Memoria
CanaryGuide

Alumno: Juan M Gómez Álvarez
Consultor: Marc Domingo Prieto

Introducción

En el presente documento se sintetizará el desarrollo de la aplicación móvil CanaryGuide.

La aplicación CanaryGuide es un proyecto real, esto significa que sobrepasa los límites de este TFC. El proyecto consta de dos partes diferenciadas: una página web y aplicaciones para consultar la información alojada en los servidores en modo offline. Se debe entender que el desarrollo de este proyecto como TFC es un subconjunto del proyecto final. Aunque se ha de destacar que, como los servidores no están implementados, se cogerá la información usando APIs de terceros como Google.

La aplicación pretende servir de guía turística y de ocio de bolsillo. Esto significa que ofrecerá respuesta a una serie de peticiones por parte del usuario, como puede ser “¿qué Parque Nacional tengo cerca?” o “¿qué playa se encuentra en los alrededores?”.

Como objetivo principal del proyecto se desarrollará la versión para Windows Phone y para iPhone. Un objetivo implícito en el presente proyecto es el desarrollo **nativo** multiplataforma usando para ello las tecnologías .NET y Mono, concretamente el SDK de Windows Phone y MonoTouch.

Se comenzará especificando la idoneidad del producto, a continuación se hablará de la planificación, seguido del análisis funcional para posteriormente especificar los requisitos de una manera formal. Posteriormente, se identificarán los casos de uso del sistema y se expondrá el modelo conceptual de la aplicación, además se expondrá una visión general de la librería PhoneCoreLibrary y como se integrará la aplicación con ella. La librería PhoneCoreLibrary es una versión PCL (Portable Class Library) de la librería CoreLibrary que ha sido previamente desarrollada por el autor del proyecto para abstraer de manera general las aplicaciones de gestión sin un gran volumen de datos.

La siguiente etapa que se tratará en el presente documento es la etapa de diseño e implementación, en la que se explicará la arquitectura que se utilizará, la relación entre las clases que componen el sistema, cómo se persistirán los datos y el prototipo de las pantallas de la aplicación.

Estudio de mercado

Funcionalidad del producto, idoneidad y riesgos

Se pretende dar una solución a un problema bastante común entre la gente que hace turismo, y es ofrecer en una misma aplicación, lugares de interés y su ubicación, la previsión meteorológica y servicios que se encuentran en el destino del turista, que en nuestro caso serán las Islas Canarias.

El problema subyacente es que en general la gente pierde mucho tiempo organizando sus viajes, por tanto, se pretende centralizar la información más relevante en un mismo dispositivo. Por tanto, el producto es idóneo por varios motivos:

- Permite acceder a los lugares de interés naturales que ofrece el destino.
- Pueden buscarse servicios cercanos utilizando la API de Google Place, que está en continuo crecimiento.
- Permite acceder a a información meteorológica actualizada utilizando la API de Google.
- La información estará accesible en tres idiomas.

El principal riesgo al que se enfrenta el proyecto es no obtener la certificación para publicarse en AppStore o en MarketPlace.

Estudio de mercado

Ni en el MarketPlace ni en la AppStore podremos encontrar aplicaciones con los objetivos que pretende cubrir nuestra App. De hecho, no hay ninguna aplicación que contemple las Islas Canarias como destino turístico. Sin embargo, sí existen aplicaciones que contemplan cada isla como destino, aunque la calidad de las mismas es bastante baja.

Planificación

Relación de actividades

Elaboración del plan de proyecto. *Duración estimada: 12 horas*
Etapa inicial que comienza con la redacción de este documento. Incluye una reserva temporal, ya que el consultor puede requerir alguna corrección. Esta etapa contempla la elaboración de la **PEC1.**

Preparación del entorno. *Duración estimada: 16 horas.*
En esta etapa se descargará e instalará todo el software necesario para poder llevar el proyecto a cabo.

Documentación sobre WP7 y MonoTouch y Google APIS: Duración estimada 24 horas.
A pesar de que el estudio será continuado y de que se tienen conocimientos previos, se reservará un espacio temporal para ampliar el conocimiento sobre WP7 y MonoTouch.

Especificación formal de requisitos. *Duración estimada 16 horas.*
Se especificará los requisitos de manera más formal, es decir a través de casos de uso.

Análisis y diseño de la aplicación. *Duración estimada: 72 horas.*
En esta etapa se hace el análisis formal de la aplicación y, además, se desarrollará un prototipo de las ventanas de la aplicación. Además, esta etapa contempla la elaboración del documento de la **PEC2**.

Implementación. *Duración estimada 88 horas.*

Etapa en la que se implementará lo especificado en la etapa de Análisis y diseño de la aplicación. Se escribirá todo el código necesario y se unirá la vista con la vista del modelo.

Introducción de datos: *Duración estimada: 40 horas.*

Llegado a este punto se procedería a introducir los datos de la aplicación, así como la traducción de los mismos.

Testing y optimización *Duración estimada: 40 horas.*

Etapa en la que se realizarán pruebas unitarias y se buscarán errores en profundidad.

Memoria y presentación virtual. *Duración estimada 36 horas***.**

La última etapa de carácter académico del proyecto. En ella se elaborará la memoria en base a la documentación generada y los contenidos requeridos por el consultor, como pueden ser los distintos manuales de la aplicación.

Publicación en marketplace y AppStore. *Duración estimada 12 horas.*

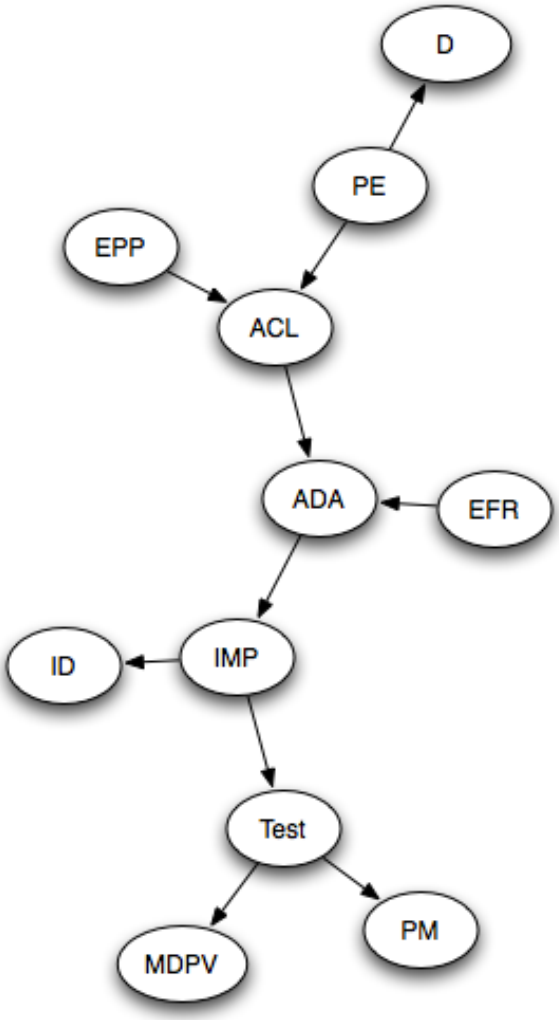
Se enviará la solicitud de publicación a Marketplace y a la AppStore.

Tabla de estimación

Hito	Acrónimo	Dependencia	Tiempo en caso óptimo	Tiempo en caso promedio	Tiempo máximo
Elaboración del plan de proyecto	EPP	-	8	12	16
Preparación del entorno	PE	-	12	16	24
Documentación	D	PE	24	24	24
Especificación formal de requisitos	EFR	EPP	8	16	20
Análisis y diseño de la aplicación	ADA	EFR	64	72	80
Implementación	Imp	ADA	76	88	120
Introducción de datos	ID	Imp	24	40	52
Testing y optimización	Test	Imp	36	40	56

Memoria,documentación y presentación virtual	MDPV	Test	32	36	48
Publicación	PM	Test	8	12	24
Total:			292	356	464
Pronóstico de horas: (en base a la media de las totales)	370.6666667				

Grafo de actividades



Resultados reales

Grosso modo se ha seguido la planificación, sin embargo hay varios puntos que se quieren comentar, ya que evolucionaron durante el proyecto:

- La CoreLibrary se decidió incorporar cuando se estaba realizando el Análisis funcional de la aplicación (PEC 2). Por tanto, no se contempló en el plan de trabajo. Dado que esta etapa está dentro de la de Implementación no varió sustancialmente el resultado final.
- Si bien se hicieron Tests manuales, no se introdujeron proyectos de Test, ni dio tiempo a optimizar el código de la vista de cada plataforma, que como sabemos implica parte del controlador.
- No se tiene en cuenta ni el tiempo de subida a las Stores, ni el tiempo de desarrollo de la memoria y la presentación.
- No se ha podido llevar a cabo la introducción de datos, lo que se ha hecho es crear un Script que los introduzca. Aunque no se comentó nada al respecto, la idea era crear una aplicación específica para introducir los datos. No obstante, dado que el 66% de los datos de la aplicación provienen de servicios de terceros, no es relevante el hecho de que no se pudieran introducir.

Análisis de necesidades y requisitos

Fuente de información

Se han encontrado multitud de aplicaciones que cumplen la función de guía turística, sobre todo aplicaciones Web, aunque en ninguna de ellas se ha encontrado la terna que hace interesante este proyecto: sitios de interés, información meteorológica y servicios. Además, cabe destacar que en la AppStore/Marketplace hay muy pocas aplicaciones sobre las Islas Canarias, y las que hay están muy limitadas.

Análisis de necesidades

La necesidad que pretende cubrir el proyecto es, cuanto menos, ambiciosa. Se pretende que un usuario pueda obtener información de los lugares más relevantes del destino en cuestión, y que además, pueda (mediante conexión a internet) conocer la información meteorológica y los servicios que hay cerca. Además, la aplicación estará disponible en al menos, tres idiomas: inglés, castellano y alemán.

Se procede a enumerar las principales necesidades que cubrirá la aplicación:

- Obtener una lista de sitios de interés en un destino en particular que pueda buscar según por una serie de filtros (playas, montaña, etc).
- Obtener información meteorológica por zona geográfica.
- Obtener información sobre los servicios que hay cerca (restauración, tiendas, etc.)

Especificación de requisitos

A continuación se exponen los requisitos no funcionales y funcionales de la aplicación.

Requisitos no funcionales

NF1: Se hará una interfaz usable en la que se llegue al objetivo, en la mayoría de los casos, en tres clicks.

NF2: La aplicación debe ser intuitiva. Esto es que cualquier persona habituada a manejar el sistema operativo WP7/iOS sea capaz de utilizar la aplicación sin mayor complicación.

NF3: La aplicación debe ser rápida. Esto es que entre dos pantallas no puede haber retraso.

NF4: La aplicación debe ser sencilla y cumplir sólo con un pequeño número de funcionalidades.

NF5: La aplicación debe cumplir los principios de diseño de Microsoft, para ser publicada en el MarketPlace, y de Apple, para ser publicada en la AppStore.

Requisitos funcionales

Visualización de lugares de interés.

Req. 1.1: Se mostrarán todos los lugares de interés disponibles para un espacio geográfico en concreto. Se hará un volcado de la aplicación de todas los lugares ordenados alfabéticamente.

Req. 1.2: Se podrá filtrar por tipo de lugar de interés. A medida que se visualizan los lugares de interés se puede seguir filtrando por cada tipo de lugar de interés, y sobre el resultado se podrá volver a filtrar, apareciendo sólo los lugares de interés que contengan dicho filtro.

Ficha de Lugar de interés

Req. 1.3: Desde la ficha de lugar de interés se podrá visualizar toda la información en relación al lugar de interés.

Información meteorológica

Req. 3.1: Desde la ficha de información meteorológica se mostrará información relativa al tiempo que hará en los lugares seleccionados.

Servicio

Req. 4.1: Se visualizarán todos los servicios que hay cerca en un mapa y se podrá filtrar por tipo de servicio.

Casos de uso

Se utilizarán los casos de uso para modelar los requisitos recogidos en el apartado anterior. Dada la limitación de funcionalidades que se aplica por tratarse de una aplicación móvil, el diagrama de caso de uso resultante es bastante simple.

Antes de visualizar el diagrama, se procederá a identificar a los dos actores principales del sistema:

Usuario: Se trata del usuario de la aplicación, es decir la persona que utilizará la aplicación.

Google: Se trata del servicio que se utilizará para hacer peticiones de ciertos datos.
El diagrama engloba todas las historias relativas al uso de la aplicación. Algunos ejemplos son:

El usuario desea buscar un lugar de interés.
El usuario desea obtener información relativa a un lugar de interés.

El usuario desea conocer la información meteorológica.

El usuario desea buscar servicios cerca.

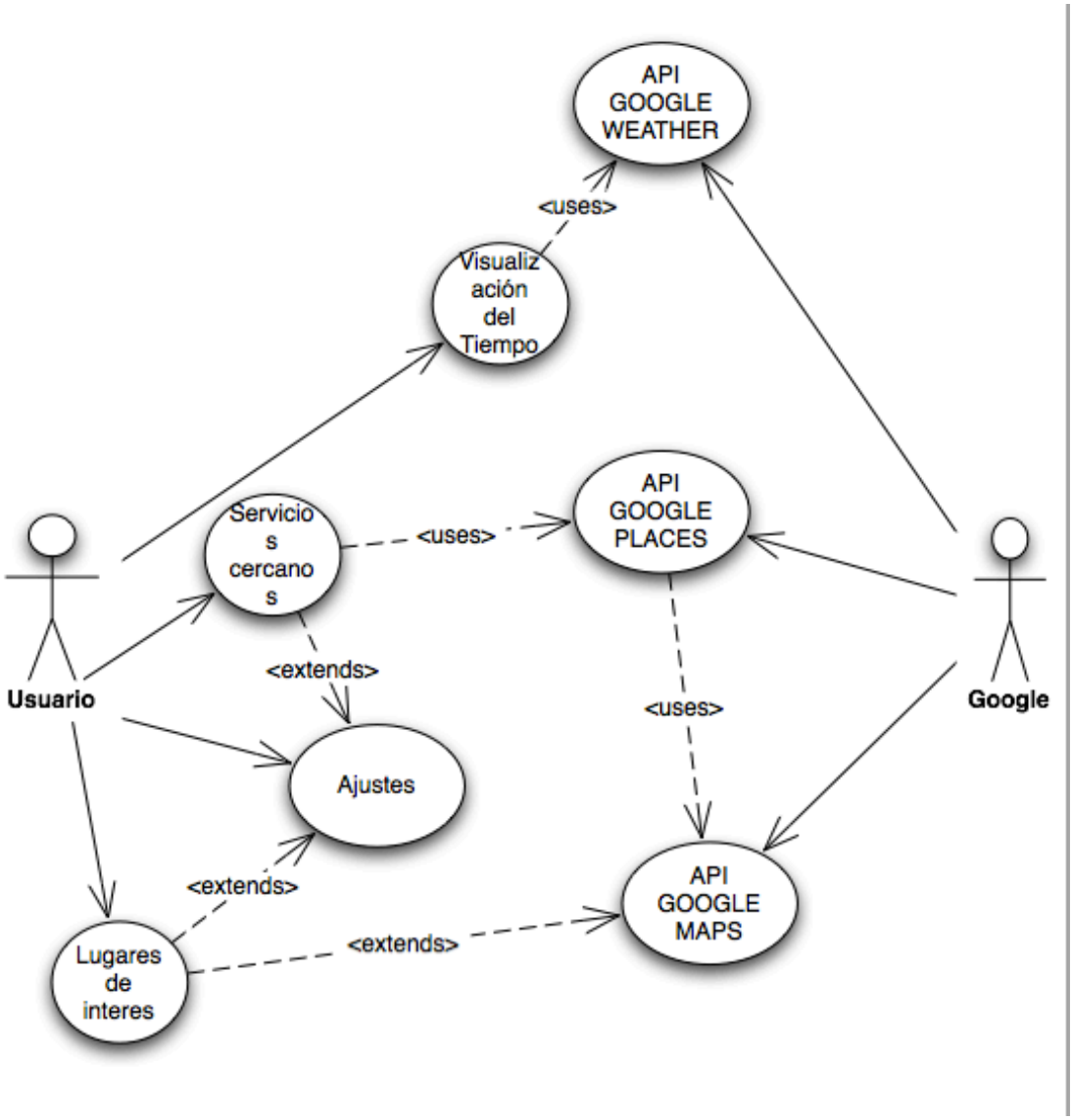


Figura 1. Diagrama de caso de uso de la aplicación CanaryGuide.

Visualizar lugares de interés

Descripción

Se muestra al usuario una lista ordenada alfabéticamente con todos los lugares de interés disponibles en la aplicación. El usuario puede filtrar por categorías los lugares de interés.

Actores

Usuario.

Casos de uso relacionados

Ninguno

Precondición

Deben existir lugares de interés previamente.

Postcondición

El usuario ha encontrado el lugar de interés.

Proceso normal principal

1.El usuario accede al listado de lugares de interés.

2.El usuario busca a través de filtros o alfabéticamente las características del lugar de interés que desea encontrar.

3.El sistema a través de los requisitos del usuario filtra el lugar de interés.

4.El sistema muestra al usuario el lugar de interés deseado.

Alternativa al proceso normal principal

4.a El sistema no encuentra en el registro el lugar de interés deseado por el usuario.

Ficha lugar de interés

Descripción

Muestra los detalles de un lugar de interés.

Actores

Usuario

Casos de uso relacionados

Visualizar lista de lugares de interés

Precondición

El usuario debe haber seleccionado un lugar de interés.

Postcondición

El usuario ha visualizado los datos del lugar de interés.

Proceso normal principal

1. El usuario selecciona un lugar de interés y entra en el proceso.
2. El sistema muestra al usuario los datos del lugar de interés.
3. El usuario visualiza los datos.

Información meteorológica

Descripción

Muestra la información meteorológica para un lugar seleccionado.

Actores

Usuario, Google

Casos de uso relacionados

Ninguno

Precondición

El usuario debe tener conexión a internet.

Postcondición

El usuario ha obtenido la información meteorológica.

Proceso normal principal

1. El usuario selecciona el lugar para el que quiere obtener la previsión y el momento.
2. El sistema realiza una petición a Google en tiempo en real y muestra los datos al usuario.
3. El usuario visualiza los datos.

Alternativa al proceso normal principal

1.a El sistema no se puede conectar a Google (probablemente por no tener conexión) y muestra un mensaje de error al usuario.

Servicios cercanos

Descripción

Muestra los servicios cercanos.

Actores

Usuario, Google

Casos de uso relacionados

Ninguno

Precondición

El usuario debe tener conexión a internet.

Postcondición

El usuario ha obtenido información relativa a los servicios que tiene cerca.

Proceso normal principal

1. El sistema realiza una petición a Google en tiempo real y muestra los datos al usuario
2. El sistema muestra un mapa con los lugares cercanos al usuario.
3. El usuario puede seleccionar los filtros deseados o interactuar con el mapa.

Alternativa al proceso normal principal

1.a El sistema no se puede conectar a Google (probablemente por no tener conexión) y muestra un mensaje de error al usuario.

Modelo conceptual

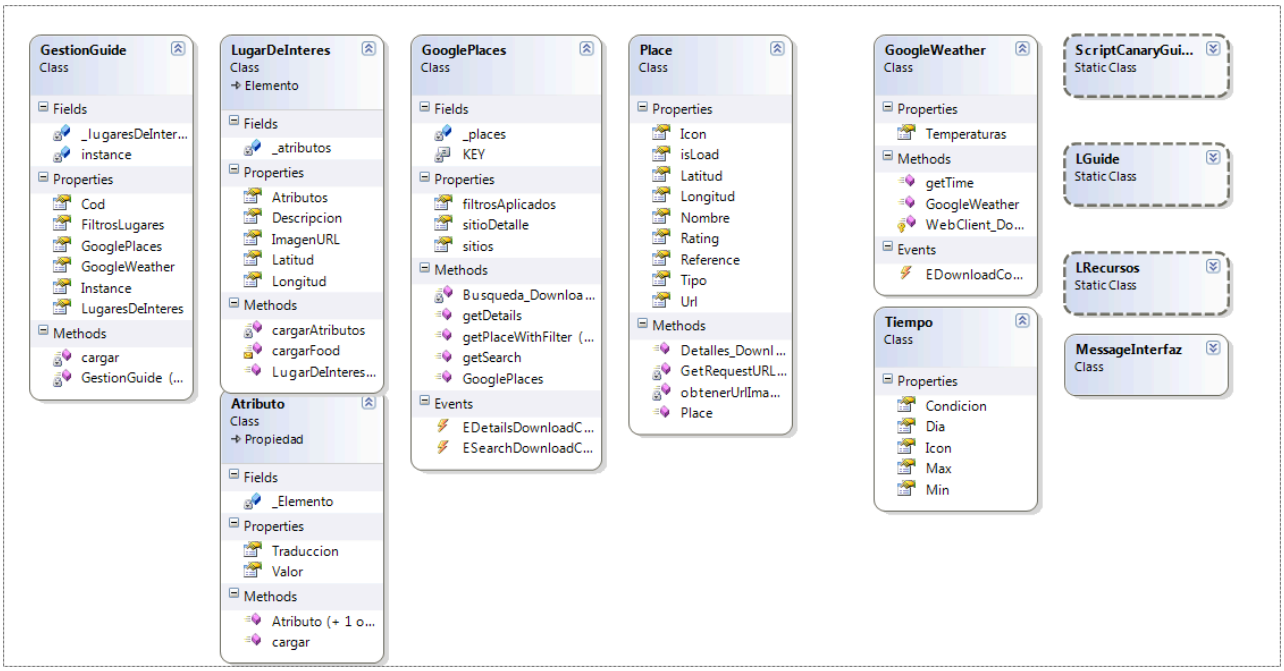


Figura 2. Diagrama con las entidades básicas de la librería CanaryGuide.

Entidades básicas

Las unidades básicas del sistema son las que se muestran en la ilustración anterior, cabe destacar que no son todas las que componen la lógica de la aplicación, pero sí las que forman el núcleo de la lógica. Se explicarán las más relevantes:

Atributo: Cada lugar de interés puede ser filtrado en función del tipo que se busque. Ese tipo puede ser, por ejemplo: playa, montaña, norte, etc. Bien, cada tipo citado anteriormente, seria una instancia de la clase Atributo. Cabe destacar que un lugar de interés puede tener más de un atributo y, para cada atributo puede haber varios lugares de interés.

Lugar de interés: Una instancia de lugar de interés tiene descripción, longitud, latitud, dirección de una imagen asociada, atributos, etc. Representa un lugar de interés tal y como se percibe en la realidad.

GestionGuide: Se trata de una caché que contiene en memoria todos los lugares de interés del sistema, así como los métodos necesarios para filtrarlo. Esta clase implementa el patrón Singleton, con lo que sólo existe una instancia de la misma para toda la aplicación.

GooglePlaces/GoogleWeather: Clases que permiten interactuar con las respectivas APIs de Google. También implementan el patrón Singleton, aunque implícitamente. Se instancian en el constructor de GestionGuide.

Visión general de PhoneCoreLibrary

La aplicación será desarrollada bajo una librería, creada por la misma persona que desarrollará la aplicación “CanaryGuide” así el presente documento. La PhoneCoreLibrary (de ahora en adelante, CoreLibrary o CL) ha sido desarrollada con el objetivo de agilizar las aplicaciones de gestión, proporcionando para ello un conjunto de clases básicas de las cuales deben heredar las aplicaciones y de tipos que utilizarán dichas clases. Esto

proporciona al programador una nueva capa de abstracción a la hora de desarrollar aplicaciones de gestión. De esta manera, el programador desarrolla como si la memoria principal o RAM fuera persistente, la librería se encarga de guardar los datos en memoria cuando sea necesario. Además permite guardar en distintas fuentes de datos implementando una interfaz. Para adaptarla al sistema operativo Windows Phone 7.1 tuvieron que reescribirse algunas partes de la librería, ya que por ejemplo, la plataforma móvil no soporta la serialización a nivel de bytes ni código no administrado. En cambio, para poder utilizarse en MonoTouch en el que no existe, por ejemplo, el concepto de IsolatedStorage, se ha tenido que utilizar la compilación condicionada, con sentencias de preprocesador. No obstante, estas pequeñas diferencias no representan un problema real y el tiempo de producción sigue siendo muchísimo más reducido.

Modelo de la librería

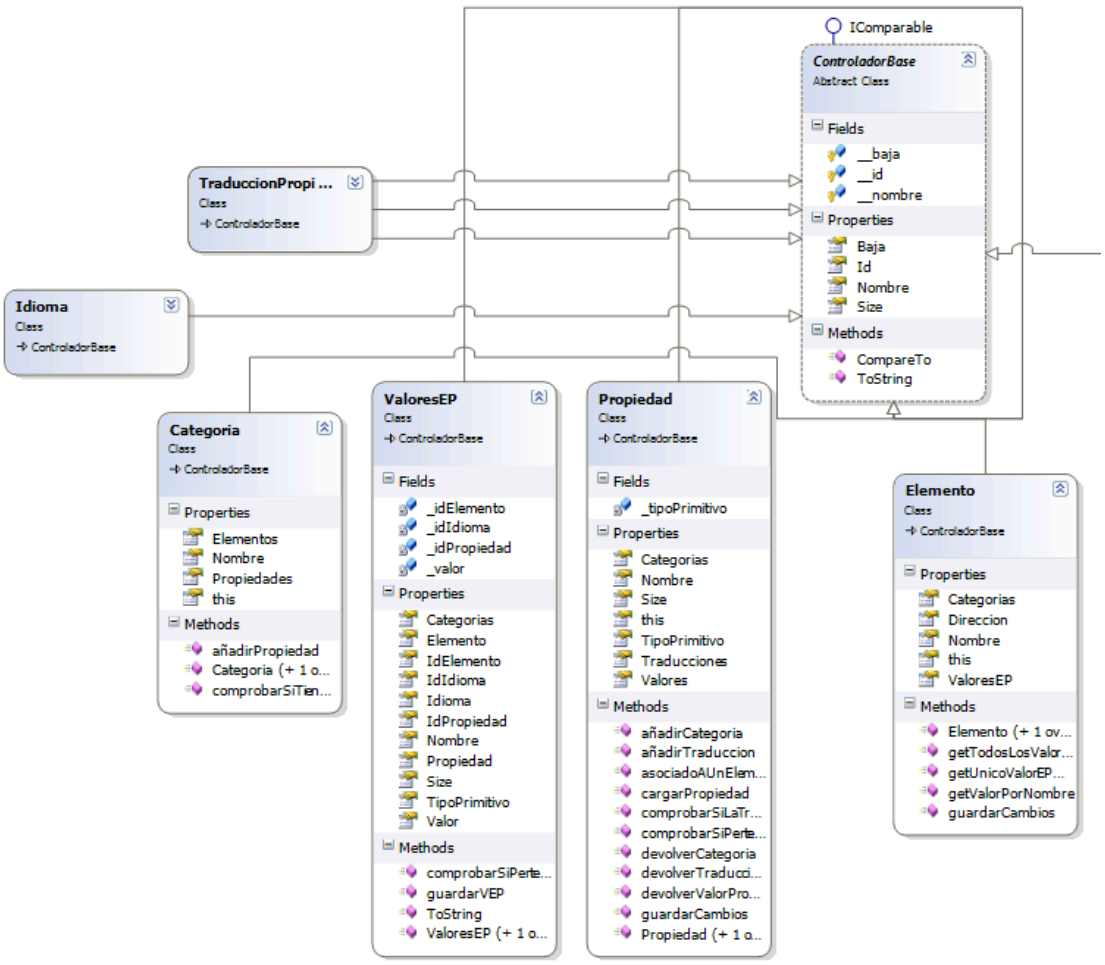


Figura 3. Entidades CoreLibrary

El diagrama de las entidades básicas de la CoreLibrary es el que se muestra arriba, a continuación se pasa a describir brevemente cada una de las entidades:

- **ControladorBase**: Clase abstracta de la que heredan todos los elementos de la librería. Representa la estructura básica de cada elemento de la librería e implementa la interfaz IComparer, lo que permite mantener la lista de elementos ordenada alfabéticamente. Además, tiene un atributo interno denominado State que devuelve el estado en el que se encuentra la instancia de la clase en cuestión. Puede ser: unchanged, modify, new, delete.

- **Elemento**: Un elemento representa una o varias entidades básicas de una aplicación, es decir, cada entidad básica de la aplicación debe heredar de esta clase. Esta clase tiene una lista de valores, categorías, así como un nombre único, además de un ID.
- **Propiedad**: Representa un atributo que puede ser categorizado, en el caso de nuestra aplicación, puede ser por ejemplo el atributo “Playa” que indica que un lugar de interés es *playa*. Además, cada propiedad puede pertenecer a varias categorías y a varios elementos, eliminando así la duplicidad de propiedades.
- **Categorías**: Representa una categoría de la aplicación. Para comprender la necesidad de esta clase, se ha de comprender la necesidad de que una propiedad pueda pertenecer a varias categorías. Esto es, si por ejemplo se tienen dos tipos de elementos distintos, para poder diferenciar internamente entre ambos, ya que ambas subclases se persistirían como Elemento. Aunque en nuestra aplicación solo habrá un subtipo de elemento, lugar de interés.
- **ValoresEP**: Para que distintos elementos -de la misma categoría- tengan distintas propiedades, se recurre a esta clase. Así esta clase representa el valor que puede tener una propiedad para un elemento en concreto, es decir, cada instancia tiene una referencia a una propiedad y a un elemento. En el siguiente apartado, se verá como es posible que una propiedad tenga distintos tipos y distintos valores gracias a esta clase. Cabe destacar, que las categorías a las que pertenece una instancia de ValoresEP son las mismas a la que pertenece la propiedad asociada, así las categorías a las que pertenece un Elemento son las categorías a las que pertenece cada ValorEP. Todo esto se resuelve en tiempo de ejecución.
- **Idioma y traducción propiedad**: Estas clases implementan toda la lógica necesaria para que la aplicación esté en múltiples idiomas. Dado el tiempo que implicaría la traducción manual de los datos que se van a incorporar en el sistema, se dejará para el final.

Jerarquía de tipos básicos de la CoreLibrary

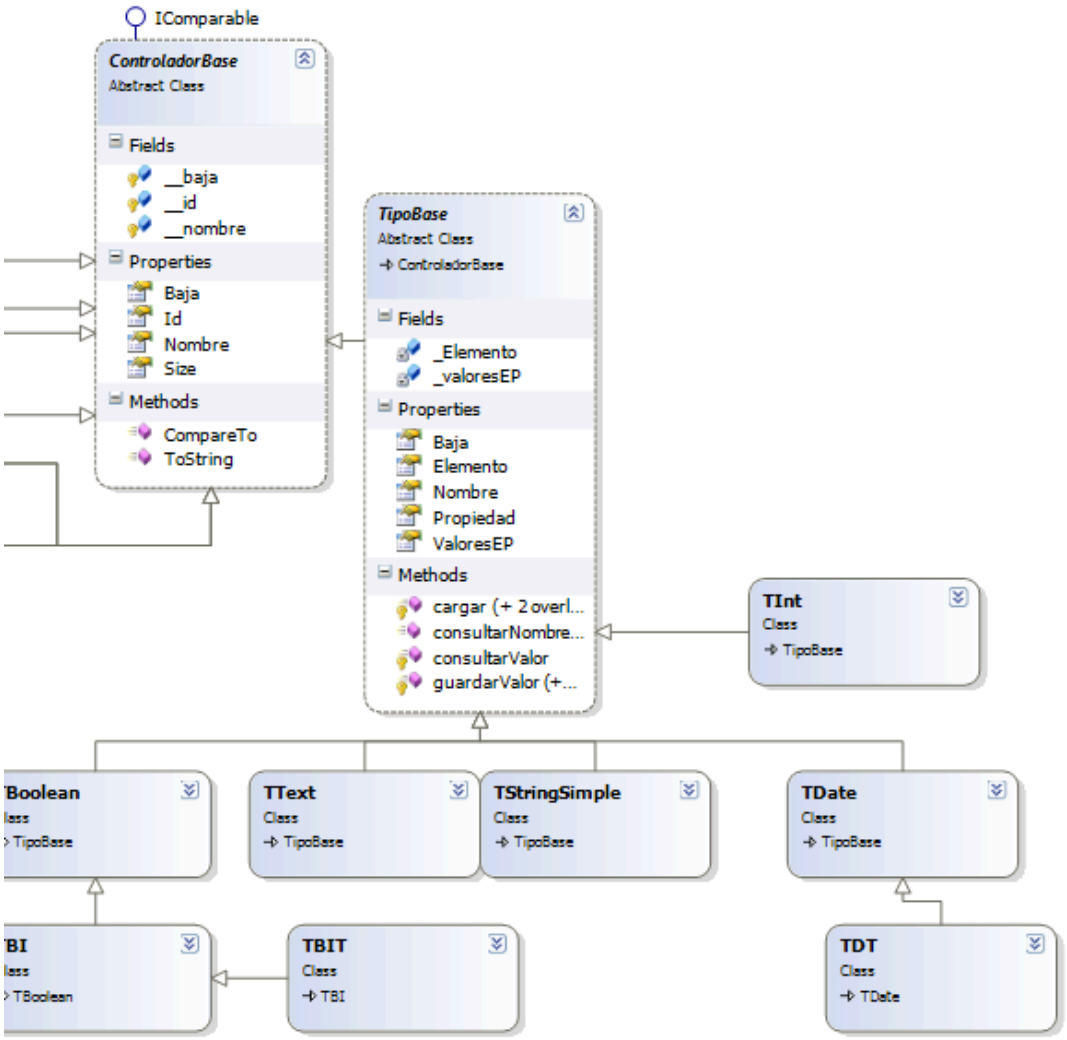


Figura 4. Jerarquía de tipos básicos CoreLibrary

La librería tiene una serie de tipos básicos que permiten principalmente al programador abstraerse totalmente de la capa de acceso a datos, por ejemplo si se quiere utilizar un tipo, al ejecutar la instrucción *Elemento.NombreDeLaPropiedad = new TBoolean()*; automáticamente se está persistiendo la relación entre la propiedad y el elemento a través del valor. En el ejemplo se utiliza el tipo TBoolean que realmente es un tipo booleano, cada propiedad sólo puede ser de un tipo.

En la jerarquía de tipos, se observa que la clase abstracta TipoBase hereda de ControladorBase, y que TipoBase también es abstracta. Se observa que cada instancia heredada de TipoBase tiene una referencia al Elemento y a la propiedad asociada. A continuación veremos los tipos más importantes:

- **TBoolean**: Representa un tipo booleano.
- **Tint**: Representa un entero.
- **TStringSimple**: Representa una cadena. Esta clase también se utiliza para guardar los tipos serializados, así se podría guardar una clase más compleja dentro de este tipo.
- **TText**: Representa una cadena para cada idioma, de esta manera el tipo posee un indexador tipo ["es"] que pasándole el código del idioma, nos devuelve la traducción para ese idioma. Internamente esta propiedad instancia para cada idioma un ValorEp.

Acceso a datos

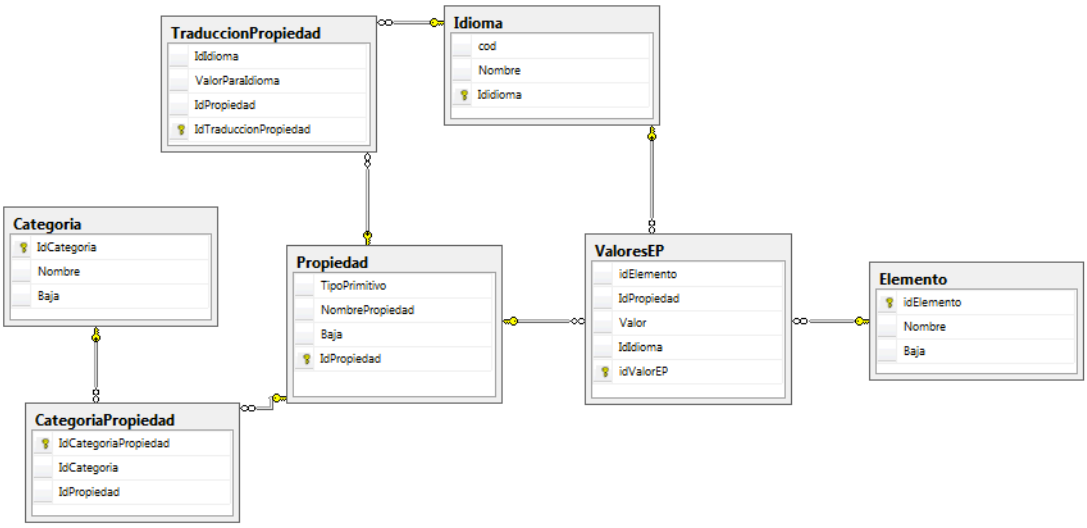


Figura 5. Diagrama de datos CoreLibrary

El diagrama que ilustra la figura 5 muestra los datos que se persisten de la CoreLibrary. A pesar de que el diagrama se muestra como un diagrama de base de datos, es meramente ilustrativo, ya que como se ha comentado con anterioridad, implementando una interfaz, la librería es capaz de comunicarse con distintas fuentes de datos, es decir, el almacén de datos es totalmente transparente a la librería.

Gestión de datos

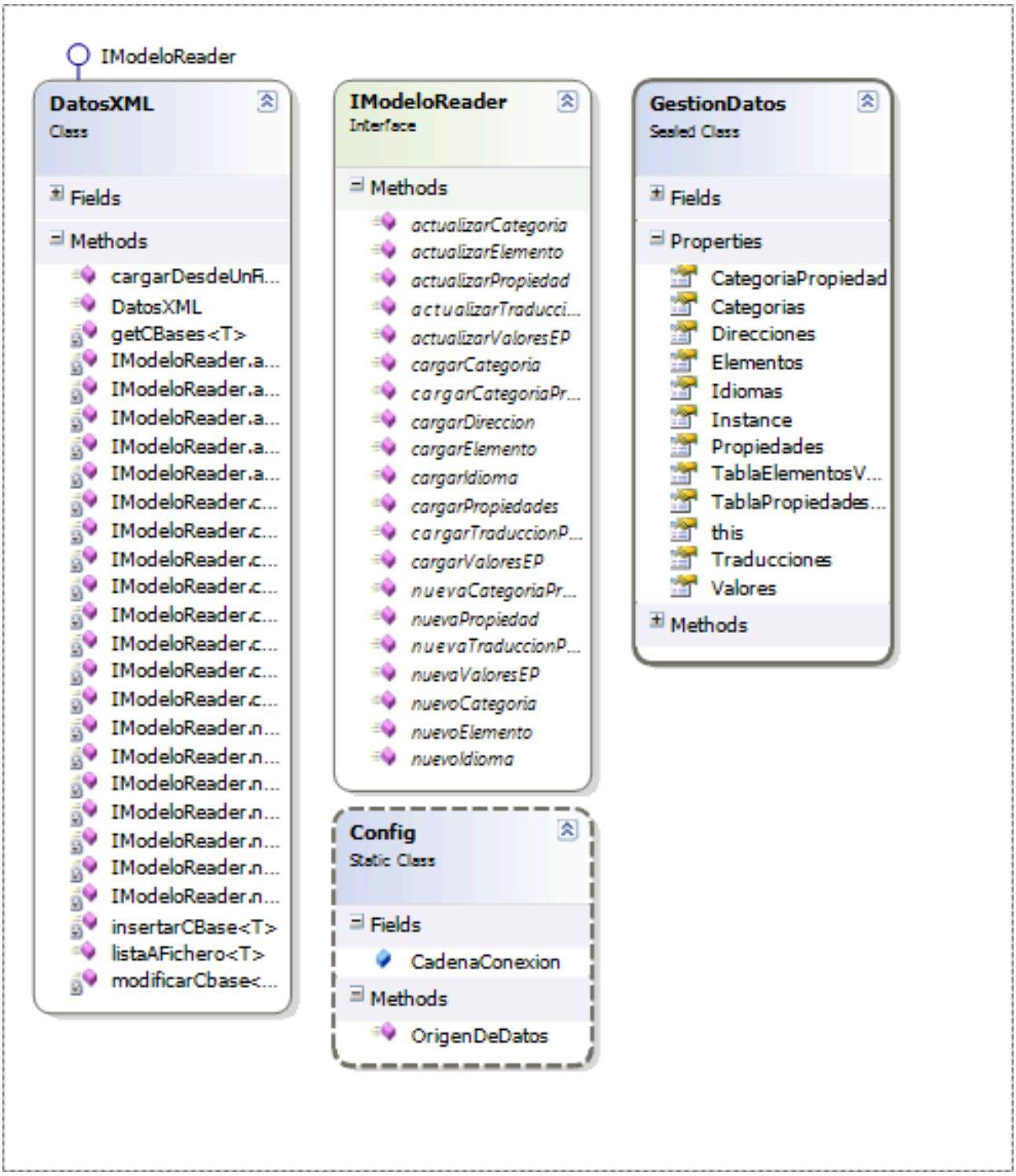


Figura 6 Gestión de datos

La gestión de datos se realiza a través de las clases que se muestran en la figura 6. A continuación se explicará brevemente cada clase:

- **Config**: a través del método OrigenDeDatos devuelve una instancia de la interfaz IModeloReader seleccionando los datos que se utilizarán según el tipo de aplicación. En nuestro caso la implementación de los datos la realizará DatosXML.
- **DatosXML**: serializa a un fichero XML, los datos de la figura 5, cada entidad la guarda en un fichero, por lo tanto habrá un fichero para Elemento, Propiedad, Categoría, etc.
- **IModeloReader**: interfaz que permite abstraerse de la capa de datos de la aplicación.
- **GestionDeDatos**: clase que implementa el patrón Singleton (solo existe una instancia en toda la aplicación) y contiene en cada momento la última versión de los datos. Cuando se levanta la aplicación lee los datos a través de la interfaz IModeloReader, y cada vez que se inserta, modifica o borra un registro, interactúa con la interfaz para persistir los cambios en la base de datos, aunque no hace más lecturas de la misma, para evitar exceso de conexiones. Esta clase se puede considerar una caché de la aplicación; permisible dado que en este tipo de aplicaciones nunca se llegará a millones de registros.

Uniendo las piezas: CanaryGuide y CoreLibrary

Como se ha explicado anteriormente, Lugar de interés hereda de la clase Elemento y Atributos heredará de Propiedades. Esto permite que Lugar de interés tenga Atributos de la misma manera que Elemento tiene Propiedades, lo que permite una enorme flexibilidad al crear nuevos tipos de interés, y generar contenido en general. Por otro lado, Categorías permanece intacta, y cada instancia contendrá una colección de filtros.

Finalmente, GestionGuide de manera análoga a GestionDeDatos implementará el patrón Singleton y permitirá la abstracción total de la CoreLibrary, gestionando todos los datos que tengan relación directa con la lógica de la aplicación.

Implementación

Se tratará de manera superficial cómo ha sido la implementación, qué tecnologías se han utilizado y qué ha faltado por implementar.

Tecnologías empleadas:

Principalmente se ha utilizado .NET para llevar a cabo el desarrollo de la aplicación. Se debe mencionar que a pesar de que se ha utilizado Mono ha sido sólo en la versión para iOS ya que no hay componentes específicos de Mono para la versión de Windows Phone.

La base de la aplicación se fundamenta en la CoreLibrary, que se ha tenido que adaptar a una librería de clases portables. Esto significa que se puede cambiar el Target en Visual Studio y es compatible con aplicaciones Silverlight 4 y 5, WP7,WP8 y Windows 8 y por supuesto, el resto de componentes de .NET (menos las versiones embebidas del Framework) dado que se trata de un subconjunto de librerías del Framework .NET las que se utilizan en las librerías portables. Lo interesante de esto es que el proyecto Mono también admite soporte para todos los tipos de aplicaciones que se pueden generar con él. Desde GTK# hasta AXT pasando por MonoDroid y por supuesto, MonoTouch. Esto significa que la dll generada por VS2010 sólo se ha tenido que importar en el proyecto para MonoTouch. Cabe mencionar que se han tenido que utilizar sentencias de preprocesado para determinadas partes.

La segunda librería que se utiliza es la GuideLibrary que ha sido creada específicamente para el dominio de este proyecto. La GuideLibrary, como se ha comentado anteriormente, es una abstracción de la CoreLibrary. No sólo contiene el modelo real de la aplicación sino que además se encarga de obtener los datos de los distintos servicios utilizados. Se debe mencionar que en este caso no se ha podido desarrollar una librería de clases portables ya que se acceden a partes del framework que no están soportadas por este tipo de proyectos. No obstante, se ha utilizado un control de versiones (SVN), y el código fuente de ambas

aplicaciones es el mismo para esta parte de la librería. Eso sí, para esta librería se han tenido que utilizar bastantes sentencias de preprocesador, sobre todo para el tratamiento de imágenes.

La tercera y última capa de la aplicación se centra en la integración de las librerías desarrolladas con las distintas plataformas de tal forma que se adapta a las peculiaridades de la plataforma.

WP7:
Con respecto a WP7 el desarrollo ha sido muy fluido ya que se tenía conocimiento previo de la plataforma. Los problemas más destacados vienen de parte de la integración con los datos de terceros. En general, las prácticas utilizadas son CodeBehind aunque se ha tratado de utilizar el patrón MVVM en la medida de lo posible. No se ha podido utilizar más dicho patrón por el hecho de que los datos estaban en una capa inferior y se requería más trabajo adicional, cosa que va en contra de la filosofía del modelo ya que persigue la productividad.

iOS:

- En el caso de iOS, ya que se realiza a través de MonoTouch, en primera instancia se expondrá de manera general en qué consiste MonoTouch:
  MonoTouch convierte el código .NET en código nativo para los procesadores ARM de iOS. Esto significa que el resultado final de la aplicación es totalmente nativo, con la única diferencia que el paquete ocupa algo más de tamaño. Además, se puede utilizar el IDE de Apple XCode para diseñar las pantallas aunque no es obligatorio, ya que se pueden crear generándolas dinámicamente. La mayor parte de las ventanas de esta aplicación se han creado dinámicamente utilizando, sobre todo, MonoTouch.Dialog que no es más que una librería que permite construir UITableViewControllers de manera ágil.
  Otras características de MonoTouch son:

- C++/Objective-C Interop.
- Enlace a la API de Objective-C en iOS.
- .NET Bass Class Library (BCL)
- Soporte para multiplataforma móvil (en conjunto con MonoDroid).

Sin embargo, no todo es positivo ya que existe una limitación muy importante y es que la compilación JIT no está permitida. Lo que implica que no se puede utilizar, por ejemplo, el espacio de nombres Reflection.Emit o lenguajes dinámicos (F#,IronRuby, IronPython, etc).

Una vez enumeradas los beneficios de MonoTouch, se puede concluir que a pesar de que ha habido problemas para adaptarse al patrón de trabajo de Cocoa, una vez superadas estas barreras el desarrollo ha sido igual o más fluido que con WP7.

Carencias del proyecto:

La más destacable y prácticamente la única parte que ha faltado por implementar en este proyecto ha sido la introducción de datos con todo lo que esto implica. Y es en primera instancia, un mayor número de “Lugares de Interés” y búsquedas más coherentes en GooglePlaces. En segundo lugar, y más notable aún si cabe, es que no se han introducido las traducciones en el resto de idiomas ni para el contenido dinámico (descripciones de los lugares de interés y filtros) ni para el contenido estático. Si se observa el proyecto y la arquitectura se verá que “sólo” queda hacer las traducciones ya que se han incluido los archivos de recursos, se ha contemplado a nivel arquitectónico (véase la CL) y se consigue identificar el idioma que tiene configurado el usuario establecido en el dispositivo.

Por otra parte, también ha habido un problema con la API de Places ya que por alguna razón sólo genera la respuesta a la petición en inglés, puede ser porque normalmente se utiliza JSON y he utilizado XML para rescatar los datos. Además, el servicio que ofrece la API es bastante pobre, está pensado para mostrar los datos en un mapa, y para obtener las imágenes reales (no los iconos) de los establecimientos. He tenido que generar un total de 3 peticiones adicionales, que pueden parecer pocas, pero si se tiene en cuenta que son tres peticiones por lugar, se pasa de tener una sola petición global a tres por sitio. Esto significa que si tenemos 20 sitios en nuestra petición actual se generan hasta 60 hilos (ya que las peticiones son asíncronas para no “congelar” la GUI). La primera petición es la general, que devuelve los sitios con coincidencias para los tipos y espacio geográfico deseado. A continuación, para cada sitio se hace una petición de los detalles (también a Google Place). Dado que GooglePlace no tiene más imágenes por defecto, se realiza otra petición a la URL del sitio para obtener la imagen. El servicio que utiliza Google para mostrar las imágenes del sitio es Panoramio, el problema de la API de Panoramio es que sólo se pueden hacer peticiones a través de unas coordenadas dadas, lo que puede llevar a que se devuelvan imágenes que no pertenecen al sitio. Por ello, se optó por crear un Script que para cada sitio, se descargue la página web (versión móvil) del sitio, busque la imagen asociada, y si la encuentra, la muestra. El problema encontrado en este punto, fue que no todos los sitios tenían una imagen asociada, de hecho hay muy pocos con imagen asociada. Por ello se decidió prescindir de la imagen asociada ya que no sólo ocupaba bastantes recursos

(recuerde el caso de los 60 hilos sólo para descargarse la imagen) sino que además, a nivel de diseño no era “presentable”.

Prototipo de la interfaz

A continuación se presentarán las pantallas de la aplicación. Se ha optado por mostrar para cada pantalla la versión para iOS y WP7, de izquierda a derecha respectivamente. Además se mostrará el Mockups del que se partió ya que así se entenderá mejor cómo se ha adaptado la app a los distintos entornos, con la mejora de usabilidad que implica, ya que el usuario debería estar acostumbrado a este tipo de controles.

Cabe destacar que se podría haber pulido el diseño un poco más, aunque sí es cierto que cumple los criterios de usabilidad y aceptación por ambas Stores, aunque no se piense publicar.

Pantalla de Inicio

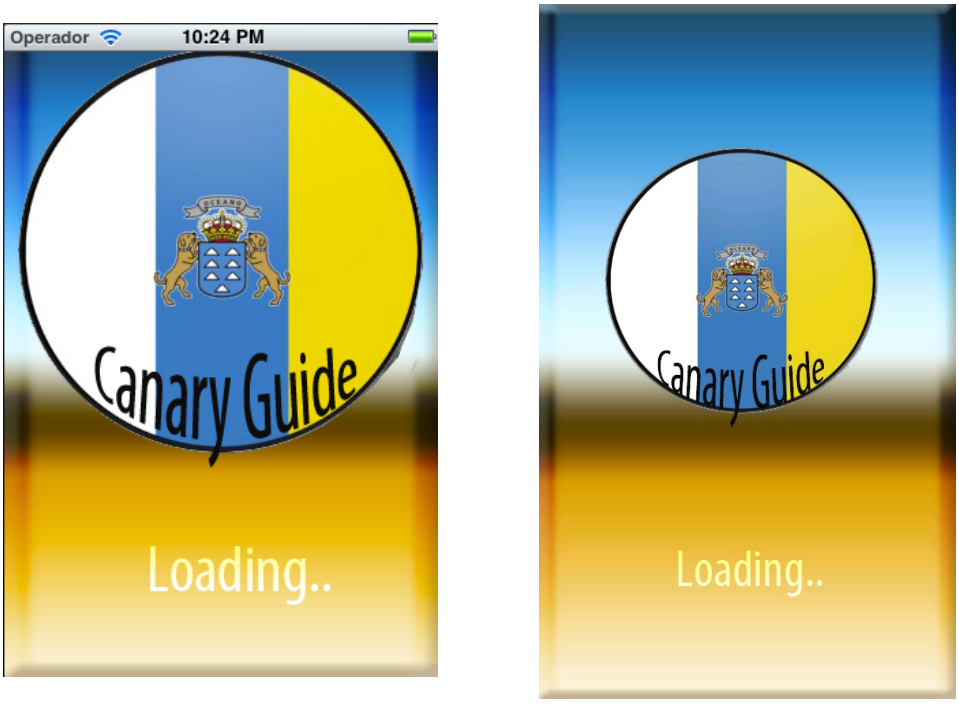


Figura 7 Pantalla de Inicio

Es la pantalla de carga de la aplicación, apenas se puede apreciar ya que la aplicación se ejecuta rápidamente.

Pantalla principal

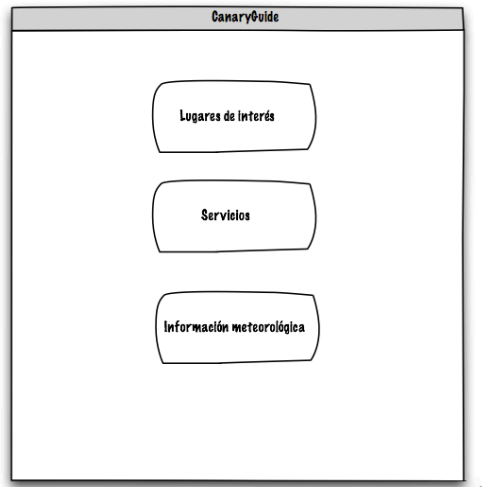


Figura 8 Menú de la maqueta.

Esta pantalla realmente no es una pantalla, aunque en la maqueta se expusiera como tal. Está implícita a través de los controles de las plataformas, y es implementada por las pantallas a las que conduce. En iOS se implementa a través de un UITabBar y en Windows Phone a través de un PanoramaControll.

Consta de tres elementos principales, cada uno da acceso a una de las tres partes principales de la aplicación.

Lugares de interés

Figura 9. Parrilla lugar de interés

Desde esta pantalla se pueden observar todos los lugares de interés de la aplicación. Se podrá filtrar una y otra vez entre los filtros y se actualizará automáticamente la parrilla de resultados. Los filtros siempre muestran el número de elementos que lo contienen, y si está marcado mostrará una X que al presionar sobre ella, se demarcará dicho filtro.

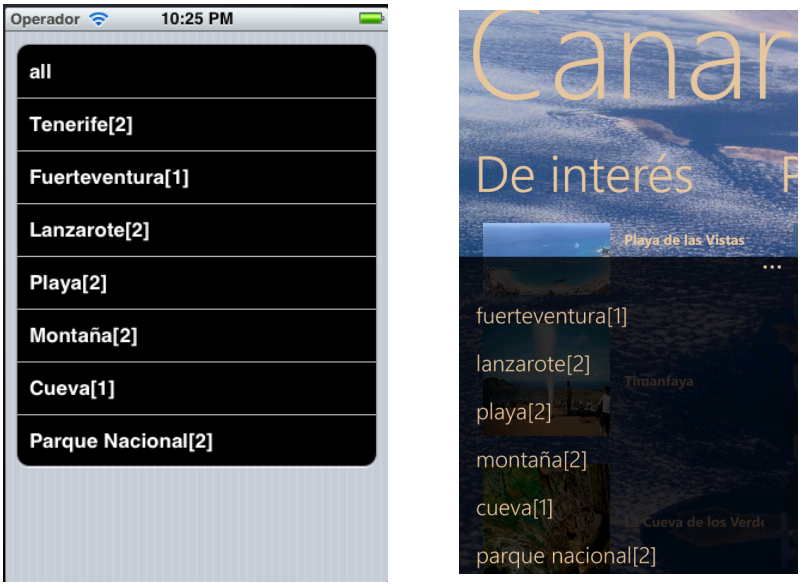


Figura 10. Filtros de Lugar de Interés.

Para Windows Phone se utilizará el AplicationBar para mostrar los filtros.
En iOS se optará por utilizar un botón en la tabla superior que al pulsarlo genera dinámicamente una pantalla con una tabla y los filtros correspondientes.

Pantalla Ficha Lugar de interés

Figura 11. Ficha lugar interés

Desde esta pantalla se podrá visualizar los detalles de cada lugar de interés. Entre los datos que se mostrarán se incluyen: Imágenes, Mapa, Descripción y atributos.
En Windows Phone se utilizará el control de mapas de Microsoft para mostrar la ubicación, que redenderiza un mapa de Bing. Aunque se puede cambiar se ha decidido no hacerlo para mostrar al usuario de Windows Phone el tipo de mapa al que está acostumbrado.
En iOS seguramente se opte por un control que muestre los mapas de Google.

Places

Figura 12. Parrilla Places

Para esta pantalla finalmente se ha optado por seguir el modelo establecido por los lugares de interés, por dos razones fundamentales que tienen que ver con la usabilidad de la aplicación.
En primer lugar, mostrar en un control mapa muchos datos se a la vez se hace engorroso de cara al usuario. La mayoría de desarrollos de aplicaciones de mapas, que muestran muchos datos, son así por el mero hecho de que los propietarios de los servicios lo establecen en la licencia de uso (por ejemplo Google).

En segundo lugar, el esfuerzo para el usuario es mínimo a la hora de poder usarlo. Esto es, que con comprender la forma en la que se interactúa con la primera parte de la aplicación, los lugares de interés, podrá utilizar esta parte sin esfuerzo adicional.

Desde esta pantalla se pueden observar todos los Sitios. Se podrá filtrar una y otra vez entre los filtros y se actualizará automáticamente la parrilla de resultados. Los filtros siempre muestran el número de elementos que lo contienen y, si está marcado mostrará una X que al presionar sobre ella, se demarcará dicho filtro.

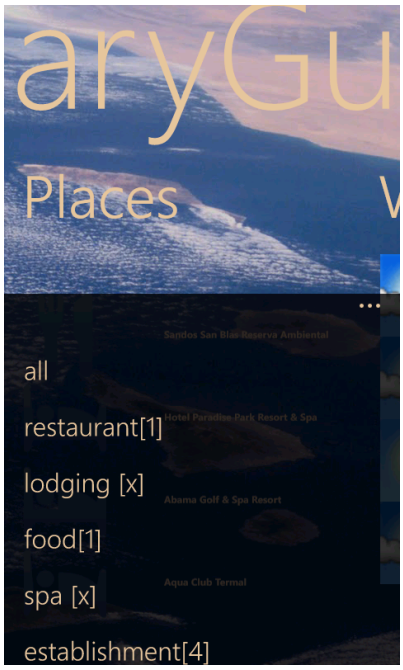


Figura 13. Filtros de places

Para Windows Phone se utilizará el AplicationBar para mostrar los filtros.
En iOS se optará por utilizar un botón en la tabla superior que al pulsarlo genera dinámicamente una pantalla con una tabla y los filtros correspondientes.

Ficha Places

Figura 14. Ficha Places

Desde esta pantalla se podrá visualizar los detalles de cada sitio. Entre los datos que se mostrarán se incluyen: Imagen, Mapa y atributos.

En Windows Phone se utilizará el control de mapas de Microsoft para mostrar la ubicación, que renderiza un mapa de Bing, aunque se puede cambiar se ha decidido no hacerlo para mostrar al usuario de Windows Phone el tipo de mapa al que está acostumbrado.

En iOS seguramente se opte por un control que muestre los mapas de Google.

Información meteorológica

Figura 15. Información meteorológica

Finalmente, en la pantalla de información meteorológica se mostrará la información relativa al tiempo que hará en los alrededores de una localización determinada. Cabe destacar que se mostrará una imagen distinta en función del tiempo que haga. También se podrá seleccionar la información para un día en particular.

Conclusiones

Se puede concluir que el proyecto se ha llevado a cabo satisfactoriamente por dos razones fundamentales: En primer lugar ha servido para conocer los pormenores del desarrollo para iOS bajo MonoTouch y de la migración de librerías para la construcción de aplicaciones multiplataforma, y, en segundo lugar, ha servido como base para las apps móviles del proyecto real que se va a llevar a cabo.

Líneas futuras

Sin lugar a dudas, el proyecto será continuado como una aplicación de mucha más envergadura. Se trata de una Guía de Canarias que pretende ofrecer al usuario sitios que visitar (restaurantes, tiendas, etc), lugares de interés, oferta cinematográfica, vivienda de corta temporada, compra de billetes de avión, reservas de hoteles, alquiler de coches, etc.

Por la parte de las aplicaciones móviles, la idea es que el usuario pueda acceder a todos estos servicios de modo offline (la app se actualiza cuando haya conexión a internet). Además, se pretende realizar una versión para Android utilizando para ello MonoDroid.

Por tanto queda mucho trabajo por realizar ya que solamente se han marcado las líneas generales que se desprenderán de este proyecto.

Bibliografía

Josué Yeray Julián;Ibón Landa. (2011). “Windows Phone 7.5 Desarrollo de aplicaciones en Silverlight.” (1º) Krasis Press.

Charles Petzold. (2010) “Programming Windows Phone 7” (1º) Microsoft Press.

Boryana Miloshevska. (2011) “Silverlight for Windows Phone Toolkit In Depth” (1º) WindowsPhoneGeek.com

Scott Olsen (2012) “Cross-Plarform Mobile Development in C#” (1º) Wrox

Wallace B. (2011) Professional iPhone Programming (1º) Wrox