista posible.
Plan de contingencia	Reorganizar tareas y dedicar más horas al día para llegar los tiempos estipulados.
Probabilidad	Alta.
Impacto	Incumplimiento de los plazos y retrasos en las fechas señaladas, lo que provocaría un retraso en la entrega del proyecto.

Tabla 7.- Gestión de riesgos 5

Descripción	Pérdida de información en el disco duro por causas medioambientales o problemas de hardware causados por el trasiego del ordenador de casa a la universidad.
Prevención	Guardar periódicamente la información en la nube.
Plan de contingencia	Volver a cargar la información guardada de manera externa una vez obtenido el equipo reparado u otro diferente.
Probabilidad	Baja.
Impacto	Pérdida de todo lo avanzando hasta el momento.

Tabla 8.- Gestión de riesgos 6

Descripción	Infección en el ordenador por virus, troyano o cualquier tipo de malware, con la consecuente pérdida de información.
Prevención	Guardar periódicamente la información en la nube y tener actualizado el antivirus.
Plan de contingencia	Volver a cargar la información guardada de manera externa una vez solventado el problema o eliminado el virus.
Probabilidad	Media.
Impacto	Posible pérdida de información y en el peor de los casos, daño en el hardware del equipo.

Tabla 9.- Gestión de riesgos 7

3.6. EVALUACIÓN ECONÓMICA

La evaluación económica determina la rentabilidad de un proyecto, mediante el cual se facilita el proceso de toma de decisiones. A continuación, se muestra la viabilidad de este proyecto y así poder analizar los riesgos que conlleva hacerlo o no hacerlo.

3.6.1. Costes

En primer lugar, se considerarán costes directos los relacionados con los ordenadores y dispositivos utilizados, así como el precio de licencias y sueldo del programador.

En primer lugar, el coste relacionado con los ordenadores y dispositivos utilizados en el proyecto será del siguiente:

- Precio de compra de un ordenador Mac Mini: 550 €.
- Precio de compra de un iPhone 4: 320 €.

A la hora de amortizar un ordenador, en este caso el Mac Mini, se ajusta a cinco años. Es decir, por cada año el coste sería de 110€. Cómo la finalización de la aplicación se ha estimado que aproximadamente será de 6 meses, el coste total del Mac Mini será de 55€.

En cuando al dispositivo móvil iPhone, se habrá amortizado en 3 años. Por lo tanto, por cada año el coste sería de 106€. Como se ha estimado que el desarrollo de la aplicación sea de 6 meses, el coste total del iPhone será de 53€.

Por otro lado, para ser desarrollador de aplicaciones en Apple, se tiene que pagar una cuota anual. El precio de esta licencia es de 80€/año.

Debido a que sólo ha habido un trabajador en ese proyecto, se ha fijado un precio por hora que cobrará el programador. El sueldo será de: 15 €/hora. Por lo tanto, considerando que el proyecto ha tardado 607 horas (~184 días) en ser finalizado, el gasto total del programador es de:15*607= 9.105 €. Sin embargo, como mucho de ese tiempo ha sido gastado en aprendizaje, es de rigor restar esas horas para que el gasto sea más real (607-75=532). Por lo tanto, considerando esto último se puede determinar que el coste del programador sería de:
532 (**duración sin aprendizaje**) * 15 (**sueldo del programador**)= 7.980 €.

La suma **total** de los costes directos, pues, será de:
7.980 (**coste programador**) + 55(**coste Mac**) + 53(**coste iPhone**) + 80(**coste licencia**) = 8.168 €.

A su vez, se estiman gastos indirectos derivados de consumo eléctrico, utilización de mobiliario, etc. Por lo que finalmente el gasto total del proyecto será de:
Costes indirectos = 8168 (**costes directos)** * 0,05 = 408,4 €

Así pues, el coste **total** del proyecto es de: 8.576,4 €

3.6.2. Ingresos

Mediante el sistema de publicidad iAd, cada mil impresiones de publicidad a través de la aplicación, se recaudará 0,25€[1]. De esta manera:
(Coste total / ingreso publicidad)*1000 = 34.305.600 veces haría falta que la publicidad se mostrase para empezar a obtener beneficio.

Colocando la publicidad estratégicamente a través de la aplicación y considerando que la aplicación sólo será usada para viajar, se estima que de media un usuario vaya a ver 40 veces la publicidad al cabo de un año. Así, la aplicación tendría que ser descargada por:

34.305.600 (**Impresiones necesaria**) / 40 (**publ. vista por usuario al año**) = 857.640 usuarios.

Esta cantidad de descargas es demasiado grande, considerando que sólo las aplicaciones con mucho éxito llegan a esas cantidades de distribución. Por lo tanto, la manera de obtener ingresos por publicidad, no se llevará a cabo. Así que para obtener mayor rentabilidad, se pondrá la aplicación a un precio inicial de 0,89€. Sin embargo, el 30% es retenido por Apple (0,89*0,30= 0,27€), por lo que los ingresos por cada app vendida quedaría en: 0,89-0,27€= 0,62€.

[1] Referencia sacada de: http://www.louesfera.com/2013/07/19/tribuna-libre-francisco-jose-gallardo-precio-apps/

3.6.3. Estimación

Para estimar cuántos usuarios tienen que descargarse la aplicación y empezar a obtener ingresos, se tiene en cuenta que la aplicación valdrá 0,89€, pero el ingreso será de 0,62€. Por lo tanto:

8.576,4 (**coste total**) / 0,62 (**ingreso por usuario**) = 13.833 usuarios tendrían que descargar la aplicación para empezar a obtener beneficios.

3.6.4. Segundo año

Lo mencionado anteriormente es válido para el primer año en el que la aplicación ha estado disponible para la descarga. Para el segundo año, el gasto es de sólo 80€ a causa de la licencia de desarrollador de Apple.

La cantidad de usuarios que tendrían que descargar la aplicación sería de:
80 (**coste licencia**) / 0,62 (**ingreso por usuario**) = 129 usuarios.

Lo que quiere decir que en dos años, si la aplicación es descargada por:
13.833 (**usuarios 1ºaño**) + 129 (**usuarios 2ºaño**) = 13.962 usuarios, se obtendría beneficio.

3.6.5. Conclusión

Como se puede observar, el riesgo que supone este proyecto es medio, ya que se necesita un número alto de descargas de la aplicación. En dos años debería de haber sido descargada por 13.962 usuarios, por lo cual la aplicación necesitaría tener éxito considerando que no todas las aplicaciones llegan a ese nivel de descarga cobrando con anterioridad.

La solución a este problema sería abordar otro tipo de ingresos como añadir en el futuro una nueva funcionalidad para permitir a establecimientos y comercios publicitarse en la aplicación tras haber pagado una cuota. Así, junto a esta idea, introducir la publicidad descrita anteriormente y poner la aplicación gratuita.

En este proyecto no se calculará la viabilidad de obtener beneficios de ese modo, ya que no se ha implementado dicha posibilidad.

4. Antecedentes

En este apartado, se evaluarán los estudios y pruebas realizados en otras aplicaciones existentes que han servido para el planteamiento de este proyecto.

Para este caso, los estudios han sido realizados sobre aplicaciones del sector turístico o de viajes. Así mismo, se dividió en varias categorías para hacer estudios independientes y a su vez tratar de relacionarlos.

En primer lugar se empezó con la prueba y revisión de aplicaciones de turismo general, es decir, aplicaciones que ofrecieran la posibilidad de ver *online* la información o de descargar guías de varios países y ciudades del mundo. Ese fue el punto de partida. Se encontró que todas ellas ofrecían guías completas, algunas de pago, provocando la descarga de toda la información de la ciudad. Observando la guía, rápidamente se podía llegar a una conclusión: había información que no tenía por qué interesar al usuario. Por lo tanto, fue una cuestión importante para tener en cuenta. La división de la información a la hora de descargarla era importante.

Concretamente, se investigó sobre las siguientes aplicaciones: Guías mTrip, Let's Go, Tripadvisor City Guides, MobilyTrip.

-**Guías mTrip**: Esta aplicación, a pesar de ser muy completa, te permite leer gratuitamente sólo una parte de las guías. Si se quiere la guía completa, se tiene que pagar 4,49€. Esto supone descargar información que quizá no interese y pagar la guía completa de toda la ciudad. Es algo que con este proyecto no se perseguía, ya que lo que se pretende es que el usuario descargue la información libre y gratuitamente. No obstante, se realizó un estudio de lo que ofrecía la versión gratuita.

Una buena funcionalidad es la lectura *offline* de la información descargada. Puedes descargarte la información en cualquier momento y leerlo cuando se está realizando una visita turística o en cualquier otra parte. No obstante, el usuario no tiene por qué cargar con información de lugares que realmente no quiera o no vaya a visitar. Esta idea era buena, pero había que segmentar la información.

-**Let's Go**: Esta aplicación no es tan completa como la anterior. Sólo deja descargarte guías de pago, con la misma penalidad que la anterior: información que pudiera no ser útil. A su vez, no te ofrecía parte de la guía gratuitamente para que vieras, al menos, cómo podría estar estructurada la información y quizá no fuera lo completa que el usuario querría. Por lo tanto, se desestimó el estudio de gestionar la información en esta aplicación.

No obstante, ofrece ver fotos *online* y algo que ninguna de las otras aplicaciones que se ha estudiado ofrecía: los vídeos. Se puede ver vídeos que los usuarios han ido subiendo, mostrando diferentes lugares y enseñando el lugar donde se encontraban. Por lo tanto, gracias a esta app, se decidió que los usuarios fueran una parte importante en la contribución de la aplicación. Se

optó por que cada usuario pudiera hacer sus propias guías y compartir información que tuviera de sus viajes. Una buena manera de hacer crecer la aplicación y dar la oportunidad a los usuarios de ofrecer su opinión y su conocimiento a cambio del de los demás, haciendo que de esta manera todos salgan ganando.

-**TripAdvisor City Guides**: Esta aplicación permite la descarga de la información de una ciudad entera. El punto más a favor que se recogió para este proyecto, era la manera en que gestiona los itinerarios o rutas para visitar la ciudad. Debido a su complejidad y el desconocimiento previo del lenguaje con el que se iba a implementar la aplicación, no se optó por hacerlo de esa manera y hacer las rutas más sencillas, dividiéndolas en días y mostrando una descripción de qué visitar. No obstante, en futuras actualizaciones, si las hubiera, se espera perseguir la idea que ofrece esta aplicación, ya que es muy completa y beneficiosa para el usuario.

A su vez, esta aplicación ofrecía la inspección del metro de la ciudad que el usuario tuviera descargada y fue una buena idea para implementar en este proyecto, de manera sencilla, y con una foto del mapa de la ciudad.

-**MobilyTrip**: A pesar de que como en todas las demás apps, al descargar una guía, te descargabas toda la información junto con las fotos, se recogieron varías ideas en cuanto a la interfaz. En un principio, se iba a implementar igual que esta aplicación la manera que gestionaba la descripción de los lugares de la ciudad. Segmenta la descripción de los lugares de una ciudad en varios apartados, como historia, arquitectura, críticas, etc. para mostrar lo que el usuario quisiera en ese momento. Sin embargo, y dada la complejidad de hacerlo de esa manera, se optó por una sola descripción del lugar, el horario a mostrar, y el coste de entrada si lo tuviera.

Con bastante información recabada de este tipo de aplicaciones, se buscó cómo ofrecían la información las aplicaciones de ciudades concretas o regiones del mundo. Para ese caso, se estudió la aplicación de Turismo Navarra.

De esta aplicación, se idearon varias ampliaciones interesantes para hacer en un futuro como dónde comer, agenda de las fiestas y eventos destacados de la ciudad, el tiempo, la ubicación de las oficinas de turismo, etc. Debido al tamaño y tiempo que suponía llevar a cabo esta aplicación, se desestimaron estas opciones, resultando como posibles ampliaciones para implementarlas en un futuro si la aplicación se enviara a la tienda de aplicaciones de Apple.

Otro tipo de aplicaciones que se estudiaron fueron las aplicaciones de aerolíneas y vuelos, como la de Kayak o AirEuropa iOS, que no aportaron mucho a la idea de este proyecto. Algunas ideas recabadas fueron el cambio de divisas, la lista de equipaje, etc., que se contemplarían en un futuro de ser viables e interesantes para la aplicación final.

De esta manera, la parte que se quería ofrecer de las guías se dio por finalizada. En resumen, se perseguía una aplicación gratuita, en la que los usuarios descargaran la información que quisieran sobre la ciudad que fueran a

visitar, de manera segmentada y que lo usuarios pudieran aportar sus conocimientos y guías.

De esta manera, se planificó el alcance que podría tener un usuario en la aplicación y las funcionalidades que pudiera llevar a cabo. Para ello, se estudió cómo gestionaban los usuarios registrados algunas de las aplicaciones mencionadas anteriormente y otras más ligadas a las redes sociales como Twitter o Facebook. Pero la que más ha contribuido como inspiración ha sido la aplicación Snapguide.

La aplicación Snapguide permite crear, compartir y ver guías paso a paso sobre cocina, manualidades, juegos, moda, etc. Y aunque no sea de turismo, se asemeja bastante a lo que pretende la aplicación de este proyecto. Por lo tanto, una vez que el usuario queda registrado e inicia sesión, desde su perfil accedería a gestionar todo lo que le perteneciese. De esa manera, podría acceder rápidamente a sus creaciones y a las valoraciones que haga sobre otras guías. Snapguide gestiona las amistades de la misma manera que Twitter, así que se decidió que sería una manera fácil e intuitiva para hacerlo de esa manera. Y por último, se estudió la manera de hacer un servicio de chat entre usuarios, pero ante la dificultad y factores que se desconocen al hacer ese tipo de funcionalidades o aplicaciones, se decidió hacer un servicio de mensajería de mensajes independientes como si fueran *emails*, para que de esta manera cada usuario pudiera contactar con otros usuarios.

En definitiva, partiendo de la idea principal de que la información sea descargada por partes o completamente, dependiendo del interés del usuario, y que contribuyan entre todos ellos, se ha conseguido diseñar la aplicación. A su vez, cada aplicación que se ha estudiado y probado para hacer este proyecto, ha ayudado en mayor o menor medida a encaminar la creación de esta aplicación.

5. Captura de Requisitos

La captura de requisitos es el proceso encargado de la identificación, asignación, verificación y modificación de los requisitos que tendrá este proyecto. Por lo tanto, a continuación se explicarán los casos de uso y el modelo de dominio de la aplicación con el fin de identificar sus funcionalidades.

5.1. Casos de uso

Los casos de uso representan la forma en cómo un actor (el cliente) opera con la aplicación. De la misma manera, se puede identificar una jerarquía de actores que quedaría de la siguiente manera:

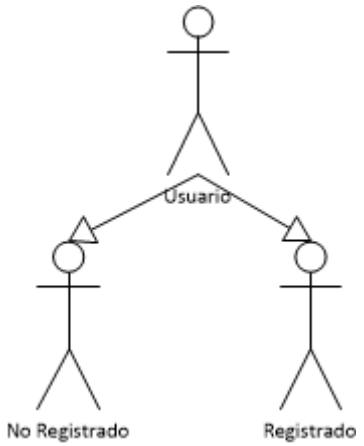


Ilustración 13.- Jerarquía de actores

El actor “Usuario” es un usuario que se ha descargado la aplicación pero ni se ha registrado, ni ha iniciado sesión. Podrá consultar información y descargarla, pero no subirla ni iniciar sesión.

El usuario “No Registrado” será aquel usuario que no se ha registrado aún. Por lo tanto, este actor sólo podrá registrarse y además, podrá hacer todo lo que hace el actor “Usuario”.

Como último actor, se encuentra el actor “Registrado” que podrá hacer todo lo que hace el actor “Usuario” y además iniciar sesión, compartir y crear guías, puntuar y relacionarse con otros usuarios. Sin embargo, no podrá registrarse en el sistema.

Así, la jerarquía de actores queda diferenciada e identificada para lo que puede hacer un actor y otro.

A continuación, se muestra detalladamente a los actores y las funcionalidades que puede ejecutar.

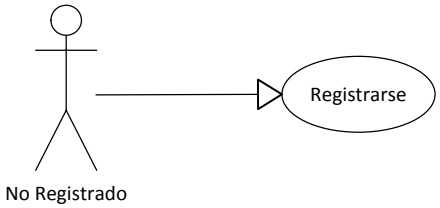


Ilustración 14.- Casos de uso. No registrado

Como se ha explicado, el actor “No Registrado” sólo podrá registrarse en el sistema. Así pues, el caso de uso Registrarse, permite registrar al usuario en el sistema.

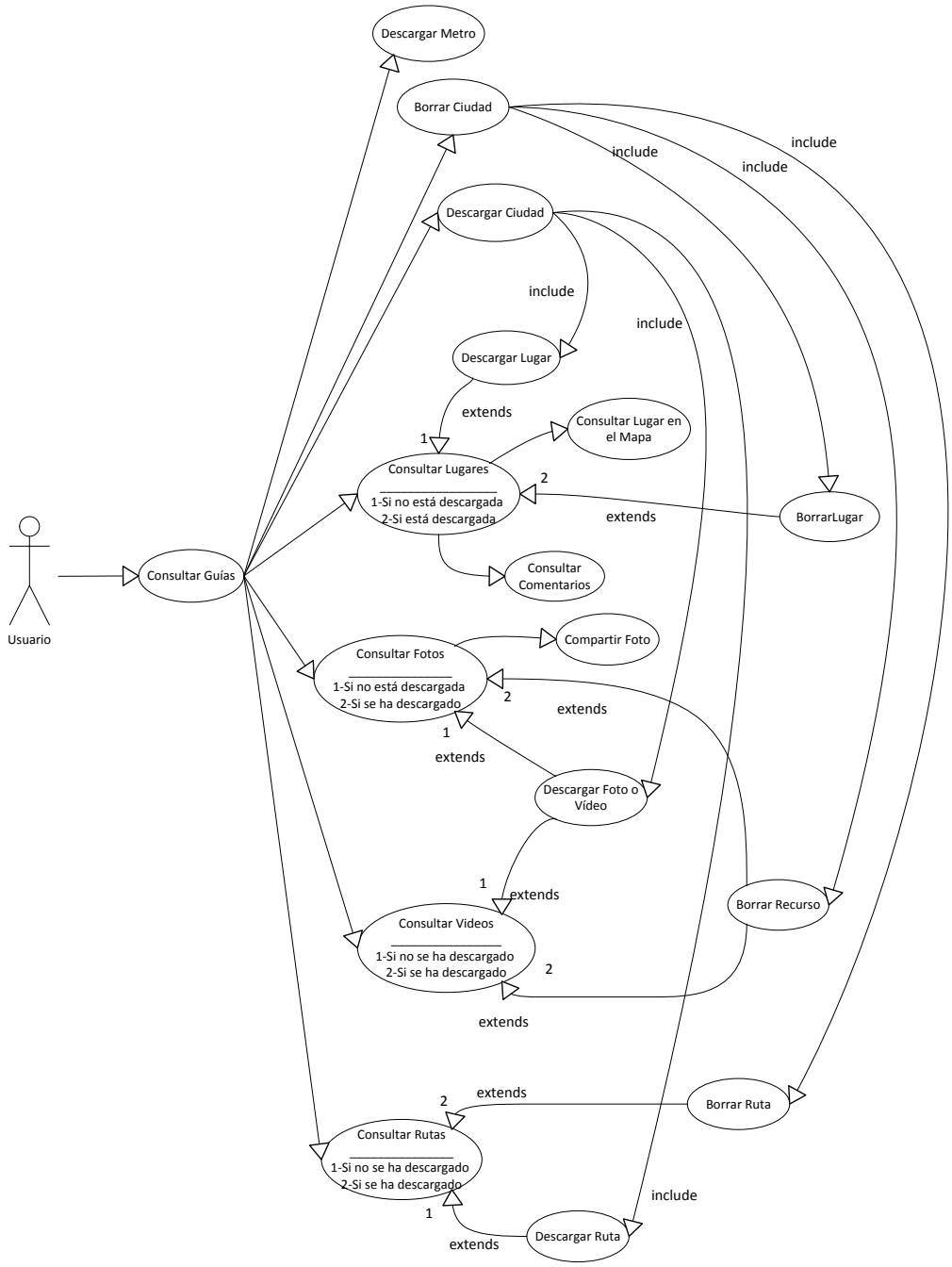


Ilustración 15.- Casos de uso. Usuario

El actor “Usuario” podrá hacer uso de una gran cantidad de acciones en la aplicación. Se procede a continuación a explicar brevemente los casos de uso individualmente:

-**Consultar guías**: Muestra todas las ciudades descargadas y las disponibles para descargar. Hay que acceder a esta consulta para ver las ciudades y a partir de ahí acceder a más información. Al pulsar sobre una ciudad, se muestra la ficha de la ciudad con la información descargada y la disponible.

-**Descarga Metro**: Esta funcionalidad permite descargar la foto del metro de una ciudad y almacenarla en el dispositivo.

-**Borrar ciudad**: Como su nombre indica, borrar la información de la ciudad que se ha descargado. También se borra todo lo relacionado con ella, como lugares, fotos, vídeos, rutas y la foto del metro.

-**Descarga ciudad**: Permite descargar todo lo referente con la ciudad a la vez, es decir, descargar todos los lugares, fotos, vídeos y rutas que pudiera tener.

-**Consultar lugares**: Se muestran todos los lugares que pertenecen a una ciudad, ordenados por diferentes criterios. Además, se puede acceder a la ficha del lugar.

-**Descarga lugar**: Se descarga todo lo relacionado con los lugares de una ciudad, siempre y cuando no se haya descargado previamente.

-**Consultar lugar en mapa**: Mediante el Geoposicionamiento, se mostrará en el mapa el lugar que se haya elegido, así como una ruta entre la posición del usuario y el lugar.

-**Borrar lugar**: Borra todo lo referente al lugar descargado. Para ello, el actor "Usuario" habrá tenido que descargar previamente el lugar.

-**Consultar comentarios**: Consulta los comentarios que haya sobre los lugares.

-**Consultar fotos**: Consulta todas las fotos descargadas, así como las que están a disposición de ser descargadas. Además, al elegir una foto, se muestra individualmente.

-**Descargar foto o vídeo**: Descarga la foto o vídeo de la ciudad o del lugar al que pertenezca y almacena el archivo en el dispositivo, si no se ha descargado previamente.

-**Borrar Recurso**: Borra la foto o vídeo del dispositivo y su información, si previamente se ha descargado.

-**Compartir fotos**: Compartir fotos permite compartir con Facebook las fotos que el usuario quiera y que se hayan subido al servidor.

-**Consultar vídeos**: Muestra todos los vídeos descargados y disponibles en el servidor. Además, al elegir un vídeo, lo reproducirá.

-**Consultar rutas**: Muestra todas las fotos descargadas y disponibles para su descarga, dividida por días. Al pulsar sobre una, se accede a su información completa.

-**Descargar ruta**: Descarga toda la información de la ruta, si no se ha descargado antes.

-**Borrar ruta**: Se borra la información de la ruta, si se ha descargado previamente.

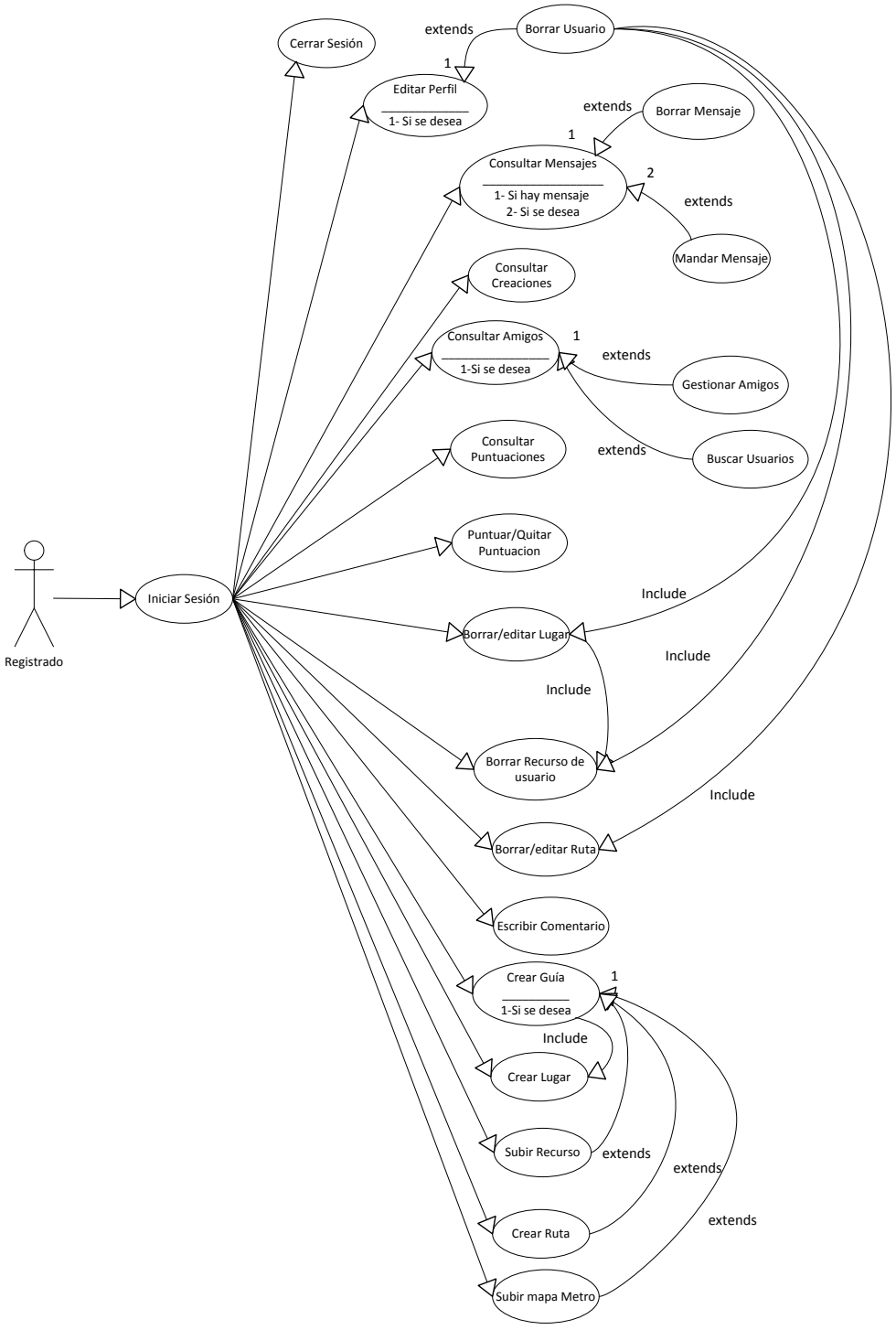


Ilustración 16.- Casos de uso. Registrado

El actor “Registrado” también podrá hacer uso de una gran cantidad de acciones en la aplicación. A continuación, se procede a explicar brevemente los casos de uso:

-**Iniciar sesión**: Para poder usar las demás funcionalidades, el usuario tendrá que iniciar sesión previamente en el sistema. Así, introduciendo su nick y contraseña, se comprueba y se accede si el resultado coincide.

-**Cerrar sesión**: Cierra la sesión abierta por el usuario.

-**Editar perfil**: Se actualiza la información que el usuario haya cambiado.

-**Borrar usuario**: Si lo desea, el usuario puede borrar su cuenta. Así mismo, se borrará todo lo que el usuario haya compartido. Sin embargo, si alguno de sus lugares está siendo compartido por otro usuario con fotos y/o vídeos, no se borrará y permanecerá en el sistema, a pesar de que la cuenta haya sido borrada.

-**Consultar mensajes**: Se mostrarán los mensajes recibidos y enviados por el usuario. Al elegir un mensaje, éste se mostrará en detalle.

-**Borrar mensajes**: Si el usuario tiene mensajes en recibidos, podrá borrar los mensajes.

-**Mandar mensaje**: Si lo desea, podrá mandar mensajes a otros usuarios o contestar a los recibidos.

-**Consultar creaciones**: Mostrará en pantalla todas las guías que haya creado el usuario y podrá acceder a su contenido.

-**Consultar amigos**: Muestra la lista de amigos a los que está siguiendo, los le siguen y las peticiones que tenga si tiene el perfil bloqueado. Si pulsa sobre un usuario, se accederá a su perfil.

-**Gestionar amigos**: Si lo desea, puede seguir o dejar de seguir a otros usuarios o aceptar peticiones de amistad.

-**Buscar usuarios**: Si el usuario desea buscar un usuario en concreto, introducirá su nick y si existe, se mostrará en una tabla.

-**Consultar puntuaciones**: Se muestra todas las votaciones que ha hecho en la aplicación, agrupadas por lugares, rutas, fotos y vídeos.

-**Puntuar/quitar puntuación**: El usuario puntúa o quita sus puntuaciones de los lugares, rutas, fotos y vídeos.

-**Borrar/editar lugar**: Si el usuario ha creado la información de un lugar, éste puede modificarla o borrarla. Si la borra, las fotos y los vídeos que haya sobre esos lugares también se borrarán. Pero si otro usuario está compartiendo fotos o vídeos sobre ese lugar, no podrá borrarlo.

-**Borrar recurso de usuario**: Se borrará del servidor la foto o el vídeo del usuario que haya decidido borrar.

-**Borrar/editar ruta**: Se borrará toda la información de la ruta que pertenezca al usuario.

-**Escribir comentario**: Esta funcionalidad permitirá escribir comentarios sobre los lugares que hayan creado los usuarios o haya en el sistema.

-**Crear guía**: Crea una guía completa en la que obligatoriamente se tendrá que crear un lugar. El usuario creará una ciudad, un lugar (como mínimo) y después, si lo desea, podrá crear rutas, subir fotos y vídeos y subir la imagen del metro de la ciudad.

-**Crear lugar**: Crea un lugar con toda la información que el usuario introduzca. Deberá rellenar la información con una descripción, y el horario y el precio si los hubiera.

-**Subir recurso**: Sube el archivo de foto o vídeo al servidor que el usuario haya elegido desde el dispositivo móvil.

-**Crear ruta**: Crea la guía de una ruta con los días que se tarda en recorrerla y una descripción.

-**Subir mapa metro**: El usuario sube el archivo de foto del metro de la ciudad que haya elegido desde su dispositivo móvil.

5.2. Modelo de Dominio

En el modelo de dominio se mostrarán las distintas entidades con sus atributos y las relaciones entre ellas. Así, en la aplicación se va a guardar información sobre las ciudades, lugares, etc. además de las relaciones que pueda haber entre ellos. A continuación se muestra el siguiente esquema, que representa el modelo de dominio de la aplicación.

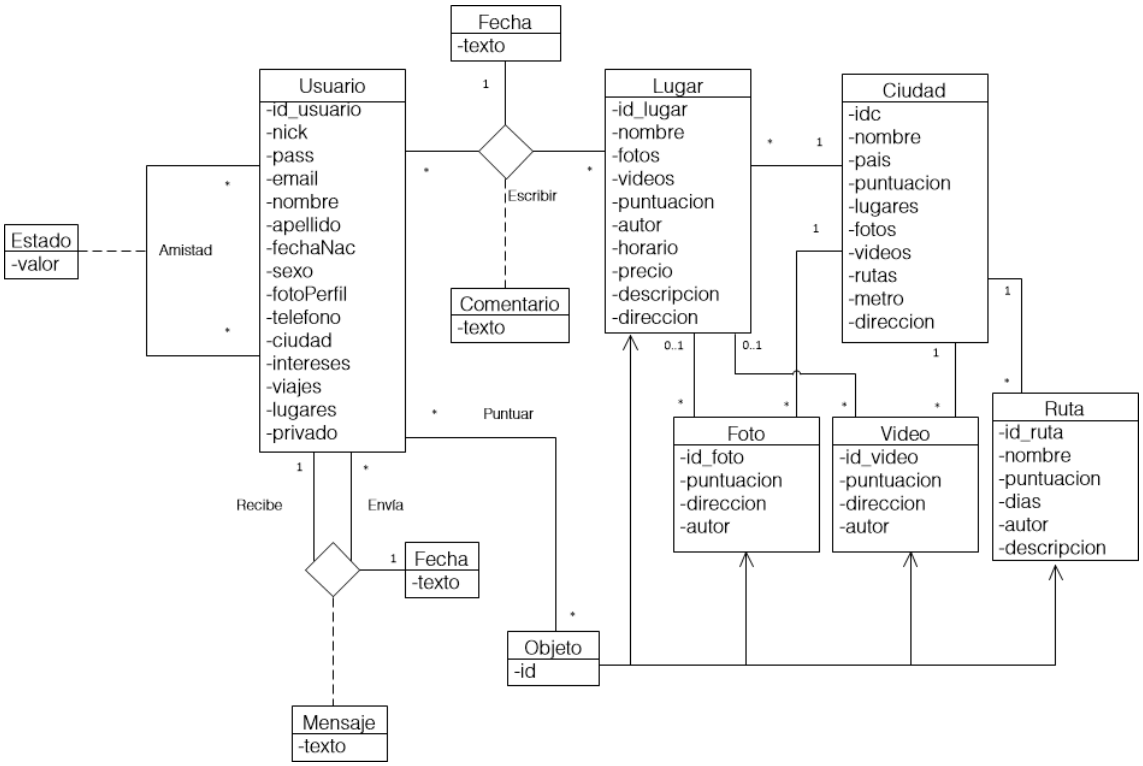


Ilustración 17.- Modelo de dominio

Como se observa, se guarda la información de la *ciudad* con el nombre, el país, y la cantidad de lugares, fotos, vídeos y rutas que le pertenecen. En la entidad *lugar* se guarda su descripción, horario y precio. Un lugar puede pertenecer a una sola *ciudad*, pero una ciudad puede tener muchos *lugares*.

En *foto* y en *vídeo* se guarda la puntuación, la dirección de la foto y el autor. Una *foto* o un *vídeo*, puede pertenecer a una *ciudad* y una *ciudad* puede tener muchas *fotos* o *vídeos*. A su vez, una *foto* o un *vídeo* pueden pertenecer a un *lugar* o no pertenecer. Si no pertenecen a ningún *lugar*, la *foto* o el *vídeo* pertenecerán a la *ciudad*. Además, un *lugar* puede tener muchas *fotos* y *vídeos*.

En la tabla *ruta,* se guarda el nombre, la puntuación, etc. y puede pertenecer a una *ciudad*, pero una *ciudad* puede tener muchas *rutas*.

La entidad *Objeto* se utiliza como padre de las entidades *Lugar*, *Foto*, *Vídeo* y *Ruta* para facilitar las puntuaciones de los usuarios.

A la hora de puntuar un *objeto* (esto es, lugares, fotos, vídeos o rutas), un usuario puede puntuar muchos *objetos* y estos *objetos* pueden ser puntuados por muchos *usuarios*.

Un *usuario* puede escribir varios comentarios en un *lugar* en una *fecha* concreta. Teniendo un lugar la posibilidad de recibir un comentario por muchos *usuarios* en una misma *fecha.*

Con la misma idea, un *usuario* puede escribir un mensaje a otro *usuario* en una *fecha* concreta, ya que en la *fecha* se guarda la hora también. Así, un usuario puede recibir varios mensajes en una misma fecha.

Además, un *usuario* puede seguir y crear una amistad con otros *usuarios*. Para ello, se almacena el estado de la amistad, ya que el *usuario* puede tener el perfil en privado. De la misma manera, un usuario puede recibir peticiones de amistad de muchos usuarios.

6. Análisis y diseño

En este apartado se explicará cómo se ha planteado este proyecto. Se mostrará el diseño de las bases de datos y las clases de la aplicación con el fin de ver como se relacionan entre ellas.

En primer lugar, se explicarán las bases de datos que estarán presentes en la aplicación. Habrá dos bases de datos diferentes. Una alojada en el servidor y otra en la aplicación de manera local. De esta manera, toda la información en la base de datos del servidor tendrá la información para ser consultada y descargada y la base de datos local guardará la información que un usuario se haya descargado.

6.1. Base de Datos Servidor

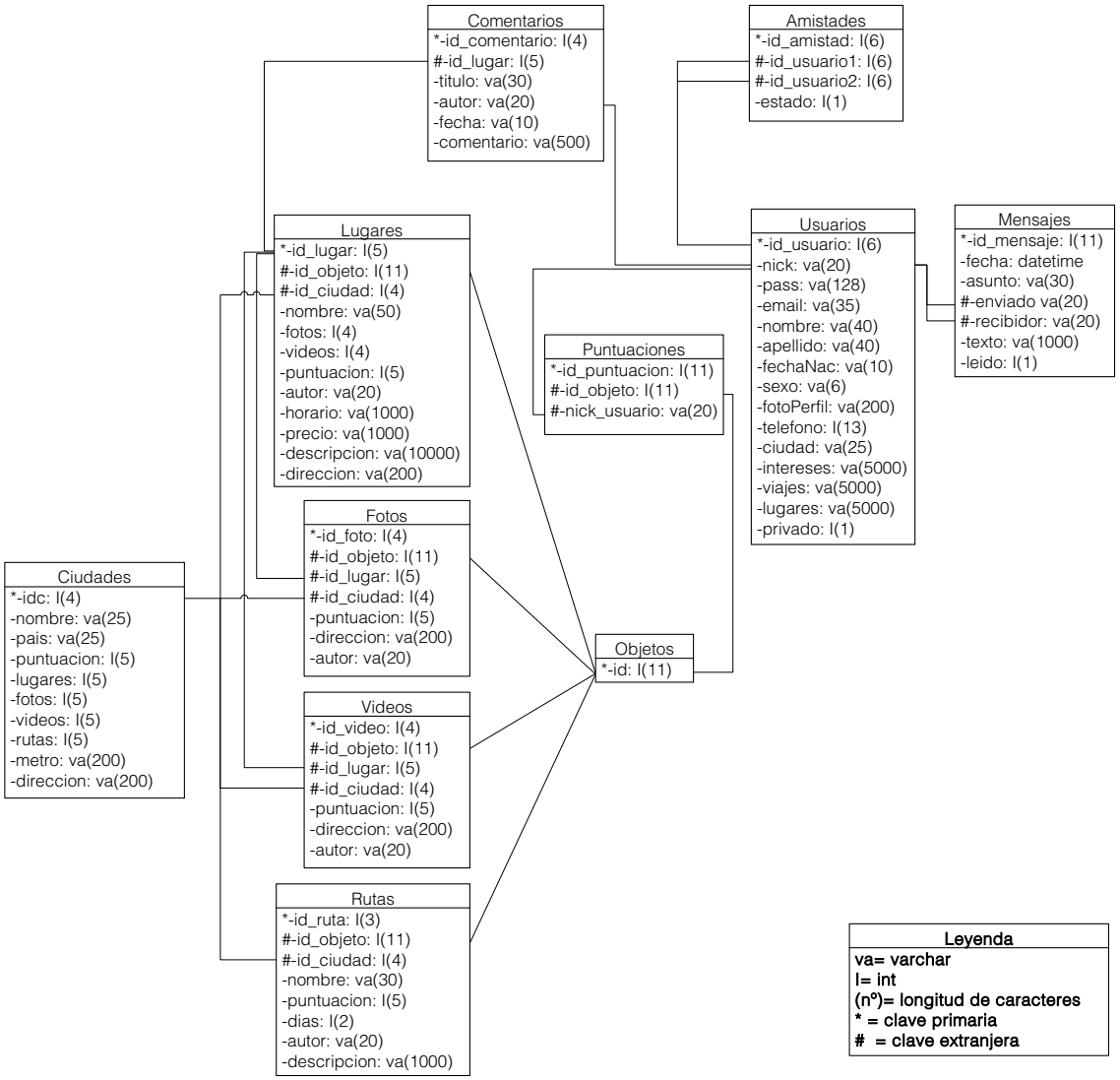


Ilustración 18.- Base de datos Servidor

En la base de datos externa o del servidor se guardará toda la información disponible para descargar y consultar. De esta manera se observa en la imagen de arriba cómo se relacionan las tablas entre ellas.

Las *Ciudades* están relacionadas por el *id_ciudad* con las tablas *Lugares*, *Fotos*, *Videos* y *Rutas*. Así, los atributos *id_ciudad* de todas esas tablas con las que se relaciona serán claves extranjeras y únicas. Los mismo ocurre con el *id* de la tabla *Objetos.* Las tablas *Lugares*, *Fotos*, *Videos* y *Rutas* tendrán los atributos id_objeto como claves extranjeras y únicas.

En concreto, la tabla *Fotos* y *Videos* pueden tener relación con los *Lugares*, así se almacenará el id del *Lugar* en el atributo *id_lugar*. Este *id_lugar* será clave extranjera y además puede ser *NULL* si la *Foto* o el *Vídeo* no tienen relación con algún *Lugar*.

Como en la aplicación se podrá escribir comentarios de los lugares, se guarda en la tabla *Comentarios* todo lo relacionado con ese comentario, el *nick* del usuario que ha escrito ese comentario y el *id* del lugar que hará referencia al lugar comentado. Estos dos últimos atributos serán claves extranjeras.

La tabla *Objeto* es utilizada para gestionar y almacenar las puntuaciones más fácilmente. Así, en la tabla *Puntuaciones* se guardará el *id* del *Objeto* y el nick del usuario que ha puntuado ese objeto. Ambos atributos serán claves extranjeras.

Los usuarios, a su vez, se pueden relacionar entre ellos. El atributo nick en la tabla *Usuarios*, será único, ya que no podrá repetirse. Así, en la tabla *Amistades* se guardará las amistades y el *estado* de ellas. Para ello se utiliza los atributos *id_usuario1* e *id_usuario2* para guardar qué usuario sigue al otro. Ambos atributos serán claves extranjeras.

Por otro lado, los *Usuarios* también pueden mandarse mensajes entre ellos. Así, en la tabla *Mensajes* se guarda la información y quién ha *enviado* y ha sido el *recibidor* del mensaje. En estos dos atributos se guardarán los *nicks* de los usuarios, y serán claves extranjeras.

6.2. Base de Datos Local

La base de datos local sigue la idea que se persigue en la BD del servidor. El esquema es el mismo, salvo que no se guarda nada relacionado con la gestión de los usuarios. A continuación se muestra su diagrama relacional:

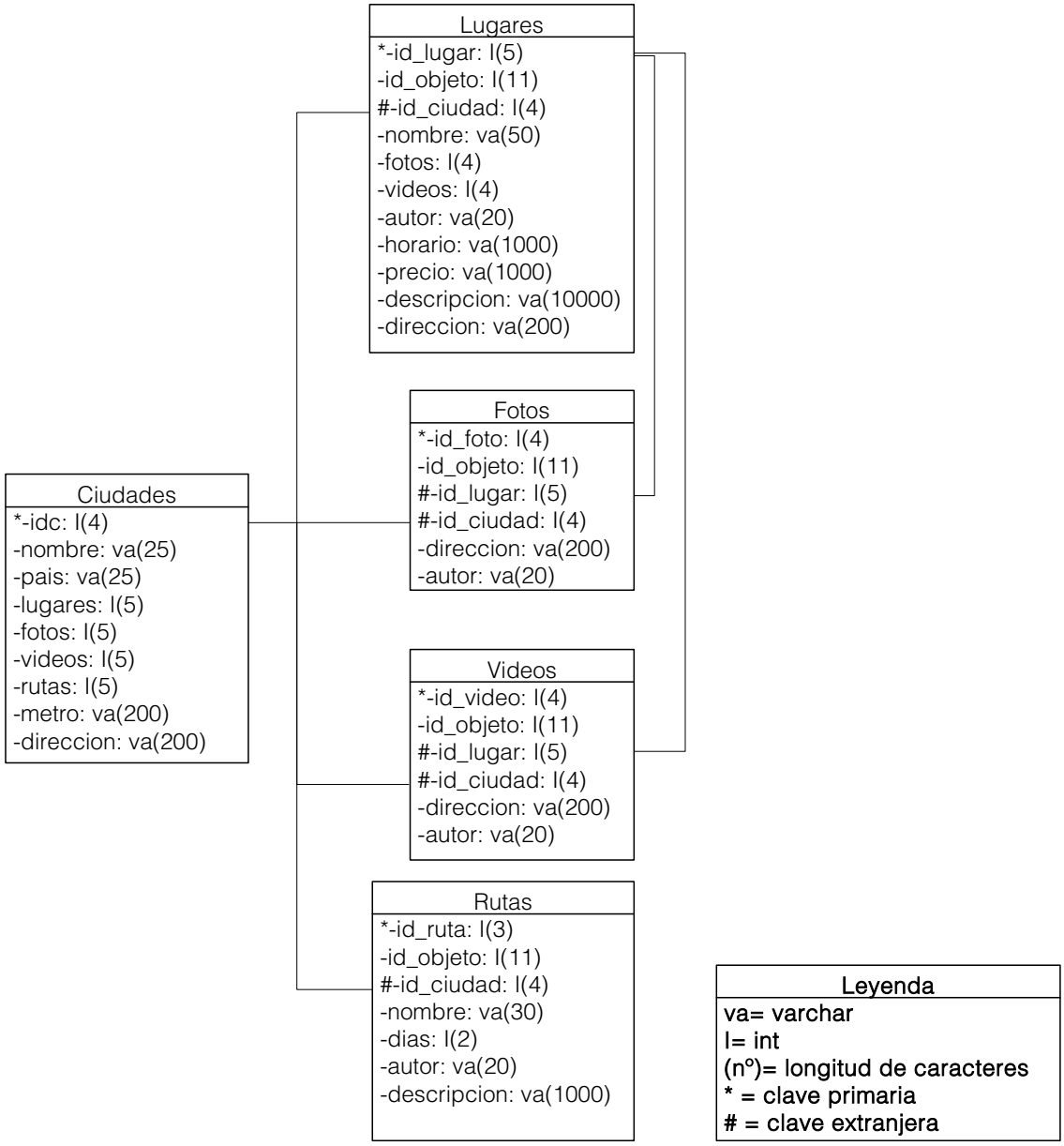


Ilustración 19.- Base de datos local

Se observa que la tabla *Ciudades* se relaciona con las demás tablas de la misma manera que en la base de datos del servidor. Así mismo las tablas *Fotos* y *Videos* se relacionan con la tabla *Lugares* por el *id_lugar*. La única diferencia es que en la base de datos local no se guarda la *puntuación* en ninguna de las tablas, ya que es un atributo que estará variando constantemente y no tiene sentido almacenarla.

6.3. Diagrama de clases

En las siguientes imágenes se ilustrará el diagrama de clases de la aplicación, en el que se mostrarán los objetos utilizados y las relaciones entre ellos.

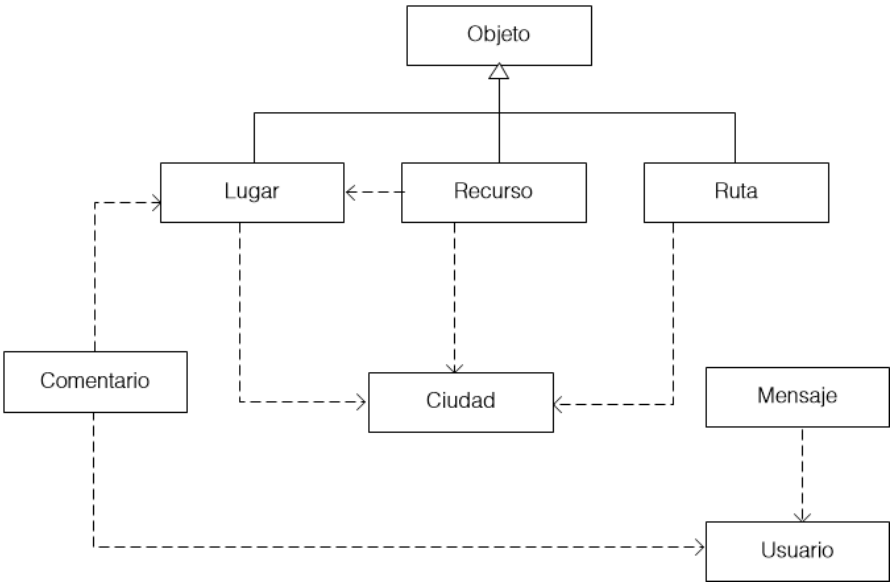


Ilustración 20.- Diagrama de clases. Objetos

Como se puede observar los objetos *Lugar, Recurso* y *Ruta* dependen de la clase *Objeto*, del cual heredarán sus atributos y métodos. La clase *Recurso* hace referencia a las fotos y los vídeos, aunándolos en una sola clase para simplificar y no repetir código, ya que las operaciones utilizadas son similares para ambos tipos de archivo.

Por ello, la clase *Recurso* se relaciona con la clase *Lugar* y *Ciudad*. Esta última, a su vez, tiene relación con las clases *Ruta* y *Lugar*. Debido a que se pueden hacer comentarios de los lugares de las ciudades, la clase Comentario tiene relación con la clase *Lugar*.

Los usuarios pueden mandarse mensajes entre ellos por lo que ambas clases estarán relacionadas.

A continuación, se ilustrará en una imagen todas las clases utilizadas en la aplicación y la dependencia entre ellas, empezando desde la pantalla principal.

Como se explica en el capítulo 2, se necesitará un ViewController por cada vista que haya en el proyecto. Así, todas las clases que se ven, serán del tipo ViewController y para diferenciarlas, se les ha puesto el sufijo VC a los nombres de la clase.

El diagrama se ha dividido en dos para su mayor legibilidad y además para diferenciar entre los dos grandes apartados de la aplicación. El apartado de descargas, donde se podrá gestionar todas las guías descargadas y disponibles y el apartado de la cuenta, donde se mostrará todo lo relacionado

con los usuarios. Las clases del modelo, se han resaltado con otro color para diferenciarlas del resto.

Apartado de las descargas:

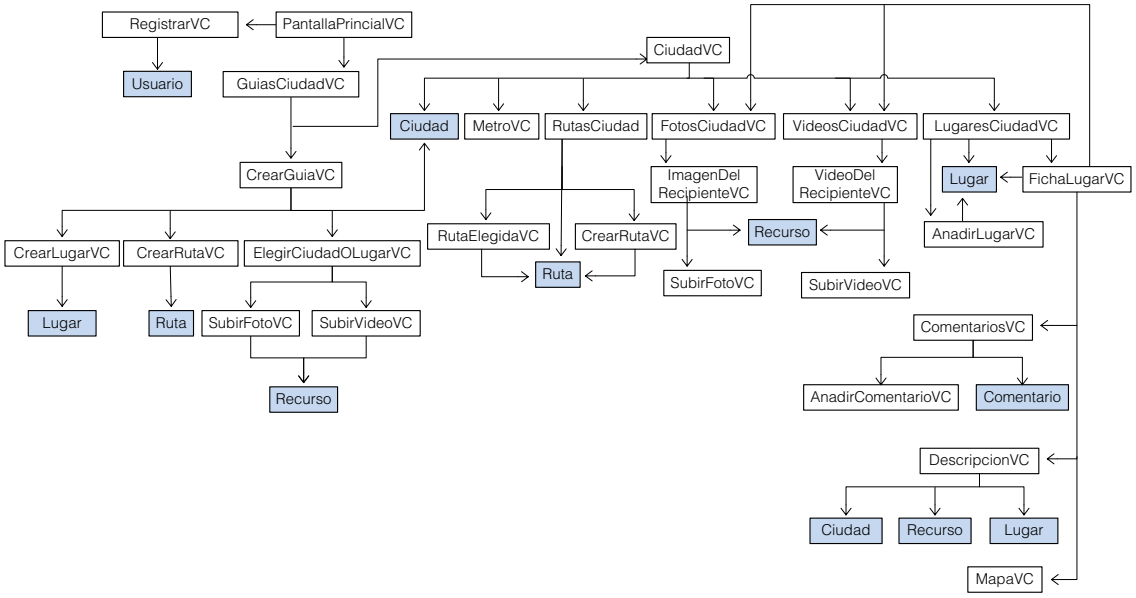


Ilustración 21.- Diagrama de clases. Descargas

Y el apartado de la cuenta:

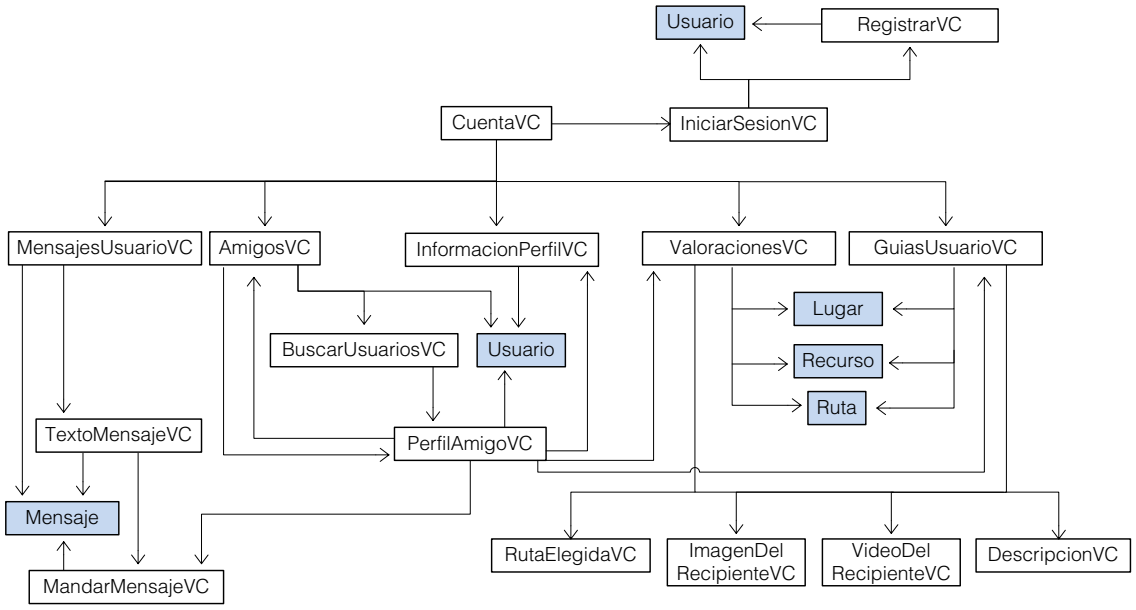


Ilustración 22.- Diagrama de clases. Cuenta

6.4. Clases

A continuación, se explicarán las clases más relevantes con sus atributos y métodos.

AppDelegate

AppDelegate
-window : UIWindows -nombreServidor : string -dataBasePath : string
+cargarBaseDeDatos() +applicationDidFinishLaunchingWithOptions(entrada : NSDictionary) : bool

Ilustración 23.- Clase AppDelegate

Esta clase es indispensable a la hora de crear cualquier aplicación. Será la encargada de *escuchar* los cambios que se vayan realizando en la aplicación, como por ejemplo, cuando al arrancar la aplicación comprueba que se ha cargado correctamente y así poder cargar la interfaz de usuario.

Así mismo, tiene otras posibilidades de las cuales esta aplicación se ha aprovechado. Una vez arrancada la aplicación, esta clase es la encargada de cargar la base de datos de la aplicación, junto con la información descargada. Cada vez que el usuario quiera descargar, actualizar o borrar algo que ha descargado, se accederá a la base de datos mediante esta clase.

Por otro lado, se ha utilizado como fichero de configuración. Simplemente se ha guardado en una variable el nombre del servidor que se vaya a utilizar en la aplicación, de manera que si se necesita mostrar información, actualizarse, borrar o cualquier operación que tenga que ver con la conexión a la base de datos del servidor o a las fotos y vídeos almacenados en el servidor, se utilizará esta clase para hacer referencia a la dirección del servidor.

Exceptuando esta primera clase y los objetos utilizados, casi todas las clases que siguen serán del tipo View Controller.

Para las celdas se han utilizado dos tipos diferentes de clases. La primera es UITableViewCell, que se ha usado para personalizar las celdas de las tablas utilizadas en la aplicación. Y la otra ha sido utilizada para mostrar las fotos y los vídeos, del tipo UICollectionViewCell que sirve para mostrar en un CollectionView elementos con forma cuadrada y ordenada.

De manera que todas las clases tienen su nombre y al final VC que hace referencia a ViewController, exceptuando las celdas que tendrán un nombre de la forma “Celda *Nombre*”.

Por lo tanto, a continuación se explicará en qué consisten las clases y las relaciones entre ellas.

Clases relacionadas con Ciudad

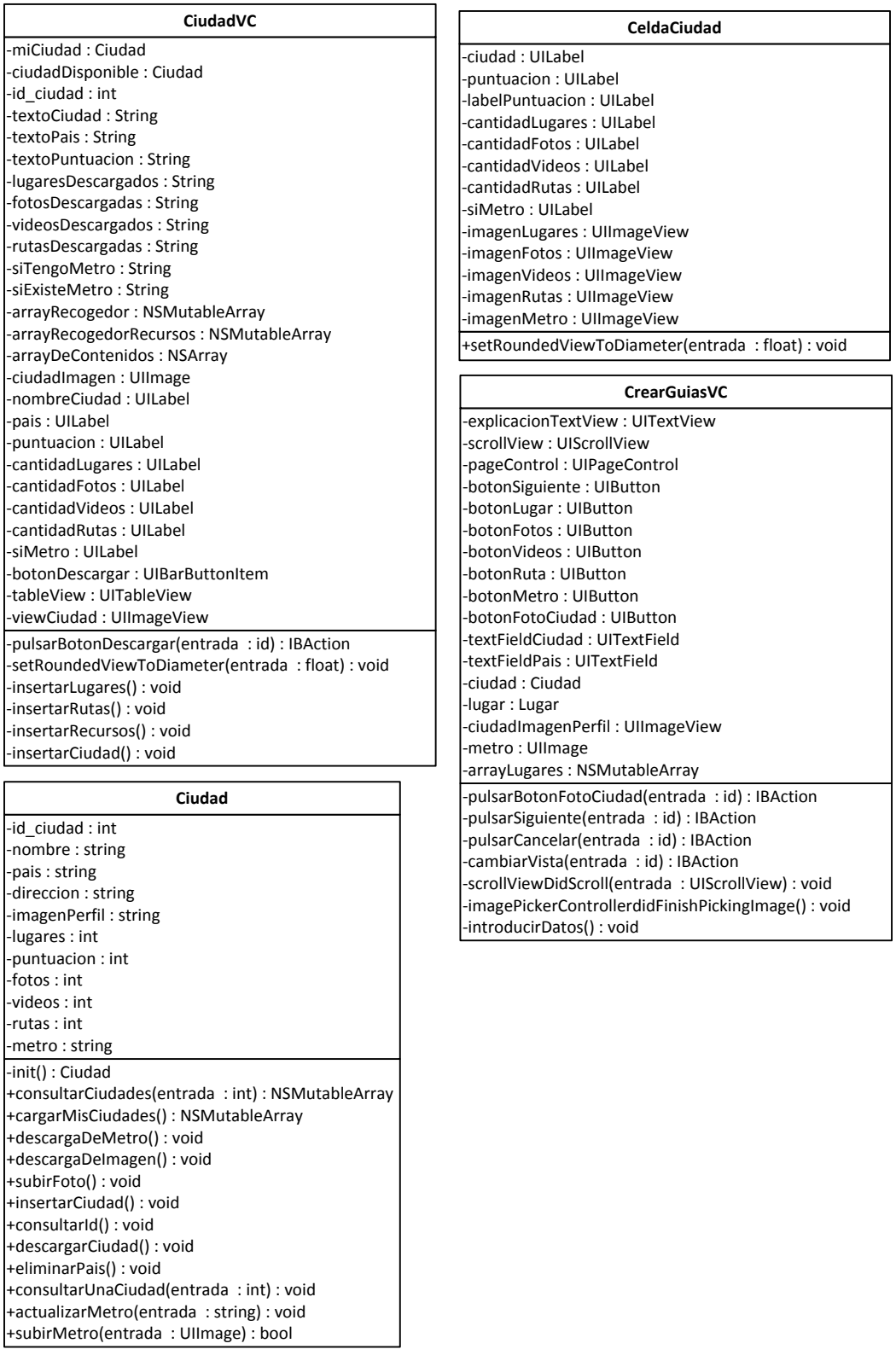


Ilustración 24.- Clases relacionadas con Ciudad 1

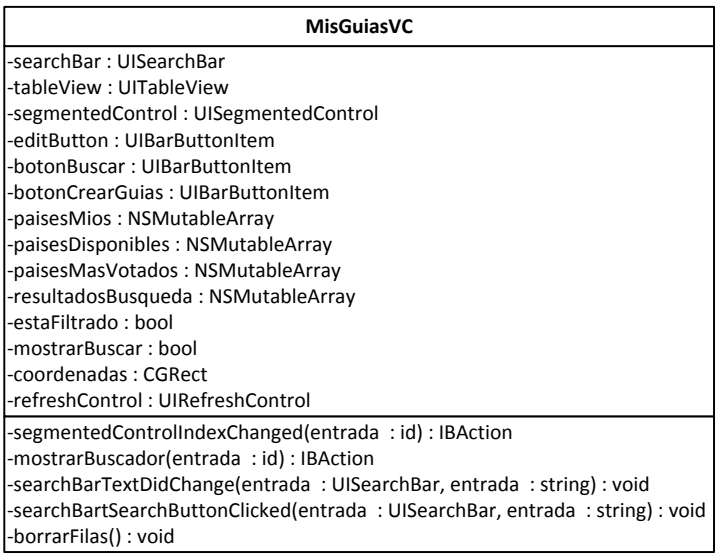


Ilustración 25.- Clases relacionadas con Ciudad 2

MisGuiasVC es la primera vista a la que se accede tras pulsar en el botón *Guías* de la pantalla principal. Aquí se llamara al AppDelegate para recibir la información de la base de datos. Tendrá tres pestañas que el usuario podrá ir seleccionando. El método segmentedControlIndexChanged será el encargado de actualizar la información de la tabla al ir cambiando de pestaña. En dos de ellas se accederá a la base de datos alojada en el servidor para recibir la información y mostrarla en la tabla. Cada ciudad aparecerá en una celda de la clase CeldaCiudad.

Al elegir una ciudad se accederá a la clase CiudadVC y se mostrará la información de la ciudad elegida.

Por otro lado, se puede acceder a crear una guía y la clase CrearGuiasVC mostrará la interfaz para que el usuario vaya creando la guía. Se usarán varias clases como SubirFotoVC, SubirVideoVC, CrearLugarVC, CrearRutaVC que se explicarán más adelante. En caso de las fotos y los vídeos, se accederá previamente a ElegirCiudadOLugarVC, que permitirá subir fotos y vídeos de la ciudad y los lugares creados hasta el momento.

Clases relacionadas con Lugar

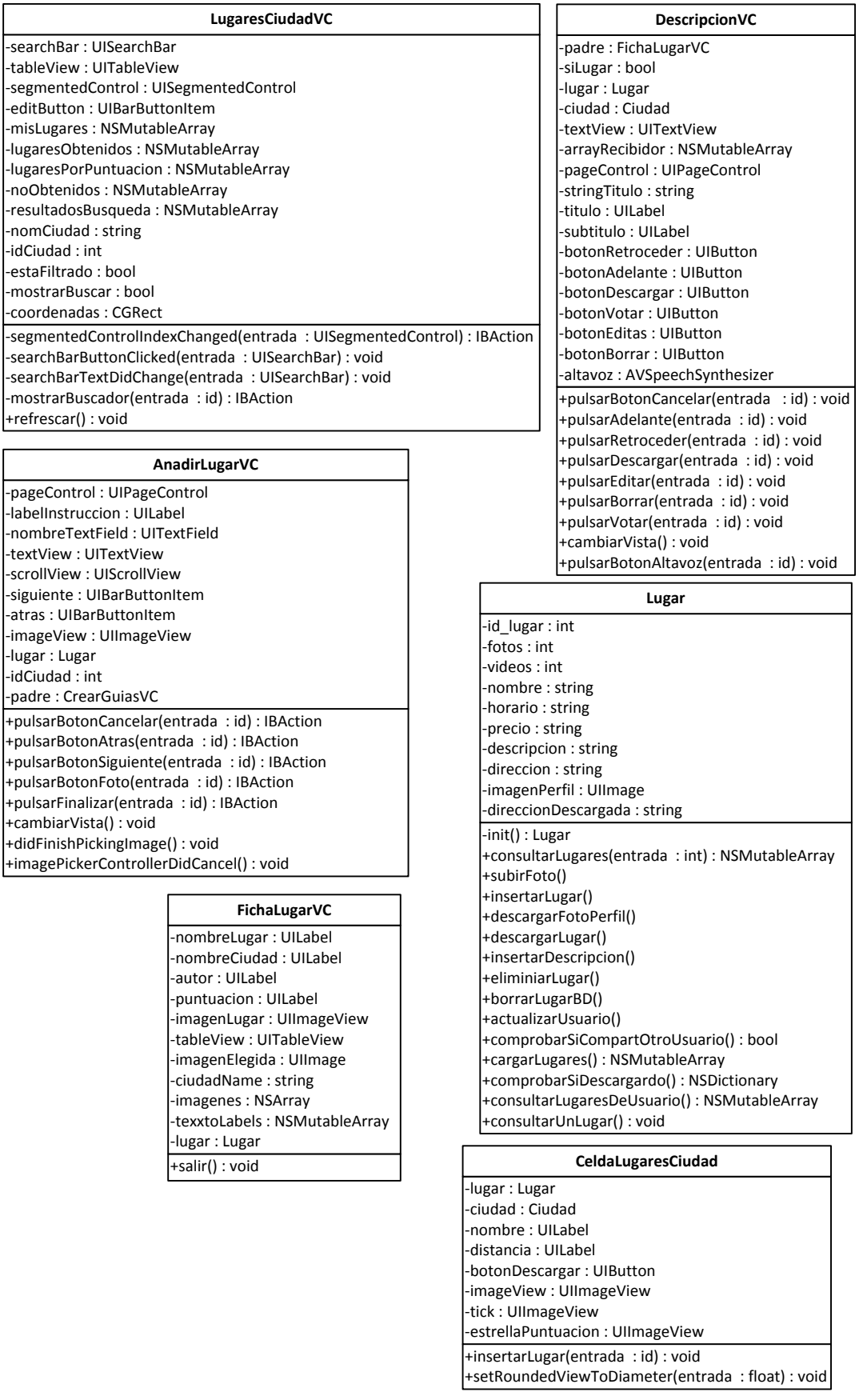


Ilustración 26.- Clases relacionadas con Lugar

Esta clase permite mostrar en una tabla todos los lugares descargados, y disponibles ordenados alfabéticamente o por puntuación, así como los no descargados aún. Cada lugar se mostrará en una celda personalizada de la clase CeldaLugaresCiudad. Si el usuario quiere crear un Lugar, pulsa el botón de añadir y se accederá a la clase AnadirLugarVC.

Cuando se pulsa en un Lugar, se accede a la ficha del lugar que corresponderá a la clase FichaLugarVC, donde se mostrará parte de su información y varias opciones a elegir, como por ejemplo la descripción, que mostrará toda la información relevante del Lugar elegido mediante la clase DescripcionVC.

Clases relacionadas con fotos (Recurso)

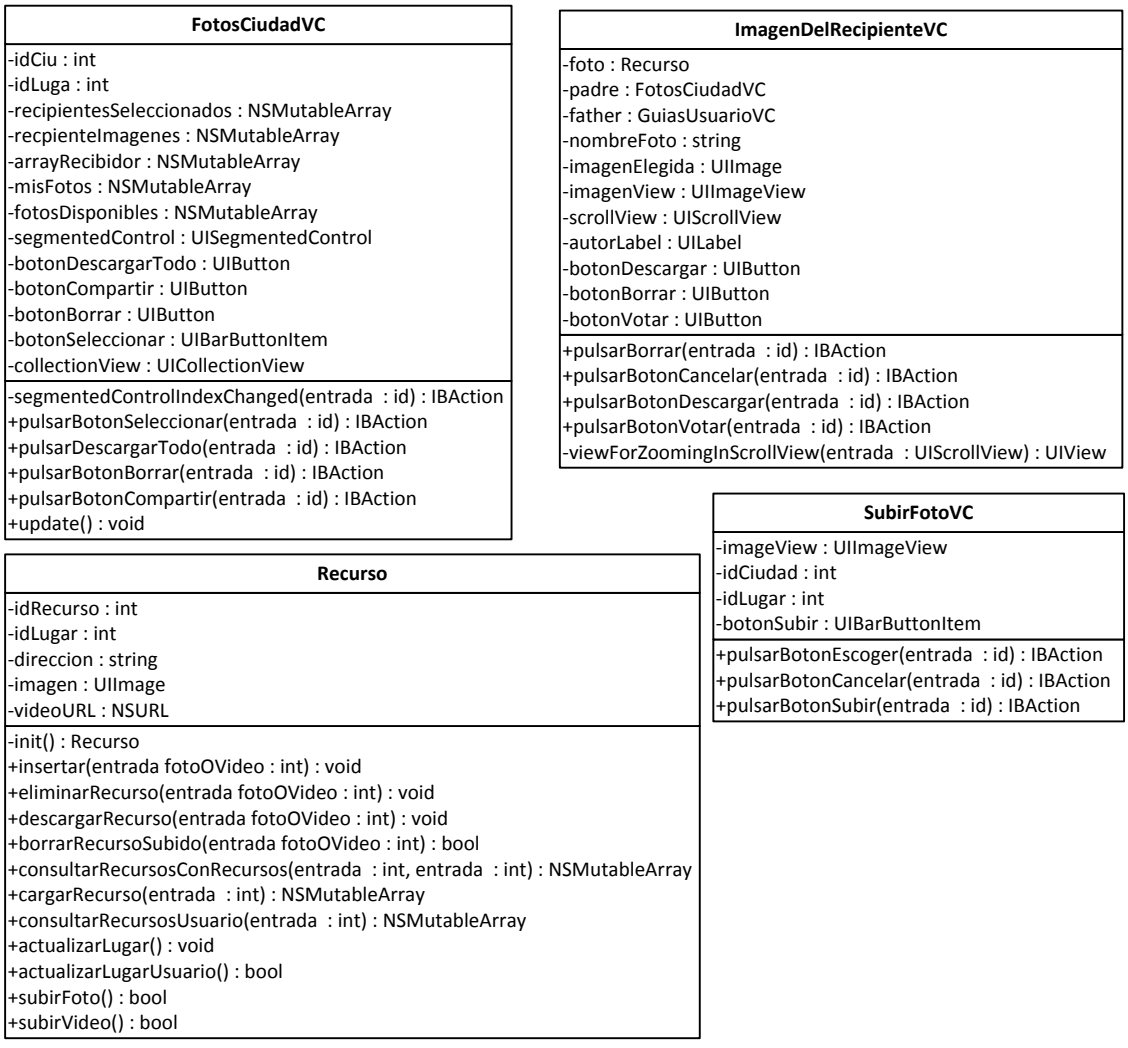


Ilustración 27.- Clases relacionadas con fotos

A esta clase se puede acceder desde la ficha de la ciudad o la ficha del lugar. Mostrará las fotos descargadas y las disponibles. Al subir una foto, se accede a

una nueva vista gestionada por la clase SubirFotoVC. Y si se quiere ver una foto en pantalla completa, la vista la gestiona ImagenDelRecipienteVC.

Clases relacionadas con Recurso (vídeos)

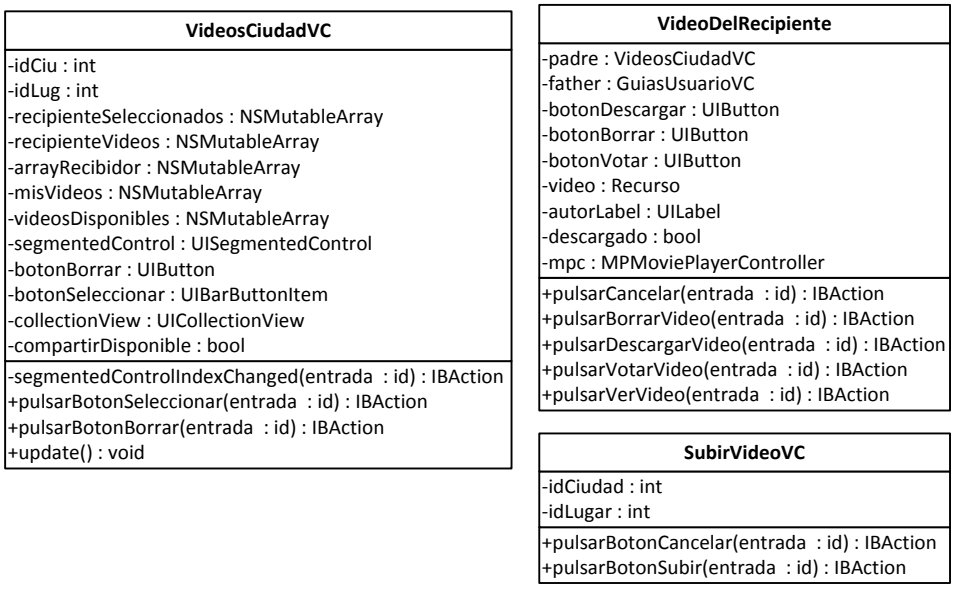


Ilustración 28.- Clases relacionas con vídeos

Al igual que en FotosCiudadVC, se accede desde la ciudad o desde un lugar concreto. Muestras los vídeos descargados y disponibles. Al pulsar en uno de ellos se accede a VideosDelRecipienteVC y si se quiere subir un vídeo, se mostrará la vista gestionada por la clase SubirVideoVC.

Clases relacionadas con Ruta

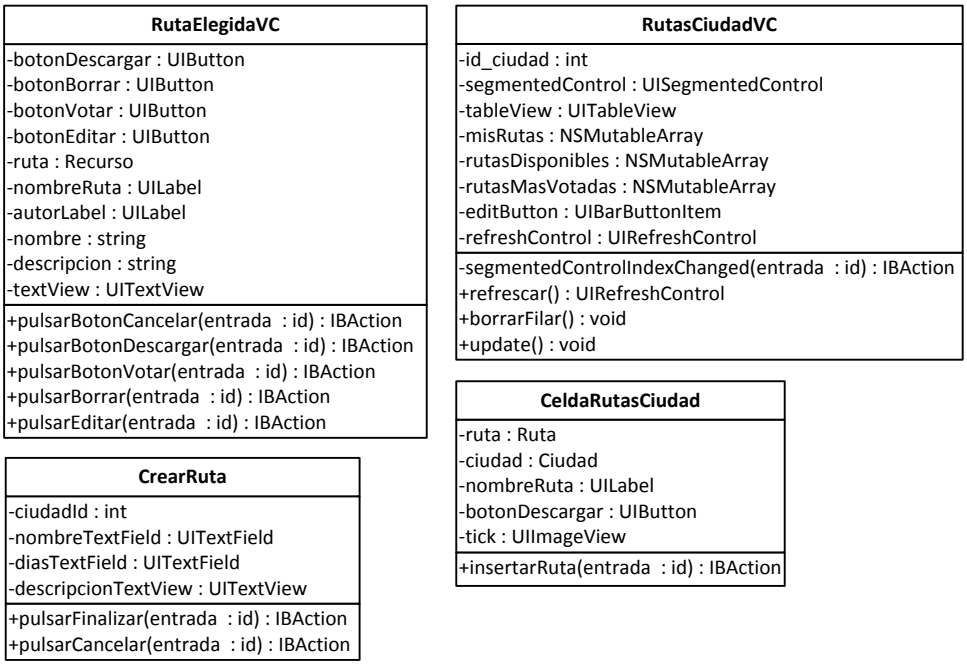


Ilustración 29.- Clases relacionadas con Ruta 1

Ruta
-idRuta : int -nombre : string -dias : int -descripcion : string
-init() : Ruta +insertarRuta() : void +descargarRuta() : void +eliminarRuta() : void +borrarRutaCreada() : void +consultarRutas(entrada : int) : NSMutableArray +cargarRutas() : NSMutableArray +consultarRutasUsuario() : NSMutableArray

Ilustración 30.- Clases relacionadas con Ruta 2

Muestras las rutas disponibles en una tabla, en la que cada celda estará personalizada con la clase CeldaRutasCiudad. Si se crea una ruta nueva, la vista que se muestra está gestionada por la clase CrearRutaVC y si se accede desde la tabla a una ruta, la clase a la que se accede es RutaElegidaVC.

Clase MapaVC

MapaVC
-titulo : UINavigationItem -distancia : UILabel -nombreLug : string -ciuNombre : string -mapView : MKMapView -location1 : CLLocation -location2 : CLLocation -peticion : MKLocalSearchRequest
+pulsarBotonTipoMapa(entrada : id) : IBAction +pulsarVolver(entrada : id) : IBAction +rutaDesde: hasta:(entrada : MKMapItem, entrada : MKMapItem) : void +inicializarMapa(entrada : string) : void -didSelectAnnotationView(entrada : MKMapView, entrada : MKAnnotationView) : void -rendererForOverlay(entrada : MKOverlay, entrada : MKMapView) : MKOverlayRenderer

Ilustración 31.- Clase MapaVC

Cuando el usuario quiera ver un lugar en el mapa se accederá a la clase MapaVC donde podrá moverse libremente por un mapa marcado con el lugar que ha elegido e incluso crear una ruta.

Clases relacionadas con Comentario

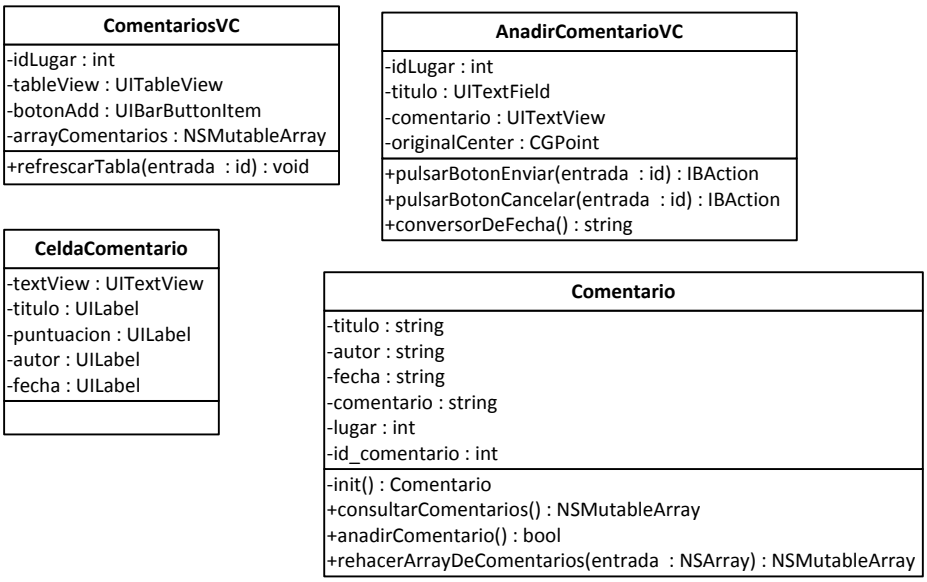


Ilustración 32.- Clases relacionadas con Comentario

Esta clase será la encargada de gestionar la vista donde se mostrarán los comentarios depositados sobre un Lugar y que aparecerán en las celdas personalizadas de la tabla por la clase CeldaComentario. Al añadir un comentario, la clase encargada de la nueva vista que se abre será AnadirComentarioVC.

Clases relacionadas con inicio sesión

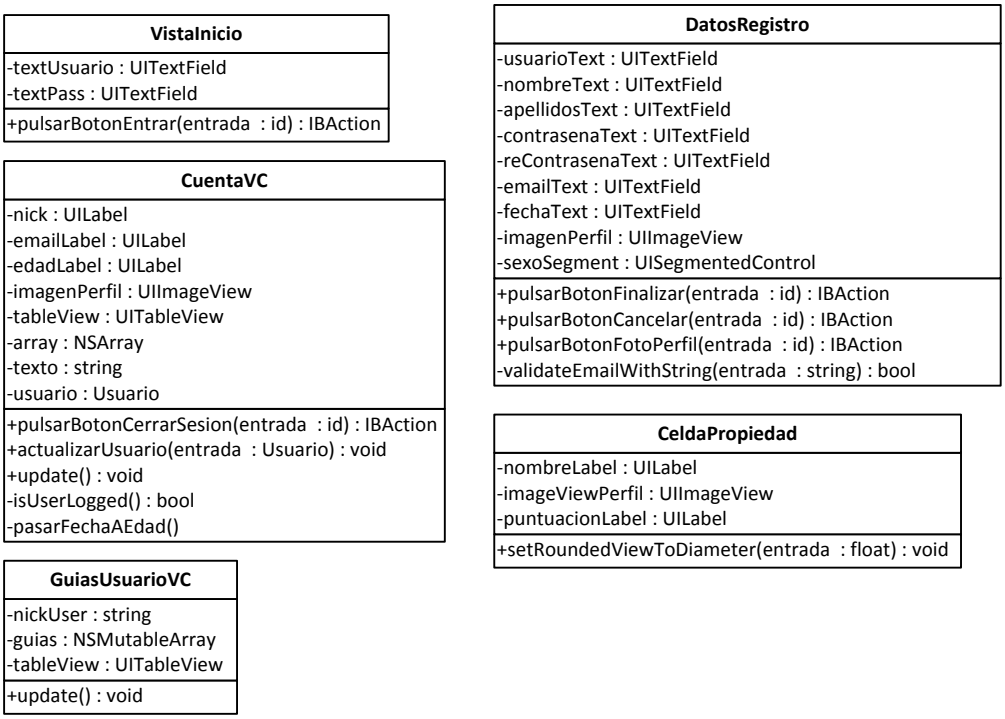


Ilustración 33.- Clases relacionadas con inicio de sesión 1

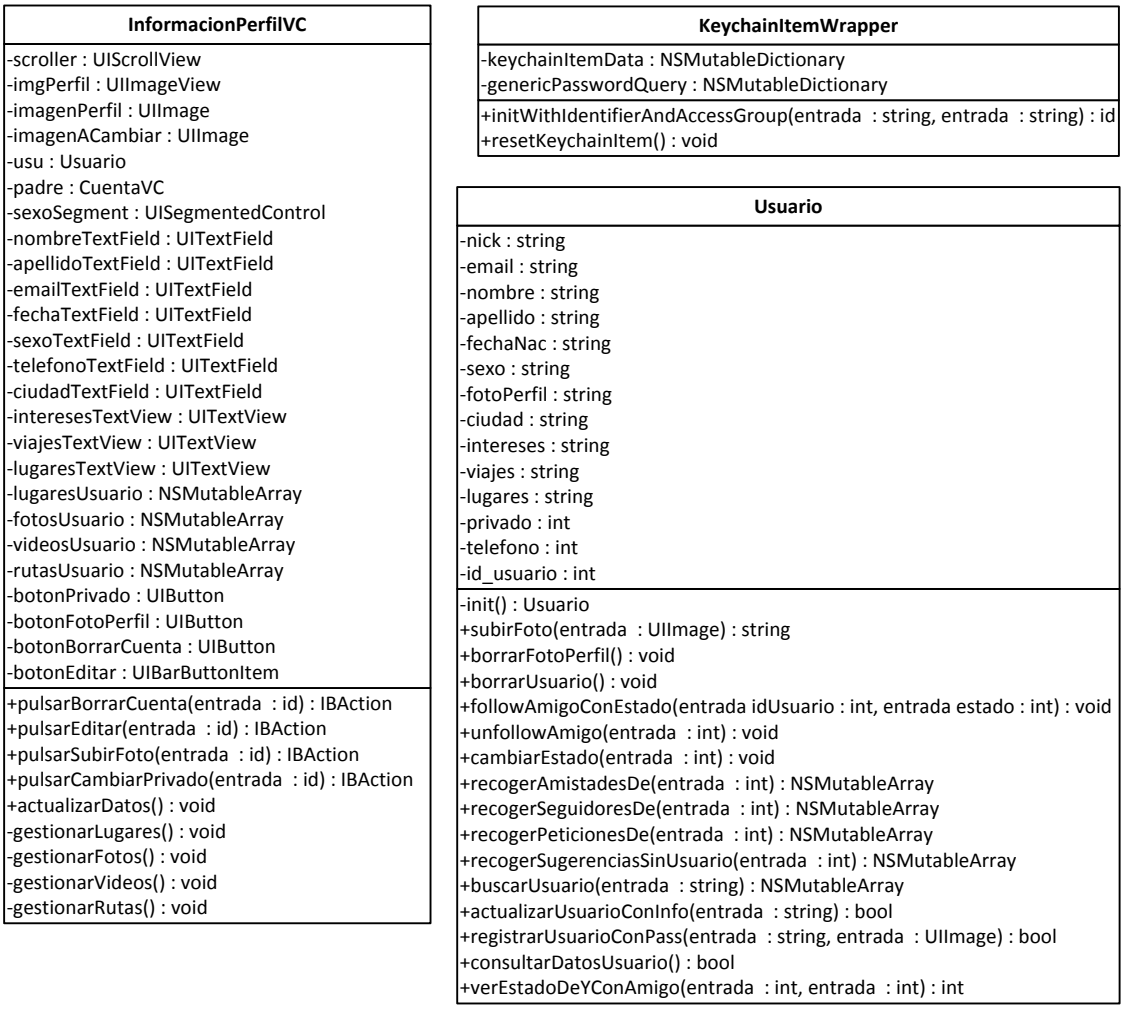


Ilustración 34.- Clases relacionadas con inicio sesión 2

Esta clase será la que aparezca en caso de querer iniciar sesión y el usuario aún no se haya *logueado*. Desde aquí, al igual que desde la pantalla principal, el usuario puede registrarse en el sistema, apareciendo una nueva vista gestionada por la clase DatosRegistro.

Si el usuario inicia sesión correctamente, aparecerá parte de la información del usuario, junto a su foto de perfil, en una nueva vista controlada por la clase CuentaVC. Además, se guardarán las credenciales de forma segura con la clase KeychainItemWrapper. Desde aquí se podrá acceder a varios apartados como la información completa del usuario desde la celda Información, que mostrará una nueva vista controlada por InformaciónPerfilVC.

A su vez el usuario puede acceder a ver las valoraciones que ha hecho sobre las diferentes guías en la vista gestionada por la clase ValoracionesVC, que se mostrarán en una tabla con las celdas personalizadas por la clase CeldaPropiedad.

De manera análoga, si el usuario accede a ver sus guías creadas accederá a una nueva vista que está gestionada por la clase GuiasUsuarioVC y que muestra en la tabla la información con la celda personalizada CeldaPropiedad.

Clases relacionadas con amistades

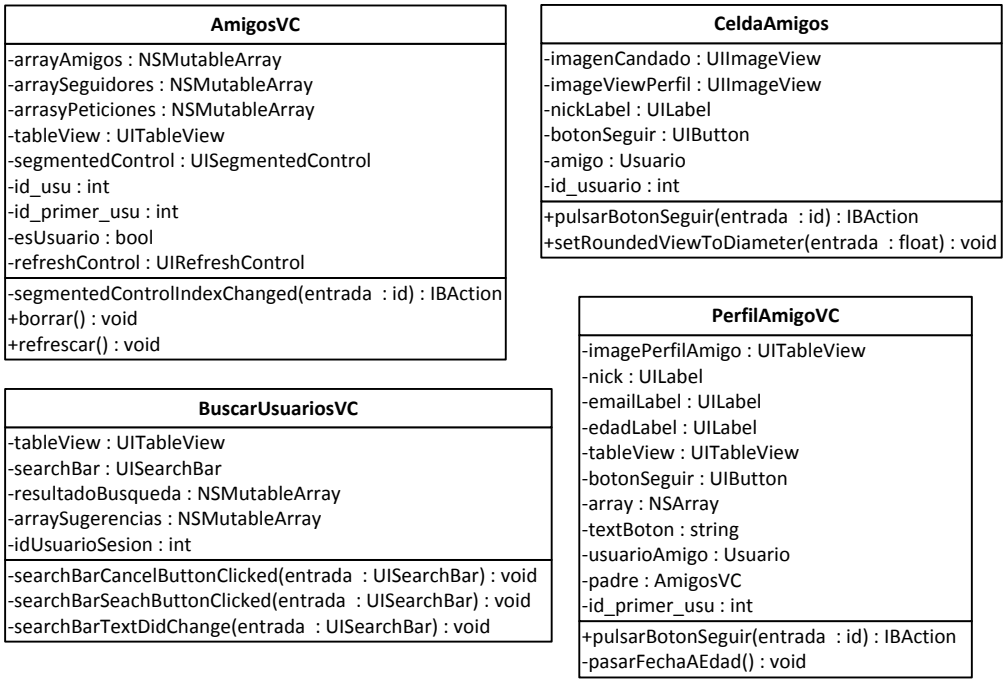


Ilustración 35.- Clases relacionadas con amistades

Esta clase mostrará en una tabla la lista de amigos que sigue el usuario con las celdas personalizadas por la clase CeldaAmigos. El objeto segmentedControl será el encargado de mostrar la información relevante a las pestañas que se pulsen mediante el método segmentedControlIndexChanged, pudiendo cambiar entre los amigos a los que sigue, los que le siguen y los que están pendientes de aceptar, si los hubiera.

Al pulsar en buscar amigos, aparecerá en una lista los usuarios con más participación en la aplicación en una tabla con la misma celda personalizada que antes. Esta nueva vista, además, podrá buscar usuarios registrados en el sistema y que estará gestionada por la clase BuscarUsuariosVC.

Al pulsar sobre un usuario, se accederá al perfil de éste y la vista será gestionada por la clase PerfilAmigoVC.

Clases relacionadas con Mensajes

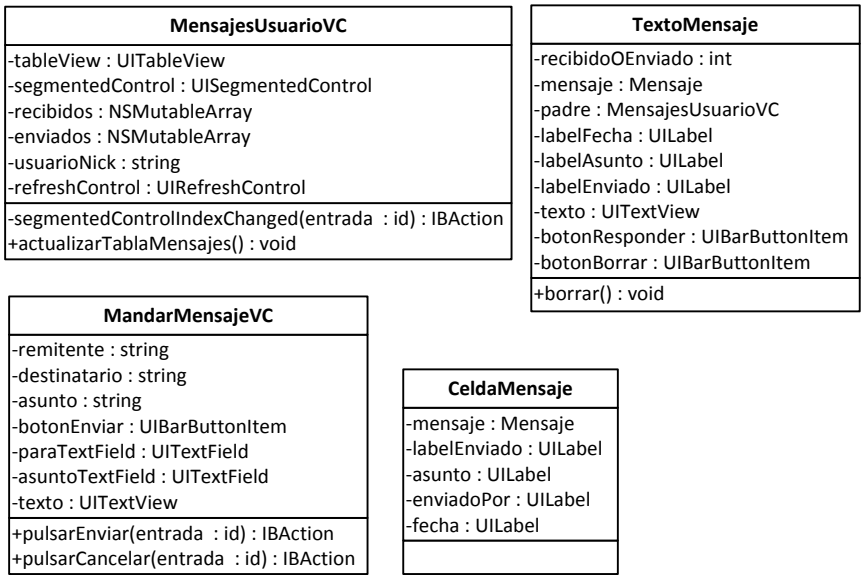


Ilustración 36.- Clases relacionadas con Mensajes

Esta clase mostrara en una tabla con las celdas personalizadas por la clase CeldaMensaje, todos los mensajes recibidos del usuario. El objeto segmentedControl será el encargado de cambiar entre los mensajes recibidos y enviados y también de actualizar la tabla.

Al pulsar sobre un mensaje se accederá a una nueva vista donde se mostrará toda la información de ese mensaje y que estará gestionada por la clase TextoMensaje.

Si el usuario escribe o contesta a un mensaje, la nueva vista será gestionada por la clase MandarMensajeVC.

7. Desarrollo

En este apartado se explicará de forma detallada parte del desarrollo de la aplicación. Como hay apartados que son muy semejantes entre ellos, se describirá sólo uno de ellos con el fin de no repetir información y hacerlo lo más escueto y entendible para el lector de esta memoria.

7.1. Bases de Datos

Para almacenar los datos de la aplicación se ha utilizado una base de datos alojada en un servidor cedido por la universidad y el tutor del proyecto. Sin embargo, para introducir la información se ha utilizado la herramienta phpMyAdmin y MAMP, para hacerlo en local. Al principio se hizo así porque no se tenía la posibilidad de tener un servidor externo, pero más tarde se consiguió tener acceso a un servidor para alojar la base de datos, archivos PHP, imágenes y vídeos relacionados con las guías de la aplicación. Así pues, simplemente se importó la BD al servidor y se dejó completamente funcional.

Para que la aplicación se conecte a esta base de datos se ha utilizado PHP, devolviendo el resultado (si así fuera) en un archivo JSON. La conexión entre Xcode y la base de datos del servidor se hace de la siguiente manera. Primero, Xcode se comunica con los PHP del servidor. Los PHP, dependiendo de la función que Xcode le haya ordenado que ejecute, leerá o escribirá en la base de datos. Una vez que lo haya hecho, devolverá el resultado (si ha leído datos) en formato JSON, para tratar el resultado en Xcode.

JSON, básicamente es un formato para el intercambio de información que lo describe con una sintaxis que se usa para identificar y gestionar esa información. Este archivo JSON es tratado en la aplicación gracias a una clase importada al proyecto llamada CJSONDeserializer. En concreto, el método usado para tal fin es:

```
- (id)deserializeAsArray:(NSData *)inData error:(NSError **)outError;
```

Se optó por utilizar JSON gracias a su simplicidad. También se podía hacer uso de XML, pero esta opción fue desechada, ya que en comparación con JSON el tamaño de los archivos era considerablemente mayor. De la misma manera, a la hora de interpretar los datos y aprender, JSON es más simple y fácil de tratar.

Así, toda consulta que se haga a la BD la aplicación lo recibirá y será almacenado en un array de datos para después mostrarlo en una tabla o hacer lo que se tenga que hacer con esa información.

Por otro lado, está la base de datos local que será la que almacene todo lo relacionado con las descargas que vaya a hacer el usuario. Para crear esta aplicación se ha utilizado una extensión del navegador Mozilla Firefox llamado SQLite Manager[2]. Con esta sencilla aplicación, la BD es exportada e insertada

[2] Más información: https://code.google.com/p/sqlite-manager

en el proyecto de la app y cargada una vez que la aplicación es lanzada. Para cargar la app se ha utilizado la clase AppDelegate que es la encargada de arrancar la aplicación y decir qué es lo que tiene que cargar una vez que la aplicación ha sido abierta.

Por ello, en AppDelegate se ha almacenado el nombre del servidor para que la aplicación se pueda conectar a los PHP y también la dirección de la BD local, que será creada la primera vez que el usuario ejecute la aplicación, con el fin de cargarla en memoria una vez arranque la aplicación desde este método:

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

El método creará todo lo que se le indique. En el caso de esta aplicación, comprobará si la base de datos existe y si no existe, la crea. En concreto, para cargar la base de datos local, se utiliza el método:

```
-(void) cargarBaseDeDatos
```

Este método guarda además la dirección en donde está almacenada para así acceder a ella cuando sea preciso en el atributo de tipo string dataBasePath.

Una vez que se ha llegado a la pantalla principal, y esté todo cargado perfectamente, el usuario tendrá la oportunidad de registrarse en el sistema o entrar al apartado de las ciudades. Ya que el registro simplemente es una conexión al servidor y una inserción de datos en la BD, se empezará a describir el apartado de las ciudades, ya que la conexión es similar en todos los casos.

7.2. Ciudades

Una vez que el usuario accede a este apartado, se accede a las guías que haya descargado, reflejándose las ciudades a las que pertenecen las guías. Se creará un objeto Ciudad, que será el encargado de devolver la lista de las ciudades mediante el método:

```
-(NSMutableArray *) cargarMisCiudades;
```

Para ello se llama al AppDelegate para recoger la ruta en la que está almacenada la Base de Datos local. En este punto, se necesitará el uso del framework sqlite3.h para realizar los pasos necesarios de consulta, inserción, actualización o borrado en la BD local. Los pasos a seguir serán siempre los mismos a lo largo de toda la aplicación. Así que se usará cómo se hace la consulta para las ciudades a modo de ejemplo general.

Hay cinco pasos generales los cuales se pueden diferenciar en: abrir la base de datos, crear la sentencia, ejecutar la sentencia, finalizar la sentencia y cerrar la base de datos.

Para abrir la base de datos se hace uso del método sqlite3_open, al cual se le asignará la dirección de la BD local y devolverá un estado de OK o error, si es esto último, no se hará nada. Si por el contrario se ha abierto correctamente, se

crea la sentencia en un string y se utiliza el método sqlite3_prepare_v2 para evaluar la sentencia. Si es correcta, el método sqlite3_step, dentro de un bucle, irá recogiendo tupla por tupla toda la información de la base de datos a la que se ha accedido. A continuación se crea un objeto Ciudad con todos los datos recogidos y se almacenan en un array, que será el que se va a devolver. Así, una vez que se hayan recogido los datos, se finalizará la sentencia y se cerrará la base de datos con slite3_finalize(sentencia) y sqlite3_close(database) respectivamente.

```
if(sqlite3_open([appDelegate.dataBasePath UTF8String], &database)==SQLITE_OK){
        NSString *sentenciaSQL= [NSString stringWithFormat:@"SELECT * FROM Ciudades
        ORDER BY pais, nombre ASC"];

        if (sqlite3_prepare_v2(database, [sentenciaSQL UTF8String], -1, &sentencia, NULL)==
        SQLITE_OK) {
                while (sqlite3_step(sentencia)==SQLITE_ROW) {

                        //SE RECOGEN LOS DATOS EN STRINGS
                        NSString *id_Ciudad = [NSString stringWithUTF8String:
                        (char *)sqlite3_column_text(sentencia, 0)];
                        (…)

    sqlite3_finalize(sentencia);
  }
  sqlite3_close(database);
```

Para insertar, actualizar o borrar, se usará el mismo esquema. Lo único que cambiará será la sentencia. Esto sirve para toda la información que se ha descargado, ya sean ciudades, lugares, fotos, etc.

Una vez que la información es recogida y almacenada en un array, se mostrará en la tabla. Las tablas tienen métodos ya establecidos para hacer determinadas cosas, como decidir el número de filas que tendrá, las secciones, la altura de la celda, etc. Y como el delegado de la tabla es el controlador de la vista en la que se va a mostrar, se hará siempre uso de varios métodos que ayudarán a crear el resultado final de la tabla.

A continuación, se mostrará en código las declaraciones de esos métodos donde el primer método devolverá la cantidad de ciudades que se haya recogido de la base de datos, para así fijar el número concreto de filas que va a tener la tabla. El segundo en cambio será las secciones que vaya a tener la tabla. En este caso, las ciudades están divididas por países, así que habrá tantas secciones como países se vayan a reflejar en la tabla. Es decir, si se van a mostrar dos ciudades del mismo país y otra ciudad de otro país diferente, aparecerán dos secciones diferenciadas con las ciudades debajo. El método cuatro simplemente devolverá la altura de la celda. Y el quinto método es para crear la cabecera de cada sección. En el caso de las ciudades, la cabecera será el nombre del país al que pertenece. Se puede personalizar el color, el tamaño que tendrá, el tipo de letra, etc.

```
1-(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
```

```
2-(NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section

3-(UITableViewCell*)tableView:(UITableView *)tableView
  cellForRowAtIndexPath:(NSIndexPath *)indexPath

4-(CGFloat) tableView:(UITableView *)tableView
  heightForRowAtIndexPath:(NSIndexPath *)indexPath

5-(UIView *) tableView: (UITableView *)tableView
  viewForHeaderInSection:(NSInteger)section
```

Una vez el controlador de la vista haya recogido las filas y las secciones que tendrá la tabla, será el momento de presentar la información de las ciudades en cada fila. Pare ello se usa el método tres, donde se usará la clase CeldaCiudad para personalizar la celda.

```
static NSString *CellIdentifier = @"CeldaCiudades";

CeldaCiudad *cell =[self.tableView dequeueReusableCellWithIdentifier:CellIdentifier];
```

Todo lo que contenga la celda, aquí se personalizará. El objeto indexPath que es mandado por el objeto tableView, indica el número de la fila y la sección en la que se encuentra en ese momento.

Hay muchos más métodos relacionados con las tablas, pero los más usados y los más importantes son estos cinco, porque sin ellos no se mostraría nada.

Para todo lo demás, se usará la misma metodología. Por ejemplo, a la hora de mostrar la lista de rutas, la lista de amistades del usuario, los menús que tiene una ciudad en concreto, etc. Sin embargo, la lista de lugares y de los comentarios de dichos lugares, no tienen cabecera, es decir, no está dividido en secciones, por lo que el quinto y último método no es usado. Así mismo, en el método dos, se devolverá un 1 indicando que sólo habrá una sección.

Volviendo a la tabla de las ciudades, ésta puede mostrar diferente información gracias al objeto Segmented Control, que será el encargado de ir intercalando la información que se muestra de las ciudades descargadas, las que se encuentran disponibles, así como las más votadas por los usuarios. Para ello, se rige por la misma norma que las tablas. Un objeto Segmented Control tiene métodos ya establecidos para que su delegado o *delegate* haga uso de ellos y que en este caso es el controlador de la vista en la que está. Para ir cambiando de pestaña y mostrando la información en la tabla, se utiliza:

```
- (IBAction)segmentedControlIndexChanged:(id)sender
```

En el momento que el usuario cambie de pestaña, se llamará a ese método. El atributo selectedSegmentIndex, indicará qué pestaña ha sido pulsada y se controlará con un switch para recargar la tabla con la información pertinente.

```
switch (self.segmentedControl.selectedSegmentIndex)
```

Para cada pestaña, se tendrá un array de ciudades diferente con el fin de dividir e identificar la información fácilmente. En el caso de las ciudades tendrá tres, por lo que cuando el usuario cambia de pestaña, si el array está vacío se accede a la BD para recoger la información. Las pestañas se dividen en las guías descargadas (Mis guías), las disponibles y las más votadas. Por lo tanto, si se pulsa en la primera pestaña, se accede a la BD local como se ha explicado anteriormente. En las otras dos habrá que conectarse a la BD del servidor mediante los PHP para recoger la información (se explica un poco más adelante). Una vez que se ha recogido la información, se almacena en el array relacionado con la información consultada y se recarga la tabla para mostrar la nueva información con el método que llamará de nuevo a los cinco métodos anteriores relacionados con la tabla:

```
[self.tableView reloadData];
```

En esta misma vista, al pulsar sobre el botón de la lupa, se abrirá una barra de búsqueda para buscar las ciudades que el usuario quiera. Este objeto llamado UISearchBar, como objeto de Objective-C, tiene métodos asociados. Para ello se utilizarán:

```
-(void)searchBar:(UISearchBar *)searchBar textDidChange:(NSString *)searchText{

-(void)searchBarSearchButtonClicked:(UISearchBar *)searchBar{
```

El primer método es usado para que al cambiar el texto, se compruebe en el array de ciudades las coincidencias con ese nombre. Para ello, se hacen dos bucles, para buscar las coincidencias.

En el siguiente código, se está comprobando con las ciudades que el usuario tiene descargadas. El array donde se almacenan se llama paisesMios. El objeto NSRange es usado para comprobar si el rango de esas letras que ha introducido el usuario, coinciden con el nombre de la ciudad. Si es así, se introduce en el array de resultadosBusqueda. Tras la búsqueda de coincidencia con todas las ciudades, se recarga la tabla con el método [tableView reloadData] de nuevo. Si el usuario estuviera en la pestaña de “Descargas” o “Más Votadas”, la búsqueda se haría con el array “paisesDisponibles” o “paisesMasVotados” respectivamente, en vez de con paisesMios.

```
for (int i=0; i<[paisesMios count]; i++) {

        for (int j=0; j< [[paisesMios objectAtIndex:i]count]; j++) {

          ciud=[[paisesMios objectAtIndex:i]objectAtIndex:j]
         NSRange nameRange=[ciud.nombre rangeOfString:searchText
                                    options:NSCaseInsensitiveSearch];

         if (nameRange.location !=NSNotFound) {
            [resultadosBusqueda addObject:[[paisesMios
                                 objectAtIndex:i]objectAtIndex:j]];
         }
    }
}
```

El segundo método hace referencia a cuando el botón de buscar del teclado es pulsado. Dentro de ese método y con esta sentencia, hará que el teclado desaparezca de la pantalla:

```
[self.searchBar resignFirstResponder];
```

Hasta ahora se ha explicado cómo hacer la conexión a la BD local y recoger las ciudades descargadas y así mostrarlas en la tabla. Sin embargo, para obtener la información de manera externa hay que conectarse a internet ya sea mediante WIFI o 3G.

Para ello, se hará una comprobación de si realmente hay conexión para realizar esa tarea. Al acceder a las guías cuando se arranca la aplicación, se hará la comprobación de conexión.

Como se puede observar en el siguiente código, se hará uso de una clase llamada Reachability que ha sido introducida en el proyecto para llevar a cabo la comprobación de la red. Esta clase es ofrecida por el propio Apple para su uso. En el if, se comprueba el estado del WIFI y el 3G. Si el estado es correcto, habrá conexión y sino, aparecerá una alerta indicando que no hay red:

```
Reachability *reachability = [Reachability reachabilityForInternetConnection];
[reachability startNotifier];
NetworkStatus status = [reachability currentReachabilityStatus];
if(status == ReachableViaWiFi || status ==ReachableViaWWAN){…}

UIAlertView *alerta= [[UIAlertView alloc]initWithTitle:@"¡No estás conectado!" message:@"No podrás
acceder a contenido externo si no estás conectado." delegate:nil cancelButtonTitle:@"OK"
otherButtonTitles:nil, nil];

[alerta show];
```

En caso de haber conexión, se llamará al método consultarCiudades que contiene el objeto Ciudad, al cual se le mandará un indicador para utilizar la función que corresponda dentro del PHP:

```
-(NSMutableArray *)consultarCiudades:(int)indicador{
```

Dentro de este método se prepara la conexión al PHP y la recogida de datos. El código usado se muestra a continuación, en el que primeramente se recoge el nombre del servidor del AppDelegate. Se le añade después el nombre del PHP al final de la dirección con la variable global nombrePHP, que contendrá el nombre del archivo del PHP al que se va a llamar. En este caso *ciudades.php*. A continuación, se crea un string llamado *post* que será la información que se le vaya a pasar al PHP para que lo recoja con el método POST. Aquí sólo se le mandará un número, que será la función que tenga que utilizar.

```
AppDelegate *appDelegate = (AppDelegate *) [[UIApplication sharedApplication]delegate];
NSString *urlServer= appDelegate.nombreServidor;

NSURL *url= [NSURL URLWithString:[urlServer stringByAppendingPathComponent:nombrePHP]];
NSMutableArray *arrayCiudades=[[NSMutableArray alloc] init];
NSString *post=[[NSString alloc] initWithFormat:@"fun=%d",indicador];;
NSData *postData = [post dataUsingEncoding:NSUTF8StringEncoding allowLossyConversion:YES];
NSString *postLength = [NSString stringWithFormat:@"%d",[postData length]];
NSMutableURLRequest *request = [[NSMutableURLRequest alloc] init];
[request setURL:url];
[request setHTTPMethod:@"POST"];
[request setValue:postLength forHTTPHeaderField:@"Content-Length"];
[request setValue:@"application/json" forHTTPHeaderField:@"Accept"];
[request setValue:@"application/x-www-form-urlencoded" forHTTPHeaderField:@"Content-Type"];
[request setHTTPBody:postData];
```

Para que el PHP recoja el valor de las variables necesita del método POST o GET. En esta aplicación se ha utilizado el método POST para mejorar la seguridad y además, permite recoger información de más tamaño que con GET. Por lo tanto, y como es posible que en algún momento se pueda necesitar recoger mucha información, se optó por este método.

Seguido del código anterior, se prepara el request o solicitud para la conexión con todo lo recogido hasta ahora. En el código que sigue se ejecuta de manera síncrona la solicitud de conexión y depende de lo que ocurra devolverá un número. Si el número está comprendido entre 200 y 299, habrá tenido éxito y se habrá devuelto los datos que se hayan solicitado a la BD del servidor. Si no, el método devolverá un array vacío y sin información. En el caso de haber recogido la información, ésta será devuelta con formato JSON. Así pues, se usará la clase descrita anteriormente con nombre CJSONDeserializer para guardarlos en un array. La información que se ha convertido son objetos del tipo id, es decir, objetos que pueden ser cualquier cosa.

```
NSError *error = nil;
NSHTTPURLResponse *response = nil;

   //Hacemos la conexión y nos devuelve la respuesta: "response"
NSData *urlData=[NSURLConnection sendSynchronousRequest:request returningResponse:&response
error:&error];

   //Comprobamos el código de estado HTTP que nos devuelve y si es 2xx habrá realizado la conexión
correctamente.
if ([response statusCode] >= 200 && [response statusCode] < 300)
{
        NSString *responseData = [[NSString alloc]initWithData:urlData encoding:NSUTF8StringEncoding];
        NSError *error=nil;
        NSData *jsonData= [responseData dataUsingEncoding:NSUTF8StringEncoding];
        NSArray *arr=[[CJSONDeserializer deserializer]deserializeAsArray:jsonData error:&error];
        if (arr) {
         arrayCiudades=[self rehacerArrayDeCiudades:arr conIndicador:indicador];
    }
}
return arrayCiudades;
}
```

Para que lo que se devuelva en el array sea del tipo Ciudad, habrá que llamar al método rehacerArrayDeCiudades y se le enviará un *indicador*, que indicará la pestaña en la que el usuario ha pulsado. Si es uno, serán las ciudades disponibles y si es 2, serán las más votadas. Por ello, el array a devolver tendrá

una información u otra. Además, se recorre todo el array para meter la información como objetos Ciudad dentro del array “ciudades”. Este array tendrá las ciudades guardadas por países, esto es, cada array dentro del array “ciudades”, significará un país. Así, finalmente, se tendrá la información de las ciudades para devolverlas y mostrarlas en las tablas.

```
-(NSMutableArray *)rehacerArrayDeCiudades:(NSArray *)arr conIndicador:(int)pIndicador

    NSMutableArray *ciudades=[[NSMutableArray alloc] init];
    if (pIndicador ==1) {
        int cuenta=0;
        for (int i=0; i<[arr count]; i++) {

            //La constructora de Ciudad. Se ha acortado para su comprensión.
                        ciu= [[Ciudad alloc]initWithNombre:[[arr objectAtIndex:i]objectForKey:@"nombre"] ...

            if ([ciudades count]==0) {
                [ciudades addObject:[[NSMutableArray alloc]init]];
                [[ciudades objectAtIndex:cuenta]addObject:ciu];

                    }else if ([[[[ciudades objectAtIndex:cuenta]objectAtIndex:0] pais] isEqualToString:ciu.pais]){

                            [[ciudades objectAtIndex:cuenta]addObject:ciu];

            }else{
                cuenta++;
                [ciudades addObject:[[NSMutableArray alloc]init]];
                [[ciudades objectAtIndex:cuenta]addObject:ciu];
            }
        } elseif (pIndicador==2){
          // los más votados
          }
```

En cuanto al PHP, el código quedaría de la siguiente manera. Al principio, se da las credenciales de la base de datos para su conexión y el número de la función se recoge con el método $_POST[‘funcion’], que será el número de la función a la que se quiera llamar. Después, se conecta y se selecciona la BD y entonces se utiliza el número de la función para llamarla.

La función consultarTodas() refleja lo que hacen todas las consultas de la aplicación. Cuando se hace una consulta (Select) a la base de datos (en este caso ordenado de forma ascendente con el criterio de país y después de nombre), se guarda en un array todos los datos obtenidos y se devuelve en modo JSON. Después se trata como se ha explicado anteriormente.

```
function consultarTodas() {
        //Se lanza la consulta
        $rs = mysql_query("SELECT * FROM ciudades ORDER BY pais,nombre ASC");
         //Se agregan las filas al array
         while($obj = mysql_fetch_object($rs)) {

                        $arr[]=$obj;
         }
         //Devolvemos el resultado
         echo json_encode($arr);
}

//Conectar al servidor de la BBDD
$link = mysql_connect ($host,$uid,$pw) or die("No se puede conectar");
mysql_set_charset('utf8');
//Se selecciona la BBDD
 mysql_select_db($db) or die("No se puede seleccionar la bbdd");
//Qué función usar
if ($funcion==0) {
        consultarUnaCiudad();
}elseif ($funcion==1) {
        consultarMasValoradas();
}elseif (…)
```

Con todos estos elementos, ahora ya se puede ver la tabla con la lista de ciudades dentro de cada celda. Toda la aplicación se rige sobre estos criterios y esta forma de hacer las conexiones y mostrar la información. Por lo tanto, a partir de ahora ya se obviará todo lo explicado hasta ahora, con el fin de no repetir lo mismo todo el rato.

7.3. Geolocalización

En este apartado se explicará cómo se usa la Geolocalización en la aplicación.

Para ello, se usará la clase MapaVC, que será el controlador de la vista del mapa. Se necesitarán dos frameworks llamados MapKit y CoreLocation. A pesar de que existe una SDK de Google Maps para utilizar en Xcode, se ha decidido usar los mapas de Apple para tener el menor problema posible a la hora de programar. Según Applesencia[3], a la hora de desarrollar para iOS, *"es preferible utilizar las herramientas que Apple ofrece antes que usar herramientas externas"*, en este caso, de Google.

Para navegar por un mapa se necesita el objeto MKMapView que será introducido en la vista. Para usar los métodos del objeto MKMapView, es necesario que el controlador de la vista sea su delegado.

A la clase MapaVC le enviará el nombre del Lugar que se quiera ver en el mapa y el nombre de la ciudad a la que pertenece.

Abajo, se muestra el código que se utilizará para inicializar el mapa. Lo primero que se hará, será utilizar el nombre del sitio que se quiere mostrar y que habrá sido enviado previamente.

El atributo *petición*, es del tipo MKLocalSearchRequest, que será utilizado para

[3] Referencia: http://applesencia.com/2013/04/apple-mapkit-vs-google-maps-sdk

especificar los parámetros de búsqueda en el mapa y que permitirá, por ejemplo, fijar la dirección o una región específica para restringir los resultados de búsqueda. Será pues, la instancia del objeto MKLocalSearch, llamada *search*, que será la encargada de realizar la operación de búsqueda en el mapa y entregar los resultados de vuelta de manera asíncrona.

Estos resultados vendrán como objetos MKMapItem, que son objetos que encapsulan la información sobre un punto específico en el mapa. Como se recibirán varios ítems o direcciones, y sólo interesa el primero, se guardará tan sólo esa primera dirección si es que ha sido encontrada y se mostrará en el mapa con una anotación o pin. En caso de que no haya encontrado ninguna coincidencia, se mostrará una alerta para que el usuario pueda introducir una nueva dirección y se repetirá el mismo proceso para buscar la dirección en el mapa.

```
-(void) inicializarMapa:(NSString *)sitio{
    peticion.naturalLanguageQuery =sitio;
    peticion.region = mapView.region;

    MKLocalSearch *search= [[MKLocalSearch alloc] initWithRequest:peticion];

[search startWithCompletionHandler:^(MKLocalSearchResponse *response,  NSError *error) {
NSMutableArray *puntosInteres= [[NSMutableArray alloc]init];
 for (MKMapItem *item in response.mapItems){
          [puntosInteres addObject:item.placemark];
 }
 [mapView removeAnnotations:mapView.annotations];
  if ([puntosInteres count]!=0) {
          [mapView showAnnotations:[[NSArray alloc]initWithObjects:[puntosInteres objectAtIndex:0], nil]
          animated:YES];

       }else{

            UIAlertView *alerta = (…)
       }
    }];
}
```

El objeto MKMapView, tiene métodos que puede usar el delegado de ese objeto, que como siempre es el controlador de la vista.

```
- (void)mapView:(MKMapView *)mapView didSelectAnnotationView:(MKAnnotationView *)view;

-(MKOverlayRenderer *) mapView:(MKMapView *)mapView rendererForOverlay:(id<MKOverlay>)overlay;
```

Estos dos métodos se usarán para crear la ruta que establece la posición del usuario con el lugar elegido y mostrado en el mapa. Si no se ha podido anotar en el mapa ninguna dirección, estos dos métodos nunca serán llamados.

El primer método será el encargado de realizar la ruta entre la posición del usuario y la anotación o pin que se muestra en el mapa al pulsar sobre ella. Los objetos CLLocation, permiten guardar la longitud y latitud de los lugares en el mapa. Así, en el código, se recoge esa información de la anotación en el mapa y además la de la posición del usuario. Como la distancia entre ambos puntos también se va a mostrar en la pantalla, se calcula con el objeto CLLocationDistance y el método distanceFromLocation y se divide entre 1000

para mostrarlo en kilómetros. Entonces, se crean los dos Items para realizar la ruta entre ambos puntos con la información recabada hasta ahora y se llama al siguiente método, llamado rutaDesde:hasta.

```
- (void)mapView:(MKMapView *)mapView didSelectAnnotationView:(MKAnnotationView *)view
{
    CLLocation *pinLocation = [[CLLocation alloc] initWithLatitude:[[view annotation] coordinate].latitude
longitude:[[view annotation] coordinate].longitude ];

    CLLocation *userLocation =(…)

    //Dividimos entre 1000 para conseguir los km
    CLLocationDistance distance = [pinLocation distanceFromLocation:userLocation]/1000;
    [distancia setText: [NSString stringWithFormat:@"%.3f km.", distance]];

    MKPlacemark *placemark= [[MKPlacemark
alloc]initWithCoordinate:CLLocationCoordinate2DMake(pinLocation.coordinate.latitude,pinLocation.coordi
nate.longitude) addressDictionary:nil];
    MKMapItem *mapItem = [[MKMapItem alloc]initWithPlacemark:placemark];

    MKPlacemark *placemarkUL= (...)
    MKMapItem *mapItemUL= [[MKMapItem alloc]initWithPlacemark:placemarkUL];
    [self rutaEntre:mapItem hasta:mapItemUL];
}
```

El método “rutaDesde:hasta”, es usado para calcular la ruta óptima entre esos dos puntos. Lo que se comprueba es que no haya errores a la hora de crear la ruta. De haberlo muestra una alerta, pero sino crea una línea “imaginaria” entre ambos puntos. Para hacerla visible se hace uso del segundo método de MKMapView que se ha nombrado más arriba. Con él, se dibujará la ruta calculada comprendida entre ambos puntos. Dicha línea podrá ser personalizada con el color, la anchura, etc.

```
MKDirections *directions = [[MKDirections alloc]initWithRequest:request];
[directions calculateDirectionsWithCompletionHandler: ^(MKDirectionsResponse *response, NSError
*error) {
        if (error) {
            UIAlertView *alerta = (…)
         }else{
            if ([response.routes count]>0) {
                [mapView removeOverlays:mapView.overlays];
                MKRoute *ruta= response.routes[0];
                [mapView addOverlay:ruta.polyline level:MKOverlayLevelAboveRoads];
            }
        }
    }];

-(MKOverlayRenderer *) mapView:(MKMapView *)mapView rendererForOverlay:(id<MKOverlay>)overlay{

MKPolylineRenderer *renderer= [[MKPolylineRenderer alloc]initWithOverlay:overlay];

renderer.strokeColor = [UIColor redColor];
renderer.lineWidth = 3.0;
return renderer;

}
```

Además, la aplicación hace uso de la opción de visualización del mapa. Usando el siguiente método, se consigue que el mapa cambie de estilos como tipo estándar, satélite o un híbrido de las anteriores.

```
-(IBAction)pulsarBotonTipoMapa:(id)sender{
    UIBarButtonItem *boton= (UIBarButtonItem *) sender;
    if (boton.tag == 1){
        [mapView setMapType:MKMapTypeSatellite];

    }else if (boton.tag ==2){
        [mapView setMapType:MKMapTypeStandard];

    }else{
        [mapView setMapType:MKMapTypeHybrid];
    }
}
```

7.4. AVSpeechSynthesizer

Esta funcionalidad ha sido incluida una vez acabada la aplicación. Debido a que había tiempo suficiente para introducir nuevos aspectos a la aplicación, se decidió introducirla porque le daba al resultado final útil y elegante.

Así, AVSpeechSynthesizer permite la reproducción de texto con voz. Para ello, se ha de incluir el *Framework* AVFoundation.

El controlador de la vista, tendrá un atributo del tipo AVSpeechSynthsizer llamado *altavoz*, que será el encargado de reproducir el texto de la descripción de un lugar. Para que eso ocurra, el usuario pulsará sobre el botón con el dibujo del altavoz que llamará al código de más abajo.

Cada vez que ese botón es pulsado se comprueba si el atributo *altavoz* se está reproduciendo o está en pausa. Al pulsar el botón, si está en pausa, continúa dictando el texto. Sino, se comprueba si está hablando o no. Si está hablando se detiene para continuar posteriormente. Pero si no está hablando, se crea una instancia del objeto AVSpeechUtterance que será el encargado de recoger el texto que se va a dictar y un conjunto de parámetros como pueden ser la voz, el tono y la velocidad del discurso. Entonces se le agrega esa configuración al *altavoz* y se ejecuta.

```
- (IBAction)pulsarVotonAltavoz:(id)sender {
    if (altavoz.isPaused) {
        [altavoz continueSpeaking];

    }else if (!altavoz.isSpeaking) {
          AVSpeechUtterance *speechUtterance= [[AVSpeechUtterance
              alloc]initWithString:textView.text];

          [speechUtterance setRate:0.25f];

          speechUtterance.voice=[AVSpeechSynthesisVoice voiceWithLanguage:[AVSpeechSynthesisVoice
          currentLanguageCode]];

           speechUtterance.pitchMultiplier=1.2;

          [altavoz speakUtterance:speechUtterance];
    }else{
          [altavoz pauseSpeakingAtBoundary:AVSpeechBoundaryImmediate];
    }
}
```

El objeto AVSpeechSynthesizer tiene también algunos métodos que su delegado puede usar. Como por ejemplo cuando el discurso ha acabado y se quiere la reproducción sea detenida, o cambiar la apariencia del botón.

```
-(void)speechSynthesizer:(AVSpeechSynthesizer *)synthesizer
didFinishSpeechUtterance:(AVSpeechUtterance *)utterance{

        [botonAltavoz setImage:[UIImage imageNamed:@"altavoz-on.png"]
        forState:UIControlStateNormal];

}
```

7.5. Imágenes y vídeos

Para subir las imágenes y los vídeos al servidor se usará PHP tanto para almacenar la información en la base de datos externa, como el propio archivo en las carpetas que están alojadas en el servidor. Ya se ha explicado cómo hacer la conexión al PHP y cómo se envía la información, así que este apartado se centrará en la subida del propio archivo al servidor.

Cabe diferenciar que habrá tres “tipos” de imágenes para subir al servidor. Imágenes de las ciudades o lugares; perfiles de los usuarios, de los lugares o de las ciudades; y por último, las imágenes de metros. Cada tipo de foto estará en una carpeta diferente en el servidor, para así tenerlas más organizadas. Se separarán, a su vez, las fotos del “tipo” perfil en las tres variantes que hay: usuarios, ciudades y lugares.

La forma de subir un archivo al servidor, ya sea vídeo o foto, se muestra en el código de más abajo. Lo primero de todo, como siempre, es recoger del AppDelegate el nombre del servidor y se le añade al final la carpeta donde se almacenará la foto y además el nombre del archivo del PHP, *upload.php*. Este archivo será usado para recoger el archivo y crearlo en la carpeta.

Después se empieza a crear la solicitud o *request*. Siempre que se vaya a subir un archivo es necesario un límite o *bundary*. Los números que se guardan en el string *bundary* son simplemente aleatorios. Este string, con estos números, es el más utilizado y para no tener complicaciones, se ha seguido los pasos que el resto de desarrolladores utiliza para la subida de ficheros al servidor.

Después de haber creado la solicitud y los límites, ahora hay que generar el cuerpo del *post*. En la variable *body* se está añadiendo todo lo relacionado con la imagen gracias al objeto NSMutableData, que es usado para el almacenamiento de la foto y asumir su comportamiento. Así, a la hora de pasarle la foto al PHP, éste puede manejar su información fácilmente.

Finalmente, el *body* se configura en el *request* y se establece la conexión al servidor.

```
NSData *imageData = UIImageJPEGRepresentation(imagenPerfil, 90);

NSString *urlString = [urlServer
        stringByAppendingPathComponent:@"imagenesPerfil/upload.php"];

NSMutableURLRequest *request = [[NSMutableURLRequest alloc] init];
[request setURL:[NSURL URLWithString:urlString]];
[request setHTTPMethod:@"POST"];
NSString *boundary = [NSString stringWithFormat:@"---------------------------
14737809831466499882746641449"];

[request addValue:contentType forHTTPHeaderField:@"Content-Type"];

 NSMutableData *body = [NSMutableData data];
[body appendData:[[NSString stringWithFormat:@"\r\n--%@\r\n", boundary]
dataUsingEncoding:NSUTF8StringEncoding]];
[body appendData:[NSData dataWithData:imageData]];
[body appendData:[[NSString stringWithFormat:@"\r\n--%@--\r\n", boundary]
dataUsingEncoding:NSUTF8StringEncoding]];

[request setHTTPBody:body];
NSData *returnData = [NSURLConnection sendSynchronousRequest:request returningResponse:nil
error:nil];
```

Una vez establecida la conexión, el PHP recogerá la imagen, usará un nombre numérico aleatorio entre 1 y 999999, con el fin de no repetir los nombres y así, crear el un archivo en la carpeta de imagenesPerfil con los datos que está recogiendo. El PHP devuelve el nombre aleatorio que se le ha dado al archivo con la sentencia echo $newName. Esto se crea si todo ha sido correcto. Como se puede observar a continuación, se recoge el archivo y se le da el nombre aleatorio. Después se sube comprobando que la imagen no sea superior a 1MB (1000000 bytes). Así, el código quedaría en el PHP de la siguiente manera:

```
if (is_uploaded_file($_FILES['userfile']['tmp_name'])) {

} else {
        exit("Archivo no subido");
}

if ($_FILES['userfile']['size']> 1000000) {
        exit("El archivo es demasiado grande");
}

if (move_uploaded_file($_FILES['userfile']['tmp_name'], $newName)) {

   echo $newName;
}
```

Lo que se hará a continuación, una vez que el PHP ha devuelto el nombre de la foto que se ha creado en el servidor, será insertar la información en la BD. El procedimiento a la hora de conectar será el mismo, con la diferencia de tener en cuenta si la foto que se ha subido es de una ciudad o de un lugar. En caso de ser de una ciudad, el campo id_lugar de la BD quedará como NULL indicando que la foto pertenece a una ciudad.

El PHP para almacenar la información en la BD, recoge la información que se le manda de la foto con el método $_POST['variable']. Pero como las fotos en la aplicación son puntuables, se tiene que crear un nuevo valor en la tabla Objetos con el fin de gestionar más adelante las puntuaciones. Por lo tanto, el

número del nuevo Objeto que se haya creado, se recoge y se inserta en el campo id_objeto de la tabla Fotos. Así, y con los datos que se ha recogido, finalmente quedará guardado en la BD toda la información relevante a esa foto, incluida la dirección donde está alojada.

Éste es el procedimiento para subir una imagen al servidor. Los vídeos usan exactamente el mismo sistema, incluido el PHP upload.php. La única diferencia que tiene el PHP de las fotos y el PHP de los vídeos, es el tamaño máximo que puede tener el fichero a subir. En el caso del vídeo es de 50MB(50000000 bytes). A continuación, se muestra el código:

```
mysql_query("INSERT INTO Objetos (id) VALUES (NULL)");
$id = mysql_insert_id();
if ($lug==0){
        $sql ="INSERT INTO Fotos… “;
}else{
        $sql ="INSERT INTO Fotos… “;
}
$rs = mysql_query($sql);
```

Así, las fotos y vídeos de la aplicación quedarían almacenados de manera ordenada y con nombres diferentes en el servidor, para que en cualquier momento se pueda acceder desde la aplicación.

7.6. Puntuaciones

El sistema de puntuaciones ha sido creado como el “Like” de Facebook. En un principio se iba a hacer con un sistema de votaciones, el cual permitiese puntuar un lugar, foto, vídeo o ruta con una valoración de 1 a 5. Como el sistema actual y final era más intuitivo y además menos engorroso para la interfaz, se optó por utilizar este sistema. Así mismo, el usuario es más propicio a darle a un botón si le gusta algo que empezar a pensar qué puntuación le va a dar. Por lo tanto, con una simple pulsación en el botón para tal fin, el usuario puede votar todo lo que le guste, más fácil y rápido.

Para ello, se tuvo que crear una tabla en la base de datos con nombre Objeto con una sola columna, que sería un valor número con el nombre *id* (haciendo alusión a identificador). Por lo tanto, cada elemento puntuable en la aplicación, tendrá un id que haga referencia a esta tabla.

También, para almacenar las puntuaciones, se creó una tabla llamada Puntuaciones. Aquí sólo se guardaría el *id_objeto* del objeto que se ha puntuado, el nick del usuario que ha hecho la puntuación y, por supuesto, un *id_puntuación* que hará referencia al identificador de esa puntuación. Así, se habrá almacenado quién ha votado qué.

En la aplicación, existe la clase de tipo Objeto. Este Objeto es el padre de las clases Lugar, Recurso y Ruta. Por lo tanto, todo lo que tenga el padre (Objeto) también lo tendrán los hijos (Lugar, Recurso, Ruta) por el principio de herencia. Así pues, el método de puntuar y quitar puntuación estará implementado en la clase Objeto, pero podrá ser usado por los *hijos*.

En esos dos métodos, se hace la conexión igual que como se ha explicado anteriormente. La única diferencia es los parámetros que se le pasa al PHP.

```
-(void)puntuar:(int)tipoObjeto ySiPerteneceAUnLugar:(int)chivato

-(void)quitarPuntuacion:(int)tipoObjeto ySiPerteneceAUnLugar:(int)chivato
```

Por consiguiente, para crear las puntuaciones en las tablas, se ha usado la siguiente función en el PHP.

En primer lugar, se le pasará al PHP cinco parámetros:

- $obj : Es el id del objeto.
- $nick : El nick del usuario.
- $chi : Es el id del lugar que servirá como chivato. Si es 0, es que la puntuación no está relacionada con ningún lugar.
- $tab : El nombre de la tabla en la BD a la que se actualizará la puntuación. Para saberlo, se hará uso del atributo "tipoObjeto" que será enviada a los métodos de la clase Objeto.
- $ciu : El id de la ciudad, para actualizar su puntuación.

Lo primero que se hace es añadir un registro de la votación en la tabla Puntuaciones. Después hay que actualizar la puntuación del objeto que ha sido votado. Por lo tanto se actualiza el campo "puntuación" de la tabla a la que pertenezca el objeto. La tabla indicará la variable "$tab". Así, hará un recuento y actualizará la puntuación total del objeto puntuado.

Sin embargo, si se puntúa un vídeo o una foto, esta puntuación afecta a la puntuación total del lugar, si es que pertenecen a algún lugar. Así, la variable $chi, tendrá el id_lugar de dicha foto o vídeo. Si su valor es 0, es que la foto o vídeo pertenece a la ciudad y por lo tanto no habría que modificar la puntuación de ningún lugar. Sea como sea, cuando un usuario puntúa alguna foto, vídeo, ruta o lugar, se actualizará la puntuación total de la ciudad a la que pertenezca. Y así, quedaría almacenada la puntuación de un usuario sobre un objeto y la puntuación total de la ciudad a la que pertenezca quedará igualmente actualizada. A continuación, todo lo dicho anteriormente queda reflejado en siguiente código:

```
function puntuar() {
  //se recogen los datos
    (...)

        mysql_query("INSERT INTO Puntuaciones…");

        mysql_query("UPDATE $tab SET puntuacion = …");

if ($chi!=0) {

        $que=mysql_query("SELECT id_objeto FROM Lugares WHERE id_lugar=$chi");
        $result= mysql_result($que,0);

           mysql_query("UPDATE Lugares SET puntuacion = …");
}
mysql_query("UPDATE ciudades SET puntuacion =…");
}
```

Cada vez que un usuario registrado y que haya iniciado sesión en el sistema, acceda a ver la información de un lugar, foto, vídeo o ruta, se hará una comprobación a la BD del servidor, con el fin de comprobar si el usuario ha votado ya ese objeto o no. Para ese caso, se hará uso del siguiente método, también alojado en la clase padre Objeto:

```
-(BOOL)comprobarSiVotado:(NSString *)pUser
```

En este método de nuevo se hará la conexión al PHP y se consultará a la base de datos si el usuario ya había votado anteriormente. Devolverá un YES o NO en consecuencia.

7.7. Iniciar Sesión

Para iniciar sesión el usuario se ha tenido que registrar previamente. Cuando lo haya hecho irá al apartado de "Cuenta". Se comprobará si previamente ya había iniciado sesión, esto es, que sus credenciales estén guardadas. Para ver las credenciales, se utiliza una clase con el fin de guardarlas y que el usuario no tenga que estar venga a meter sus datos cada vez que accede a la aplicación. Esta clase se llama KeychainItemWrapper. Lo que permite es guardar en modo seguro las credenciales del usuario al iniciar sesión. Para ello, encripta la información al introducir la información, y la des-encripta al recogerla. Por ello, es usada de forma segura por muchas aplicaciones. Para llevar a cabo esto, se utiliza los siguientes métodos:

```
//Guardar información
KeychainItemWrapper *keychainItem = [[KeychainItemWrapper alloc] initWithIdentifier:@"FollowLogin"
accessGroup:nil];

[keychainItem setObject:[NSString stringWithFormat:@"%@",textPass.text ] forKey:(__bridge
id)kSecValueData];
[keychainItem setObject:[NSString stringWithFormat:@"%@",textUsuario.text ]forKey:(__bridge
id)kSecAttrAccount];
```

```
//Recoger información
KeychainItemWrapper *wrapper = [[KeychainItemWrapper alloc] initWithIdentifier:@"FollowLogin"
accessGroup:nil];

NSString *username = [wrapper objectForKey:(__bridge id)kSecAttrAccount];
NSString *password = [wrapper objectForKey:(__bridge id)kSecValueData];
```

Como se ve en el código, es muy fácil tanto guardar la información como recogerla. Para guardarla, se ha cogido el texto de los campos habilitados para introducir el nombre de usuario y el *password*. Estos campos se llaman textPass y textUsuario, que recogiendo el texto que hay en ellos, se guardan en el llavero gracias a esas sentencias.

Para recogerlo se utiliza la misma idea. Se accede al lugar donde está guardada la información y se recoge. El identificador de la clase, en este caso "FollowLogin", indicará que pertenece a esta aplicación. Así no habrá ningún problema cuando se quiera acceder a esta información. Para acceder a esta información en toda la aplicación, se hará siempre de la misma manera.

Para que el usuario inicie sesión, se comprueba previamente si lo ha hecho. Si lo ha hecho, se recoge la información del usuario desde la BD externa, con las mismas conexiones que se ha hecho hasta ahora, y se muestra en pantalla. Si no ha iniciado sesión todavía, aparecerá la vista de iniciar sesión (VistaInicio). Aquí se dará la oportunidad de registrarse al usuario, si no lo ha hecho aún.

El usuario meterá los datos en los campos que se indica y al introducirlos, pulsa en entrar.

Se comprobará que se han rellenado y que ninguno de los dos campos esté vacío. Así, el primer paso es hacer una consulta a la BD del servidor, para ver si el usuario existe. La contraseña del usuario, al registrarse, es encriptada en la base de datos con el fin de aumentar la seguridad. Para encriptar la contraseña del usuario y guardarla en la base de datos, se utilizó en primer lugar el algoritmo md5, pero este algoritmo tiene vulnerabilidades, así que se rechazó. Con el fin de que la contraseña quedara almacenada lo más segura posible, se utilizó finalmente el algoritmo SHA-512, uno de las funciones hash más seguras y robustas hoy en día. Para hacerlo, en el PHP se utilizará la siguiente sentencia:

```
$pass=hash('sha512', $_POST['pass']);
```

Esta sentencia será usada para comparar la contraseña almacenada mediante esta función en la base de datos. Si coinciden significará que es correcta. Pero antes, se comprobará que el usuario esté registrado. Si no existe, mostrará un error. Si existe, se recoge su información de la BD y se guardan sus credenciales con los métodos explicados arriba.

Sin embargo, para indicar a la clase donde se ha comprobado si se ha iniciado sesión (CuentaVC), se utiliza el objeto NSNotificationCenter, que permitirá interactuar entre las dos clases.

Lo que se hace en las líneas de código que se presentan más abajo, es primero crear el objeto NSNotificationCenter con un identificador para saber cuándo es llamado. Este identificador se llamará “DoUpdate”, aunque se le puede dar el nombre que uno quiera. Así mismo, una vez que se llame mediante este identificador, usará el método “update” para actualizar la información del usuario.

La llamada se hace en la clase VistaInicio. Lo que hace es simplemente ejecutar el método update en la clase CuentaVC:

```
//EN CuentaVC
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(update) name:@"DoUpdate"
object:nil];

-(void)update{

[usuario consultarDatosUsuario];
nick.text=usuario.nick;
emailLabel.text=usuario.email;
[self pasarFechaAEdad];
(…)
}

//En VistaInicio
[[NSNotificationCenter defaultCenter] postNotificationName:@"DoUpdate" object:nil userInfo:nil];
```

En el código anterior, se puede observar que se llama al método "pasarFechaAEdad", que lo que hará es convertir la fecha de nacimiento del usuario, a la edad que tiene ahora. Se hará de la siguiente manera:

En primer lugar se le da un formato a la fecha. En este caso, día/mes/año. Después se crea el objeto NSDate con la fecha de nacimiento del usuario. Se recoge la fecha del día actual y se guarda esa información en el objeto NSCalendar. Este objeto tiene un método que convierte la fecha a edad de manera muy sencilla gracias al método "year". ([ageComponents year]).
Se hace de la siguiente manera:

```
-(void)pasarFechaAEdad{
        NSDateFormatter *dateFormat = [[NSDateFormatter alloc] init];
        [dateFormat setDateFormat:@"dd/MM/yyyy"];
        NSDate *birthday = [dateFormat dateFromString:[NSString
        stringWithFormat:@"%@",usuario.fechaNac]];

        NSDate *ahora = [NSDate date];
        NSDateComponents *ageComponents = [[NSCalendar currentCalendar]
                                                components:NSYearCalendarUnit
                                                  fromDate:birthday
                                                toDate:ahora
                                                options:0];

    NSInteger age = [ageComponents year];
    edadLabel.text =[NSString stringWithFormat:@"%d",age];
}
```

Así pues, el usuario habrá iniciado sesión y habrá guardado sus credenciales en el "llavero" de la aplicación con el fin de no introducirlos cada vez que ejecuta la aplicación.

7.8. Amistades entre usuarios

Un usuario puede gestionar a quién seguir o dejar de seguir. Para ello se ha utilizado la idea de "Twitter" y su gestión de las amistades.

El usuario cuando accede a la apartado de amigos, verá tres pestañas con las que interactuar: Siguiendo, Seguidores y Peticiones.

Un usuario puede bloquear su perfil con el fin de que otros usuarios que no le

siguen no puedan ver su información. Esto será controlado por el campo "privado" de la tabla de Usuarios en la base de datos externa.

La idea que se persigue es la siguiente: Si un usuario A quiere seguir a un usuario B, se comprueba si el usuario B tiene su perfil bloqueado. Si es así, se hará una petición de amistad. En la BD se creará un nuevo campo con el identificador (*id_usuario*) de ambos usuarios, pero también con un estado que indicará la amistad que hay entre ambos. Así, si el estado tiene un 0, es que el usuario A está siguiendo al usuario B, pero si es 1, es que el usuario A ha hecho una petición al usuario B porque éste tiene el perfil bloqueado.

Por lo tanto, el usuario B, al acceder a su lista de amigos, verá que tiene una petición de amistad y podrá aceptarla con el botón habilitado para tal efecto. Si el usuario B acepta la petición, ésta se borrará de la tabla y aparecerá en el apartado de "Seguidores".

Para hacer todo esto, se ha modificado el botón para hacer el seguimiento de las amistades con la información del estado en el que se encuentra la amistad. Al pulsar el botón, se mira qué nombre tiene el botón, que puede ser "Siguiendo", "Seguir", "Pendiente" o "Aceptar".

Básicamente lo que se hace es el estado en el que se encuentra el botón en el momento de pulsarlo. Se comprueba y se ejecuta la acción en consecuencia. En resumidas cuentas, cuando se pulsa el botón se haría lo siguiente:

-Si "Siguiendo": La amistad se borra de la BD externa.
-Si "Seguir": Se comprueba si el usuario B tiene el perfil bloqueado, si es así, se crea en la BD un registro con el campo estado a 1. El botón quedaría cambiado a "Pendiente".
-Si "Aceptar": La amistad cambia de estado y el campo *estado* de la tabla Amistades, cambia a 0.
-Si "Pendiente": Se hace una comprobación de si está seguro de cancelar la petición de amistad. Si pulsa sí, se borra el registro de la BD.
El código a continuación, refleja cómo se hace:

```
if ([botonSeguir.titleLabel.text isEqualToString:@"Siguiendo"]) {
        (…)
        [amigo unFollowAmigo:id_usuario];

}else if ([botonSeguir.titleLabel.text isEqualToString:@"Seguir"]){
        (…)
         [amigo followAmigo:id_usuario conEstado:amigo.privado];

}else if ([botonSeguir.titleLabel.text isEqualToString:@"Aceptar"]){
        (…)
        [amigo cambiarEstado:id_usuario];

}else if ([botonSeguir.titleLabel.text isEqualToString:@"Pendiente"]){
        UIAlertView *updateAlert = [[UIAlertView alloc] initWithTitle: @"Cancelar Petición" message:
        @"¿Realmente quieres cancelar la petición de amistad?" delegate: self cancelButtonTitle: @"NO"
        otherButtonTitles:@"SI",nil];

         [updateAlert show];
}
```

7.9. Enviar Mensajes

El envío de mensajes entre usuarios se hace de manera básica. Es decir, un destinatario, un asunto, el texto o cuerpo del mensaje y la fecha/hora en la que se ha enviado. En un principio, la idea era hacer un servicio de chat como Whatsapp, Line o Telegram. Sin embargo, en aquel momento no se tenía un servidor disponible, ni los conocimientos necesarios para llevarlo a cabo. Así, y viendo que se iba a gastar mucho tiempo en la búsqueda de la información y después en implementarlo, se decidió hacerlo de la manera que se explica a continuación.

Cada mensaje es guardado en la base de datos del servidor y sólo podrá ser borrado por el que lo recibe. Para recoger la fecha se utiliza las líneas de código descritas más abajo.

El método “date” del objeto NSDate, ofrece la fecha en la siguiente forma: “2014-03-10 09:54:03 +0000”. Cómo los últimos cinco dígitos no interesan, al string de la fecha se le quita los últimos seis caracteres con el método “substringToIndex”. Así, la fecha quedará acortada simplemente con la fecha y hora actual.

```
NSDate *ahora= [NSDate date];
NSString *fecha= [NSString stringWithFormat:@"%@",ahora];
fecha=[fecha substringToIndex:[fecha length]-6];
```

El registro en la BD externa se viene haciendo como hasta ahora.

7.10. Navigation Controller

Un Navigation Controller gestiona una pila de ViewControllers. Esto es, es el encargado de gestionar y coordinar las vistas de la aplicación. Es quien almacenará toda la jerarquía de vistas que haya en la aplicación y será el encargado de mostrarlas cuando sea necesario, pasando de un nivel a otro en dicha jerarquía.

En la aplicación, ir de una vista a otra es producido al pulsar un botón o una celda. Este botón o celda, estará *conectado* con la vista a la que se accederá. Esta conexión tendrá un identificador con el nombre que uno quiera, y será usado en un método con el fin de identificar a que vista se accederá.

Para ello, se utilizará el código descrito más abajo. Cuando se pulsa un botón que está ligado a otra vista, se llama a ese método. El ejemplo que se va a mostrar, es cuando el usuario pulsa en la celda de comentarios en la ficha de un lugar, para así consultar los comentarios que haya sobre ese lugar.

Así pues, todo lo que se vaya a querer pasar a la otra vista se hará con el método *prepareForSegue*. A este método, se le pasará el objeto “*segue*” que contendrá información acerca de los ViewControllers implicados en la transición. Se le llama a este método antes de que ocurra la transición, para

que se pueda pasar cualquier dato necesario para el ViewController que está a punto de mostrarse.

En este ejemplo, en primer lugar, se tendrá que recoger qué celda ha sido pulsada. Si fuera un botón, no habría que recoger esta información, simplemente se comprobaría el identificador. Para recoger el número de celda, haremos uso del indexPath de la tabla, que tendrá guardado la sección y la fila que se ha pulsado.

A continuación, se comprueba el nombre del identificador que se le ha puesto a la conexión entre la celda y la vista de los comentarios. Se tiene que comprobar con varios "if" o con un "switch", ya que puede haber otros botones y otras celdas que al ser pulsados vayan a otra vista diferente. Así, siempre se sabrá con certeza la vista a la que se accede pulsando la celda o el botón.

Para pasar la información de una vista a otra, se hará uso del método "*destinationViewController*" que es un método del objeto "*segue*". De esta manera, se pasa toda la información que se quiera a las variables de la nueva vista que se va a mostrar.

A continuación se muestra el código para realizar la transición entre dos vistas:

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    NSArray *indexPaths = [self.tableView indexPathsForSelectedRows];
    NSIndexPath *indexPath= [indexPaths objectAtIndex:0];
    [self.tableView deselectRowAtIndexPath:indexPath animated:NO];
    if ([segue.identifier isEqualToString:@"descripcion"] ) {
       (...)
    }
    if ([segue.identifier isEqualToString:@"fotos"] ) {
       (...)
    }
    if ([segue.identifier isEqualToString:@"videos"] ) {
       (...)
    }
    if ([segue.identifier isEqualToString:@"mapa"] ) {
        (...)
    }
    if ([segue.identifier isEqualToString:@"comentarios"] ) {

        [segue.destinationViewController setTitle:@"Comentarios"];
        [segue.destinationViewController setIdLugar:lugar.id_lugar];

    }
}
```

8. Verificación y evaluación

Cada vez que se ha acabado de implementar un prototipo, se ha hecho su prueba correspondiente. En este apartado se describirá las pruebas llevadas a cabo. Para ello, durante el desarrollo de la aplicación, así como en su finalización, las pruebas han sido realizadas con el simulador de Xcode y con un IPhone 4.

Exceptuando algunos casos, y ante el desconocimiento previo del lenguaje utilizado, se han ido comprobando los diferentes métodos al tiempo que se modificaban, con el fin de estar seguro que hacían lo que se esperaba. Esto ha causado que fuera más costoso en el tiempo, pero más eficaz a la hora de ir corrigiendo errores o buscar nuevas soluciones a los problemas que iban surgiendo. Además, en algunos casos ha sido muy instructivo y una buena forma de aprender este nuevo lenguaje.

A continuación, se explica cómo han sido las pruebas, definiendo los problemas que se han ido encontrando durante el desarrollo de la aplicación, y su resolución. Se mostrarán las más relevantes e importantes que se han dado, ya que en muchos casos, los problemas que ocurrían eran por problemas de desconocimiento que se solucionaban al conocer más a fondo las complicaciones. Se irán referenciando a grandes rasgos los diferentes apartados de la aplicación.

8.1. Bases de datos

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-BD-1	Hacer consultas en las tablas de ambas BD	Resultado lógico entre la sentencia y el resultado	Se ha mostrado la información pedida	Éxito
Pru-BD-2	Insertar información mediante sentencia en ambas BD	Guardar la información introducida en la tabla pedida	Se ha guardado correctamente la información en la tabla indicada	Éxito
Pru-BD-3	Borrar información mediante sentencia en ambas BD	Borrar las tuplas ya sea en cascada o no	En la BD del servidor se han borrado las tuplas. En la BD local no borra en cascada	**Error** No borra en cascada en la BD local
Pru-BD-4	Actualizar información mediante sentencia	Actualizar las tuplas mediante el criterio introducido	Se ha actualizado la información en las tuplas indicadas	Éxito

Tabla 10.- Pruebas de las bases de datos

Hubo un problema en la base de datos local cuando se borraba la información en cascada. Esto era problema del programa utilizado para hacer la base de datos, que había que habilitar manualmente las claves foráneas. En uno de los menús del programa, se pulsa en Open On-connect SQL Tab y se introduce la siguiente frase**:** PRAGMA foreign_keys=ON, para activar las claves foráneas y por lo tanto poder borrar en cascada.

Sin embargo, esto no fue suficiente a la hora de borrar en cascada desde la aplicación ya que no lo hace. La solución se da en el siguiente apartado.

8.2. Ciudad

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-Ci-1	Cargar base de datos local	Si la base de datos no ha sido creada, se crea. Si no, se habilita	Base de datos cargada correctamente	Éxito
Pru-Ci-2	Mostrar las ciudades descargadas	Mostrar las ciudades con sus imágenes en la tabla	Se han mostrado las ciudades en las tablas con sus imágenes	Éxito
Pru-Ci-3	Mostrar las ciudades de la base de datos del servidor	Mostrar las ciudades en la tabla, mostrando las fotos fluidamente	Muestra las ciudades, pero la carga de las fotos provoca que no haya fluidez en la tabla	**Bug** La carga de imágenes hace que no haya fluidez al subir o bajar la tabla
Pru-Ci-4	Mostrar buscador y buscar ciudades	Se muestra en la tabla la búsqueda realizada	Se ha mostrado las ciudades mediante el criterio introducido por el usuario	Éxito
Pru-Ci-5	Al pulsar en los lugares de las ciudades, mostrar los lugares descargados de esa ciudad	Aparecer los lugares descargados en las tablas con sus imágenes correspondientes	Se ha mostrado los lugares en la tabla con sus imágenes	Éxito

Tabla 11.- Pruebas de las ciudades 1

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-Ci-6	Borrar una ciudad descargada	Se borra la ciudad de la base de datos local y su imagen correspondiente del dispositivo. Y además, todo lo descargado de esa ciudad quedará borrado	Se ha borrado la ciudad y su imagen, pero no todo lo relacionado con esa ciudad (fotos, vídeos, lugares o rutas)	**Bug** No se ha borrado todo lo relacionado con la ciudad
Pru-Ci-7	Recargar la tabla de ciudades	Al estirar hacia abajo la tabla, se actualiza accediendo de nuevo a la BD de datos	Se ha mostrado las ciudades actualizadas en la tabla	Éxito
Pru-Ci-8	Subir foto del metro	Se sube la imagen al servidor y se actualiza la ciudad en la BD del servidor	Se ha subido la imagen y se ha actualizado la ciudad en la BD	Éxito
Pru-Ci-9	Descargar metro	Se guarda la imagen en el dispositivo. Si la ciudad está descargada en la BD local, se actualiza, si no se crea un nuevo registro	Se ha guardado la foto en el dispositivo y se ha actualizado la ciudad. Si no, se ha creado el registro	Éxito
Pru-Ci-10	Crear guía completa	Se inserta la información de la ciudad y se guarda en la BD del servidor	Se ha guardado la información de la ciudad en el servidor	**Bug**. Sin subir lugares, fotos, vídeos, rutas o el mapa del metro

Tabla 12.- Prueba de las ciudades 2

En primer lugar se creó una tabla para que apareciera el nombre de las ciudades y parte de su información, sin foto de perfil. Con el fin de que quedara más visual para el usuario, se decidió añadir más adelante la foto de perfil de la ciudad. Sin embargo, la aplicación se ralentizaba al mandar la tabla hacia arriba porque tenía que acceder constantemente a las imágenes de cada ciudad y "renderizarlas" en la pantalla (véase, Pru-Ci-3). Se consiguió mayor fluidez creando el procedimiento que cargaba las imágenes en segundo plano, lo cual sirvió para que cargara y mostrara las imágenes de las ciudades donde la tabla estaba detenida. Esto no llegó a ser suficiente y se optimizó con la clase SDWebImage y que permitía cargar de manera óptima y sencilla las fotos desde el servidor.

Para el refresco de la información en la tabla de las ciudades se utilizó en un principio un botón en la barra de navegación. Sin embargo, se acabó utilizando el objeto que se usa para esos casos y que la tabla se actualizase automáticamente al arrastrar la tabla hacia abajo una vez que ésta está arriba del todo. El resultado fue positivo, más elegante y más intuitivo para el usuario.

En cuanto a la barra de búsqueda se tuvo problemas para que apareciera y desapareciera ya que se movía todo constantemente y no quedaba en el sitio que se deseaba. Por lo tanto, se optó por ocultarla guardando las coordenadas de su posición y al pulsar el botón de la lupa hacer que bajara hasta la posición donde se encontraba las pestañas para cambiar entre las guías descargadas, disponibles y más votadas, haciendo que se situara encima y lo tapara. Al final quedó como una pequeña animación cuando la barra de búsqueda aparecía, obteniendo un buen resultado.

A la hora de borrar las ciudades descargadas, se ha tenido problemas al borrar en cascada en la base de datos (véase, Pru-Ci-6). La información relacionada (ya sean lugares, fotos, vídeos y rutas) con la ciudad, no se borraba automáticamente y se desconoce el porqué. De esta manera, se ha arreglado consultando la información y borrándola una a una. Esto mismo ocurrió con los lugares.

La creación de una guía completa (véase, Pru-Ci-10) se acabó completamente más adelante. No obstante se fue probando en primer lugar que la información de la ciudad que se iba a crear se guardara en la base de datos externa. Debido a que en una guía completa se pueden crear lugares y rutas, así como subir fotos, vídeos y el mapa del metro, se fue añadiendo a esta funcionalidad a medida que se iba realizando en su debido apartado. Así, al final, la creación de una guía completa quedaba garantizada y probada.

8.3. Lugar

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-L-1	Mostrar los lugares de la base de datos del servidor	Mostrar los lugares en la tabla, cargando las imágenes correspondientes de manera fluida	Se ha mostrado los lugares, pero la carga de las imágenes no es fluida	**Bug.** La carga de imágenes hace que no haya fluidez al subir o bajar la tabla
Pru-L-2	El usuario añade un lugar nuevo	Guardar la información introducida por el usuario y subir la imagen, si la hubiera, al servidor	Se ha guardado la información y la imagen en el servidor	Éxito

Tabla 13.- Pruebas de los lugares 1

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-L-3	Ver la descripción de un lugar y oírlo por el altavoz al pulsar el botón	Mostrar el texto de la descripción y reproducirlo mediante sonido y así mismo pausarlo	El texto se ha mostrado y se ha reproducido y pausado	Éxito
Pru-L-4	Borrar lugar descargado	Borrar la información del lugar y su imagen, y además borrar las fotos y vídeos que el usuario haya descargado sobre esa ciudad.	Se ha borrado el lugar y su imagen y todas las fotos y vídeos relacionados	Éxito
Pru-L-5	Borrar lugar que un usuario haya creado	Borrar el lugar y su imagen de la base de datos del servidor. Si hay otro usuario compartiendo, aparecerá una alerta	Se ha borrado el lugar del usuario y su imagen. En lugares que otros usuarios comparten algún recurso, no se ha permitido borrar	Éxito
Pru-L-6	Mostrar lugar en el mapa	Indicar en el mapa el lugar elegid	Algunos lugares se muestran y otros no	**Bug** Hay direcciones que el mapa no encuentra. Además se muestra más de un indicación en el mapa
Pru-L-7	Calcular una ruta entre el usuario y el lugar señalado en el mapa y mostrar la distancia	Mostrar la línea entre el usuario y el lugar marcado y la distancia entre ambos	Se ha creado la línea y mostrado la distancia	Éxito
Pru-L-8	Mostrar comentarios de un lugar	Mostrar los comentarios ordenador de más recientes a más antiguos	Se ha mostrado los comentarios de más recientes a más antiguos	Éxito

Tabla 14.- Prueba de los lugares 2

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-L-9	Escribir comentario	Se guarda la información del comentario en la BD del servidor en la fecha que se ha escrito	No se guarda parte de la información debido a la codificación del texto	**Bug** Los textos con caracteres especiales no se registran
Pru-L-10	Recargar la tabla de lugares	Al estirar hacia abajo la tabla, se actualiza accediendo de nuevo a la BD de datos	Se ha mostrado las ciudades actualizadas en la tabla	Éxito

Tabla 15.- Prueba de los lugares 3

Al igual que con las ciudades, en la tabla de los lugares se utilizaron las mismas pruebas y los mismos cambios, como la carga de la foto de perfil (véase, Pru-L-1) y el refresco de la tabla y la barra de búsqueda.

Para mostrar el lugar en el mapa (véase, Pru-L-6), se hizo de forma que al meter el nombre del lugar y de su ciudad, se buscara en el mapa y lo marcara con un pin. Sin embargo, marcaba en el mapa más de un lugar diferente (aunque cercanos) que tenían relación con el nombre del lugar o de la ciudad. Por lo tanto, se optó recoger el primer resultado de la búsqueda y que sólo marcara ese, ya que el primero que la búsqueda guardaba era el lugar que verdaderamente interesaba.

No obstante, había lugares que no se mostraban en el mapa debido a que, en algunos casos, la manera de escribir un lugar en inglés o en castellano es diferente y en el listado de direcciones que tienen los mapas de Apple, no había coincidencias cuando se buscaba. Por lo tanto, se optó que en estos casos se mostrara un aviso para que el usuario probara introduciendo él mismo el nombre del lugar en otro idioma o de otra forma y así poder marcarlo en el mapa.

Para poder crear la ruta entre la posición del usuario y la que se quiere visitar, primero se probó en el simulador. Sin embargo, el simulador no puede localizar la posición del usuario con la geolocalización, ya que hay que meter las coordenadas manualmente. Se fue probando de esta manera, hasta que se tuvo acceso a probar la aplicación en el móvil gracias a la licencia de desarrollador de Apple y se ajustó para que cogiera las coordenadas de la posición del usuario con el GPS.

Al escribir un comentario y guardar la información en la base de datos, hubo problemas relacionados con la codificación del texto. Al principio, en el JSON no aparecía nada, excepto números. El problema venía de las “ñ”-s y las tildes, es decir, por los caracteres especiales. Se cambió a “UTF-8 General” el tipo de texto que se guardaba en la base de datos y el problema se solucionó.

Ocurrió algo parecido con los textos que tenían espacios. Al mandar un texto de estas características a PHP no lo reconocía, así que para solucionar este problema, se cambiaron los espacios por "$$" y al recibir la información en el PHP, darle la vuelta al texto cambiando los "$$" por espacio. Así, el texto se almacena correctamente en la base de datos. Esto fue valido para todos los textos de la aplicación.

8.4. Recurso

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-Re-1	Consultar fotos o vídeos de una ciudad o lugar	Mostrar en la tabla las fotos o vídeos de la ciudad o de un lugar	Se ha mostrado las fotos o vídeos en la tabla.	Éxito
Pru-Re-2	Borrar fotos o vídeos descargados	Borrar las fotos o vídeos seleccionados del dispositivo y su información de la BD local	No ha dejado seleccionar ninguna foto o vídeo	**Bug**. Al pulsar sobre una foto o vídeo no queda seleccionada y accede a una nueva vista
Pru-Re-3	Ver una foto para acercarla y alejarla	Mostrar la foto y permitir hacer zoom	Se ha mostrado la foto y permite acercar y alejar	Éxito
Pru-Re-4	Descargar una foto o vídeo	Descargar la foto o el vídeo al dispositivo y guardar la información en la BD local	Se ha descargado la imagen o el vídeo y se ha guardado su información	Éxito
Pru-Re-5	Compartir fotos en Facebook	Seleccionar las fotos y y al pulsar en compartir, publicarlas en Facebook	Se han publicado las fotos en Facebook	Éxito
Pru-Re-6	Subir una foto o vídeo	Seleccionar una foto o vídeo y subirla al servidor y registrar la información en la BD	Se ha guardado la información (incompleta) en la BD, pero no se ha subido el archivo	**Bug**. Al subir la foto o vídeo al servidor, no creaba el archivo por falta de permisos

Tabla 16.- Prueba de los recursos 1

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-Re-7	Borrar foto o vídeo que ha subido un usuario	Se borra la foto o vídeo del servidor y la información de la BD del servidor	Se ha borrado el archivo y la información	Éxito
Pru-Re-8	Reproducir vídeo	Al pulsar en "ver" se accede al vídeo descargado o al vídeo en el servidor y se reproduce en una nueva vista	El vídeo ha sido reproducido	Éxito

Tabla 17.- Prueba de los recursos 2

Con las fotos y los vídeos se hizo pruebas para visualizarlas, descargarlas y subirlas. Hubo cambios con las descargas ya que en un principio no se sabía en dónde se debía guardar los ficheros para descargar.

Se llegó a la conclusión de que se debía guardar en la carpeta *Documents Directory*, que es el directorio para guardar este tipo de ficheros que se descargan en la aplicación. La otra alternativa era *Caché Directory,* pero el problema con esta carpeta es que con el tiempo, los ficheros en su interior, se borran. Es decir, sólo guarda la información durante un periodo de tiempo, y además a la hora de hacer copias de seguridad, esta carpeta no se contempla.

A la hora de seleccionar más de una foto o vídeo a la vez (véase, Pru-Re-2), se tuvo problemas ya que al pulsar sobre una foto accedía a esta foto y por lo tanto a un cambio de vista. El problema se solucionó con un método de la tabla (en este caso la tabla es del tipo UICollectionView), que permitía seleccionar pulsando sobre las fotos sin salir de la vista actual.

Para subir las fotos (véase, Pru-Re-6), en un principio se hacía en local, hasta que se tuvo el servidor. Con el cambio, se tuvo un pequeño problema de permisos en las carpetas del servidor para crear los archivos en su interior, pero se solucionó dando los permisos pertinentes.

8.5. Ruta

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-Ru-1	Consultar rutas descargadas	Se recoge la información de la BD local y es mostrada en la tabla	Se ha mostrado las rutas descargadas en la tabla	Éxito
Pru-Ru-2	Consultar rutas del servidor	Se recoge la información de la BD y se muestra en la tabla	Se ha mostrado las rutas ordenadas por días	Éxito
Pru-Ru-3	Descargar ruta	Se guarda la información de la ruta en la BD local. Si la ciudad está descargada, se actualiza la cantidad de rutas descargadas. Si no está descargada, se crea un registro de la ciudad	Se ha guardado la información de la ruta y se ha actualizado o creado un registro de la ciudad	Éxito
Pru-Ru-4	Crear ruta	Tras insertar la información se pulsa el botón y se crea un nuevo registro en la BD del servidor	Se ha creado el registro en la BD externa	Éxito
Pru-Ru-5	Recargar rutas	Al estirar hacia abajo la tabla, se actualiza accediendo de nuevo a la BD de datos	Se ha mostrado las ciudades actualizadas en la tabla	Éxito
Pru-Ru-6	Borrar ruta creado por el usuario	Borrar la ruta de la base de datos del servidor	Se ha borrado la ruta del usuario de la BD del servidor	Éxito

Tabla 18.- Pruebas de las rutas

Con las rutas no se ha tenido muchos problemas, ya que no tiene fotos con las que tratar y cuando se implementó ya se había corregido problemas como por ejemplo la codificación del texto en la BD del servidor.

8.6. Usuario

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-U-1	Registrarse	Se guarda la información en la BD del servidor	Se ha guardado la información en la BD	Éxito
Pru-U-2	Iniciar sesión	El usuario introduce el usuario y contraseña, y si es correcto, se accede a su cuenta. Además se guardan sus credenciales	El usuario ha introducido correctamente los datos y se ha recogido su información. Sus credenciales han sido guardadas	Éxito
Pru-U-3	Consultar amistades	Se muestran las amistades(los que se sigue, los seguidores y los pendientes) en la tabla	Se han mostrado todas las amistades del usuario	Éxito
Pru-U-4	Seguir, dejar de seguir y hacer una petición de amistad	Se pulsa el botón y se crea la amistad en la BD. Si acepta una petición, se cambia el estado de la amistad en la BD	Se ha cambiado el estado del botón y se ha registrado en la BD	Éxito
Pru-U-5	Buscar usuarios	Se muestran las sugerencias. Al pulsar en el barra de búsqueda e introducir el usuario, se comprueba en la BD y se muestra en la tabla si existe	Se han mostrado las sugerencias. El usuario buscado se muestra en la tabla.	Éxito
Pru-U-6	Ver perfil de usuario	Se pulsa sobre un usuario y se accede a su perfil. Si el perfil está bloqueado, no deja acceder a su información si el usuario no le sigue	Se muestra el perfil del usuario elegido. Al estar bloqueado, no deja acceder a su información	Éxito

Tabla 19.- Pruebas de los usuarios 1

ID	Descripción	Resultado esperado	Resultado obtenido	Observaciones
Pru-U-7	Mandar mensaje	Se rellenan el destinatario, el asunto y el cuerpo del mensaje y se envía. Se guarda la información en la BD del servidor	Se ha guardado el mensaje en la BD del servidor	Éxito
Pru-U-8	Borrar Mensaje	El usuario borra el mensaje y borra la información de la BD	El usuario ha borrado el mensaje y borrado de la BD. El usuario que ha enviado el mensaje también lo puede borrar	**Bug** El usuario que ha enviado borra el mensaje, también lo borra del buzón del que lo recibe
Pru-U-9	Actualizar información de usuario	Se ha modificado los campos y tras pulsar el botón de “Hecho” se actualiza la información en la BD.	Se ha modificado la información del usuario en la BD	Éxito
Pru-U-10	Puntuar o quitar puntuación	Al pulsar el botón para puntuar, si no se había puntuado previamente se crear la puntuación en la BD, sino, se borra	Se ha guardado la puntuación del usuario. Si ya se había puntuado, se borra de la BD	Éxito.
Pru-U-11	Consultar puntuaciones	Se muestra en la tabla las puntuaciones del usuario separadas en lugares, rutas, fotos y vídeos	Se han mostrado en la tabla todas las puntuaciones que ha hecho el usuario	Éxito
Pru-U-12	Consultar guías creadas	Se muestra en la tabla las guías creadas por el usuario, separadas por cada tipo.	Se muestra el nombre de las guías, pero las imágenes no corresponden con los objetos	**Bug**. Las fotos de las guías no se correspondían con ellas

Tabla 20.- Prueba de los usuario 2

A la hora de que un usuario iniciara sesión, hubo que decidir cómo podía aparecer la vista para iniciar sesión. En un principio, se iba a utilizar dicha vista para que cuando el usuario quisiera crear alguna guía o subir algún recurso, tuviera que iniciar sesión. Sin embargo, se creyó más conveniente tener que hacerlo una sola vez desde el apartado de la cuenta y en caso contrario, no permitir crear ninguna guía. Por lo tanto, se comprobó que al iniciar sesión, dejara crear cualquier guía o subir recursos satisfactoriamente.

Cuando un usuario manda un mensaje a otro, se almacena en la base de datos, pero cuando uno de los dos, (el emisor o el receptor) borraba dicho mensaje (véase, Pru-U-8), al otro usuario también le desaparecía el mensaje. Con el fin de que no hubiera problemas de este tipo, se decidió que el usuario que enviaba los mensajes, una vez almacenado en la base de datos, sólo lo pudiera borrar el que lo recibe, por lo que el mensaje quedaría guardado por el usuario receptor hasta que decidiera borrarlo.

Al consultar las guías que el usuario ha creado, las fotos no correspondían con las guías. Esto se solucionó haciendo que cada vez que fuera a poner la foto en la celda, pusiera a null la imagen y luego ponerla. El motivo es que al pasar rápidamente la tabla hacia arriba o hacia abajo, cargue la foto en la celda equivocada. Además, se hace uso de SDWebImage para mejorar la fluidez de la tabla.

Finalmente, los resultados obtenidos de las pruebas realizadas y su posterior resolución llevan a concluir que se ha alcanzado el correcto funcionamiento de la aplicación, cubriendo todas las funcionalidades planteadas en los casos de uso. Se consigue por ello el objetivo de ofrecer una solución al problema planteado al inicio del documento.

9. Conclusiones y líneas futuras

9.1. Conclusión Personal

En este apartado me dispongo a expresar a nivel personal lo que ha supuesto este TFG para mi formación académica y profesional.

El desarrollo del citado proyecto no solo tiene como objetivo proporcionar una solución a un problema planteado o la finalización de los estudios. Se pretendía también que el proyecto llevado a cabo me proporcionase conocimientos sobre Objective-C y las competencias necesarias para elaborar aplicaciones para los dispositivos de Apple. Considero que dicho objetivo ha sido cumplido, no sólo he aprendido Objective-C y a utilizar el SDK Xcode, he aprendido a llevar a cabo el desarrollo completo de un proyecto, y sobre todo y lo más importante, a afrontar todas aquellas adversidades que pueden surgir en dicho proceso.

La mayor complicación que he tenido en el desarrollo de esta aplicación, ha sido el tener que empezar de cero con un lenguaje diferente y nuevo, que va cambiando gradualmente y añadiendo nuevas características.

En un principio, ante esta situación de desconocimiento, pensé en hacerlo de la manera más sencilla posible, sin embargo, con la ayuda de foros e información en internet, fui consciente de las posibilidades que tenía para hacer una aplicación grande y que pudiera ser más productiva para un hipotético futuro usuario. Esto ha derivado en querer aprender todo lo posible y retarme a mí mismo para conseguir llevar a cabo los objetivos planteados al principio e incluso, como ha ocurrido, a añadir algunas características extras.

A pesar de haberme llevado más tiempo del estipulado, no he desaprovechado la oportunidad de seguir aprendiendo e intentar expandir aún más el proyecto con funcionalidades nuevas e interesantes que me permitirían seguir conociendo más a fondo Objective-C.

Es de entender que aún me queda mucho por aprender y mejorar muchos de los conocimientos adquiridos, pero creo haber conseguido una férrea experiencia y así poder salir al mercado laboral habiendo aprendido algo adicional que pueda complementar lo cursado en el Grado.

Por otro lado, la mala estimación de tiempos, modificaciones en las funcionalidades de la aplicación, imposibilidad de seguir haciendo el proyecto en fechas señaladas por viajes y por la consiguiente ausencia del ordenador y sobre todo, por los problemas a la hora de implementar debido a la inexperiencia sobre el lenguaje, ha provocado el retraso para la fecha prevista de entrega. Al ser un proyecto tan grande, el tiempo invertido ha sido mucho mayor de lo esperado y, por consiguiente, el haber llevado a entregar el proyecto en una fecha tan tardía.

No obstante, y ahora sabiendo lo que sé, podría hacer una mejor estimación de los tiempos y un trabajo mucho más fluido acabando antes la aplicación. Ahora

sé cómo puedo enfrentarme a un proyecto de estas características y el poder afrontar un proyecto grande con más seguridad.

Por último, comentar mi satisfacción sobre todo lo realizado. Ha sido todo un reto, como ya he mencionado, pero sobre todo me ha servido para animarme a seguir aprendiendo otros lenguajes, aunque sea de forma autodidacta y así seguir ampliando mis conocimientos para poder hacer aplicaciones que puedan ser usadas por muchos más usuarios. Me ha servido para ver que soy capaz de conseguirlo, si se tiene insistencia y ganas. El resultado, al final, de poder utilizarlo en el dispositivo me ha resultado muy gratificante.

9.2. Conclusión en la gestión

A lo largo del desarrollo de este proyecto se han encontrado diferentes dificultades que han influido en ocasiones al diseño inicial así como en la implementación.

A continuación se muestra un gráfico que muestra la estimación de tiempos realizada y la real:

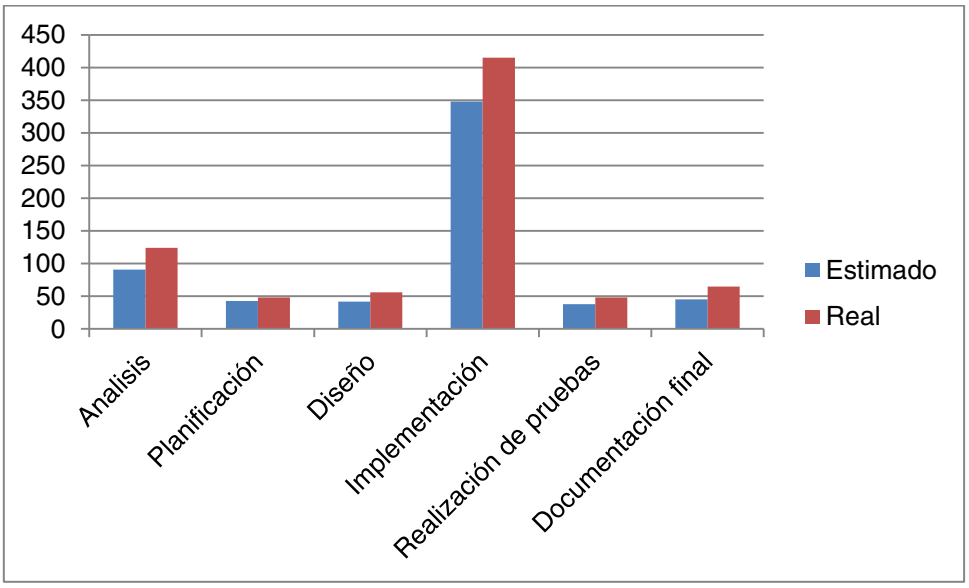


Ilustración 37.- Gráfico de tiempos

Como se puede apreciar en el gráfico, donde peor se ha estimado el tiempo ha sido en el análisis y en la implementación. Esto es muy significativo, ya que el gasto de tiempo real y que por consiguiente ha provocado que no se entregara el proyecto en abril ha sido a causa del desconocimiento del lenguaje. He tenido que aprender durante más tiempo de lo estipulado y por consiguiente acarreando una carga de tiempo a la implementación de la aplicación. Además, ha habido cambios constantes en la interfaz, que aunque pequeños, ha supuesto un aumento en el tiempo. Por ejemplo, con la creación de iconos y el acabado de las vistas.

Además de esto, se ha tenido que implementar los archivos PHP y las consultas a las bases de datos. En el grado, los conceptos sobre PHP y JSON

son escasos e impartidos de forma ligera. Así, el código para implementar en los PHP me ha ocasionado un gasto de tiempo adicional ya que no conocía cómo Xcode se conectaba con PHP y éste a su vez con la base de datos.

Todas las dificultades han supuesto un retraso respecto a la planificación inicial, junto con el respectivo aumento de trabajo que se estimaba inicialmente.

9.3. Líneas futuras

A pesar de que probablemente no se continúe con la aplicación, hay mejoras que podrían haberse llevado a cabo con más tiempo o siendo un proyecto real. La primera se ha mencionado en el apartado 3.6 en la evaluación económica, y es intentar hacer la aplicación gratuita mediante la publicidad de establecimientos y con el servicio iAd. Además, para que las conversaciones entre usuarios sean más efectivas y familiares, sería conveniente cambiar el modo de enviar mensajes como si fuera un chat y que al llegar un mensaje se enviara una notificación. El trabajo que lleva esto último es alto y por eso no se ha introducido en esta primera versión.

A continuación se presentan otras mejoras que podrían realizarse en el presente proyecto:

-Ampliar apartados: Este proyecto está pensado para ampliarlo en cualquier momento, gracias a la barra de abajo (Tab Bar) donde se pueden meter todos los apartados de la aplicación que se quiera, y por esta razón el trabajo que supondría hacer cada apartado nuevo dependerá del tamaño que tenga o de lo que se quiera hacer. Entre ellos, se podría destacar los siguientes puntos:

- **Cerca de mí:** Esta opción sería adecuada para que el usuario pudiera ver lo que hay a su alrededor mediante la geolocalización.
- **Herramientas de viaje:** Serían herramientas que pudieran servir para viajar, como por ejemplo, el tiempo que va a hacer, la longitud y latitud de un lugar concreto, conversor de divisas, etc.
- **Compartir viaje:** Si un usuario quiere ir a Madrid desde Bilbao, pueda anunciarse para tal fin y compartir coche o vehículo para reducir gastos de gasolina y que resulte más económico.
- **Ofrecerse como guía:** Muchas personas conocen bien sus ciudades o pueblos y pueden ofrecer conocimientos, actividades o establecer un recorrido para hacer de guía al precio que ellos consideren o incluso gratuitamente.
- **Dónde comer y alojarse:** Recomendar restaurantes y hoteles.

-Obtención de ingresos: Con la idea del último apartado del punto anterior, se podría añadir un sistema de ingresos en el cual los dueños de los establecimientos y restaurantes puedan promocionarse pagando una cantidad con el fin de poder publicitarse. Esto supondría controlar los pagos y aumentar la seguridad para las transferencias.

A parte de estas nuevas funcionalidades, se podría expandir la aplicación para iPads y además, añadir la opción de multilenguaje, para así internacionalizar la

aplicación. Por último, a nivel estético, podría mejorarse la interfaz para hacerla más agradable y atractiva, además de organizar mejor las guías para su consulta.

10. Bibliografía

Libros

-Lewis, R. (2010). *Desarrollo de aplicaciones para iPhone e iPad para principiantes*. Nueva York: Apress. (Versión traducida).

-Mark, David. Nutting, Jack. LaMarche, Jeff. Olsson, Fredrik. (2010). *Beginning iOS 6 Development – Exploring the iOS SDK* –. Nueva York: Apress.

-Tamarit, Cecilio C. (2014). *Desarrollar Apps para iOS7 es fenomenal!.* Valencia: Cecetaca.

Youtube

-iPadEsfera. (2011). *Aprende a programar para iOS*. http://www.youtube.com/user/ipadsfera/videos

-Códigofacilito. (2012). *Curso de Xcode*. http://www.youtube.com/playlist?list=PLB56C1606C990DE61

-Code With Chris. (2013). *Learn Objective-C Tutorial*. http://www.youtube.com/user/CodeWithChris/videos

-David Román. (2012). *Programación iOS*. http://www.youtube.com/user/davidromanaguirre/videos

-Videos Manzana Mágica. (2012-2014). *Tutoriales iOS*. http://www.youtube.com/user/videosmanzanamagica/videos

-Pecs Xcode. (2012). *PecsXcode UICollectionView Delete Cell.* http://www.youtube.com/watch?v=CZ1Vw5OnmTU

-Appdesignvault. (2011). *iPhone App Design Tutorial – Adding Webservice Data To Your App*. http://www.youtube.com/watch?v=M7butieUoBg

Páginas WEB

-Apple Inc. (2010). *Apple Presents iPhone 4.* http://www.apple.com/pr/library/2010/06/07Apple-Presents-iPhone-4.html

-Avelino Felipe Policarpio. (2012). *Modelo incremental*. http://www.slideshare.net/AvelinoFelipePolicarpio/modelo-incremental-12729581

-Apple. (2013). *iOS Developer Library*. https://developer.apple.com/library/ios/navigation

-ITIOX. (2013). *Tutoriales para iPhone*. http://blog.itiox.com

-Apprendemos. (2012). *Obtener coordenadas del GPS del iPhone o iPad*. http://www.apprendemos.com/tutoriales/ios/obtener-coordenadas-gps-iphone-ipad

-Razeware LLC. (2014). *iPhone Tutorials*. http://www.raywenderlich.com/tutorials

-Stack Exchange inc. (2014). *Preguntas del foro*. http://stackoverflow.com

-NSCodeCenter.com. (2010-2011). *Preguntas del foro*. http://www.nscodecenter.com

-AppCoda. (2012). *Tutorials*. http://www.appcoda.com/tutorials/

-Luis Cardenas. (2014). *Parsear y crear ficheros en formato JSON en iOS*. http://www.thxou.com/2012/09/11/parsear-y-crear-ficheros-en-formato-json-en-ios

-Jorgelig. (2011). *Suma de dos tablas con posible valor null*. Cristalab. http://foros.cristalab.com/suma-de-dos-tablas-con-posible-valor-null-t67142

-The PHP Group. (2001-2014). *Manual de PHP*. http://es1.php.net/manual/es/

-NSCookBook.com. (2013). *iOS Programming Recipe 7: Using the UIPickerView*. http://nscookbook.com/2013/01/ios-programming-recipe-7-using-the-uipickerview/

-Dipin Krishna. (2014). *iOS Login screen tutorial with server authentication – JSON – PHP*. http://dipinkrishna.com/blog/2012/03/iphoneios-programming-login-screen-post-data-url-parses-json-response/

-DevelopPHP.com. (2013). *Delete a file from the server*. http://www.developphp.com/view_lesson.php?v=453

-World Tourism Organization Network. (2014). *Turismo sostenible*. http://sdt.unwto.org/es/content/definicion

-Marta Lorenzo. (2012). *En 2030 habrá 1.800 millones de turistas viajando por el mundo*. ZINE Consultores. http://www.tendencias21.net/En-2030-habra-1-800-millones-de-turistas-viajando-por-el-mundo_a13038.html

-Louesfera. (2014). *Tribuna libre de Francisco Jose Gallardo sobre el precio de las apps*. http://www.louesfera.com/2013/07/19/tribuna-libre-francisco-jose-gallardo-precio-apps/

Imágenes

-Ilustración 1 [fotografía]. Recuperado de https://sites.google.com/site/tecnologiaiostm/desarrollo-de-aplicaciones/arquitectura-ios
-Ilustración 4 [fotografía]. Modificado de https://developer.apple.com/library/ios/featuredarticles/viewcontrollerpgforiphoneos/AboutViewControllers/AboutViewControllers.html#//apple_ref/doc/uid/TP40007457-CH112-SW10

-Ilustración 5 [fotografía]. Recuperado de https://developer.apple.com/library/ios/documentation/general/conceptual/devpedia-cocoacore/MVC.html

Páginas para información de la BD:

-Civitatis. (2014). *Disfruta Roma – Tu guía de Roma*. http://www.disfrutaroma.com/

-Civitatis. (2014). *Disfruta Tokio – Tu guía de Tokio*. http://www.disfrutatokio.com

-Civitatis. (2014). *Londres – Guía turística*. http://www.londres.es

-Civitatis. (2014). *Disfruta Berlín – Tu guía de Berlín*. http://www.disfrutaberlin.com

-Civitatis. (2014). *Nueva York – Tu guía de Nueva York*. http://www.nuevayork.net

-Civitatis. (2014). *París – Guía turística*. http://www.paris.es

-Civitatis. (2014). *Conocer Barcelona*. http://www.conocerbarcelona.com

-Civitatis. (2014). *Edimburgo – Tu guía de Edimburgo*. http://www.edimburgo.com

-Civitatis. (2014). *Disfruta Madrid*. http://www.disfrutamadrid.com

-Civitatis. (2014). *Disfruta Florencia – Tu guía de Florencia*. http://www.disfrutaflorencia.com/

-Civitatis. (2014). *Disfruta Praga*. http://www.disfrutapraga.com

-Civitatis. (2014). *Disfruta Praga*. http://www.disfrutaamsterdam.com

ANEXO I: CASOS DE USO EXTENDIDOS

Nombre: Registrarse.
Descripción: Permite al usuario darse de alta en el sistema.
Actores: No registrado.
Precondiciones: Ninguna.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsará el botón de "Registrarse", entonces rellenará los campos de "Usuario", "contraseña", "correo", "fecha de nacimiento", "sexo" y subirá una foto de perfil si el usuario lo desea. Una vez hecho esto, pulsará "Finalizar". Si no se han rellenado todos los formularios, aparecerá una advertencia. La foto no es obligatoria.
2- Si el usuario pulsa el botón de cámara, aparecerá la vista del carrete de fotos del iPhone. Seleccionará una foto y ésta aparecerá la vista de Registro.
3.- Cuando haya terminado de completar el formulario se comprobará si el usuario existe. En caso de existir aparecerá un error y si no existe, se continuará.
4.- Una vez recibida la confirmación, se pasará a almacenar los datos del usuario en la base de datos y volverá a la vista anterior.
Postcondiciones: Se habrá creado un nuevo usuario.
Interfaz gráfica:

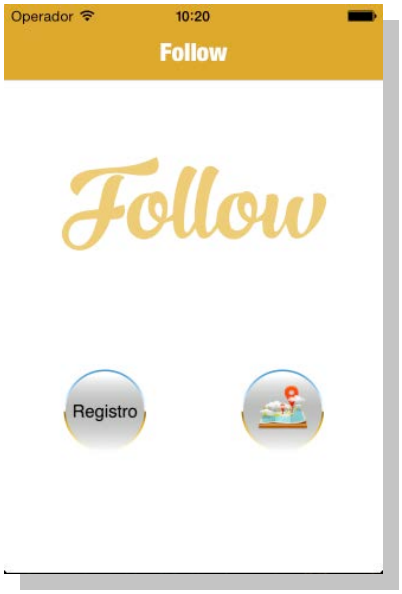


Ilustración 38.- Interfaz-1

Ilustración 39.- Interfaz-2

Nombre: Consultar guías.
Descripción: Permite al usuario ver la ficha de una ciudad y buscar una ciudad descargada o la que quiera descargar.
Actores: Usuario.
Precondiciones: Ninguna.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario tendrá dos formas de consultar una ciudad, buscándola manualmente o mediante el botón de búsqueda.

1.1.- Manualmente: El usuario se situará en una de las tres pestañas: Mis Guías, Descargar o Más Votadas.

1.1.1.- Mis Guías: Al acceder a la vista "MisGuíasVC" se recogerá automáticamente la información de la BD interna por orden alfabético de países y nombre de las ciudades.

1.1.2.- Descargar: Al acceder a esta pestaña, si no se hubiera obtenido aún las ciudades, se consultaría en la BD externa ordenándolas alfabéticamente por países y por el nombre de las ciudades.

1.1.3.- Más votadas: Al pulsar en ella se comprobará si no se han obtenido aún las ciudades más votadas, si no se accede a la BD externa y se recoge las cinco ciudades más votadas ordenadas de mayor a menor puntuación.

1.2.- Pinchará sobre una ciudad que elija y se accederá a la ficha de la Ciudad.

2.- Si el usuario decide buscar mediante el botón de búsqueda:

2.1.- El usuario pulsará el botón "Mis Guías" o "Descargar" para situarse en el apartado que desea buscar. Debido a la poca cantidad de Ciudades en la pestaña "Más votadas", se imposibilitará la opción de buscar.

2.2.- Pulsará el botón de la lupa y aparecerá una barra de búsqueda. Al pulsar sobre ella permitirá escribir. Cuando el usuario inserta una letra, se busca las ciudades que coincidan con el rango de la letra introducida. A medida que se introduce más letras, el rango se hace mayor y la cantidad de ciudades, menor.

2.3.- Si en el rango de letras escrito no hay una ciudad que coincida, la tabla aparecerá vacía.

2.4.-Cuando haya encontrado la ciudad puede pulsar sobre ella y acceder a la ficha de la ciudad. Si por el contrario pulsa el botón de la lupa de nuevo, la búsqueda desaparecerá y aparecerán la ciudad descargadas si se está en el apartado "Mis Guías" y todas las ciudades para descargar si el usuario se encuentra en el apartado "Descargar".

3.- Una vez accedido a la ficha de una ciudad, se muestra la cantidad de información (lugares, fotos, vídeos, rutas o metro) que el usuario pudiera tener descargada así como lo que hay almacenado en la BD.

Postcondiciones: Se habrá accedido a la ficha de la ciudad ya sea con una búsqueda manual o mediante la búsqueda por letras, apareciendo en pantalla la información descargada por el usuario y la almacenada en la BD externa.
Interfaz gráfica:

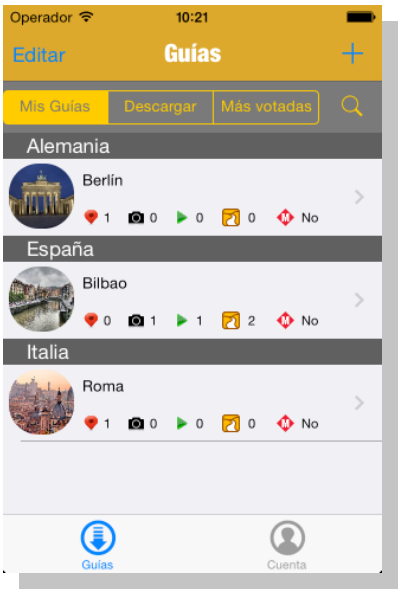


Ilustración 40.- Interfaz-3

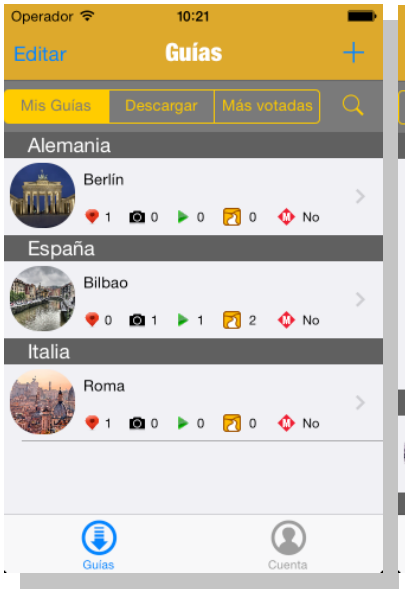


Ilustración 41.- Interfaz-4

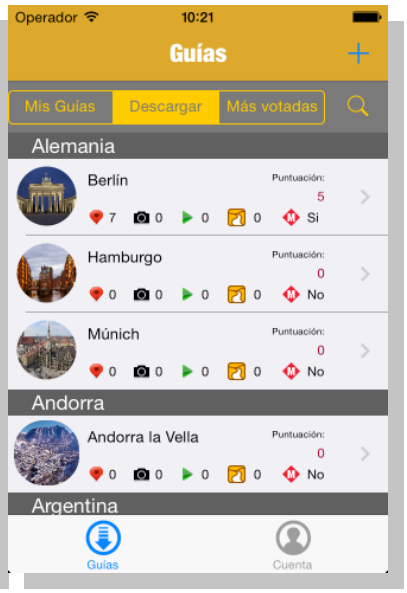


Ilustración 42.- Interfaz-5

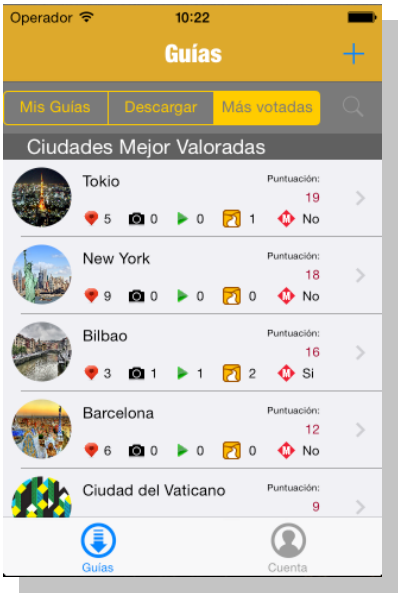


Ilustración 43.- Interfaz-6

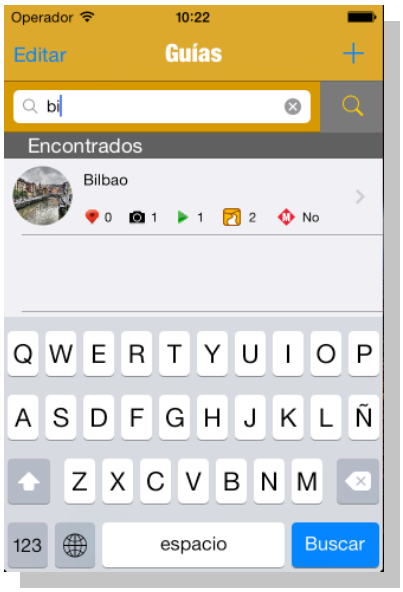


Ilustración 44.- Interfaz-7

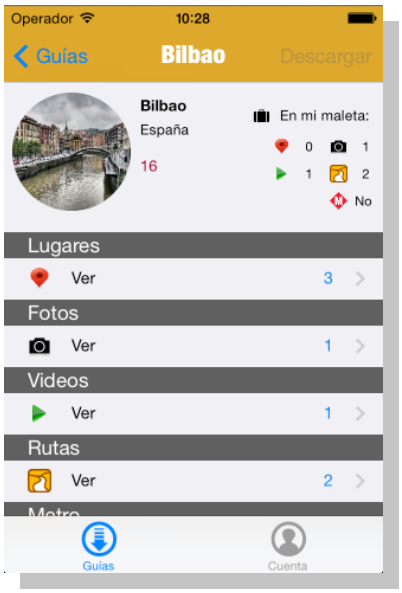


Ilustración 45.- Interfaz-8

Nombre: Descargar ciudad
Descripción: Permite al usuario descargar una guía completa de la ciudad. Se descarga todo.
Actores: Usuario.
Precondiciones: Conexión a Internet y no tener descargada la ciudad.
Requisitos no funcionales: Ninguno
Flujo de eventos:
1.- El usuario elegirá la ciudad a "Descargar" desde la pestaña descargar y así, acceder a la ficha de la Ciudad.
2.- Pulsará el botón Descargar y empezará la descarga.
3.- Primero, se guardará la información de la Ciudad y se descargará la imagen del metro, si es que la hubiera.
4.- Se comprobará toda la información que le pertenezca. Primero se comprobará si hay lugares que pertenezcan a esa Ciudad. Si los hay, se descargan las fotos de perfil al dispositivo y se guarda la información.
5.- Después las fotos y vídeos que puedan pertenecer a los lugares o a la ciudad, si los hay, se descargarán al dispositivo y se creará un registro.
6.- Por último, se comprobará si hay rutas registradas en esa Ciudad. Si las hay, se descargará la información.
7.- Los parámetros de "En mi maleta" cambiarán acorde de lo descargado.
Postcondiciones: El usuario tendrá a su disposición la guía de la ciudad desde "Mis Guías" sin necesidad de conectarse a Internet.
Interfaz gráfica:

Ilustración 46.- Interfaz- 9

Nombre: Borrar Ciudad.
Descripción: Permite al usuario borrar una guía descargada.
Actores: Usuario.
Precondiciones: Tener una guía descargada.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario se situará en la pestaña de "Mis Guías".
2.- Pulsará el botón "Editar" y las celdas cambiarán de estado, al estado "borrar".
3.- Elegirá una ciudad y pulsará el botón de la izquierda. A continuación, pulsará en el botón "Delete" que habrá aparecido.
4.- Se borrará la información de la Ciudad así como la foto del metro, si la hubiera.
5.- Por consiguiente, se borrará todo lo que el usuario se ha descargado de esa ciudad. Se comprobará las fotos descargadas de las ciudades y lugares y se borrarán del dispositivo, junto con su información.
6.- Se comprobará los lugares descargados y se borrarán las fotos de perfil y la información descargada.
7.- Y por último, se comprueba las rutas descargadas y se borran.
8.- Se borrará de la tabla la ciudad borrada y se saldrá del estado "borrar", devolviendo el botón "editar" a su estado normal.
Postcondiciones: La ciudad quedará borrada, así como todo lo que esté relacionado con ella. La ciudad estará borrada también en la tabla de "Mis Guías".
Interfaz gráfica:

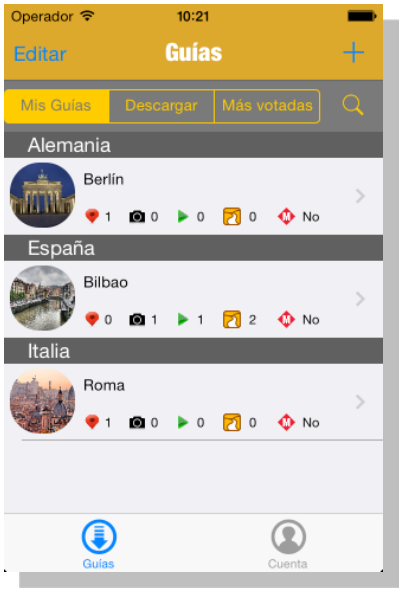


Ilustración 47.- Interfaz-10

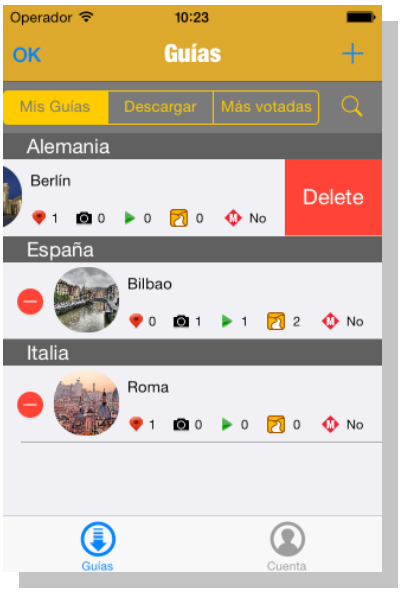


Ilustración 48.- Interfaz-11

Nombre: Descargar foto o vídeo.
Descripción: Permite al usuario descargar una foto o todas a la vez.
Actores: Usuario.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- Si el usuario va a descargar una foto o vídeo, lo elige desde la pestaña "Todas".
2.- Pulsa sobre una de las fotos o vídeo que quiera ver. A continuación, se abrirá una nueva vista que mostrará la foto en la que el usuario podrá acercar o alejar la foto para verla mejor o si es un vídeo poder reproducirlo.
3.- El usuario pulsará el botón descargar y la foto o vídeo se descargará si todo ha ido bien mostrando un mensaje, guardándose en el dispositivo y almacenando la dirección en la que se ha guardado. Si no, mostrará un mensaje de error.
Postcondiciones: La foto o vídeo de la Ciudad o del Lugar, se ha almacenado en el dispositivo.
Interfaz gráfica:

Ilustración 50.- Interfaz-12

Ilustración 49.- Interfaz-13

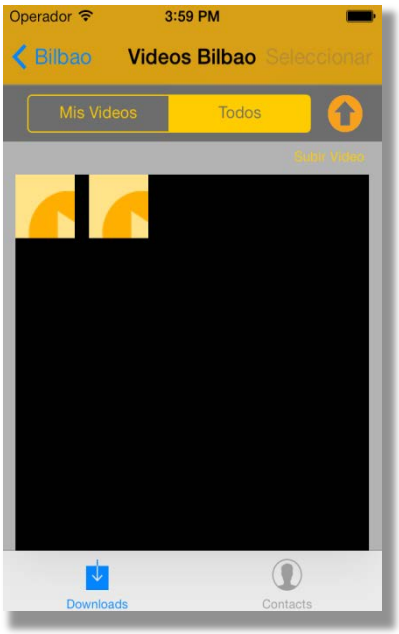


Ilustración 52.- Interfaz-14

Ilustración 51.- Interfaz-15

Ilustración 53.- Interfaz-16

Nombre: Subir recurso.
Descripción: Permite al usuario subir una foto o vídeo de la Ciudad o del Lugar que haya elegido.
Actores: Registrado.
Precondiciones: Conexión a Internet y estar registrado en el sistema.
Requisitos no funcionales: El archivo no puede ser mayor de 1MB si es una foto y 50MB si es un vídeo.
Flujo de eventos:
FOTO:
1a.- El usuario irá al apartado de fotos de la Ciudad o del Lugar.
2a.- Pulsará el botón "Subir Foto" que estará indicado con una flecha hacia arriba.
3a.- A continuación, se abrirá una nueva vista, y el usuario pulsará el botón "Escoger".
4a.- Aparecerá la vista del carrete de fotos y le dará la opción de elegir el álbum de fotos que quiera. Una vez elegido, elegirá la foto que él crea conveniente y pulsará sobre ella.
5a.- La foto aparecerá en grande en la pantalla del punto 3. Y entonces, si pulsa el botón "Subir" representado con un cuadrado y una flecha hacia arriba la imagen empezará a subirse al servidor.
ViDEO:
1b.- El usuario irá al apartado de vídeos de la Ciudad o del Lugar.
2b.- Pulsará el botón "Subir Vídeo" que estará indicado con una flecha hacia arriba.
3b.- A continuación, se abrirá una nueva vista, y el usuario pulsará el botón "Subir" representado con un cuadrado y una flecha hacia arriba.
4b.- Aparecerá la vista con los álbumes de vídeos del usuario, permitiendo que elija la que desee. Una vez seleccionado, elegirá el vídeo que él crea conveniente y pulsará sobre él. El vídeo se mostrará en una nueva ventana, que podrá ser reproducido al pulsar en el botón "Elegir" (Choose, en inglés).
5b.-El vídeo comenzará a comprimirse, y una vez finalizado, empezará a subirse al servidor.

6ab.- Tanto a una foto como a un vídeo, en primer lugar, se le dará un nombre aleatorio entre 1 y 999999, se guardará en el servidor y a continuación se guardará la URL que hace referencia a donde está guardado.
7ab.- Si ha ocurrido algún error, se mostrará una alerta.
Postcondiciones: La foto o el vídeo se ha subido al servidor y se ha guardado la referencia.
Interfaz gráfica:

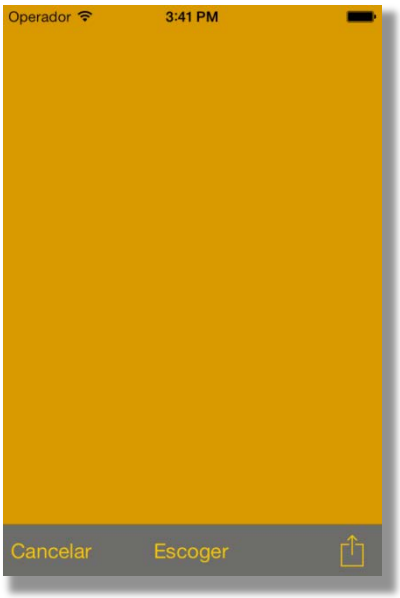


Ilustración 55.- Interfaz-17

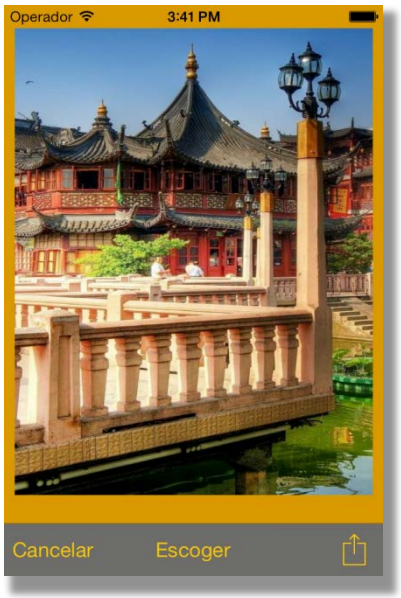


Ilustración 54.- Interfaz-18

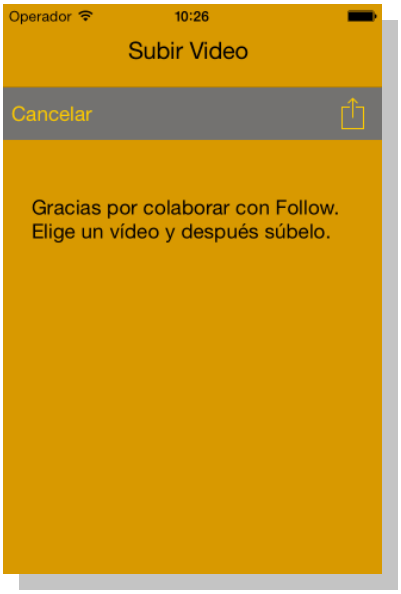


Ilustración 57.- Interfaz-19

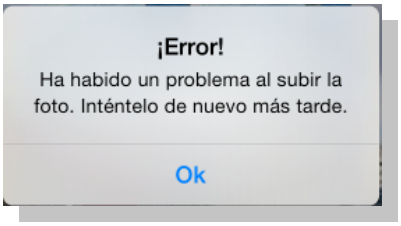


Ilustración 56.- Interfaz-20

Nombre: Borrar foto o vídeo.
Descripción: El usuario tiene la posibilidad de borrar varias fotos o vídeos descargadas.
Actores: Usuario.
Precondiciones: Haber descargado alguna foto o vídeo.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario irá al apartado de fotos o vídeos de la Ciudad o del Lugar.
2.- Deberá encontrarse en la pestaña de "Mis Fotos". Entonces, el usuario pulsará el botón "Seleccionar".
3.- El sistema entrará en el modo "selección" y aparecerá un nuevo botón llamado "Borrar".
4.- Si el usuario pulsa el botón borrar sin seleccionar nada, se saldrá del modo "selección". Si no seleccionará todas las fotos o vídeos que quiera pulsando una vez sobre ellas. Si pulsa de nuevo, la selección desaparecerá.
5.- Una vez decididas las fotos o vídeos a borrar, el usuario pulsará el botón "Borrar".
6.- Se borrarán los recursos seleccionados de la tabla, así como de la memoria del móvil y sus referencias guardadas.
7.- Una vez terminado, se saldrá del modo "selección".

Postcondiciones: Las fotos o vídeos han sido borradas con éxito de la memoria y sus referencias guardadas.
Interfaz gráfica:

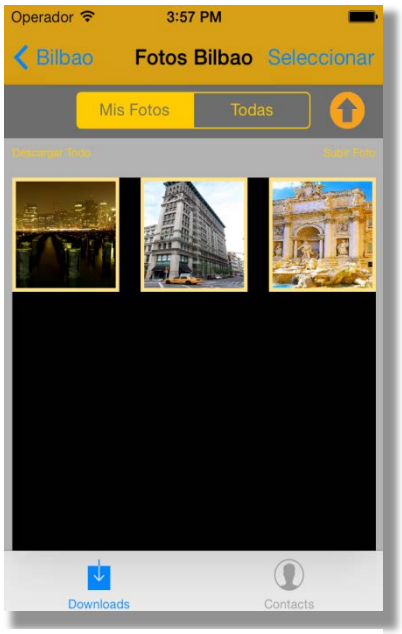


Ilustración 59.- Interfaz-21

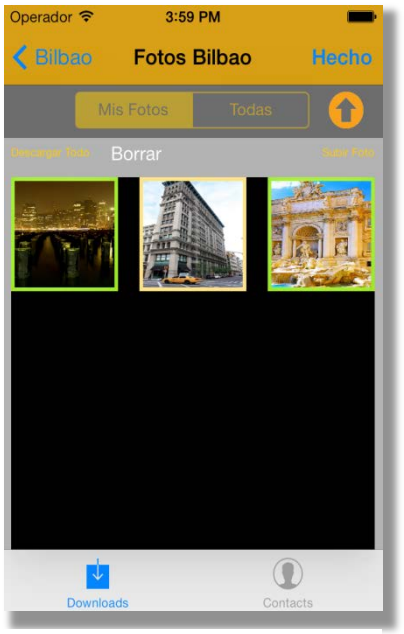


Ilustración 60.- Interfaz-22

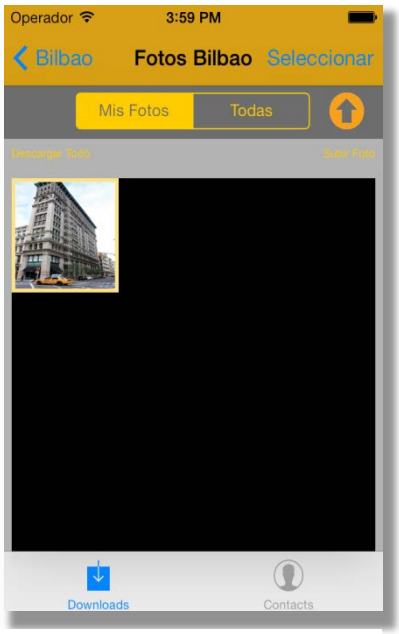


Ilustración 58.- Interfa-23

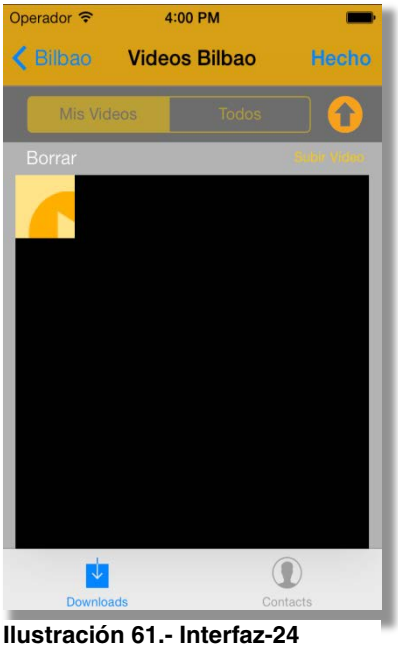


Ilustración 61.- Interfaz-24

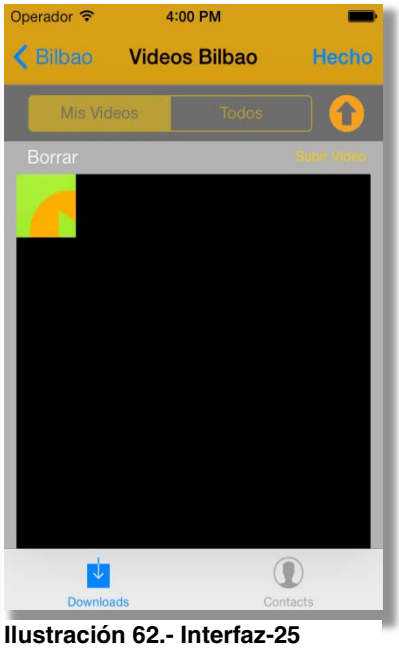


Ilustración 62.- Interfaz-25

Ilustración 63.- Interfaz-26

Nombre: Compartir Fotos.
Descripción: El usuario tiene la posibilidad de compartir las fotos que quiera en Facebook.
Actores: Usuario.
Precondiciones: Conexión a internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario irá al apartado de fotos de la Ciudad o del Lugar.
2.- Deberá encontrarse en la pestaña de "Todas". Entonces, el usuario pulsará el botón "Seleccionar".
3.- El sistema entrará en el modo "selección" y aparecerá un nuevo botón llamado "Compartir".
4.- Si el usuario pulsa el botón "Compartir" sin seleccionar nada, se saldrá del modo "selección". Si no, seleccionará todas las fotos que quiera pulsando una vez sobre ellas. Si pulsa de nuevo, la selección desaparecerá.
5.- Una vez decididas las fotos para compartir, el usuario pulsará el botón "Compartir".
6.- Aparecerá una nueva ventana en la que mostrará un título preconfigurado que el usuario podrá cambiar y escribir lo que quiera. También se mostrará la cantidad de fotos seleccionadas.
6.1.- Si quisiera que las fotos se subieran en Facebook a un álbum concreto, el usuario puede seleccionar "Album" y seleccionar donde almacenarlas.
6.2.- Si quisiera mostrar su localización, pulsará en "Location".
6.3.- También puede cambiar la gente que las pueda ver en Facebook pulsando en "Audience".
7.- Una vez personalizado, el usuario pulsará en Post y las fotos se subirán a Facebook.
8.- Se volverá a la vista anterior y se saldrá del modo "selección".
Postcondiciones: Las fotos han sido publicadas en Facebook con un título, guardadas en un álbum concreto, o no, con la localización elegida y mostrada a la gente con la que ha querido compartirlas.
Interfaz gráfica:

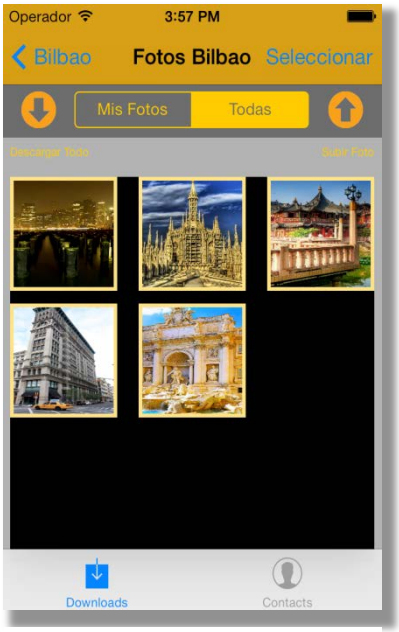


Ilustración 65.- Interfaz-27

Ilustración 64.- Interfaz-28

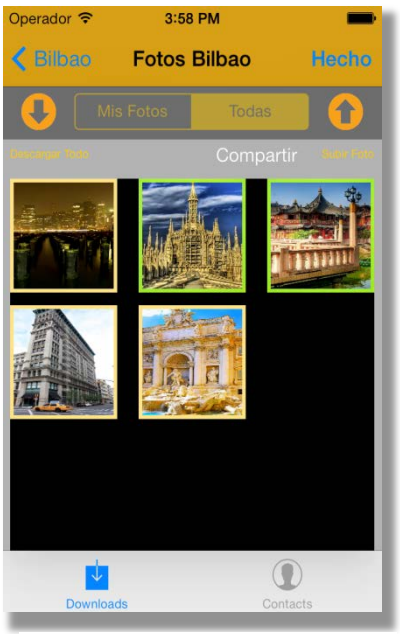


Ilustración 67.- Interfaz-29

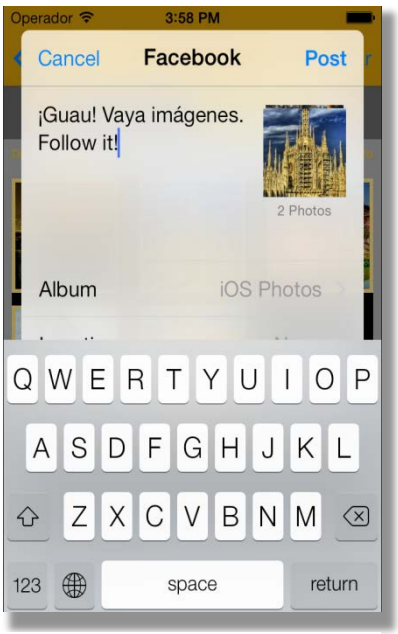


Ilustración 66.- Interfaz-30

Nombre: Descargar Ruta.
Descripción: Permite que el usuario descargue las rutas que quiera.
Actores: Usuario.
Precondiciones: Conexión a internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario irá al apartado de Rutas de la Ciudad.
2.- Deberá encontrarse en la pestaña de "Descargar" o "Más votadas". Entonces, se recogerán las rutas de esa ciudad para ser mostradas en la tabla.
3.- El usuario pulsará el botón "Descargar" que estará situado a la izquierda de la celda.
4.- La información de la ruta será almacenada.
4.1.- Si el usuario tendría la ciudad descargada, se actualizará la ciudad con una ruta más.
4.2.- Si no, se crear un nuevo registro de la ciudad y se añadiría la ruta posteriormente.
5.- Aparecerá un "tick" sustituyendo el botón de descarga y confirmando la misma.
Postcondiciones: La ruta descargada ha sido almacenada con éxito y actualizada la cantidad de rutas descargadas de la ciudad o de no haber descargado la ciudad previamente, se habrá creado un nuevo registro de la ciudad.
Interfaz gráfica:

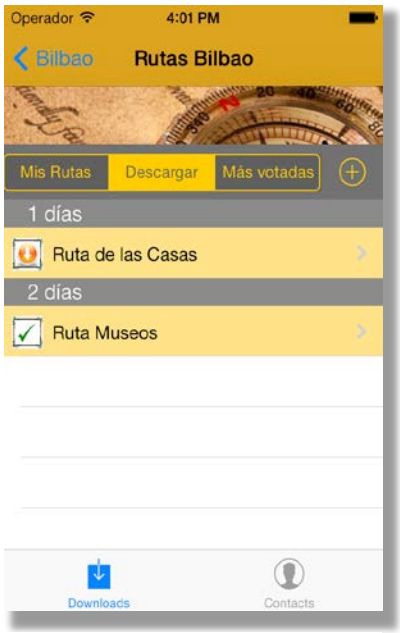


Ilustración 69.- Interfaz-31

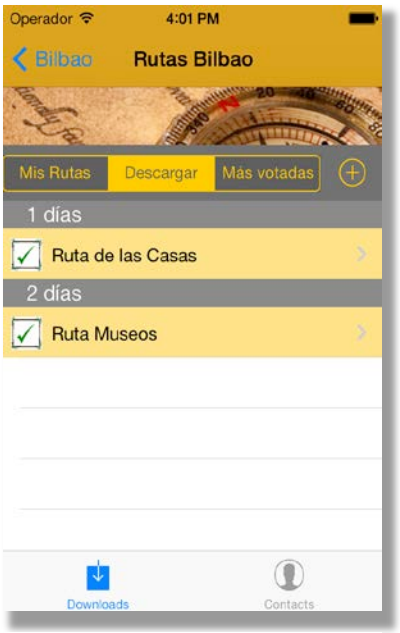


Ilustración 68.- Interfaz-32

Nombre: Borrar Ruta.
Descripción: Permite que el usuario borre las rutas que haya descargado.
Actores: Usuario.
Precondiciones: Tener al menos una Ruta descargada.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario se situará en la pestaña de "Mis Rutas".
2.- Pulsará el botón "Editar" y las celdas cambiarán de estado, al estado "borrar".
3.- Elegirá una ruta y pulsará el botón de la izquierda. A continuación, pulsará el botón "Delete" que aparecerá en rojo.
4.- Se borrará la información de la ruta seleccionada. A su vez, se actualizará la cantidad de rutas descargadas de la ciudad. La celda que contiene la ruta, se quitará de la tabla.
Postcondiciones: Se habrá borrado la información de la ruta seleccionada y además se habrá actualizado la cantidad de rutas descargadas de la Ciudad.
Interfaz gráfica:

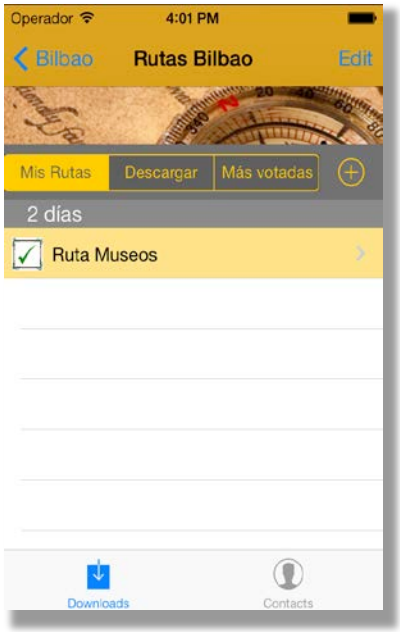


Ilustración 70.- Interfaz-33

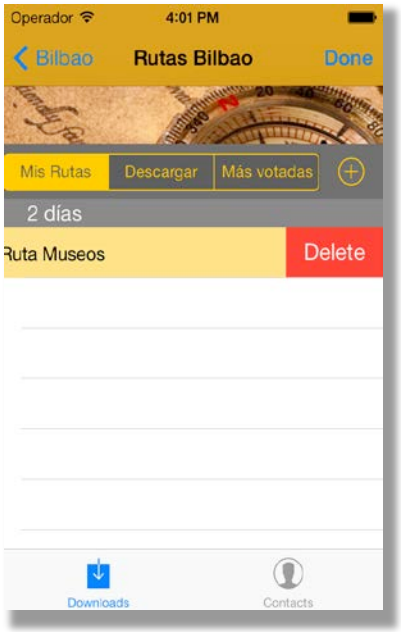


Ilustración 71.- Interfaz-34

Nombre: Crear Ruta.
Descripción: Permite que el usuario cree rutas en una ciudad concreta.
Actores: Registrado.
Precondiciones: Ser usuario registrado y conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsará el botón "+".
2.- Se accederá a una nueva vista, "CrearRuta".
3.- El usuario añadirá el nombre de la ruta, los días y una descripción.
3.1.- Si no, saldrá un mensaje de advertencia indicando que tiene que rellenar todos los campos.
4.- El usuario pulsará "Finalizar" y se almacenará la información de la ruta.
5.- Aparecerá un mensaje de error si algo ha salido mal.

Postcondiciones: Una nueva ruta ha sido almacenada en la BD externa.
Interfaz gráfica:

Ilustración 74.- Interfaz-35

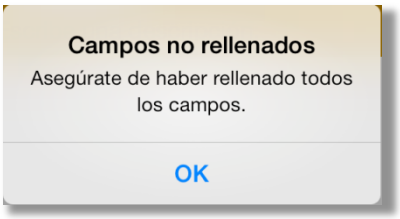


Ilustración 72.- Interfaz-36

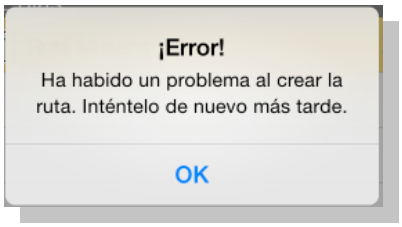


Ilustración 73.- Interfaz-37

Nombre: Descargar Lugar.
Descripción: Permite que el usuario descargue lugares de la ciudad.
Actores: Usuario.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario deberá encontrarse en la pestaña de "A-Z", "Punt." o "No desc." Entonces, se recogerá la información de los lugares de esa ciudad ordenados dependiendo de la pestaña elegida.
2.- El usuario pulsará el botón "Descargar" que estará situado a la izquierda de la celda.
3.- La información del lugar será almacenado y la foto de perfil se descargará al dispositivo.
3.1.- Si el usuario tendría la ciudad descargada, se actualizará la ciudad sumando uno a la cantidad de lugares descargados.
3.2.- Si no, se crea se crea un registro de la ciudad con su foto de perfil y se añadiría el lugar posteriormente.
4.- Aparecerá un "tick" sustituyendo el botón de descarga y confirmando la misma.
Postcondiciones: La información de un lugar ha sido almacenada y podrá consultarse *offline*.
Interfaz gráfica:

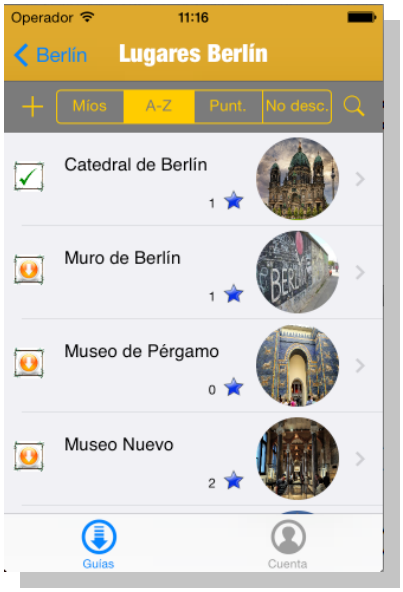


Ilustración 76.- Interfaz-38

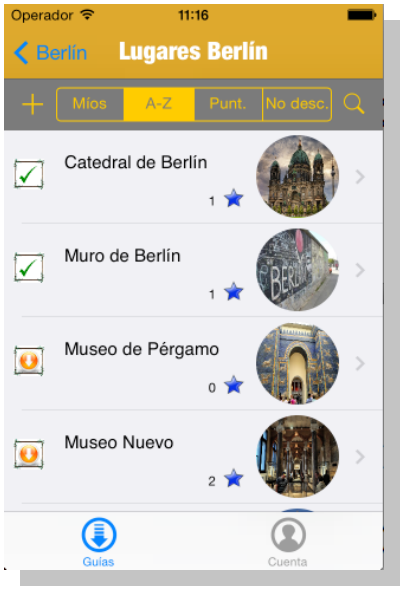


Ilustración 75.- Interfaz-39

Nombre: Borrar Lugar.
Descripción: Permite que el usuario borre los lugares que haya descargado.
Actores: Usuario.
Precondiciones: Tener al menos un Lugar descargado.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario se situará en la pestaña “Míos”.
2.- Pulsará el botón “Editar” y las celdas cambiarán al estado “borrar”.
3.- Elegirá un lugar pulsando el botón rojo de la izquierda. A continuación, pulsará el botón “Delete” que aparecerá en rojo.
4.- Se borrará la información del lugar seleccionado y la foto de perfil de la memoria del dispositivo. A su vez, se actualizará la cantidad de lugares descargados de la ciudad.
5.- La celda que contiene el lugar, se quitará de la tabla.
Postcondiciones: La información de los lugares seleccionados se habrán borrado y la cantidad de lugares descargados de la Ciudad se habrá actualizado.
Interfaz gráfica:

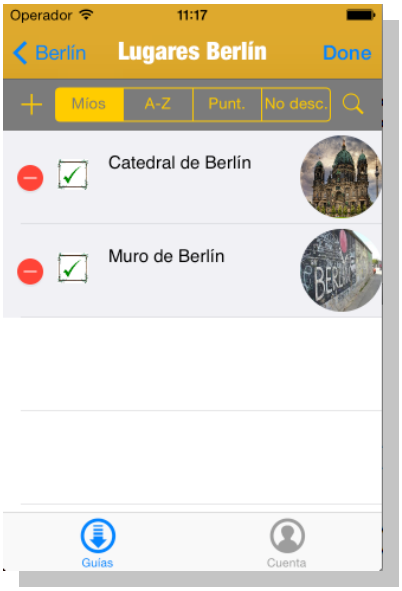


Ilustración 78.- Interfaz-40

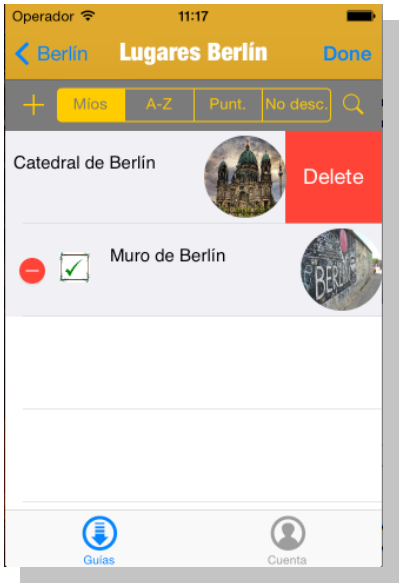


Ilustración 77.- Interfaz-41

Nombre: Crear Lugar.
Descripción: Permite que el usuario añada un Lugar.
Actores: Registrado.
Precondiciones: Ser usuario registrado y conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsará en el botón "+" en el apartado de Lugares de la ciudad.
2.- Aparecerá una nueva ventana y el usuario tendrá que añadir un nombre.
3.- Para añadir la descripción, el horario y el precio, pulsará el botón "Siguiente" para ir escribiendo el texto.
3.1.- Si pulsa "Atrás" volverá al apartado anterior con la posibilidad de modificar el texto introducido o seguir introduciendo texto.
4.- El usuario puede pulsar el botón de la cámara de fotos para subir una foto del Lugar como foto de perfil. No es obligatorio.
5.- Si el usuario pulsa el botón "Finalizar" y no ha rellenado todos los datos, saldrá un mensaje de advertencia.
6.- Si no, saldrá un mensaje de éxito y volverá a la vista anterior.
Postcondiciones: Se ha guardado la información del nuevo Lugar. Si se ha subido una foto de perfil, se habrá subido al servidor y se habrá guardado la referencia.
Interfaz gráfica:

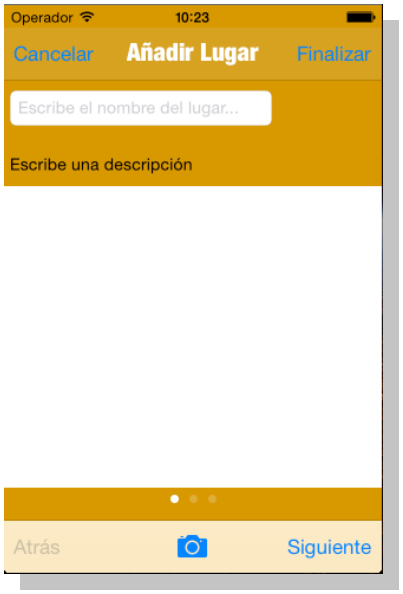


Ilustración 80.- Interfaz-42

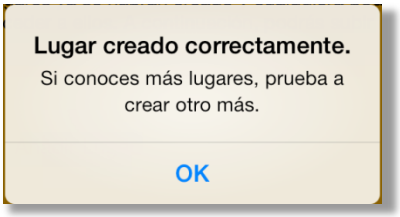


Ilustración 79.- Interfaz-43

Nombre: Consultar lugar en Mapa.
Descripción: El usuario puede ver el lugar en un mapa.
Actores: Usuario.
Precondiciones: Conexión a Internet. Permitir la Localización del usuario en el dispositivo.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accederá al apartado de "Mapa" de un lugar.
2.- El sistema recogerá el nombre del lugar y el nombre de la ciudad.
3.- Hará una búsqueda que coincida con ambos nombres y de entre todas las posibilidades elegirá la primera, que es la más cercana al lugar que se está buscando.
4.- Si no encuentra el lugar, aparecerá un mensaje indicando que el usuario ponga el nombre en otro idioma o lo busque de alguna otra manera. Hará el mismo procedimiento que en el punto 2 y 3.
5.- Si encuentra el lugar correctamente, marcará con un pin el lugar elegido.
6.- El usuario ahora tendrá la oportunidad de pinchar sobre el pin para crear una ruta.
6.1.- El sistema recogerá la ubicación del usuario y calculará el camino más corto.
6.2.- Aparecerá una línea roja que indique el camino a seguir para que el usuario pueda llegar a ese lugar.

Postcondiciones: Aparecerá una indicación (un pin) en el mapa, indicando al usuario ese lugar. Si se ha pinchado sobre el pin, aparecerá una ruta en rojo y la distancia que le separa al usuario de ese lugar.
Interfaz gráfica:

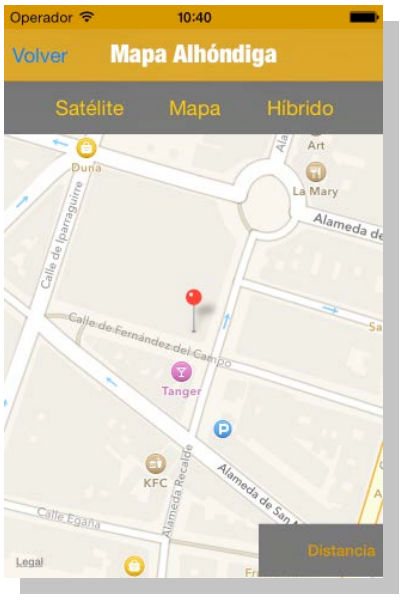


Ilustración 83.- Interfaz-44

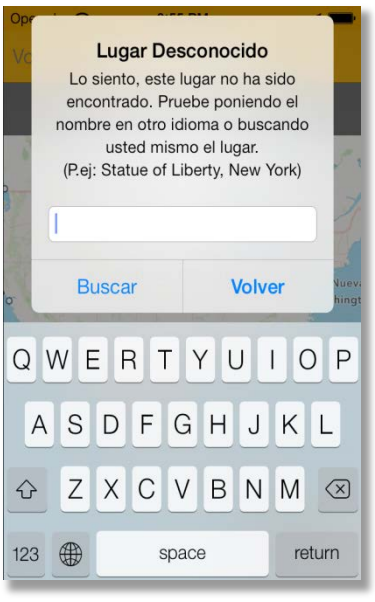


Ilustración 82.- Interfaz-45

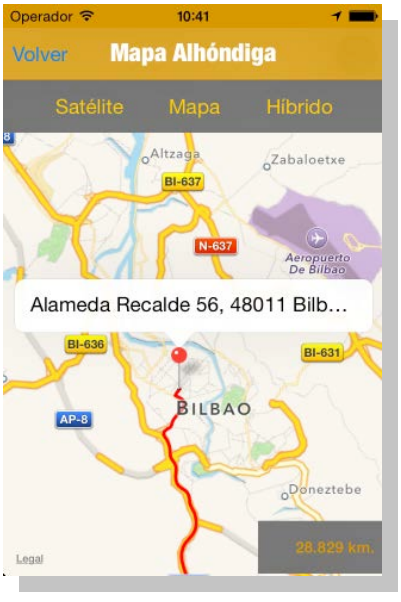


Ilustración 81.- Interfaz-46

Nombre: Escribir comentario.
Descripción: Permite que el usuario escriba un comentario sobre el lugar.
Actores: Registrado.
Precondiciones: Conexión a Internet. Estar registrado en el sistema.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsará el botón de "Añadir" referenciado con un cuadrado y un lápiz.
2.- Se mostrará una nueva ventana donde el usuario introducirá un título y el comentario que quiera escribir.
3.- Cuando haya acabado, pulsará el botón enviar.
3.1.- Se almacenará la información introducida con el nombre del usuario con el que esté conectado y la fecha en la que se ha escrito el comentario.
4.- Si todo ha ido bien se mostrará un mensaje de éxito y se volverá a la vista anterior. Si no, se mostrará un mensaje de error.
Postcondiciones: Los comentarios sobre el lugar se mostrarán por pantalla.
Interfaz gráfica:

Ilustración 84.- Interfaz-47

Ilustración 85.- Interfaz-48

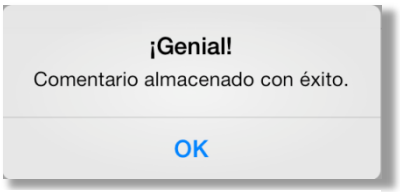


Ilustración 86.- Interfaz-49

Nombre: Consultar Lugares.
Descripción: Permite que el usuario acceda a la información de los lugares.
Actores: Usuario.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario entra al apartado de Lugares de una ciudad concreta.
2.- El sistema recoge la información de todos los lugares descargados marcándolos con un tick que marca que está descargado, así como introduciendo el nombre y la foto de perfil si tuviera, sino, pondría la imagen por defecto.
3.- Si el usuario quiere consultar los lugares que hay disponibles dependiendo del orden que quieran tener, ya sea alfabéticamente, por puntuación o los no obtenidos, marcará la pestaña "A-Z", "Punt." o "No obt.", respectivamente.
4.-El usuario tiene la posibilidad de buscar los lugares pulsando en la lupa. Aparecerá una barra de búsqueda tras pulsar el botón. Al pulsar sobre ella permitirá escribir. Cuando el usuario inserta una letra, busca los lugares que coincidan con el rango de la letra introducida. A medida que se introduce más letras, el rango se hace mayor y la cantidad de lugares, menor.
5.- Si en el rango de letras escrito no hay un lugar que coincida, la tabla aparecerá vacía.
6.- Cuando el usuario haya encontrado el lugar puede pulsar sobre ella y acceder a la ficha de la ciudad. Si por el contrario pulsa el botón de la lupa de nuevo, la búsqueda desaparecerá y aparecerán todos los lugares de nuevo, dependiendo de la pestaña en la que el usuario se encuentre.
7.- Tras haber pulsado sobre un lugar, se accederá a una nueva vista que mostrará información sobre el lugar elegido, así como una imagen de perfil. Si no tuviera una imagen, aparecerá un dibujo por defecto.
Postcondiciones: Se habrá mostrado los lugares ordenados en las diferentes pestañas, se mostrará en la tabla los lugares buscados y se habrá accedido correctamente a la ficha del lugar elegido.
Interfaz gráfica:

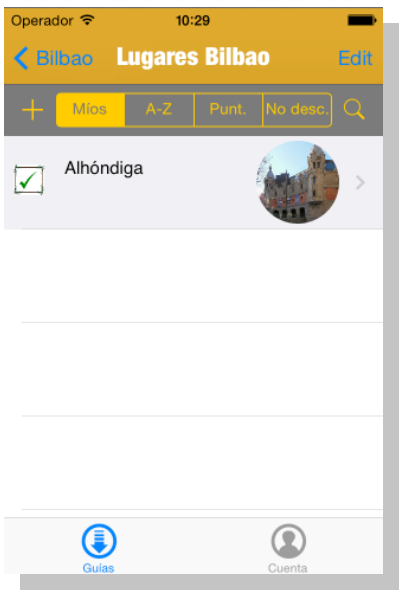


Ilustración 89.- Interfaz-50

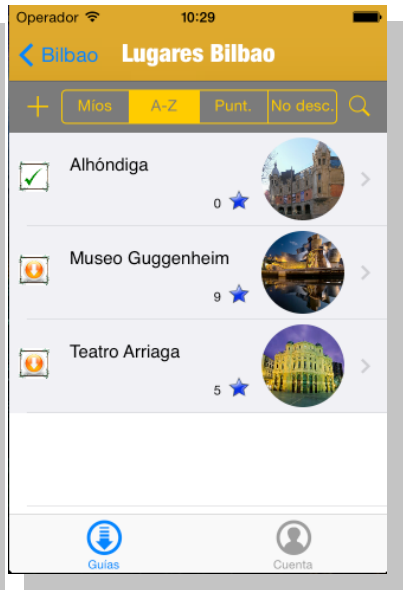


Ilustración 87.- Interfaz-51

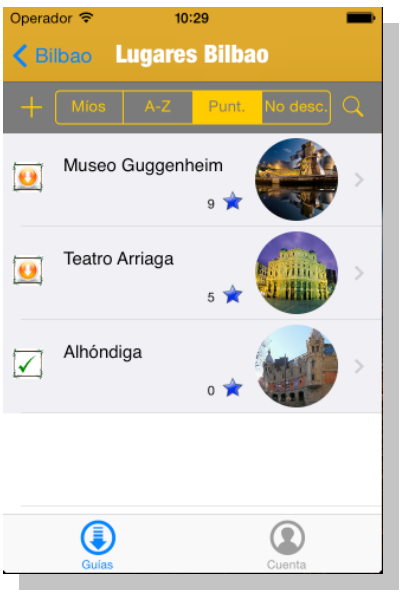


Ilustración 88.- Interfaz-52

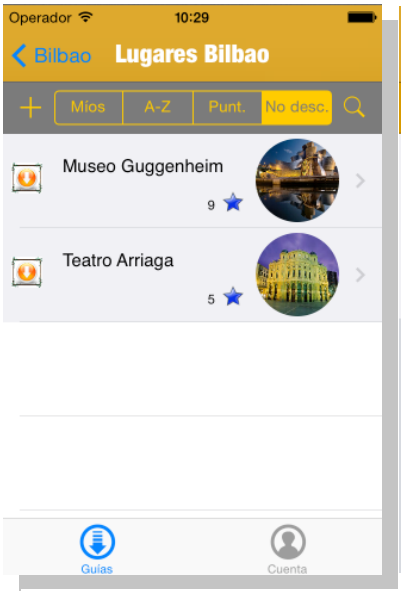


Ilustración 90.- Interfaz-53

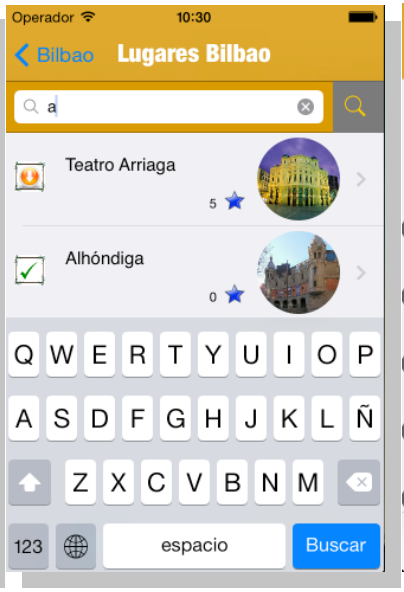


Ilustración 91.- Interfaz-54

Ilustración 92.- Interfaz-55

Nombre: Consultar Fotos y Vídeos
Descripción: Permite ver las fotos o los vídeos descargados o los del servidor.
Actores: Usuario.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario entra al apartado de fotos o vídeos de una ciudad concreta o un lugar concreto.
2.- El sistema recogerá toda la información de las fotos y los vídeos descargados en el dispositivo.
3.- Si pulsa en la pestaña Todos, recogerá la información y dirección de las fotos y los vídeos alojados en el servidor.
4.- Una vez que el usuario decida qué foto o vídeo ver, pulsará sobre él y se irá a una nueva vista donde el usuario tendrá oportunidad de acercar o alejar la foto o de ver el vídeo.
5.- Pulsando sobre el botón “Ver Vídeo” se abrirá la ventana del reproductor de vídeo, accediendo a éste con la dirección que se le entregue. Si el vídeo no puede ser cargado, simplemente no cargará y se quedará esperando. El usuario puede cancelar el vídeo en cualquier momento.
6.- Una vez visto el vídeo, el usuario pulsará el botón “Done” y volverá a la ventana anterior.
Postcondiciones: Se mostrará las fotos y los vídeos, tanto descargados como no y se habrá mostrado una foto y reproducido un vídeo correctamente tanto si está descargado como si está en el servidor.
Interfaz gráfica:

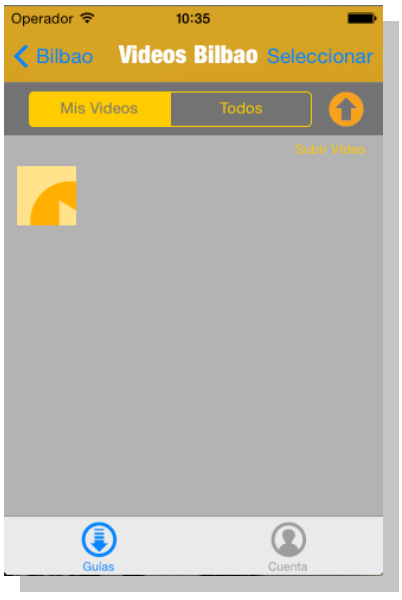


Ilustración 94.- Interfaz-56

Ilustración 93.- Interfaz-57

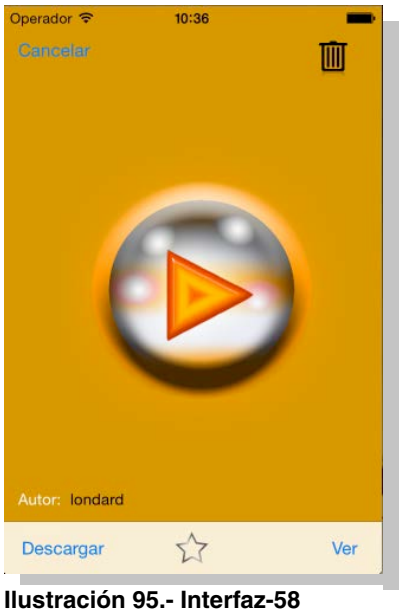


Ilustración 95.- Interfaz-58

Ilustración 96.- Interfaz-59

Nombre: Iniciar Sesión.
Descripción: Permite a un usuario iniciar sesión en del sistema.
Actores: Registrado.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsa el apartado de "Cuenta", en la barra de abajo.
2.- El sistema comprobará si ha iniciado sesión.
2.1.- Si el usuario no ha iniciado sesión previamente, se mostrará la ventana de Iniciar Sesión.
2.2.- El usuario introducirá el *nick* de usuario y la contraseña.
2.3.- Si el usuario ha introducido correctamente los datos, se guardará el Nick y la contraseña en el dispositivo para futuros accesos. Si no, aparecerá un mensaje de error.
3.- Una vez que el usuario se ha conectado correctamente, se recogerá su información y se mostrará tan solo el nick y edad, así como su foto de perfil. Si no, se mostrará la imagen por defecto.
Postcondiciones: El usuario habrá iniciado sesión correctamente en el sistema o borrado las credenciales en caso de haber cerrado la sesión.
Interfaz gráfica:

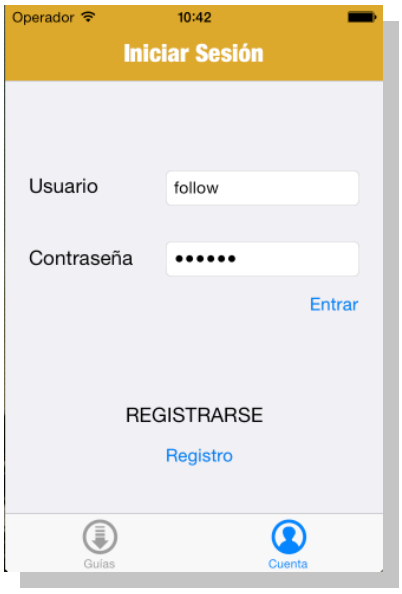


Ilustración 97.- Interfaz-60

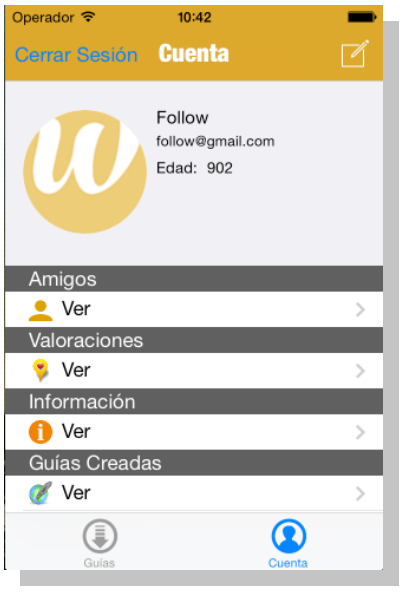


Ilustración 98.- Interfaz-61

Nombre: Cerrar Sesión.
Descripción: Permite a un usuario cerrar sesión.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- Si el usuario quiere cerrar la sesión, pulsará en “Cerrar Sesión” y el Nick y *password* se borrarán del dispositivo, volviendo a la pantalla principal.
Postcondiciones: El usuario habrá cerrado la sesión correctamente en el sistema y se habrá borrado sus credenciales guardadas.
Interfaz gráfica:

Ilustración 99.- Interfaz-62

Ilustración 100.- Interfaz-63

Nombre: Editar Perfil.
Descripción: Permite a un usuario modificar su información personal o agregar nueva información a su perfil.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsa en "Información" desde la vista "Cuenta".
2.- A la nueva vista se le pasará la información del usuario y se mostrará en pantalla.
3.- Si el usuario decide editar el perfil, pulsará en el botón "Editar" para entrar en modo "edición", cambiando el botón de "Editar" a "Hecho".
4.- Los cuadros de texto se habilitarán para su edición así como un botón para subir una nueva foto de perfil y un botón de candado para bloquear el usuario o desbloquearlo.
4.1.- Si pulsa en el botón de la cámara aparecerá una nueva ventana que da la posibilidad de elegir una nueva foto.
4.2.- Cuando haya elegido la foto se le ofrecerá editarla para encajarla en un cuadrado. Cuando lo haya editado, pulsará finalizar para cerrar la vista y volver a la vista de la información del usuario. La imagen que estaba anteriormente se borrará del servidor.
5.- Una vez que el usuario haya hecho los cambios pertinentes, pulsará en "Hecho" y se saldrá del modo "edición" y se bloquearán de nuevo los cuadrados de texto.
6.- Toda la información del usuario se recogerá y se actualizará. Si ocurre algún problema, aparecerá un mensaje de error indicando que no ha podido actualizarse y se reestablecerán los valores anteriores a las cajas de texto.
7.- Si ha salido bien, al volver a la vista de "Cuenta", el usuario verá modificada la foto, el correo o la edad si esta información hubiera sufrido algún cambio.
Postcondiciones: El usuario habrá modificado o agregado información a su perfil correctamente.
Interfaz gráfica:

Ilustración 102.- Interfaz-64

Ilustración 101.- Interfaz-65

Nombre: Consultar Mensajes.
Descripción: Permite a un usuario acceder a sus mensajes recibidos y enviados.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsa en el botón de mensajes desde la vista "Cuenta".
2.- Se accederá a una nueva vista donde se recogerán los últimos diez mensajes recibidos del usuario, poniendo con letra azul y en negrita los mensajes que no se hayan leído. En la última fila, el usuario podrá pulsar en "Mostrar más mensajes" para que se recojan los siguientes diez mensajes más recientes. Si hubiera menos de diez, se cargaría la cantidad de mensajes que se recojan de igual manera.
3.- Si el usuario arrastra la tabla hacia abajo, cargará de nuevo la tabla recogiendo los diez últimos mensajes recibidos.
4.- Se mostrará el asunto, la fecha y hora que se escribió el mensaje y el autor del mismo.
5.- Si el usuario pulsa en la pestaña "Enviados", se recogerá los diez mensajes más actuales que el usuario haya enviado. De igual manera que antes, si pulsa en la última celda se cargarán los siguientes diez, y así sucesivamente. Se mostrará la misma información que antes.
6.- Estando en cualquiera de las dos pestañas, si quiere ver un mensaje pulsará sobre la celda del mensaje que quiera ver y se abrirá una nueva vista a la que el sistema enviará la información de ese mensaje.
7.- Mostrará la información de antes y a su vez el texto del mensaje. Si el mensaje ya estaba leído, no pasará nada, pero si no estaba leído, se actualizará el estado a "leído" en consecuencia.
8.- El usuario tendrá la oportunidad de borrar o responder el mensaje que está viendo sólo en el caso de haberlo recibido.
Postcondiciones: El usuario podrá ver los mensajes que ha recibido o enviado, pudiendo leerlos en detalle. Si el mensaje no ha sido leído previamente, se cambiará a leído en el caso de haberlo seleccionado.
Interfaz gráfica:

Ilustración 103.- Interfaz-66

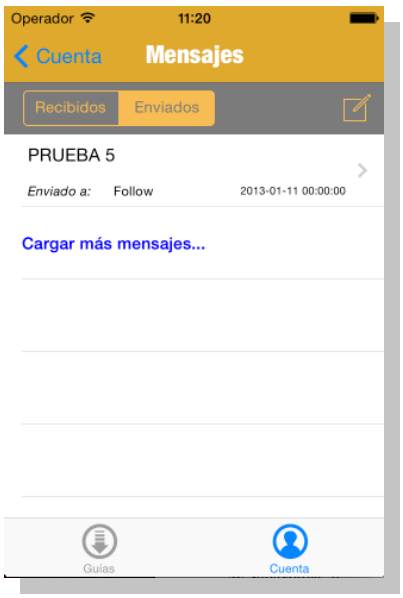


Ilustración 104.- Interfaz-67

Ilustración 105.- Interfaz-68

Nombre: Borrar Mensajes.
Descripción: Permite a un usuario borrar los mensajes recibidos.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión y haber recibido al menos un mensaje.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- En la vista de "Mensajes", una vez cargados los mensajes en la tabla se podrá borrar los mensajes de dos maneras.
1.1.- Si el usuario desliza con el dedo de derecha a izquierda, aparecerá un botón de "Delete" rojo que pulsará y borrará el mensaje de la tabla, así como la información de ese mensaje.
1.2.- Si el usuario entra en un mensaje concreto, y en la vista que se abre pulsa el botón de borrado, toda la información del mensaje se borrará, se volverá a la vista anterior y la tabla quedará actualizada sin el mensaje borrado.
1.3.- En ambos casos, si el mensaje no ha podido ser borrado, mostrará un mensaje de error.
Postcondiciones: El mensaje borrado quedará eliminado así como de la tabla de mensajes del usuario.
Interfaz gráfica:

Ilustración 106.- Interfaz-69

Nombre: Enviar Mensaje.
Descripción: Permite a un usuario escribir mensajes a otros usuarios.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- Se accederá a la vista de "Escribir mensaje".
2.- Dependiendo de dónde se haya accedido mostrará cierta información o no.
2.1.- Si el usuario ha pulsado en "Responder a usuario" en la caja de texto perteneciente a la persona que se le va a enviar, mostrará el *nick* de ese usuario. Así mismo, en el asunto aparecerá el nombre del asunto del mensaje recibido por el usuario con el texto "RE:" delante.
2.2.- Si el usuario ha pulsado en el botón de "Escribir mensaje" desde la vista del perfil de un usuario, se mostrará únicamente el nombre del usuario al que se va a mandar el mensaje en la caja de texto de "destinatario".
2.3.- Si el usuario ha pulsado el botón de "Escribir mensaje" desde la vista donde está la tabla de los mensajes del usuario, creará un mensaje vacío.
3.- El usuario rellenará las cajas de texto de "destinatario", "asunto" y "texto del mensaje" y pulsará enviar.
3.1.- Si el usuario no ha escrito un destinatario, asunto o texto en el mensaje, se mostrará un mensaje de error.
3.2.- Si el usuario ha escrito su nombre de usuario en destinatario, se mostrará un mensaje de error.
3.3.- Si el usuario al que está escribiendo el mensaje no existe, saldrá un mensaje de error.
4.- Si se ha escrito el mensaje correctamente, se guardará en la BD externa. Si hubiera algún problema con la conexión, saldrá un mensaje de error, si no, se volverá a la vista anterior.
Postcondiciones: El mensaje ha sido enviado con éxito y almacenado como no leído.
Interfaz gráfica:

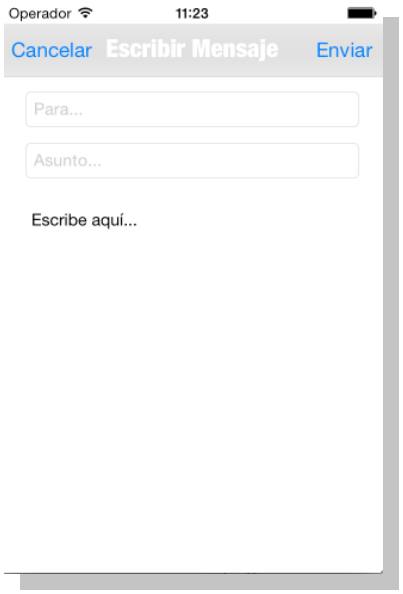


Ilustración 107.- Interfaz-70

Nombre: Consultar creaciones.
Descripción: El usuario puede consultar sus creaciones, ya sean lugares, rutas, fotos o vídeos.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno
Flujo de eventos:
1.- El usuario accederá al apartado de la tabla de "Ver Creaciones". Puede ser de dos maneras: las propias del usuario desde la vista "Cuenta" o las de otro usuario desde el perfil del usuario.
2.- Una vez pulsado en la celda, se accederá a otra vista donde se mostrará un listado de las creaciones que ha hecho el propio usuario o el de otro usuario, dividido en cuatro secciones: "Lugares", "Rutas, "Fotos" y "Vídeos.
3.- Se recogerán los datos y se mostrará en todos los casos el nombre de la "creación" y su puntuación. En los Lugares, se mostrará la imagen del perfil de ese lugar si la tuviera, si no aparecería la imagen por defecto. Y en las fotos aparecerá en pequeño la imagen de dicha foto.
4.- Si el usuario pulsa en cualquier celda podrá acceder a la información de la creación elegida.
5.- Si pulsa en lugar, irá a la vista que mostrará la información del elemento elegido o la imagen/vídeo si se trata de fotos o vídeos.
Postcondiciones: Se mostrarán en una tabla todas las creaciones que el usuario ha creado o las que otro usuario ha creado, dando la oportunidad de acceder con más detalle a esos elementos.
Interfaz gráfica:

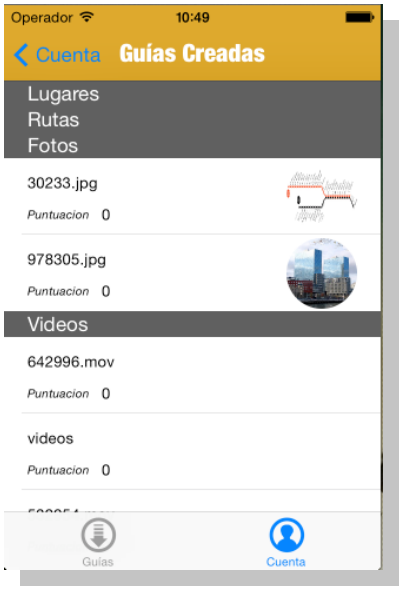


Ilustración 108.- Interfaz-71

Nombre: Borrar/editar lugar de usuario.
Descripción: El usuario puede borrar el lugar que ha creado si no hay nadie más compartiendo fotos o vídeos de ese lugar.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión. Haber creado algún Lugar.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario habrá accedido a la vista de "Descripción" del lugar que él haya creado.
2.- Se comprobará si el autor del lugar es el mismo que el *nick* utilizado al iniciar la sesión y que está almacenado en el dispositivo. Si es correcto, se habilitará el botón de editar.
3.- Pulsará sobre el botón editar y los campos de texto se habilitarán para poder modificarlos y el botón editar pasará a llamarse "Hecho" y aparecerá el botón de "Borrar".
3.1.- Si el usuario decide modificar el texto, cambiará lo que crea oportuno y pulsará "Hecho". Entonces, la información de los campos guardará para actualizar el horario, precio y descripción de ese lugar. Si fallara la actualización, se mostraría un mensaje de error y se reestablecería la información anterior.
3.2.- Si decidiera borrar el lugar, pulsaría el botón de "Borrar". Aparecerá un mensaje que si pulsa "No" se detendrá el borrado, si no, se comprobará que no hay ninguna foto o vídeo que otro usuario haya podido compartir en ese lugar.
3.2.1.- Si no hay nadie compartiendo alguna foto o vídeo, se comprobará si el usuario tiene alguna foto o vídeo almacenado sobre ese lugar. De ser así, se borrará toda la información y los archivos de fotos y/o vídeos del servidor.
3.2.2.- Si alguien está compartiendo algún recurso en ese lugar, aparecerá un mensaje y no permitirá el borrado de ese lugar.
4.- Si finalmente el lugar es borrado, se volverá a la vista de "Ciudades" si el usuario ha accedido desde la "Ficha del lugar" o a la tabla de creaciones si ha accedido desde la tabla de sus creaciones.
5.- Así mismo, se actualizará la puntuación y la cantidad de lugares de la ciudad a la que corresponda el lugar borrado.
Postcondiciones: Se habrá borrado el lugar que había creado el, así como las fotos y vídeos que pertenecieran a ese lugar. Finalmente, la ciudad de ese lugar quedará actualizada en la BD externa recalculando la puntuación y la cantidad de lugares de la ciudad.
Interfaz gráfica:

Ilustración 109.- Interfaz-72

Nombre: Borrar Recurso de Usuario.
Descripción: El usuario podrá borrar las fotos o los vídeos que haya subido al servidor.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión. Haber subido alguna foto o vídeo al servidor.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accederá a la vista donde se muestra la imagen o si lo que quiere borrar es el vídeo, a la vista donde puede ver, descargar y votar el vídeo.
2.- Se comprobará si el autor del recurso es el mismo que el *nick* utilizado al iniciar la sesión. Si es correcto, se habilitará el botón de "Borrar".
3.- Al pulsar el botón de borrar, se comprobará si es una foto o un vídeo lo que se va a borrar.
4.- Se borrará la información del recurso así como el fichero alojado en el servidor.
5.- Si el recurso pertenecía a una ciudad, se actualizará la puntuación y la cantidad de fotos o vídeos de la ciudad. Pero si el recurso pertenecía a un lugar, se actualizará la puntuación y la cantidad de fotos o vídeos de ese lugar, y así mismo, la puntuación de la ciudad.
6.- Si ha habido algún problema aparecerá un mensaje de error.
Postcondiciones: La información y el fichero del recurso quedarán borrados del servidor.
Interfaz gráfica:

Ilustración 111.- Interfaz-73

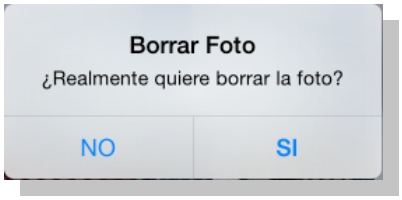


Ilustración 110.- Interfaz-74

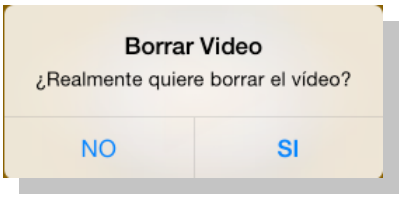


Ilustración 112.- Interfaz-76

Ilustración 113.- Interfaz-75

Nombre: Borrar/editar ruta de usuario.
Descripción: El usuario podrá borrar o editar las rutas que haya creado.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión. Haber creado al menos una ruta.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accederá a la vista donde se mostrará la información de la ruta.
2.- Se comprobará si el autor de la ruta es el mismo que el *nick* utilizado al iniciar la sesión. Si es correcto, se habilitará el botón de editar.
3.- Al pulsar el botón editar, cambiará de nombre a "Hecho" que permitirá finalizar la edición, habilitando las cajas de texto. Así mismo, aparecerá el botón de borrar.

3.1.- Si el usuario decide editar la información, escribirá lo que quiera modificar y pulsará en "Hecho". La información de la ruta se recogerá y se modificará. Si hubiera algún problema, aparecería un mensaje de error. El botón "Hecho" cambiará a editar y desaparecerá el botón borrar.

3.2.- Si el usuario decide borrar la ruta, pulsará en el botón borrar y aparecerá un mensaje para confirmar. Si pulsa que sí, se borrará el registro de esa ruta, actualizando la cantidad de rutas y la puntuación de la ciudad a la que pertenece. Si hubiera algún problema, aparecerá un mensaje de error, si no, se volverá a la vista anterior y la ruta se borrará de la tabla de la que se ha accedido.

Postcondiciones: La información de la ruta quedará borrada y la puntuación, al igual que la cantidad de rutas, estará actualizada en el registro de la ciudad.
Interfaz gráfica:

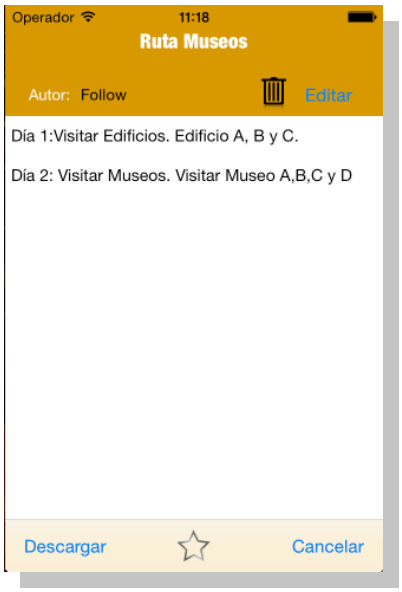


Ilustración 115.- Interfaz-77

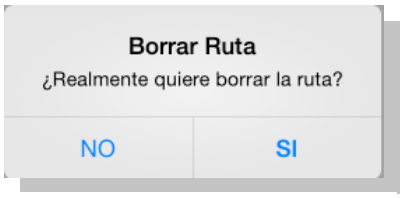


Ilustración 114.- Interfa-78

Nombre: Gestionar Amigos.
Descripción: Permite seguir o dejar de seguir a otros usuarios o pedirles una petición de amistad.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- Si el usuario pulsa el botón de *seguimiento* con el nombre "Siguiendo", se borrará el registro de la amistad. Si ocurre un error, aparecerá un mensaje de error, si no, el botón cambiará a "Seguir".
2.- Si el usuario pulsa el botón seguimiento con el nombre "Petición", se mostrará un mensaje de confirmación. Si pulsa que sí, se borrará la relación de amistad. Si ocurre un error, aparecerá un mensaje de error, si no, el botón cambiará a "Seguir".
3.- Si el usuario pulsa el botón seguimiento con el nombre "Seguir", dependerá de si el usuario tiene el perfil bloqueado o no.
3.1.- Si lo tiene bloqueado se creará un registro de amistad con estado pendiente. Si ocurre un error, aparecerá un mensaje de error, si no, el botón cambiará a "Petición".
3.2.- Si no, se creará un registro de amistad. Si ocurre un error, aparecerá un mensaje de error, si no, el botón cambiará a "Siguiendo".
4.- Si el usuario pulsa el botón con nombre "Aceptar", se quitará el usuario de la tabla y se cambiará el estado a aceptado. Si hay algún error, mostrará un mensaje.

Postcondiciones: Se habrá borrado o cambiado el estado de la amistad, o se habrá creado un registro nuevo de la amistad entre los dos usuarios.
Interfaz gráfica:

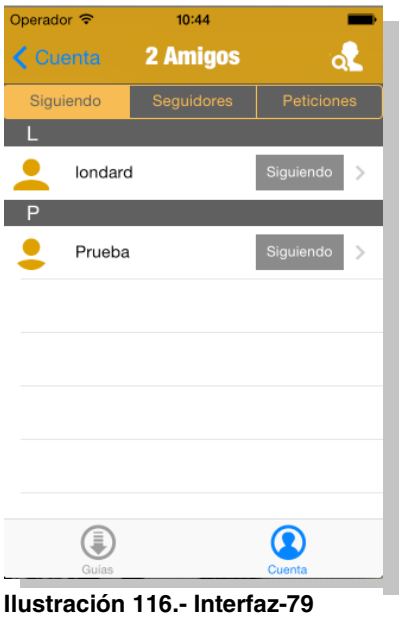


Ilustración 116.- Interfaz-79

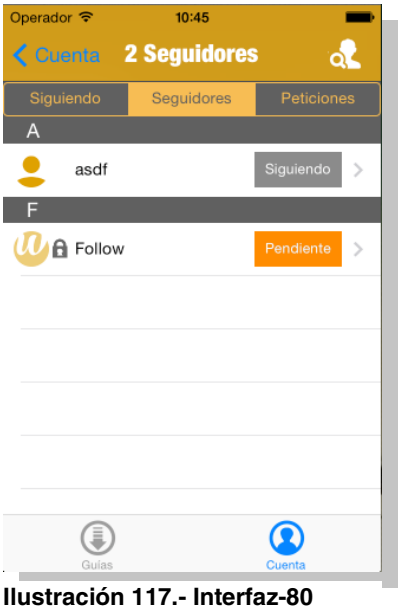


Ilustración 117.- Interfaz-80

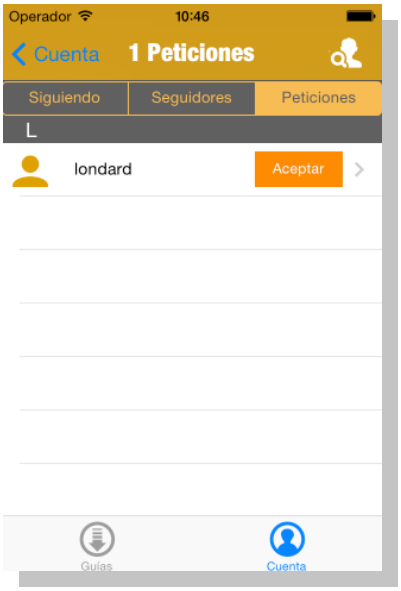


Ilustración 119.- Interfaz-81

Ilustración 118.- Interfaz-82

Nombre: Puntuar/Quitar puntuación.
Descripción: El usuario podrá puntuar lugares, rutas, vídeos y fotos.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accederá a la vista dónde aparezca el botón "Votar".
1.1.- Si es un lugar, en la vista "Descripción".
1.2.- Si es una foto, en la vista de esa foto.
1.3.- Si es un vídeo, en la vista para ver el vídeo.
1.4.- Si es una ruta, en la vista "Descripción" de la ruta.
2.- En cualquier caso, al darle a votar, se comprobará si es un Lugar, Ruta, Foto o Vídeo y se añadirá un registro de la votación que ha hecho el usuario.
3.- La puntuación se actualizará dependiendo de lo que se haya votado.
3.1.- Si es un lugar, se actualizará la puntuación del lugar y de la ciudad a la que pertenezca.
3.2.- Si es una ruta, se actualizará la puntuación de la ruta y de la ciudad a la que pertenezca.
3.3.- Si es una foto o vídeo, se comprobará si pertenece a una ciudad o a un lugar.
3.3.1.- Si pertenece a una ciudad, se actualizará la puntuación de la foto o vídeo y de la ciudad a la que pertenezca.
3.3.2.- Si pertenece a un lugar, se actualizará la puntuación de la foto o el vídeo y del lugar al que pertenezca. Y a su vez, la puntuación de la ciudad tendrá que ser actualizada también.
3.4.- Si en cualquier caso se produjera un error, aparecería una alerta.
4.- Si todo ha ido bien, el botón de votar cambiará a "Votado".
5.- Si el usuario pulsa el botón de nuevo, se repetirá el proceso desde el punto 2, sólo que esta vez se borrará el registro de la votación en la BD externa. Si algo sale mal, aparecerá un mensaje de error.

Postcondiciones: Se habrá creado un registro de la puntuación que ha hecho el usuario. Se habrá actualizado, así mismo, las puntuaciones de la ciudad o del lugar que tengan relación con lo votado.
Interfaz gráfica:

Ilustración 120.- Interfaz-83

Ilustración 121.- Interfaz-84

Nombre: Consultar puntuaciones.
Descripción: El usuario podrá consultar las puntuaciones que ha hecho o las de otro usuario.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario tendrá dos maneras de acceder a la vista de las votaciones. Podrá comprobar las suyas o las de otro usuario.
2.- Si accede desde la vista "Cuenta", mostrará la tabla con las puntuaciones que ha hecho y que han sido registrados, diferenciando en secciones si dicha votación pertenece a un lugar, ruta, foto o vídeo.
3.- Si accede desde el perfil de otro usuario, se mostrará de igual manera las votaciones que ha hecho el usuario que está visualizando. Sin embargo, si el usuario tiene el perfil bloqueado, a menos que el usuario le esté siguiendo, no podrá ver las votaciones que ha hecho. De ser así, tendrá que pedirle amistad y que éste acepte.

Postcondiciones: Se habrá mostrado por pantalla todos los objetos que el usuario haya votado. Si el usuario está mirando las votaciones de otro usuario, se mostrarán las votaciones de ese usuario.
Interfaz gráfica:

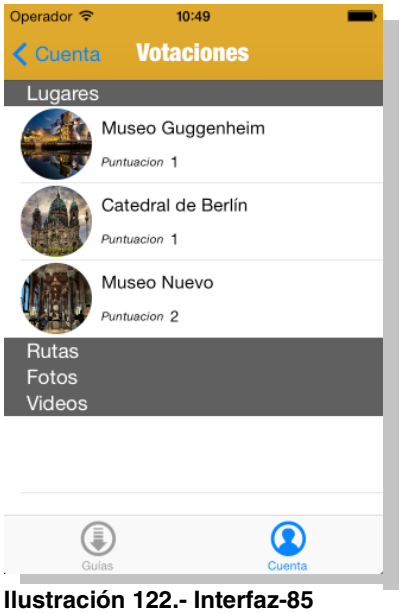


Ilustración 122.- Interfaz-85

Nombre: Borrar usuario.
Descripción: El usuario borrará su cuenta y toda su información.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario habrá accedido a la vista donde se muestra toda su información.
2.- Pulsará el botón “Borrar cuenta”.
3.- Se le preguntará si realmente quiere borrar la cuenta.
4.- Si pulsa no, no ocurrirá nada. Si pulsa *sí,* se borrará toda la información del usuario.
5.- Así mismo, se borra las credenciales guardadas en el dispositivo al inicar la sesión.
6.- Se cerrará la vista y se volverá a la pantalla principal.

Postcondiciones: Se habrá borrado toda la información del usuario.
Interfaz gráfica:

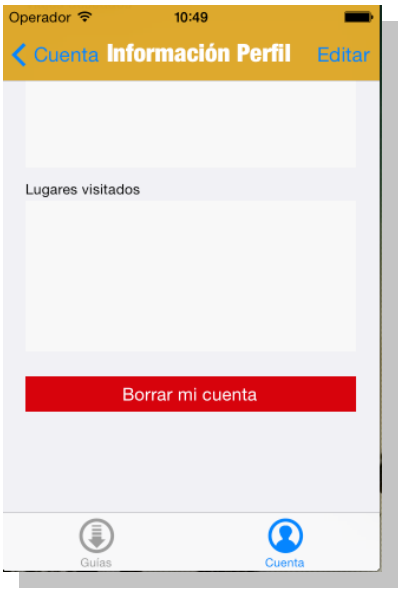


Ilustración 123.- Interfaz-86

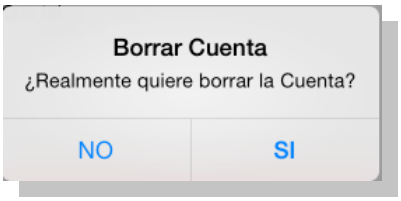


Ilustración 124.- Interfaz-87

Nombre: Buscar usuario.
Descripción: El usuario podrá buscar a un usuario por su nick y acceder a su perfil. Además, se mostrarán las sugerencias de otros usuarios que más han colaborado y compartido sus guías con la aplicación.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- Desde la vista de "Amigos", pulsará en el botón de buscar y se accederá a la nueva vista.
2.- Se recogerá las sugerencias de usuarios que hayan creado más de cuatro guías (lugares, rutas, fotos o vídeos).
3.- Si aparece el usuario que ha iniciado sesión, el botón de seguir estará oculto.
4.- Si elige "Buscar", pulsará sobre el botón y se mostrará la barra de buscar. Entonces, el usuario introduce un nombre y se consulta si existe. Si existe, se devuelve la información de ese usuario y se muestra en la tabla y sino la tabla quedará vacía mostrando en la cabecera que "No se ha encontrado ningún resultado".
5.- Si pulsa sobre el usuario, se accede a la vista del perfil del usuario buscado.

Postcondiciones: Se mostrarán las sugerencias de usuarios y los usuarios buscados mediante el buscador, en la tabla de la vista.
Interfaz gráfica:

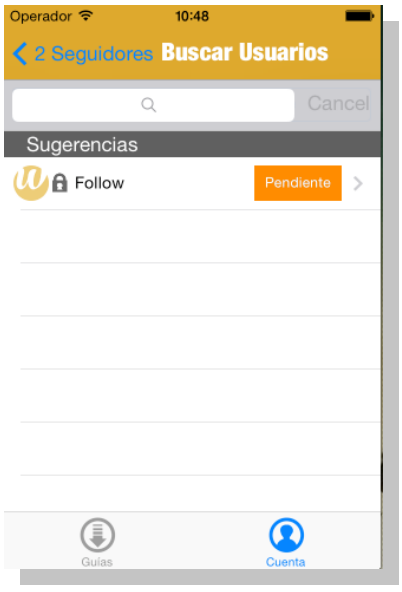


Ilustración 125.- Interfaz-88

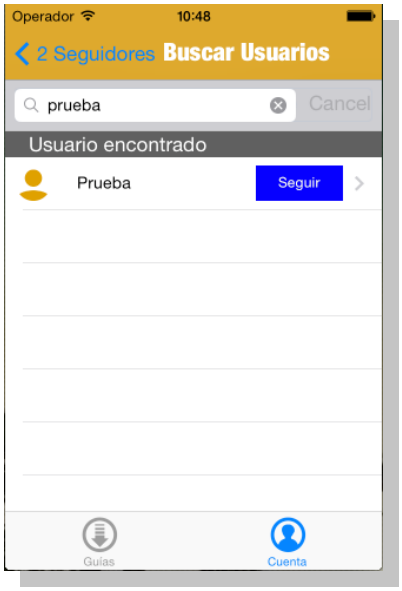


Ilustración 126.- Interfaz-89

Nombre: Consultar amigos.
Descripción: Se mostrarán las amistades del usuario, tanto los que sigue, los que le siguen y los pendientes por aceptar, si los hubiera.
Actores: Usuario Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accederá al apartado de "Amigos" desde cuenta.
2.- Se recogerá de la BD externa las personas a las que el usuario sigue, mostrando en la tabla el nombre, un candado si tuviera su perfil bloqueado, la imagen de perfil si tuviera (sino la imagen por defecto) y un botón con el nombre "Siguiendo".
3.- Si el usuario pulsa la pestaña de "Seguidores", se accederá a la BD externa para recoger la información de los usuarios que están siguiendo al usuario. Se mostrará la misma información que antes. Sin embargo, se comprueba si el usuario está siguiendo a sus seguidores.
3.1.- Si el usuario sigue a esa persona, el botón aparecerá como "Siguiendo".
3.2.- Si el usuario no lo sigue, aparecerá como "Seguir".
3.3.- Si el usuario ha mandado una petición de seguimiento, debido a que el otro usuario tiene el perfil bloqueado, aparecer el botón como "Petición".
4.- Si el usuario pulsa la pestaña "Peticiones", se cogerán las amistades pendientes por aceptar que tenga, en el caso de que tenga el perfil bloqueado y alguna petición de otro usuario.
5.- Si el usuario decide acceder a ver el perfil del usuario, podrá gestionar de igual manera la amistad con ese usuario mediante el botón de seguimiento que aparecerá en la nueva vista del "Perfil del Usuario", a la cual se le enviará toda la información del usuario elegido.

Postcondiciones: Se mostrará todas las amistades del usuario y además, al pulsar sobre un usuario, se mostrará su foto de perfil y su información de perfil.
Interfaz gráfica:

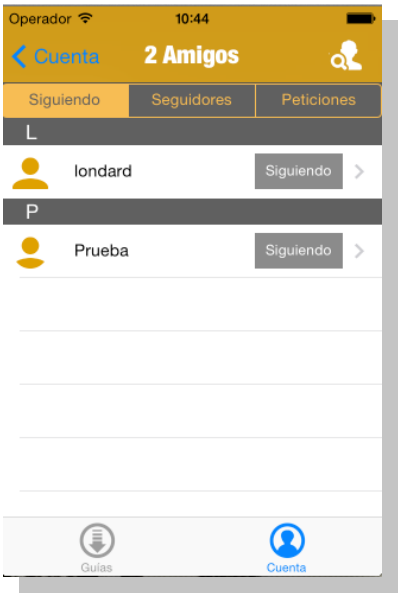


Ilustración 128.- Interfaz-90

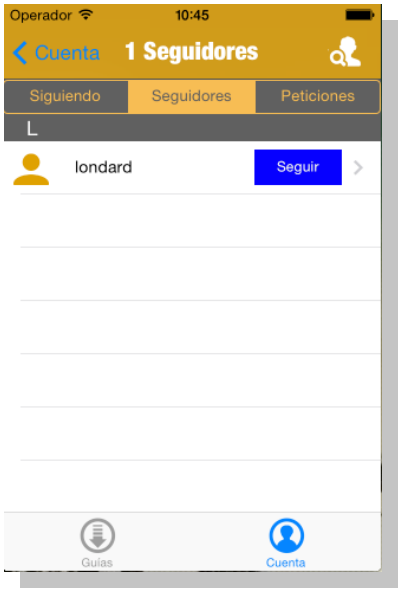


Ilustración 127.- Interfaz-91

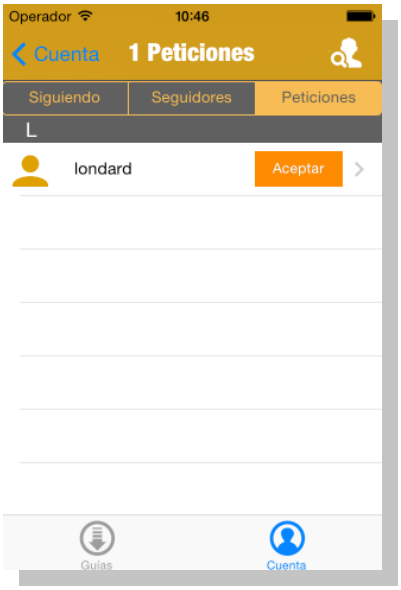


Ilustración 130.- Interfaz-92

Ilustración 129.- Interfaz-93

Nombre: Consultar comentarios.
Descripción: Se mostrarán los comentarios de un lugar.
Actores: Usuario.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accede a la vista “Comentarios” desde la ficha del lugar.
2.- Se recogerá todos los comentarios que los usuarios hayan hecho de ese lugar ordenados de los más recientes a los más antiguos.
3.- Se mostrará el texto del comentario con la fecha, un título y el autor.
Postcondiciones: Los comentarios aparecerán en la tabla ordenados de más nuevos a más antiguos.
Interfaz gráfica:

Ilustración 131.- Interfaz-94

Nombre: Consultar rutas.
Descripción: Se mostrará las rutas de la ciudad, tanto las descargadas como las que no.
Actores: Usuario.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accederá al apartado de “Rutas” desde la ficha de una ciudad.
2.- Se recogerá las rutas que el usuario se haya descargado ordenados por los días en que se tarda en recorrer la ruta.
3.- Si el usuario pulsa la pestaña de “Descargas”, se recogerá la información de las rutas que hay disponibles. Se mostrará la información ordenada por los días que se tarda en recorrer la ruta.
4.- Si el usuario pulsa la pestaña de “Más votadas” se mostrará las cinco rutas más votadas de esa ciudad.
5.- En la pestaña “Descargas” y “Más votadas” se permitirá la descarga de las rutas pulsando en el botón de la celda.
6.- Si pulsa sobre una ruta, se le enviará a una nueva vista toda la información de la ruta para mostrarla por pantalla.
Postcondiciones: Se mostrarán las rutas ordenadas por días. Al pulsar sobre una ruta, aparecerá toda la información detalladamente.
Interfaz gráfica:

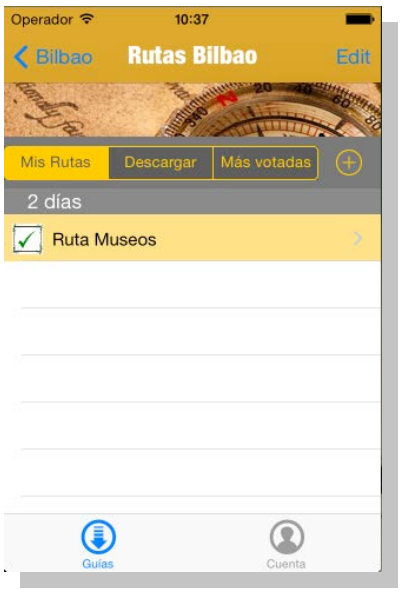


Ilustración 132.- Interfaz-95

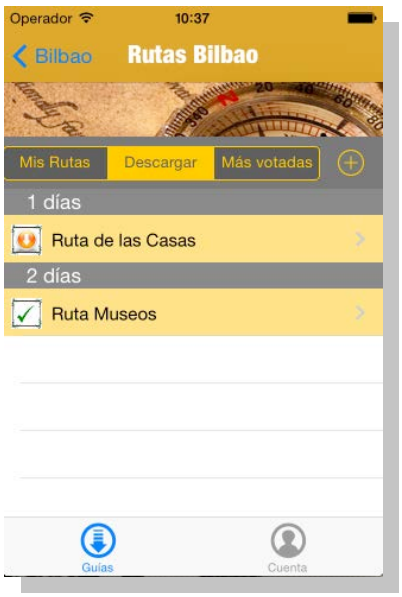


Ilustración 133.- Interfaz-96

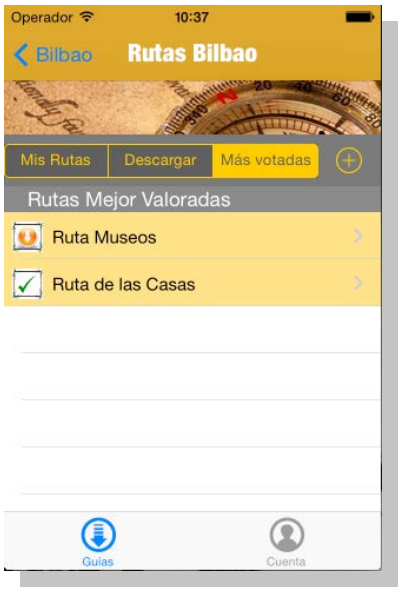


Ilustración 134.- Interfaz-97

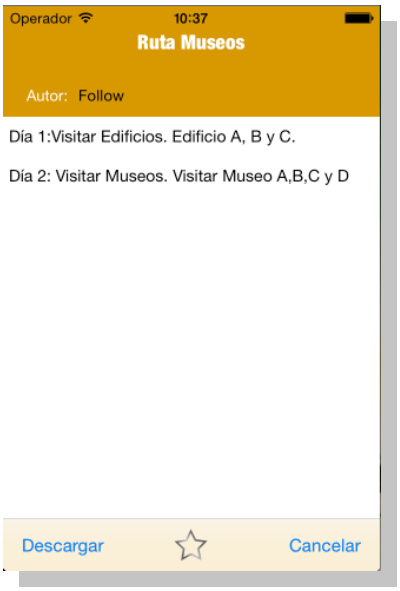


Ilustración 135.- Interfaz-98

Nombre: Crear guía.
Descripción: Permite crear una guía completa en la que se puede crear una ciudad, lugares, rutas, subir fotos, vídeos y el metro de la ciudad.
Actores: Registrado.
Precondiciones: Conexión a Internet. Haber iniciado sesión.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario pulsa en el botón para crear una guía desde la vista "Mis Guías".
2.- Aparecerá una nueva vista de cinco pasos con un texto en el que explicará al usuario qué tiene que hacer.
3.- Al pulsar en siguiente, el usuario tendrá que introducir el nombre de la ciudad y el país al que pertenece obligatoriamente, sino no podrá avanzar. También puede seleccionar una foto suya para poner como perfil de la ciudad.
4.- Pulsa siguiente y se comprueba si la ciudad existe o no. Si no existe, se muestra una nueva vista con un botón para crear lugares. Si existe, se mostrará un aviso. Para poder continuar, al menos tendrá que crear un lugar.
5.- Cuando haya creado uno o más lugares, pulsará en "siguiente".
6.- Se registrará la información tanto de la ciudad como de los lugares creados.
7.- A continuación, en el nuevo apartado, se mostrará varios botones en los que el usuario podrá subir una foto, un vídeo o el mapa del metro, así como crear rutas.
7.1.- Si pulsa en foto o vídeo, se mostrará una nueva vista en la que tendrá que elegir entre la ciudad o los lugares creados anteriormente. Una vez que lo elija, pulsará en seleccionar y se mostrará la vista para subir una foto o un vídeo.
8.- El usuario pulsará en finalizar y se cerrará la vista, volviendo a la anterior.
Postcondiciones: Se habrá creado una guía completa y toda la información introducida habrá sido registrada para su futura consulta.
Interfaz gráfica:

Ilustración 137.- Interfaz-99

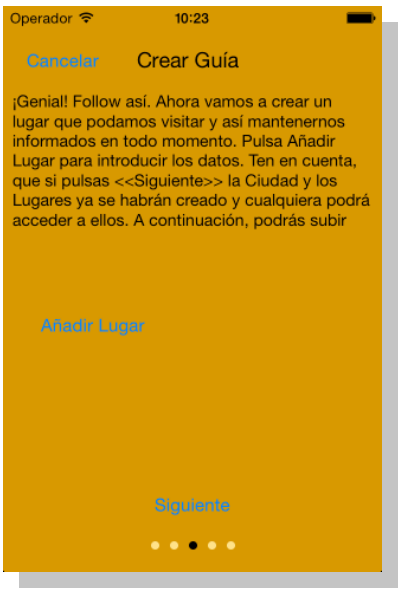


Ilustración 136.- Interfaz-100

Ilustración 144.- Interfaz-102

Ilustración 145.-Interfaz-101

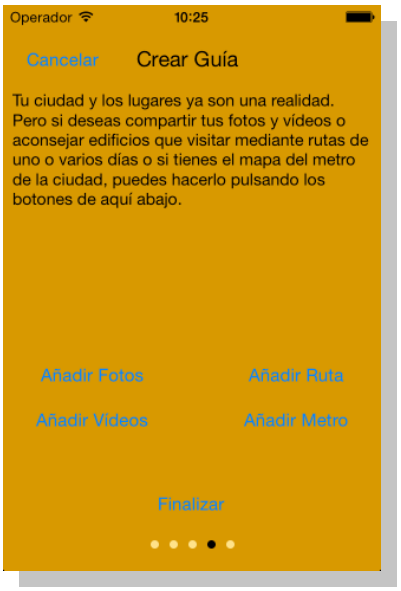


Ilustración 143.- Interfaz-103

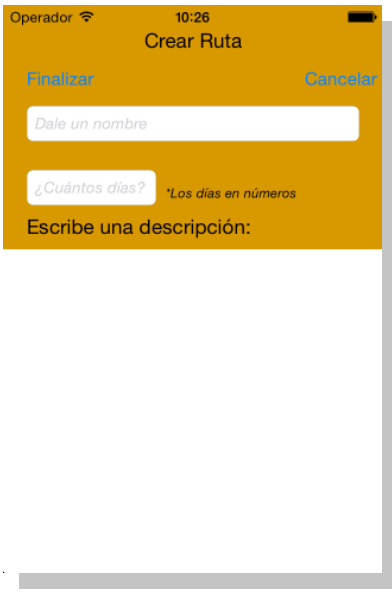


Ilustración 141.- Interfaz-104

Ilustración 142.- Interfaz-105

Ilustración 140.- Interfaz-106

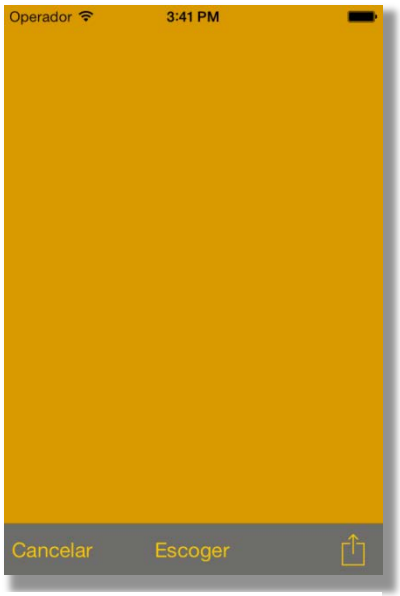


Ilustración 139.- Interfaz-107

Ilustración 138.-Interfaz-108

Nombre: Descargar metro.
Descripción: Permite descargar el mapa del metro.
Actores: Usuario.
Precondiciones: Conexión a Internet.
Requisitos no funcionales: Ninguno.
Flujo de eventos:
1.- El usuario accede a la vista del metro desde la ficha de la ciudad, donde se le envía la dirección del metro.
2.- Se mostrará la imagen en la nueva vista accediendo al archivo alojado en el servidor.
3.- El usuario pulsa en el botón “descargar” y se comprueba si tiene otra información descargada en la ciudad.
3.1.- Si tiene descargada parte de la información que tenga la ciudad, se actualizará la información guardada con la dirección de la foto del metro donde se ha guardado el archivo en el dispositivo.
3.2.- Si no, se creará un registro nuevo con la dirección del metro que señalará dónde está alojado el archivo en el dispositivo.
4.- Si hay algún error, se mostrará una alerta.
Postcondiciones: Se habrá creado el fichero de la imagen del metro en el dispositivo y se habrá guardado su dirección.
Interfaz gráfica:

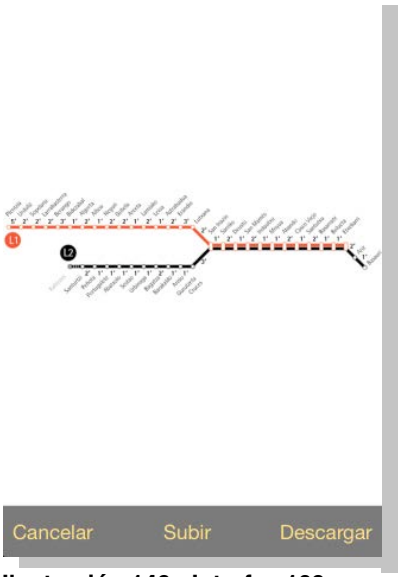


Ilustración 146.- Interfaz-109

Nombre: Subir metro.
Descripción: Permite al usuario subir una foto del metro de una ciudad.
Actores: Registrado.
Precondiciones: Conexión a Internet y estar registrado en el sistema.
Requisitos no funcionales: El archivo no puede ser mayor de 1MB.
Flujo de eventos:
1.- El usuario irá al apartado de metro de la ciudad.
2.- Pulsará el botón "Subir".
3.- Aparecerá la vista del carrete de fotos y le dará la opción de elegir el álbum de fotos que quiera. Una vez elegido, elegirá la foto que él crea conveniente y pulsará sobre ella.
4.- La foto aparecerá en grande en la pantalla del punto 3. Y entonces, si pulsa el botón "Subir", que habrá cambiado de color, mostrará un mensaje para confirmar la subida de la foto. Si pulsa que sí, comenzará a subirse al servidor. Si no, se deshará la subida.
5.- Al subirse, en primer lugar se le dará un nombre aleatorio entre 1 y 999999, se guardará en el servidor y a continuación se almacenará la dirección que hace referencia en dónde está alojado.
Postcondiciones: La foto se ha subido al servidor y guardada su dirección en la ciudad.
Interfaz gráfica:

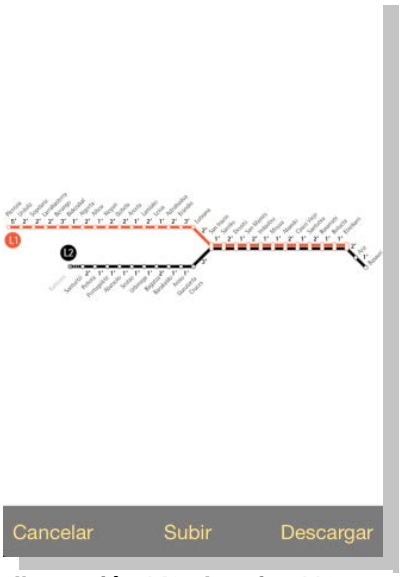


Ilustración 148.- Interfaz-110

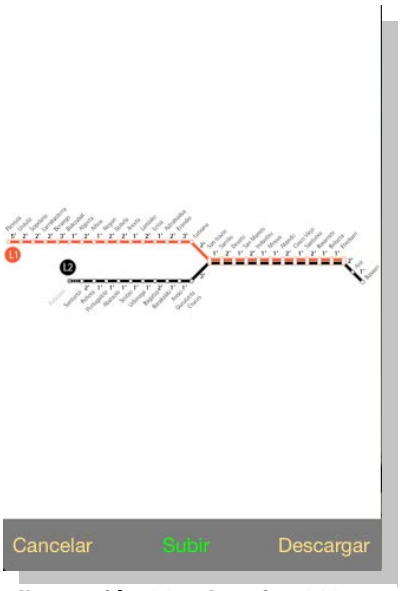


Ilustración 149.- Interfaz-111

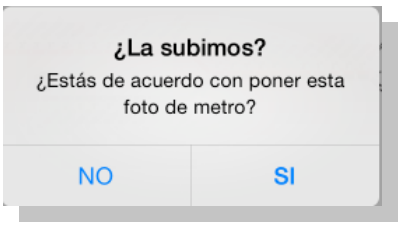


Ilustración 147.- Interfaz-112

ANEXO II: DIAGRAMAS DE SECUENCIA

1-Borrar Ciudad

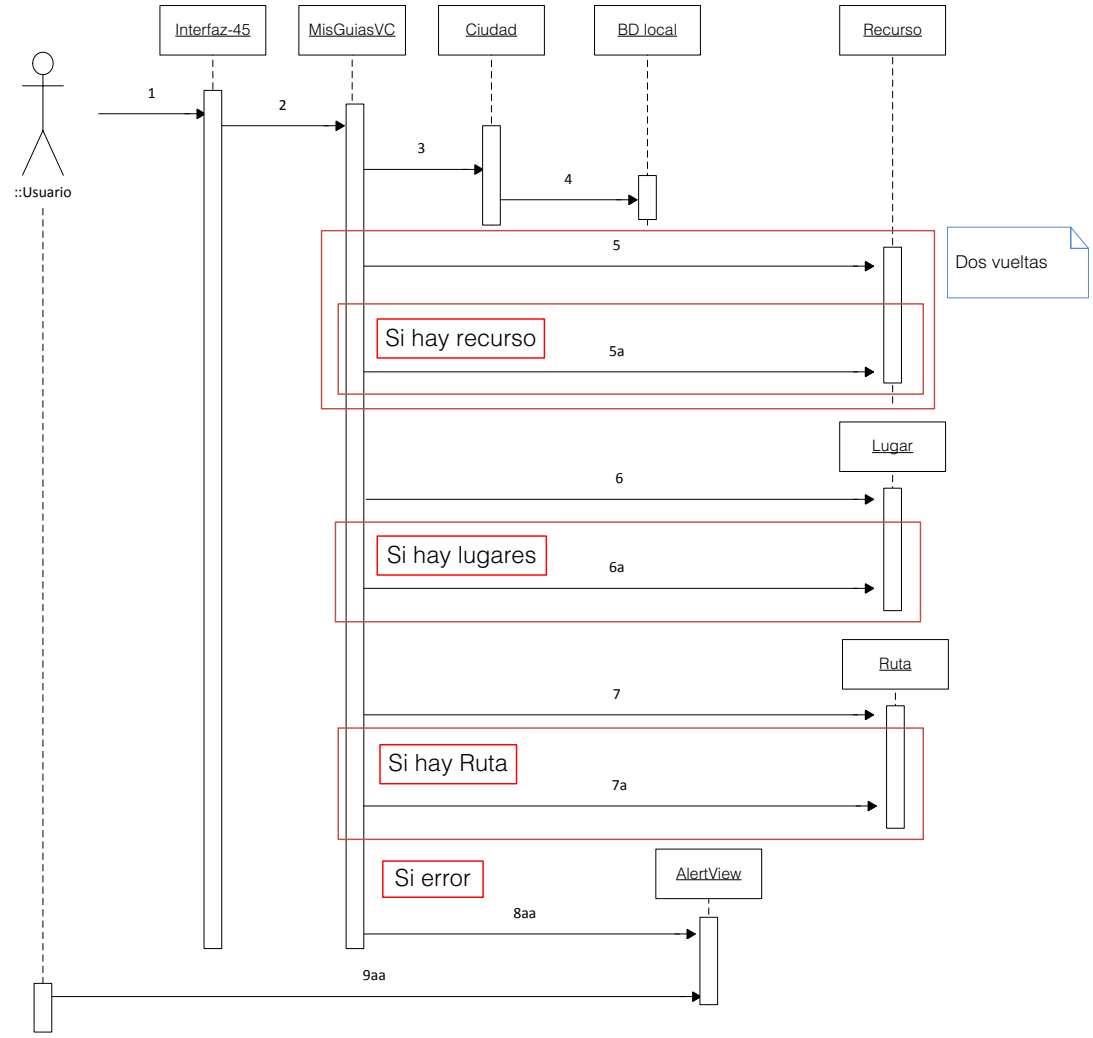


Ilustración 150.- Diagrama borrar ciudad

1- El usuario pulsa el botón borrar de la tabla en la ciudad que quiere borrar

2- borarFilas( ): void

3- eliminarPais(): void

4- execSQL("DELETE FROM Ciudades WHERE id_ciudad=%id_ciudad%")

5- consultarRecurso(id_lugar) conRecurso(siFotoOVideo): ListaRecursos

[Si hay recurso]

5a- eliminarRecurso (fotoOVideo): void

6- consultarLugares(indicador): ListaLugares

[Si hay lugar]

6a- eliminarLugar(): void

7- consultarRutas(indicador): ListaRutas

[Si hay rutas]

7a- eliminarRuta(): void

[Si error]

8aa- new()

9aa- El usuario pulsa OK.

2-Borrar Usuario

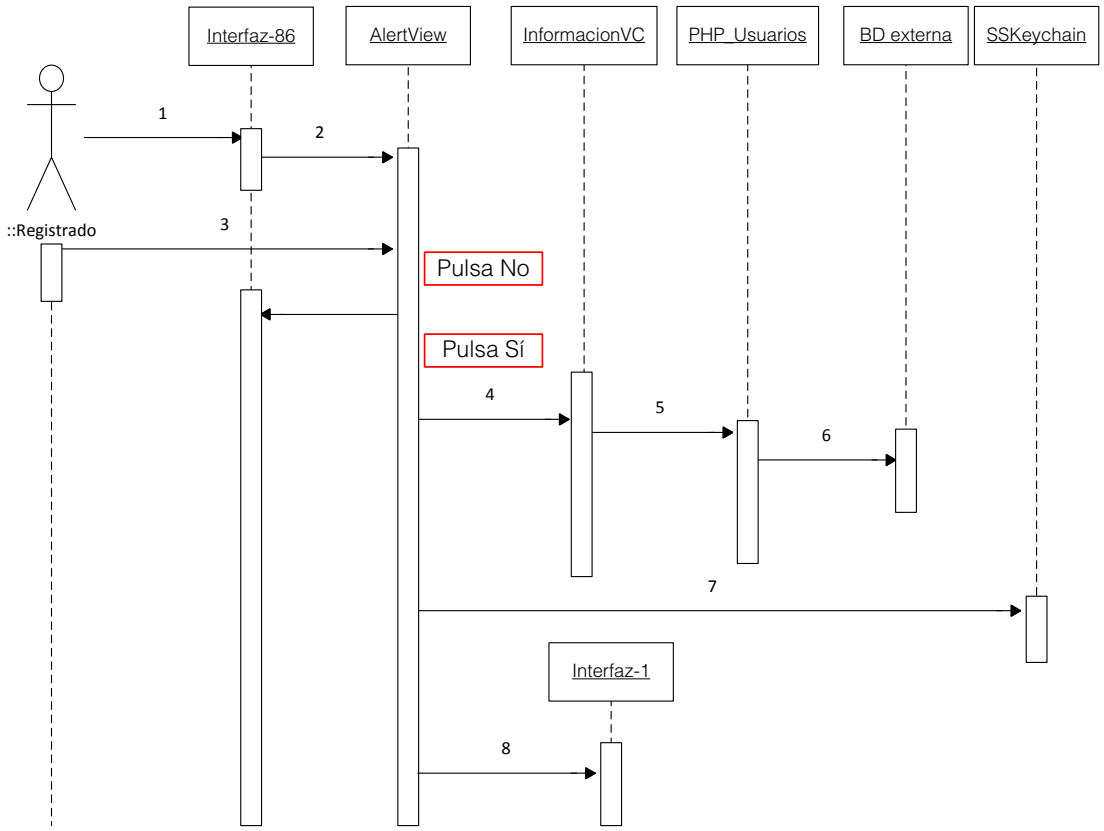


Ilustración 151.- Diagrama borrar usuario

1- El usuario pulsa el botón borrar cuenta.

2- new()

3- El usuario pulsa el botón de confirmación.

4- alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex

5- borrarUsuario(): void

6- execSQL("DELETE FROM Usuarios where id_usuario=%usu%")

7- resetKeychainItem( ): void

8- new( )

3- Borrar Foto o Vídeo

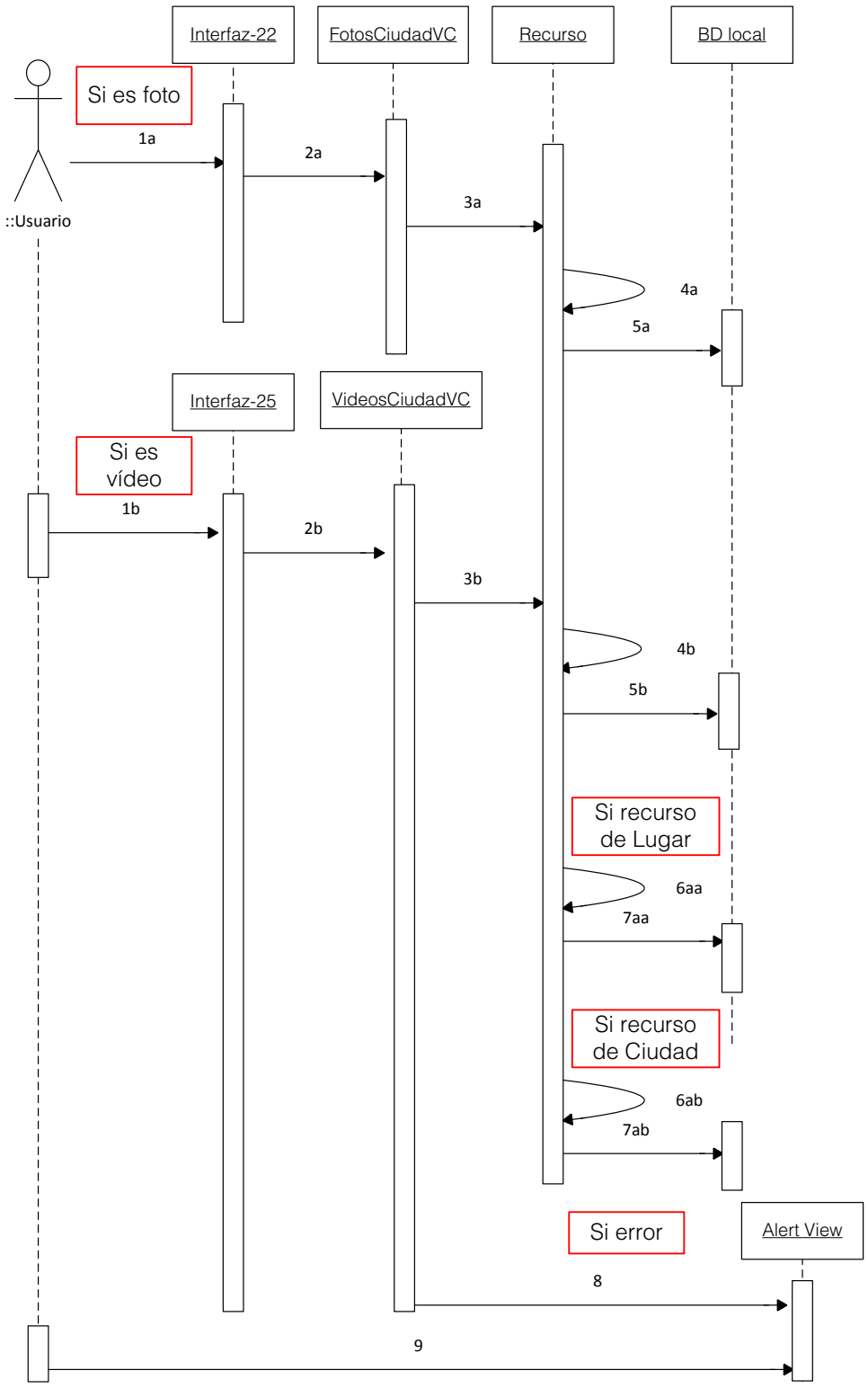


Ilustración 152.- Diagrama borrar foto o vídeo

[Si foto]

1a- El usuario pulsa el botón borrar.

2a- pulsarBorrar(sender): IBAction

3a- eliminarRecurso(fotoOVideo): void

4a- borrarArchivo(): void

5a- execSQL("DELETE FROM Fotos WHERE id_foto = %id_foto%")

[Si vídeo]

1b- El usuario pulsa el botón borrar.

2b- pulsarBorrar(sender): IBAction

3b- eliminarRecurso(fotoOVideo): void

4b- borrarArchivo(): void

5b- execSQL("DELETE FROM Videos WHERE id_video = %id_video%")

[Si recurso Lugar]

6aa- actualizarLugarDescargado(): void

7aa- execSQL("UPDATE Lugares SET fotos = (SELECT COUNT(id_foto) FROM Fotos WHERE id_lugar=%d), videos = (SELECT COUNT(id_video) FROM Videos WHERE id_lugar=%id_lugar%) WHERE id_lugar=%id_lugar%")

[si recurso de ciudad]

6ab- actualizarCiudadDescargada(): void

7ab- execSQL("UPDATE Ciudades SET lugares = (SELECT count(id_lugar) FROM Lugares WHERE id_ciudad=%id_ciudad%), fotos = (SELECT COUNT(id_foto) FROM Fotos WHERE id_ciudad=%id_ciudad%), videos = (SELECT count(id_video) FROM Videos WHERE id_ciudad=%id_ciudad%), rutas = (SELECT COUNT(nombre) FROM Rutas WHERE id_ciudad=%id_ciudad%) WHERE idc=%id_ciudad%")

[Si error]

8- new()

9- El usuario pulsa OK.

4-Borrar Lugar

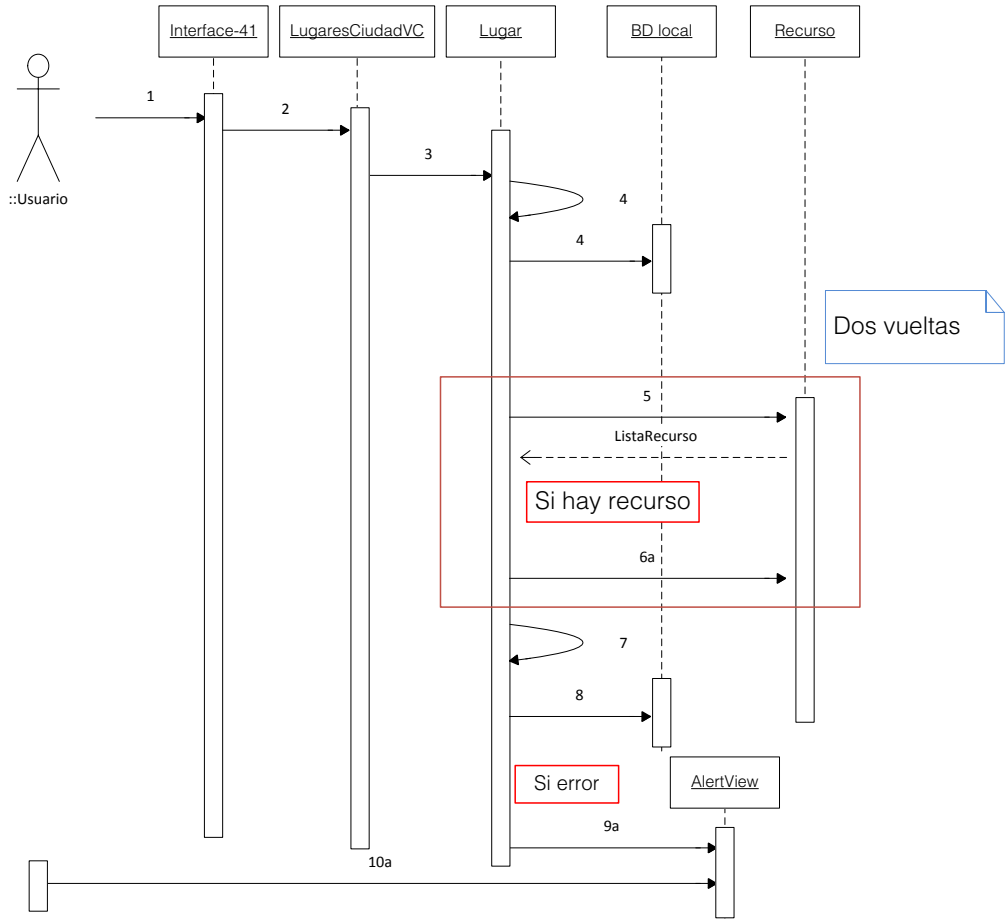


Ilustración 153.- Diagrama borrar lugar

1- El usuario pulsar el botón borrar de la tabla en el lugar que ha decido.

2- borrarFilas(): void

3- eliminarLugar(): void

4- execSQL("DELETE FROM Lugares where id_lugar = %d")

5- consultarRecurso(id_lugar) conRecurso(siFotoOVideo): ListaRecursos

[Si hay recurso]

6a- eliminarRecurso (fotoOVideo): void

7- actualizarCiudad(): void

8- execSQL("UPDATE Ciudades SET lugares = (SELECT count(id_lugar) FROM Lugares WHERE id_ciudad=%id_ciudad%), fotos = (SELECT COUNT(id_foto) FROM Fotos WHERE id_ciudad=%id_ciudad%), videos = (SELECT count(id_video) FROM Videos WHERE id_ciudad=%id_ciudad%), rutas = (SELECT COUNT(nombre) FROM Rutas WHERE id_ciudad=%id_ciudad%) WHERE idc=%id_ciudad% ")

[Si error]

9a- new()

10a- El usuario pulsa OK.

5- Borrar Lugar de Usuario

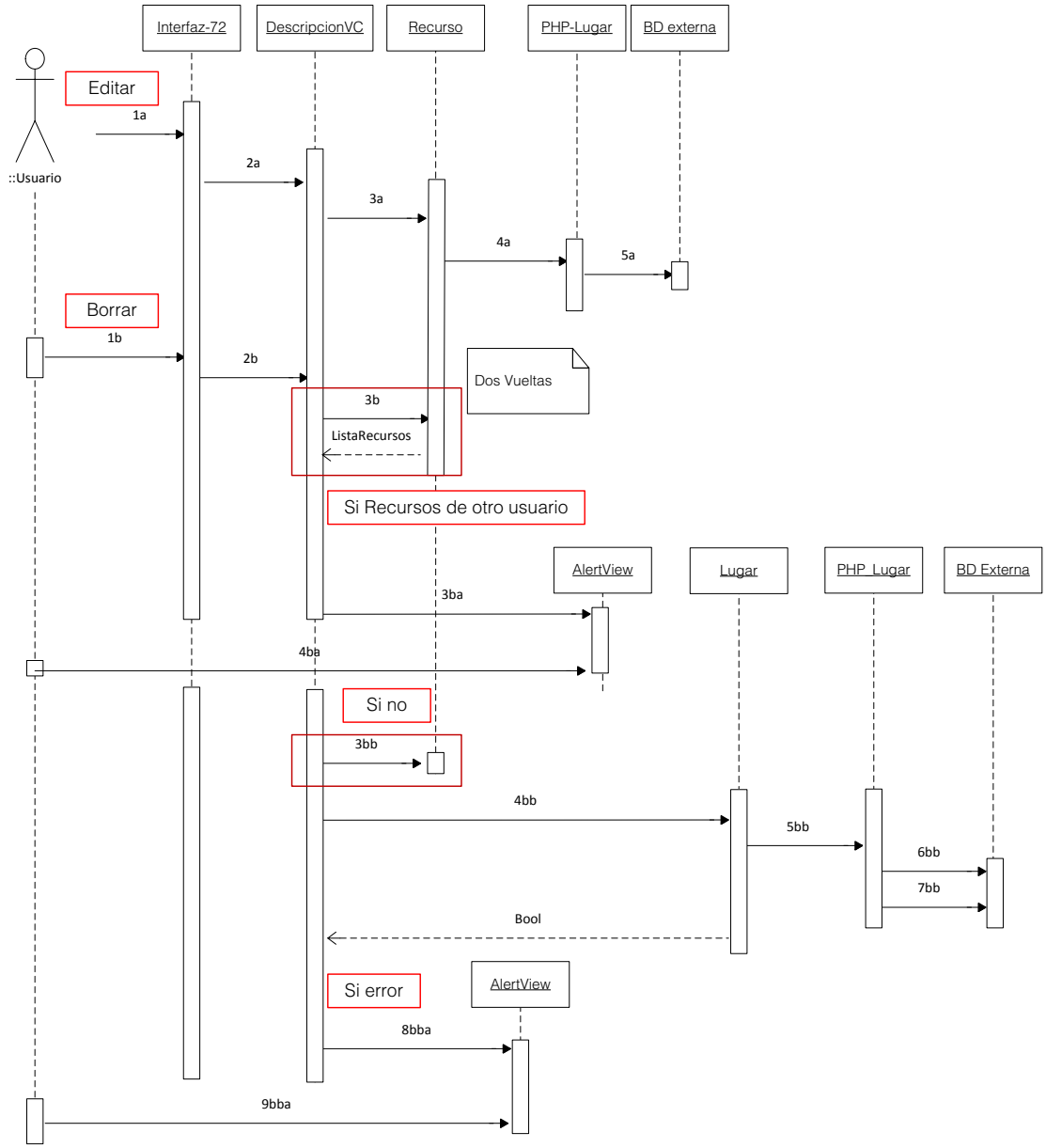


Ilustración 154.- Diagrama borrar lugar de usuario

[Si editar]

1a- El usuario, tras pulsar el botón Editar, pulsa en Hecho para actualizar la descripción del Lugar.

2a- pulsarEditar(id): IBAction

3a- actualizarLugar(): void

4a- actualizarDescripcion(función,id_lugar, descripcion, horario, precio);

5a- execSQL(UPDATE Lugares SET descripcion=%desc%, horario=%hor%, precio=%pre% WHERE id_lugar=%id_lugar%)

[Si borrar]

1b- El usuario, tras pulsar en el botón Editar, pulsa en el botón borrar.

2b- pulsarBorrar(): void

3b- consultarRecursos(id_lugar) conRecurso(siFotoOVideo): ListaRecursos

[Si recursos de otro usuario]

3ba-new()

4ba-El usuario pulsa OK.

[si no]

3bb- borrarRecursoSubido(fotoOVideo): bool

4bb- borrarLugarBD(función,id_objeto,foto): bool

5bb- borrarLugaresUsuario ()

6bb- execSQL("DELETE FROM Objetos WHERE id=%id_objeto%")

7bb- execSQL("UPDATE ciudades SET puntuacion = (SELECT (COALESCE((SELECT SUM(puntuacion) FROM Lugares WHERE id_ciudad =%id_ciudad%),0) + COALESCE((SELECT SUM(puntuacion) FROM Rutas WHERE id_ciudad =%id_ciudad%),0)+ COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0)) AS suma), lugares=(SELECT count(id_lugar) FROM Lugares WHERE id_ciudad=%id_ciudad%) WHERE idc=%id_ciudad%")

[Si error]

8bba- new()

9bba- El usuario pulsa OK.

6-Borrar Mensaje

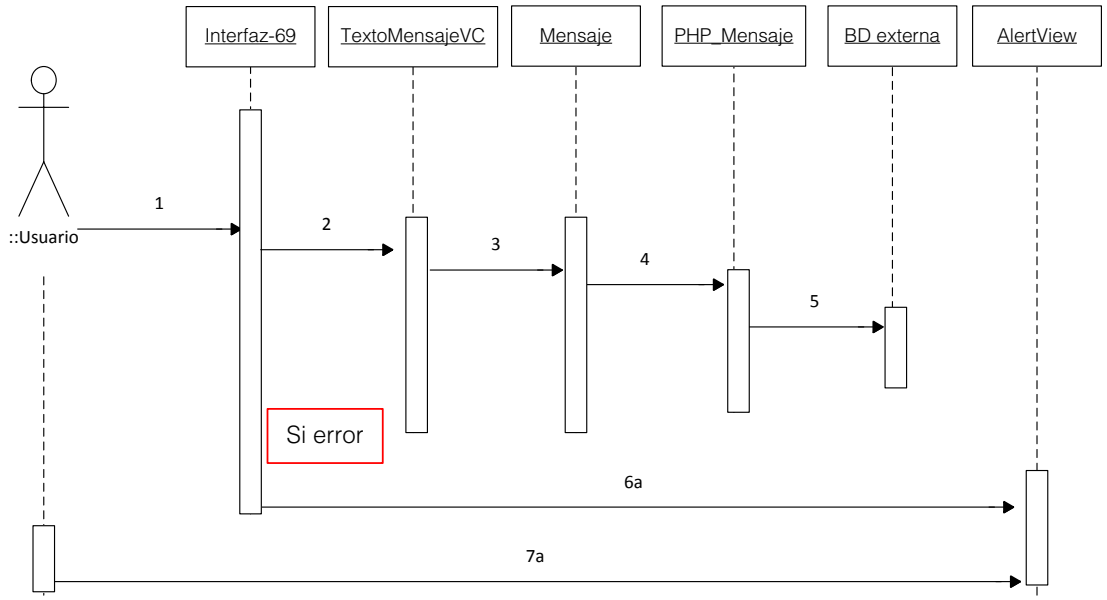


Ilustración 155.- Diagrama borrar mensaje

1- El usuario pulsa sobre el botón borrar.

2- pulsarBorrar(id): IBAction

3- borrarMensaje(): void

4- borrarMensaje(id_mensaje, funcion): void

5- execSQL("DELETE FROM Mensajes WHERE id_mensaje=%id%")

[Si error]

6a- new()

7a- El usuario pulsa OK.

7- Borrar Recurso de Usuario

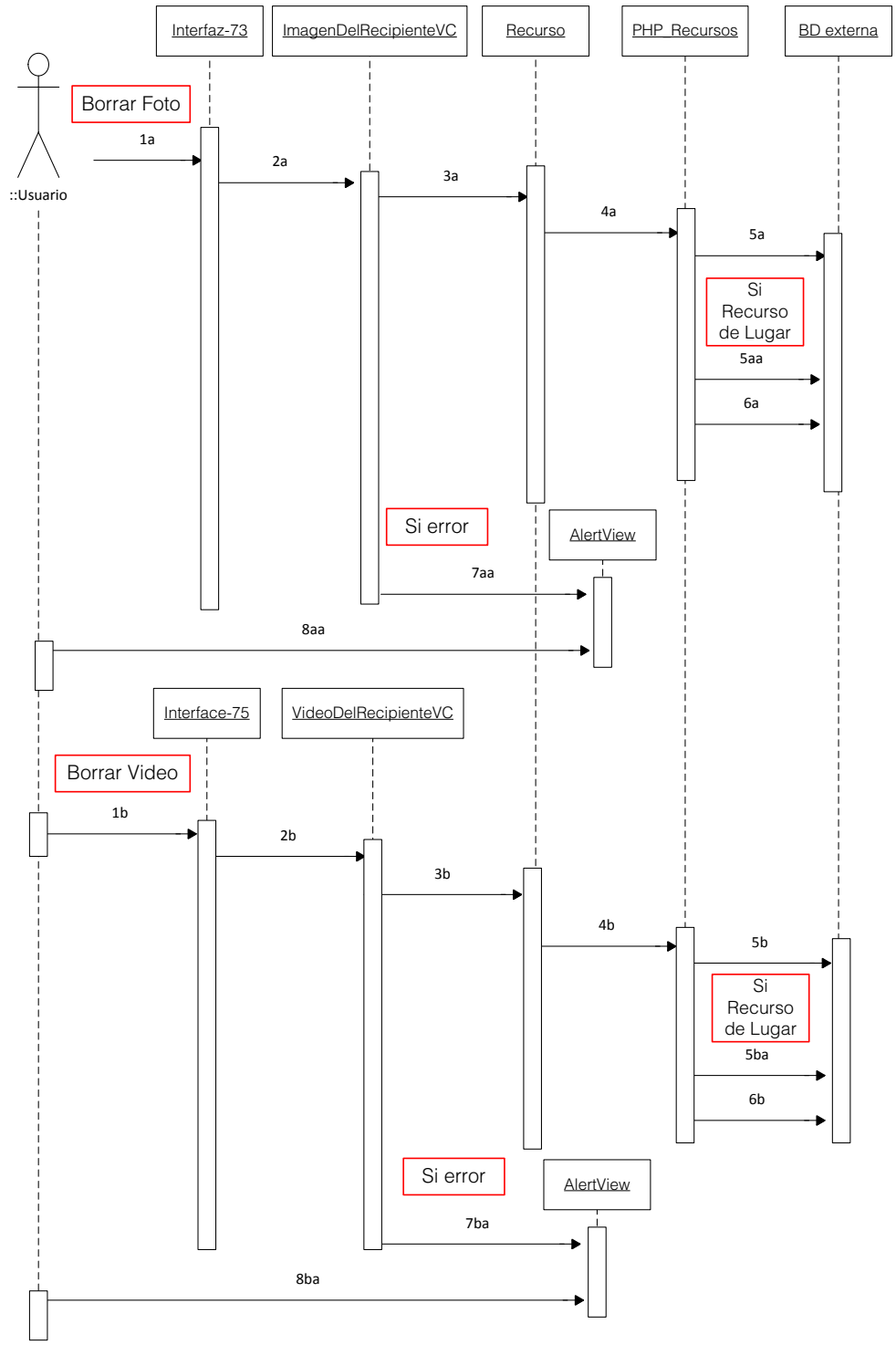


Ilustración 156.- Diagrama borrar recurso de usuario

[Borrar Foto]

1a- El usuario pulsar el botón de borrar y le da Sí.

2a- pulsarBorrar() :void

3a- borrarRecursoSubido(fotoOVideo): void

4a- borrarFoto(funcion, id_recurso, nombre): void

5a- execSQL("DELETE FROM Objetos WHERE id=%rec%")

[si Recurso de Lugar]

5aa- execSQL("UPDATE Lugares SET puntuacion = (SELECT (COALESCE((SELECT COUNT(id_puntuacion) FROM Puntuaciones WHERE id_objeto=%id_lugar%),0) + COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_lugar=%id_lugar%),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_lugar=%id_lugar%),0)) AS suma), fotos=(SELECT count(id_foto) FROM Fotos WHERE id_lugar=%id_lugar%) WHERE id_lugar=%id_lugar%")

6a- execSQL("UPDATE ciudades SET puntuacion = (SELECT (COALESCE((SELECT SUM(puntuacion) FROM Lugares WHERE id_ciudad =%id_ciudad%),0) + COALESCE((SELECT SUM(puntuacion) FROM Rutas WHERE id_ciudad =%id_ciudad%),0)+ COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0)) AS suma), fotos=(SELECT count(id_foto) FROM Fotos WHERE id_ciudad=%id_ciudad%) WHERE idc=%id_ciudad%")

[Si error]

7aa- new()

8aa- El usuario pulsa OK.

[Borrar Vídeo]

1b- El usuario pulsar el botón de borrar y pulsa Sí.

2b- pulsarBorrar(): void

3b- borrarRecursoSubido(fotoOVideo): void

4b- borrarFoto(funcion, id_recurso, nombre): void

5b- execSQL("DELETE FROM Objetos WHERE id=%rec%")

[si Recurso de Lugar]

5ba- execSQL("UPDATE Lugares SET puntuacion = (SELECT (COALESCE((SELECT COUNT(id_puntuacion) FROM Puntuaciones WHERE id_objeto=%id_lugar%),0) + COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_lugar=%id_lugar%),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_lugar=%id_lugar%),0)) AS suma),

videos=(SELECT count(id_video) FROM Videos WHERE id_lugar=%id_lugar%)
WHERE id_lugar=%id_lugar%")

6b- execSQL("UPDATE ciudades SET puntuacion = (SELECT (COALESCE((SELECT SUM(puntuacion) FROM Lugares WHERE id_ciudad =%id_ciudad%),0) + COALESCE((SELECT SUM(puntuacion) FROM Rutas WHERE id_ciudad =%id_ciudad%),0)+ COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0)) AS suma), videos=(SELECT count(id_video) FROM Videos WHERE id_ciudad=%id_ciudad%) WHERE idc=%id_ciudad%")

[Si error]

7ba- new()

8ba- El usuario pulsa OK.

8-Borrar Ruta

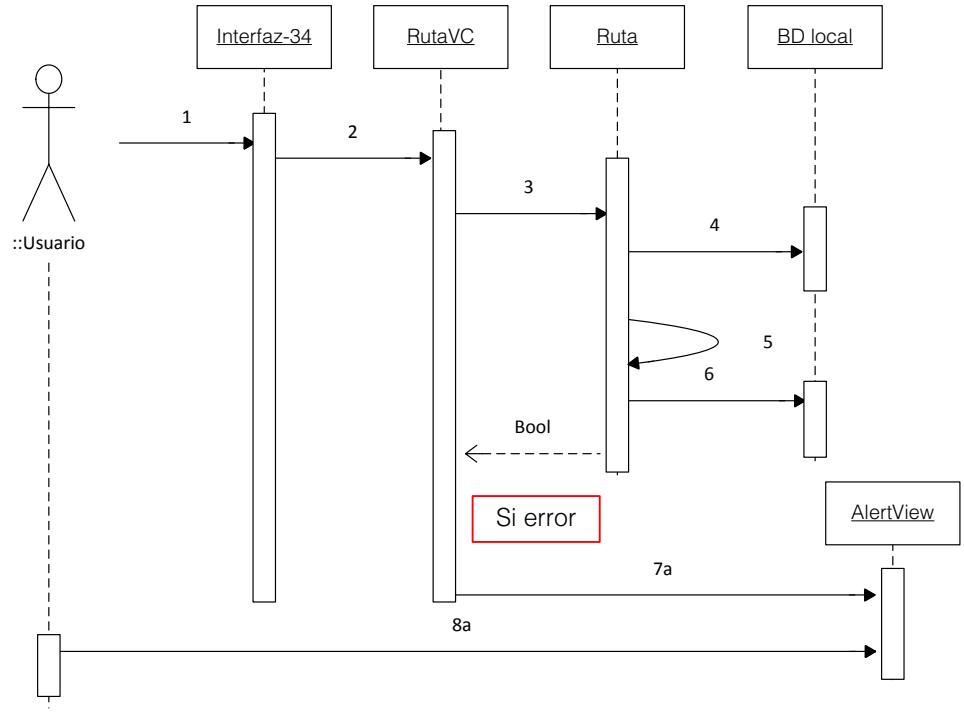


Ilustración 157.- Diagrama borrar ruta

1- El usuario pulsar el botón borrar de la tabla.

2- borrarFilas(): void

3- eliminarRuta(): void

4- execSQL("DELETE FROM Rutas WHERE id_ruta=%id_ruta%")

5- actualizarCiudadDescargada(): void

6- execSQL("UPDATE Ciudades SET lugares = (SELECT count(id_lugar) FROM Lugares WHERE id_ciudad=%id_ciudad%), fotos = (SELECT COUNT(id_foto) FROM Fotos WHERE id_ciudad=%id_ciudad%), videos = (SELECT count(id_video) FROM Videos WHERE id_ciudad=%id_ciudad%), rutas = (SELECT COUNT(nombre) FROM Rutas WHERE id_ciudad=%id_ciudad%) WHERE idc=%id_ciudad%")

[Si error]

7a-new()

8a-El usuario pulsa OK.

9-Borrar/editar Ruta de usuario

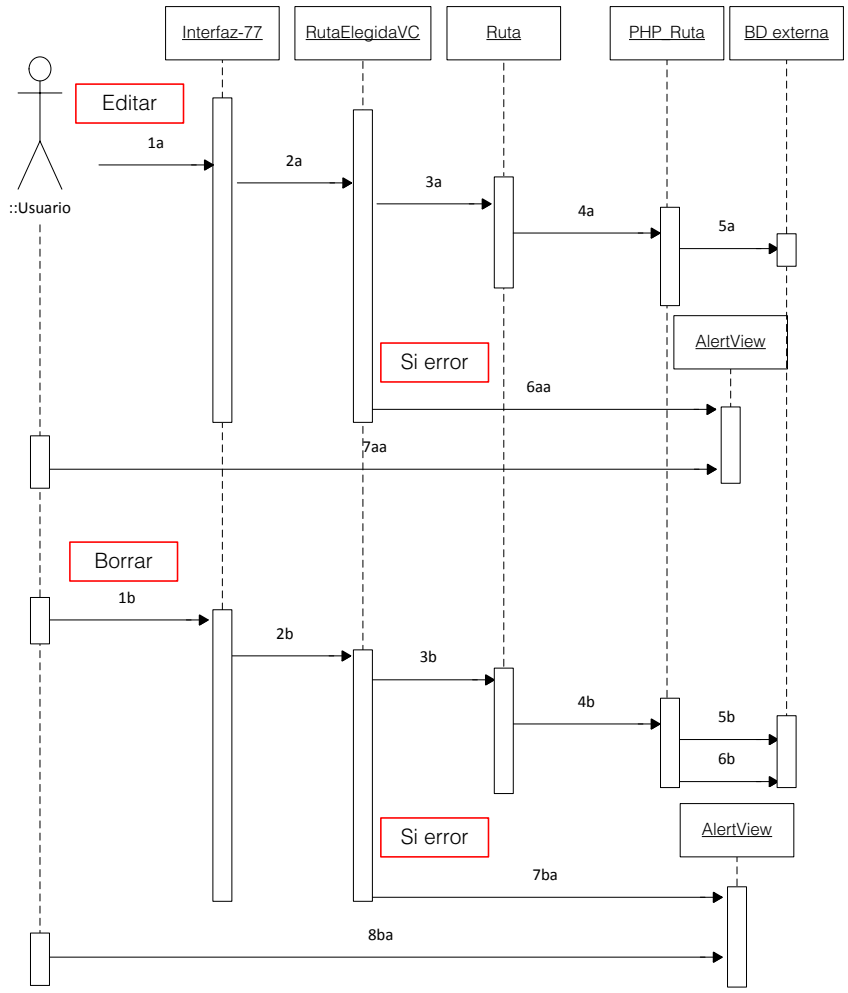


Ilustración 158.- Diagrama borrar/editar ruta de usuario

[Si editar]

1a- El usuario pulsa el botón Hecho para actualizar la Ruta.

2a- pulsarEditar(): void

3a- actualizarRutaUsuario(): void

4a- actualizarRuta(función, descripcion, id_ruta): void

5a- execSQL(UPDATE Ruta SET descripcion=%descripcion% WHERE id_ruta=%id_ruta%)

[Si error]

6aa-new()

7aa-El usuario pulsa OK.

[Si borrar]

1b- El usuario pulsa el botón Borrar para borrar la Ruta y pulsa Sí.

2b- pulsarBorrar(): void

3b- borrarRutaCreada(): void

4b- borrarRuta (función, id_objeto): void

5b- execSQL(DELETE FROM Objetos where id=%id%);

6b- execSQL("UPDATE ciudades SET puntuacion = (SELECT (COALESCE((SELECT SUM(puntuacion) FROM Lugares WHERE id_ciudad =%id_ciudad%),0) + COALESCE((SELECT SUM(puntuacion) FROM Rutas WHERE id_ciudad =%id_ciudad%),0)+ COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0)) AS suma), rutas=(SELECT count(id_ruta) FROM Rutas WHERE id_ciudad=%id_ciudad%) WHERE idc=%id_ciudad%")

[Si error]

9ba-new()

8ba-El usuario pulsa OK.

10-Buscar Usuario

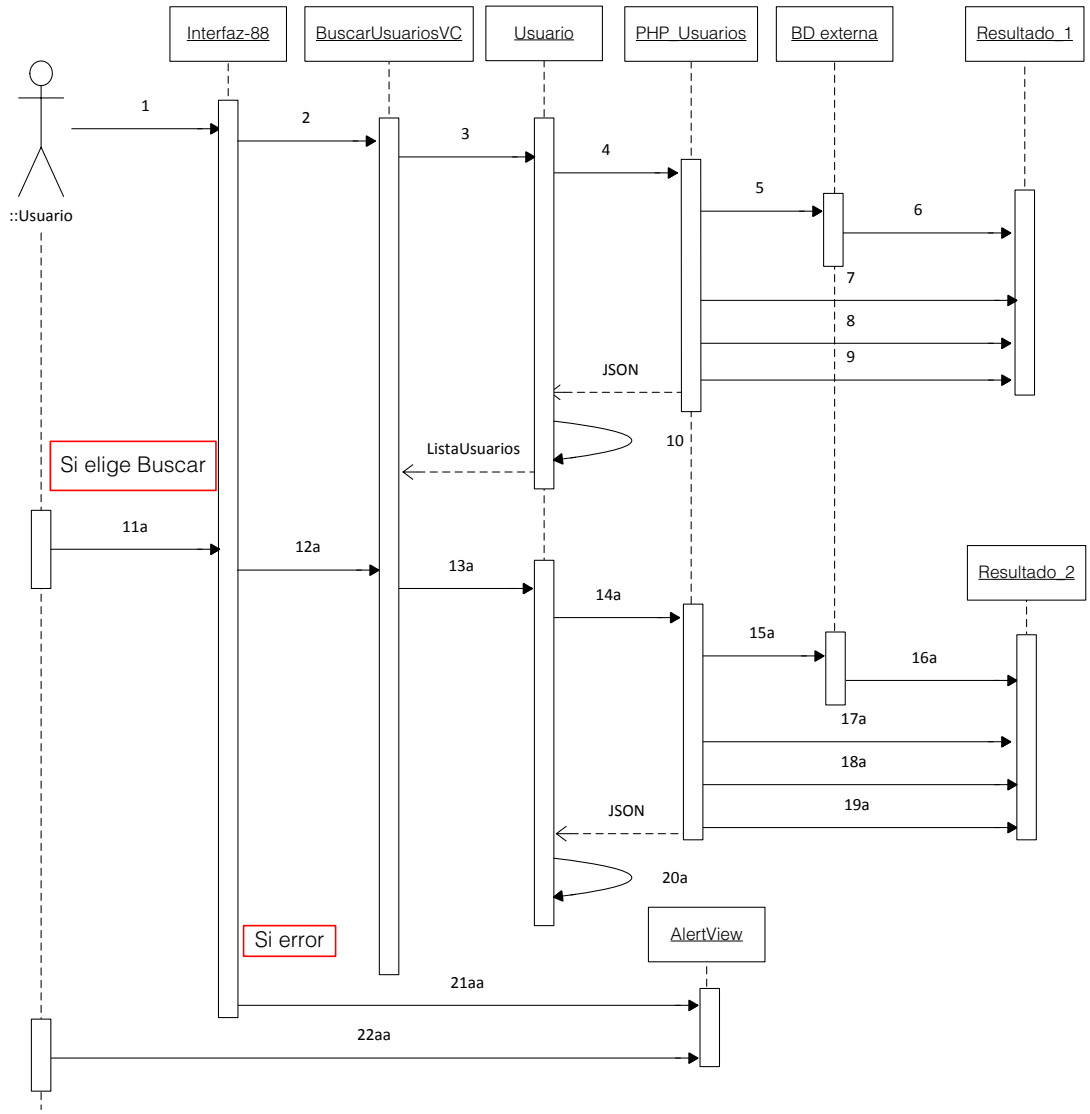


Ilustración 159.- Diagrama buscar usuario

1-El usuario pulsa botón BuscarAmigos y se accede a esa interfaz.

2- inicializar(idUsuarioSesion): void

3- recogerSugerenciasSinUsuario(idUser): ListaUsuarios

4- consultarSugerencias(funcion): JSON

5- execSQL("SELECT u.*, COUNT(l.autor) AS members FROM Usuarios AS u LEFT JOIN Lugares AS l ON u.nick = l.autor GROUP BY u.nick HAVING members > 4");

6- new()

7- next()

8- getData(): Data

9- close()

10-rehacerArray(array) conIdUser(idUser): ListaUsuarios

[Si elige Buscar]

11a-El usuario pulsa botón Buscar, introduce el nombre y pulsa buscar.

12a- searchBarSearchButtonClicked(searchBar): void

13a- buscarUsuario(textoIntroducido): ListaUsuaros

14a- buscarUsuario(funcion, nick): JSON

15a- execSQL("SELECT * FROM Usuarios WHERE nick=%nick%");

16a- new()

17a- next()

18a- getData(): Data

19a- close()

20a- rehacerArrayUsuario(array) conIdUser(idUser): ListaUsuarios

[Si error]

21aa-new()

22aa-El usuario pulsa OK.

11-Cerrar Sesión

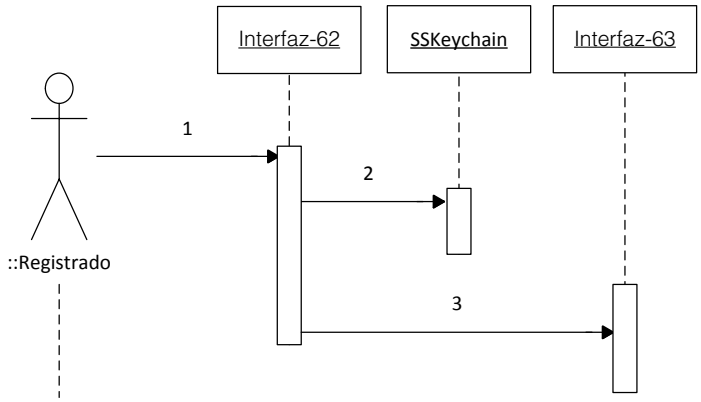


Ilustración 160.- Diagrama cerrar sesión

1- El usuario pulsa cerrar sesión.

2- resetKeychainItem( ): void

3- new()

12-Consultar Amigos

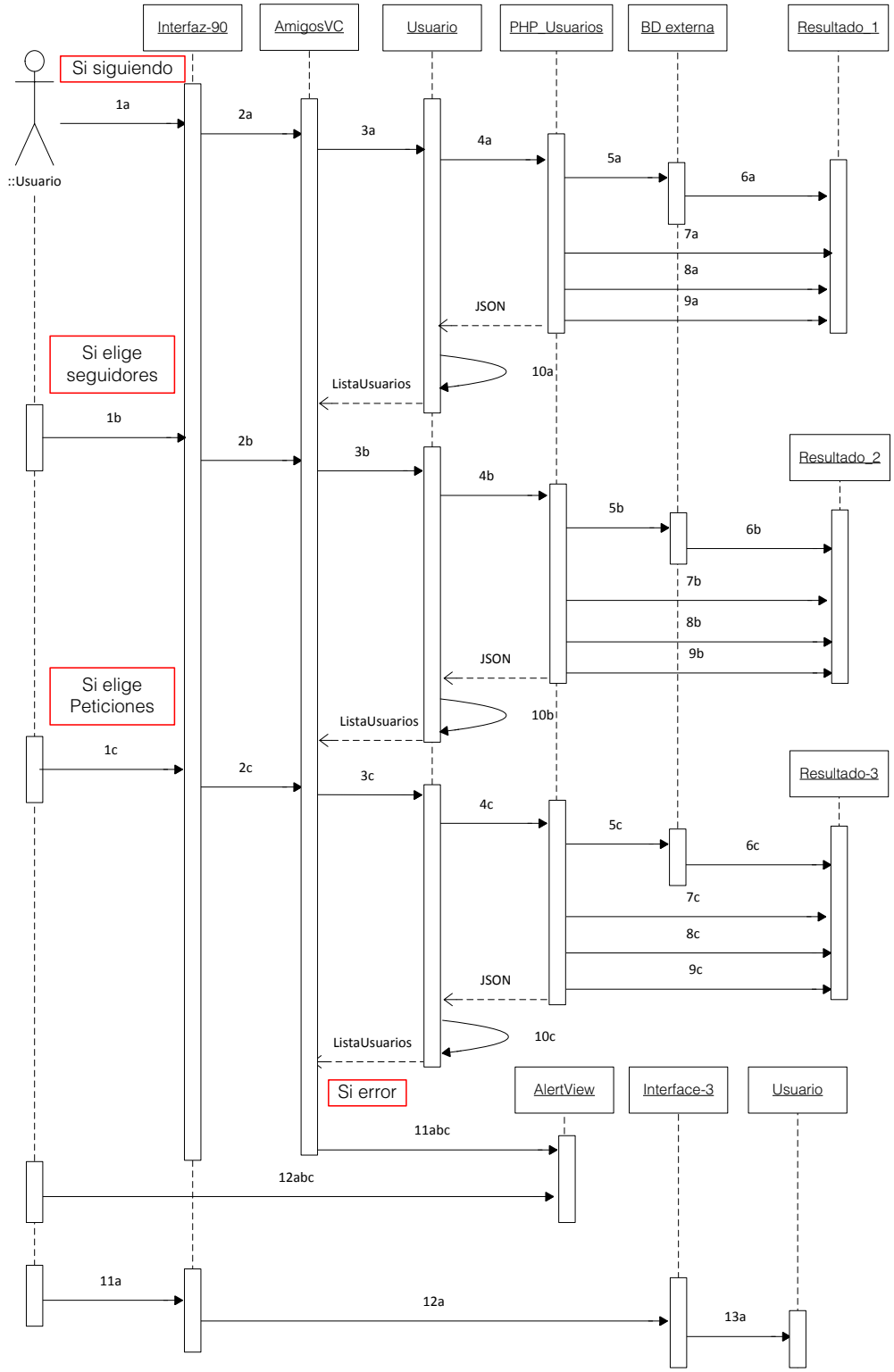


Ilustración 161.- Diagrama consultar amigos

[Siguiendo]

1a- El usuario pulsa en Amigos y se accede a la nueva interfaz en la pestaña "Siguiendo".

2a- inicializar(): void

3a- recogerAmistadesDe(id_usu): ListaUsuaros

4a- consultarAmistades(funcion, id_usuario): JSON

5a- execSQL("SELECT * FROM Usuarios WHERE id_usuario in (SELECT id_usuario2 FROM Amistades WHERE id_usuario1=%id_usuario% and estado=1) ORDER BY nick ASC")

6a- new()

7a- next()

8a- getData(): Data

9a- close()

10a- rehacerArrayUsuario(array) conIndice(0): ListaUsuarios

[Si elige seguidores]

1b- El usuario pulsa en la pestaña "Seguidores".

2b- segmentedControlIndexChanged(id): IBAction

3b- recogerSeguidoresDe(id_usu): ListaUsuaros

4b- consultarSeguidores(funcion, id_usuario): JSON

5b- execSQL("SELECT * FROM Usuarios WHERE id_usuario in (SELECT id_usuario1 FROM Amistades WHERE id_usuario2=%id_usuario% and estado=1) ORDER BY nick ASC")

6b- new()

7b- next()

8b- getData(): Data

9b- close()

10b- rehacerArrayUsuario(array) conIndice(1): ListaUsuarios

[Si elige seguidores]

1c- El usuario pulsa en la pestaña "Peticiones".

2c- segmentedControlIndexChanged(id): IBAction

3c- recogerPeticionesDe(id_usu): ListaUsuaros

4c- consultarPeticiones(funcion, id_usuario): JSON

5c- execSQL("SELECT * FROM Usuarios WHERE id_usuario in (SELECT id_usuario1 FROM Amistades WHERE id_usuario2=%id_usuario% and estado=0) ORDER BY nick ASC")

6c- new()

7c- next()

8c- getData(): Data

9c- close()

10c- rehacerArrayUsuario(array) conIndice(2): ListaUsuarios

[Si error]

11abc- new()

12abc- El usuario pulsa OK.

[Si el usuario pulsa sobre un usuario]

11a-El usuario pulsa sobre un usuario.

12a- new(Usuario,idUsuarioSesion)

13a- inicializar(): void

13- Consultar Comentarios

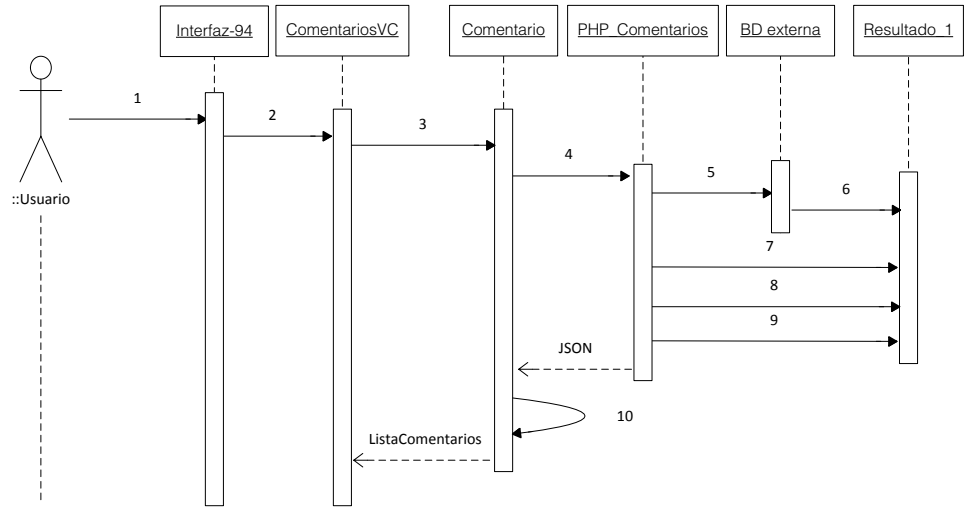


Ilustración 162.- Diagrama consultar comentarios

1- El usuario pulsa en Comentarios de un Lugar y se accede a la interfaz de comentarios.

2- inicializar(): void

3- consultarComentarios(): ListaComentarios

4- consultarComentarios(funcion, id_lugar): JSON

5- execSQL("SELECT * FROM Comentarios WHERE id_lugar=%id_lugar%")

6- new()

7- next()

8- getData(): Data

9- close()

10- rehacerArrayDeComentarios(array): ListaComentarios

14-Consultar Creaciones

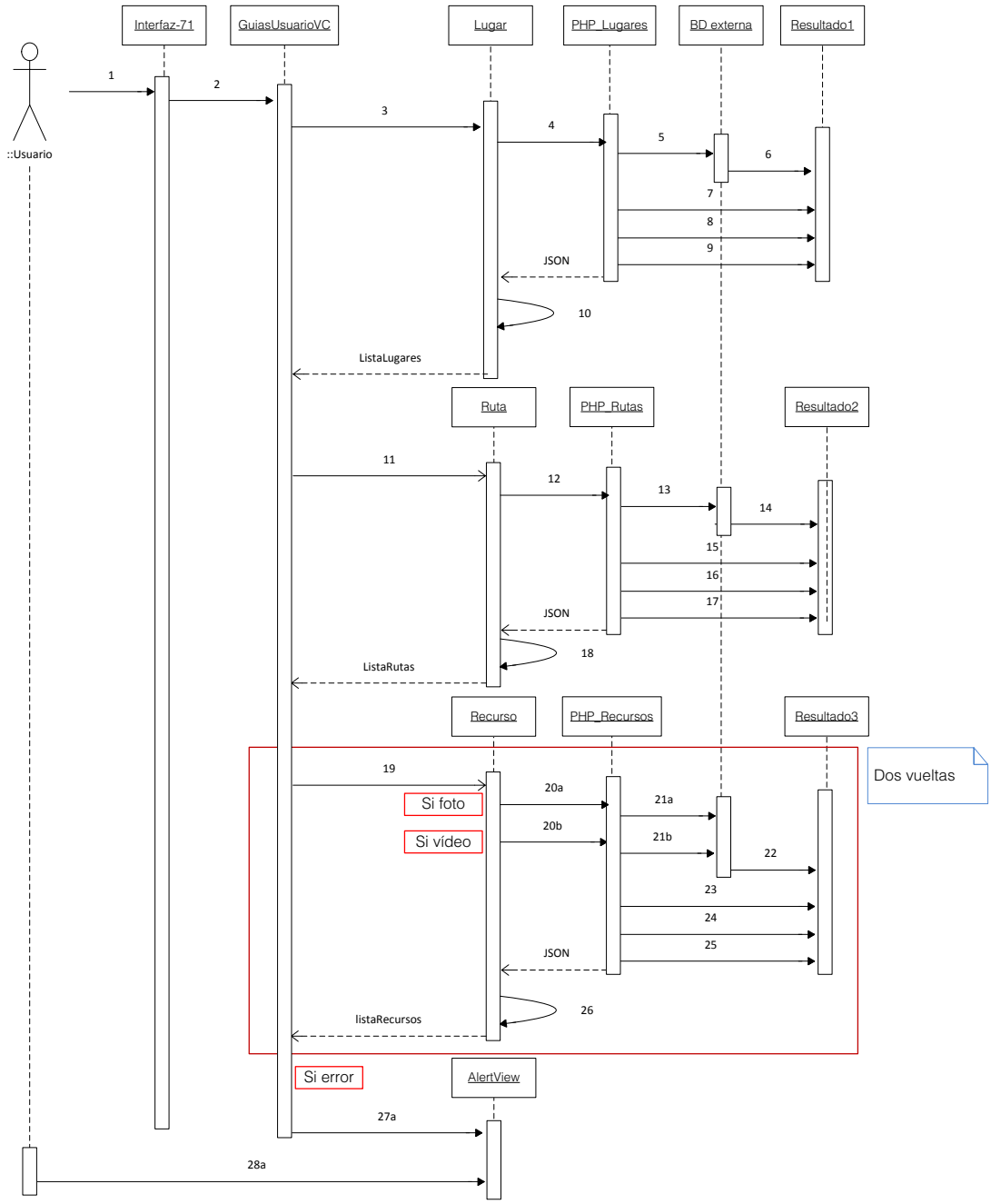


Ilustración 163.- Diagrama consultar creaciones

1- El usuario pulsa en Guías Creadas en su perfil y se accede a la interfaz de sus creaciones.

2- inicializar(): void

3- consultarLugaresDeUsuario(): ListaLugares

4- consultarLugaresUsuario(funcion, nick): JSON

5- execSQL("SELECT * FROM Lugares WHERE autor=%nick% ORDER BY nombre ASC")

6- new()

7- next()

8- getData(): Data

9- close()

10- rehacerArrayDeLugares(array): ListaLugares

11- consultarRutasDeUsuario(): ListaRutas

12- consultarRutasUsuario(funcion, nick): JSON

13- execSQL("SELECT * FROM Rutas WHERE autor=%nick% ORDER BY nombre ASC")

14- new()

15- next()

16- getData(): Data

17- close()

18- rehacerArrayDeRutas(array) conIndicador(2): ListaRutas

19- consultarRecursosUsuario(siFotoOVideo): ListaRercursos

[Si foto]

20a- consultarFotosUsuario(funcion, nick): JSON;

21a- execSQL("SELECT * FROM Fotos WHERE autor=%nick%)

[Si vídeo]

20b- consultarVideosUsuario(funcion, nick): JSON

21b- execSQL("SELECT * FROM Videos WHERE autor=%nick%)

22- new()

23- next()

24- getData(): Data

25- close()

26- rehacerArrayRecurso(array) conRecurso(fotoOVideo) conLugar(0): ListaRecursos

[Si error]

27a- new()

28a- El usuario pulsa OK.

15-Consultar Foto y Vídeo

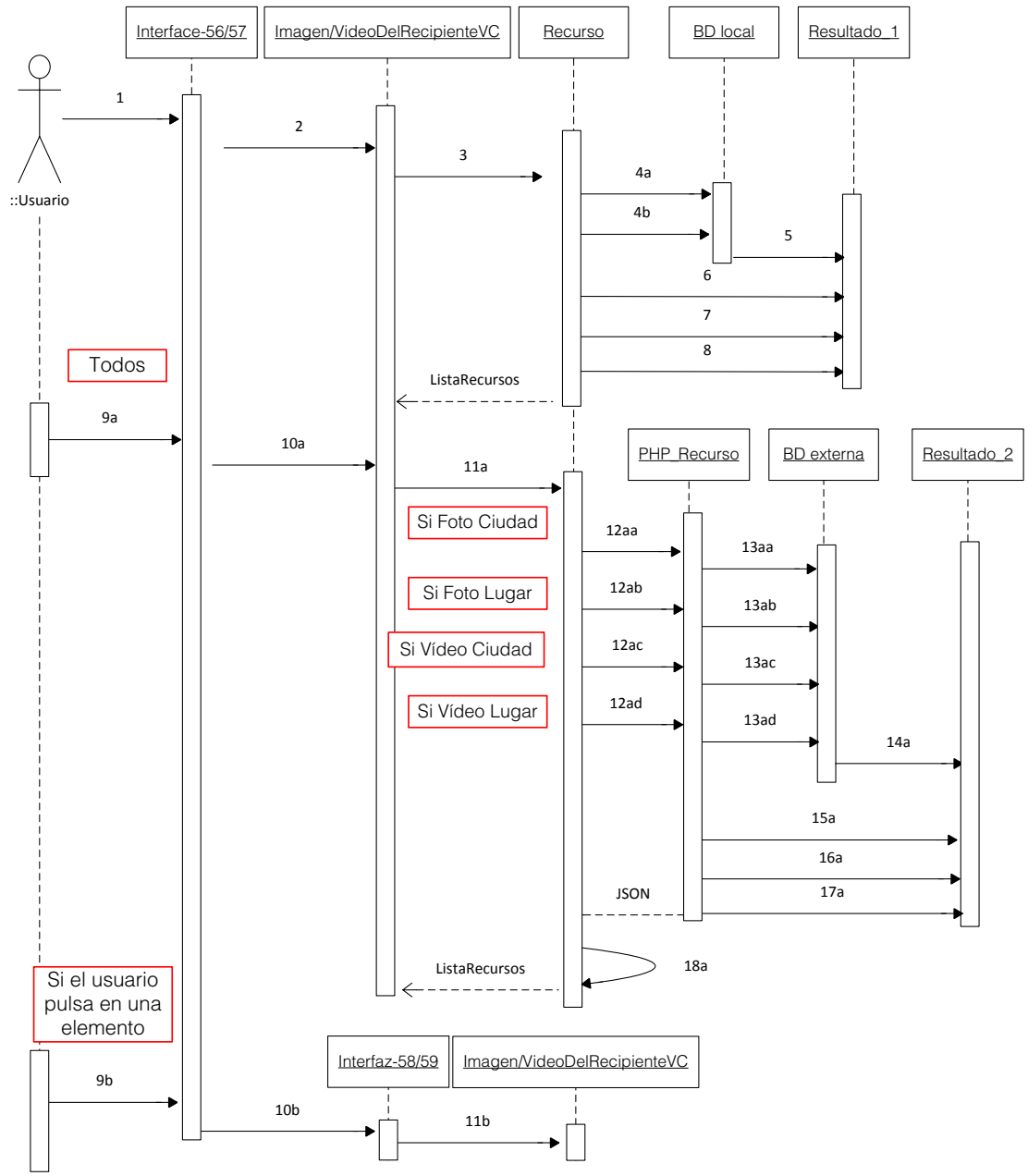


Ilustración 164.- Diagrama consultar fotos y vídeos

[Descargadas]

1- El usuario pulsa en la celda de Fotos y se accede a la interfaz de las fotos en la pestaña de "Mis Fotos".

2- inicializar(): void

3- cargarRecurso(fotoOVideo): ListaRecursos

[Si foto o vídeo]

4a- execSQL("SELECT * FROM Fotos WHERE id_ciudad=%id_ciudad% and id_lugar=%id_ciudad%")

4b- execSQL("SELECT * FROM Videos WHERE id_ciudad=%id_ciudad% and id_lugar=%id_ciudad%")

5- new()

6- next()

7- getData(): Data

8- close()

[Si pulsa en Todas]

9a- El usuario pulsa en la pestaña "Todas".

10a- segmentedControlIndexChanged(id): IBAction

11a- consultarRecursos(pIdLugar) conRecurso(fotoOVideo): ListaRecursos

[Si foto(a) o vídeo(b)]

[Si es foto de Ciudad]

12aa- consultarFotosCiudad(funcion, id_ciudad): JSON

13aa- execSQL("SELECT * FROM Fotos WHERE id_ciudad=%ciudad% and id_lugar is NULL")

[Si es foto de Lugar]

12ab-consultarFotosLugar(funcion, id_lugar): JSON

13ab- execSQL("SELECT * FROM Fotos WHERE id_lugar=%id_lugar%")

[Si es vídeo de Ciudad]

12ac-consultarVideosCiudad(funcion, id_ciudad): JSON;

13ac- execSQL("SELECT * FROM Videos WHERE id_lugar=%id_lugar%")

[Si es vídeo de Lugar]

12ad-consultarVideosLugar(funcion, id_lugar): JSON;

13ad- execSQL("SELECT * FROM Videos WHERE id_lugar=%id_lugar%")

14a- new()

15a- next()

16a- getData(): Data

17a- close()

18a- rehacerArrayRecurso(array) conRecurso(fotoOVideo) conLugar(id_lugar): ListaRecursos

[Pulsar sobre un recurso]

9b- El usuario pulsa sobre un recurso.

10b- new(recurso)

11b- inicializar(): void

16-Consultar guías

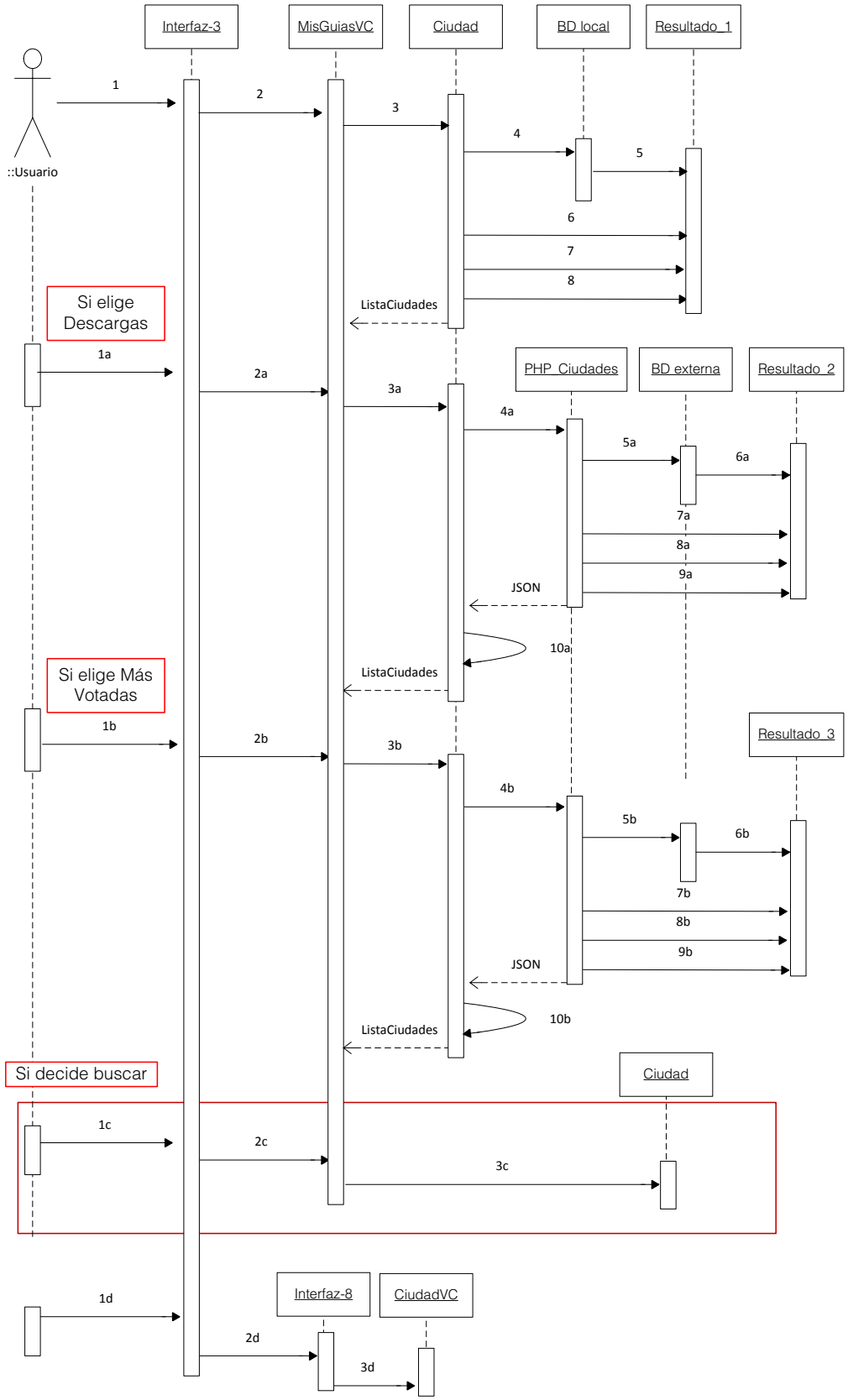


Ilustración 165.- Diagrama consultar guías

1- El usuario ha pulsado el botón de guías de la pantalla principal y se ha accedido a la vista de las ciudades.
2- inicializar(): void
3- cargarMisCiudades(): ListaCiudades
4- execSQL("SELECT * FROM Ciudades ORDER BY pais, nombre ASC")
5- new()
6- next()
7- getData(): Data
8- close()

[Si elige Descargas]

1a- El usuario pulsa en la pestaña "Descargas".
2- segmentedControlIndexChanged(id): IBAction
3a- consultarCiudades(indicador): ListaCiudades
4-consultarTodas(funcion): JSON
5a- execSQL("SELECT * FROM ciudades ORDER BY pais,nombre ASC");
6a- new()
7a- next()
8a- getData(): Data
9a- close()
10a- rehacerArrayCiudades(array) conIndicador(indicador):ListaCiudades

[Si elige más votadas]

1b- El usuario pulsa en la pestaña "Más Votadas".
2b- segmentedControlIndexChanged(id): IBAction
3b- consultarCiudades(indicador): ListaCiudades
4b-consultarMasVotadas(funcion): JSON
5b- execSQL("SELECT * FROM ciudades ORDER BY puntuacion DESC LIMIT 5")
6b- new()
7b- next()
8b- getData(): Data
9b- close()
10b- rehacerArrayCiudades(array) conIndicador(indicador): ListaCiudades

[Elige buscar]

1c- El usuario pulsa en la pestaña "Más Votadas".
2c- searchBar(searchBarra) textDidChange(searchText): void
3c- getNombre(): string

[Pulsa en una ciudad]

1d- El usuario pulsa sobre una ciudad.

2d- new(ciudadDescargada,ciudadExterna)

3d- inicializar(): void

17-Consultar Lugar en mapa

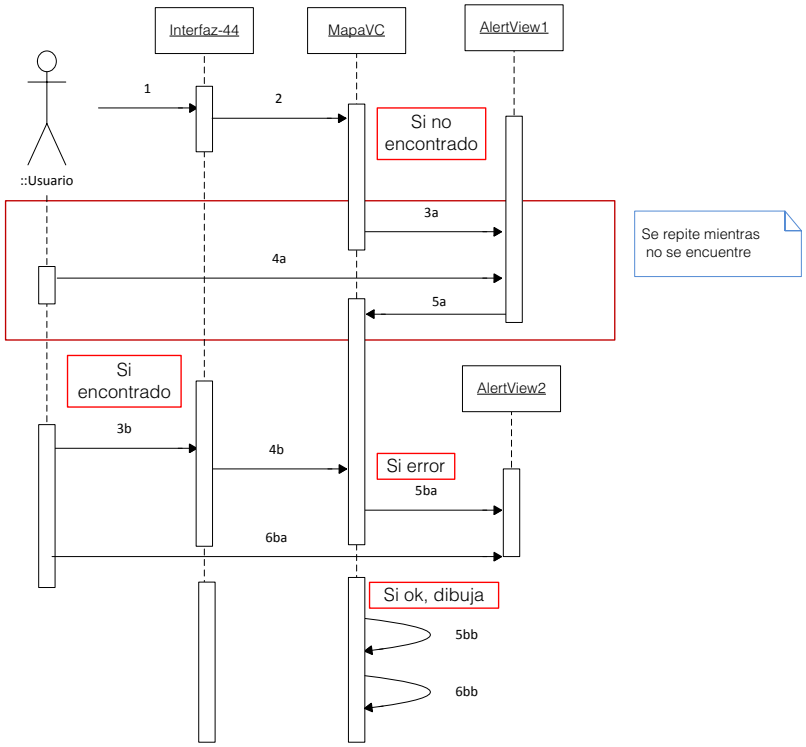


Ilustración 166.- Diagrama consultar lugar en mapa

1- El usuario pulsa en el botón de mapas y se accede a la vista con el mapa.

2- inicializarMapa(string): void

[Si dirección no encontrada]

3a- new()

4a- El usuario introduce un nombre y pulsa en buscar.

5a- inicializarMapa(string): void

[Si encontrado]

3b- El usuario pulsa sobre la anotación que ha aparecido en la vista.

4b- didSelectAnnotationView(mapView, anotacionView): void

[si error]

5ba- new()

6ba- El usuario pulsa ok.

[si ok, dibuja]

5bb- rutaDesde(mapView, mapItem): void

6bb- rendererForOverlay(overlay, mapView): MKOverlayRenderer

18- Consultar lugares

Este diagrama está dividido en dos para mejorar su legibilidad y claridad.

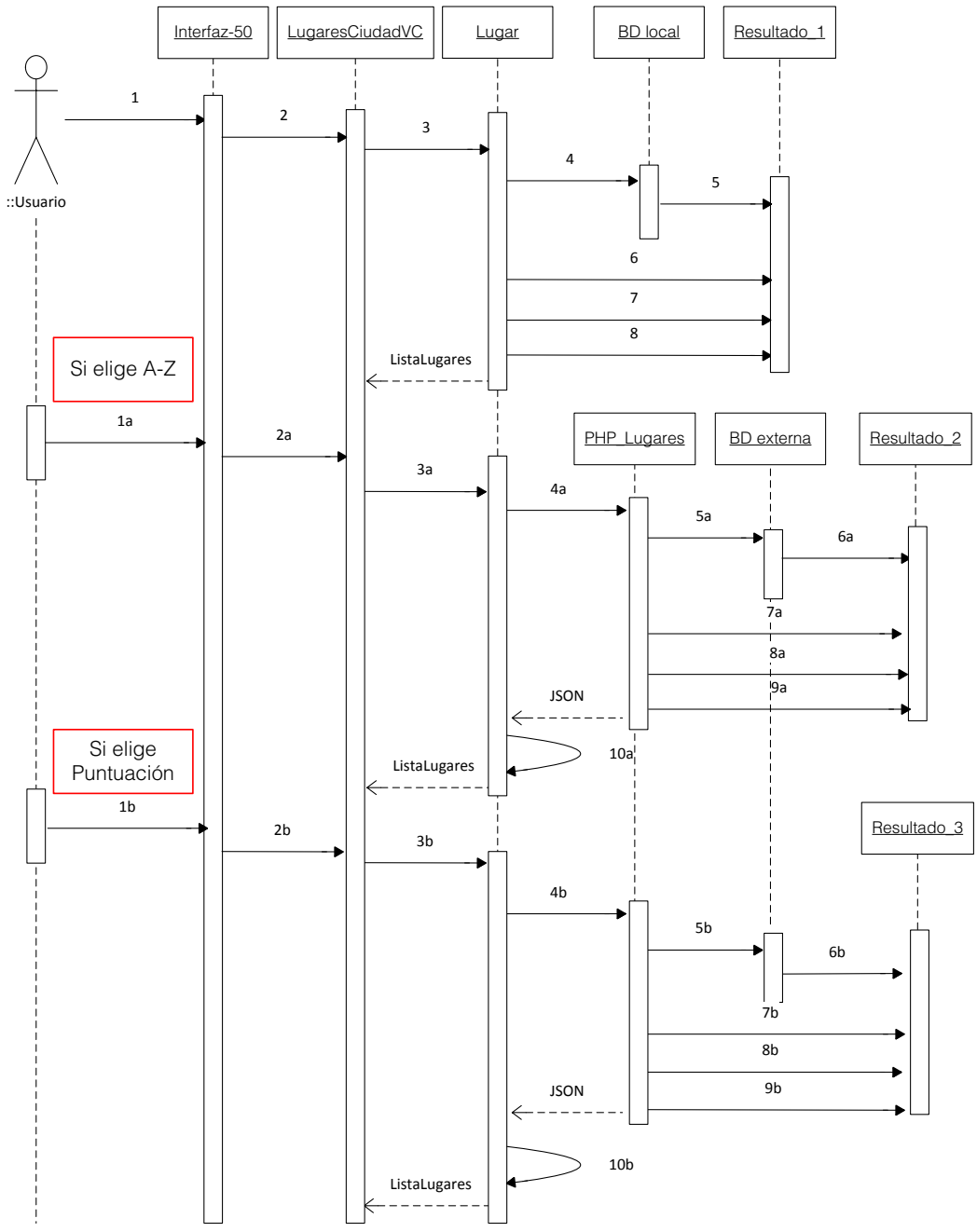


Ilustración 167.- Diagrama consultar lugares 1

[Descargados]

1- El usuario ha pulsado el botón de lugares de la ciudad.

2- inicializar(): void

3- cargarLugares(): ListaLugares

4- execSQL("SELECT * FROM Lugares ORDER BY nombre ASC")

5- new()

6- next()

7- getData(): Data

8- close()

[Si elige AZ]

1a- El usuario pulsa en la pestaña "Descargas".

2a- segmentedControlIndexChanged(id): IBAction

3a- consultarLugares(indicador): ListaLugares

4a-consultarLugaresAZ(id_ciudad,funcion): JSON

5a- execSQL("SELECT * FROM Lugares WHERE id_ciudad= %id_ciudad% ORDER BY nombre ASC")

6a- new()

7a- next()

8a- getData(): Data

9a- close()

10a- rehacerArrayLugares(array): ListaLugares

[Si elige puntuación]

1b- El usuario pulsa en la pestaña "Más Votadas".

2b- segmentedControlIndexChanged(id): IBAction

3b- consultarLugares(indicador): ListaLugares

4b- consultarLugaresPuntuacion (funcion, id_ciudad): JSON

5b- execSQL("SELECT * FROM Lugares WHERE id_ciudad=%id_ciudad% ORDER BY puntuacion DESC")

6b- new()

7b- next()

8b- getData(): Data

9b- close()

10b- rehacerArrayLugares(array): ListaLugares

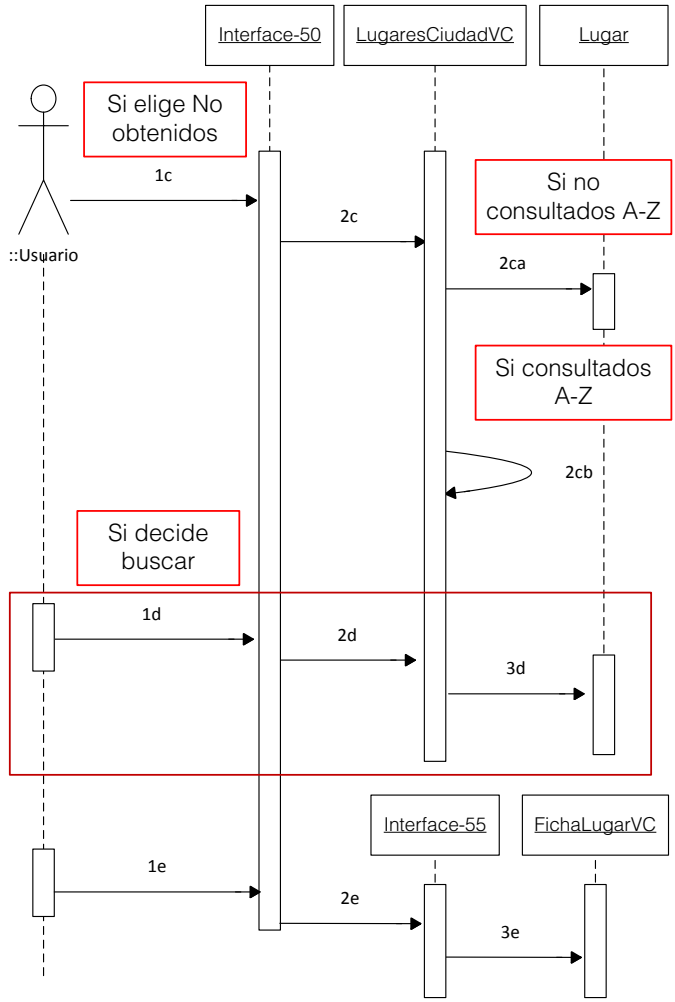


Ilustración 168.- Diagrama consultar lugares 2

[Si elige No Obtenidos]

1c- El usuario pulsa en la pestaña "Más Votadas".

2c- segmentedControlIndexChanged(id): IBAction

[Si no consultados A-Z]

2ca- consultarLugares(indicador): ListaLugares

[Si consultados]

2cb-comprobarObetnidos(): bool

[Elige buscar]

1d- El usuario pulsa en la pestaña "Más Votadas".

2d- searchBar(searchBarra) textDidChange(searchText): void;

3d- getNombre(): string

[El usuario pulsa sobre un lugar]

1e- El usuario pulsa sobre un lugar.

2e- new(lugar)

3e- inicializar(): void

19-Consultar mensajes

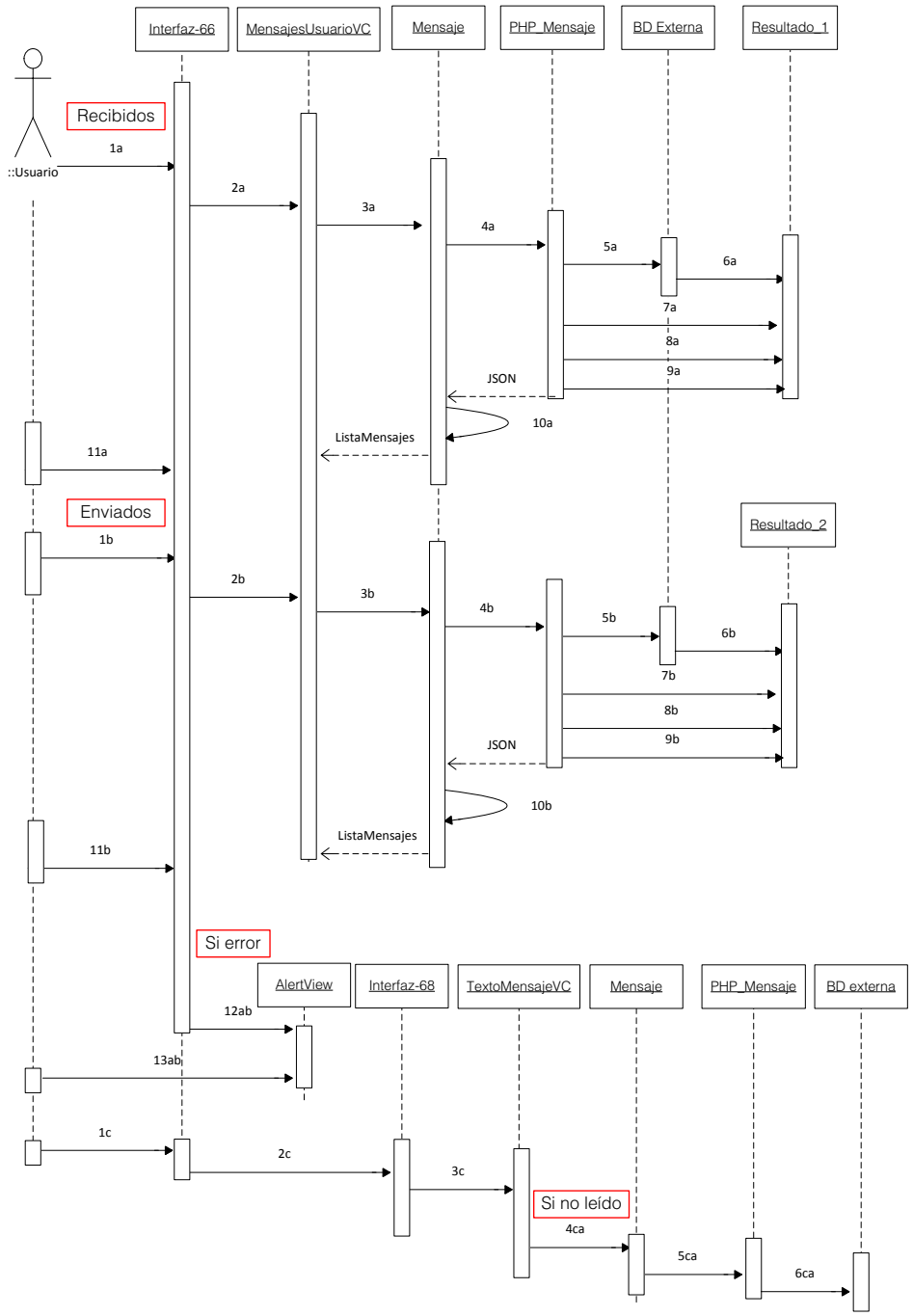


Ilustración 169.- Diagrama consultar mensajes

[Recibidos]

1a- El usuario pulsa en el botón y mensajes y se accede a la interfaz.

2a- inicializar(): void

3a- recogerMensajesDe(usuarioNick) yLimite(limite): ListaMensajes;

4a- consultarRecibidos(funcion, nick, limite): JSON

5a- execSQL("(SELECT * FROM Mensajes WHERE recibidor=%nick% ORDER BY fecha DESC) LIMIT %limite%,10")

6a- new()

7a- next()

8a- getData(): Data

9a- close()

10a- rehacerArray(array): ListaMensajes

11a- El usuario pulsar en cargar más mensajes y se ejecuta desde el punto 2a.

[Enviados]

1b- El usuario pulsa en la pestaña "Descargas".

2b- segmentedControlIndexChanged(id): IBAction

3b- recogerMensajesEnviadorPor(usuarioNick) yLimite(limite): ListaMensajes;

4b- consultarEnviados(funcion, nick, limite): JSON

5b- execSQL("(SELECT * FROM Mensajes WHERE enviado=%nick% ORDER BY fecha DESC) LIMIT %limite%,10")

6b- new()

7b- next()

8b- getData(): Data

9b- close()

10b- rehacerArray (array): ListaMensajes

11b- El usuario pulsar en cargar más mensajes y se ejecuta desde el punto 2b.

[si error]

12ab- new()

13ab- El usuario pulsa OK.

[Si el usuario pulsa sobre un mensaje]

1c- El usuario pulsa sobre un mensaje.

2c- new(mensaje)

3c- inicializar()

[Si no leído]

4ca- actualizarALeido(): void

5ca- actualizarALeido(id_mensaje,funcion): JSON

6ca- execSQL("UPDATE Mensajes SET leido=1 WHERE id_mensaje=%id_mensaje%)

20- Consultar Puntuaciones

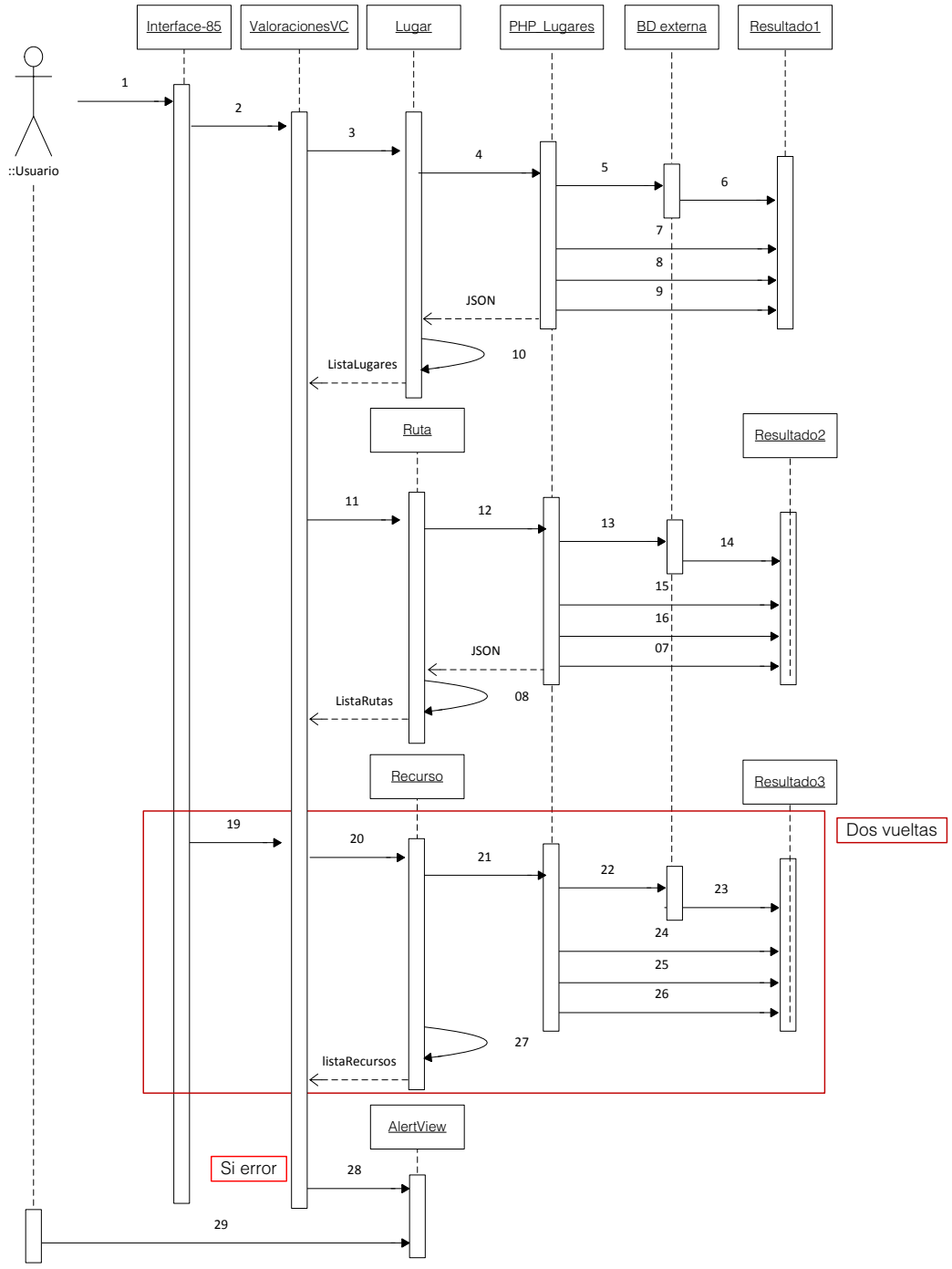


Ilustración 170.- Diagrama consultar puntuaciones

1- El usuario pulsa en Valoraciones en su perfil y se accede a la interfaz de las puntuaciones que ha hecho.
2- inicializar()
3- consultarLugaresDeUsuario(): ListaLugares
4- consultarLugaresUsuario(funcion, nick): JSON;
5- execSQL("SELECT * FROM Lugares WHERE autor=%nick% ORDER BY nombre ASC")
6- new()
7- next()
8- getData(): Data
9- close()
10- rehacerArrayDeLugares(array): ListaLugares
11- consultarRutasDeUsuario(): ListaRutas
12- consultarRutasUsuario(funcion, nick): JSON
13- execSQL("SELECT * FROM Rutas WHERE autor=%nick% ORDER BY nombre ASC")
14- new()
15- next()
16- getData():Data
17- close()
18- rehacerArrayDeRutas(array) conIndicador(2): ListaRutas
19- consultarRecursosUsuario(siFotoOVideo): ListaRercursos
[Si foto(a) o vídeo(b)]
    20a- consultarFotosUsuario(funcion, nick): JSON;
    21a- execSQL("SELECT * FROM Fotos WHERE autor=%nick%)
    20b- consultarVideosUsuario(funcion, nick): JSON
    21b- execSQL("SELECT * FROM Videos WHERE autor=%nick%)
22- new()
23- next()
24- getData():Data
25- close()
26- rehacerArrayRecurso(array) conRecurso(fotoOVideo) conLugar(0): ListaRecursos;
27- new()
28- El usuario pulsa OK.

21- Consultar Rutas

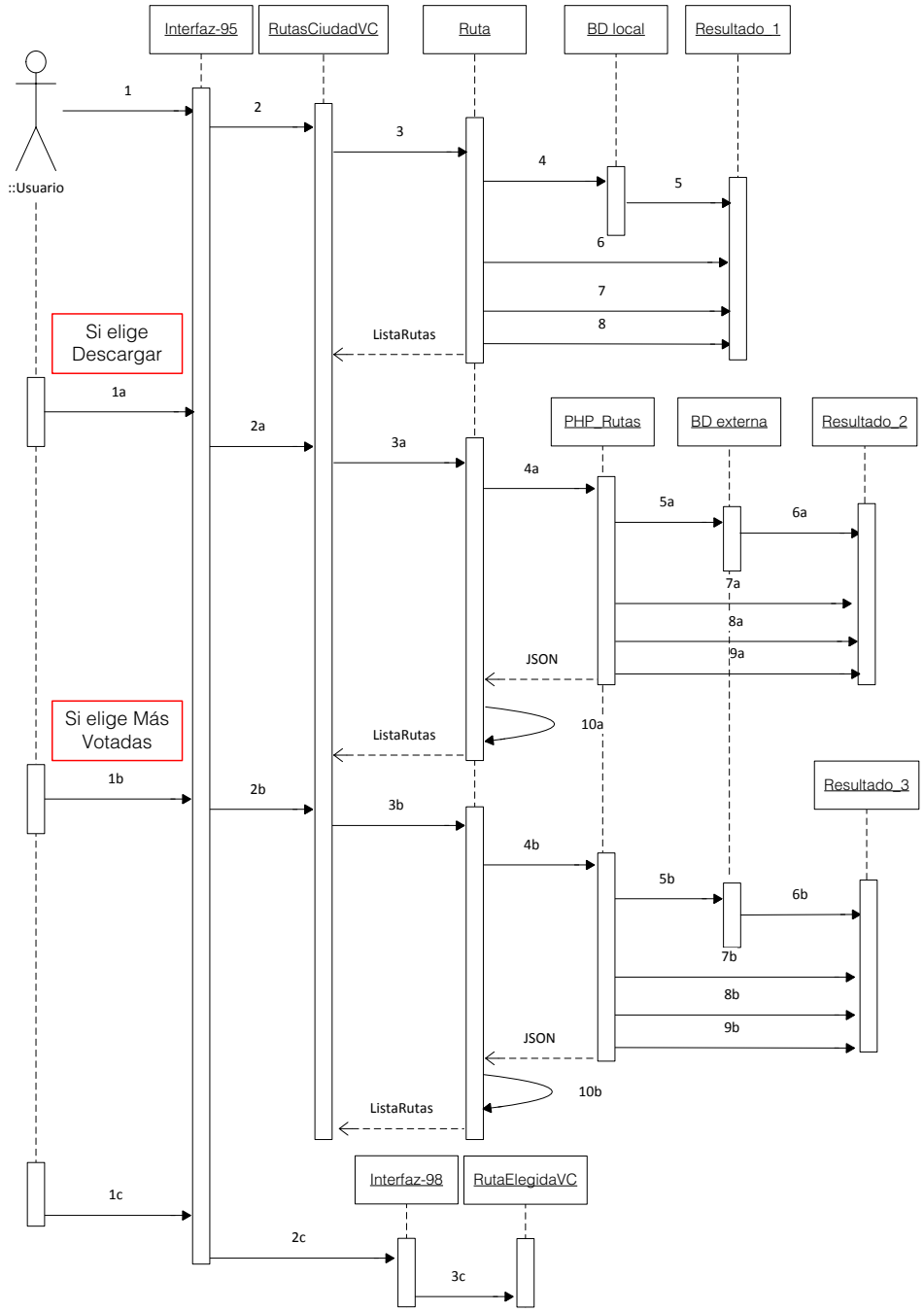


Ilustración 171.- Diagrama consultar rutas

1- El usuario ha pulsado el botón de rutas de la ciudad.

2- inicializar(): void

3- cargarRutas(): ListaRutas

4- execSQL("SELECT * FROM Rutas ORDER BY dias ASC")

5- new()

6- next()

7- getData():Data

8- close()

[Si elige Descargar]

1a- El usuario pulsa en la pestaña "Descargar".

2a- segmentedControlIndexChanged(id): IBAction

3a- consultarRutas(indicador): ListaRutas

4a-consultarRutasPorDias(id_ciudad,funcion): JSON

5a- execSQL("SELECT * FROM Rutas WHERE id_ciudad=%id_ciudad% ORDER BY dias ASC")

6a- new()

7a- next()

8a- getData():Data

9a- close()

10a- rehacerArray (array) conIndicador(indicador): ListaRutas

[Si elige Más Votasdas]

1b- El usuario pulsa en la pestaña "Más Votadas".

2b- segmentedControlIndexChanged(id): IBAction

3b- consultarRutas (indicador): ListaRutas

4b- consultarRutasMasVotadas (funcion, id_ciudad): JSON

5b- execSQL("SELECT * FROM Rutas where id_ciudad=%id_ciudad% ORDER BY puntuacion DESC LIMIT 5")

6b- new()

7b- next()

8b- getData():Data

9b- close()

10b- rehacerArray(array) conIndicador(indicador): ListaRutas

[El usuario pulsa en una ruta]

1c- El usuario pulsa sobre una ruta.

2c- new(ruta)

3c- inicializar(): void

22- Crear Guía

Se ha dividido este diagrama en dos debido a su tamaño.

1º diagrama:

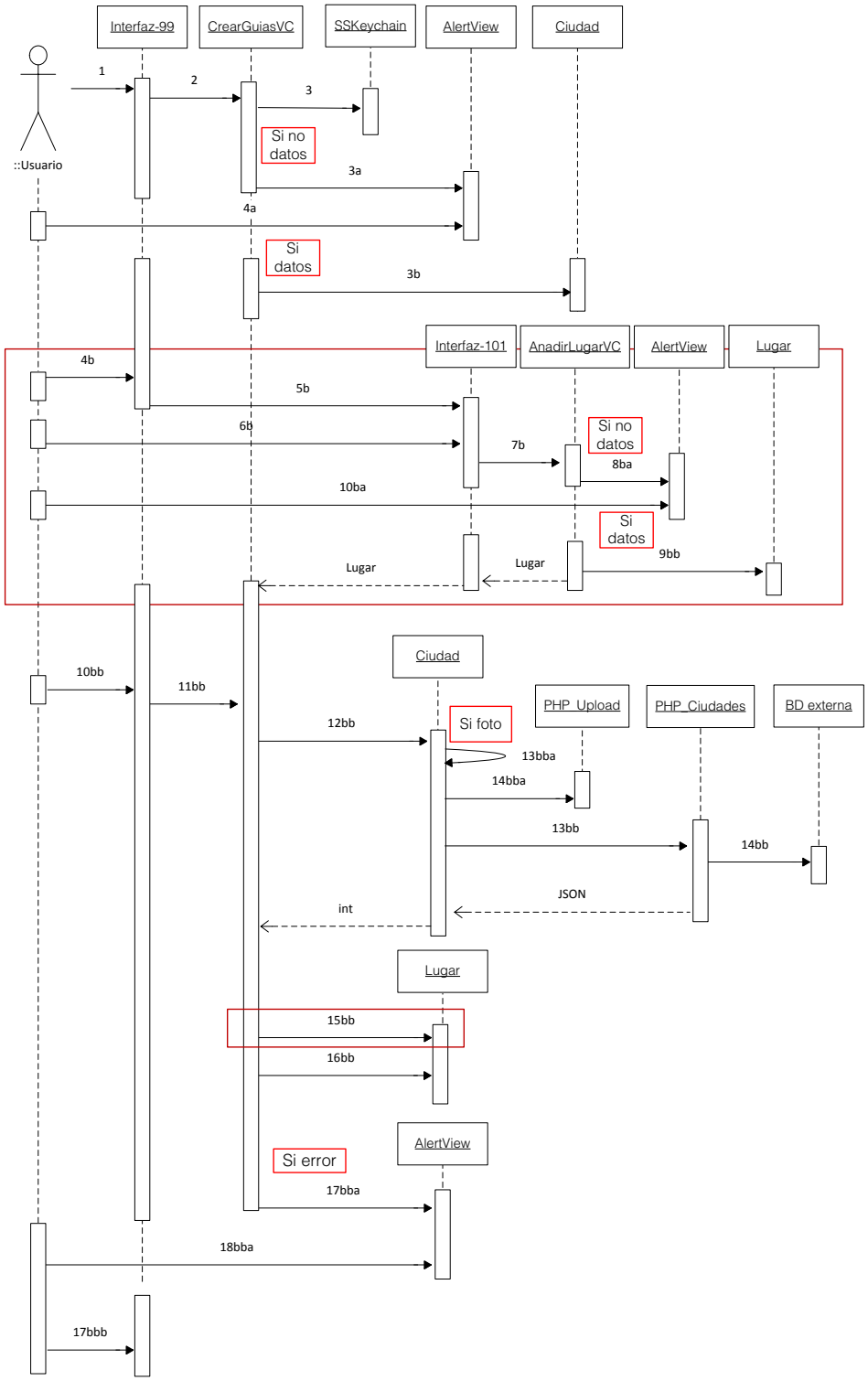


Ilustración 172.- Diagrama consultar guías 1

1- El usuario introduce los datos de la ciudad y pulsa en siguiente.

2- pulsarSiguiente(): void

[Si no ha introducido todos los datos]

2a- new()

3a- El usuario pulsa ok.

[Si ha introducido los datos]

2b- getNickUsuario(): string

3b- init(): Ciudad

4b- El usuario pulsa en "Siguiente", y después pulsa en el botón de "Añadir Lugar".

5b- new()

6b- El usuario introduce los datos y pulsa finalizar.

7b- pulsarBotonFinalizar(): Lugar

[Si datos insuficientes]

8ba- new()

9ba- El usuario pulsa ok.

[Si datos]

9bb- init(): Lugar

10bb- El usuario pulsa en el botón "Siguiente"

11bb- pulsarSiguiente(): void

12bb- insertarCiudad():void

[Si foto]

13bba- subirFoto(): void

14bba- subirArchivo(imagen): string

13bb- insertarCiudad(funcion, nombre, país, lugares, direccion): void

14bb- execSQL("INSERT INTO ciudades (nombre,pais,puntuacion,lugares,fotos,videos,rutas,metro,direccion) VALUES ('$nommod','$paimod',0,%lugares%,0,0,0,0,%direccion%)")

15bb- insertarLugar(): void

16bb- consultarLugares(): ListaLugares

[Si error]

17bba- new()

18bba- El usuario pulsa ok.

2ºdiagrama:

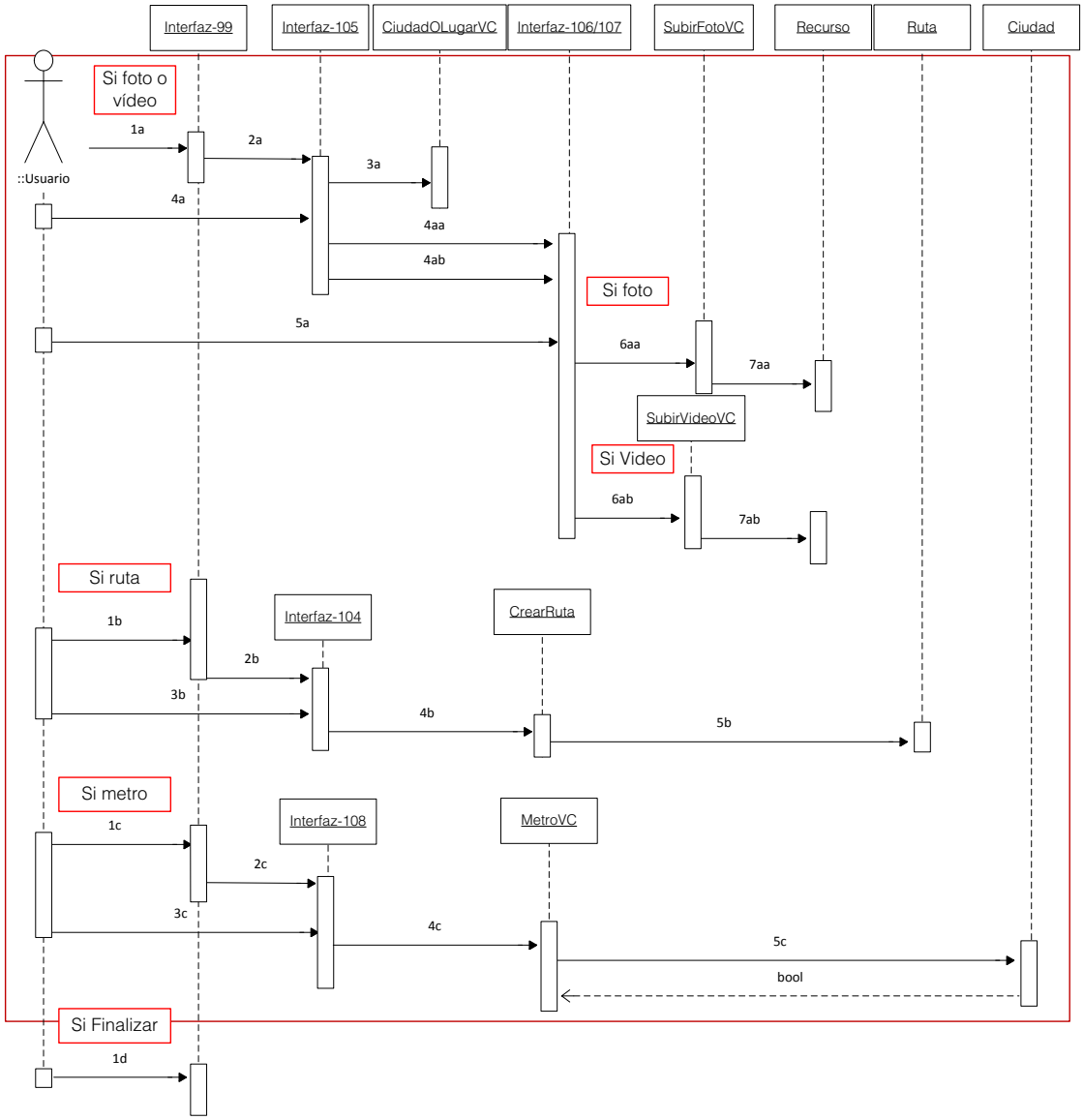


Ilustración 173.- Diagramas consultar guías 2

Una vez introducido la ciudad y los lugares, el usuario puede pulsar en los botones *Subir foto, Subir vídeo, Crear ruta, Subir metro*.

[Si foto o vídeo]

1a- El usuario pulsa en el botón Subir Foto o Subir Vídeo.

2a- new()

3a- inicializar(ciudad, listaLugares): void

4a- El usuario elige entre la ciudad o los lugares que ha creado.

[Si elige ciudad]

4aa- new(ciudad)

[Si elige un lugar]

4ab- new(lugar)

5a- El usuario introduce los datos en los campos.

[Si foto]

6aa- pulsarFinalizar(): void

7aa- subirFoto(): void

[Si vídeo]

6ab- pulsarFinalizar(): void

7ab- subirVideo():void

[Si ruta]

1b- El usuario pulsa el botón "Crear ruta".

2b- new()

3b- El usuario introduce todos los datos y pulsa finalizar.

4b- pulsarFinalizar(): void

5b- insertarRuta(): void

[Si metro]

1c- El usuario pulsa el botón "Subir metro".

2c- new()

3c- El usuario elige una foto y pulsa "Subir".

4c- pulsarSubir(): void

5c- subirMetro(imagen): bool

[Si finalizar]

1d- El usuario pulsa el botón "Finalizar" y se cierra la vista.

23- Crear Lugar

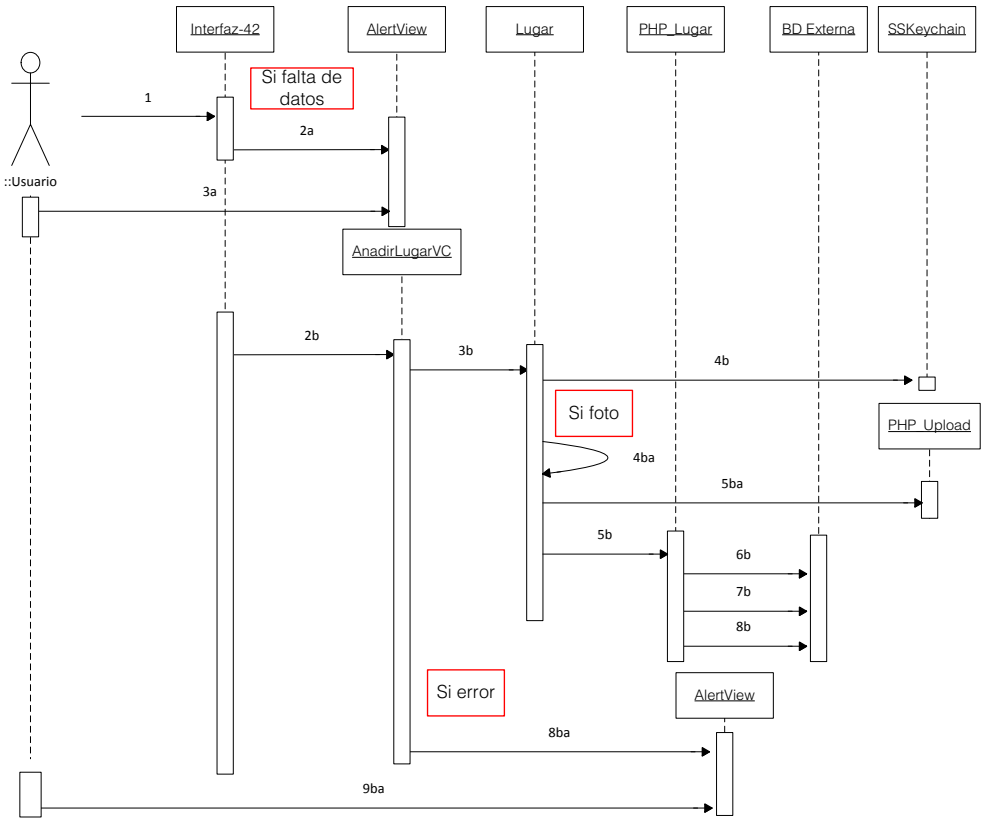


Ilustración 174.- Diagrama crear lugar

1-El usuario introduce los datos en los campos habilitados para ello.

[Si falta algún dato]

2a- new()

3a- El usuario pulsa OK.

[Si introducido correctamente]

2b- pulsarFinalizar(): IBAction

3b- insertarLugar(): void

4b- getNombreUsuario(): string

[Si foto de perfil]

4ba- descargarFotoPerfil(): void

5ba- subirArchivo(imagen): string

5b- insertarLugar (funcion, id_ciudad, nombre,autor,horario,precio,descripcion,nombreFoto): JSON

6b- execSQL("INSERT INTO Objetos (id) VALUES (NULL)");

7b- execSQL("INSERT INTO Lugares (id_objeto,id_ciudad,nombre,fotos,videos,puntuacion,autor,horario,precio,desc

ripcion,direccion) VALUES (%id_objeto%, %id_ciudad%, %nombre%, %fotos%, %videos%, %puntuacion%, %autor%, %horario%, %precio%, %descripcion%,%direccion%)");

8b- execSQL("UPDATE ciudades SET lugares = (SELECT count(nombre) FROM Lugares WHERE id_ciudad= %id_ciudad%);

[si error]

8ba- new()

9ba- El usuario pulsa ok.

24-Crear Ruta

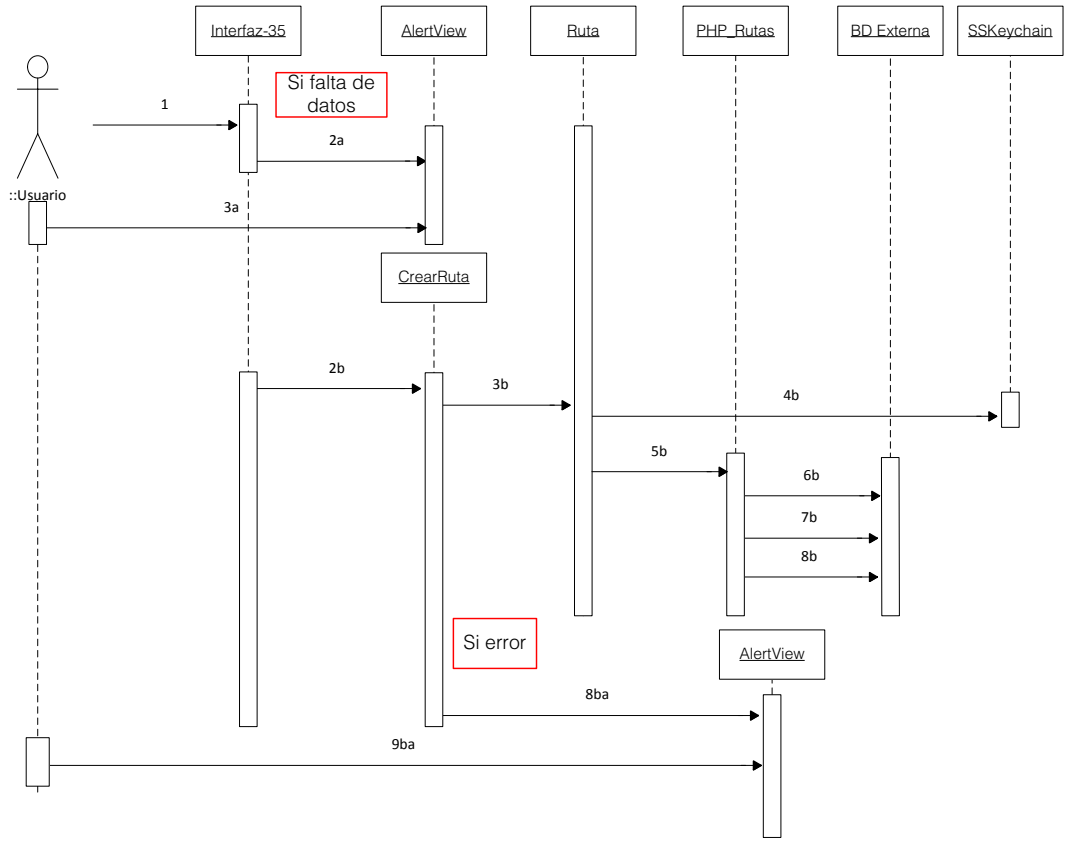


Ilustración 175.- Diagrama crear ruta

1-El usuario introduce los datos en los campos habilitados para ello.

[Si falta algún dato]

2a- new()

3a- El usuario pulsa OK.

[Si introducido correctamente]

2b- pulsarFinalizar() :IBAction

3b- insertarRuta(): Bool

4b- getNombreUsuario(): string

5b- insertarRuta (funcion, id_ciudad, nombre, días, autor,descripcion): JSON

6b- execSQL("INSERT INTO Objetos (id) VALUES (NULL)");

7b- execSQL("INSERT INTO Rutas (id_objeto,id_ciudad,nombre,dias,autor,descripcion) VALUES (%id_objeto%, %id_ciudad%, %nombre%, %dias% ,%autor%, %descripcion%)");

8b- execSQL("UPDATE ciudades SET rutas = (SELECT count(id_ruta) FROM Rutas WHERE id_ciudad= %id_ciudad%);

[si error]

8ba- new()

9ba- El usuario pulsa ok.

25- Descarga Metro

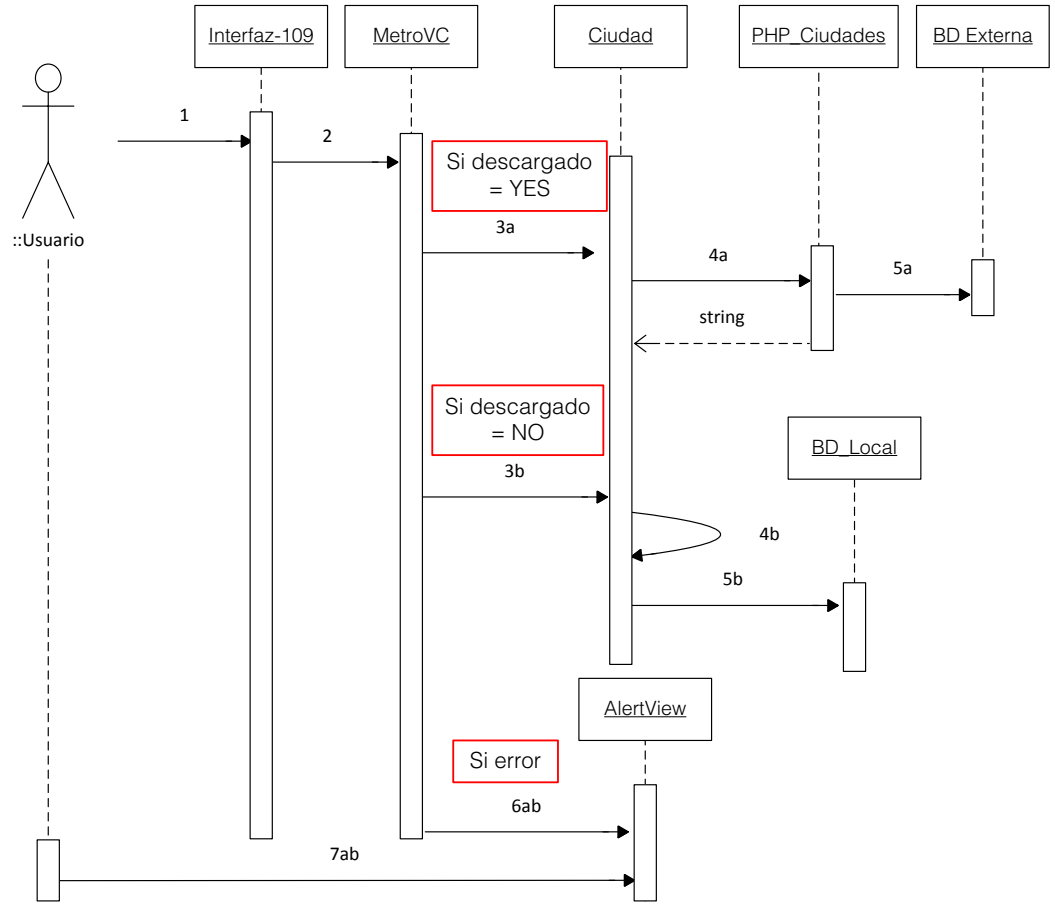


Ilustración 176.- Diagrama descarga metro

1- El usuario pulsa descargar.

2- pulsarBotonDescargar(): IBAction;

[si ciudad descargada]

3a- actualizarMetro(): void

4a- descargaDeMetro(direccion): string

5a- execSQL("UPDATE Ciudades SET metro='%@' WHERE idc=%id_ciudad%");

[si ciudad no descargada]

3b- descargarCiudad(): void;

4b- descargaDeMetro(direccion): string

5b- execSQL("INSERT INTO Ciudades (idc,nombre,pais,lugares,fotos,videos,rutas,metro,direccion) VALUES (%idc%, %nombre%, %pais%, %lugares%, %fotos%, videos%, %rutas%, %metro%, %direccion%)");

[si error]

6ab- new()

7ab- El usuario pulsa ok.

26- Descargar Ciudad

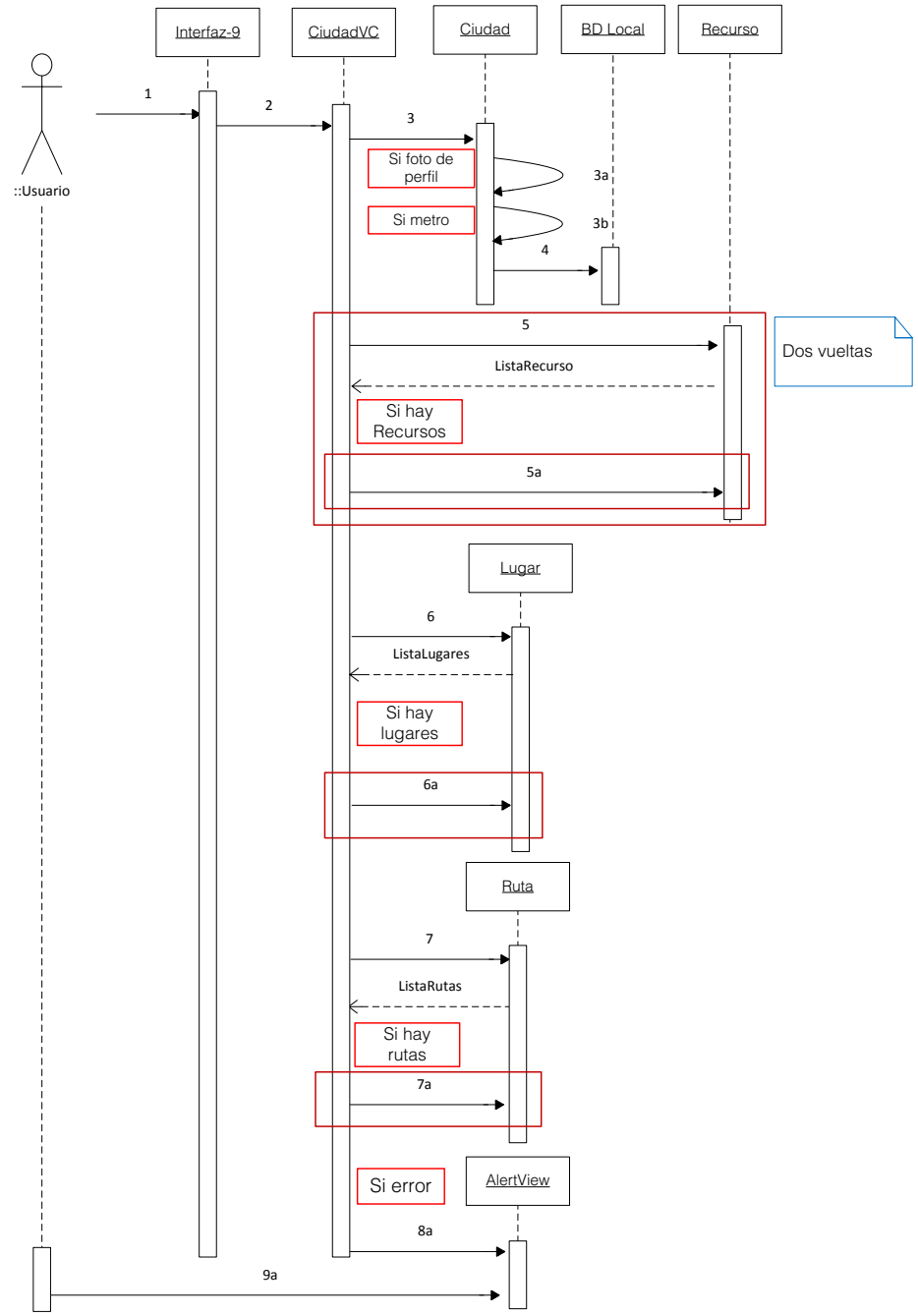


Ilustración 177.- Diagrama descargar ciudad

1- El usuario pulsa el botón Descargar.

2- pulsarBotonDescargar(id): IBAction

3- descargarCiudad(): void

[Si foto de perfil]

3a- descargaDeImagen(): void

[si metro]

3b-descargaDeMetro(direccion): string

4- execSQL("INSERT INTO Ciudades (idc, nombre, pais, lugares, fotos, videos, rutas, metro, direccion) VALUES (%idc%, %nombre%, %pais%, %lugares%, %fotos%, %videos%, %rutas%, %metro%, %direccion%)");

5-consultarRecursos(0) conRecurso(fotoOVideo): ListaRecursos

[Si hay recursos]

5a- insertar(fotoOVideo): void

6- consultarLugares(indicador): ListaLugares

[Si hay lugares]

6a- decargarLugar(): void

7- consultarRutas(indicador): ListaLugares

[Si hay Rutas]

7a- decargarRuta(): void

[si error]

8a- new()

9a- El usuario pulsa ok.

27- Descargar foto y vídeo

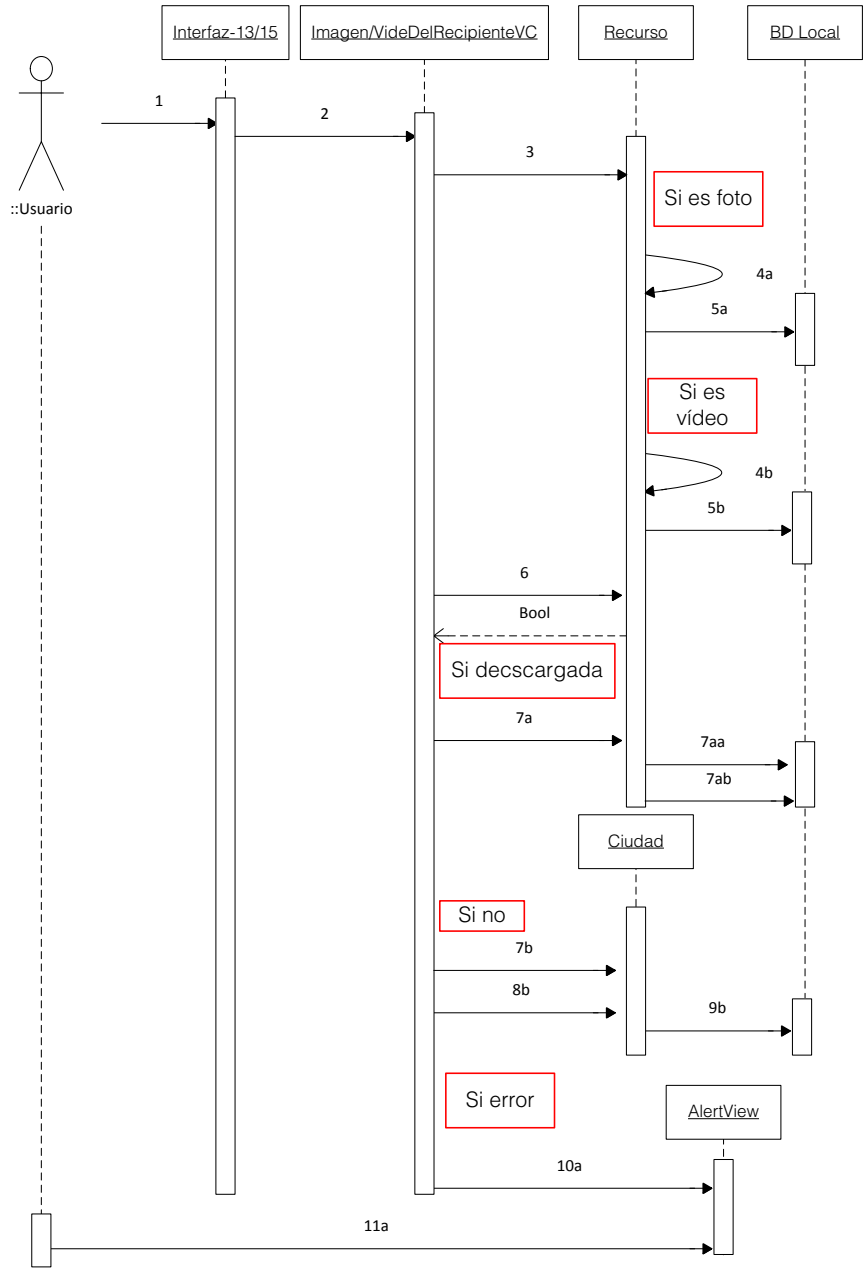


Ilustración 178.- Diagrama descargar foto y vídeo

1- El usuario pulsar descargar.

2- pulsarBotonDescargar(id): IBAction

3- insertar(fotoOVideo): void

[Si es foto]

4a- descargarRecurso(): void

5a- exceSQL("INSERT INTO Fotos (id_foto, id_lugar, id_ciudad, direccion, autor, id_objeto) VALUES (%id_foto%, %id_lugar%, %id_ciudad%, %direccion%, %autor%, %id_objeto%)")

[Si es vídeo]

4b- descargarRecurso(): void

5b- exceSQL("INSERT INTO Videos (id_video, id_lugar, id_ciudad, direccion, autor, id_objeto) VALUES (%id_foto%, %id_lugar%, %id_ciudad%, %direccion%, %autor%, %id_objeto%)")

6-comprobarCiudad(): bool

[Si ciudad descargada]

7a- actualizarCiudadDescargada(): void

[si es foto]

7aa- execSQL("UPDATE Ciudades SET fotos=(SELECT count(id_foto) FROM Fotos WHERE id_ciudad=%id_ciudad%)");

[si es vídeo]

7ab- execSQL("UPDATE Ciudades SET videos=(SELECT count(id_video) FROM Videos WHERE id_ciudad=%id_ciudad%)");

[sino]

7b- consultarUnaCiudad(id_ciudad): Ciudad

8b-descargarCiudad(): void

9b- execSQL("INSERT INTO Ciudades (idc, nombre, pais, lugares, fotos, videos, rutas, metro, direccion) VALUES (%idc%, %nombre%, %pais%, %lugares%, %fotos%, %videos%, %rutas%, %metro%, %direccion%)");

[Si error]

10a-new()

11a-El usuario pulsa OK.

28- Descargar Lugar

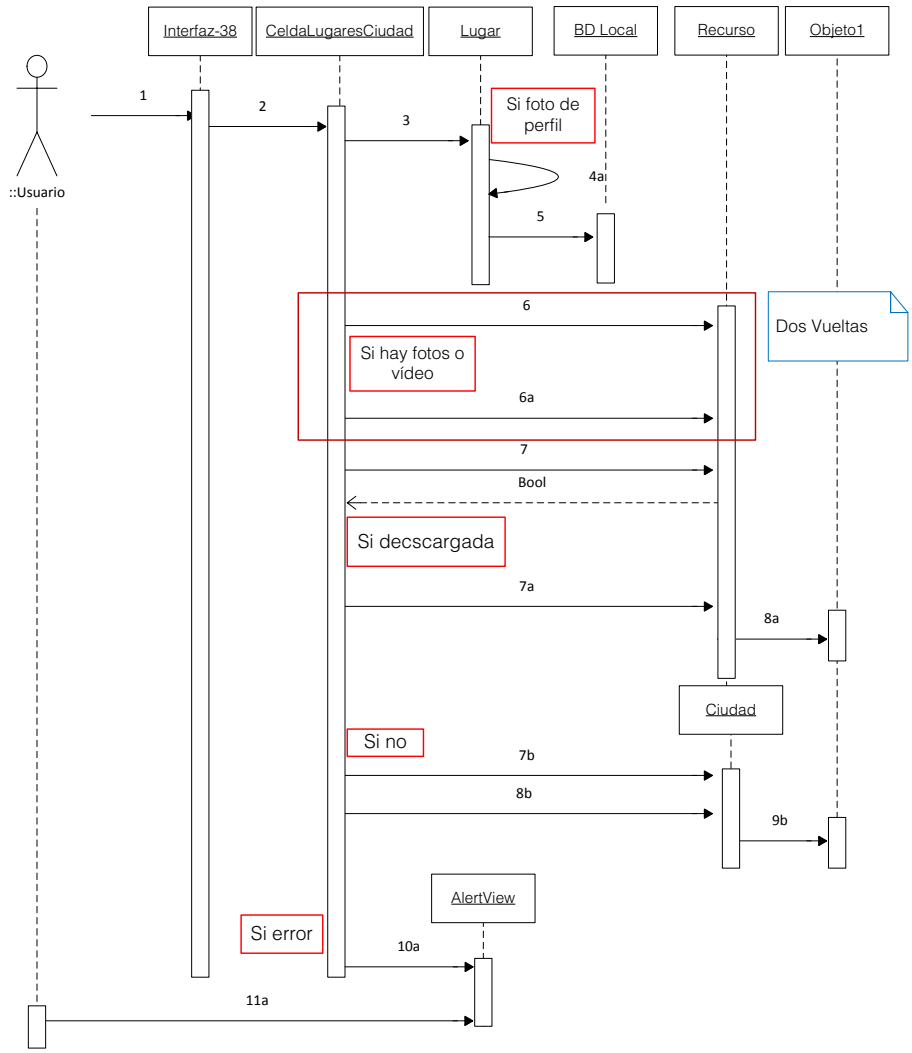


Ilustración 179.- Diagrama descargar lugar

1-El usuario pulsa Descargar.

2- insertarLugar(id): IBAction

3- descargarLugar(): void

[Si foto de perfil]

4a- descargaFotoPerfil(): void

5- execSQL("INSERT INTO Lugares (id_lugar, id_ciudad, nombre, fotos, videos, autor, horario, precio, descripcion, direccion, id_objeto) VALUES(%id_lugar%, %id_ciudad%, %nombre%, %fotos%, %videos%, %autor%, %horario%, %precio%, %descripcion%, %direccion%, %id_objeto%)")

6-consultarRecursos(id_lugar) conRecurso(fotoOVideo): ListaRecursos

[Si hay recursos]

6a- insertar(fotoOVideo): void

7-comprobarCiudad(): bool

[Si ciudad descargada]

7a- actualizarCiudadDescargada(): void

8a- execSQL("UPDATE Ciudades SET lugares=(SELECT count(id_lugar) FROM Lugares WHERE id_ciudad=%id_ciudad%)");

[sino]

7b- consultarUnaCiudad(id_ciudad): void

8b- descargarCiudad(): void

9b- execSQL("INSERT INTO Ciudades (idc, nombre, pais, lugares, fotos, videos, rutas, metro, direccion) VALUES (%idc%, %nombre%, %pais%, %lugares%, %fotos%, %videos%, %rutas%, %metro%, %direccion%)");

[Si error]

10a- new()

11a- El usuario pulsa ok.

29- Descargar Ruta

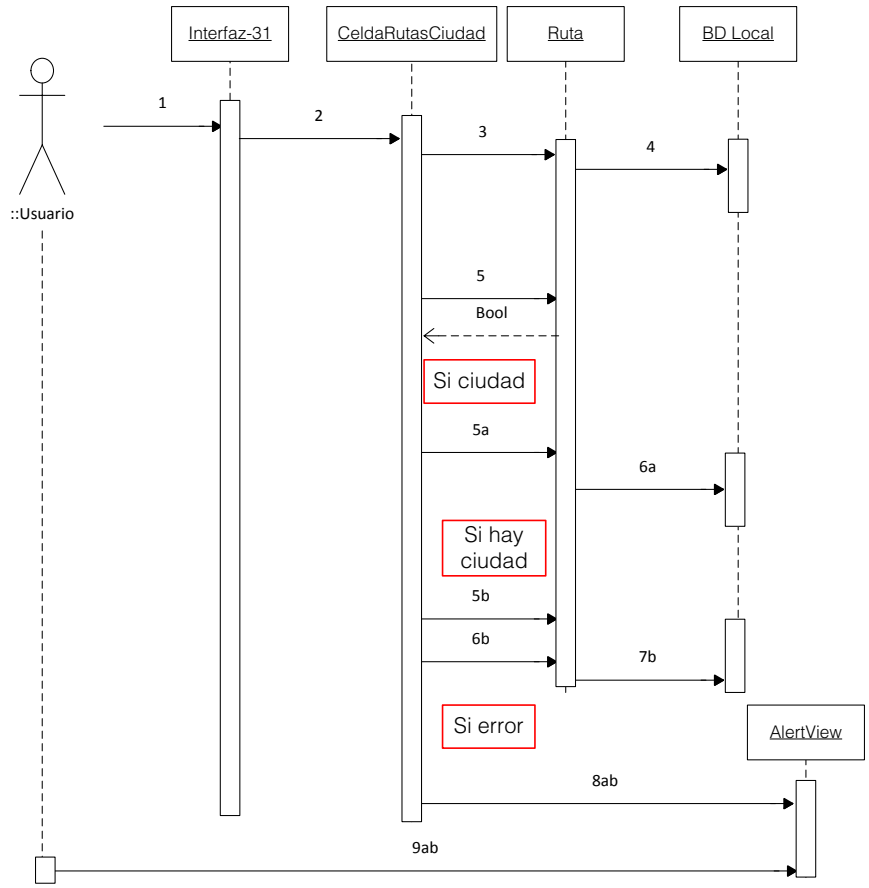


Ilustración 180.- Diagrama descargar ruta

1- El usuario pulsa Descargar.

2- insertarRuta(id): IBAction

3- descargarRuta(): void

4- execSQL("INSERT INTO Rutas (id_ruta, id_ciudad, nombre, dias, autor, descripcion, id_objeto) VALUES (%id_ruta%, %id_ciudad%, %nombre%, %dias%, %autor%, %descripcion%, %id_objeto%)")

5-comprobarCiudad(): bool;

[Si ciudad descargada]

5a- actualizarCiudadDescargada(): void;

6a- execSQL("UPDATE Ciudades SET rutas=(SELECT count(id_ruta) FROM Rutas WHERE id_ciudad=%id_ciudad%)");

[sino]

5b- consultarUnaCiudad(id_ciudad): void;

6b-descargarCiudad(): void;

7b-execSQL("INSERT INTO Ciudades (idc, nombre, pais, lugares, fotos, videos, rutas, metro, direccion) VALUES (%idc%, %nombre%, %pais%, %lugares%, %fotos%, %videos%, %rutas%, %metro%, %direccion%)");

[si error]

8ab-new()

9ab-El usuario pulsa OK.

30- Editar Perfil

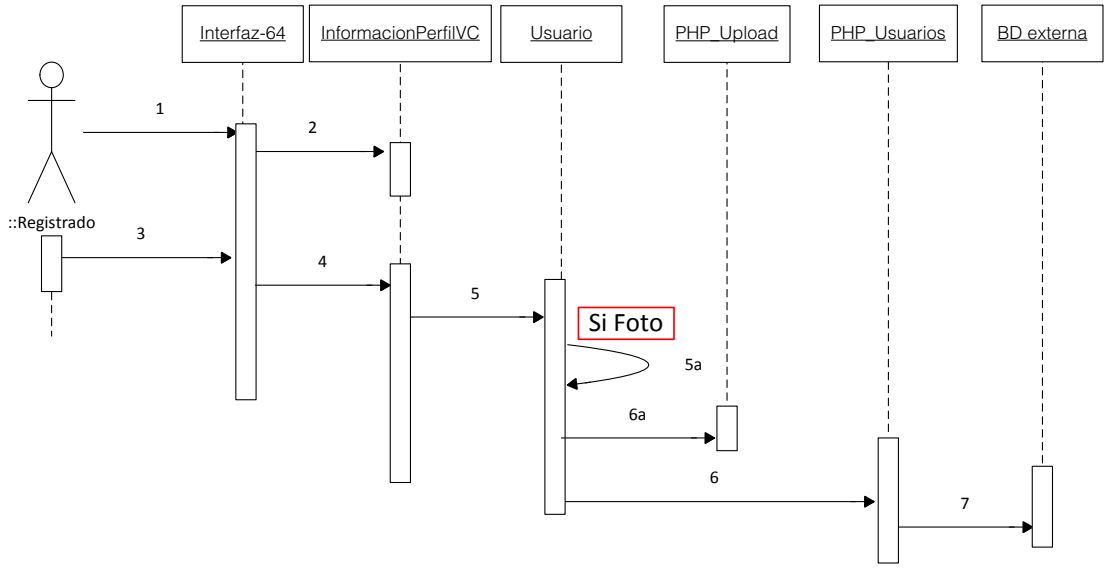


Ilustración 181.- Diagrama editar perfil

1- El usuario pulsar en Editar

2- pulsarEditar(id): IBAction

3- El usuario modifica los campos que quiera cambiar y pulsa en "Hecho".

4- pulsarEditar(id): IBAction

5- actualizarUsuarioConInfo(post): bool

[Si cambió foto perfil]

5a- subirFoto(imagen): string

6a- subirFoto(archivo): string

6- actualizarUsuario(): void

7- execSQL("UPDATE Usuarios SET email=%email%, nombre=%nombre%, apellido=%apellido%, fechaNac=%fechaNac%, sexo=%sexo%, fotoPerfil=%fotoPerfil%, telefono=%telefono%, ciudad=%ciudad%, intereses=%intereses%, viajes=%viajes%, lugares=%lugares%, privado=%privado% WHERE id_usuario=%id_usuario%")

31- Enviar Mensajes

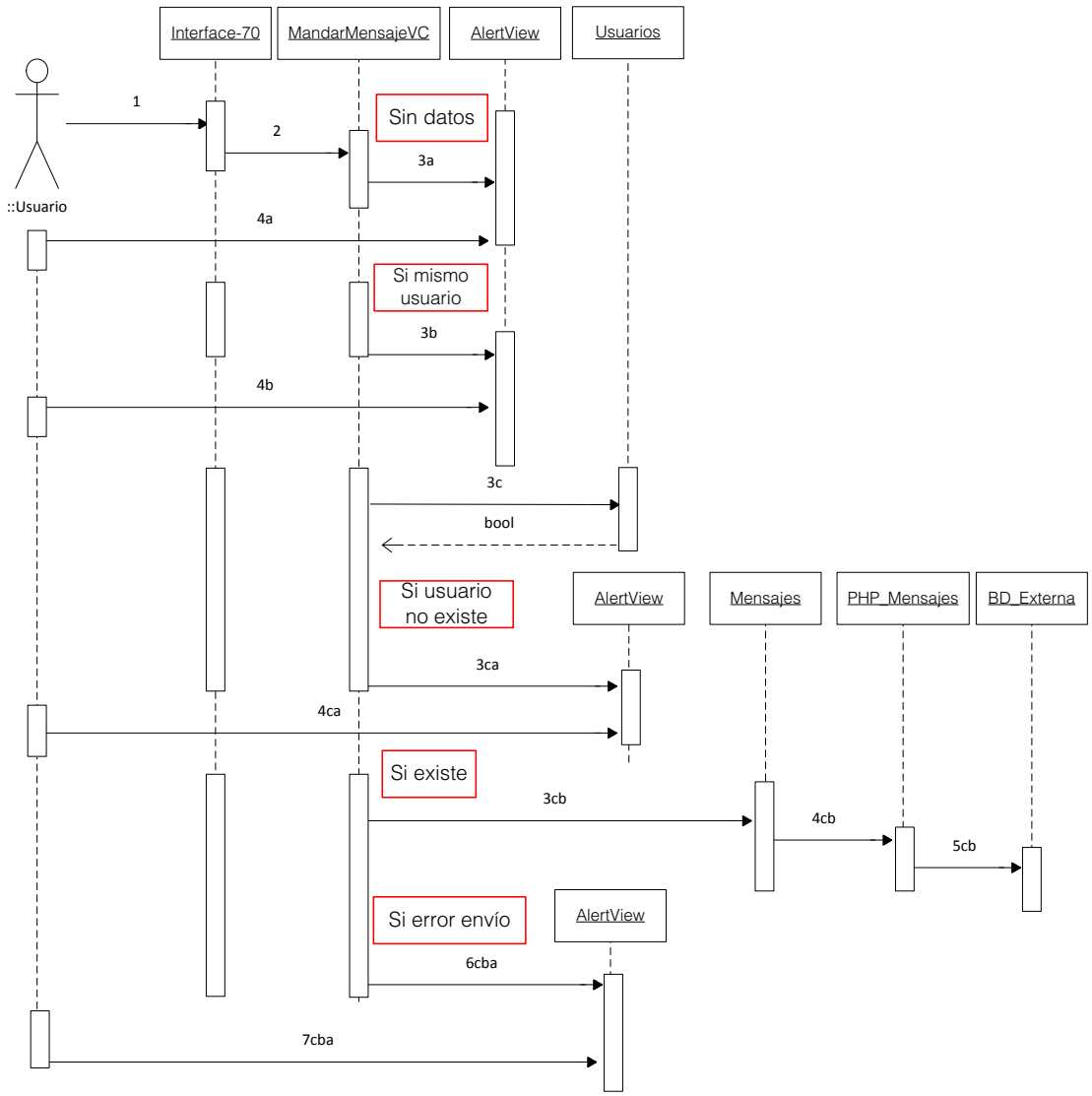


Ilustración 182.- Diagrama enviar mensajes

1- El usuario pulsar "Enviar".

2- pulsarEnviar(id): IBAction

[Sin datos]

3a- new()

4a- El usuario pulsa OK.

[El nombre del usuario en destinatario]

3b-new()

4b- El usuario pulsa OK.

[Si ok]

3c- consultarDatosUsuario(): bool

[Si el usuario no existe]

3ca- new()

4ca- El usuario pulsa OK.

[Si el usuario existe]

3cb- enviarMensaje(): void

4cb- enviarMensaje(función, fecha, asunto, emisor, enviado, recibidor, texto): void

5cb- execSQL("INSERT INTO Mensajes (fecha, asunto, enviado, recibidor, texto, leido) VALUES (%fecha%, %asunto%, %enviado%, %recibidor%, %texto%, %leido%, 0");

[Si error en envío]

6cba- new()

7cba- El usuario pulsa OK.

32- Escribir comentario

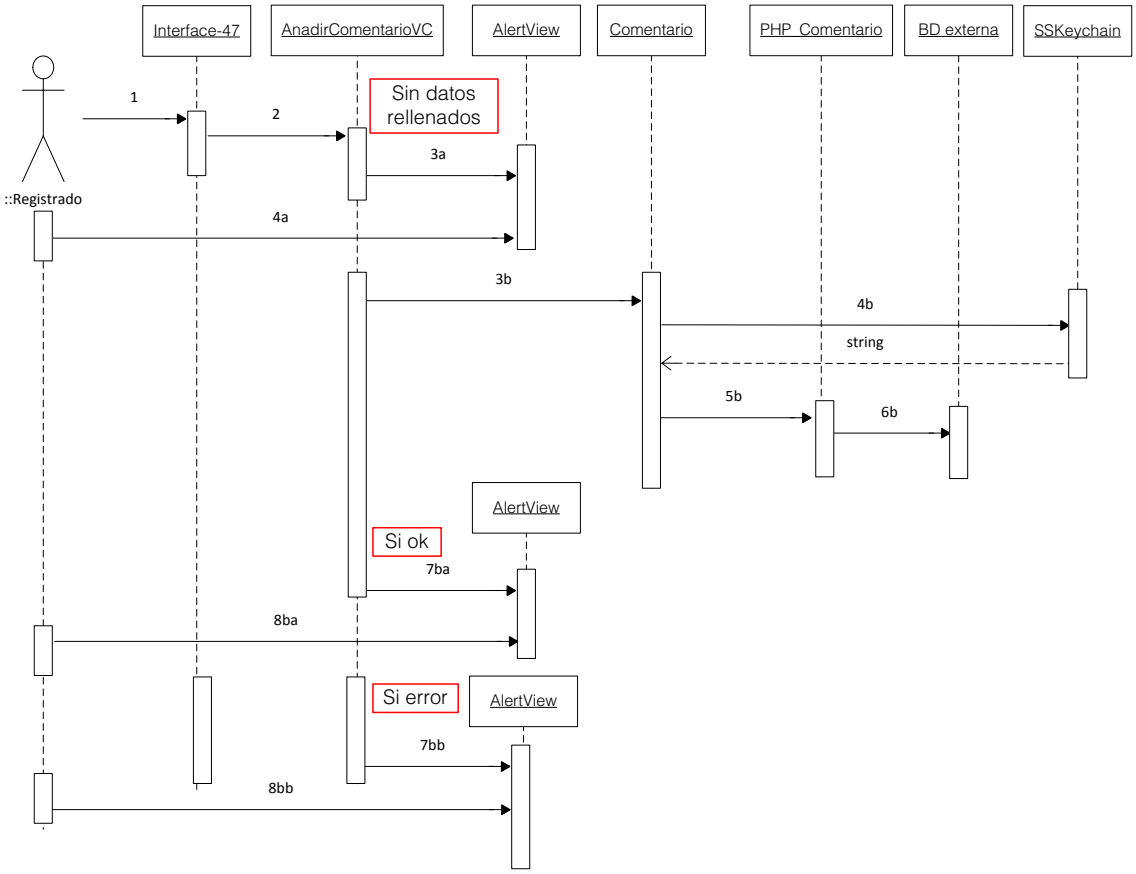


Ilustración 183.- Diagrama escribir comentario

1-El usuario rellena los campos y pulsa "Enviar"

2- pulsarBotonEnviar(id): IBAction

[sin datos rellenados]

3a- new()

4a- El usuario pulsa OK.

[si ok]

3b- anadirComentario(): bool

4b- getNick(): string

5b- insertarComentarios(funcion, id_lugar, titulo, autor, fecha, texto): void

6b- execSQL("INSERT INTO Comentarios (id_lugar, titulo, autor, fecha, comentario) VALUES (%id_lugar%, %titulo%, %autor%, %fecha%, %comentario%)")

[Si mandado]

7ba-new()

8ba- El usuario pulsa Ok.

[Si error]

7bb-new()

8bb- El usuario pulsa Ok.

33- Gestionar amigos

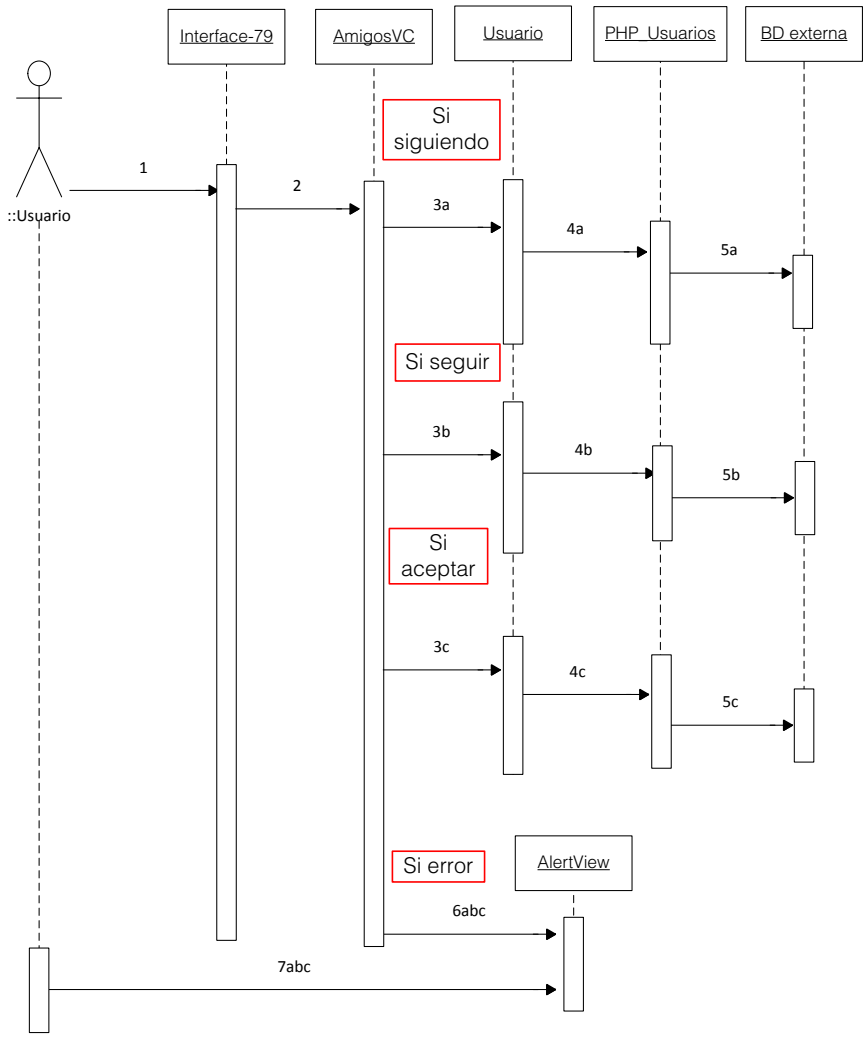


Ilustración 184.- Diagrama gestionar amigos

1- El usuario pulsar en el botón para seguir las amistades o dejar de seguirlas.

2- pulsarBotonSeguir(id): IBAction

[Si siguiendo]

3a- unFollowAmigo(id_usuario): void

4a- unfollow(id_usuario,id_usuario_amigo, funcion): void

5a- execSQL("DELETE FROM Amistades WHERE id_usuario1=%id_usuario1% and id_usuario2=%id_usuario2%")

[Si seguir]

3b- followAmigo(id_usuario) conEstado(privado): void

4b- follow(id_usuario,id_usuario_amigo, estado, funcion): void

5b- execSQL("INSERT INTO Amistades (id_usuario1,id_usuario2,estado)
VALUES (%id_usuario1%, %id_usuario2%, %estado%)")

[Si aceptar]

3b- cambiarEstado(id_usuario): void

4b- cambiarEstadoAmistad(id_usuario,id_usuario_amigo, funcion):void

5b- execSQL("UPDATE Amistades SET estado=1 WHERE
id_usuario1=%id_usuario1% and id_usuario2=%id_usuario2%")

[si error]

6abc- new()

7abc- El usuario pulsa ok.

34-Iniciar Sesión

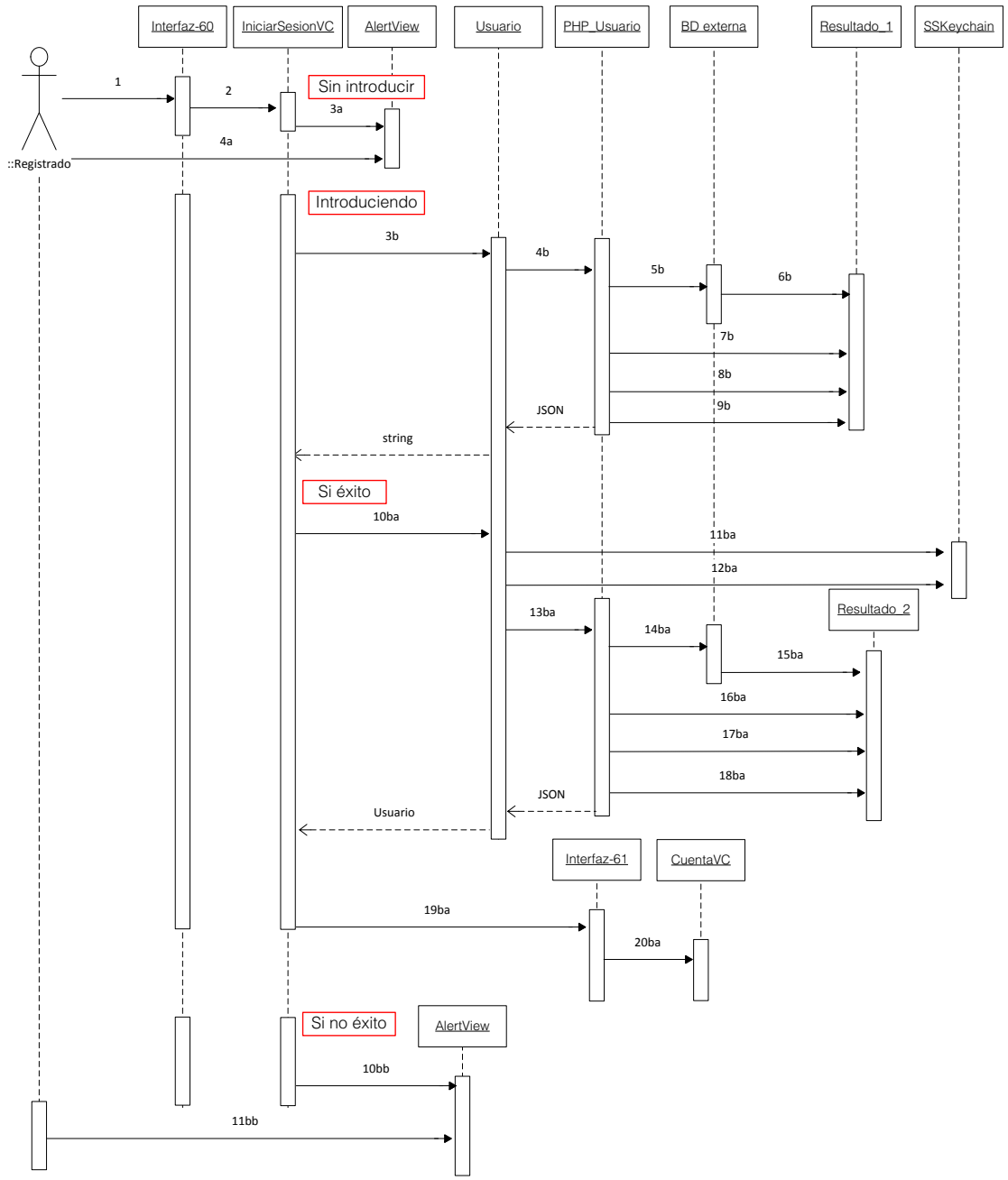


Ilustración 185.- Diagrama iniciar sesión

1- El usuario introduce las credenciales y pulsa "Entrar".

2- pulsarBotonEntrar(id): IBAction

[Sin introducir]

3a- new()

4a- El usuario pulsa OK.

[Comprobando]

3b- comprobarUsuario(): string

4b- comprobarSiExiste(nick, pass, funcion): JSON

5b- execSQL("SELECT * FROM Usuarios WHERE nick=%nick% AND pass=%pass%")

6b- new()

7b- next()

8b- getData():Data

9b- close()

[Si éxito]

10ba- setNombre(nick): void

11ba- setPass(pass): void

12ba- consultarDatosUsuario(): bool

13ba- consultarInfoUsuario(nick, funcion): JSON

14ba- execSQL("SELECT * FROM Usuarios WHERE nick=%nick%")

15ba- new()

16ba- next()

17ba- getData(): Data

18ba- close()

19ba- new(usuario)

20ba- inicializar(): void

[Si error]

10bb- new()

11bb- El usuario pulsa ok.

35- Puntuar/quitar puntuación

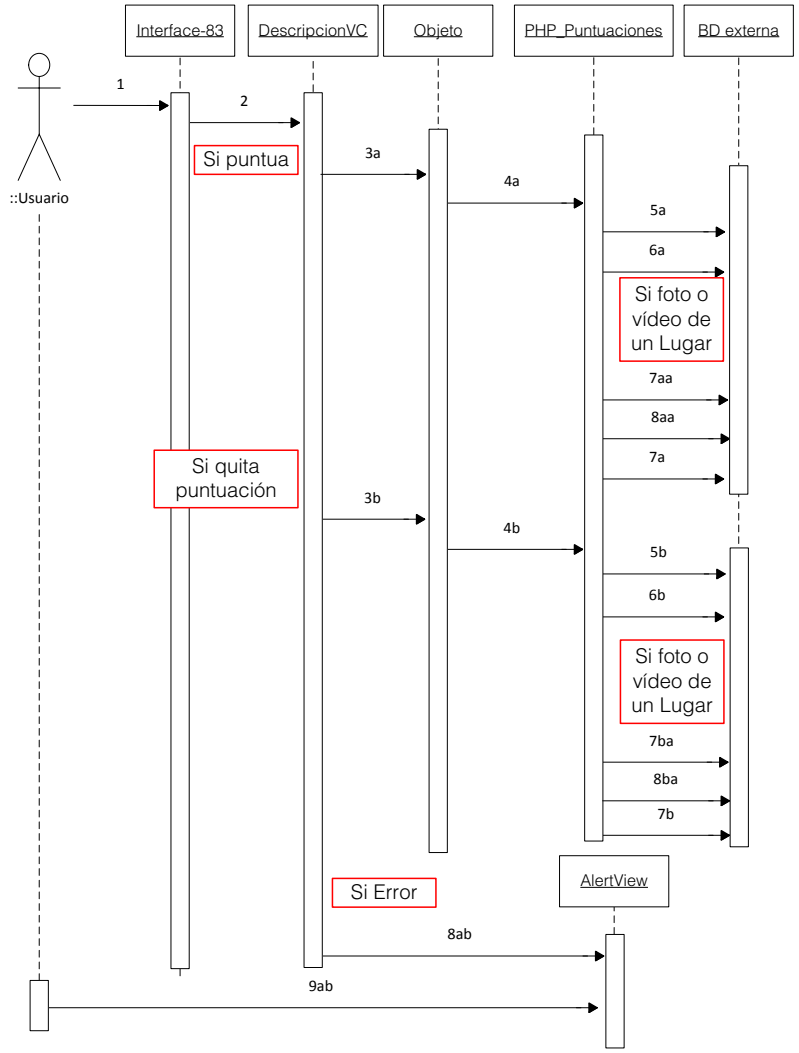


Ilustración 186.- Diagrama puntuar/quitar puntuación

1- El usuario pulsa sobre el botón de puntuar.

2- pulsarVotar(id): IBAction

[Si puntua]

3a- puntuar(tipoObjeto) ySiPerteneceAUnLugar(chivato): void

4a- puntuar(id_objeto, nick, chivato, tabla, id_ciudad, funcion): void

5a- execSQL("INSERT INTO Puntuaciones (id_objeto,nick_usuario) VALUES (%id_objeto%, %nick%)")

6a- execSQL("UPDATE %tabla% SET puntuacion = (SELECT COUNT(id_puntuacion) FROM Puntuaciones WHERE id_objeto=%id_objeto%) WHERE id_objeto=%id_objeto%")

[Si es una foto o vídeo de un lugar]

7aa- execSQL("SELECT id_objeto FROM Lugares WHERE id_lugar=%chivato%")

8aa- execSQL("UPDATE Lugares SET puntuacion = (SELECT (COALESCE((SELECT COUNT(id_puntuacion) FROM Puntuaciones WHERE id_objeto=%id_objeto%),0) + COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_lugar=%chivato%),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_lugar=%chivato%),0)) AS suma) WHERE id_lugar=%chivato%")

7a- execSQL("UPDATE ciudades SET puntuacion = (SELECT (COALESCE((SELECT SUM(puntuacion) FROM Lugares WHERE id_ciudad =%id_ciudad%),0) + COALESCE((SELECT SUM(puntuacion) FROM Rutas WHERE id_ciudad = %id_ciudad%),0)+ COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0)) AS suma) WHERE idc =%id_ciudad%")

[Si quita puntuacion]

3b- quitarPuntuacion(tipoObjeto) ySiPerteneceAUnLugar(chivato): void

4b- quitarPuntuacion(id_objeto, nick, chivato, tabla, id_ciudad, funcion): void

5b- execSQL("INSERT INTO Puntuaciones (id_objeto,nick_usuario) VALUES (%id_objeto%, %nick%)")

6b- execSQL("DELETE FROM Puntuaciones WHERE id_objeto=%id_objeto% and nick_usuario=%nick%")

[Si es una foto o vídeo de un lugar]

7ba- execSQL("SELECT id_objeto FROM Lugares WHERE id_lugar=%chivato%")

8ba- execSQL("UPDATE Lugares SET puntuacion = (SELECT (COALESCE((SELECT COUNT(id_puntuacion) FROM Puntuaciones WHERE id_objeto=%id_objeto%),0) + COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_lugar=%chivato%),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_lugar=%chivato%),0)) AS suma) WHERE id_lugar=%chivato%")

7b- execSQL("UPDATE ciudades SET puntuacion = (SELECT (COALESCE((SELECT SUM(puntuacion) FROM Lugares WHERE id_ciudad =%id_ciudad%),0) + COALESCE((SELECT SUM(puntuacion) FROM Rutas WHERE id_ciudad = %id_ciudad%),0)+ COALESCE((SELECT SUM(puntuacion) FROM Fotos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0) + COALESCE((SELECT SUM(puntuacion) FROM Videos WHERE id_ciudad =%id_ciudad% and id_lugar IS NULL),0)) AS suma) WHERE idc =%id_ciudad%")

[Si error]

8ab- new()

9ab- El usuario pulsa OK.

36-Registrarse

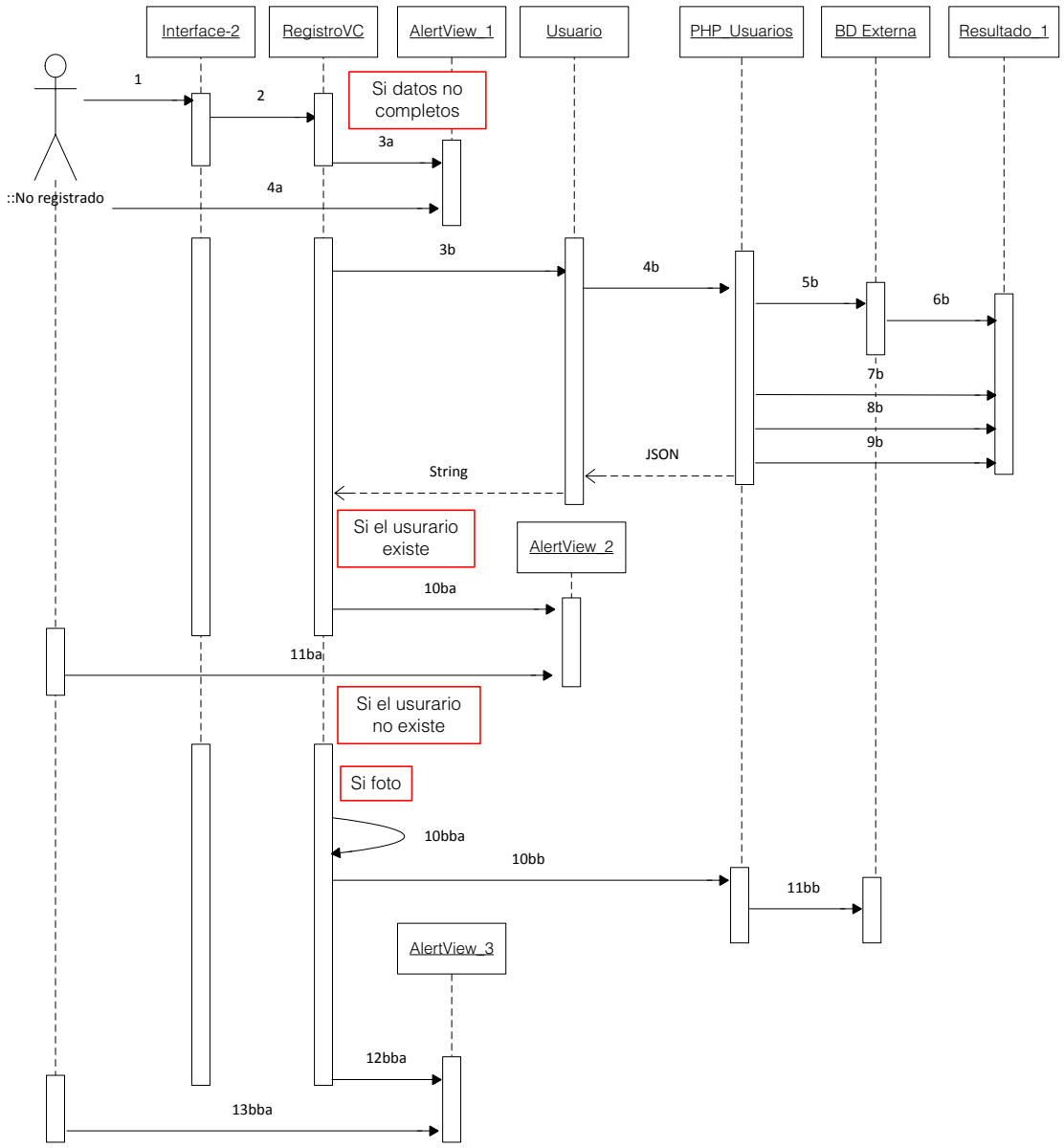


Ilustración 187.- Diagrama registrarse

1- El usuario introduce los datos en los campos

2- pulsarBotonFinalizar(id): IBAction

[Sin campos sin rellenar]

3a- new()

4a- El usuario pulsa OK.

[Si ok]

3b- registrarUsuarioConPass(password) eImagenPerfil(imagenPerfil): bool

4b- comprobarUsuario(): bool

5b- comprobarUsuario(funcion, nick): string

6b- execSQL(“SELECT id_usuario FROM Usuarios WHERE nick=%nick%”)

7b- new()

8b- next()

9b- getData():Data

10b- close()

[Si el usuario existe]

10ba- new()

11ba- El usuario pulsa ok

[Si el usuario no existe]

[si hay foto de perfil]

10bba- subirFoto(imagenPerfil): string

10bb- insertarUsuario(nick, email,pass,nombre,apellido,fecha,sexo,foto)

11bb- execSQL(“INSERT INTO Usuarios (nick, pass, email, nombre, apellido, fechaNac, sexo, fotoPerfil) VALUES (%nick%, %pass%, %email%, %nombre%, %apellido%, %fechaNac%, %sexo%, %fotoPerfil%)”)

[Si error]

12bba- new()

13bba- El usuario pulsa ok.

37- Subir Rercurso

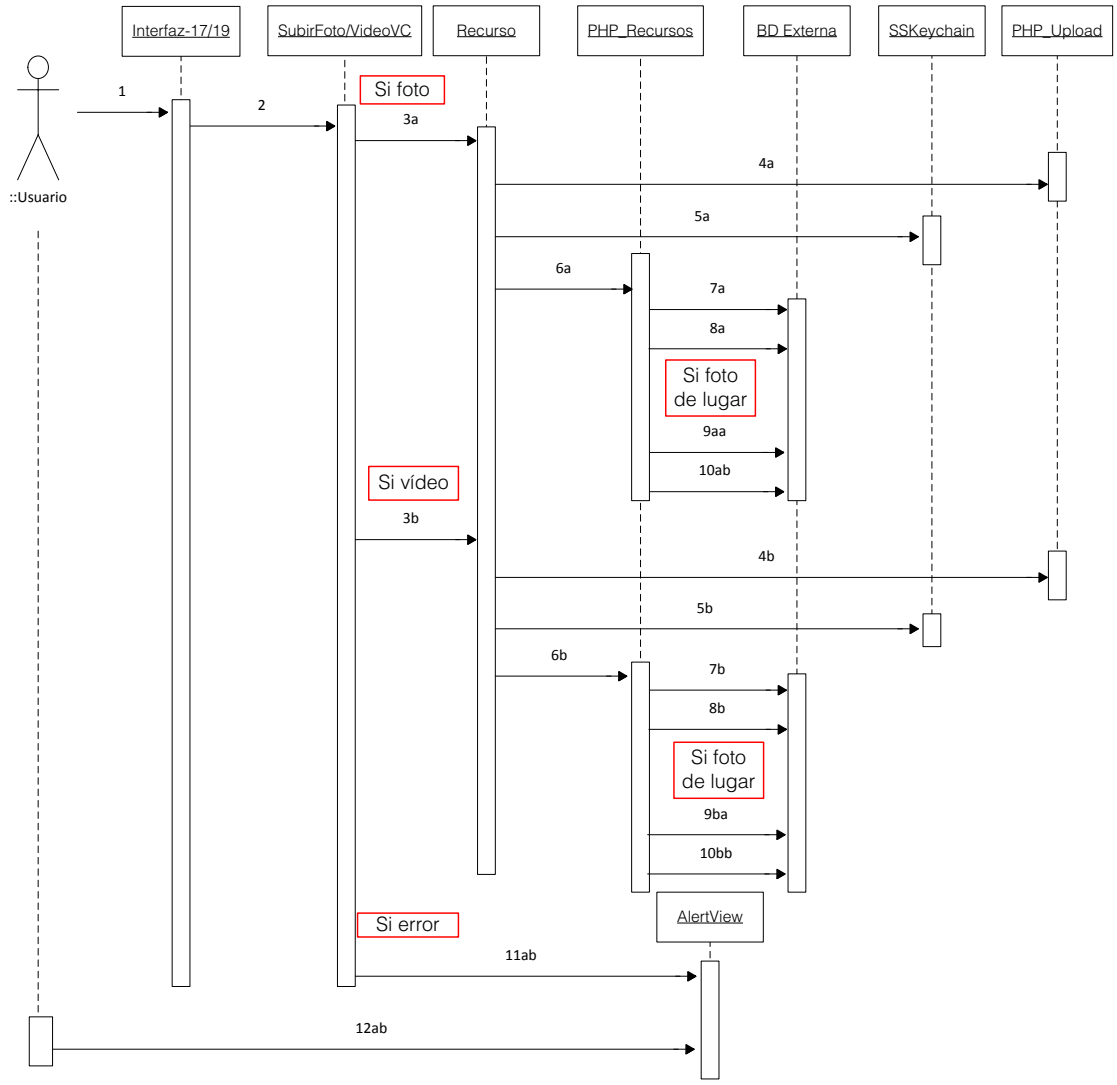


Ilustración 188.- Diagrama subir recurso

1-El usuario ha elegido una foto o un vídeo para subir y pulsar en el botón de subir.

2- pulsarBotonSubir(id): IBAction

[Si foto]

3a- subirFoto(): bool

4a- subirArchivo(): void

5a- getNick(): string

6a- insertarFoto(id_ciudad, id_lugar, nombre, autor): void

7a- execSQL("INSERT INTO Objetos (id) VALUES (NULL)")

8a- execSQL("INSERT INTO Fotos (id_objeto, id_lugar, id_ciudad, puntuacion, direccion, autor) VALUES (%id_objeto%, %id_lugar%, %id_ciudad%, %puntuacion%, %direccion%, %autor%)")

[Si es foto de un lugar]

9aa- execSQL("UPDATE Lugares SET fotos=(SELECT count(id_foto) FROM Fotos WHERE id_lugar=%id_lugar%) WHERE id_lugar=%id_lugar%")

[Si es foto de una ciudad]

10ab- execSQL("UPDATE Ciudades SET fotos=(SELECT count(id_foto) FROM Fotos WHERE id_ciudad=%id_ciudad% and id_lugar is NULL) WHERE id_ciudad=%id_ciudad%")

[Si vídeo]

3b- subirVideo(): bool

4b- subirArchivo(): void

5b- getNick(): string

6b- insertarVideo(id_ciudad, id_lugar, nombre, autor): void

7b- execSQL("INSERT INTO Objetos (id) VALUES (NULL)")

8b- execSQL("INSERT INTO Videos (id_objeto, id_lugar, id_ciudad, puntuacion, direccion, autor) VALUES (%id_objeto%, %id_lugar%, %id_ciudad%, %puntuacion%, %direccion%, %autor%)")

[Si es vídeo de un lugar]

9ba- execSQL("UPDATE Lugares SET videos=(SELECT count(id_video) FROM Fotos WHERE id_lugar=%id_lugar%) WHERE id_lugar=%id_lugar%")

[Si es vídeo de una ciudad]

10bb- execSQL("UPDATE Ciudades SET videos=(SELECT count(id_video) FROM Fotos WHERE id_ciudad=%id_ciudad% and id_lugar is NULL) WHERE id_ciudad=%id_ciudad%")

[si error]

11ab- new()

12ab- El usuario pulsa ok.

38-Subir Metro

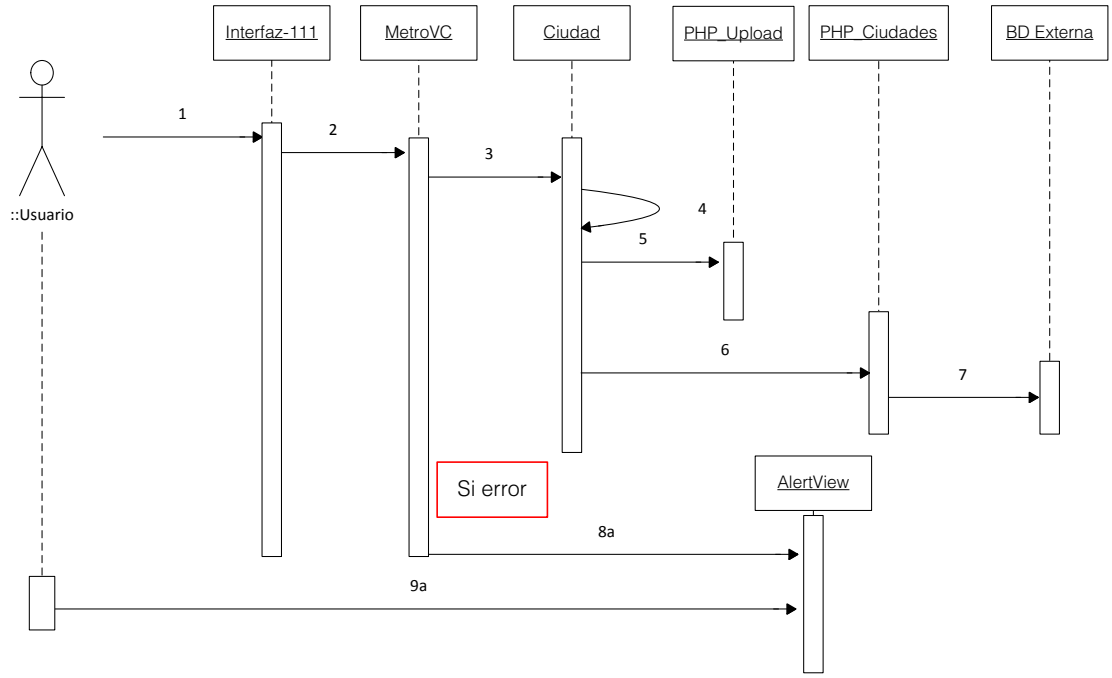


Ilustración 189.- Diagrama subir metro

1- El usuario ha elegido una foto y pulsar en subir.

2- pulsarBotonSubir(id): IBAction

3- actualizarMetro(direccionMetro)

4- subirMetro(imagen): void

5- subirArchivo(foto): void

6- actualizarMetro(id_ciudad, metro, nombre)

7- execSQL("UPDATE ciudades SET metro=%metro% WHERE idc=%id_ciudad%")

[Si error]

8a- new()

9a- El usuario pulsa OK.

39- Compartir Fotos

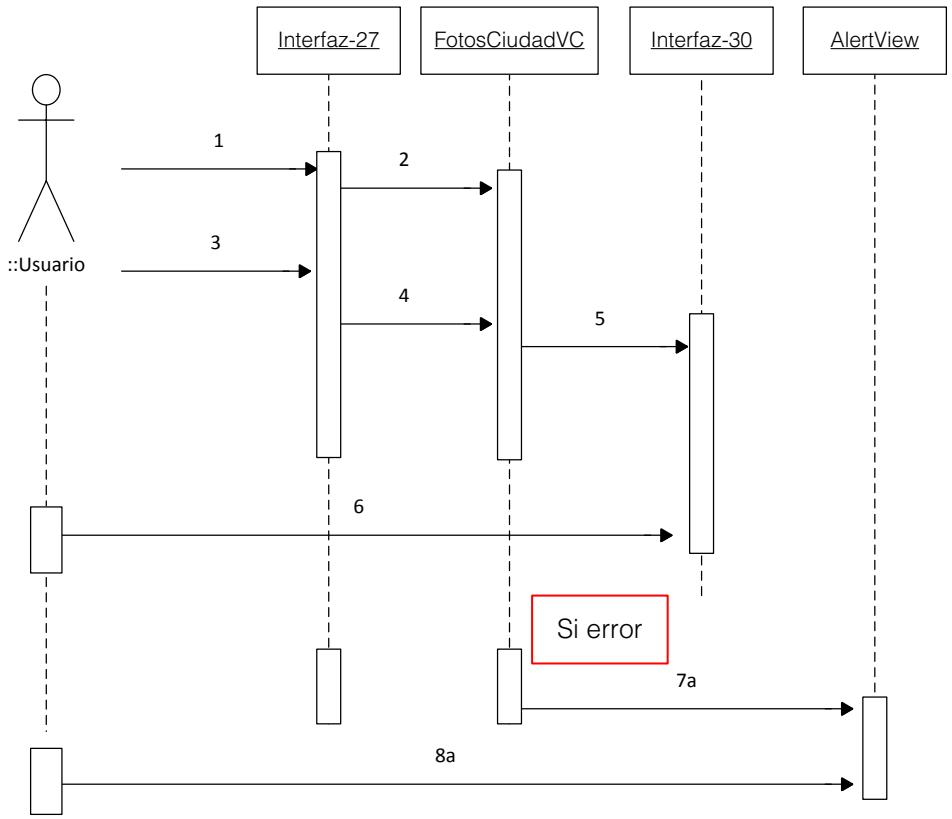


Ilustración 190.- Diagrama compartir fotos

1- El usuario pulsa en el botón "Seleccionar".

2- pulsarBotonSeleccionar(): void

3- El usuario pulsa sobre las imágenes que quiere compartir y pulsa en el botón "Compartir".

4- pulsarBotonCompartir(): void

5- new(listaFotos)

6- El usuario configura en esa vista como compartir las fotos en Facebook, introduciendo una explicación, su localización, etc. y pulsa "Post".

[Si hay error]

7a- new()

8a- El usuario pulsa ok.